

QGIS Desktop 3.44 User Guide

QGIS Project

Oct 24, 2025

CONTENTS

1 Getting Started **3**
- 1.1 Installing QGIS . . . 3
  - 1.1.1 Installing from binaries . . . 3
  - 1.1.2 Installing from source . . . 3
  - 1.1.3 Installing on external media . . . 3
  - 1.1.4 Downloading sample data . . . 4
- 1.2 Starting and stopping QGIS . . . 4
- 1.3 Sample Session: Loading raster and vector layers . . . 5

2 Working with Project Files **13**
- 2.1 Introducing QGIS projects . . . 13
- 2.2 Handling broken file paths . . . 15
- 2.3 Generating output . . . 16

3 QGIS GUI **17**
- 3.1 Menu Bar . . . 18
  - 3.1.1 Project . . . 18
  - 3.1.2 Edit . . . 19
  - 3.1.3 View . . . 21
  - 3.1.4 Layer . . . 24
  - 3.1.5 Settings . . . 26
  - 3.1.6 Plugins . . . 27
  - 3.1.7 Vector . . . 27
  - 3.1.8 Raster . . . 29
  - 3.1.9 Database . . . 30
  - 3.1.10 Web . . . 30
  - 3.1.11 Mesh . . . 31
  - 3.1.12 Processing . . . 31
  - 3.1.13 Help . . . 31
  - 3.1.14 QGIS . . . 32
- 3.2 Panels and Toolbars . . . 32
  - 3.2.1 Toolbars . . . 32
  - 3.2.2 Panels . . . 33
- 3.3 Status Bar . . . 34
  - 3.3.1 Locator bar . . . 34
  - 3.3.2 Reporting actions . . . 35
  - 3.3.3 Control the map canvas . . . 35
  - 3.3.4 Messaging . . . 36

4 The Browser panel **37**
- 4.1 Resources that can be opened / run from the Browser . . . 39
- 4.2 Browser panel top-level entries . . . 40
  - 4.2.1 Favorites . . . 40
  - 4.2.2 Spatial Bookmarks . . . 40

4.2.3 Project Home . . . 40
4.2.4 Drives and file system . . . 40
4.2.5 Data type . . . 41
4.2.6 Database entries . . . 41
4.2.7 Tiles and Web Services . . . 47
4.2.8 Project and supported file types . . . 48

5 QGIS Configuration 49
5.1 Options . . . 49
5.1.1 General Settings . . . 50
5.1.2 System Settings . . . 52
5.1.3 User Profiles Settings . . . 54
5.1.4 CRS and Transforms Settings . . . 54
5.1.5 Data Sources settings . . . 58
5.1.6 Rendering Settings . . . 62
5.1.7 Canvas and Legend Settings . . . 66
5.1.8 Map tools Settings . . . 68
5.1.9 Elevation settings . . . 72
5.1.10 3D settings . . . 72
5.1.11 Colors settings . . . 73
5.1.12 Fonts Settings . . . 74
5.1.13 Layouts settings . . . 75
5.1.14 Variables settings . . . 76
5.1.15 Authentication settings . . . 76
5.1.16 Network settings . . . 77
5.1.17 GPS settings . . . 79
5.1.18 Locator settings . . . 81
5.1.19 Acceleration settings . . . 83
5.1.20 IDE settings . . . 83
5.1.21 Processing settings . . . 87
5.1.22 Advanced settings . . . 88
5.2 Working with User Profiles . . . 89
5.2.1 The concept . . . 89
5.2.2 Setting user profile . . . 90
5.3 Project Properties . . . 91
5.3.1 General Properties . . . 91
5.3.2 Metadata Properties . . . 94
5.3.3 View Settings . . . 95
5.3.4 CRS Properties . . . 95
5.3.5 Transformations Properties . . . 96
5.3.6 Styles Properties . . . 96
5.3.7 Colors properties . . . 97
5.3.8 Data Sources Properties . . . 98
5.3.9 Relations Properties . . . 100
5.3.10 Variables Properties . . . 101
5.3.11 Macros Properties . . . 101
5.3.12 QGIS Server Properties . . . 102
5.3.13 Temporal Properties . . . 103
5.3.14 Elevation Properties . . . 103
5.3.15 Sensors Properties . . . 106
5.4 Customization . . . 107
5.5 Keyboard shortcuts . . . 109
5.6 Running QGIS with advanced settings . . . 110
5.6.1 Command line and environment variables . . . 110
5.6.2 Deploying QGIS within an organization . . . 116

6 Working with Projections 119
6.1 Overview of Projection Support . . . 119

6.2 Layer Coordinate Reference Systems . . . 119
6.3 Project Coordinate Reference Systems . . . 121
6.4 Coordinate Reference System Selector . . . 123
6.5 Custom Coordinate Reference System . . . 124
6.5.1 Integrate an NTv2-transformation in QGIS . . . 125
6.6 Datum Transformations . . . 126

7 Visualizing Maps **129**
7.1 2D Map View . . . 129
7.1.1 Exploring the map view . . . 130
7.1.2 Controlling map rendering . . . 131
7.1.3 Time-based control on the map canvas . . . 132
7.1.4 The Elevation Controller panel . . . 134
7.1.5 Bookmarking extents on the map . . . 134
7.1.6 Decorating the map . . . 136
7.1.7 Annotation Tools . . . 144
7.1.8 Measuring . . . 151
7.1.9 Setting additional map views . . . 153
7.1.10 Exporting the map view . . . 154
7.2 3D Map View . . . 157
7.2.1 Scene Configuration . . . 159
7.2.2 Navigation options . . . 166
7.2.3 Creating an animation . . . 167
7.2.4 3D vector layers . . . 167
7.3 Elevation Profile View . . . 167
7.3.1 The interface . . . 168
7.3.2 Creating an elevation profile . . . 170
7.3.3 Interacting with the profile Views . . . 171
7.3.4 Exporting from the elevation profile . . . 172

8 General Tools **173**
8.1 Context help . . . 173
8.2 Panels . . . 173
8.2.1 Layers Panel . . . 173
8.2.2 Layer Styling Panel . . . 180
8.2.3 Layer Order Panel . . . 183
8.2.4 Overview Panel . . . 183
8.2.5 Log Messages Panel . . . 184
8.2.6 Undo/Redo Panel . . . 184
8.2.7 Statistical Summary Panel . . . 184
8.2.8 Debugging/Development Tools Panel . . . 185
8.3 Embedding layers from external projects . . . 189
8.4 Interacting with features . . . 190
8.4.1 Selecting features . . . 190
8.4.2 Identifying Features . . . 193
8.5 Save and Share Layer Properties . . . 198
8.5.1 Managing Custom Styles . . . 198
8.5.2 Storing Styles in a File or a Database . . . 199
8.5.3 Layer definition file . . . 201
8.6 Documenting your data . . . 201
8.6.1 Metadata . . . 202
8.6.2 Layer notes . . . 203
8.7 Storing values in Variables . . . 204
8.8 Authentication . . . 205
8.9 Common widgets . . . 206
8.9.1 Color Selector . . . 206
8.9.2 Symbol Widget . . . 210
8.9.3 Remote or embedded file selector . . . 210

8.9.4 Visibility Scale Selector . . . 211
8.9.5 Spatial Extent Selector . . . 211
8.9.6 Font Selector . . . 212
8.9.7 Unit Selector . . . 213
8.9.8 Number Formatting . . . 214
8.9.9 Blending Modes . . . 215
8.9.10 Data defined override setup . . . 217

9 Level up with Expressions **221**
9.1 Expressions . . . 221
9.1.1 The Expression string builder . . . 221
9.1.2 Function Editor . . . 226
9.2 List of functions . . . 228
9.2.1 Aggregates Functions . . . 228
9.2.2 Array Functions . . . 239
9.2.3 Color Functions . . . 251
9.2.4 Conditional Functions . . . 260
9.2.5 Conversions Functions . . . 263
9.2.6 CRS Functions . . . 270
9.2.7 Custom Functions . . . 271
9.2.8 Date and Time Functions . . . 271
9.2.9 Fields and Values . . . 283
9.2.10 Files and Paths Functions . . . 283
9.2.11 Form Functions . . . 286
9.2.12 Fuzzy Matching Functions . . . 286
9.2.13 General Functions . . . 288
9.2.14 Geometry Functions . . . 290
9.2.15 Layout Functions . . . 364
9.2.16 Map Layers . . . 365
9.2.17 Maps Functions . . . 367
9.2.18 Mathematical Functions . . . 371
9.2.19 Meshes Functions . . . 379
9.2.20 Operators . . . 381
9.2.21 Processing Functions . . . 390
9.2.22 Rasters Functions . . . 390
9.2.23 Record and Attributes Functions . . . 392
9.2.24 Relations . . . 401
9.2.25 Sensors Functions . . . 401
9.2.26 String Functions . . . 401
9.2.27 User Expressions . . . 412
9.2.28 Variables . . . 412
9.2.29 Recent Functions . . . 416

10 The Style Library **417**
10.1 The Style Manager . . . 417
10.1.1 The Style Manager dialog . . . 417
10.1.2 Setting a Color Ramp . . . 422
10.1.3 Creating a Legend Patch Shape . . . 424
10.2 The Symbol Selector . . . 426
10.2.1 The symbol layer tree . . . 427
10.2.2 Configuring a symbol . . . 428
10.3 Setting a label . . . 446
10.3.1 Formatting the label text . . . 448
10.3.2 Configuring interaction with labels . . . 457
10.4 Creating 3D Symbols . . . 467
10.4.1 Point Layers . . . 468
10.4.2 Line layers . . . 469
10.4.3 Polygon Layers . . . 470

10.4.4 Shading the texture . . . 472
10.4.5 Application example . . . 472

11 Managing Data Source **473**
11.1 Opening Data . . . 473
11.1.1 The Browser Panel . . . 475
11.1.2 The DB Manager . . . 478
11.1.3 Provider-based loading tools . . . 479
11.1.4 The Layer Metadata Search Panel . . . 501
11.1.5 QGIS Custom formats . . . 502
11.1.6 QLR - QGIS Layer Definition File . . . 502
11.1.7 Connecting to web services . . . 503
11.2 Creating Layers . . . 510
11.2.1 Creating new vector layers . . . 511
11.2.2 Creating new layers from an existing layer . . . 519
11.2.3 Creating new DXF files . . . 524
11.2.4 Creating new layers from the clipboard . . . 525
11.2.5 Creating SQL Query Layers . . . 526
11.2.6 Creating virtual layers . . . 528
11.3 Georeferencer . . . 531
11.3.1 Georeferencing raster layer . . . 532
11.3.2 Georeferencing vector layer . . . 537
11.3.3 Show and adapt layer properties . . . 538
11.3.4 Configure the georeferencer . . . 538
11.4 Exploring Data Formats and Fields . . . 539
11.4.1 Raster data . . . 539
11.4.2 Vector Data . . . 539
11.4.3 Layers crossing 180° longitude . . . 547

12 Working with Vector Data **549**
12.1 The Vector Properties Dialog . . . 549
12.1.1 Information Properties . . . 550
12.1.2 Source Properties . . . 550
12.1.3 Symbology Properties . . . 553
12.1.4 Labels Properties . . . 576
12.1.5 Diagrams Properties . . . 588
12.1.6 Masks Properties . . . 593
12.1.7 3D View Properties . . . 594
12.1.8 Fields Properties . . . 596
12.1.9 Attributes Form Properties . . . 597
12.1.10 Joins Properties . . . 609
12.1.11 Auxiliary Storage Properties . . . 609
12.1.12 Actions Properties . . . 617
12.1.13 Display Properties . . . 622
12.1.14 Rendering Properties . . . 625
12.1.15 Temporal Properties . . . 627
12.1.16 Variables Properties . . . 628
12.1.17 Elevation Properties . . . 628
12.1.18 Metadata Properties . . . 630
12.1.19 Dependencies Properties . . . 630
12.1.20 Legend Properties . . . 630
12.1.21 QGIS Server Properties . . . 631
12.1.22 Digitizing Properties . . . 633
12.2 Working with the Attribute Table . . . 635
12.2.1 Foreword: Spatial and non-spatial tables . . . 635
12.2.2 Introducing the attribute table interface . . . 636
12.2.3 Interacting with features in an attribute table . . . 641
12.2.4 Editing attribute values . . . 645

12.2.5 Exploring features attributes through the Identify Tool . . . 649
12.2.6 Storing and fetching an external resource . . . 650
12.3 Editing . . . 652
12.3.1 Setting the snapping tolerance and search radius . . . 653
12.3.2 Snapping and Digitizing Options . . . 653
12.3.3 Topological editing . . . 656
12.3.4 Digitizing an existing layer . . . 658
12.3.5 Advanced digitizing . . . 668
12.3.6 Shape digitizing . . . 677
12.3.7 The Advanced Digitizing panel . . . 679
12.3.8 The Processing in-place layer modifier . . . 688
12.4 Connecting and Editing Data Across Layers . . . 690
12.4.1 Joining features between two layers . . . 690
12.4.2 Setting relations between multiple layers . . . 692

13 Working with Raster Data **707**
13.1 Raster Properties Dialog . . . 707
13.1.1 Information Properties . . . 708
13.1.2 Source Properties . . . 708
13.1.3 Symbology Properties . . . 709
13.1.4 Transparency Properties . . . 719
13.1.5 Labels Properties . . . 720
13.1.6 Histogram Properties . . . 722
13.1.7 Rendering Properties . . . 723
13.1.8 Temporal Properties . . . 724
13.1.9 Pyramids Properties . . . 726
13.1.10 Elevation Properties . . . 727
13.1.11 Metadata Properties . . . 729
13.1.12 Legend Properties . . . 730
13.1.13 Display Properties . . . 730
13.1.14 QGIS Server Properties . . . 731
13.1.15 Identify raster cells . . . 732
13.2 Raster Analysis . . . 733
13.2.1 Raster Calculator . . . 733

14 Working with Mesh Data **737**
14.1 What's a mesh? . . . 737
14.2 Supported formats . . . 739
14.3 Mesh Dataset Properties . . . 739
14.3.1 Information Properties . . . 740
14.3.2 Source Properties . . . 741
14.3.3 Symbology Properties . . . 742
14.3.4 Labels Properties . . . 748
14.3.5 3D View Properties . . . 750
14.3.6 Rendering Properties . . . 752
14.3.7 Temporal Properties . . . 753
14.3.8 Elevation Properties . . . 753
14.3.9 Metadata Properties . . . 755
14.4 Editing a mesh layer . . . 755
14.4.1 Overview of the mesh digitizing tools . . . 755
14.4.2 Exploring the Z value assignment logic . . . 756
14.4.3 Selecting mesh elements . . . 757
14.4.4 Modifying mesh elements . . . 759
14.4.5 Reindexing meshes . . . 763
14.5 Mesh Calculator . . . 763

15 Working with Vector Tiles **767**
15.1 What are Vector Tiles? . . . 767
15.2 Supported Formats . . . 768

15.3 Vector Tiles Dataset Properties . . . 769
15.3.1 Information Properties . . . 769
15.3.2 Source Properties . . . 769
15.3.3 Symbology and Labels Properties . . . 770
15.3.4 Rendering Properties . . . 771
15.3.5 Metadata Properties . . . 772

16 Working with Point Clouds **773**
16.1 Introduction to Point Clouds . . . 773
16.2 Point Clouds Properties . . . 773
16.2.1 Information Properties . . . 774
16.2.2 Source Properties . . . 775
16.2.3 Symbology Properties . . . 778
16.2.4 3D View Properties . . . 786
16.2.5 Rendering Properties . . . 788
16.2.6 Elevation Properties . . . 788
16.2.7 Metadata Properties . . . 790
16.2.8 Statistics Properties . . . 790
16.3 Virtual point cloud . . . 791

17 Working with 3D Tiles **793**
17.1 What are 3D Tiles? . . . 793
17.2 3D Tiles Properties . . . 794
17.2.1 Information Properties . . . 794
17.2.2 Source Properties . . . 794
17.2.3 Symbology Properties . . . 794
17.2.4 3D View Properties . . . 795
17.2.5 Rendering Properties . . . 796
17.2.6 Elevation Properties . . . 796
17.2.7 Metadata Properties . . . 796

18 Laying out the maps **797**
18.1 Overview of the Print Layout . . . 797
18.1.1 Sample Session for beginners . . . 797
18.1.2 The Layout Manager . . . 798
18.1.3 Menus, tools and panels of the print layout . . . 799
18.2 Layout Items . . . 813
18.2.1 Layout Items Common Options . . . 813
18.2.2 The Map Item . . . 819
18.2.3 The 3D Map Item . . . 829
18.2.4 The Label Item . . . 830
18.2.5 The Legend Item . . . 835
18.2.6 The Scale Bar Item . . . 843
18.2.7 The Table Items . . . 846
18.2.8 The Marker, Picture and North Arrow Items . . . 855
18.2.9 The Elevation Profile Item . . . 859
18.2.10 The HTML Frame Item . . . 860
18.2.11 The Shape Items . . . 863
18.3 Creating an Output . . . 866
18.3.1 Export settings . . . 867
18.3.2 Export as Image . . . 867
18.3.3 Export as SVG . . . 868
18.3.4 Export as PDF . . . 870
18.3.5 Generate an Atlas . . . 871
18.4 Creating a Report . . . 877
18.4.1 What is it? . . . 877
18.4.2 Get started . . . 877
18.4.3 Layout Report Workspace . . . 878
18.4.4 Export settings . . . 893

19 Working with OGC / ISO protocols **895**
19.1 WMS/WMTS Client . . . 895
19.1.1 Overview of WMS Support . . . 895
19.1.2 Overview of WMTS Support . . . 896
19.1.3 Selecting WMS/WMTS Servers . . . 897
19.1.4 Loading WMS/WMTS Layers . . . 899
19.1.5 Tilesets . . . 901
19.1.6 Using the Identify Tool . . . 901
19.1.7 Viewing Properties . . . 902
19.1.8 Show WMS legend graphic in table of contents and layout . . . 905
19.2 WCS Client . . . 906
19.3 WFS and WFS-T Client . . . 907
19.4 SensorThings . . . 910
19.4.1 Setting connection . . . 910
19.4.2 Loading SensorThings data . . . 912
19.4.3 Working with a vector layer from SensorThings . . . 913
19.5 STAC (SpatioTemporal Asset Catalogs) . . . 913
19.5.1 Setting connection . . . 914
19.5.2 Browsing STAC Catalogs . . . 915
19.5.3 Filtering and Searching STAC Items . . . 916

20 Working with GPS Data **919**
20.1 Introducing GNSS/GPS Data . . . 919
20.1.1 What is GPS? . . . 919
20.1.2 Transferring or loading GPS data . . . 919
20.2 Live GPS tracking . . . 921
20.2.1 GPS toolbar . . . 921
20.2.2 GPS Information Panel . . . 923
20.2.3 Connect to a Bluetooth GPS for live tracking . . . 925
20.2.4 Examples of GPS devices connection in QGIS . . . 926

21 Authentication System **929**
21.1 Authentication System Overview . . . 929
21.1.1 Authentication database . . . 929
21.1.2 Custom authentication databases . . . 930
21.1.3 Master password . . . 931
21.1.4 Authentication Configurations . . . 932
21.1.5 Authentication Methods . . . 934
21.1.6 Master Password and Auth Config Utilities . . . 938
21.1.7 Using authentication configurations . . . 939
21.1.8 Python bindings . . . 940
21.2 User Authentication Workflows . . . 940
21.2.1 HTTP(S) authentication . . . 940
21.2.2 Database authentication . . . 941
21.2.3 PKI authentication . . . 942
21.2.4 Handling bad layers . . . 949
21.2.5 Changing authentication config ID . . . 950
21.2.6 QGIS Server support . . . 951
21.2.7 SSL server exceptions . . . 951

22 GRASS GIS Integration **955**
22.1 Demo dataset . . . 955
22.2 Loading GRASS raster and vector layers . . . 955
22.3 Importing data into a GRASS LOCATION via drag and drop . . . 956
22.4 Managing GRASS data in QGIS Browser . . . 956
22.5 GRASS Options . . . 956
22.6 Starting the GRASS plugin . . . 956
22.7 Opening GRASS mapset . . . 957
22.8 GRASS LOCATION and MAPSET . . . 957

22.9 Importing data into a GRASS LOCATION . . . 957
22.9.1 Creating a new GRASS LOCATION . . . 958
22.9.2 Adding a new MAPSET . . . 959
22.10 The GRASS vector data model . . . 960
22.11 Creating a new GRASS vector layer . . . 960
22.12 Digitizing and editing a GRASS vector layer . . . 961
22.13 The GRASS region tool . . . 963
22.14 The GRASS Toolbox . . . 963
22.14.1 Working with GRASS modules . . . 964
22.14.2 GRASS module examples . . . 967
22.14.3 Customizing the GRASS Toolbox . . . 970

23 QGIS processing framework **973**
23.1 Introduction . . . 973
23.2 Configuring the Processing Framework . . . 976
23.2.1 General . . . 976
23.2.2 Menus . . . 978
23.2.3 Models and Scripts . . . 978
23.2.4 Providers . . . 978
23.3 The Toolbox . . . 979
23.3.1 The algorithm dialog . . . 981
23.3.2 Data objects generated by algorithms . . . 989
23.4 The history manager . . . 990
23.4.1 The processing history . . . 990
23.4.2 The processing log . . . 991
23.5 The model designer . . . 991
23.5.1 The model designer interface . . . 992
23.5.2 Creating a model . . . 995
23.5.3 Saving and loading models . . . 1004
23.5.4 Editing a model . . . 1004
23.6 The batch processing interface . . . 1006
23.6.1 Introduction . . . 1006
23.6.2 The parameters table . . . 1007
23.6.3 Filling the parameters table . . . 1008
23.6.4 Executing the batch process . . . 1009
23.7 Using processing algorithms from the console . . . 1009
23.7.1 Calling algorithms from the Python console . . . 1010
23.7.2 Creating scripts and running them from the toolbox . . . 1014
23.7.3 Pre- and post-execution script hooks . . . 1017
23.8 Using processing from the command line . . . 1017
23.9 Writing new Processing algorithms as Python scripts . . . 1019
23.9.1 Extending QgsProcessingAlgorithm . . . 1019
23.9.2 The @alg decorator . . . 1023
23.9.3 Handing algorithm output . . . 1025
23.9.4 Communicating with the user . . . 1025
23.9.5 Documenting your scripts . . . 1026
23.9.6 Flags . . . 1026
23.9.7 Best practices for writing script algorithms . . . 1026
23.10 Input and output types for Processing Algorithms . . . 1026
23.10.1 Input types . . . 1026
23.10.2 Output types . . . 1029
23.11 Configuring external applications . . . 1030
23.11.1 A note for Windows users . . . 1030
23.11.2 A note on file formats . . . 1030
23.11.3 A note on vector layer selections . . . 1030
23.11.4 Using third-party Providers . . . 1031

24 Processing providers and algorithms **1043**

24.1 QGIS algorithm provider . . . 1043
24.1.1 3D Tiles . . . 1043
24.1.2 Cartography . . . 1045
24.1.3 Check Geometry . . . 1066
24.1.4 Database . . . 1097
24.1.5 File tools . . . 1105
24.1.6 Fix Geometry . . . 1108
24.1.7 GPS . . . 1126
24.1.8 Interpolation . . . 1129
24.1.9 Layer tools . . . 1141
24.1.10 Mesh . . . 1144
24.1.11 Metadata tools . . . 1157
24.1.12 Modeler tools . . . 1162
24.1.13 Network analysis . . . 1172
24.1.14 Plots . . . 1185
24.1.15 Point Cloud Conversion . . . 1194
24.1.16 Point Cloud Data Management . . . 1200
24.1.17 Point Cloud Extraction . . . 1213
24.1.18 Raster analysis . . . 1219
24.1.19 Raster Creation . . . 1281
24.1.20 Raster terrain analysis . . . 1300
24.1.21 Raster tools . . . 1313
24.1.22 Vector analysis . . . 1324
24.1.23 Vector coverage . . . 1351
24.1.24 Vector creation . . . 1354
24.1.25 Vector general . . . 1383
24.1.26 Vector geometry . . . 1422
24.1.27 Vector overlay . . . 1541
24.1.28 Vector selection . . . 1561
24.1.29 Vector table . . . 1579
24.1.30 Vector Tiles . . . 1596
24.2 GDAL algorithm provider . . . 1600
24.2.1 Raster analysis . . . 1600
24.2.2 Raster conversion . . . 1630
24.2.3 Raster extraction . . . 1638
24.2.4 Raster miscellaneous . . . 1647
24.2.5 Raster projections . . . 1666
24.2.6 Vector conversion . . . 1671
24.2.7 Vector geoprocessing . . . 1678
24.2.8 Vector miscellaneous . . . 1687

25 Plugins **1699**
25.1 QGIS Plugins . . . 1699
25.1.1 Core and External plugins . . . 1699
25.1.2 The Plugins Dialog . . . 1699
25.2 Using QGIS Core Plugins . . . 1703
25.2.1 DB Manager Plugin . . . 1703
25.2.2 Geometry Checker Plugin . . . 1705
25.2.3 MetaSearch Catalog Client . . . 1709
25.2.4 Offline Editing Plugin . . . 1716
25.2.5 Topology Checker Plugin . . . 1718
25.3 QGIS Python console . . . 1720
25.3.1 The Interactive Console . . . 1721
25.3.2 The Code Editor . . . 1723

26 Appendices **1727**
26.1 Appendix C: QGIS File Formats . . . 1727
26.1.1 QGS/QGZ - The QGIS Project File Format . . . 1727

26.1.2 QLR - The QGIS Layer Definition file . . . 1729
26.1.3 QML - The QGIS Style File Format . . . 1730
26.2 Appendix D: QGIS R script syntax . . . 1731
26.2.1 Inputs . . . 1732
26.2.2 Outputs . . . 1732
26.2.3 Syntax Summary for QGIS R scripts . . . 1732
26.2.4 Examples . . . 1734
26.3 Appendix E: QGIS Application Network Connections . . . 1736

27 Literature and Web References **1739**

CHAPTER

ONE

GETTING STARTED

This chapter provides a quick overview of installing QGIS, downloading QGIS sample data, and running a first simple session visualizing raster and vector data.

1.1 Installing QGIS

QGIS project provides different ways to install QGIS depending on your platform.

1.1.1 Installing from binaries

Standard installers are available for  MS Windows and  macOS. Binary packages (rpm and deb) or software repositories are provided for many flavors of GNU/Linux  .

For more information and instructions for your operating system check https://download.qgis.org.

1.1.2 Installing from source

If you need to build QGIS from source, please refer to the installation instructions. They are distributed with the QGIS source code in a file called `INSTALL`. You can also find them online at https://github.com/qgis/QGIS/blob/release-3_44/INSTALL.md.

If you want to build a particular release and not the version in development, you should replace `master` with the release branch (commonly in the `release-X_Y` form) in the above-mentioned link (installation instructions may differ).

1.1.3 Installing on external media

It is possible to install QGIS (with all plugins and settings) on a flash drive. This is achieved by defining a *–profiles-path* option that overrides the default *user profile* path and forces **QSettings** to use this directory, too. See section *System Settings* for additional information.

1.1.4 Downloading sample data

This user guide contains examples based on the QGIS sample dataset (also called the `Alaska dataset`). Download the sample data from https://github.com/qgis/QGIS-Sample-Data/archive/master.zip and unzip the archive on any convenient location on your system.

The Alaska dataset includes all GIS data that are used for the examples and screenshots in this user guide; it also includes a small GRASS database. The projection for the QGIS sample datasets is Alaska Albers Equal Area with units feet. The EPSG code is 2964.

```
PROJCS["Albers Equal Area",
GEOGCS["NAD27",
DATUM["North_American_Datum_1927",
SPHEROID["Clarke 1866",6378206.4,294.978698213898,
AUTHORITY["EPSG","7008"]],
TOWGS84[-3,142,183,0,0,0,0],
AUTHORITY["EPSG","6267"]],
PRIMEM["Greenwich",0,
AUTHORITY["EPSG","8901"]],
UNIT["degree",0.0174532925199433,
AUTHORITY["EPSG","9108"]],
AUTHORITY["EPSG","4267"]],
PROJECTION["Albers_Conic_Equal_Area"],
PARAMETER["standard_parallel_1",55],
PARAMETER["standard_parallel_2",65],
PARAMETER["latitude_of_center",50],
PARAMETER["longitude_of_center",-154],
PARAMETER["false_easting",0],
PARAMETER["false_northing",0],
UNIT["us_survey_feet",0.3048006096012192]]
```

If you intend to use QGIS as a graphical front end for GRASS, you can find a selection of sample locations (e.g., Spearfish or South Dakota) at the official GRASS GIS website, https://grass.osgeo.org/download/data/.

1.2 Starting and stopping QGIS

QGIS can be started like any other application on your computer. This means that you can launch QGIS by:

- using the Applications menu, the Start menu, or the Dock
- double clicking the icon in your Applications folder or desktop shortcut
- double clicking an existing QGIS project file (with `.qgz` or `.qgs` extension). Note that this will also open the project.
- typing `qgis` in a command prompt (assuming that QGIS is added to your PATH or you are in its installation folder)

To stop QGIS, use:

- the menu option *Project* ► *Exit QGIS* or use the shortcut `Ctrl+Q`
- *QGIS* ► *Quit QGIS*, or use the shortcut `Cmd+Q`
- or use the red cross at the top-right corner of the main interface of the application.

1.3 Sample Session: Loading raster and vector layers

Now that you have *QGIS installed* and a *sample dataset* available, we will demonstrate a first sample session. In this example, we will visualize a raster and a vector layer. We will use:

- the `landcover` raster layer (`qgis_sample_data/raster/landcover.img`)
- and the `lakes` vector layer (`qgis_sample_data/gml/lakes.gml`)

Where `qgis_sample_data` represents the path to the unzipped dataset.

1. Start QGIS as seen in *Starting and stopping QGIS*.
2. The data we will be working with are in `Albers Equal Area`, so let's set the project's CRS accordingly:
    1. Click the Select projection button in the bottom right of QGIS interface. The project properties dialog opens with the *CRS* tab active.
    2. Type `2964` in the *Filter* text area.
    3. Select the row with `NAD27 / Alaska Albers` CRS name.

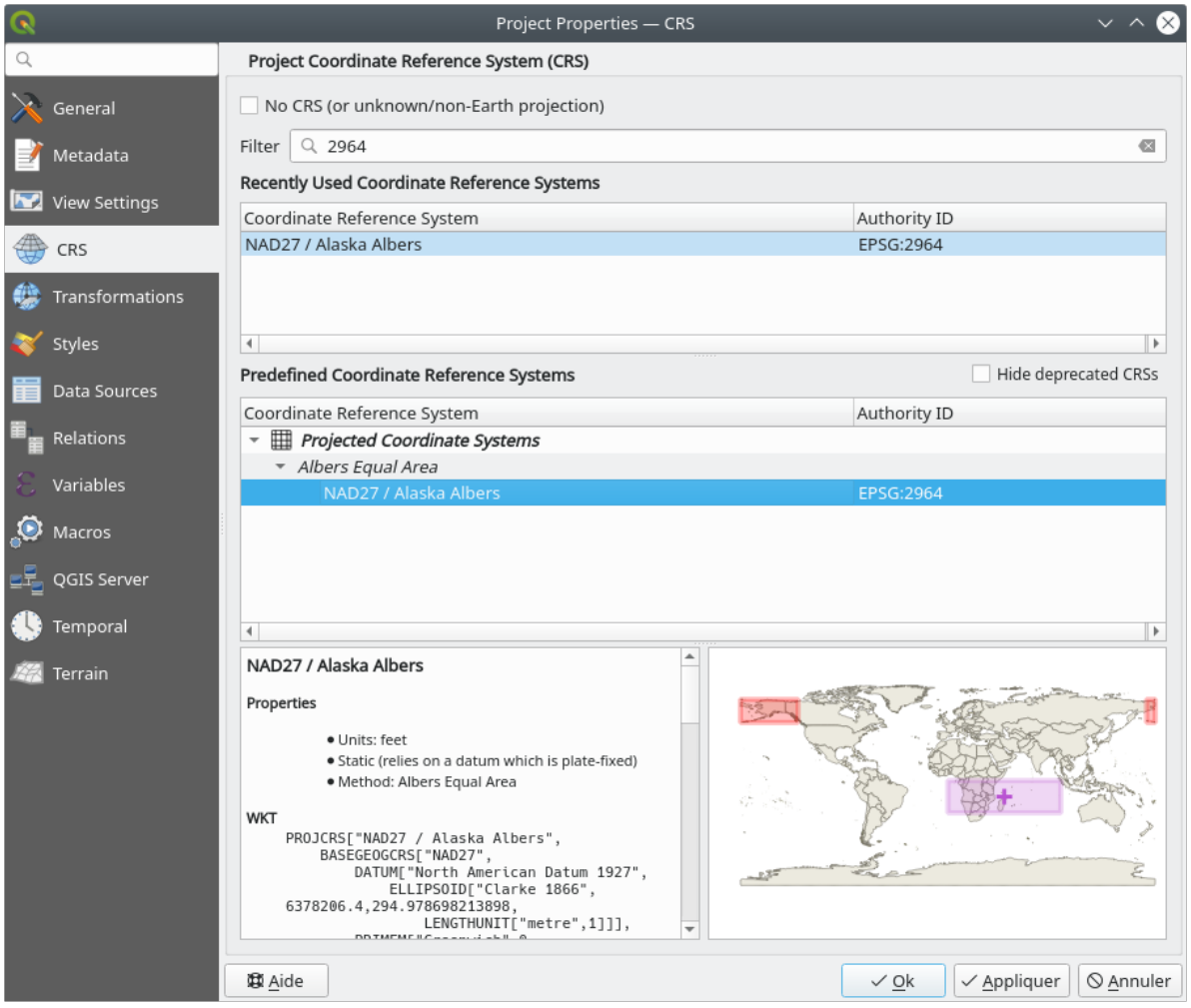


Fig. 1.1: Select the Coordinate Reference System of data

4. Press *OK*

> **Note**
>
> You can ignore/close for now the “ballpark transform” message that could display.

3. Load the files in QGIS:

   1. Click on the  Open Data Source Manager icon. The Data Source Manager should open in Browser mode.
   2. Browse to the folder `qgis_sample_data/raster/`
   3. Select the ERDAS IMG file  *landcover.img* and double-click it. The landcover layer is added in the background while the Data Source Manager window remains open.

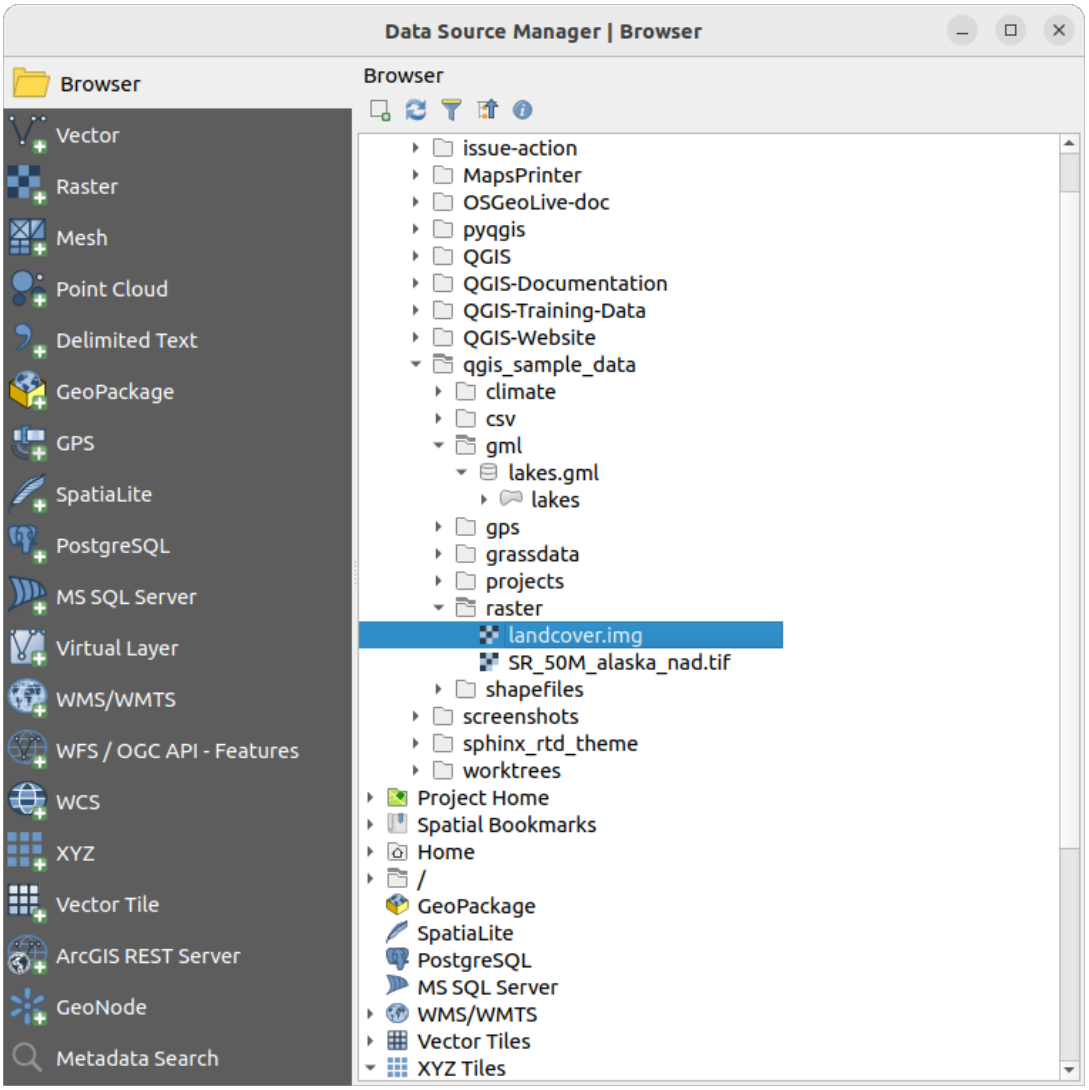


Fig. 1.2: Adding data to a new project in QGIS

4. To load the lakes data, browse to the folder `qgis_sample_data/gml/`, and drag and drop the  *lakes.gml* file over QGIS main dialog. (Or just double-click as mentioned above.)
5. The *Select Items to Add* dialog opens, scanning the file. This is due to `.gml` file format being able to store more than one layer at a time.

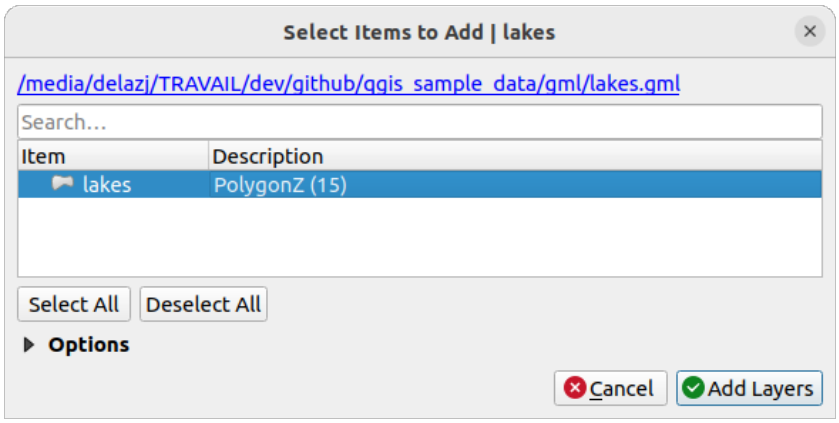


Fig. 1.3: Select layers within a file

6. In our case there is a single *lakes* layer. Select it and press *Add Layers*.
7. The layer is added to the *Layers* panel

4. Close the Data Source Manager window

In the *Layers* panel, you can notice that the *lakes* layer displays Layer has no coordinate reference system set next to it. Let's adjust that.

1. Click the icon. The *Coordinate Reference System Selector* dialog opens.
2. As done earlier, find and select the *NAD27 / Alaska Albers* CRS entry.
3. Click *OK*

You now have the two layers available in your project in some random colours. Let's do some customization on the lakes layer.

1. Select the Zoom In tool on the *Navigation* toolbar
2. Zoom to an area with some lakes
3. Double-click the `lakes` layer in the map legend to open the *Properties* dialog
4. To change the lakes color:
    1. Click on the *Symbology* tab
    2. Select blue as fill color.

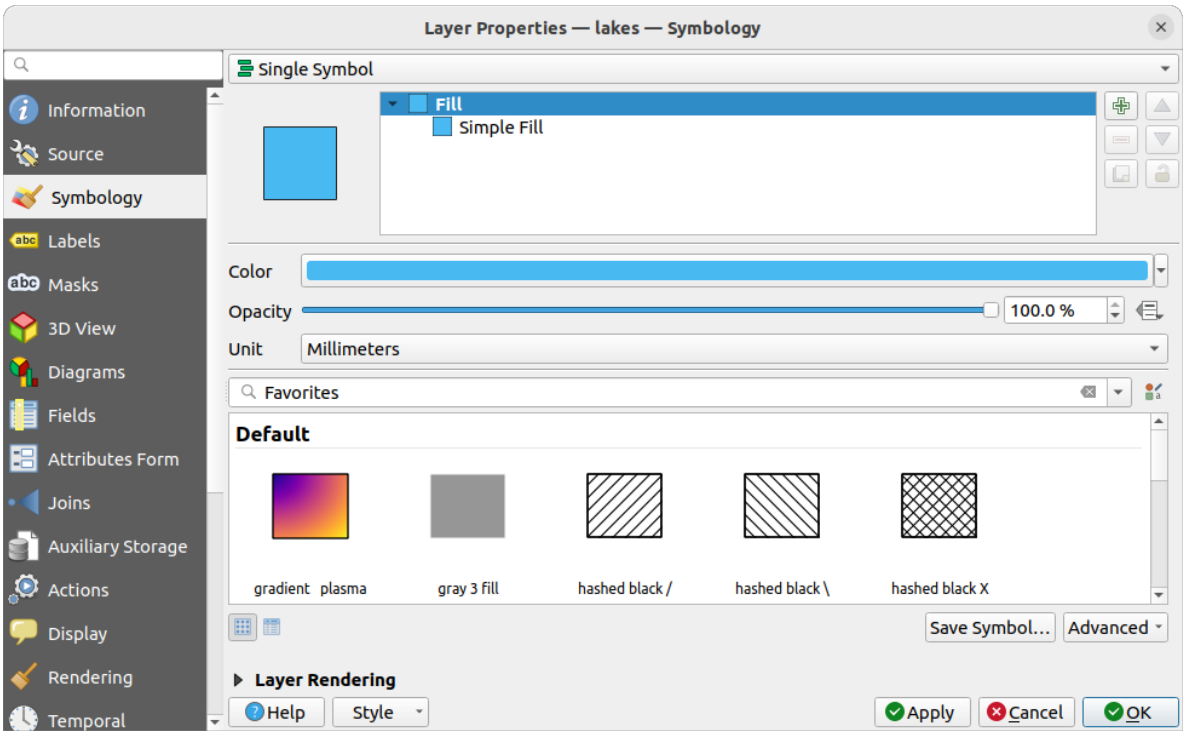


Fig. 1.4: Selecting Lakes color

3. Press *OK*. Lakes are now displayed in blue in the map canvas.

5. To display the name of the lakes:
   1. Reopen the `lakes` layer *Properties* dialog
   2. Click on the *Labels* tab
   3. Select *Single labels* in the drop-down menu to enable labeling.
   4. From the *Label with* list, choose the `NAMES` field.

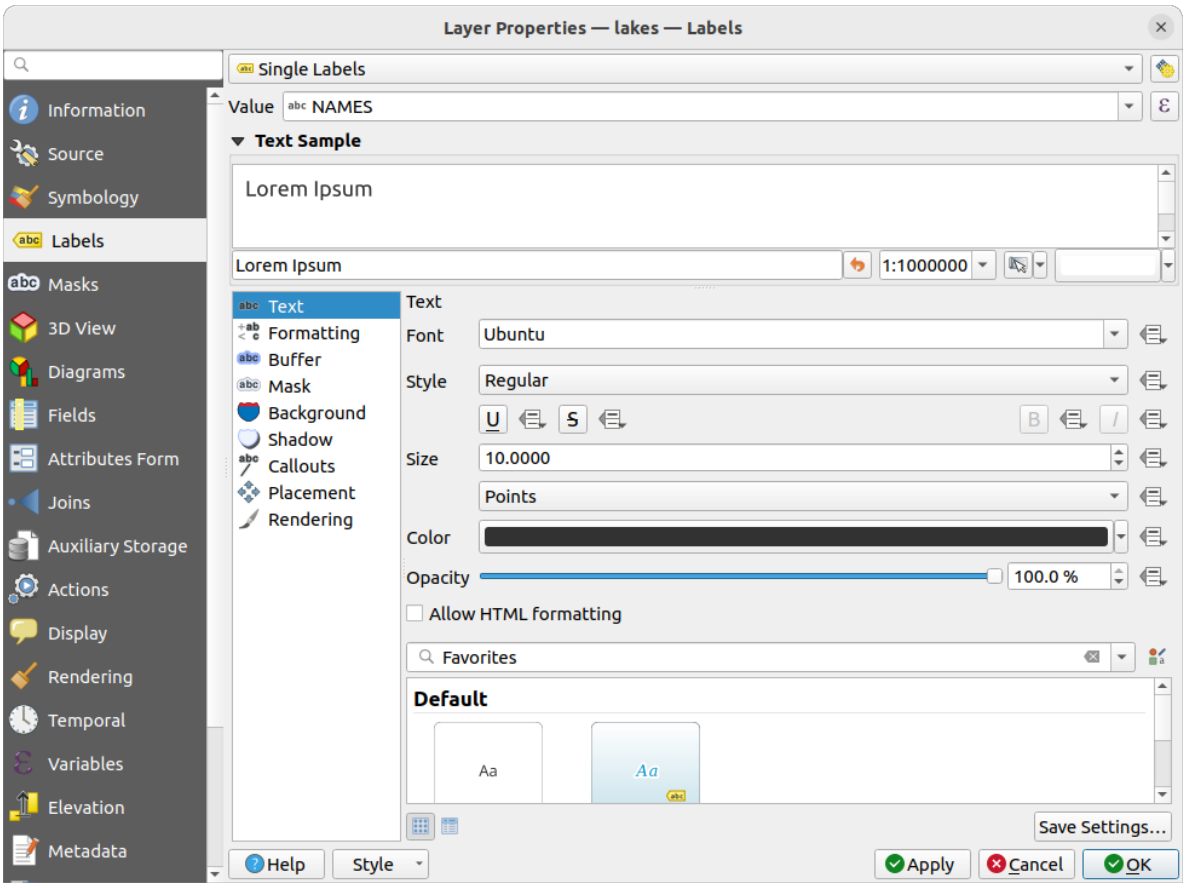


Fig. 1.5: Showing Lakes names

5. Press *Apply*. Names will now load over the boundaries.

6. You can improve readability of the labels by adding a white buffer around them:

   1. Click the *Buffer* tab in the list on the left
   2. Check *Draw text buffer*
   3. Choose 3 as buffer size
   4. Click *Apply*
   5. Check if the result looks good, and update the value if needed.
   6. Finally click *OK* to close the *Layer Properties* dialog and apply the changes.

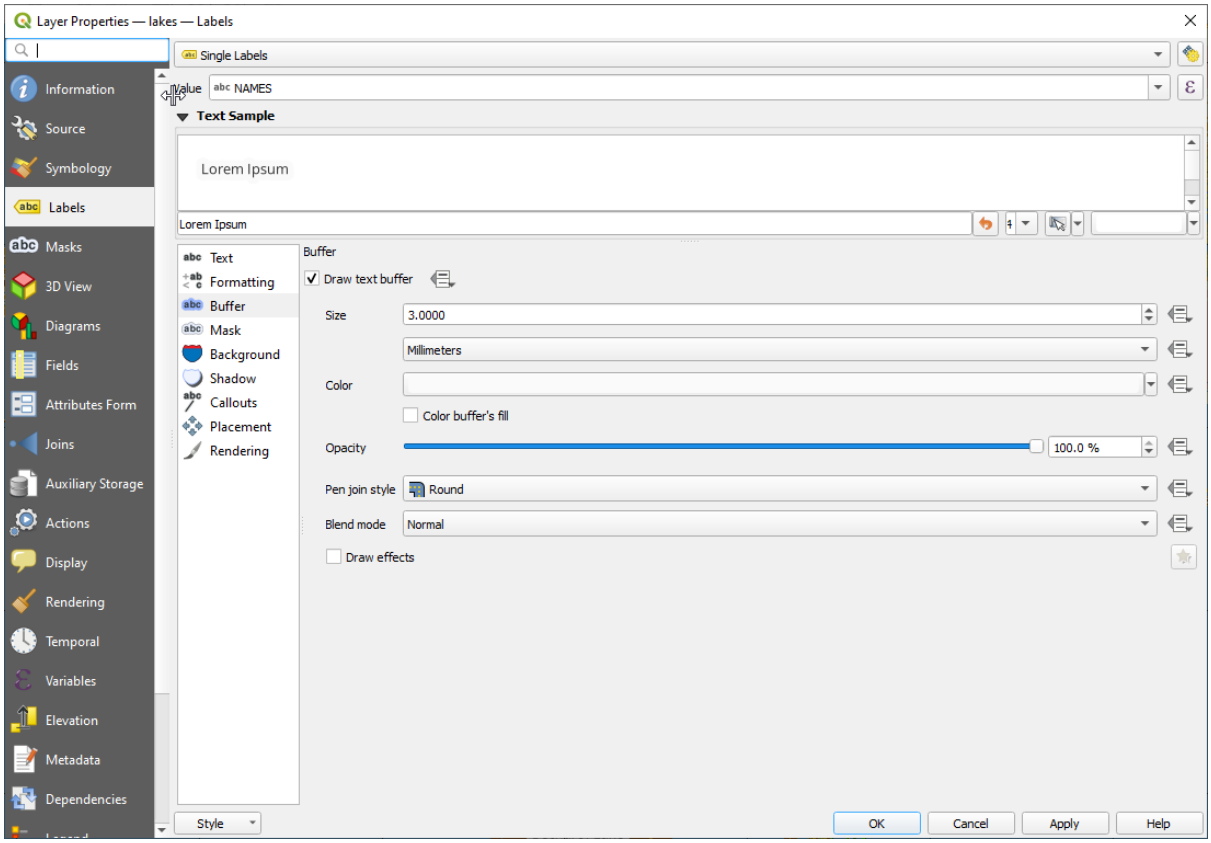


Fig. 1.6: Showing Buffers around Labels

Let's now add some decorations in order to shape the map and export it out of QGIS:

1. Select *View* ► *Decorations* ► *Scale Bar* menu
2. In the dialog that opens, check *Enable Scale Bar* option
3. Customize the options of the dialog as you want
4. Press *Apply*
5. Likewise, from the decorations menu, add more items (north arrow, copyright…) to the map canvas with custom properties.
6. Click *Project* ► *Import/Export* ► *Export Map to Image…*
7. Press *Save* in the opened dialog
8. Select a file location, a format and confirm by pressing *Save* again.
9. Press *Project* ► *Save…* to store your changes as a `.qgz` project file.

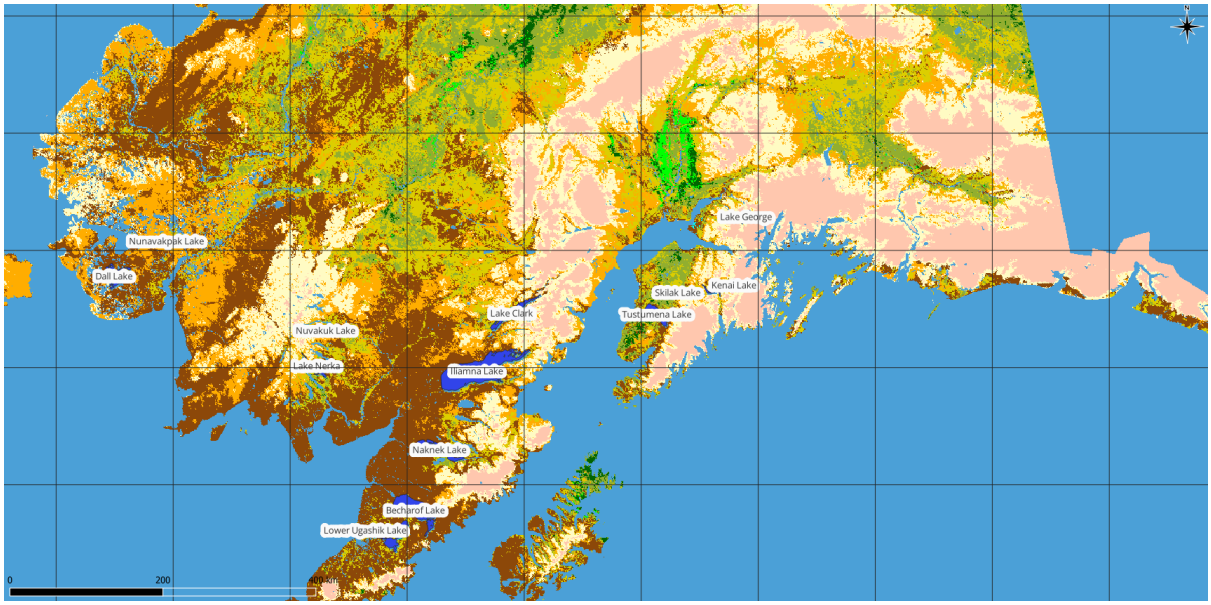


Fig. 1.7: Showing Exported Map with Decorations

That's it! You can see how easy it is to visualize raster and vector layers in QGIS, configure them and generate your map in an image format you can use in other softwares. Let's move on to learn more about the available functionality, features and settings, and how to use them.

> **Note**
>
> To continue learning QGIS through step-by-step exercises, follow the Training manual.

CHAPTER

TWO

WORKING WITH PROJECT FILES

2.1 Introducing QGIS projects

The state of your QGIS session is called a project. QGIS works on one project at a time. A setting can be project-specific or an application-wide default for new projects (see section *Options*). QGIS can save the state of your workspace into a *QGIS project file* using the menu options *Project* ► *Save* or *Project* ► *Save As...*.

Note

If the project has been modified the * symbol will appear in the title bar and QGIS will, by default, ask you if you would like to save the changes. This behavior is controlled by the *Prompt to save project and data source changes when required* setting under *Settings* ► *Options* ► *General*.

You can load existing projects through *Project* ► *Open...*, *Project* ► *New from template* or *Project* ► *Open Recent* ►.

From the *Browser panel* you can *Open Project*, *Extract Symbols...* – opens the style manager to export symbols to XML, add them to the default style, or export as PNG/SVG. You can also access the *File Properties...* dialog. The project file can be expanded to see its layers. The context menu of a layer offers the same actions as elsewhere in the browser.

At startup, a list of *Project Templates* and *Recent Projects* are displayed, including screenshots, names and file paths (for up to ten projects). The *Recent Projects* list is handy to access recently used projects. Double-click an entry to open the project or project template. Right-click an entry to *Pin to List*, *Open Directory...* or *Remove from List*. If you see your project on the *Recent Projects* list but can't find it in your file manager use the *Open Directory...* option to help you locate projects that may be missing, moved or renamed. You can also go to *Clear List* if you want to remove all projects from the *Recent Projects* list. If you have pinned projects, the *Clear List* action will be followed by message box asking whether the pinned projects should also be removed. You can also add a layer to create a new project automatically. The lists will then disappear, giving way to the map canvas.

If you want to clear your session and start fresh, go to *Project* ► *New*. This will prompt you to save the existing project if changes have been made since it was opened or last saved.

When you open a fresh project, the title bar will show `Untitled Project` until you save it.

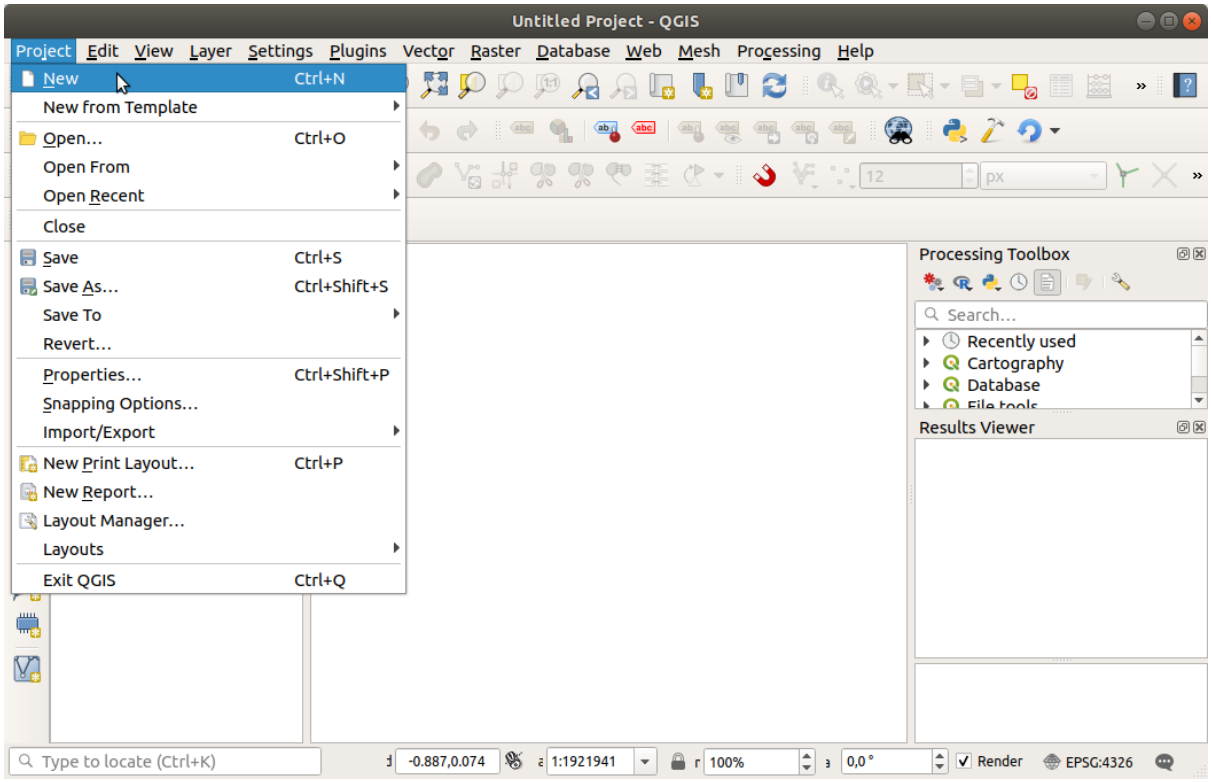


Fig. 2.1: Starting a new project in QGIS

The information saved in a project file includes:

- Layers added
- Which layers can be queried
- Layer properties, including symbolization and styles
- Layer notes
- 2D and 3D map views
- Projection for each map view
- Last viewed extent for each map
- Print layouts
- Print layout elements with settings
- Print layout atlas settings
- Digitizing settings
- Table Relations
- Project Macros
- Project default styles
- Plugins settings
- QGIS Server settings from the OWS settings tab in the Project properties
- Queries stored in the DB Manager

The project file is saved in XML format (see *QGS/QGZ - The QGIS Project File Format*). This means that it is possible to edit the file outside of QGIS if you know what you are doing. The project file format has been updated several times. Project files from older QGIS versions may not work properly any more.

> **Note**
>
> By default, QGIS will warn you of version differences. This behavior is controlled in the *General* tab of *Settings* ► *Options* (☑ *Warn when opening a project file saved with an older version of QGIS*).

Whenever you save a `.qgs` project file in QGIS, a backup of the file is created in the same directory as the project file, with the extension `.qgs~`.

The extension for QGIS projects is `.qgs` but when saving from QGIS, the default is to save using a compressed format with the `.qgz` extension. The `.qgs` file is embedded in the `.qgz` file (a zip archive), together with its associated SQLite database (`.qgd`) for *auxiliary data*. You can get to these files by unzipping the `.qgz` file.

> **Note**
>
> The *Auxiliary Storage Properties* mechanism makes a zipped project particularly useful, since it embeds auxiliary data.

Projects can also be saved/loaded to/from a PostgreSQL, GeoPackage or Oracle database using the following Project menu items:

- *Project* ► *Open from*
- *Project* ► *Save to*

Both menu items have a sub-menu with a list of extra project storage implementations (PostgreSQL, GeoPackage and Oracle). Clicking the action will open a dialog to pick a GeoPackage connection and project, a PostgreSQL connection, schema and project or Oracle connection, owner and project.

Projects stored in GeoPackage, PostgreSQL or Oracle can also be loaded through the QGIS browser panel, either by double-clicking them or by dragging them to the map canvas.

2.2 Handling broken file paths

When opening a project, QGIS may fail to reach some data sources due to unavailable service/database, or to a renamed or moved file. QGIS then opens the *Handle Unavailable Layers* dialog, referencing the unfound layers. You can:

- Double-click in the *Datasource* field, adjust the path of each layer and click *Apply changes*;
- Select a row, press *Browse* to indicate the correct location and click *Apply changes*;
- Press *Auto-Find* to browse the folders and try to automatically fix all or selected broken path(s). Be aware that the browsing may take some time. Then click *Apply changes*.
- Ignore the message and open your project with the broken path(s) by clicking *Keep Unavailable Layers*. Your layer is then displayed in the *Layers* panel, but without any data until you fix the path using the ⚠ Unavailable layer! icon next to it in the *Layers* panel, or *Repair Data Source…* in the layer contextual menu.

  With the *Repair Data Source…* tool, once a layer path has been fixed, QGIS scans through all other broken paths and tries to auto-fix those that have the same broken file path.
- 🗑 *Remove Unavailable Layers* from the project.

Launching QGIS from command line using the *--skipbadlayers* option can help you skip the *Handle Unavailable Layers* dialog at startup.

2.3 Generating output

There are several ways to generate output from your QGIS session. We have already discussed saving as a project file in *Introducing QGIS projects*. Other ways to produce output files are:

- Creating images: *Project* ► *Import/Export* ►  *Export Map to Image…* outputs the map canvas rendering to an image format (PNG, JPG, TIFF…) at custom scale, resolution and size. Including georeference information in the exported image is possible, simply enable  *Append georeference information (embedded or via world file)*. See *Exporting the map view* for more details.
- Exporting to PDF files: *Project* ► *Import/Export* ► *Export Map to PDF…* outputs the map canvas rendering to PDF at custom scale, resolution, and with some advanced settings (simplification, georeferencing, …). See *Exporting the map view* for more details.
- Exporting to DXF files: *Project* ► *Import/Export* ► *Export Project to DXF…* opens a dialog where you can define the 'Symbology mode', the 'Symbology scale' and vector layers you want to export to DXF. Through the 'Symbology mode', symbols from the original QGIS Symbology can be exported with high fidelity (see section *Creating new DXF files*).
- Designing maps: *Project* ►  *New Print Layout…* opens a dialog where you can layout and print the current map canvas (see section *Laying out the maps*).

CHAPTER

THREE

QGIS GUI

The QGIS graphical user interface (GUI) is shown in the figure below (the numbers 1 through 5 in yellow circles indicate important elements of the QGIS GUI, and are discussed below).

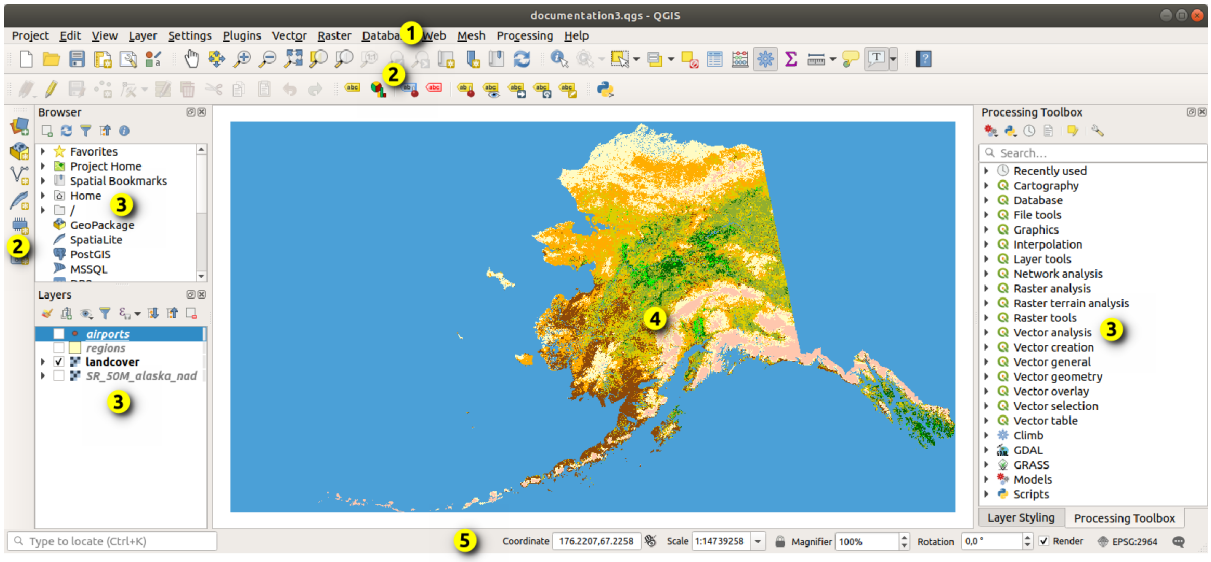


Fig. 3.1: QGIS GUI with Alaska sample data

> **Note**
>
> Your window decorations (title bar, etc.) may appear different depending on your operating system and window manager.

The main QGIS GUI (Fig. 3.1) consists of five components / component types:

1. *Menu Bar*
2. *Toolbars*
3. *Panels*
4. *Map View*
5. *Status Bar*

Scroll down for detailed explanations of these.

3.1 Menu Bar

The Menu bar provides access to QGIS functions using standard hierarchical menus. The Menus, their options, associated icons and keyboard shortcuts are described below. The keyboard shortcuts can be reconfigured (*Settings* ► *Keyboard Shortcuts*).

Most menu options have a corresponding tool and vice-versa. However, the Menus are not organized exactly like the toolbars. The locations of menu options in the toolbars are indicated below in the table. Plugins may add new options to Menus. For more information about tools and toolbars, see *Toolbars*.

> **Note**
>
> QGIS is a cross-platform application. Tools are generally available on all platforms, but they may be placed in different menus, depending on the operating systems. The lists below show the most common locations, including known variations.

3.1.1 Project

The *Project* menu provides access and exit points for *project files*. It provides tools to:

- Create a *New* project file from scratch or use another project file as a template (see *Project files options* for template configuration)
- *Open…* a project from a file, a GeoPackage, PostgreSQL or Oracle database
- *Close* a project or revert it to its last saved state
- *Save* a project in `.qgs` or `.qgz` file format, either as a file or within a GeoPackage, PostgreSQL or Oracle database
- Export the map canvas to different formats or use a *print layout* for more complex output
- Set project properties and snapping options for geometry editing.

Table 3.1: The Project menu items

Menu Option	Shortcut	Tool-bar	Reference
New	`Ctrl+N`	*Project*	*Introducing QGIS projects*
New from template ►			*Introducing QGIS projects*
Open…	`Ctrl+O`	*Project*	*Introducing QGIS projects*
Open from ►			*Introducing QGIS projects*
► *GeoPackage…*			*Introducing QGIS projects*
► *PostgreSQL…*			*Introducing QGIS projects*
► *Oracle…*			*Introducing QGIS projects*
Open Recent ►	`Alt+J` + `R`		*Introducing QGIS projects*
► *Clear List*			*Introducing QGIS projects*
Close			*Introducing QGIS projects*
Save	`Ctrl+S`	*Project*	*Introducing QGIS projects*
Save As…	`Ctrl+Shift+S`	*Project*	*Introducing QGIS projects*
Save to ►			*Introducing QGIS projects*
► *Templates…*			*Introducing QGIS projects*
► *GeoPackage…*			*Introducing QGIS projects*
► *PostgreSQL…*			*Introducing QGIS projects*
Revert…			

continues on next page

Table 3.1 – continued from previous page

Menu Option	Shortcut	Tool-bar	Reference
Properties…	Ctrl+Shift+P		*Project Properties*
Snapping Options…			*Snapping and Digitizing Options*
Import/Export ►			
► *Export Map to Image…*			*Exporting the map view*
► *Export Map to PDF…*			*Exporting the map view*
► *Export Project to DXF…*			*Creating new DXF files*
► *Import Layers from DWG/DXF…*			*Importing a DXF or DWG file*
New Print Layout…	Ctrl+P	*Project*	*Laying out the maps*
New Report…			*Creating a Report*
Layout Manager…		*Project*	*Laying out the maps*
Layouts ►			*Laying out the maps*
Models ►			*The model designer*
Exit QGIS	Ctrl+Q		

Under macOS, the *Exit QGIS* command corresponds to *QGIS* ► *Quit QGIS* (Cmd+Q).

3.1.2 Edit

The *Edit* menu provides most of the native tools needed to edit layer attributes or geometry. To enable *Edit* menu options you need to switch to editing mode by clicking on Toggle editing (see *Editing* for details).

Table 3.2: The Edit menu items

Menu Option	Shortcut	Toolbar	Reference
Undo	Ctrl+Z	*Digitizing*	*Undo and Redo*
Redo	Ctrl+Shift+Z	*Digitizing*	*Undo and Redo*
Cut Features	Ctrl+X	*Digitizing*	*Cutting, Copying and Pasting Features*
Copy Features	Ctrl+C	*Digitizing*	*Cutting, Copying and Pasting Features*
Paste Features	Ctrl+V	*Digitizing*	*Cutting, Copying and Pasting Features*
Paste Features as ►			*Working with the Attribute Table*
► *New Vector Layer…*			*Working with the Attribute Table*
► *Temporary Scratch Layer…*	Ctrl+Alt+V		*Working with the Attribute Table*
Delete Selected		*Digitizing*	*Deleting Selected Features*
Select ►			*Selecting features*
► *Select Feature(s)*		*Selection*	*Selecting features*
► *Select Features by Polygon*		*Selection*	*Selecting features*

continues on next page

Table 3.2 – continued from previous page

Menu Option	Shortcut	Toolbar	Reference
► *Select Features by Freehand*		*Selection*	*Selecting features*
► *Select Features by Radius*		*Selection*	*Selecting features*
► *Select Features by Value…*	F3	*Selection*	*Selecting features*
► *Select Features by Expression…*	Ctrl+F3	*Selection*	*Selecting features*
► *Deselect Features from All Layers*	Ctrl+Alt+A	*Selection*	*Selecting features*
► *Deselect Features from the Current Active Layer*	Ctrl+Shift+A	*Selection*	*Selecting features*
► *Reselect Features*			*Selecting features*
► *Select All Features*	Ctrl+A	*Selection*	*Selecting features*
► *Invert Feature Selection*		*Selection*	*Selecting features*
Add Record	Ctrl+.	*Digitizing*	
Add Point Feature	Ctrl+.	*Digitizing*	*Adding Features*
Add Line Feature	Ctrl+.	*Digitizing*	*Adding Features*
Add Polygon Feature	Ctrl+.	*Digitizing*	*Adding Features*
Add Annotation ►			*Annotation Tools*
► *Form Annotation*		*Annotations*	*Annotation Tools*
► *HTML Annotation*		*Annotations*	*Annotation Tools*
Edit Attributes ►			
► *Modify Attributes of Selected Features*		*Digitizing*	*Editing attribute values*
► *Merge Attributes of Selected Features*		*Advanced Digitizing*	*Merge attributes of selected features*
Edit Geometry ►			
► *Move Feature(s)*		*Advanced Digitizing*	*Move Feature(s)*
► *Copy and Move Feature(s)*		*Advanced Digitizing*	*Move Feature(s)*
► *Rotate Feature(s)*		*Advanced Digitizing*	*Rotate Feature(s)*
► *Scale Feature(s)*		*Advanced Digitizing*	*Scale Feature*
► *Simplify Feature*		*Advanced Digitizing*	*Simplify Feature*
► *Add Ring*		*Advanced Digitizing*	*Add Ring*
► *Add Part*		*Advanced Digitizing*	*Add Part*
► *Fill Ring*		*Advanced Digitizing*	*Fill Ring*
► *Delete Ring*		*Advanced Digitizing*	*Delete Ring*

continues on next page

Table 3.2 – continued from previous page

Menu Option	Shortcut	Toolbar	Reference
► *Delete Part*		*Advanced Digitizing*	*Delete Part*
► *Reshape Features*		*Advanced Digitizing*	*Reshape Features*
► *Offset Curve*		*Advanced Digitizing*	*Offset Curves*
► *Split Features*		*Advanced Digitizing*	*Split Features*
► *Split Parts*		*Advanced Digitizing*	*Split parts*
► *Merge Selected Features*		*Advanced Digitizing*	*Merge selected features*
► *Vertex Tool (All Layers)*		*Digitizing*	*Vertex tool*
► *Vertex Tool (Current Layer)*		*Digitizing*	*Vertex tool*
► *Reverse Line*		*Advanced Digitizing*	*Reverse Line*
► *Trim/extend Feature*		*Advanced Digitizing*	*Trim/Extend Feature*
Rotate Point Symbols		*Advanced Digitizing*	*Rotate Point Symbols*
Offset Point Symbols		*Advanced Digitizing*	*Offset Point Symbols*

Tools that depend on the selected layer geometry type i.e. point, polyline or polygon, are activated accordingly:

Table 3.3: The “Move feature” geometry based icons

Menu Option	Point	Polyline	Polygon
Move Feature(s)			
Copy and Move Feature(s)			

3.1.3 View

The map is rendered in map views. You can interact with these views using the *View* tools. For example, you can:

- Create new 2D or 3D map views next to the main map canvas
- *Zoom or pan* to any place
- Query the attributes or geometry of the displayed features
- Enhance the map view with preview modes, annotations or decorations
- Access any panel or toolbar

The menu also allows you to reorganize the QGIS interface itself using actions like:

- *Toggle Full Screen Mode*: covers the whole screen while hiding the title bar

- *Toggle Panel Visibility*: shows or hides enabled *panels* - useful when digitizing features (for maximum canvas visibility) as well as for (projected/recorded) presentations using QGIS main canvas
- *Toggle Map Only*: hides panels, toolbars, menus and status bar and only shows the map canvas. Combined with the full screen option, it makes your screen display only the map

Table 3.4: The View menu items

Menu Option	Shortcut	Toolbar	Reference
New Map View	`Ctrl+M`	*Map Navigation*	*2D Map View*
3D Map Views ►			*3D Map View*
► *New 3D Map View*	`Ctrl+Alt+M`	*Map Navigation*	*3D Map View*
► *Manage 3D Map Views*			*3D Map View*
Pan Map		*Map Navigation*	*Exploring the map view*
Pan Map to Selection		*Map Navigation*	*Exploring the map view*
Zoom In	`Ctrl+Alt++`	*Map Navigation*	*Exploring the map view*
Zoom Out	`Ctrl+Alt+-`	*Map Navigation*	*Exploring the map view*
Data Filtering ►			*2D Map View*
► *Temporal Controller*		*Map Navigation*	*The temporal controller panel*
► *Elevation Controller*			*The Elevation Controller panel*
Identify Features	`Ctrl+Shift+I`	*Attributes*	*Identifying Features*
Measure ►		*Attributes*	*Measuring*
► *Measure Line*	`Ctrl+Shift+M`	*Attributes*	*Measuring*
► *Measure Area*	`Ctrl+Shift+J`	*Attributes*	*Measuring*
► *Measure Bearing*		*Attributes*	*Measuring*
► *Measure Angle*		*Attributes*	*Measuring*
Statistical Summary		*Attributes*	*Statistical Summary Panel*
Elevation Profile			*Elevation Profile View*
Zoom Full	`Ctrl+Shift+F`	*Map Navigation*	*Exploring the map view*
Zoom To Selection	`Ctrl+J`	*Map Navigation*	*Exploring the map view*
Zoom To Layer(s)		*Map Navigation*	*Exploring the map view*
Zoom To Native Resolution (100%)		*Map Navigation*	*Exploring the map view*
Zoom Last		*Map Navigation*	*Exploring the map view*

continues on next page

Table 3.4 – continued from previous page

Menu Option	Shortcut	Toolbar	Reference
Zoom Next		*Map Navigation*	*Exploring the map view*
Decorations ►	Alt+V + D		*Decorating the map*
► *Grid…*			*Grid*
► *Scale Bar…*			*Scale Bar*
► *Image…*			*Image Decoration*
► *North Arrow…*			*North Arrow*
► *Title Label…*			*Title Label*
► *Copyright Label…*			*Copyright Label*
► *Layout Extents…*			*Layout Extents*
Preview mode ►			
► *Normal*			
► *Simulate Monochrome*			
► *Simulate Achromatopsia Color Blindness (Grayscale)*			
► *Simulate Protanopia Color Blindness (No Red)*			
► *Simulate Deuteranopia Color Blindness (No Green)*			
► *Simulate Tritanopia Color Blindness (No Blue)*			
Show Map Tips		*Attributes*	*Display Properties*
New Spatial Bookmark…	Ctrl+B	*Map Navigation*	*Bookmarking extents on the map*
Show Spatial Bookmarks	Ctrl+Shift+B	*Map Navigation*	*Bookmarking extents on the map*
Show Spatial Bookmark Manager			*Bookmarking extents on the map*
Refresh	F5	*Map Navigation*	
Layer Visibility ►			*Layers Panel*
► *Show All Layers*	Ctrl+Shift+U		*Layers Panel*
► *Hide All Layers*	Ctrl+Shift+H		*Layers Panel*
► *Show Selected Layers*			*Layers Panel*
► *Hide Selected Layers*			*Layers Panel*
► *Toggle Selected Layers*			*Layers Panel*
► *Toggle Selected Layers Independently*			*Layers Panel*
► *Hide Deselected Layers*			*Layers Panel*
Panels ►			*Panels and Toolbars*
Toolbars ►			*Panels and Toolbars*
Toggle Full Screen Mode	F11		
Toggle Panel Visibility	Ctrl+Tab		
Toggle Map Only	Ctrl+Shift+Tab		

Under Linux KDE, *Panels* ►, *Toolbars* ► and *Toggle Full Screen Mode* are in the *Settings* menu.

3.1.4 Layer

The *Layer* menu provides a large set of tools to *create* new data sources, *add* them to a project or *save modifications* to them. Using the same data sources, you can also:

- *Duplicate* a layer to generate a copy where you can modify the name, style (symbology, labels, …), joins, … The copy uses the same data source as the original.
- *Copy* and *Paste* layers or groups from one project to another as a new instance whose properties can be modified independently. As for *Duplicate*, the layers are still based on the same data source.
- or *Embed Layers and Groups…* from another project, as read-only copies which you cannot modify (see *Embedding layers from external projects*)

The *Layer* menu also contains tools to configure, copy or paste layer properties (style, scale, CRS…).

Table 3.5: The Layer menu items

Menu Option	Shortcut	Toolbar	Reference
Data Source Manager	`Ctrl+L`	*Data Source Manager*	*Opening Data*
Create Layer ►			*Creating new vector layers*
► *New GeoPackage Layer…*	`Ctrl+Shift+N`	*Data Source Manager*	*Creating a new GeoPackage layer*
► *New Shapefile Layer…*		*Data Source Manager*	*Creating a new Shapefile layer*
► *New SpatiaLite Layer…*		*Data Source Manager*	*Creating a new SpatiaLite layer*
► *New Temporary Scratch Layer…*		*Data Source Manager*	*Creating a new Temporary Scratch Layer*
► *New Mesh Layer…*		*Data Source Manager*	*Creating a new Mesh layer*
► *New GPX Layer…*		*Data Source Manager*	*Creating a new GPX layer*
► *New Virtual Layer…*		*Data Source Manager*	*Creating virtual layers*
Add Layer ►			*Opening Data*
► *Add Vector Layer……*	`Ctrl+Shift+V`	*Manage Layers*	*Loading a layer from a file*
► *Add Raster Layer…*	`Ctrl+Shift+R`	*Manage Layers*	*Loading a layer from a file*
► *Add Mesh Layer…*		*Manage Layers*	*Loading a mesh layer*
► *Add Delimited Text Layer…*	`Ctrl+Shift+T`	*Manage Layers*	*Importing a delimited text file*
► *Add PostgreSQL Layer…*	`Ctrl+Shift+D`	*Manage Layers*	*Database related tools*
► *Add SpatiaLite Layer…*	`Ctrl+Shift+L`	*Manage Layers*	*SpatiaLite Layers*
► *Add MS SQL Server Layer…*		*Manage Layers*	*Database related tools*
► *Add Oracle Spatial Layer…*		*Manage Layers*	*Database related tools*
► *Add SAP HANA Spatial Layer…*		*Manage Layers*	*Database related tools*

continues on next page

Table 3.5 – continued from previous page

Menu Option	Shortcut	Toolbar	Reference
► *Add/Edit Virtual Layer…*		*Manage Layers*	*Creating virtual layers*
► *Add WMS/WMTS Layer…*	Ctrl+Shift+W	*Manage Layers*	*Loading WMS/WMTS Layers*
► *Add XYZ Layer…*			*Using XYZ Tile services*
► *Add WCS Layer…*		*Manage Layers*	*WCS Client*
► *Add WFS / OGC API - Features Layer…*		*Manage Layers*	*WFS and WFS-T Client*
► *Add ArcGIS REST Server Layer…*		*Manage Layers*	*Using ArcGIS REST Servers*
► *Add Vector Tile Layer…*			*Using Vector Tiles services*
► *Add Point Cloud Layer…*			*Working with Point Clouds*
► *Add GPX Layer…*			*Introducing GNSS/GPS Data*
Embed Layers and Groups…			*Embedding layers from external projects*
Add from Layer Definition File…			*Layer definition file*
Georeferencer…			*Georeferencer*
Copy Style			*Save and Share Layer Properties*
Paste Style			*Save and Share Layer Properties*
Copy Layer			
Paste Layer/Group			
Open Attribute Table	F6	*Attributes*	*Working with the Attribute Table*
Filter Attribute Table ►			*Working with the Attribute Table*
► *Open Attribute Table (Selected Features)*	Shift+F6	*Attributes*	*Working with the Attribute Table*
► *Open Attribute Table (Visible Features)*	Ctrl+F6	*Attributes*	*Working with the Attribute Table*
► *Open Attribute Table (Edited and New Features)*		*Attributes*	*Working with the Attribute Table*
Toggle Editing		*Digitizing*	*Digitizing an existing layer*
Save Layer Edits		*Digitizing*	*Saving Edited Layers*
Current Edits ►		*Digitizing*	*Saving Edited Layers*
► *Save for Selected Layer(s)*		*Digitizing*	*Saving Edited Layers*
► *Rollback for Selected Layer(s)*		*Digitizing*	*Saving Edited Layers*
► *Cancel for Selected Layer(s)*		*Digitizing*	*Saving Edited Layers*
► *Save for all Layers*		*Digitizing*	*Saving Edited Layers*
► *Rollback for all Layers*		*Digitizing*	*Saving Edited Layers*
► *Cancel for all Layers*		*Digitizing*	*Saving Edited Layers*
Save As…			*Creating new layers from an existing layer*
Save As Layer Definition File…			*Layer definition file*

continues on next page

Table 3.5 – continued from previous page

Menu Option	Shortcut	Toolbar	Reference
Remove Layer/Group	`Ctrl+D`		
Duplicate Layer(s)			
Set Scale Visibility of Layer(s)			*Visibility Scale Selector*
Set CRS of Layer(s)	`Ctrl+Shift+C`		*Layer Coordinate Reference Systems*
Set Project CRS from Layer			*Project Coordinate Reference Systems*
Layer Properties…			• *vector* • *raster* • *mesh* • *point cloud* • *vector tiles*
Filter…	`Ctrl+F`		*Query Builder*
Labeling			*Labels Properties*
Show in Overview			*Overview Panel*
Show All in Overview			*Overview Panel*
Hide All from Overview			*Overview Panel*

3.1.5 Settings

Table 3.6: The Settings menu items

Menu Option	Reference
User Profiles ►	*Working with User Profiles*
► *default*	*Working with User Profiles*
► *Open Active Profile Folder*	*Working with User Profiles*
► *New Profile…*	*Working with User Profiles*
Style Manager…	*The Style Manager*
Custom Projections…	*Custom Coordinate Reference System*
Keyboard Shortcuts…	*Keyboard shortcuts*
Interface Customization…	*Customization*
Options…	*Options*

Under Linux KDE, you'll find more tools in the *Settings* menu such as *Panels* ►, *Toolbars* ► and *Toggle Full Screen Mode*.

3.1.6 Plugins

Table 3.7: The Plugins menu items

Menu Option	Shortcut	Toolbar	Reference
Manage and Install Plugins…			*The Plugins Dialog*
Python Console	`Ctrl+Alt+P`	*Plugins*	*QGIS Python console*

When starting QGIS for the first time not all core plugins are loaded.

3.1.7 Vector

This is what the *Vector* menu looks like if all core plugins are enabled.

Table 3.8: The Vector menu items

Menu Option	Shortcut	Tool-bar	Reference
Check Geometries…			*Geometry Checker Plugin*
Topology Checker		*Vector*	*Topology Checker Plugin*
Geoprocessing Tools ►	`Alt+O` + `G`		
► *Buffer…*			*Buffer*
► *Clip…*			*Clip*
► *Convex Hull…*			*Convex hull*
► *Difference…*			*Difference*
► *Dissolve…*			*Dissolve*
► *Intersection…*			*Intersection*
► *Symmetrical Difference…*			*Symmetrical difference*
► *Union…*			*Union*
► *Eliminate Selected Polygons…*			*Eliminate selected polygons*
Geometry Tools ►	`Alt+O` + `E`		
► *Centroids…*			*Centroids*
► *Collect Geometries…*			*Collect geometries*
► *Extract Vertices…*			*Extract vertices*
► *Multipart to Singleparts…*			*Multipart to singleparts*
► *Polygons to Lines…*			*Polygons to lines*
► *Simplify…*			*Simplify*
► *Check Validity…*			*Check validity*
► *Delaunay Triangulation…*			*Delaunay triangulation*
► *Densify by Count…*			*Densify by count*
► *Add Geometry Attributes…*			*Add geometry attributes*
► *Lines to Polygons…*			*Lines to polygons*

continues on next page

Table 3.8 – continued from previous page

Menu Option	Shortcut	Tool-bar	Reference
► *Voronoi Polygons…*			*Voronoi polygons*
Analysis Tools ►	Alt+O + A		
► *Line Intersection…*			*Line intersections*
► *Mean Coordinate(s)…*			*Mean coordinate(s)*
► *Basic Statistics for Fields…*			*Basic statistics for fields*
► *Count Points in Polygon…*			*Count points in polygon*
► *Distance Matrix…*			*Distance matrix*
► *List Unique Values…*			*List unique values*
► *Nearest Neighbour Analysis…*			*Nearest neighbour analysis*
► *Sum Line Lengths…*			*Sum line lengths*
Data Management Tools ►	Alt+O + D		
► *Merge Vector Layers…*			*Merge vector layers*
► *Reproject Layer…*			*Reproject layer*
► *Create Spatial Index…*			*Create spatial index*
► *Join Attributes by Location…*			*Join attributes by location*
► *Split Vector Layer…*			*Split vector layer*
Research Tools ►	Alt+O + R		
► *Create Grid…*			*Create grid*
► *Extract Layer Extent…*			*Extract layer extent*
► *Random Points in Extent…*			*Random points in extent*
► *Random Points in Polygons…*			*Random points in polygons*
► *Random Points on Lines…*			*Random points on lines*
► *Select by Location…*			*Select by location*
► *Select Within Distance…*			*Select within distance*
► *Random Points in Layer Bounds…*			*Random points in layer bounds*
► *Random Points Inside Polygons…*			*Random points inside polygons*
► *Random Selection…*			*Random selection*
► *Random Selection Within Subsets…*			*Random selection within subsets*
► *Regular Points…*			*Regular points*

By default, QGIS adds *Processing* algorithms to the *Vector* menu, grouped by sub-menus. This provides shortcuts for many common vector-based GIS tasks from different providers. If not all these sub-menus are available, enable the Processing plugin in *Plugins* ► *Manage and Install Plugins…*.

Note that the list of algorithms and their menu can be modified/extended with any Processing algorithms (read *Configuring the Processing Framework*) or some external *plugins*.

3.1.8 Raster

This is what the *Raster* menu looks like if all core plugins are enabled.

Table 3.9: The Raster menu items

Menu Option	Shortcut	Tool-bar	Reference
Raster calculator…			*Raster Calculator*
Align Rasters…			*Align rasters*
Analysis ►			
► *Aspect…*			*Aspect*
► *Fill nodata…*			*Fill NoData*
► *Grid (Moving Average)…*			*Grid (Moving average)*
► *Grid (Data Metrics)…*			*Grid (Data metrics)*
► *Grid (Inverse Distance to a Power)…*			*Grid (Inverse distance to a power)*
► *Grid (Nearest Neighbor)…*			*Grid (IDW with nearest neighbor searching)*
► *Near Black…*			*Near black*
► *Hillshade…*			*Hillshade*
► *Proximity (Raster Distance)…*			*Proximity (raster distance)*
► *Roughness…*			*Roughness*
► *Sieve…*			*Sieve*
► *Slope…*			*Slope*
► *Topographic Position Index (TPI)…*			*Topographic Position Index (TPI)*
► *Terrain Ruggedness Index (TRI)…*			*Terrain Ruggedness Index (TRI)*
Projections ►			
► *Assign Projection…*			*Assign projection*
► *Extract Projection…*			*Extract projection*
► *Warp (Reproject)…*			*Warp (reproject)*
Miscellaneous ►			
► *Build Virtual Raster…*			*Build virtual raster*
► *Raster Information…*			*Raster information*
► *Merge…*			*Merge*
► *Build Overviews (Pyramids)…*			*Build overviews (pyramids)*
► *Tile Index…*			*Tile index*
Extraction ►			
► *Clip Raster by Extent…*			*Clip raster by extent*
► *Clip Raster by Mask Layer…*			*Clip raster by mask layer*
► *Contour…*			*Contour*
Conversion ►			
► *PCT to RGB…*			*PCT to RGB*

continues on next page

Table 3.9 – continued from previous page

Menu Option	Shortcut	Tool-bar	Reference
► *Polygonize (Raster to Vector)…*			*Polygonize (raster to vector)*
► *Rasterize (Vector to Raster)…*			*Rasterize (vector to raster)*
► *RGB to PCT…*			*RGB to PCT*
► *Translate (Convert Format)…*			*Translate (convert format)*

By default, QGIS adds *Processing* algorithms to the *Raster* menu, grouped by sub-menus. This provides a shortcut for many common raster-based GIS tasks from different providers. If not all these sub-menus are available, enable the Processing plugin in *Plugins* ► *Manage and Install Plugins…*.

Note that the list of algorithms and their menu can be modified/extended with any Processing algorithms (read *Configuring the Processing Framework*) or some external *plugins*.

3.1.9 Database

The *Database* menu provides tools for Database interaction. If no database-related plugins are enabled, there will be no *Database* menu. This is what the *Database* menu looks like if all the core plugins are enabled.

Table 3.10: The Database menu items

Menu Option	Shortcut	Toolbar	Reference
Offline editing…	Alt+D + O		*Offline Editing Plugin*
► *Convert to Offline Project…*		*Database*	*Offline Editing Plugin*
► *Synchronize*		*Database*	*Offline Editing Plugin*
Query History…		*Database*	*Creating SQL Query Layers*
DB Manager…		*Database*	*DB Manager Plugin*

When starting QGIS for the first time not all core plugins are loaded.

3.1.10 Web

This is what the *Web* menu looks like if all the core plugins are enabled. If no web plugins are enabled, there will be no *Web* menu.

Table 3.11: The Web menu items

Menu Option	Shortcut	Toolbar	Reference
MetaSearch ►	Alt+W + M		*MetaSearch Catalog Client*
► *Metasearch*		*Web*	*MetaSearch Catalog Client*
► *Help*			*MetaSearch Catalog Client*

When starting QGIS for the first time not all core plugins are loaded.

3.1.11 Mesh

The *Mesh* menu provides tools needed to manipulate *mesh layers*. Third-party plugins can add items to this menu.

Table 3.12: The Mesh menu items

Menu Option	Shortcut	Toolbar	Reference
Mesh Calculator…			*Mesh Calculator*
Reindex Faces and Vertices			*Reindexing meshes*

3.1.12 Processing

Table 3.13: The Processing menu items

Menu Option	Shortcut	Tool-bar	Reference
Toolbox	`Ctrl+Alt+T`		*The Toolbox*
Model Designer…	`Ctrl+Alt+G`		*The model designer*
History…	`Ctrl+Alt+H`		*The history manager*
Results Viewer	`Ctrl+Alt+R`		*Configuring external applications*
Edit Features In-Place			*The Processing in-place layer modifier*

When starting QGIS for the first time not all core plugins are loaded.

3.1.13 Help

Table 3.14: The Help menu items

Menu Option	Shortcut	Toolbar	Reference
Help Contents	`F1`	*Help*	
API Documentation			
Plugins ►			
Report an Issue			
Need commercial support?			
QGIS Home Page	`Ctrl+H`		
Check QGIS Version			
About			
QGIS Sustaining Members			

3.1.14 QGIS

This menu is only available under macOS and contains some OS related commands.

Table 3.15: The QGIS menu items

Menu Option	Shortcut
Preferences	
About QGIS	
Hide QGIS	
Show All	
Hide Others	
Quit QGIS	`Cmd+Q`

Preferences corresponds to *Settings* ► *Options*, *About QGIS* corresponds to *Help* ► *About* and *Quit QGIS* corresponds to *Project* ► *Exit QGIS* for other platforms.

3.2 Panels and Toolbars

From the *View* menu (or *Settings*), you can switch QGIS widgets (*Panels* ►) and toolbars (*Toolbars* ►) on and off. To (de)activate any of them, right-click the menu bar or toolbar and choose the item you want. Panels and toolbars can be moved and placed wherever you like within the QGIS interface. The list can also be extended with the activation of *Core or external plugins*.

3.2.1 Toolbars

The toolbars provide access to most of the functions in the menus, plus additional tools for interacting with the map. Each toolbar item has pop-up help available. Hover your mouse over the item and a short description of the tool's purpose will be displayed.

Available toolbars are:

Table 3.16: QGIS Toolbars

Toolbar name	Main Reference for tools
Advanced Digitizing	*Advanced digitizing*
Annotations	*Annotation Tools*
Attributes	*Working with the Attribute Table*, *General Tools*
Data Source Manager	*Managing Data Source*
Database	*DB Manager Plugin*
Digitizing	*Digitizing an existing layer*
GPS	*Live GPS tracking*
GRASS	*GRASS GIS Integration*
Help	
Label	*The Label Toolbar*
Manage Layers	*Opening Data*
Map Navigation	*Exploring the map view*
Mesh Digitizing	*Editing a mesh layer*
Plugins	*Plugins*
Project	*Working with Project Files*, *Laying out the maps*, *The Style Library*
Processing Algorithms	*Configuring the Processing Framework*
Raster	*Plugins*
Selection	*Selecting features*
Shape digitizing	*Shape digitizing*
Snapping	*Setting the snapping tolerance and search radius*
Vector	*Plugins*
Web	*Plugins*, *MetaSearch Catalog Client*

Note

Third-party plugins can extend the default toolbar with their own tools or provide their own toolbar.

Tip

Restoring toolbars

If you have accidentally hidden a toolbar, you can get it back using *View* ► *Toolbars* ► (or *Settings* ► *Toolbars* ►). If, for some reason, a toolbar (or any other widget) totally disappears from the interface, you'll find tips to get it back at *restoring initial GUI*.

3.2.2 Panels

QGIS provides many panels. Panels are special widgets that you can interact with (selecting options, checking boxes, filling values…) to perform more complex tasks.

Below is a list of the default panels provided by QGIS:

Table 3.17: QGIS Panels

Panel name	Shortcut	Reference
Advanced Digitizing	`Ctrl+4`	*The Advanced Digitizing panel*
Browser	`Ctrl+2`	*The Browser Panel*
Browser (2)		*The Browser Panel*
Debugging/Development Tools	`F12`	*Debugging/Development Tools Panel*
Elevation Profile		
Geometry Validation		*Digitizing Properties*
GPS Information	`Ctrl+0`	*Live GPS tracking*
GRASS Tools		*GRASS GIS Integration*
Layer Order	`Ctrl+9`	*Layer Order Panel*
Layer Styling	`Ctrl+3`	*Layer Styling Panel*
Layers	`Ctrl+1`	*Layers Panel*
Log Messages		*Log Messages Panel*
Overview	`Ctrl+8`	*Overview Panel*
Processing Toolbox		*The Toolbox*
Results Viewer		*The Toolbox*
Snapping and Digitizing Options		*Setting the snapping tolerance and search radius*
Spatial Bookmark Manager	`Ctrl+7`	*Bookmarking extents on the map*
Statistics	`Ctrl+6`	*Statistical Summary Panel*
Temporal Controller		*The temporal controller panel*
Tile Scale		*Tilesets*
Undo/Redo	`Ctrl+5`	*Undo/Redo Panel*
Vertex Editor		*The Vertex Editor Panel*

3.3 Status Bar

The status bar provides you with general information about the map view and processed or available actions, and offers you tools to manage the map view.

3.3.1 Locator bar

On the left side of the status bar, the locator bar, a quick search widget, helps you find and run any feature or options in QGIS:

1. Click in the text widget to activate the locator search bar or press `Ctrl+K`.
2. Type a text associated with the item you are looking for (name, tag, keyword, …). By default, results are returned for the enabled locator filters, but you can limit the search to a certain scope by prefixing your text with the *locator filters* prefix, ie. typing `l cad` will return only the layers whose name contains `cad`.

   The filter can also be selected with a double-click in the menu that shows when accessing the locator widget.
3. Click on a result to execute the corresponding action, depending on the type of item.

Tip

Limit the lookup to particular field(s) of the active layer

By default, a search with the "active layer features" filter (`f`) runs through the whole attribute table of the layer. You can limit the search to a particular field using the `@` prefix. E.g., `f @name sal` or `@name sal` returns only the features whose "name" attribute contains 'sal'. Text autocompletion is active when writing and the suggestion can be applied using `Tab` key.

A more advanced control on the queried fields is possible from the layer *Fields* tab. Read *Fields Properties* for details.

Searching is handled using threads, so that results always become available as quickly as possible, even if slow search filters are installed. They also appear as soon as they are encountered by a filter, which means that e.g. a file search filter will show results one by one as the file tree is scanned. This ensures that the UI is always responsive, even if a very slow search filter is present (e.g. one which uses an online service).

Note

The Nominatim locator tool may behave differently (no autocompletion search, delay of fetching results, …) with respect to the OpenStreetMap Nominatim usage policy.

Tip

Quick access to the locator's configurations

Click on the  icon inside the locator widget on the status bar to display the list of filters you can use and a *Configure* entry that opens the *Locator* tab of the *Settings* ► *Options…* menu.

3.3.2 Reporting actions

In the area next to the locator bar, a summary of actions you've carried out will be shown when needed (such as selecting features in a layer, removing layer, pan distance and direction) or a long description of the tool you are hovering over (not available for all tools).

In case of lengthy operations, such as gathering of statistics in raster layers, executing Processing algorithms or rendering several layers in the map view, a progress bar is displayed in the status bar.

3.3.3 Control the map canvas

The  *Coordinate* option shows the current position of the mouse, following it while moving across the map view. You can set the units (and precision) in the *Project* ► *Properties…* ► *General* tab. Click on the small button at the left of the textbox to toggle between the Coordinate option and the  *Extents* option that displays the coordinates of the current bottom-left and top-right corners of the map view in map units.

Next to the coordinate display you will find the *Scale* display. It shows the scale of the map view. There is a scale selector, which allows you to choose between *predefined and custom scales*.

On the right side of the scale display, press the  button to lock the scale to use the magnifier to zoom in or out. The magnifier allows you to zoom in to a map without altering the map scale, making it easier to tweak the positions of labels and symbols accurately. The magnification level is expressed as a percentage. If the *Magnifier* has a level of 100%, then the current map is not magnified, i.e. is rendered at accurate scale relative to the monitor's resolution (DPI). A default magnification value can be defined within *Settings* ► *Options* ► *Rendering* ► *Rendering Behavior*, which is very useful for high-resolution screens to enlarge small symbols. In addition, a setting in *Settings* ► *Options* ► *Canvas & Legend* ► *DPI* controls whether QGIS respects each monitor's physical DPI or uses the overall system logical DPI.

To the right of the magnifier tool you can define a current clockwise rotation for your map view in degrees.

On the right side of the status bar, the  *Render* checkbox can be used to temporarily suspend the map view rendering (see section *Controlling map rendering*).

To the right of the  *Render* function, you find the  *EPSG:code* button showing the current project CRS. Clicking on this opens the *Project Properties* dialog and lets you reproject the map view or adjust any other project property.

Tip

Calculating the Correct Scale of Your Map Canvas

When you start QGIS, the default CRS is `WGS 84 (EPSG 4326)` and units are degrees. This means that QGIS will interpret any coordinate in your layer as specified in degrees. To get correct scale values, you can either manually change this setting in the *General* tab under *Project* ► *Properties…* (e.g. to meters), or you can use the  EPSG:code icon seen above. In the latter case, the units are set to what the project projection specifies (e.g., `+units=us-ft`).

Note that CRS choice on startup can be set in *Settings* ► *Options* ► *CRS Handling*.

3.3.4 Messaging

The  Messages button next to it opens the *Log Messages Panel* which has information on underlying processes (QGIS startup, plugins loading, processing tools…)

CHAPTER

FOUR

THE BROWSER PANEL

- *Resources that can be opened / run from the Browser*
- *Browser panel top-level entries*
  - *Favorites*
  - *Spatial Bookmarks*
  - *Project Home*
  - *Drives and file system*
  - *Data type*
  - *Database entries*
  - *Tiles and Web Services*
  - *Project and supported file types*

The QGIS Browser panel is a great tool for browsing, searching, inspecting, copying and loading QGIS resources. Only resources that QGIS knows how to handle are shown in the browser.

Using the Browser panel you can locate, inspect and add data, as described in *The Browser Panel*. In addition, the Browser panel supports drag and drop of many QGIS resources, such as project files, Python scripts, Processing scripts and Processing models.

Python scripts, Processing scripts and Processing models can also be opened for editing in an external editor and the graphical modeller.

You can drag and drop layers from the *Layers* panel to the *Browser* panel, for instance into a GeoPackage or a PostgreSQL database.

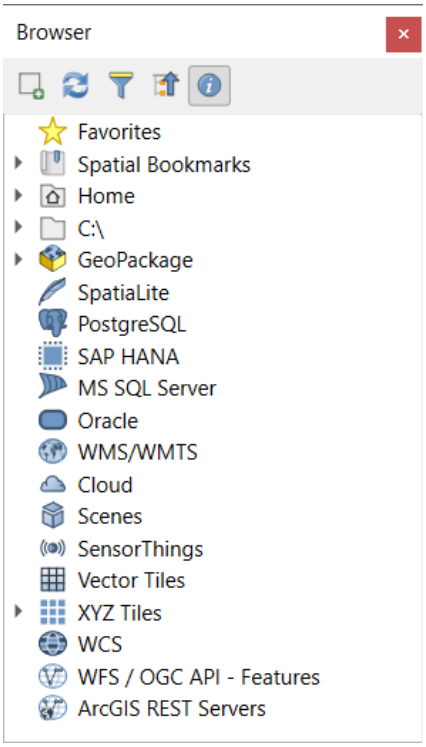


Fig. 4.1: The Browser panel

The browser panel (Fig. 4.1) is organised as an expandable hierarchy with some fixed top-level entries that organise the resources handled by the browser. Node entries are expanded by clicking on to the left of the entry name. A branch is collapsed by clicking on . The Collapse All button collapses all top-level entries.

In *Settings* ► *Interface Customization* it is possible to disable resources. If you, for instance, would not like to show Python scripts in the browser, you can uncheck the *Browser* ► *py* entry, and if you want to get rid of your home folder in the browser, you can uncheck the *Browser* ► *special:Home* entry.

A filter (Filter Browser) can be used for searching based on entry names (both leaf entries and node entries in the hierarchy). Using the Options pull-down menu next to the filter text field, you can

- toggle *Case Sensitive* search
- set the *Filter pattern syntax* to one of
  - *Normal*
  - *Wildcard(s)*
  - *Regular Expressions*

The *Properties widget*, showing useful information about some entries / resources, can be enabled / disabled using the Enable/disable properties widget button. When enabled, it opens at the bottom of the browser panel, as shown in Fig. 4.2.

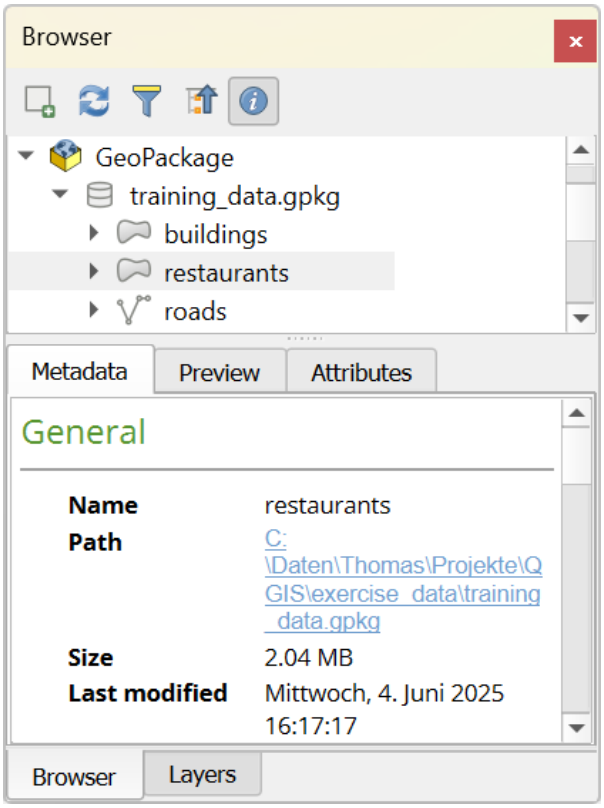


Fig. 4.2: The properties widget

A second browser panel can be opened by activating the *Browser (2)* panel in *View* ► *Panels*. Having two browser panels can be useful when copying layers between resources that are locationed deep down in different branches of the browser hierarchy.

4.1 Resources that can be opened / run from the Browser

A lot can be accomplished in the Browser panel

- Add vector, raster and mesh layers to your map by double-clicking, dragging onto the map canvas or clicking the Add Selected Layers button (after selecting layers)
- Run Python scripts (including Processing algorithms) by double-clicking or dragging onto the map canvas
- Run models by double-clicking or dragging onto the map canvas
- *Extract Symbols…* from QGIS Project files using the context menu
- Open files with their default applications (*Open <file type> Externally…* in the context menu). Examples: HTML files, spreadsheets, images, PDFs, text files, …
- Copy entries
- Rename and delete (multiple) layers (context menu: *Manage* ►)
- Open a file explorer window and directly select the file *Show in Files*

Resource specific actions are listed for the different resource groups sorted under the top-level entries listed below.

4.2 Browser panel top-level entries

4.2.1 Favorites

Often used file system locations can be tagged as favorites. The ones you have tagged will appear here.

In addition to the operations described under *Home*, the context menu allows you to *Rename Favorite…* and *Remove Favorite*.

4.2.2 Spatial Bookmarks

This is where you will find your spatial bookmarks, organised into *Project Bookmarks* and *User Bookmarks*.

From the top level context menu, you can create a bookmark (*New Spatial Bookmark…*), *Show the Spatial Bookmark Manager*, *Import Spatial Bookmarks…* and *Export Spatial Bookmarks…*.

For bookmark group entries you can *Export Spatial Bookmarks…*, create a bookmark (*New Spatial Bookmark…*), *Rename Bookmark Group*, and *Delete Bookmark Group*.

For bookmark entries you can *Zoom to Bookmark*, *Edit Spatial Bookmark…* and *Delete Spatial Bookmark*

4.2.3 Project Home

Available once the project file has been saved, the *Project home* entry is a folder containing data and other contents (scripts, models, text, …) that may be used within the current project. Displayed in the *Browser* panel, it allows you to quickly access data and other files of the project.

It defaults to the project file folder but can be changed through the *Project* ► *Properties…* ► *General* ► *Project home* option, or by right-clicking on the *Project Home* item of the Browser panel and selecting *Set project home…*. Customizing that folder is especially useful in contexts where QGIS projects are not stored in the root folder of an organisational 'project', along with datasets.

4.2.4 Drives and file system

The next items of the *Browser* panel depend on the OS in use and concern the top level entries of its file system.

They are mainly:

- The *Home* folder, pointing to the current user home folder
- on Unix-based machines, the root `/` folder
- the connected drives, either local or network. Depending on the OS, they are directly listed (eg, `C:\`, `D:\`) or through the `/Volumes` entry.

From the contextual menu of each of these folders or drives, you can:

- refresh the contents
- create a *New* ► subitem that is a *Directory*, *GeoPackage* or ESRI *Shapefile* format dataset
- hide the directory (*Hide from Browser*)
- *Set color*: customize the folder icon color, aiding in rapid browser navigation of complex folder structures
- enable *Scanning*:
  - *Monitor for changes*: allows to manually control whether a particular directory should be monitored and automatically updated. This setting applies to the selected directory and all subdirectories. This means that you can manually opt-in to monitoring of network drives if you know there's no issue, or manually opt-out of monitoring of large directories which you don't want monitored for other reasons. By default, remote or network drives are not automatically monitored.

  -  *Fast scan this directory*
- open the directory in your file manager (*Open Directory…*)
- open the directory in a terminal window (*Open in Terminal…*)
- inspect the *Properties…* or the parent *Directory Properties…*

4.2.5 Data type

Level	Context menu	Types of layers  Raster (*Reference*)	 Vector (*Reference*)	 Point clouds (*Reference*)	 Mesh (*Reference*)
File	*Delete File <dataset name>*	☑	☑		
	Show in Files	☑	☑	☑	☑
	Manage	☑	☑	☑	☑
	File Properties…	☑	☑	☑	☑
	Open <file type> Externally…	☑			
Layer	*Add Layer to Project*	☑	☑	☑	☑
	Export Layer	☑	☑		
	Layer Properties…	☑	☑	☑	☑
	Open with Data Source Manager…	☑	☑		
	Execute SQL…		☑		

4.2.6 Database entries

Depending on your OS and installed drivers, you might have access to different database types to use in QGIS. Below are listed the different entries of contextual menu at each level of the dataset tree.

Level	Context menu	Type of database  GeoPackage ([1])	 SpatiaLite	 PostgreSQL	 SAP HANA	 MS SQL Server	 Oracle
Top menu	Create a *New Connection…* to an existing database	☑	☑	☑	☑	☑	☑
	Create Database…	☑	☑				
	Create Database and Layer…	☑					

continues on next page

Table 4.1 – continued from previous page

	Save Connections… details to a file			☑	☑	☑		
	Load Connections…			☑	☑	☑		
Connection / Database	*Refresh* a connection			☑	☑	☑	☑	
	Edit Connection… settings			☑	☑	☑	☑	
	Duplicate Connection			☑	☑	☑	☑	
	Remove Connection…	☑	☑	☑	☑	☑	☑	
	Import Vector Layer…	☑	☑	☑	☑	☑		
	New Field Domain ► New Range Domain	☑						
	New Field Domain ► New Coded Values Domain	☑						
	New Field Domain ► New Glob Domain	☑						
	Delete <database_n	☑	☑					
	Compact Database (VACUUM)	☑						
	Create a *New Schema…*			☑	☑	☑		
	Create a *New Table…*	☑	☑	☑	☑			
	Execute SQL… query	☑	☑	☑	☑			
Schema	*Refresh* a schema			☑	☑	☑		

continues on next page

Table 4.1 – continued from previous page

	Schema Operations ► Rename Schema…			☑	☑	☑	
	Schema Operations ► Delete Schema…			☑	☑	☑	
	Create a *New Table…*			☑	☑		
	Execute SQL… query			☑	☑		
Table / Layer	*Table Operations ► Rename Table…*			☑	☑	☑	
	Table Operations ► Truncate Table…			☑		☑	
	Execute SQL… query	☑	☑	☑			
	Export Layer ► To file…	☑	☑	☑	☑	☑	
	Manage ► Rename Layer <layer_name	☑	☑				
	Manage ► Delete Layer <layer_name	☑	☑	☑	☑	☑	
	Manage ► Delete Selected Layers	☑	☑	☑	☑	☑	
	Manage ► Add Layer to Project	☑	☑	☑	☑	☑	
	Manage ► Add Selected Layers to Project	☑	☑	☑	☑	☑	

continues on next page

Table 4.1 – continued from previous page

	Manage ► Move to Another Schema			☑		
	Open *Layer Properties…* dialog	☑	☑	☑	☑	☑
	Open *File Properties…* dialog	☑				
	Open with Data Source Manager	☑	☑			
Fields	*Add New Field…*	☑	☑	☑	☑	
Field	*Set Field Domain*	☑				
	Rename Field ([2])	☑		☑		
	Set Alias…	☑				
	Set Comment…	☑		☑		
	Delete Field…	☑	☑	☑	☑	

Note

When loading vector layers into QGIS, fields with *Field Domains* (such as those defined in a GeoPackage or ESRI File Geodatabase) are automatically detected. These domains are database-level constraints, meaning they are enforced by the database itself and apply across different applications, not just QGIS.

Importing vector layers into databases using Browser panel

QGIS allows you to import vector layers directly into a database from the *Browser* panel. This provides a simple way to transfer datasets into supported database formats without using DB Manager.

You can start the import in two ways:

- Drag and drop a single vector layer from the Browser panel or from the Project into a database connection or schema item.
- Right-click on a database connection or schema item and choose *Import Vector Layer…* from the context menu.

[1] The different entries may also be available for some GDAL supported vector file formats such as ESRI File Geodatabase, FlatGeobuf, GeoParquet, NetCDF, … when compatible.
[2] For PostgreSQL connections: only supported with ALTER TABLE privilege.

> **Note**
>
> When multiple layers are dragged into a database, they are imported immediately without further options. All imports run in the background, so the QGIS interface remains responsive during the process.

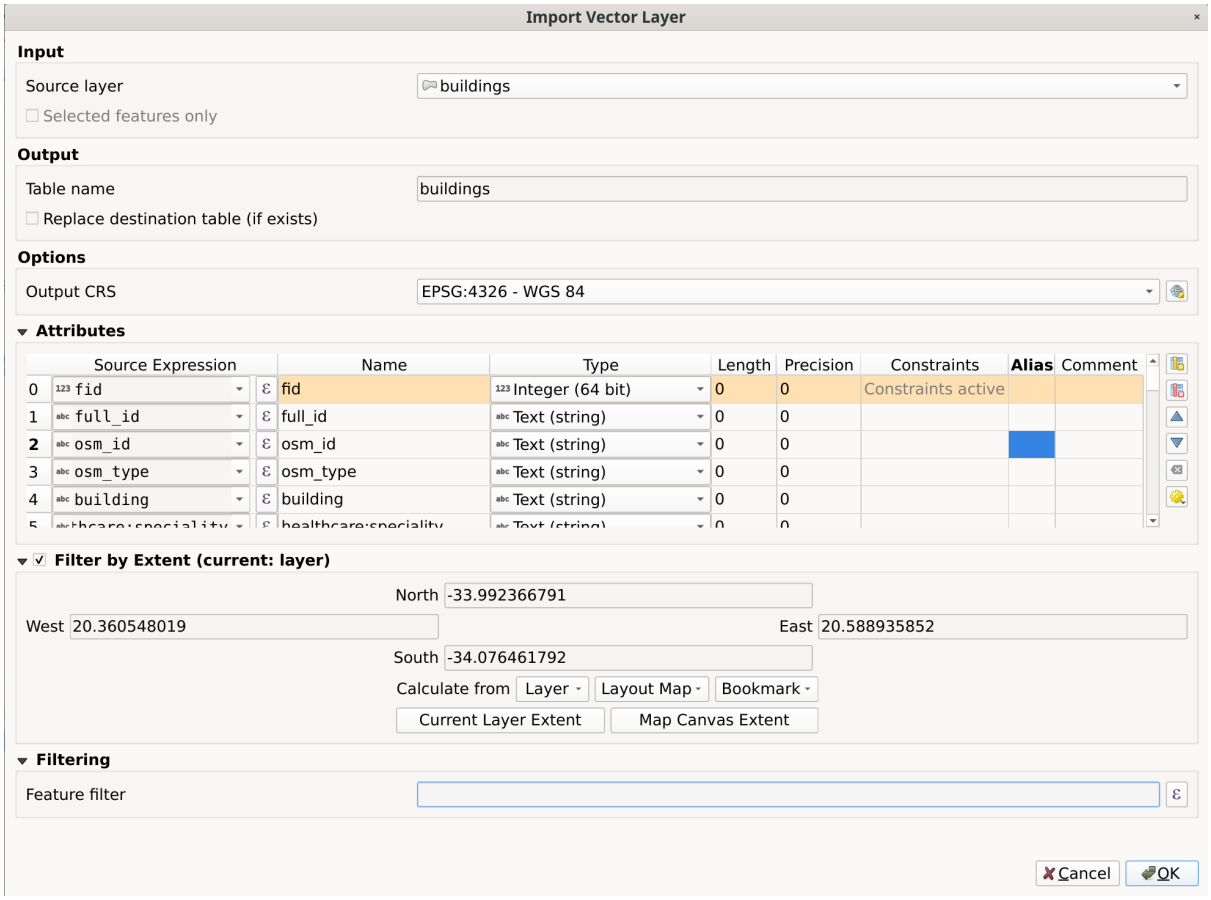


Fig. 4.3: Import Vector Layer dialog

To add a layer using *Import Vector Layer…* follow these steps:

1. Add *Source layer*: the layer to import, chosen from the current QGIS project. To import only selected features of the source layer use the *Selected features only* option.
2. Set the *Table name*: the name of the destination table in the database. By default, it is taken from the source layer name but can be modified. You can also use the *Replace destination table (if exists)* to overwrite an existing table.
3. Define the *Output CRS*: the coordinate reference system of the destination table. By default, it is the CRS of the source layer, but it can be changed using the *CRS selector*.
4. Configure the *Attributes*:
   - *Source expression*: expression used to populate the field. By default, this corresponds to the original source field.
   - *Name*: name of the field in the destination table.
   - *Type*: field data type (integer, text, real, etc.).
   - *Length* and *Precision*: length and precision of the field, where applicable.

- *Constraints*: rules to enforce on the field values (e.g. NOT NULL, UNIQUE, DEFAULT). Support depends on the destination database.
- *Alias*: an alternate display name for the field.
- *Comment*: a description or note attached to the field.

You can use buttons on the right side to:

-  Add new field to add a new field to the destination table.
-  Delete selected field to remove a field from the destination table.
-  Move selected field up or  Move selected field down to reorder fields.
-  Reset all fields to clear all fields and start from scratch.
-  Convert All Fields To Lowercase or  Convert All Fields To Uppercase, to change the case of all field names.

5. Optionally, apply a spatial filter with *Filter by Extent*. This restricts the import to features within a defined bounding box.
6. Optionally, apply an attribute filter with *Feature filter*. This allows you to use an expression to select which features are imported.

4.2.7 Tiles and Web Services

Level	Context menu	Type of services WMS / WMTS (*Reference*)	XYZ Tiles (*Reference*)	WCS (*Reference*)	WFS / OGC API - Features (*Reference*)	ArcGIS REST Servers (*Reference*)
Top menu	Create a *New Connection…*	☑	☑	☑	☑	☑
	Save Connections… details to a file	☑	☑	☑	☑	☑
	Load Connections…	☑	☑	☑	☑	☑
Connection	*Refresh* connection	☑	☑	☑	☑	☑
	Edit Connection… settings	☑	☑	☑	☑	☑
	Duplicate Connection	☑	☑	☑	☑	☑
	Remove Connection…	☑	☑	☑	☑	☑
	View Service Info in Web browser					☑
Table / Layer	*Export Layer ► To File…*	☑	☑	☑	☑	☑
	Add layer to Project	☑	☑	☑	☑	☑
	Open *Layer properties…* dialog	☑	☑	☑	☑	☑
	View Service Info in Web browser					☑

Level	Context menu	Type of services Vector Tiles (*Reference*)	Scene (*Reference*)	Cloud (*Reference*)	SensorThings (*Reference*)	STAC (*Reference*)
Top menu	Create a *New Connection…*		☑	☑	☑	☑
	Create a *New Generic Connection…*	☑				
	Create a *New ArcGIS Vector Tile Service Connection…*	☑				
	Save Connections… details to a file	☑	☑	☑	☑	☑
	Load Connections…	☑	☑	☑	☑	☑
Connection	*Edit Connection…* settings	☑	☑	☑	☑	☑
	Duplicate Connection	☑	☑	☑	☑	
	Delete Connection…	☑	☑	☑	☑	☑
	Refresh					☑
	Details…					☑
Table / Layer	*Export Layer ► To File…*				☑	
	Add layer to Project	☑	☑		☑	
	Open *Layer properties…* dialog	☑	☑		☑	
	Refresh					☑
	Download assets…					☑
	Details…					☑

4.2.8 Project and supported file types

File type	Context menu
QGIS Project file (.qgz, .qgs)	*Open Project*
	Extract Symbols… (opens Style Manager to export symbols to XML, add to default style, or export as PNG/SVG)
	File Properties…
QGIS Layer Definition (.qlr)	*Export Layer ► To File…*
	Add Layer to Project
	Layer Properties…
Processing Model (.model3)	*Run Model…*
	Edit Model…
Print Composer Template (.qpt)	*New Layout from Template*
Python script (.py)	*Run Script…*
	Open in External Editor

CHAPTER

FIVE

QGIS CONFIGURATION

QGIS is highly configurable. Through the *Settings* menu, it provides different tools to:

- *Style Manager…*: create and manage *symbols, styles and color ramps*.
- *Custom Projections…*: create your own *coordinate reference systems*.
- *Keyboard Shortcuts…*: define your own set of *keyboard shortcuts*. Also, they can be overridden during each QGIS session by the *project properties* (accessible under *Project* menu).
- *Interface Customization…*: configure the *application interface*, hiding dialogs or tools you may not need.
- *Options…*: set global *options* to apply in different areas of the software. These preferences are saved in the active *User profile* settings and applied by default whenever you open a new project with this profile.

5.1 Options

Some basic options for QGIS can be selected using the *Options* dialog. Select the menu option *Settings* ► *Options*. You can modify the options according to your needs. Some of the changes may require a restart of QGIS before they will be effective.

The tabs where you can customize your options are described below.

Note

Plugins can embed their settings within the Options dialog

While only Core settings are presented below, note that this list can be extended by *installed plugins* implementing their own options into the standard Options dialog. This avoids each plugin having their own config dialog with extra menu items just for them…

5.1.1 General Settings

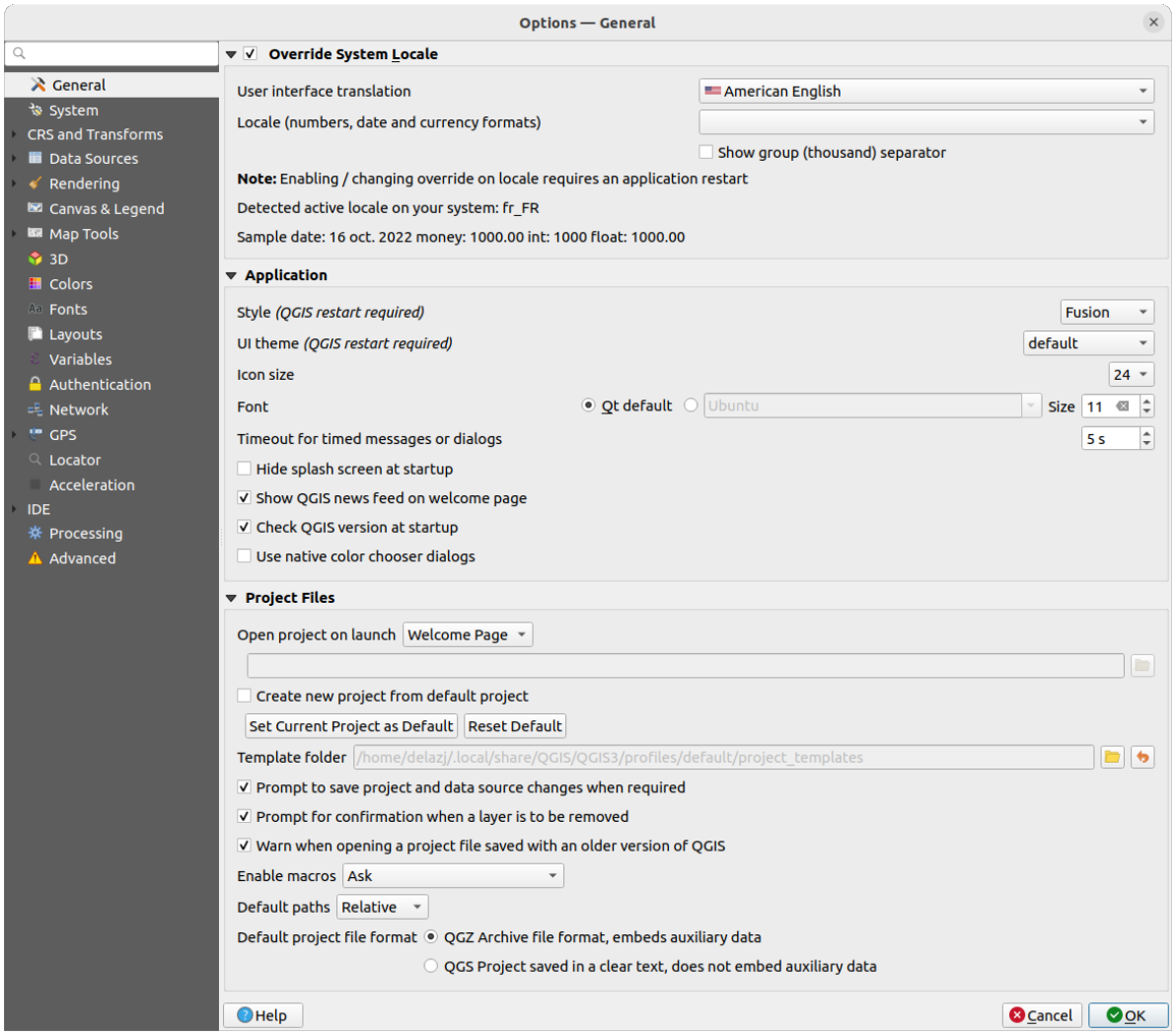


Fig. 5.1: General Settings

Override System Locale

By default, QGIS relies on your Operating System configuration to set language and manipulate numerical values. Enabling this group allows you to customize the behavior.

- Select from *User interface translation* the language to apply to the GUI
- Select in *Locale (number, date and currency formats)* the system on which date and numeric values should be input and rendered
- *Show group (thousand) separator*

A summary of the selected settings and how they would be interpreted is displayed at the bottom of the frame.

Application

- Select the *Style (QGIS restart required)* ie, the widgets look and placement in dialogs. Possible values depend on your Operating System.
- Define the *UI theme (QGIS restart required)* . It can be 'default', 'Night Mapping', or 'Blend of Gray'
- Define the *Icon size*

- Define the *Font* and its *Size*. The font can be *Qt default* or a user-defined one
- Change the *Timeout for timed messages or dialogs*
- *Hide splash screen at startup*
- *Show QGIS news feed on welcome page*: displays a curated QGIS news feed on the welcome page, giving you a direct way to be aware of project news (user/developer meetings date and summary, community surveys, releases announcements, various tips…)
- *Check QGIS version at startup* to keep you informed if a newer version is released
- *Use native color chooser dialogs* (see *Color Selector*)

Project files

- *Open project on launch*
  - 'Welcome Page' (default): can display the "News" feed, the project template(s) and the most recent projects (with thumbnails) of the *user profile*. No project is opened by default.
  - 'New': opens a new project, based on the default template
  - 'Most recent': reopens the last saved project
  - and 'Specific': opens a particular project. Use the … button to define the project to use by default.
- *Create new project from default project*. You have the possibility to press on *Set current project as default* or on *Reset default*. You can browse through your files and define a directory where you find your user-defined project templates. This will be added to *Project* ► *New From Template*. If you first activate *Create new project from default project* and then save a project in the project templates folder.
- *Prompt to save project and data source changes when required* to avoid losing changes you made.
- *Prompt for confirmation when a layer is to be removed*
- *Warn when opening a project file saved with an older version of QGIS*. You can always open projects created with older version of QGIS but once the project is saved, trying to open with older release may fail because of features not available in that version.
- *Enable project's embedded Python code* . This option handles execution of macros that are written to perform an action on project events, as well as custom Python functions to be used as expressions. You can choose between 'Never', 'Ask', 'For this session only' and 'Always (not recommended)'.
- *Default paths*: defines whether paths to files and layers used in new projects are stored as 'Absolute' or 'Relative' to the project file. This setting can be overwritten at the project level.
- *Default project file format*
  - *QGZ Archive file format, embeds auxiliary data* (see *auxiliary data*)
  - *QGS Project saved in a clear text, does not embed auxiliary data*: the auxiliary data is stored in a separate `.qgd` file along with the project file.

5.1.2 System Settings

SVG paths

Add or Remove *Path(s) to search for Scalable Vector Graphic (SVG) symbols*. These SVG files are then available to symbolize or label the features or decorate your map composition.

Also read *Remote or embedded file selector* for different ways to refer to svg files in a QGIS path.

Plugin paths

Add or Remove *Path(s) to search for additional C++ plugin libraries*.

Documentation paths

Add or Remove *Documentation Path(s)* to use for QGIS help. By default, a link to the official online User Manual corresponding to the version being used is added (i.e., `https://docs.qgis.org/$qgis_short_version/$qgis_locale/docs/user_manual/`). You can however add other links and prioritize them from top to bottom: each time you click on a *Help* button in a dialog, the topmost link is checked and if no corresponding page is found, the next one is tried, and so on.

> **Note**
>
> Documentation is versioned and translated only for QGIS Long Term Releases (LTR), meaning that if you are running a regular release (eg, QGIS 3.0), the help button will by default open the next LTR manual page (ie. 3.4 LTR), which may contain description of features in newer releases (3.2 and 3.4). If no LTR documentation is available then the *testing* doc, with features from newer and development versions, is used.

Settings

It helps you *Reset user interface to default settings (restart required)* if you made any *customization*.

Environment

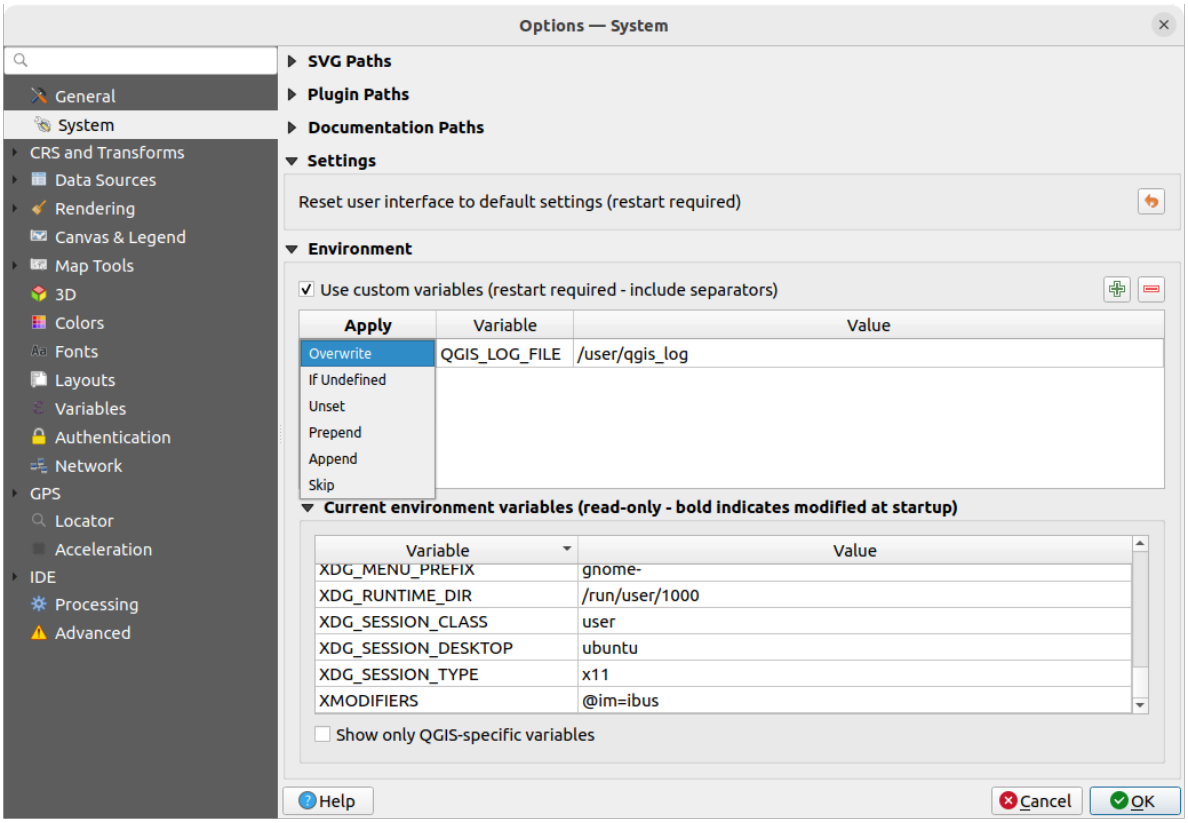


Fig. 5.2: System environment variables

System environment variables can be viewed, and many configured, in the **Environment** group. This is useful for platforms, such as Mac, where a GUI application does not necessarily inherit the user's shell environment. It's also useful for setting and viewing environment variables for the external tool sets controlled by the Processing toolbox (e.g., SAGA, GRASS), and for turning on debugging output for specific sections of the source code.

Check *Use custom variables (restart required - include separators)* and you can Add and Remove environment variables. For each new item, you can configure a *Variable* name, its *Value* and the *Apply* method to use, among which:

- *Overwrite*: replace any preexisting value of the variable
- *If undefined*: use this value for the variable if not already defined at a higher level (e.g., OS or application levels)
- *Unset*: remove the variable from the environment (the *Value* parameter is not used)
- *Prepend*: prepend the value to the preexisting value of the variable
- *Append*: append the value to the preexisting value of the variable
- *Skip*: the item is kept in the list for future reference but unused

Already defined environment variables are displayed in *Current environment variables*, and it's possible to filter them by activating *Show only QGIS-specific variables*.

5.1.3 User Profiles Settings

> **Note**
>
> For more information on how to manage user profiles, please read the dedicated section at *Working with User Profiles*.

5.1.4 CRS and Transforms Settings

> **Note**
>
> For more information on how QGIS handles layer projection, please read the dedicated section at *Working with Projections*.

CRS Handling

In the  *CRS Handling* tab you can configure which CRS will be used for a new project or layer.

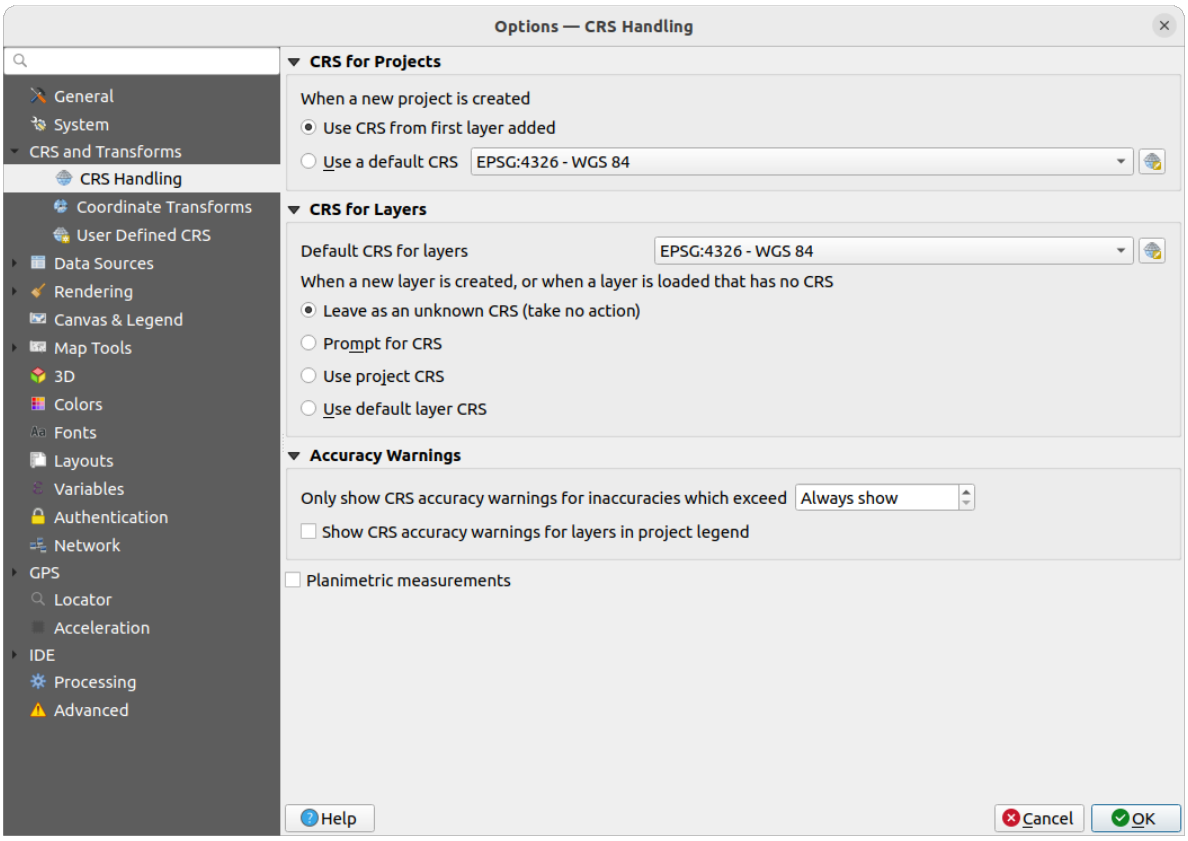


Fig. 5.3: CRS Settings

CRS for Projects

There is an option to automatically set new project's CRS:

-  *Use CRS from first layer added*: the CRS of the project will be set to the CRS of the first layer loaded into it

- *Use a default CRS*: a preselected CRS is applied by default to any new project and is left unchanged when adding layers to the project.

The choice will be saved for use in subsequent QGIS sessions. The Coordinate Reference System of the project can still be overridden from the *Project* ► *Properties…* ► *CRS* tab.

CRS for Layers

Default CRS for layers: select a default CRS to use when you create a layer

You can also define the action to take when a new layer is created, or when a layer without a CRS is loaded.

- *Leave as unknown CRS (take no action)*
- *Prompt for CRS*
- *Use project CRS*
- *Use default layer CRS*

Accuracy Warnings

Only show CRS accuracy warnings for inaccuracies which exceed a given distance: occurs when you are explicitly creating or modifying a dataset and select a CRS based on a datum ensemble with lower accuracy. The default is to `Always show` the warning if any inaccuracy. Requires a QGIS version using at least PROJ 8.0.

Show CRS accuracy warning for layers in project legend: If checked, any layer with a CRS with accuracy issues (i.e., a dynamic crs with no coordinate epoch available, or a CRS based on a datum ensemble with inherent inaccuracy exceeding the user-set limit) will have the warning icon in the *Layers* panel reflecting that it is a low-accuracy layer.

This is designed for use in engineering, BIM, asset management, and other fields where inaccuracies of meter/submeter level are potentially very dangerous or expensive!

Planimetric measurements: sets the default for the *planimetric measurements* property for newly created projects.

Coordinate Transforms

The *Coordinate Transforms* tab helps you set coordinate transformations and operations to apply when loading a layer to a project or reprojecting a layer.

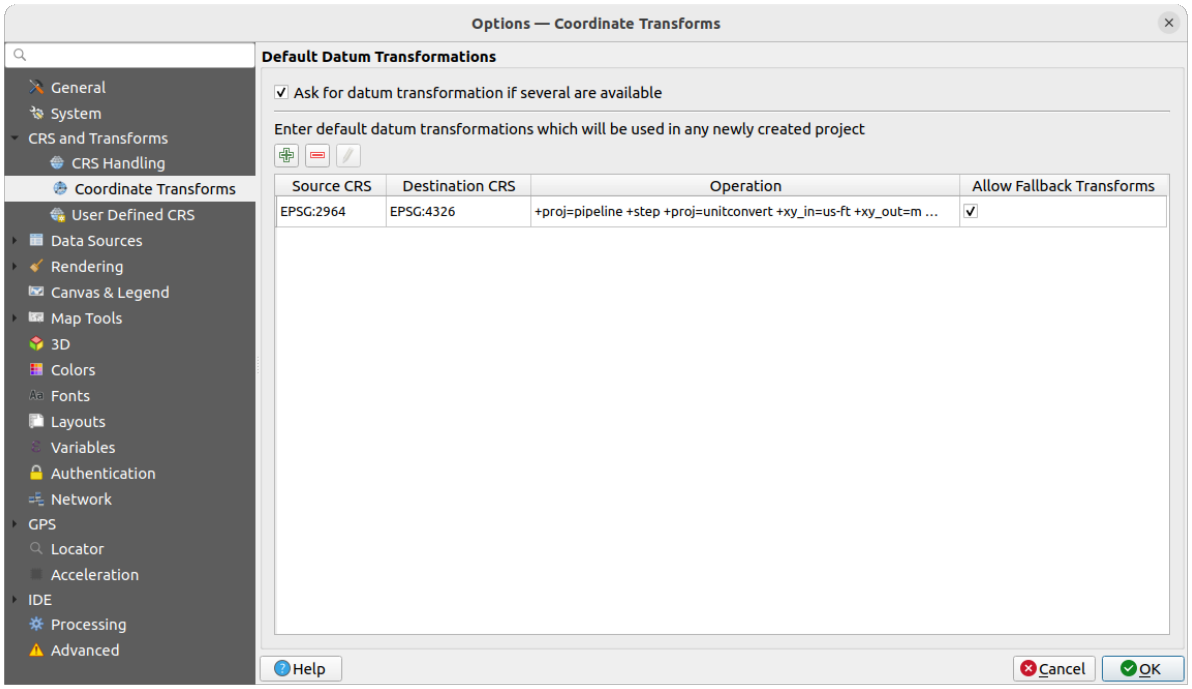


Fig. 5.4: Transformations settings

Default Datum Transformations

Here you can control whether reprojecting layers to another CRS should be:

- automatically processed using QGIS default transformations settings;
- and/or more controlled by you with custom preferences such as:
  - *Ask for datum transformation if several are available*
  - a predefined list of datum transformations to apply by default. See *Datum Transformations* for more details.

You can Add, Remove or Edit transformations, which will be used in any newly created project.

User Defined CRS

The *User Defined CRS* tab helps you to define a custom CRS which must conform to a WKT or Proj string format.

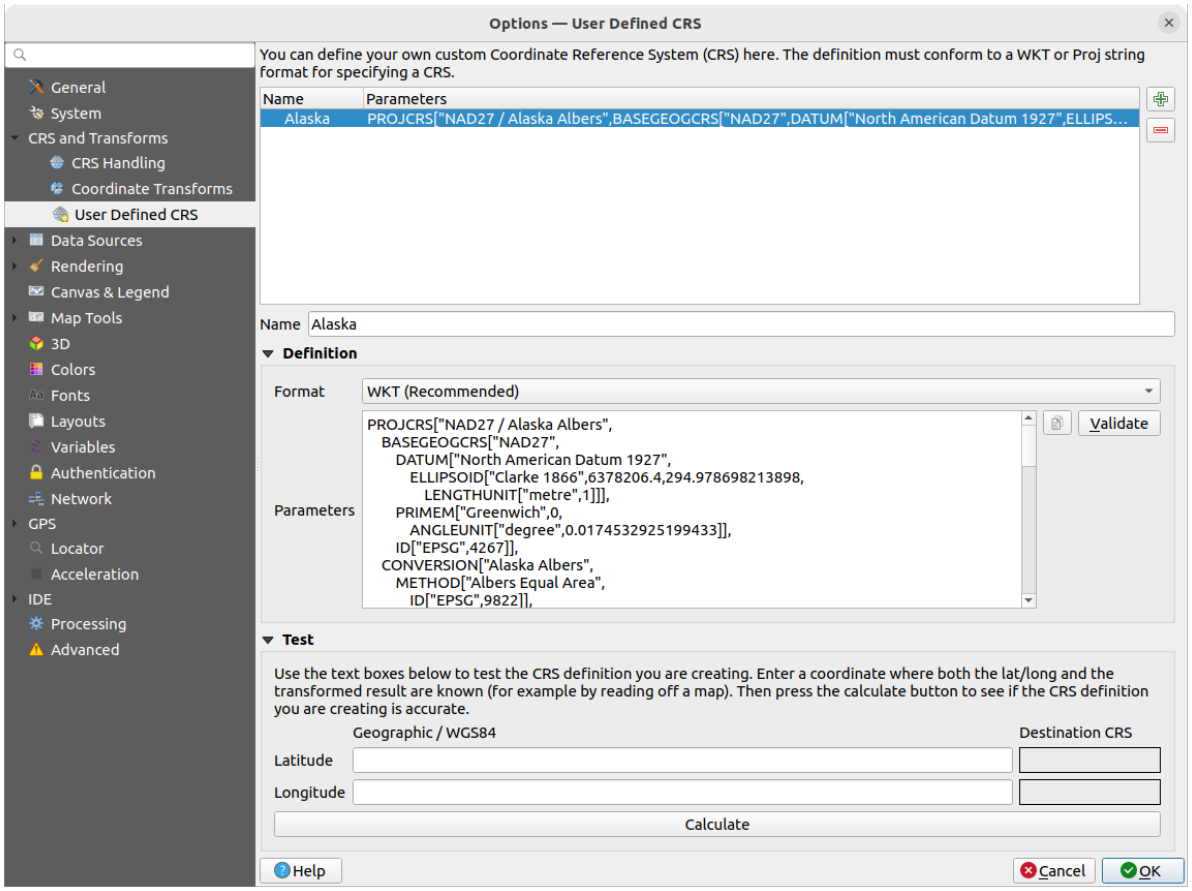


Fig. 5.5: User Defined CRS

Set a *Name* and use Add new CRS. If you want to delete an existing one you can use Remove CRS.

Definition

- ***Format***
  - WKT (Recommended)
  - Proj String (Legacy - Not Recommended)
- ***Parameters***
  - Copy parameters from an existing CRS.
  - *Validate* tests if your expression is correct.

Test

Here you can test your created CRS definition by Latitude and Longitude. Use a known coordinate to control if your definition is accurate.

5.1.5 Data Sources settings

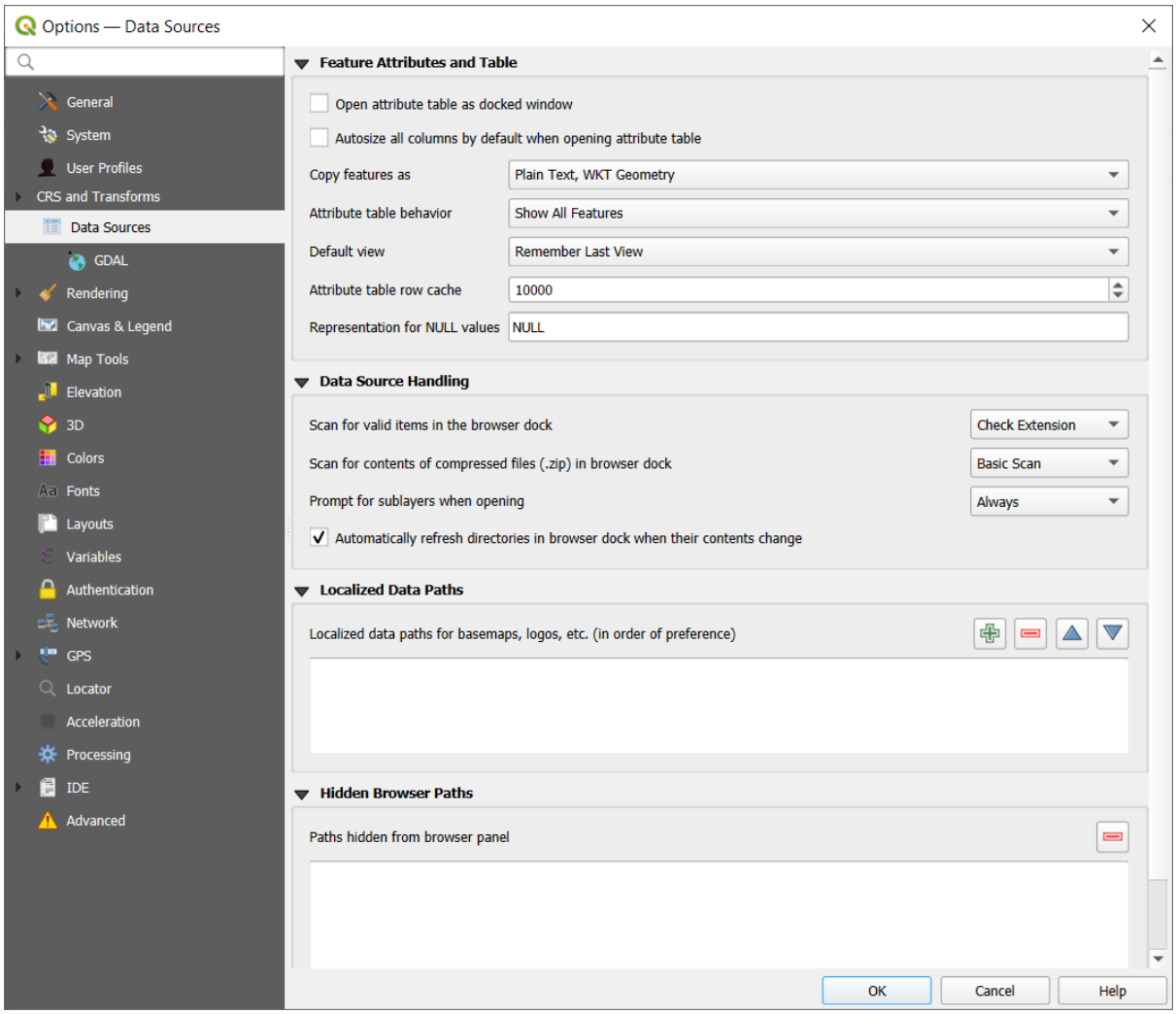


Fig. 5.6: Data Sources settings

Feature attributes and table

- *Open attribute table as docked window*
- *Autosize all columns by default when opening attribute table*
- *Copy features as*: defines the format to use for data when pasting features in other applications.
  - *Plain text, no geometry*: attributes are copied as text but the geometry is skipped
  - *Plain text, WKT geometry*: attributes are copied as text and the geometry is returned in WKT
  - *Plain text, WKB geometry*: attributes are copied as text and the geometry is returned in WKB (hex format)
  - *GeoJSON*: attributes and geometry are formatted as GeoJSON data
- *Attribute table behavior* : set filter on the attribute table at the opening. There are three possibilities: 'Show all features', 'Show selected features' and 'Show features visible on map'.
- *Default view*: define the view mode of the attribute table at every opening. It can be 'Remember last view', 'Table view' or 'Form view'.
- *Attribute table row cache* 1,00 . This row cache makes it possible to save the last loaded N attribute rows so that working with the attribute table will be quicker. The cache will be deleted when closing the attribute table.

- *Representation for NULL values*. Here, you can define a value for data fields containing a NULL value.

> **Tip**
>
> **Improve opening of big data attribute table**
>
> When working with layers with big amount of records, opening the attribute table may be slow as the dialog request all the rows in the layer. Setting the *Attribute table behavior* to **Show features visible on map** will make QGIS request only the features in the current map canvas when opening the table, allowing a quick data loading.
>
> Note that data in this attribute table instance will be always tied to the canvas extent it was opened with, meaning that selecting **Show All Features** within such a table will not display new features. You can however update the set of displayed features by changing the canvas extent and selecting **Show Features Visible On Map** option in the attribute table.

Data source handling

- *Scan for valid items in the browser dock*  . You can choose between 'Check extension' and 'Check file contents'.
- *Scan for contents of compressed files (.zip) in browser dock*  defines how detailed is the widget information at the bottom of the Browser panel when querying such files. 'No', 'Basic scan' and 'Full scan' are possible options.
- *Prompt for sublayers when opening*. Some rasters support sublayers — they are called subdatasets in GDAL. An example is netCDF files — if there are many netCDF variables, GDAL sees every variable as a subdataset. The option allows you to control how to deal with sublayers when a file with sublayers is opened. You have the following choices:
  - 'Always': Always ask (if there are existing sublayers)
  - 'If needed': Ask if layer has no bands, but has sublayers
  - 'Never': Never prompt, will not load anything
  - 'Load all': Never prompt, but load all sublayers
-  *Automatically refresh directories in browser dock when their contents change*: Allows you to manually opt-out of monitoring directories in the *Browser* panel by default (eg, to avoid potential slow down due to network latency).

Localized data paths

It is possible to use localized paths for any kind of file based data source. They are a list of paths which are used to abstract the data source location. For instance, if `C:\my_maps` is listed in the localized paths, a layer having `C:\my_maps\my_country\ortho.tif` as data source will be saved in the project using `localized:my_country\ortho.tif`.

The paths are listed by order of preference, in other words QGIS will first look for the file in the first path, then in the second one, etc.

Hidden browser paths

This widget lists all the folders you chose to hide from the *Browser panel*. Removing a folder from the list will make it available in the *Browser* panel.

GDAL Settings

GDAL is a data exchange library for geospatial data that supports a large number of vector and raster formats. It provides drivers to read and (often) write data in these formats. The *GDAL* tab exposes the drivers for raster and vector formats with their capabilities.

GDAL raster and vector drivers

The *Raster Drivers* and *Vector Drivers* tabs allow you to define which GDAL driver is enabled to read and/or write files, as in some cases more than one GDAL driver is available.

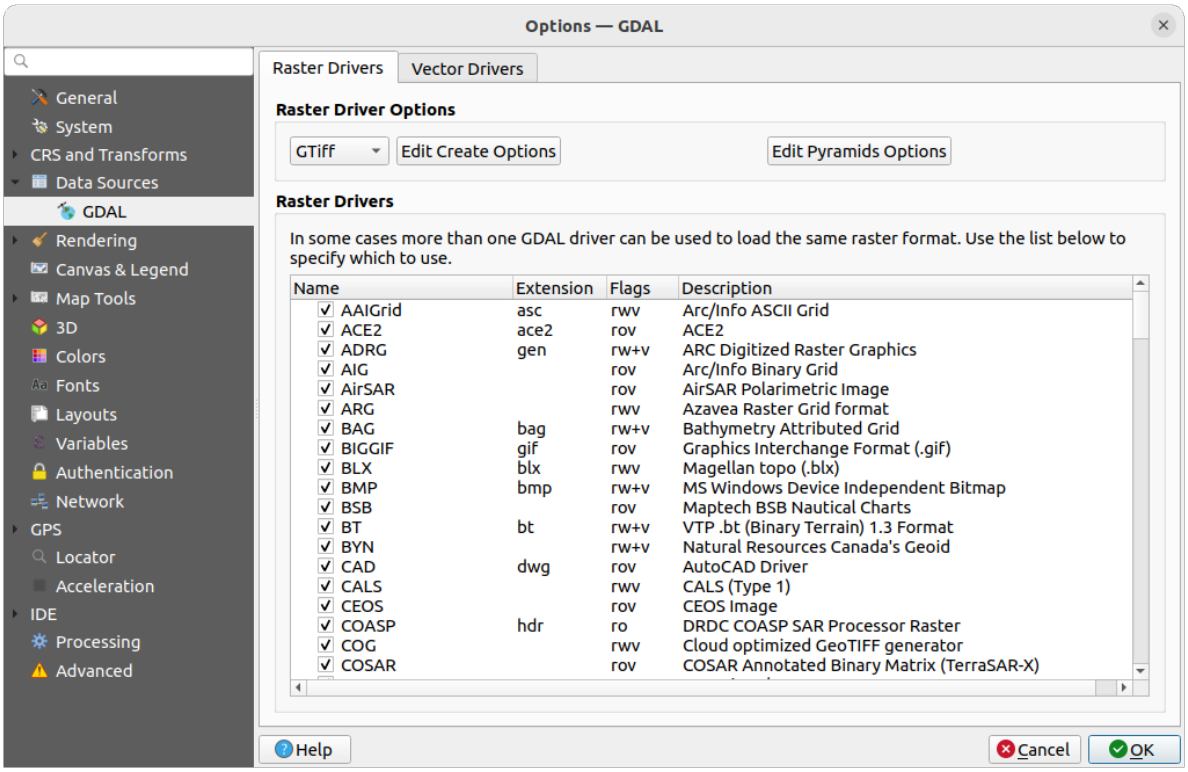


Fig. 5.7: GDAL Settings - Raster drivers

> **Tip**
>
> Double-click a raster driver that allows read and write access (`rw+(v)`) opens the *Edit Create options* dialog for customization.

Raster driver options

This frame provides ways to customize the behavior of raster drivers that support read and write access:

- *Edit create options*: allows you to edit or add different profiles of file transformation, i.e., a set of predefined combinations of parameters (type and level of compression, BIGTIFF support, blocks size, overview, colorimetry, alpha…) to use when outputting raster files. The parameters depend on the driver.

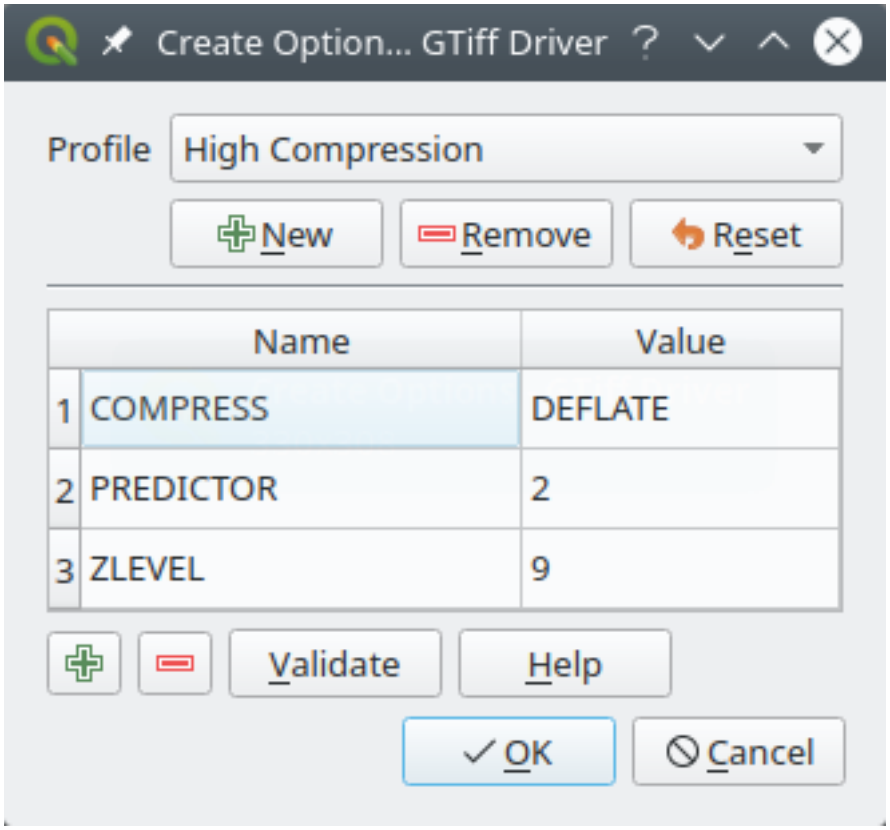


Fig. 5.8: Sample of create options profile (for GeoTiff)

The upper part of the dialog lists the current profile(s) and allows you to add new ones or remove any of them. You can also reset the profile to its default parameters if you have changed them. Some drivers (eg, GeoTiff) have some sample of profiles you can work with.

At the bottom of the dialog:

- The button lets you add rows to fill with the parameter name and value
- The button deletes the selected parameter
- Click the *Validate* button to check that the creation options entered for the given format are valid
- Use the *Help* button to find the parameters to use, or refer to the GDAL raster drivers documentation.

- *Edit Pyramids Options*

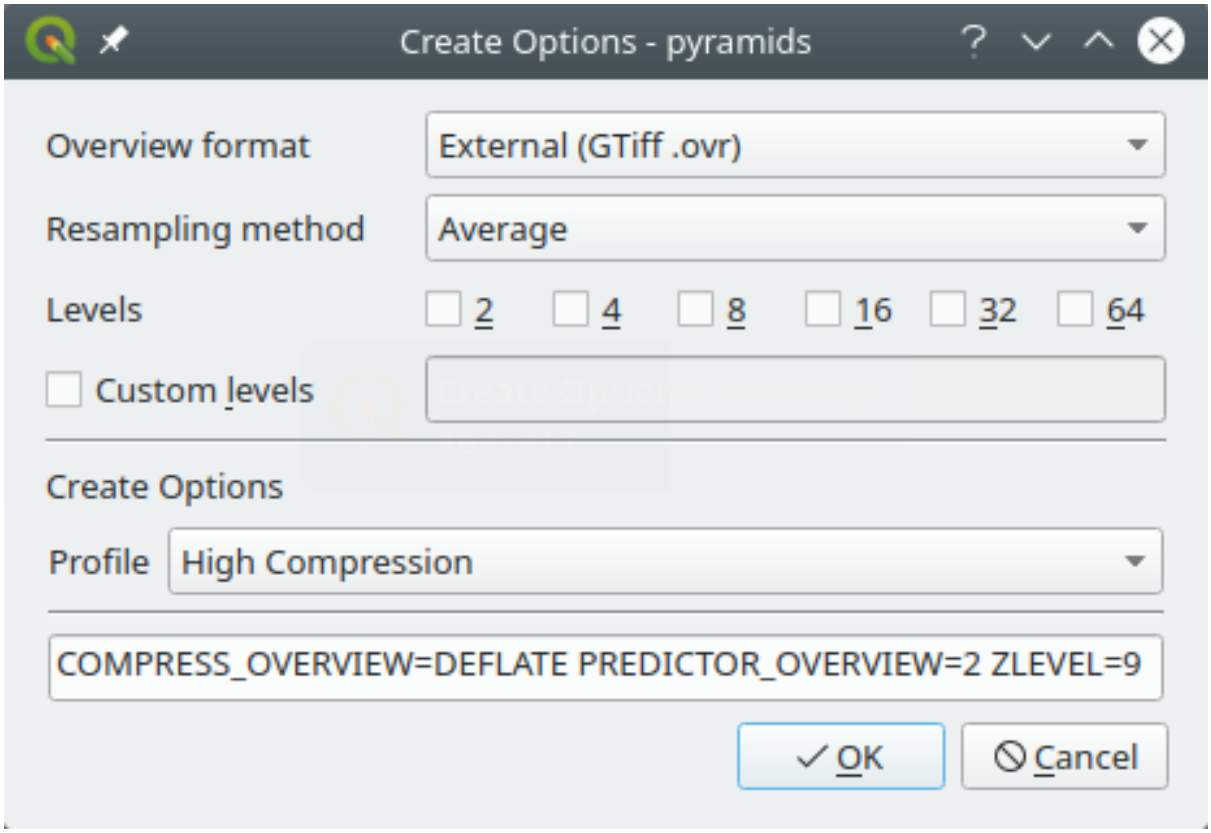


Fig. 5.9: Sample of pyramids profile

5.1.6 Rendering Settings

The  *Rendering* tab provides settings for controlling layers rendering in the map canvas.

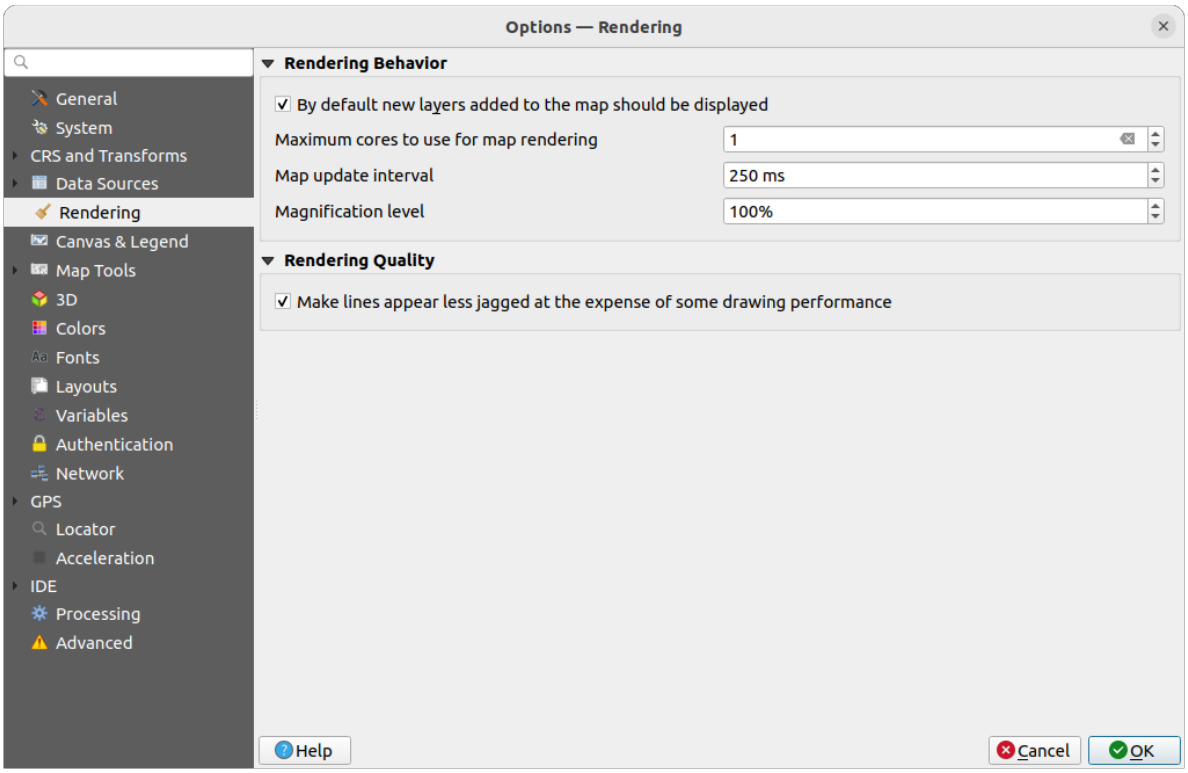


Fig. 5.10: Rendering settings

Rendering Behavior

- *By default new layers added to the map should be displayed*: unchecking this option can be handy when loading multiple layers to avoid each new layer being rendered in the canvas and slow down the process
- Set the *Maximum cores to use for map rendering*
- The map canvas renders in the background onto a separate image and at each *Map update interval* (defaults to 250 ms), the content from this (off-screen) image will be taken to update the visible screen representation. However, if rendering finishes faster than this duration, it will be shown instantaneously.
- *Magnification level* (see the *magnifier*)

Rendering Quality

- *Make lines appear less jagged at the expense of some drawing performance*

Vector rendering settings

The *Vector* tab contains specific settings for rendering vector layers.

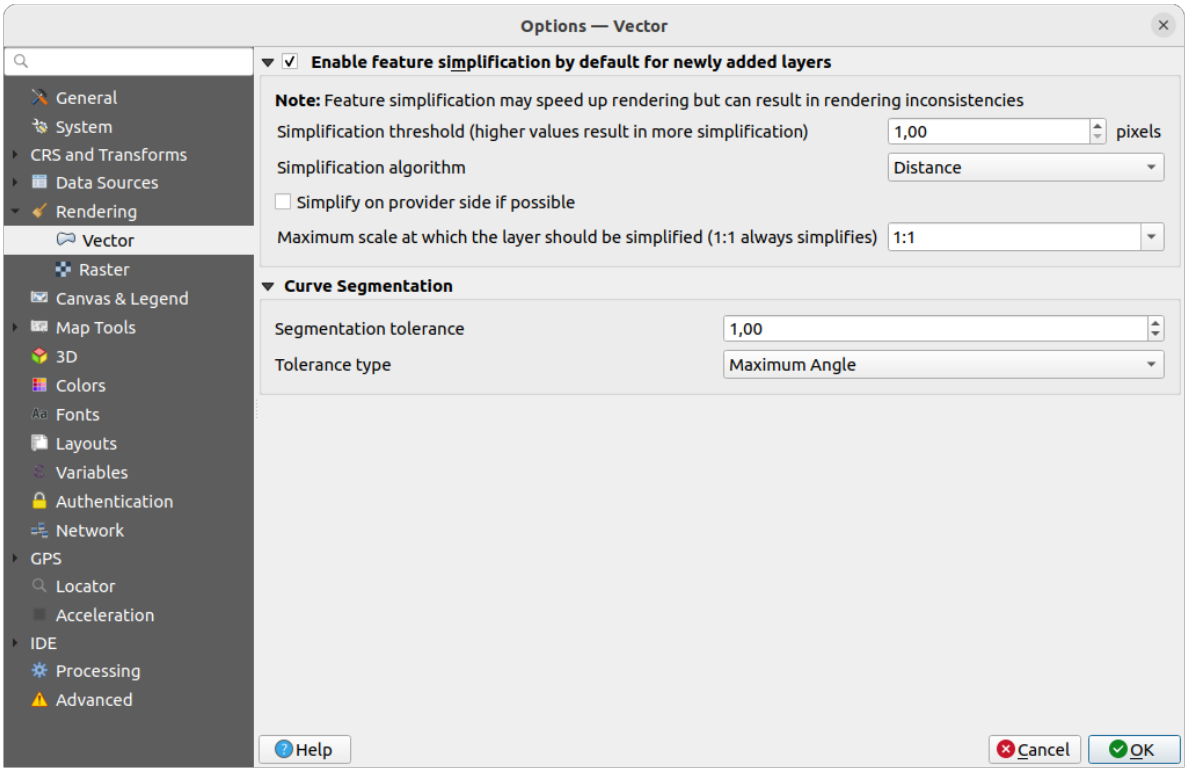


Fig. 5.11: Vector rendering settings

- *Enable Feature Simplification by Default for Newly Added Layers*: you simplify features' geometry (fewer nodes) and as a result, they display more quickly. Be aware that this can cause rendering inconsistencies. Available settings are:
  - *Simplification threshold (higher values result in more simplification)*
  - *Simplification algorithm*: This option performs a local "on-the-fly" simplification on feature's and speeds up geometry rendering. It doesn't change the geometry fetched from the data providers. This is important when you have expressions that use the feature geometry (e.g., calculation of area) - it ensures that these calculations are done on the original geometry, not on the simplified one. For this purpose, QGIS provides three algorithms: 'Distance' (default), 'SnapToGrid' and 'Visvalingam'.
  - *Simplify on provider side if possible*: the geometries are simplified by the provider (PostgreSQL, Oracle…) and unlike the local-side simplification, geometry-based calculations may be affected
  - *Maximum scale at which the layer should be simplified (1:1 always simplifies)*

> **Note**
>
> Besides the global setting, feature simplification can be set for any specific layer from its *Layer properties* ► *Rendering* menu.

- *Curve Segmentation*
  - *Segmentation tolerance*: this setting controls the way circular arcs are rendered. **The smaller** maximum angle (between the two consecutive vertices and the curve center, in degrees) or maximum difference

(distance between the segment of the two vertices and the curve line, in map units), the **more straight line** segments will be used during rendering.

- *Tolerance type*: it can be *Maximum angle* or *Maximum difference* between approximation and curve.

Raster rendering settings

The *Raster* tab contains specific settings for rendering raster layers.

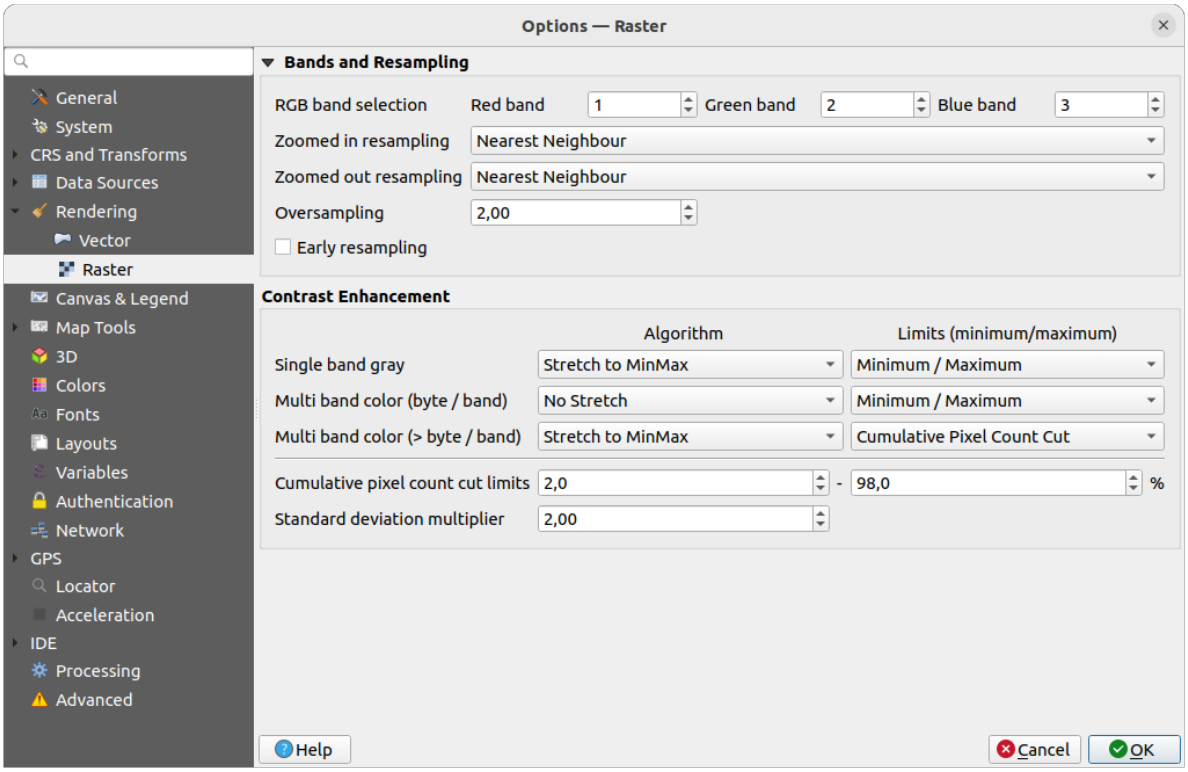


Fig. 5.12: Raster rendering settings

Under *Bands and Resampling*:

- With *RGB band selection*, you can define the number for the Red, Green and Blue band.
- The *Zoomed in resampling* and the *Zoomed out resampling* methods can be defined. For *Zoomed in resampling* you can choose between three resampling methods: 'Nearest neighbour', 'Bilinear (2x2 kernel)' and 'Cubic (4x4 kernel)'. For *Zoomed out resampling* you can choose between 'Nearest Neighbour' and 'Average'. You can also set the *Oversampling* value (between 0.0 and 99.99 - a large value means more work for QGIS - the default value is 2.0).
- *Early resampling*: allows to calculate the raster rendering at the provider level where the resolution of the source is known, and ensures a better zoom in rendering with QGIS custom styling. Really convenient for tile rasters loaded using an *interpretation method*. The option can also be set at the layer level (*Symbology* properties)

Contrast Enhancement options can be applied to *Single band gray*, *Multi band color (byte/band)* or *Multi band color (>byte/band)*. For each, you can set:

- the *Algorithm* to use, whose values can be 'No stretch', 'Stretch to MinMax', 'Stretch and Clip to MinMax' or 'Clip to MinMax'
- the *Limits (minimum/maximum)* to apply, with values such as 'Cumulative pixel count cut', 'Minimum/Maximum', 'Mean +/- standard deviation'.

The *Contrast Enhancement* options also include:

- *Cumulative pixel count cut limits*
- *Standard deviation multiplier*

5.1.7 Canvas and Legend Settings

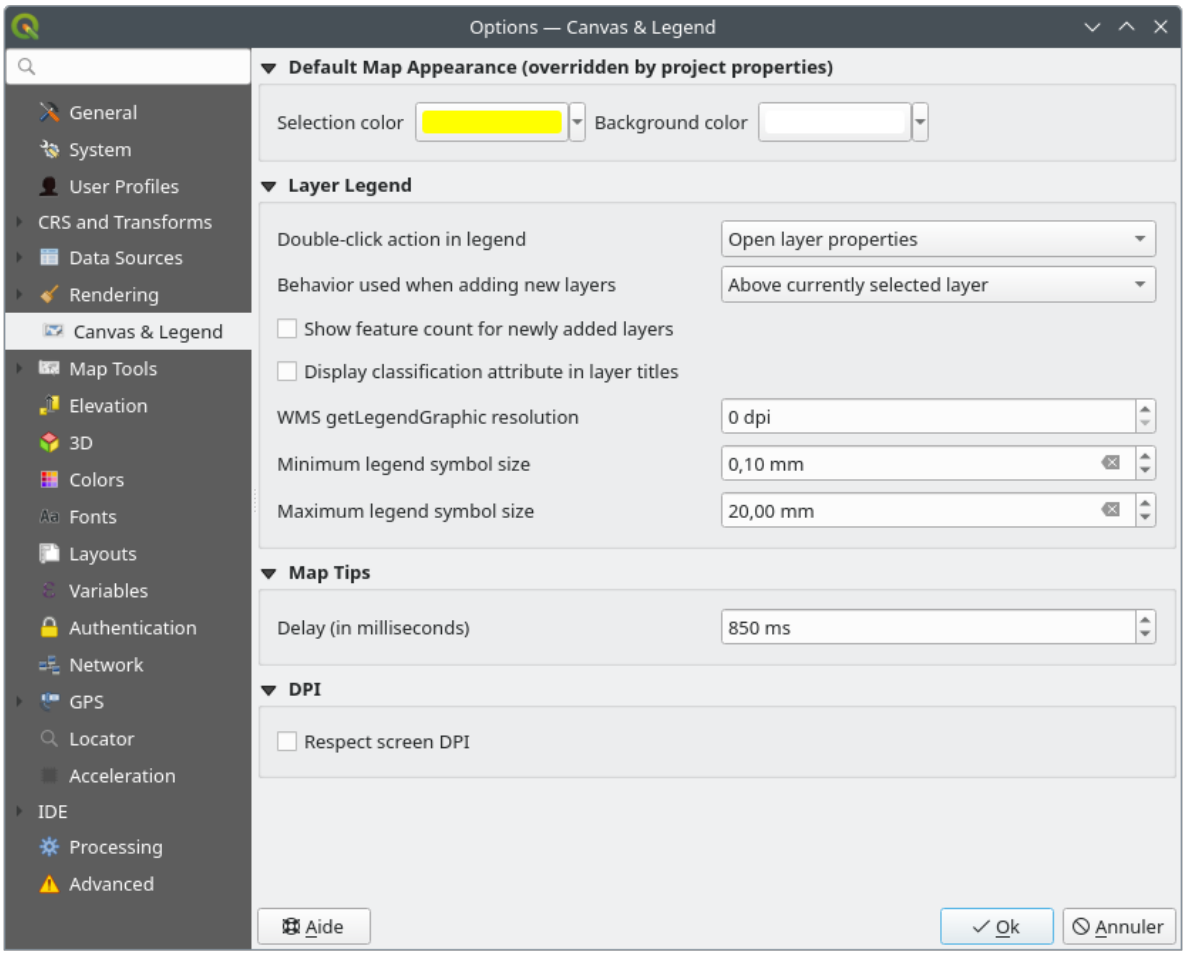


Fig. 5.13: Canvas & Legend settings

These properties let you set:

- the **Default map appearance (overridden by project properties)**: the *Selection color* and *Background color*.
- **Layer legend** interaction:
  - *Double click action in legend*: whether a double-click on a layer should either *Open layer properties* (default), *Open attribute table* or *Open layer styling dock*.
  - *Behavior used when adding new layers*: determines where layers are placed in the *Layers* panel when loaded into QGIS. It can be:
    - *Above currently selected layer*
    - *Always on top of the layer tree*
    - *Optimal index within current layer tree group*: Unlike the other options that sort the new layers among them and place them as a stack at the desired location, this option extents the sorting logic to the whole layer tree (or the active group) and inserts new layers in an "optimal" fashion by insuring that

point layers sit on top of point layers, followed by line layers on top of lines, followed by polygon layers, etc.

  - *Show feature count for newly added layers*: displays in the *Layers* panel the number of features next to the layer name. Feature count of classes, if any, is as well displayed. You can right-click on a layer to turn on/off its feature count.
  - *Display classification attribute names* in the Layers panel, e.g., when applying a categorized or rule-based renderer (see *Symbology Properties* for more information).
  - the *WMS getLegendGraphic Resolution*
  - *Minimum* and *Maximum legend symbol size* to control symbol size display in the *Layers* panel
- the *Delay* in milliseconds of layers *map tips* display
- Whether QGIS should *Respect screen DPI*: If enabled, QGIS will attempt to display the canvas with physically accurate scale on screen, depending on the monitor's physical DPI. Symbology with specified display size will also be rendered accurately, e.g., a 10mm symbol will show as 10mm on screen. However, label font sizes on canvas may differ from those in QGIS' UI or other applications. If this setting is turned off, QGIS will use the operating system's logical DPI, which will be consistent with other applications on the system. However, canvas scale and symbology size may be physically inaccurate on screen. In particular, on high-dpi screens, symbology is likely to appear too small.

  For best experience, it is recommended to enable *Respect screen DPI*, especially when using multiple or different monitors and preparing visually high-quality maps. Disabling *Respect screen DPI* will generate output that may be more suitable for mapping intended for on-screen use only, especially where font sizes should match other applications.

Note

Rendering in layouts is not affected by the *Respect screen DPI* setting; it always respects the specified DPI for the target output device. Also note that this setting uses the physical screen DPI as reported by the operating system, which may not be accurate for all displays.

5.1.8 Map tools Settings

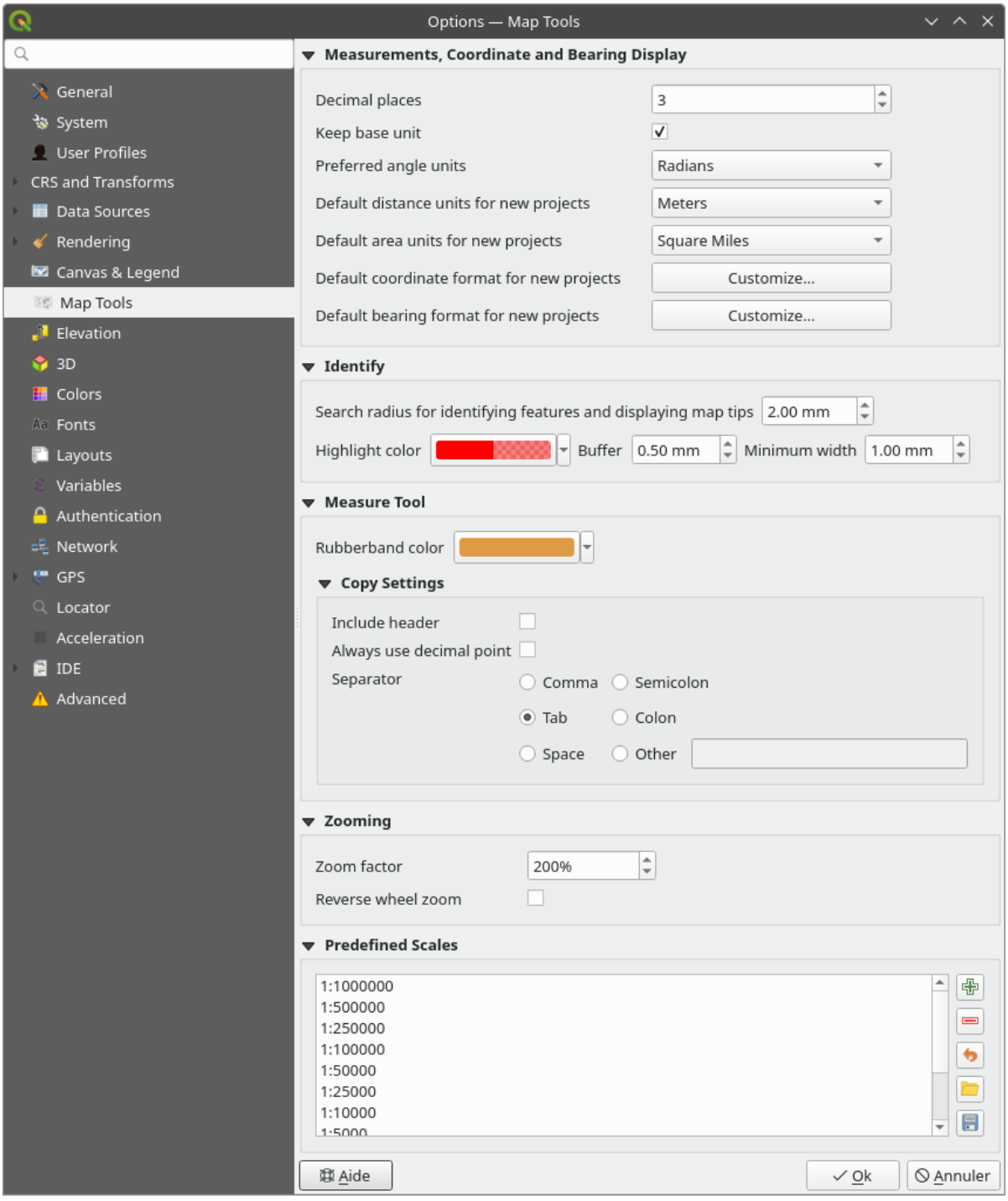


Fig. 5.14: Map tools settings

Measurements, Coordinate and Bearing Display

This section provides ways to configure default units parameters:

- Define *Decimal places*
- *Keep base unit* to not automatically convert large numbers (e.g., meters to kilometers)

- *Preferred angle units*: options are 'Degrees', 'Radians', 'Gon/gradians', 'Minutes of arc', 'Seconds of arc', 'Turns/revolutions', milliradians (SI definition) or mil (NATO/military definition)
- *Default distance units for new projects*: options are 'Meters', 'Kilometers', 'Feet', 'Yards', 'Miles', 'Nautical Miles', 'Centimeters', 'Millimeters', 'Inches', 'Degrees' or 'Map Units' (automatically matches the project's CRS units)
- *Preferred area units for new projects*: options are 'Square meters', 'Square kilometers', 'Square feet', 'Square yards', 'Square miles', 'Hectares', 'Acres', 'Square nautical miles', 'Square centimeters', 'Square millimeters', 'Square inches', 'Square degrees' or 'Map Units' (automatically matches the project's CRS units)
- *Default coordinate format for new projects*, as displayed in the *Coordinates* box on QGIS status bar and in the *Derived* section of the Identify features tool's results
- *Default bearing format for new projects*, as displayed in the status bar for the map canvas panning direction and by the Measure bearing tool.

The global options can be overridden at the project level, *here* for measurement units and *here* for coordinate and bearing formatting.

Identify tool

This group offers some options regarding the behavior of the *Identify tool*.

- *Search radius for identifying features and displaying map tips* is a tolerance distance within which the identify tool will depict results as long as you click within this tolerance.
- *Highlight color* allows you to choose with which color features being identified should be highlighted.
- *Buffer* determines a buffer distance to be rendered from the outline of the identify highlight.
- *Minimum width* determines how thick should the outline of a highlighted object be.

Measure tool

- Define *Rubberband color* for measure tools

Measure Tool Copy Settings

These settings allow you to control the behavior of the *Copy* button in the Measure Line window. You can choose to *Include header* to keep column names, and you can also select *Separator* of your choice. You can also choose *Always use decimal point* to keep your coordinates copied to the clipboard with a dot as a decimal separator, even if your language settings in QGIS options are set up to use a comma. In this case, if you don't have the *Always use decimal point* option checked, you will be unable to select comma as a field separator.

Panning and zooming

- Define a *Zoom factor* for zoom tools or wheel mouse
- *Reverse wheel zoom* allows you to adjust mouse wheel scrolling direction.

Predefined scales

Here, you find a list of predefined scales to display by default in the scale-related drop-down widgets, such as the status bar *Scale*, the visibility scales selector or secondary 2D map view settings,… With the and buttons you can add or remove your personal scales. You can also import or export scales from/to a `.XML` file. Note that you still have the possibility to remove your changes and reset to the predefined list.

From the project properties dialog, you can also set your own list of scales, overriding this global one in the widgets.

Digitizing settings

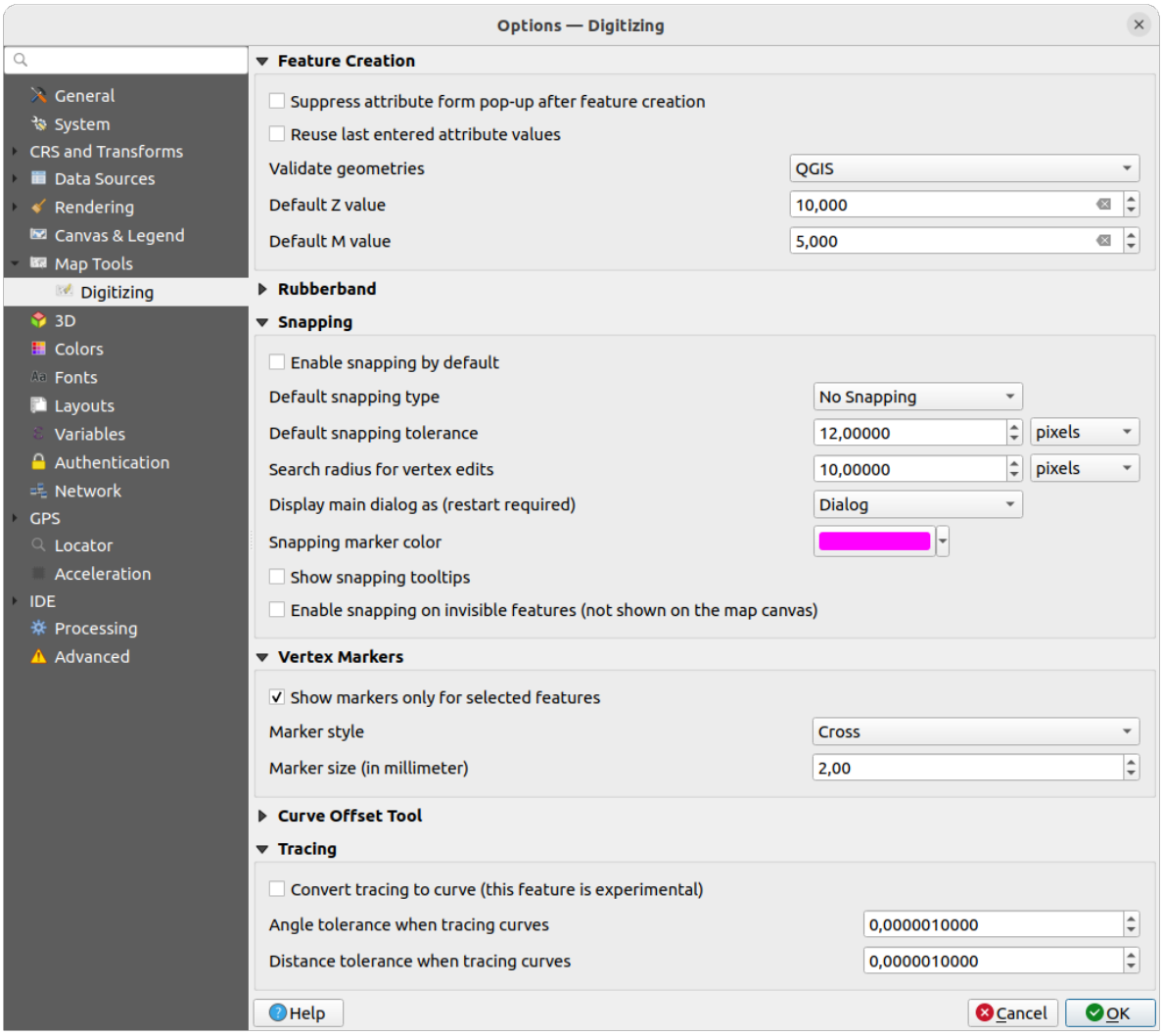


Fig. 5.15: Digitizing settings

This tab helps you configure general settings when *editing vector layer* (attributes and geometry).

Feature creation

- *Suppress attribute form pop-up after feature creation*: this choice can be overridden in each layer properties dialog.
- *Reuse last entered attribute values*: remember the last used value of every attribute and use it as default for the next feature being digitized. Works per layer. This behavior can also be controlled on a per-field basis (see *Configure the field behavior*).
- *Validate geometries*. Editing complex lines and polygons with many nodes can result in very slow rendering. This is because the default validation procedures in QGIS can take a lot of time. To speed up rendering, it is possible to select GEOS geometry validation (starting from GEOS 3.3) or to switch it off. GEOS geometry validation is much faster, but the disadvantage is that only the first geometry problem will be reported.

  Note that depending on the selection, reports of geometry errors may differ (see *Types of error messages and their meanings*)
- *Default Z value* to use when creating new 3D features.

Rubberband

- Define Rubberband *Line width*, *Line color* and *Fill color*.
- *Don't update rubberband during vertex editing*.

Snapping

- *Enable snapping by default* activates snapping when a project is opened
- Define *Default snap mode* (‘Vertex’, ‘Segment’, ‘Centroid’, ‘Middle of segments’, ‘Line endpoints’, ‘Area’)
- Define *Default snapping tolerance* in map units or pixels
- Define the *Search radius for vertex edits* in map units or pixels
- *Display main dialog as (restart required)*: set whether the Advanced Snapping dialog should be shown as ‘Dialog’ or ‘Dock’.
- *Snapping marker color*
- *Show snapping tooltips* such as name of the layer whose feature you are about to snap. Helpful when multiple features overlap.
- *Enable snapping on invisible features (not shown on the map canvas)*

Vertex markers

- *Show markers only for selected features*
- Define vertex *Marker style* (‘Cross’ (default), ‘Semi transparent circle’ or ‘None’)
- Define vertex *Marker size (in millimeter)*

Curve offset tool

The next options refer to the Offset Curve tool in *Advanced digitizing*. Through the various settings, it is possible to influence the shape of the line offset.

- *Join style*: specifies whether ‘Round’, ‘Miter’ or ‘Bevel’ joins should be used when offsetting corners
- *Quadrant segments*: controls the number of line segments to use to approximate a quarter circle when creating rounded offsets
- *Miter limit*: Sets the maximum distance from the offset geometry to use when creating a mitered join as a factor of the offset distance (only applicable for miter join style)
- *End cap style*: controls how line endings are handled; they can be ‘Round’, ‘Flat’ or ‘Square’.

Tracing

By activating the *Convert tracing to curve* you can create curve segments while digitizing. Keep in mind that your data provider must support this feature.

5.1.9 Elevation settings

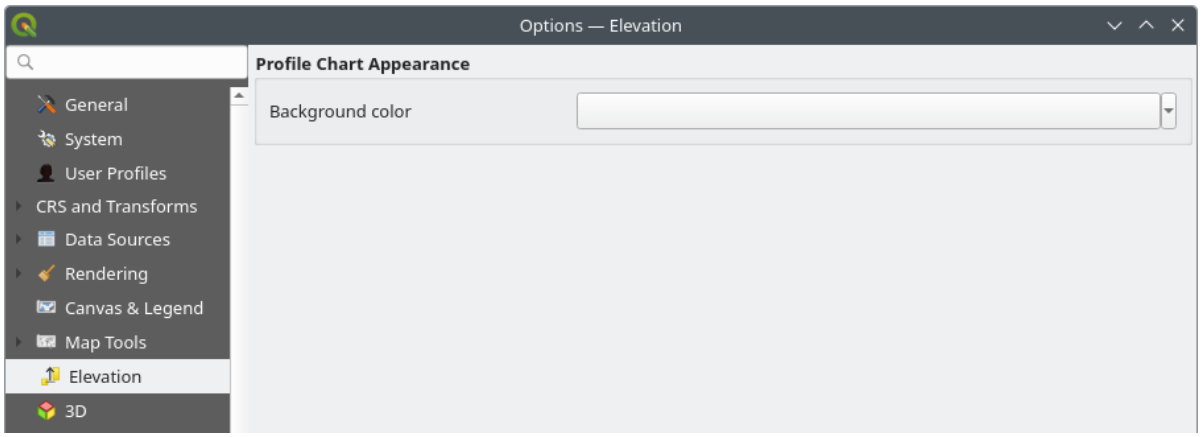


Fig. 5.16: Elevation settings

In *Elevation* menu, you can set a specific color to use as the *Background color* for *elevation profiles*. This can make the chart more readable for certain datasets, e.g., point clouds with RGB coloring, where the default background color is too similar to point colors to be easily discernable. If unchanged, the elevation profiles will continue to display using the standard system background color.

5.1.10 3D settings

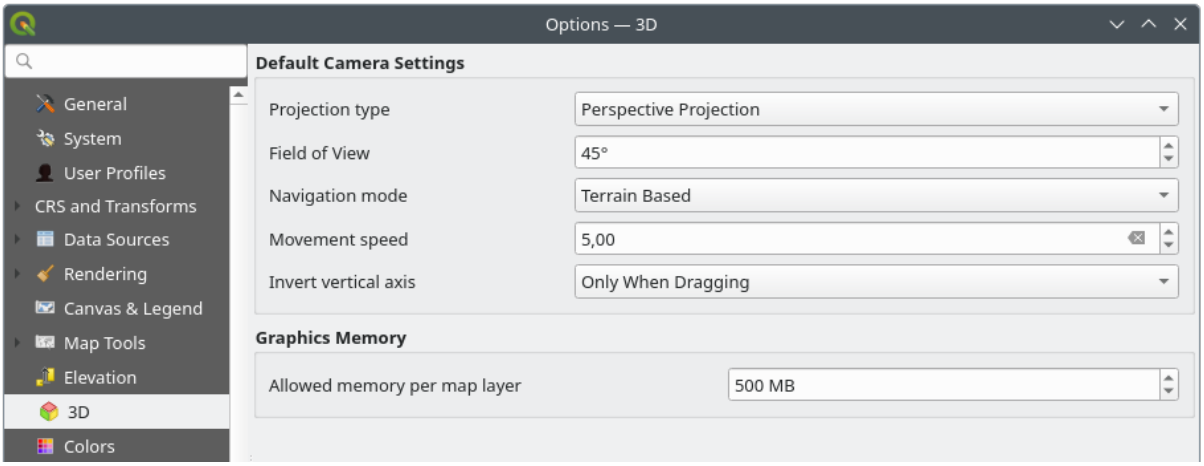


Fig. 5.17: 3D settings

The *3D* menu helps you configure some default settings to use for any *3D Map view*. These can refer to:

- *Default Camera Settings*:
  - *Projection type*: allowing to view the 3D scene in a:
    - *Perspective projection* (default): Parallel lines appear to meet in the distance. Objects appear to shrink the farther they are from the camera.
    - or an *Orthogonal projection*: Parallel lines appear parallel. Objects appear the same size regardless of distance.

  - Camera's *Field of view*: only relevant in perspective projection, specifies the current vertical field of view in degrees and determines how much of the scene is visible to the camera. Default value is 45°.
  - *Navigation mode*: provides different means to interact with the 3D scene. Available modes are:
    * *Terrain based*: the camera follows around a fixed position on the surface of the terrain as the scene is navigated.
    * *Walk mode (first person)*

    Depending on the selected mode, *navigation commands* differ.
  - *Movement speed*
  - *Invert vertical axis*: Controls whether vertical axis movements should be inverted from their normal behaviour. Only affects movement in the *Walk mode*. It can be set to:
    * *Never*
    * *Only when dragging*: causes the vertical motion to inverted only when performing a click-and-drag camera rotation
    * and *Always*: causes the motions to be inverted when both click-and-dragging and when the camera movement is locked to the cursor (via a ~ key press)

- Under *Graphics memory*, the *Allowed memory per layer* option lets you set the GPU memory limit configuration on each layer. This is useful for users utilizing large 3D scenes which exhaust the available GPU memory resources. When a limit is hit, a warning is also displayed, which should assist in troubleshooting large scenes.

5.1.11 Colors settings

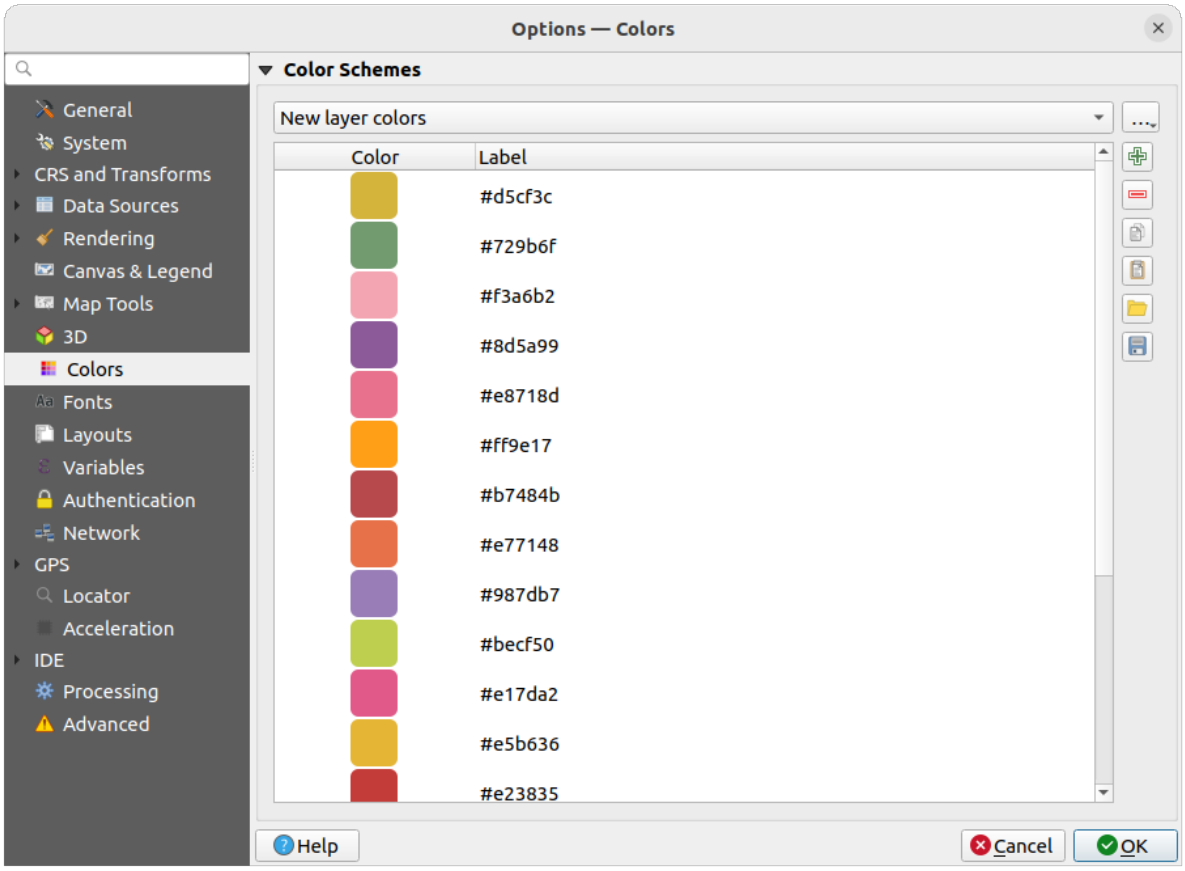


Fig. 5.18: Colors settings

This menu allows you to create or update palettes of colors used throughout the application in the *color selector widget*. You can choose from:

- *Recent colors* showing recently used colors
- *Standard colors*, the default palette of colors
- *Project colors*, a set of colors specific to the current project (see *Styles Properties* for more details)
- *New layer colors*, a set of colors to use by default when new layers are added to QGIS
- or custom palette(s) you can create or import using the … button next to the palette combobox.

By default, *Recent colors*, *Standard colors* and *Project colors* palettes can not be removed and are set to appear in the color button drop-down. Custom palettes can also be added to this widget thanks to the *Show in Color Buttons* option.

For any of the palettes, you can manage the list of colors using the set of tools next to the frame, ie:

- *Add* or *Remove* color
- *Copy* or *Paste* color
- *Import* or *Export* the set of colors from/to `.gpl` file.

Double-click a color in the list to tweak or replace it in the *Color Selector* dialog. You can also rename it by double-clicking in the *Label* column.

5.1.12 Fonts Settings

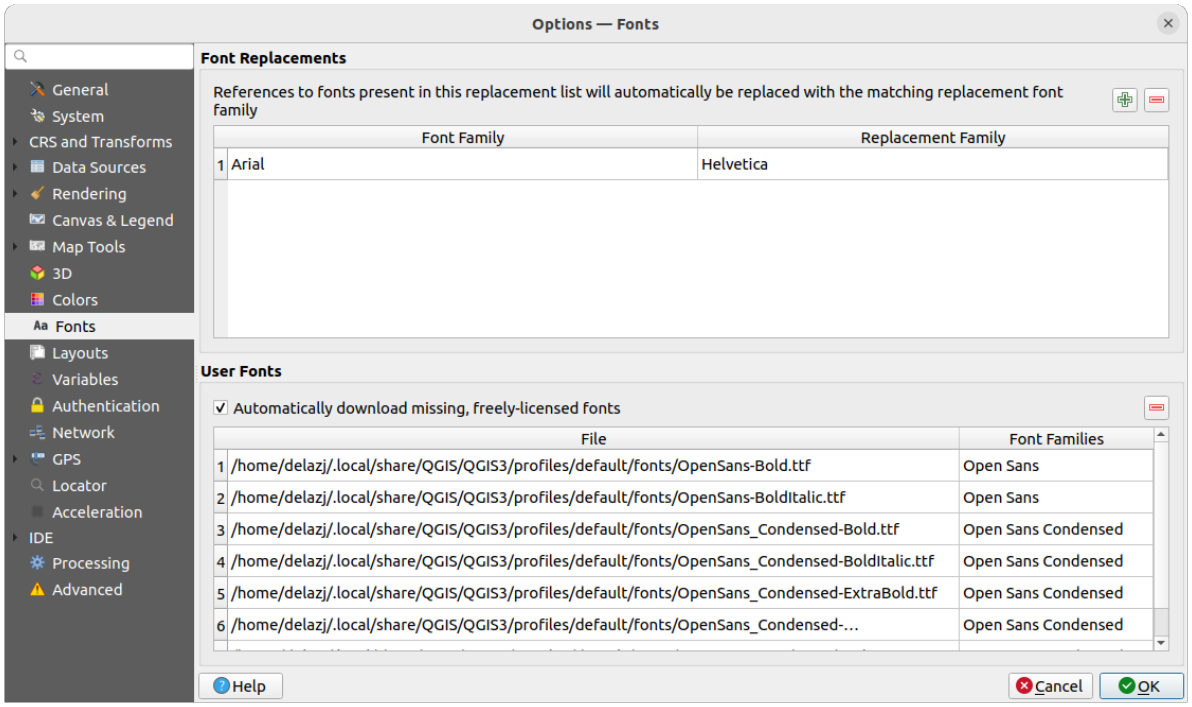


Fig. 5.19: Fonts settings

The *Fonts* tab provides support to manage fonts used across the projects:

- *Font Replacements*: gives ability to populate a list of automatic font replacements to apply when loading projects or styles, allowing better support for projects and styles to be used across different operating systems (e.g., replace “Arial” with “Helvetica”).

- *User Fonts*: Allows you to place TTF or OTF fonts in the `fonts` sub-folder of the *user profile*. These fonts can be automatically loaded at QGIS startup time. This provides a way to use fonts without requiring them to be installed on an operating system level, which is often blocked in enterprise environments. The panel lists all installed user fonts and allows you to manage (i.e., remove) previously installed user fonts.

  It is also possible to  *Automatically download missing, freely-licensed fonts*: E.g., if you open a project or style, or try to load a vector tile layer that references fonts that aren't currently available, then a hard-coded list of freely licensed fonts to download via URL is consulted to determine whether it's possible to automatically download the font to the user profile font directory (with notification of the font license).

5.1.13 Layouts settings

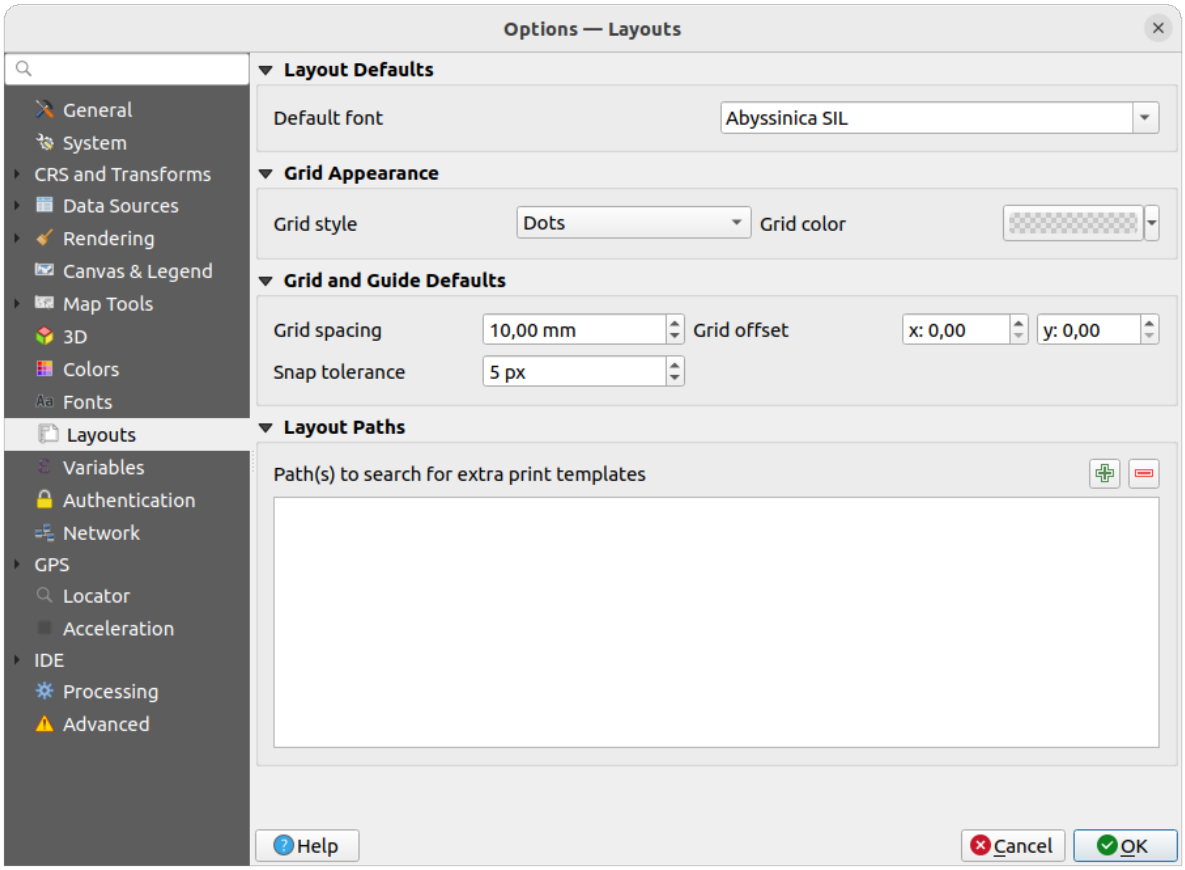


Fig. 5.20: Layouts settings

Composition defaults

You can define the *Default font* used within the *print layout*.

Grid appearance

- Define the *Grid style* ('Solid', 'Dots', 'Crosses')
- Define the *Grid color*

Grid and guide defaults

- Define the *Grid spacing*
- Define the *Grid offset* for X and Y
- Define the *Snap tolerance*

Layout Paths

- Define *Path(s) to search for extra print templates*: a list of folders with custom layout templates to use while creating new one.

5.1.14 Variables settings

The *Variables* tab lists all the variables available at the global-level.

It also allows the user to manage global-level variables. Click the button to add a new custom global-level variable. Likewise, select a custom global-level variable from the list and click the button to remove it.

More information about variables in the *Storing values in Variables* section.

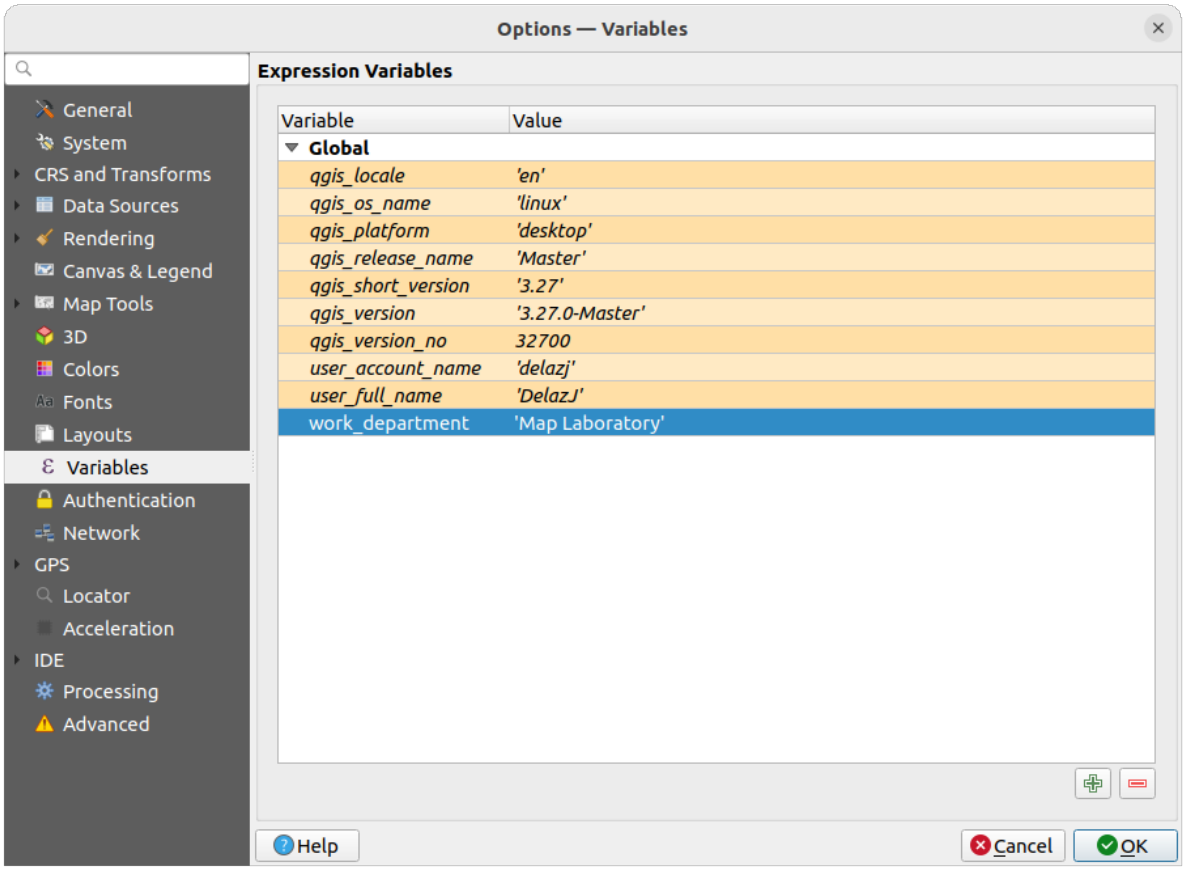


Fig. 5.21: Variables settings

5.1.15 Authentication settings

In the *Authentication* tab you can set authentication configurations and manage PKI certificates. See *Authentication System* for more details.

To manage authentications, you can use the list of tools next to the frame, ie:

- Add new authentication configuration
- Remove selected authentication configuration
- Edit selected authentication configuration

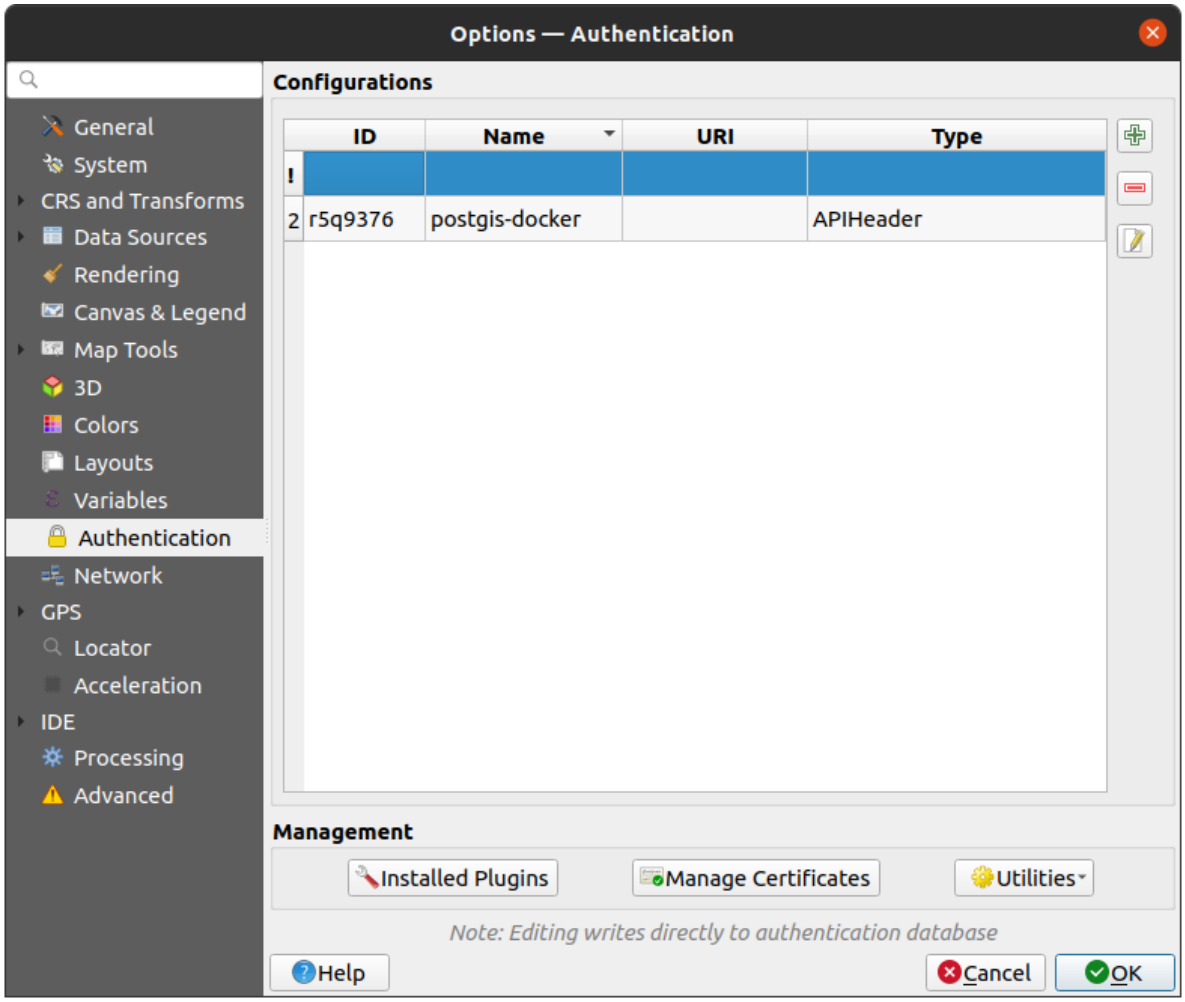


Fig. 5.22: Authentication settings

5.1.16 Network settings

General

- Define *Timeout for network requests (ms)* - default is 60000
- Define *Default expiration period for WMS Capabilities (hours)* - default is 24
- Define *Default expiration period for WMS-C/WMTS tiles (hours)* - default is 24
- Define *Max retry in case of tile or feature request errors*
- Define *User-Agent prefix* which defaults to `Mozilla/5.0`. This value will be prepended to both QGIS and OS version e.g., `Mozilla/5.0 QGIS/33801/Ubuntu 22.04.4 LTS` to shape the user-agent.

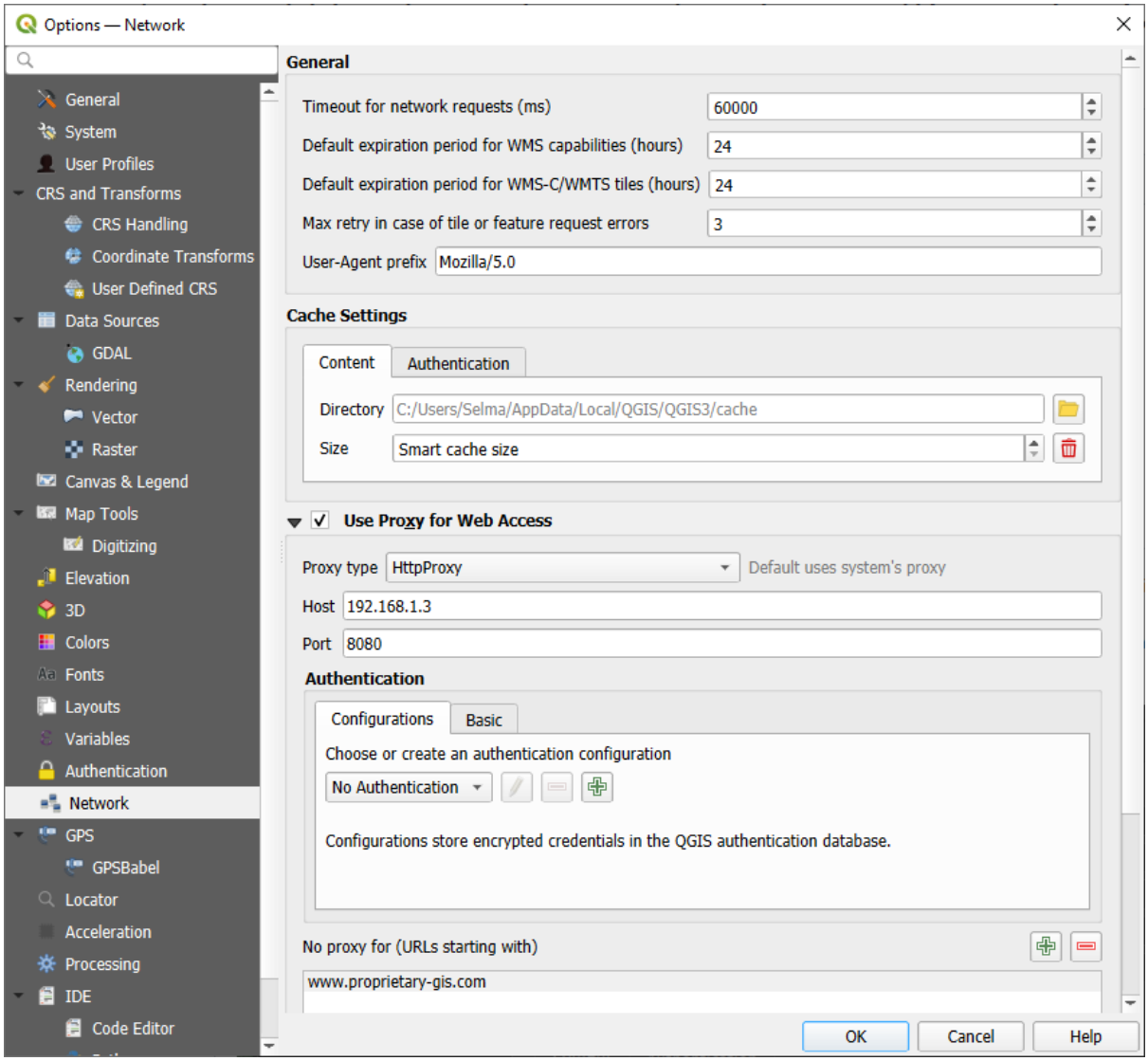


Fig. 5.23: Network and proxy settings

Cache settings

Defines the *Directory* and a *Size* for the cache specified in megabytes. You can clear the value to enable the *Smart cache size*, which sets the maximum cache size based on available disk space. Also offers tools to *automatically clear the connection authentication cache on SSL errors (recommended)*.

Proxy for web access

- *Use proxy for web access*
- Set the *Proxy type* according to your needs and define 'Host' and 'Port'. Available proxy types are:
  - *Default Proxy*: Proxy is determined based on system's proxy
  - *Socks5Proxy*: Generic proxy for any kind of connection. Supports TCP, UDP, binding to a port (incoming connections) and authentication.
  - *HttpProxy*: Implemented using the "CONNECT" command, supports only outgoing TCP connections; supports authentication.
  - *HttpCachingProxy*: Implemented using normal HTTP commands, it is useful only in the context of HTTP requests.
  - *FtpCachingProxy*: Implemented using an FTP proxy, it is useful only in the context of FTP requests.

Credentials of proxy are set using the *authentication widget*.

Excluding some URLs can be added to the text box below the proxy settings (see Fig. 5.23). No proxy will be used if the target url starts with one of the string listed in this text box.

If you need more detailed information about the different proxy settings, please refer to the manual of the underlying QT library documentation at https://doc.qt.io/archives/qt-5.9/qnetworkproxy.html#ProxyType-enum

Tip

Using Proxies

Using proxies can sometimes be tricky. It is useful to proceed by 'trial and error' with the above proxy types, to check if they succeed in your case.

5.1.17 GPS settings

The *GPS* dialog helps you configure GPS devices connections and properties in QGIS. It also provides settings for GPS tracking and data digitizing.

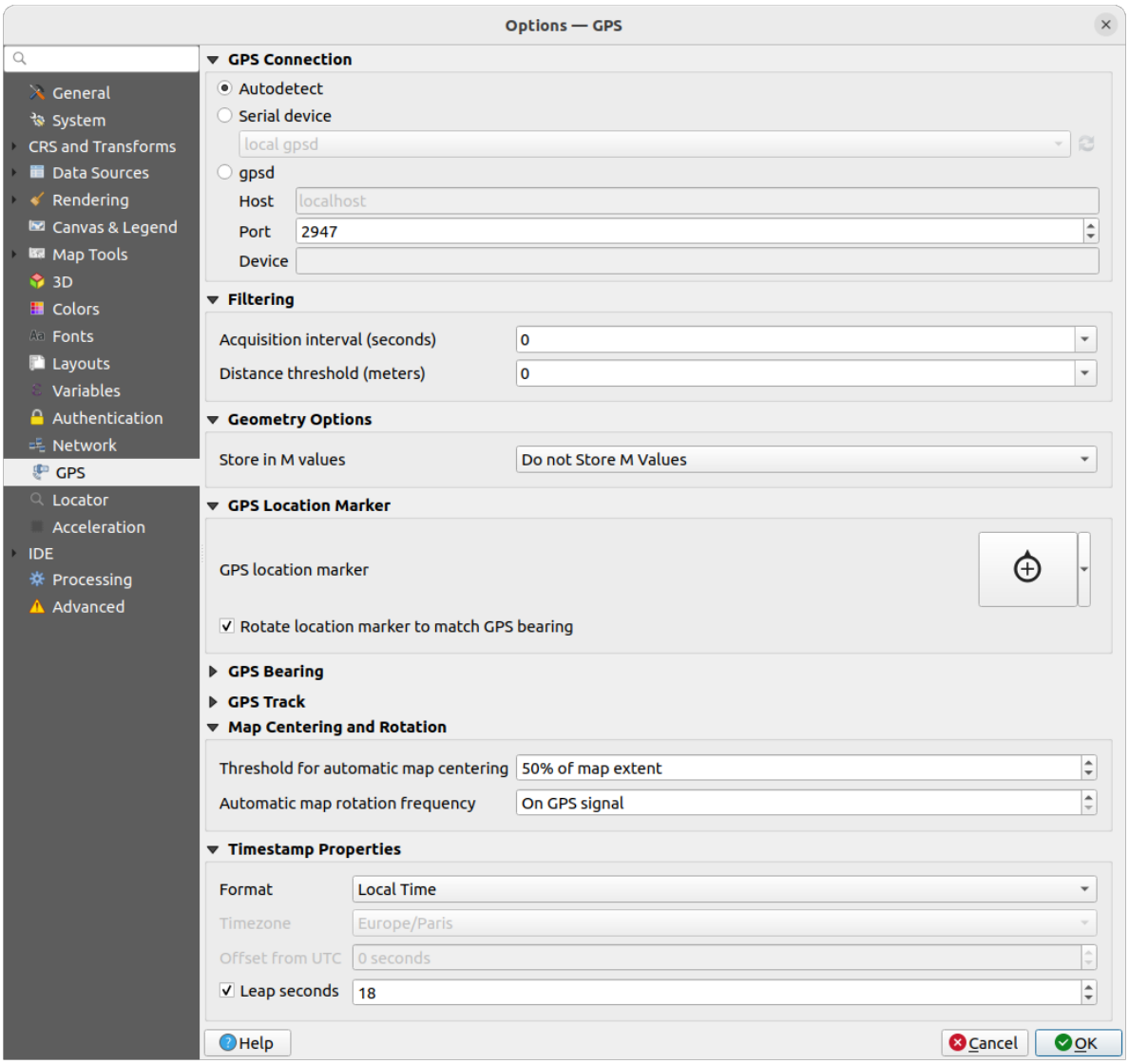


Fig. 5.24: GPS settings

You can specify:

- *GPS Connection*: provides different means to connect the device to QGIS
  - *Autodetect*
  - *Serial device* (reload required if a new GPS Device is connected)
  - *gpsd* (selecting the *Host* and *Port* your GPS is connected to, and providing a *Device* name)

  In case of connection problems, you can try to switch from one to another.
- *Filtering*: You can set an *Acquisition interval (seconds)* and/or a *Distance threshold (meters)* parameters to keep the cursor still active when the receiver is in static conditions.
- Under *Geometry Options*, the *Store in M values* option allows to create geometries with M values from the inbuilt GPS logging tools. This applies to both features digitized from GPS logs and from the *Log to Geopackage/SpatiaLite* functionality… Options include storing timestamps (as ms since epoch), ground speed, altitudes, bearings, and accuracy components as m values.
- *GPS Location Marker*
  - Configure a symbol for the *GPS location marker*, indicating the current GPS position

  - *Rotate location marker to match GPS bearing*: whether the marker symbol should be rotated to match the GPS direction

- *GPS Bearing*:
  - configure a *Bearing line style* using QGIS line symbol properties
  - set whether to *Calculate Bearings from travel direction*: If checked, the bearing reported by the GPS device will be ignored and the bearing will instead be calculated by the angle between the previous two GPS locations.
- *GPS Track*: set symbol to use for the *Track line style*
- *Map Centering and Rotation*: defines when the map canvas is updated according to the user displacement on the field:
  - *Threshold for automatic map centering*: defines the minimal offset of the GPS position from the map canvas center to trigger an automatic *map recentering* if enabled.
  - *Automatic map rotation frequency*: defines how often the map rotation to match the GPS bearing could happen; it can be on a custom duration or *On GPS signal*.
- *Timestamp Properties* to configure how time values are displayed and stored in the data. Parameters include the *Format* which can be:
  - *Local time*
  - *UTC*
  - *UTC with offset*, to account for daylight savings offsets or other complex time zone issues
  - a specific *Time zone*

  Moreover, *Leap seconds* correction can be applied, by adding the seconds to GPS timestamp.

GPSBabel

GPSBabel converts waypoints, tracks, and routes between popular GPS receivers such as Garmin or Magellan and mapping programs like Google Earth or Basecamp. Literally hundreds of GPS receivers and programs are supported. QGIS relies on GPSBabel to interact with these devices and *manipulate their data*.

For details on how-to, please refer to *Loading to or from a device*.

5.1.18 Locator settings

The *Locator* tab lets you configure the *Locator bar*, a quick search widget available on the status bar to help you perform searches in the application. It provides some default filters (with prefix) to use:

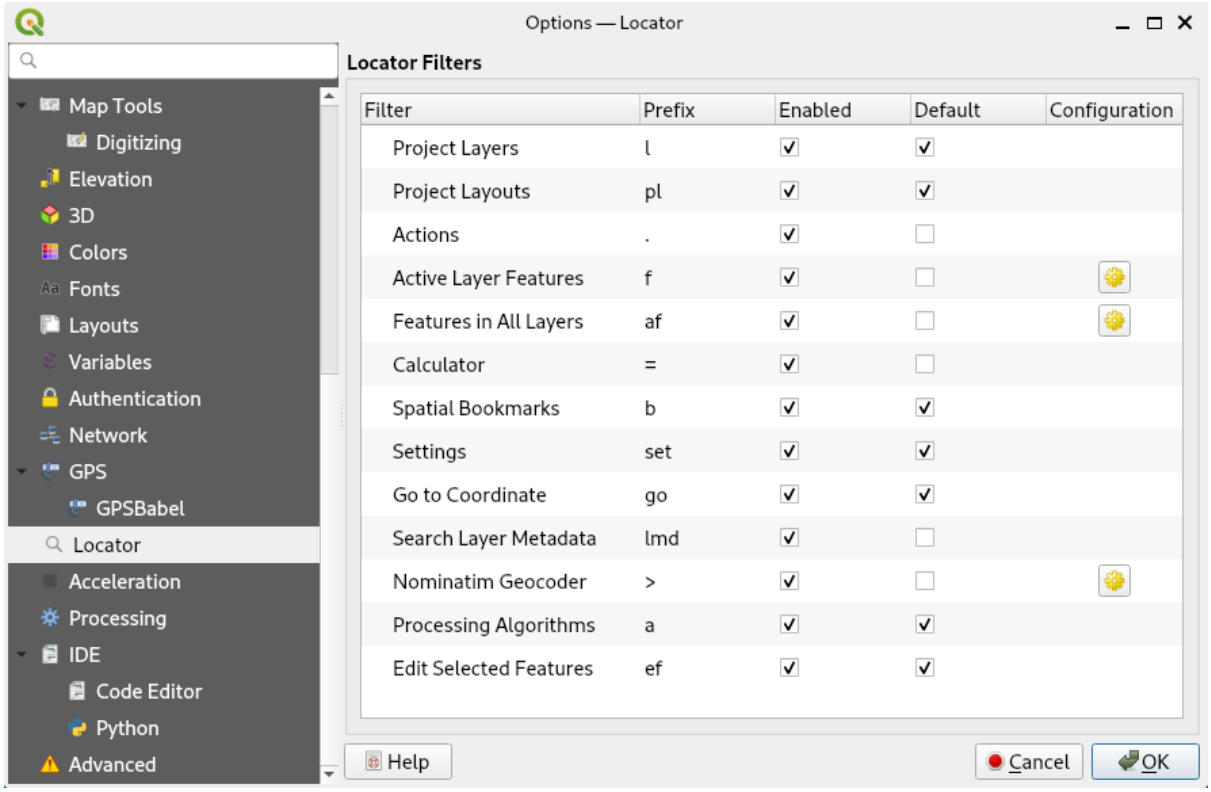


Fig. 5.25: Locator settings

- *Project Layers* (`l`): finds and selects a layer in the *Layers* panel.
- *Project Layouts* (`pl`): finds and opens a print layout.
- Actions (`.`): finds and executes a QGIS action; actions can be any tool or menu in QGIS, opening a panel…
- *Active Layer Features* (`f`): searches for matching attributes in any field from the current active layer and zooms to the selected feature. Press to configure the maximum number of results.
- *Features in All Layers* (`af`): searches for matching attributes in the *display name* of each *searchable layers* and zooms to the selected feature. Press to configure the maximum number of results and the maximum number of results per layer.
- *Calculator* (=): allows evaluation of any QGIS expression and, if valid, gives an option to copy the result to the clipboard.
- *Spatial Bookmarks* (`b`): finds and zooms to the bookmark extent.
- *Settings* (`set`): browses and opens project and application-wide properties dialogs.
- *Go to Coordinate* (`go`): pans the map canvas to a location defined by a comma or space separated pair of x and y coordinates or a formatted URL (e.g., OpenStreetMap, Leaflet, OpenLayer, Google Maps, …). The coordinate is expected in WGS 84 (`epsg:4326`) and/or map canvas CRS.
- *Nominatim Geocoder* (>): geocodes using the Nominatim geocoding service of the OpenStreetMap Foundation. Press to limit results to one or more countries using list of country codes. This helps narrow down the search to specific countries, avoiding ambiguous results from other locations. For example, entering `DE` or `DE,FR` restricts results to Germany or Germany and France, respectively.
- Processing Algorithms (`a`): searches and opens a Processing algorithm dialog.
- *Edit Selected Features* (`ef`): gives quick access and runs a compatible *modify-in-place* Processing algorithm on the active layer.

In the dialog, you can:

- customize the filter *Prefix*, i.e., the keyword to use to trigger the filter
- set whether the filter is *Enabled*: the filter can be used in the searches and a shortcut is available in the locator bar menu
- set whether the filter is *Default*: a search not using a filter returns results from only the default filters categories.
- Some filters provide a way to configure the number of results in a search.

The set of default locator filters can be extended by plugins, eg for OSM nominatim searches, direct database searching, layer catalog searches, …

5.1.19 Acceleration settings

OpenCL acceleration settings.

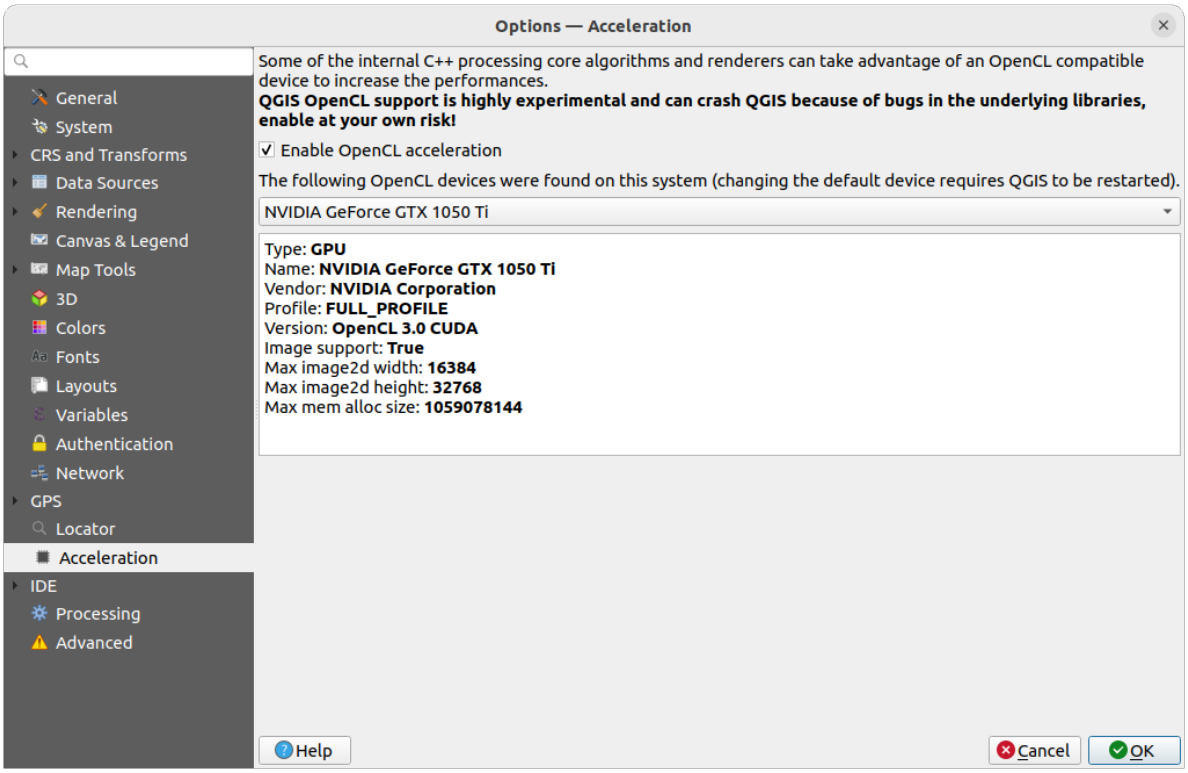


Fig. 5.26: Acceleration settings

Depending on your hardware and software, you may have to install additional libraries to enable OpenCL acceleration.

5.1.20 IDE settings

Under *GitHub access token*, you can generate a personal token allowing you to share code snippets from within the Python code editor. More details on GitHub authentication

Code Editor settings

In the *Code Editor* tab, you can control the appearance and behaviour of code editor widgets (Python interactive console and editor, expression widget and function editor, …).

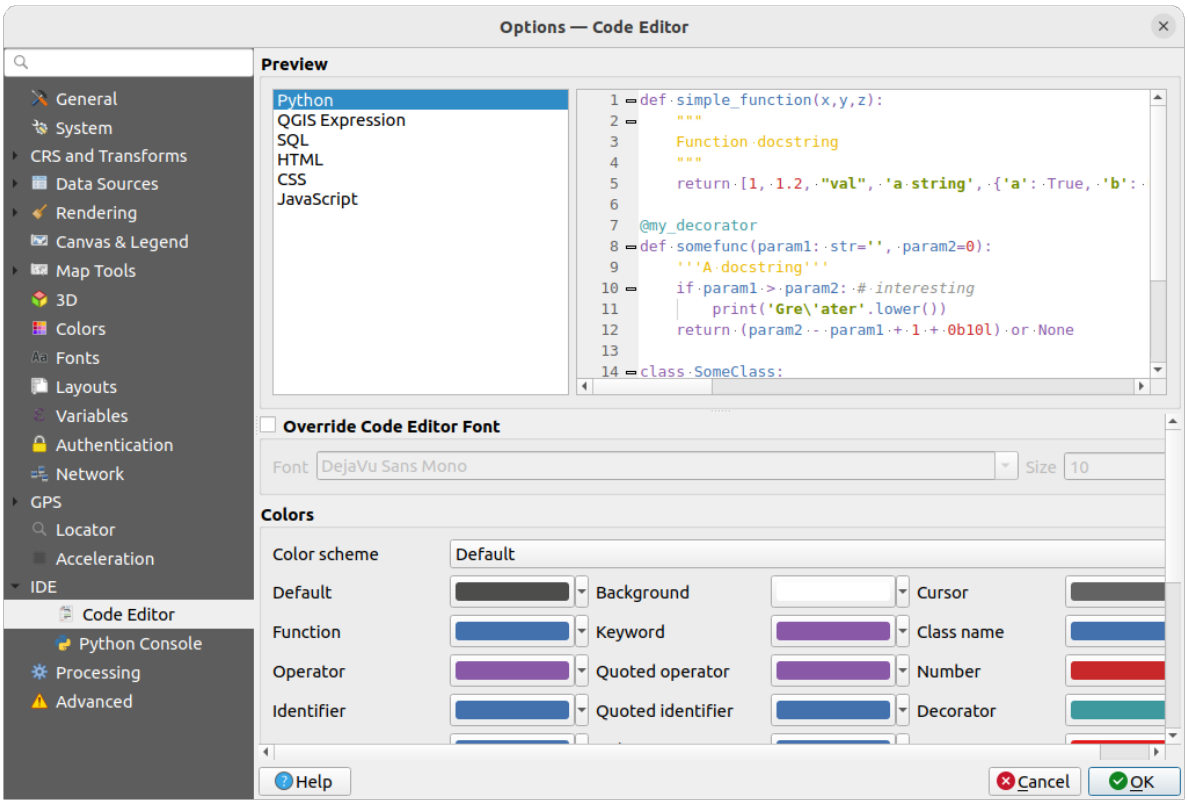


Fig. 5.27: Code Editor settings

At the top of the dialog, a widget provides a live preview of the current settings, in various coding languages (Python, QGIS expression, HTML, SQL, JavaScript). A convenient way to adjust settings.

- Check *Override Code Editor Font* to modify the default *Font* family and *Size*.
- Under the *Colors* group, you can:
  - select a *Color scheme*: predefined settings are `Default`, `Solarized Dark` and `Solarized Light`. A `Custom` scheme is triggered as soon as you modify a color and can be reset with selecting a predefined scheme.
  - change the *color* of each element in code writing, such as the colors to use for comments, quotes, functions, background, …

Python settings

The *Python* settings help you manage and control the behavior of the Python editors (*interactive console*, *code editor*, *project macros*, *custom expressions*, …). It can also be accessed using the Options… button from:

- the *Python console* toolbar
- the contextual menu of the *Python console* widget
- and the contextual menu of the code editor.

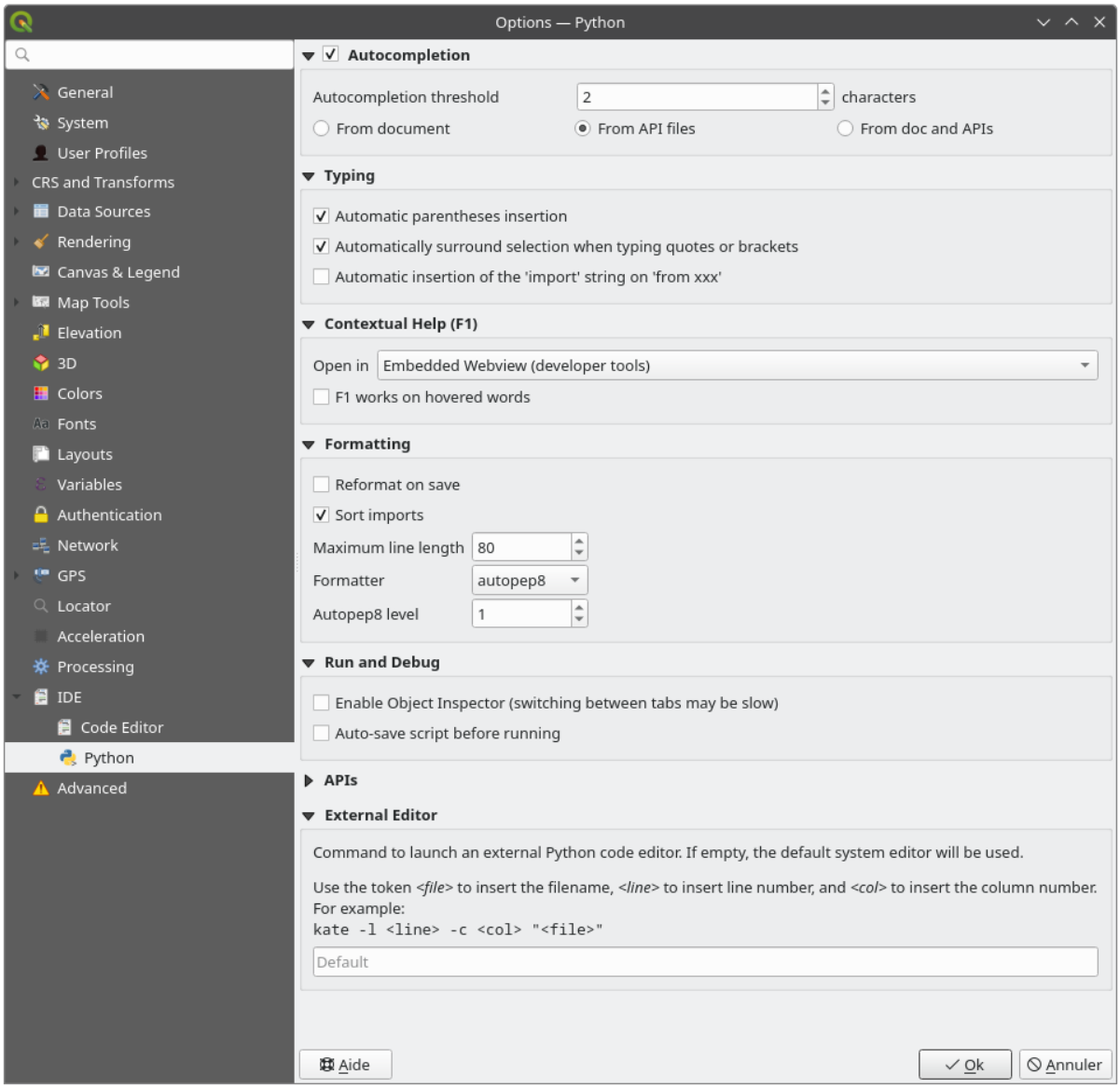


Fig. 5.28: Python settings

You can specify:

- *Autocompletion*: Enables code completion. You can get autocompletion from the current document, the *installed API* files or both.
  - *Autocompletion threshold*: Sets the threshold for displaying the autocompletion list (in number of characters)
- under *Typing*

  - *Automatic parentheses insertion*: When no text is selected, if an opening character (parentheses, quotes, brackets, …) is entered, inserts the matching closing character just after the cursor. Note that this behavior is disabled if the current cursor is inside a string or comment.
  - *Automatically surround selection when typing quotes or brackets*: When an opening character is entered, the selected text is enclosed in the opening/closing pair. Selection remains the same, so it is possible to quote a selected word and enclose it in parentheses just by typing `"` then `(`.

    Special case for multiline selection with quotes and double quotes: selection is enclosed in triple single/double quotes.
  - *Automatic insertion of the 'import' string on 'from xxx'*: Enables insertion of 'import' when specifying imports
- *Contextual Help (F1)* provides options to control QGIS behavior when requesting help on a PyQGIS or Qt object in a Python code sample. When pressing `F1`, QGIS will open the API page (or a search results page) matching:
  1. the selected text if any OR
  2. the word at the mouse position (if *F1 works on hovered words* is enabled) OR
  3. the word at the text cursor position

  This works for both Qt and PyQGIS API documentation. The page is opened either in QGIS, as an *Embedded webview (developer tools)*, or in the *Default system web browser*.
- under *Formatting*, you can add automated tools to reformat the code you are writing:
  - *Reformat on save*: formatting is applied just before saving the script
  - *Sort imports*: sorts 'import' statements using the isort library
  - *Maximum line length*: controls how the formatter will wrap the lines, and controls the editor ruler
  - *Formatter* - supported tools are *autopep8* and *black*, with dedicated option:
    - *Autopep8 level* - more details at autopep8 advanced usage
    - *Normalize quotes*: replaces all single quotes with double quotes if possible
- under *Run and Debug*
  - *Enable Object Inspector (switching between tabs may be slow)*
  - *Auto-save script before running*: Saves the script automatically when executed. This action will store a temporary file (in the temporary system directory) that will be deleted automatically after running.

Under *APIs*, you can specify which APIs your code should rely on:

- *Using preloaded APIs file*: You can choose if you would like to use the preloaded API files. By default, supported APIs are:
  - Python
  - PyQGIS
  - PyQt
  - QScintilla2
  - osgeo-gdal-ogr
  - osgeo-geos

  You can also choose to add specific API files.

- *Using prepared APIs file*: If checked, the chosen `*.pap` file will be used for code completion. To generate a prepared API file you have to load at least one `*.api` file and then compile it by clicking the *Compile APIs...* button.

The *External Editor* group allows you to provide command line instructions to launch an external Python code editor, given a file name, a line and column number. If empty, the default system editor will be used.

5.1.21 Processing settings

The *Processing* tab provides you with general settings of tools and data providers that are used in the QGIS Processing framework. More information at *Configuring the Processing Framework*.

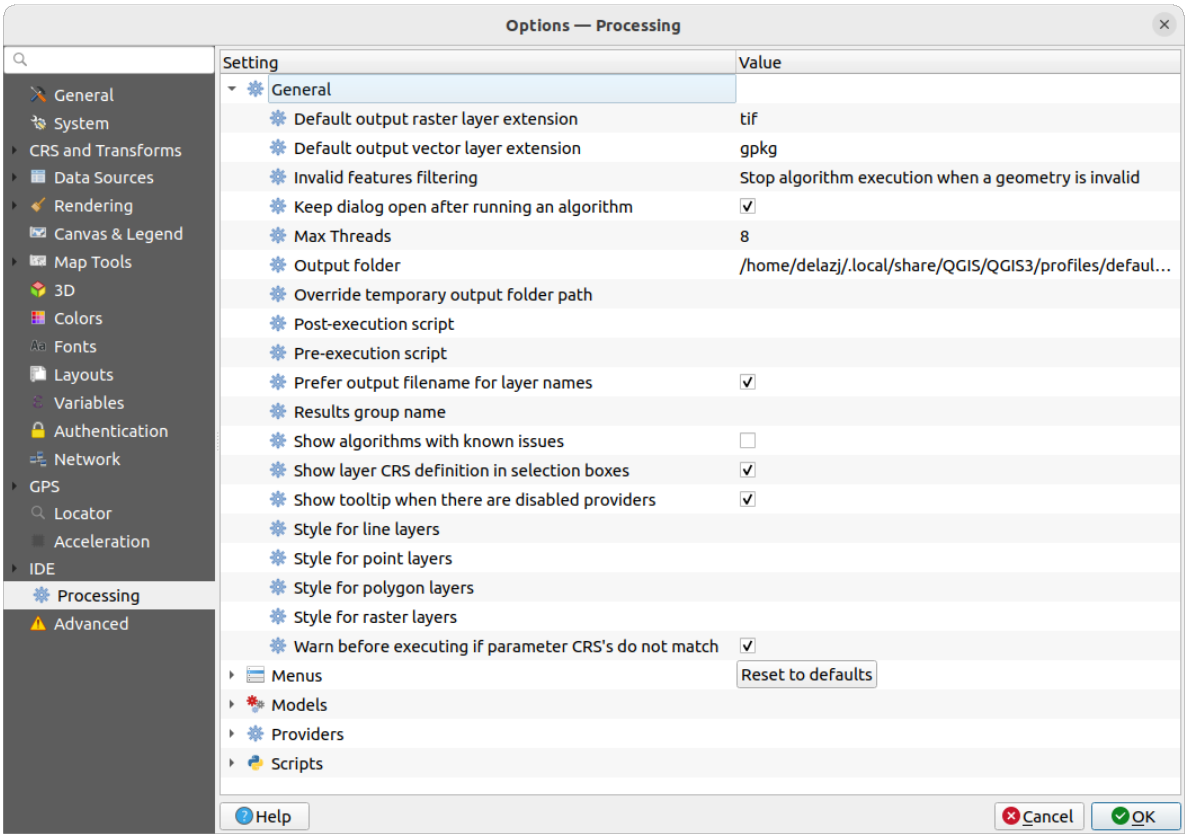


Fig. 5.29: Processing settings

5.1.22 Advanced settings

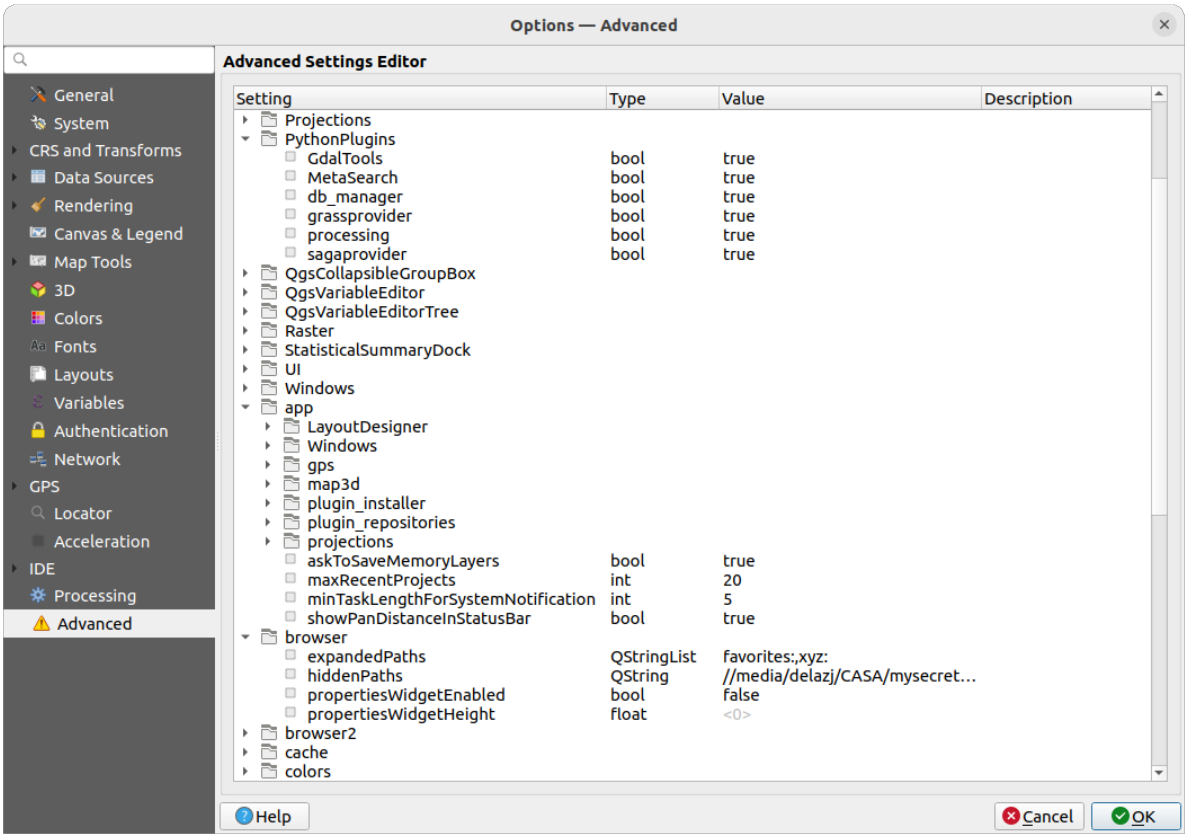


Fig. 5.30: Advanced settings

All the settings related to QGIS (UI, tools, data providers, Processing configurations, default values and paths, plugins options, expressions, geometry checks…) are saved in a `QGIS/QGIS3.ini` file under the active *user profile* directory. Configurations can be shared by copying this file to other installations.

From within QGIS, the *Advanced* tab offers a way to manage these settings through the *Advanced Settings Editor*. After you promise to be careful, the widget is populated with a tree of all the existing settings, and you can edit their value. Right-click over a setting or a group and you can delete it (to add a setting or group, you have to edit the `QGIS3.ini` file). Changes are automatically saved in the `QGIS3.ini` file.

Warning

Avoid using the Advanced tab settings blindly

Be careful while modifying items in this dialog given that changes are automatically applied. Doing changes without knowledge can break your QGIS installation in various ways.

5.2 Working with User Profiles

5.2.1 The concept

The *Settings* ► *User Profiles* menu provides functions to set and access user profiles. A user profile is a unified application configuration that allows to store in a single folder:

- all the *global settings*, including locale, projections, authentication settings, color palettes, shortcuts…
- GUI configurations and *customization*
- grid files and other proj helper files installed for datum transformation
- installed *plugins* and their configurations
- project templates and history of saved project with their image preview
- *processing settings*, logs, scripts, models.

By default, a QGIS installation contains a single user profile named `default`. But you can create as many user profiles as you want:

1. Click the *Settings* ► *User Profiles* ► *New profile…* entry.
2. You'll be prompted to provide a profile name, creating a folder of the same name under `~/<UserProfiles>/` where:
   - `~` represents the **HOME** directory, which on Windows is usually something like `C:\Users\<username>`.
   - and `<UserProfiles>` represents the main profiles folder, i.e.:
     - `.local/share/QGIS/QGIS3/profiles/`
     - `%AppData%\Roaming\QGIS\QGIS3\profiles\`
     - `Library/Application Support/QGIS/QGIS3/profiles/`

   The user profile folder can be opened from within QGIS using the *Open Active Profile Folder*.
3. A new instance of QGIS is started, using a clean configuration. You can then set your custom configurations.

If you have more than one profile in your QGIS installation, the name of the active profile is shown in the application title bar between square brackets.

As each user profile contains isolated settings, plugins and history they can be great for different workflows, demos, users of the same machine, or testing settings, etc. And you can switch from one to the other by selecting them in the *Settings* ► *User Profiles* menu. You can also run QGIS with a specific user profile from the *command line*.

Tip

Run QGIS under a new user profile to check for bug persistence

Bugs you may encounter with functions in QGIS can be related to leftovers in the current user profile. Running QGIS under another user profile can help you solve them or check the origin of the issue. The general advice is to launch QGIS under a new user profile, thus a cleaner configuration, and run the commands again.

If the bug prevents you to create a new user profile from within the *Settings* ► *User Profiles* menu, you can either:

- Rename in the file explorer, the "broken" user profile folder in the `QGIS3/profiles` folder and restart QGIS. A new `default` user profile will be created and executed.
- Start QGIS from the command line, using the new *profile name* argument:

```
qgis-ltr --profile newprofilename
```

5.2.2 Setting user profile

By default, QGIS opens a new session with the profile of the last closed session. This, among other settings, can be customized in *Settings* ► *Options* ► *User Profiles* tab:

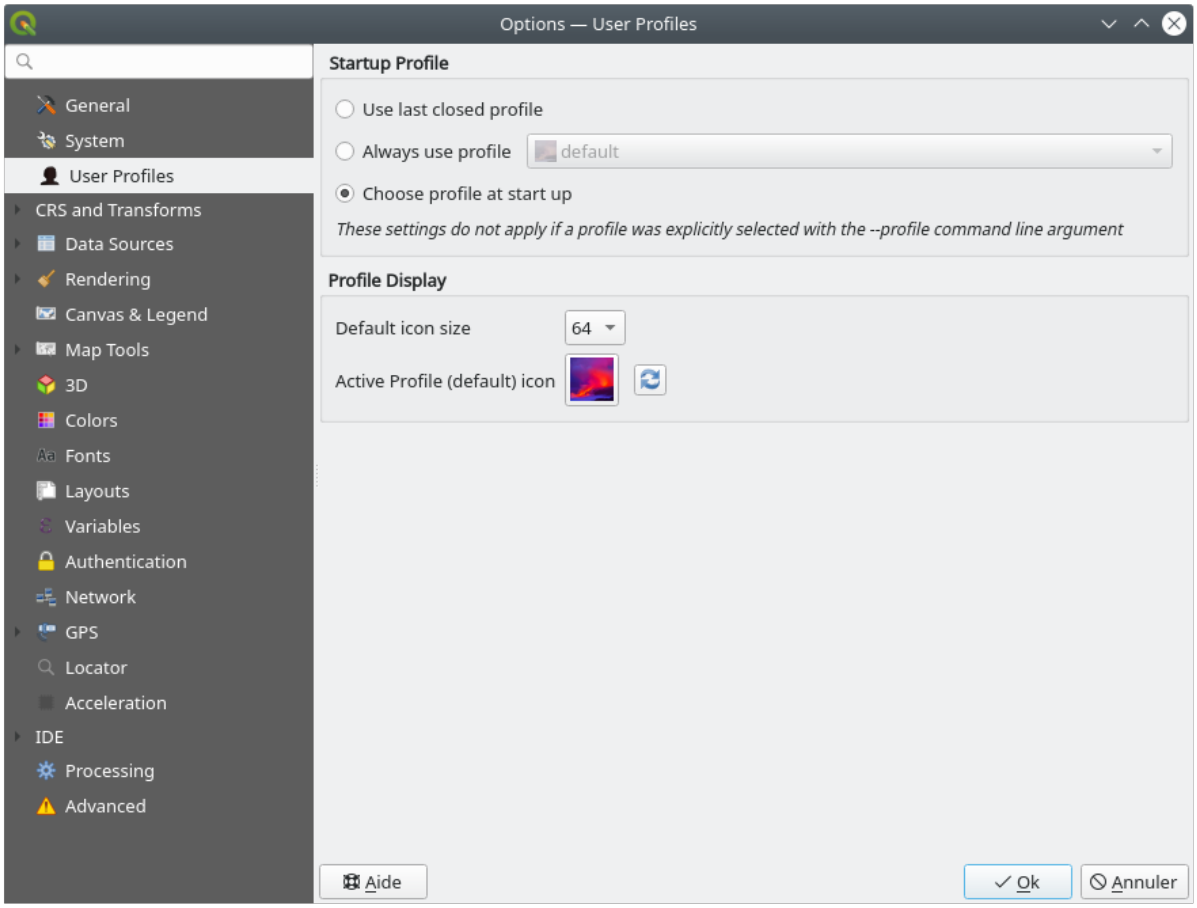


Fig. 5.31: User profiles settings

- *Startup profile*: indicates the user profile to use when starting a QGIS session. It can be:
  - *Use last closed profile*
  - a specific user profile to select from *Always use profile* drop-down menu
  - *Choose profile at start up*: Opens the *User Profile Selector* dialog listing available user profiles. Double-click an entry or select a profile and press *OK* to use that user profile for the starting session. It is also possible to *Add new profile* to the list: a profile is created from scratch and automatically run with the opening session.
- Under *Profile display*, you can set:
  - The size of icons to use when selecting a profile from the *User Profile Selector* dialog.
  - A specific icon to display next to the current profile in the *Settings* ► *User profiles* menu or the *User Profile Selector* dialog. Press Reset profile icon to remove custom modifications.

5.3 Project Properties

In the properties window for the project under *Project* ► *Project Properties*, you can set project-specific options. The project-specific options overwrite their equivalent in the *Options* dialog described above.

5.3.1 General Properties

In the  *General* tab, the *General settings* let you:

- see the location of the project file
- set the folder for the project home (available in the *Project home* item of the *Browser* panel). The path can be relative to the folder of the project file (type it in) or absolute. The project home can be used for storing data and other content that is useful for the project. Convenient when dataset and project files are not stored at the same place. If not filled, the *Project home* defaults to the project file folder.
- give a title to the project beside the project file path
- choose the color to use for features when they are selected
- choose the background color: the color to use for the map canvas
- set whether the path to layers in the project should be saved as absolute (full) or as relative to the project file location. You may prefer relative path when both layers and project files can be moved or shared or if the project is accessed from computers on different platforms.
- choose to avoid artifacts when project is rendered as map tiles. Note that checking this option can lead to performance degradation.
- *Remember attribute tables windows and docks between sessions*: If checked for a project, then any opened attribute tables will be saved into the project and immediately restored when loading that project. This can improve workflows when you constructed a project with a particular set of attribute table configurations for your requirements, and re-setting up these attribute tables is a hassle.

Calculating areas and distances is a common need in GIS. However, these values are really tied to the underlying projection settings. The *Measurements* frame lets you control these parameters. You can indeed choose:

- the *Ellipsoid*, on which distance, area and angle calculations are entirely based; it can be:
  - **None/Planimetric**: returned values are in this case cartesian measurements. This option can be set as default for new projects from the *Settings* ►  *Options* ►  *CRS Handling* menu
  - a **Custom** one: you'll need to set values of the semi-major and semi-minor axes.
  - or an existing one from a predefined list (Clarke 1866, Clarke 1880 IGN, New International 1967, WGS 84…).
- the *Units for distance measurements* for length and perimeter, and the *Units for area measurements*. These settings default to their corresponding *global options* and can be manually changed for the current project. They are used in:
  - Attribute table field update bar
  - Field calculator calculations
  - *Identify tool* derived length, perimeter and area values
  - *measure dialog*
  - *scale bar decoration*

> **Note**
>
> To ensure measurement units (for distance and area) always reflect the units of the current project's CRS, select *Map Units* for these options.

- the *Scale calculation method*: when working on large map extents, especially with geographic CRS, the scale bar may not reflect actual distance measurement depending on the latitude at which the calculation is done. This setting helps you define the area you want the calculation to be taken from:
  - *Average top, middle and bottom scales*
  - *Calculate along top of map*
  - *Calculate along middle of map* (the default)
  - *Calculate along bottom of map*
  - *Always calculate at equator*, regardless of the actual visible map extent. It can be used to provide a consistent, static scale for maps in geographic reference systems, regardless of the latitudes actually visible in the map (permitting consistent appearance of these maps when rendering relies on scale based visibility or calculations). Otherwise a project in e.g. `EPSG:4326` which uses scale based visibility of layers and symbols will see layers and features "randomly" disappear as the map is panned, even though you have not zoomed in or out of the map.

    > **Note**
    >
    > This method is only applicable when calculating scales with a degree based reference system, and while it ensures that the scale remains constant and does not change as the map is panned, it will calculate misleading scales when the map extent is not close to the equator.

  Changing the scale calculation method has the following impacts:
  - New layout scale bars will default to the project's scale calculation method
  - The scale calculations for the status bar widget and map renders will be changed (including flow on impacts like the value of `@map_scale` variable, scale based visibility of layers and symbols)
  - This also affects QGIS server map rendering
  - Processing algorithms which render maps will respect the project's scale calculation method

> **Note**
>
> Symbology sizes in map units are NOT affected by this setting.

The *Coordinate and Bearing display* allows you to customize the display of:

- the coordinates shown in the *Coordinates* box on QGIS status bar and in the *Derived* section of the  Identify features tool's results
- the bearing value displayed in the status bar for the map canvas panning direction and by the  Measure bearing tool.

Available parameters are:

- *Display coordinates using* either:
  - `Map Units`, based on the project CRS
  - `Map Geographic (degrees)`: based on the project CRS if it is of geographic type, otherwise uses its associated geographic CRS. This is helpful e.g., for non-earth celestial bodies.

  - or `Custom Projection Units`: allows to rely on any CRS you desire for coordinates display

- In the *Coordinate CRS* option, you can view or define the CRS to use depending on your display mode.
- *Coordinate format*: you can configure it as `Decimal Degrees`, `Degrees, Minutes` or `Degrees, Minutes, Seconds`, and whether it should display:
  - ☐ *Show directional suffix*
  - ☐ *Show leading zeros for minutes and seconds*
  - ☐ *Show leading zeros for degrees*
  - ☐ *Show trailing zeros*
- *Coordinate precision*: the number of decimal places can be automatic (derived from the type of CRS) or set manually
- *Coordinate order*: you can opt to display the coordinates in the native order of the CRS (`Default`) or switch it to either `Easting, Northing (Longitude, Latitude)` or `Northing, Easting (Latitude, Longitude)` order
- *Bearing format* possible values are `0 to 180°, with E/W suffix`, `-180 to +180°` or `0 to 360°`. The number of *Decimal places* as well as whether to *Show trailing zeros* can be set.

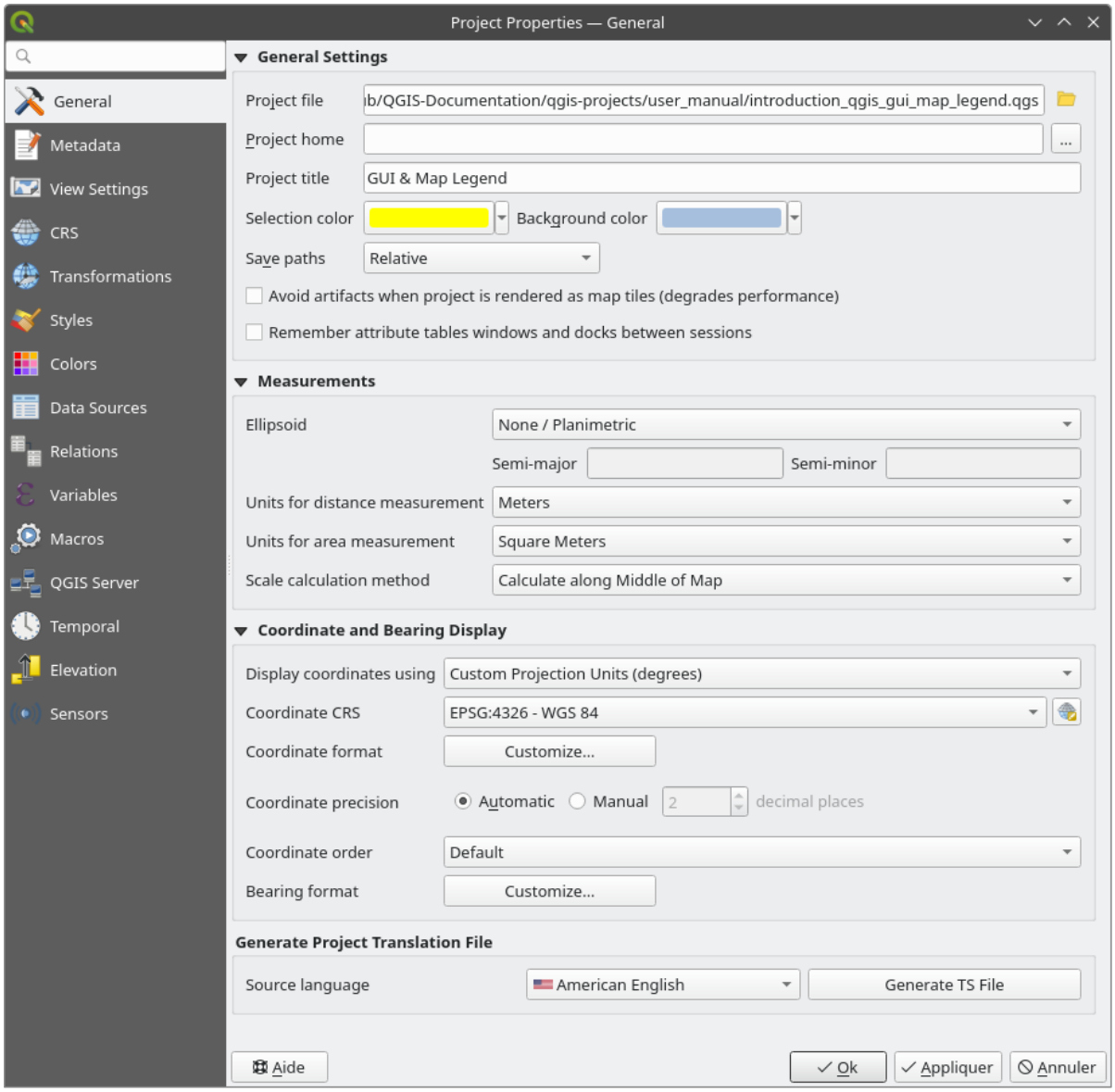


Fig. 5.32: General tab of the Project Properties dialog

5.3.2 Metadata Properties

The *Metadata* tab allows detailed metadata to be defined, including (among the others): author, creation date, language, abstracts, categories, keywords, contact details, links, history. There is also a validation functionality that checks if specific fields were filled, anyway this is not enforced. See *vector layer metadata properties* for some details.

5.3.3 View Settings

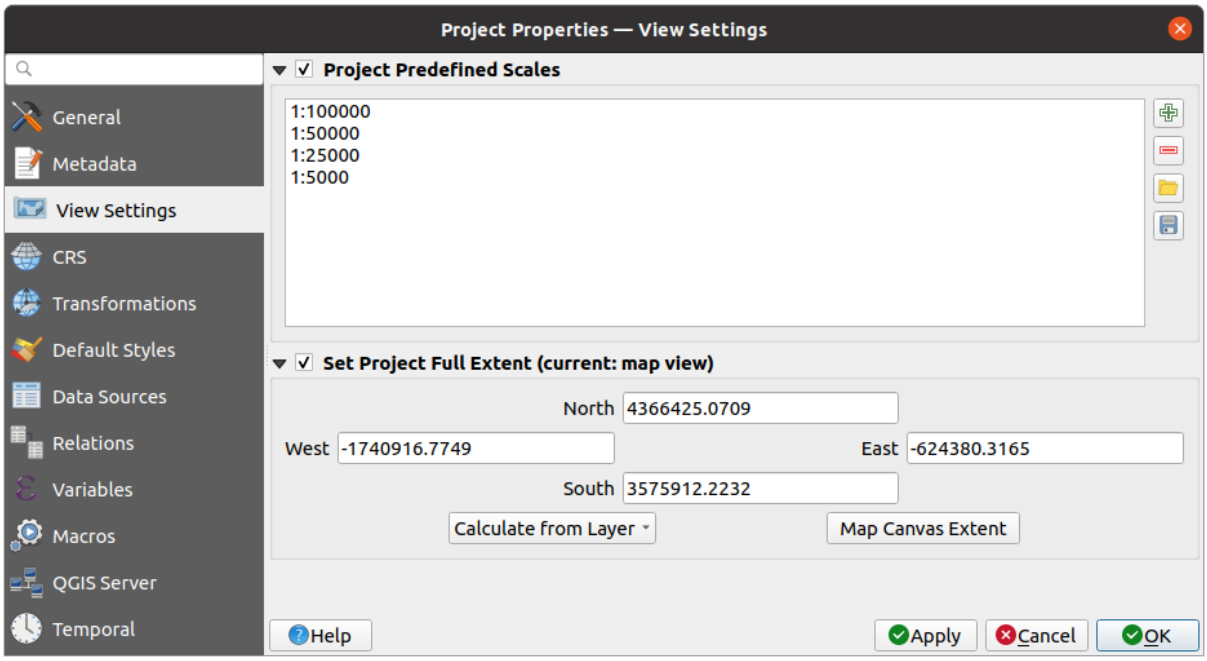


Fig. 5.33: View Settings tab of the Project Properties dialog

The *View Settings* tab provides means to control the project map canvas. You can:

- set *Project predefined scales*: the list of scales to display in scale-related drop-down widgets, such as the status bar *Scale*, the visibility scales selector or secondary 2D map view settings,… in replacement of the global *predefined scales*.
- *Set Project full Extent*: this extent will be used instead of the extent of all layers when zooming to full map extent ( ). It's useful when a project contains web layers/national layers/global layers yet the actual area of interest for the project is a smaller geographic area. The project full extent coordinates can be set with the *extent selector* widget.

5.3.4 CRS Properties

> **Note**
>
> For more information on how QGIS handles project projection, please read the dedicated section at *Working with Projections*.

The *CRS* tab helps you set the coordinate reference system to use in this project. It can be:

- *No CRS (or unknown/non-Earth projection)*: layers are drawn based on their raw coordinates
- or an existing coordinate reference system that can be *geographic*, *projected* or *user-defined*. Layers added to the project are translated on-the-fly to this CRS in order to overlay them regardless their original CRS.

5.3.5 Transformations Properties

The *Transformations* tab helps you control the layers reprojection settings by configuring the datum transformation preferences to apply in the current project. As usual, these override any corresponding global settings. See *Datum Transformations* for more details.

5.3.6 Styles Properties

Under *Styles* tab, you can configure symbols and colors inherent to the project, allowing to safely share the project among different machines.

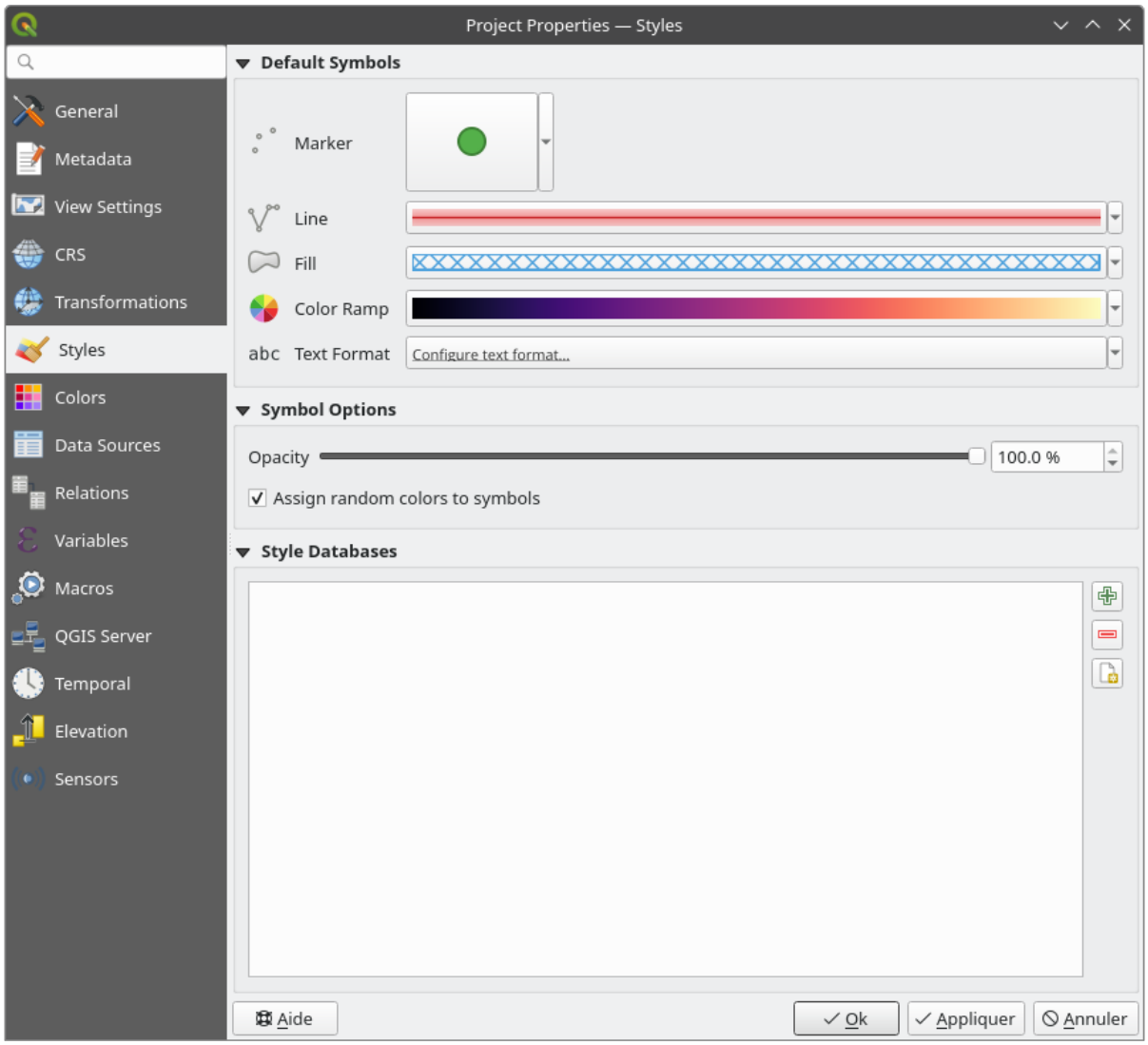


Fig. 5.34: Styles tab

Default symbols

The *Default Symbols* group lets you control how new layers will be drawn in the project when they do not have an existing `.qml` style defined. You can set *Marker*, *Line*, *Fill* to apply depending on the layer geometry type as well as default *Color Ramp* and *Text Format* (e.g., when enabling labeling). Any of these items can be reset using the *Clear* entry from the corresponding drop-down widget.

Symbol options

In the *Symbol options* group, you can:

- Apply a default *Opacity* to new layers
- *Assign random colors to symbols*, modifying the symbols fill colors, hence avoiding same rendering for all layers.

Style Database

A style database in QGIS is a structured repository designed to store symbols, text formats, and other styling elements. It serves as a centralized location where you can organize and manage your symbology resources efficiently. You can create a dedicated style database for a specific client, housing symbols tailored to that client's need. This ensures a clean and organized approach, without cluttering the default style database. In multi-user environments, it's possible to store project-specific styles in a shared location. By linking a project to these styles, all users within the project gain access to common symbology, eliminating the need for manual import or updates to individual local style database. Storing symbols in a project file offers a practical solution, preventing users from overcrowding their global style database with project-specific symbology. This approach guarantees that other users working on the same project immediately have access to all the necessary symbology upon loading the project.

In the *Style Database* section you can choose to *Add* or *Remove* style database or you can *Create new style database*. When you add or remove a style database in this section, the changes will be automatically reflected in the *Symbology Properties*.

5.3.7 Colors properties

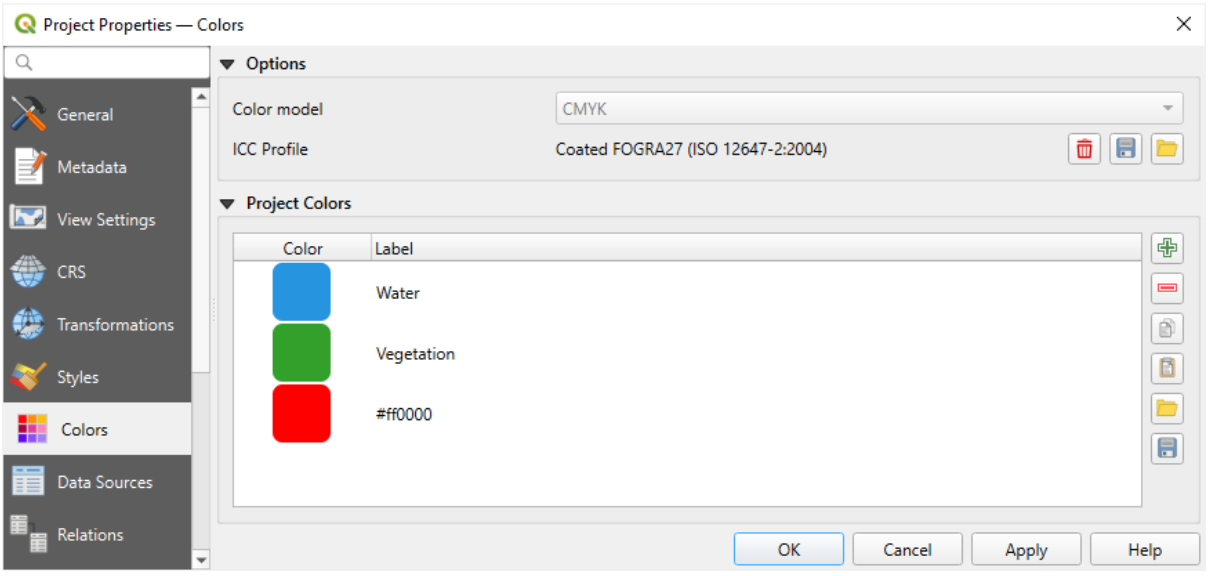


Fig. 5.35: Colors tab

In the  *Colors* tab, you can select the *Color model*, i.e., whether selecting a color should by default be defined as `RGB` or `CMYK`. Any color defined in a different color model than the one specified here will be converted to this color model when exporting a layout.

Additionally, an option is available for selecting an *ICC Profile* which allows you to  *Load* a specific ICC profile that defines the color space for your project. After you load the ICC profile, you can choose to  *Save ICC Profile* to a file on disk or to  *Remove selected ICC Profile*. Once applied, QGIS will enforce the selected color model to ensure consistent color representation.

Note

Note that the *ICC Profile* option will only be visible if QGIS is built with Qt 6.8.0 or higher.

There is also an additional section where you can define specific colors for the running project. Like the *global colors*, you can:

-  *Add* or  *Remove* color
-  *Copy* or  *Paste* color
-  *Import* or  *Export* the set of colors from/to `.gpl` file.

Double-click a color in the list to tweak or replace it in the *Color Selector* dialog. You can also rename it by double-clicking in the *Label* column.

These colors are identified as *Project colors* and listed as part of *color widgets*.

Tip

Use project colors to quickly assign and update color widgets

Project colors can be refered to using their label and the color widgets they are used in are bound to them. This means that instead of repeatedly setting the same color for many properties and, to avoid a cumbersome update you can:

1. Define the color as a project color
2. Click the *data defined override widget* next to the color property you want to set
3. Hover over the *Color* menu and select the project color. The property is then assigned the expression `project_color('color_label')` and the color widget reflects that color.
4. Repeat steps 2 and 3 as much as needed
5. Update the project color once and the change is reflected EVERYWHERE it's in use.

5.3.8 Data Sources Properties

In the  *Data Sources* tab, you can:

- *Transaction mode*, defines how edits are sent to the data provider:
  - *Local Edit Buffer*: edits are buffered locally and sent to the provider when toggling layer editing mode or clicking *Save edits*.
  - *Automatic Transaction Groups*: on supported datasources (postgres and geopackage databases) the edit state of all tables that originate from the same database are synchronized and executed in a server side transaction. Also, instead of buffering edit changes locally, they are directly sent to a transaction in the database which gets committed when toggling layer editing mode or clicking *Save edits*.

  - *Buffered Transaction Groups*: all editable layers, disregarding from which provider, are toggled synchronously and all edits are saved in a local edit buffer. Saving changes is executed within a single transaction on all layers (per provider).

  Note that you can change this option only if no layer is being edited in the project.

- *Evaluate default values on provider side*: When adding new features in a PostgreSQL table, fields with default value constraint are evaluated and populated at the form opening, and not at the commit moment. This means that instead of an expression like `nextval('serial')`, the field in the *Add Feature* form will display expected value (e.g., `25`).
- *Remember editable layer status between sessions*: makes sure that all layers that are editable in a project will be remembered as such when saving the project, as well as making sure that those layers are immediately made editable whenever the project is restored.
- Configure the *Layers Capabilities*, i.e.:
  - Set (or disable) which layers are `identifiable`, i.e., will respond to the *identify tool*. By default, layers are set queryable.
  - Set whether a layer should appear as `read-only`, meaning that it can not be edited by the user, regardless of the data provider's capabilities. Although this is a weak protection, it remains a quick and handy configuration to avoid end-users modifying data when working with file-based layers.
  - Define which layers are `searchable`, i.e., could be queried using the *locator widget*. By default, layers are set searchable.
  - Define which layers are defined as `required`. Checked layers in this list are protected from inadvertent removal from the project.
  - Define which layers are `private`, i.e., hidden from the *Layers* panel. This is meant for accessory layers (basemap, join, lookups for value-relations, most probably aspatial layers, …) that you still need in a project but you don't want them to pollute the legend tree and other layer selection tools. If set visible, they are still displayed in the map canvas and rendered in the print layout legend. Use the *Filter legend* ► *Show private layers* option in the *Layers* panel top toolbar to temporarily turned them on for any interaction.

  The *Layers Capabilities* table provides some convenient tools to:

  - Select multiple cells and press *Toggle Selection* to have them change their checkbox state;
  - *Show spatial layers only*, filtering out non-spatial layers from the layers list;
  - *Filter layers…* and quickly find a particular layer to configure.
- Under the *Advanced Settings* group, you can select *Trust project when data source has no metadata*: To speed up project loading by skipping data checks. Useful in QGIS Server context or in projects with huge database views/materialized views. The extent of layers will be read from the QGIS project file (instead of data sources) and when using the PostgreSQL provider the primary key unicity will not be checked for views and materialized views.

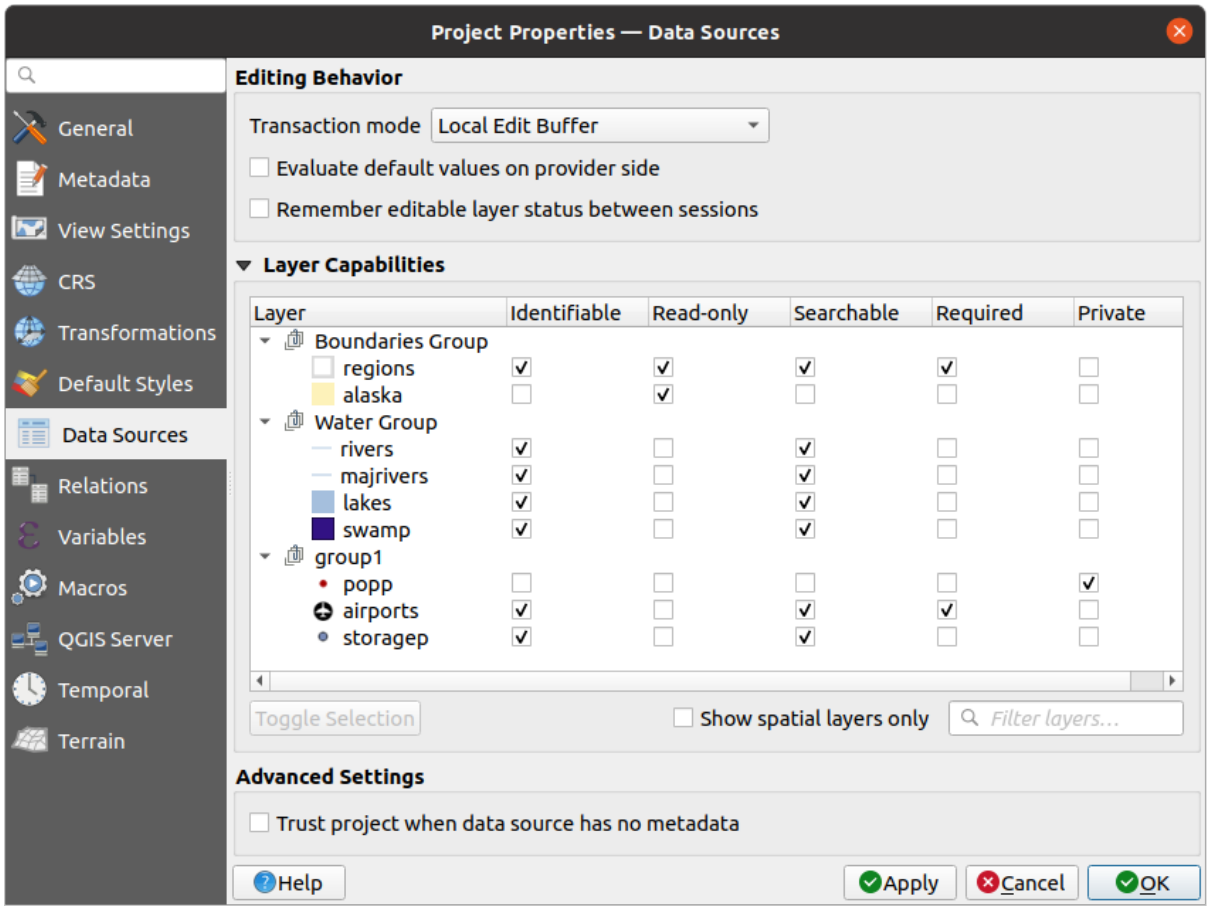


Fig. 5.36: Data Sources tab

5.3.9 Relations Properties

The *Relations* tab is used to define relations between layers. The relations can be of one to one, many to many or polymorphic type. They are defined in the project properties dialog. Once relations exist for a layer, a new user interface element in the form view (e.g., when identifying a feature and opening its form) will list the related entities. This provides a powerful way to express e.g., the inspection history on a length of pipeline or road segment. You can find out more about relations support in section *Setting relations between multiple layers*.

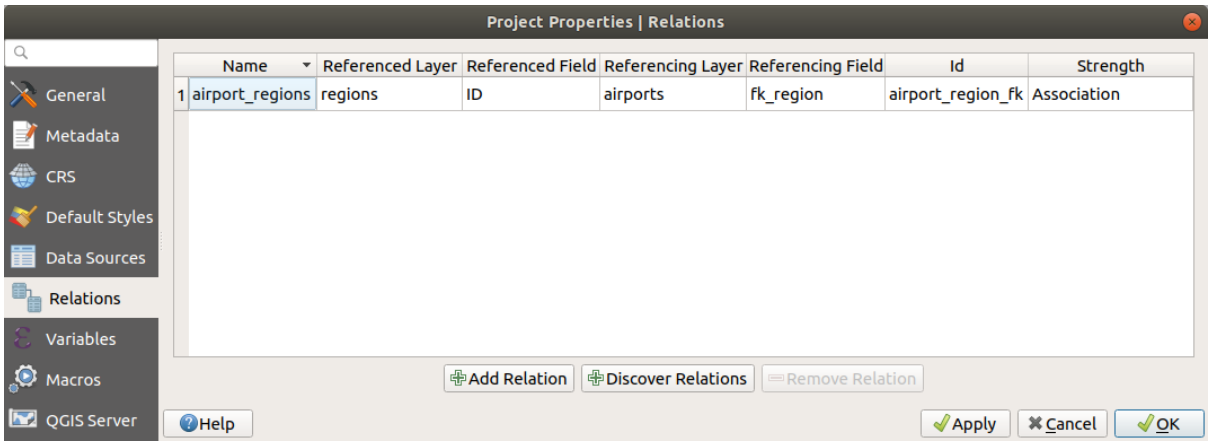


Fig. 5.37: Relations tab

5.3.10 Variables Properties

The *Variables* tab lists all the variables available at the project's level (which includes all global variables). Besides, it also allows the user to manage project-level variables. Click the button to add a new custom project-level variable. Likewise, select a custom project-level variable from the list and click the button to remove it. More information on variables usage in the General Tools *Storing values in Variables* section.

5.3.11 Macros Properties

The *Macros* tab is used to edit Python macros for projects. Currently, only three macros are available: `openProject()`, `saveProject()` and `closeProject()`.

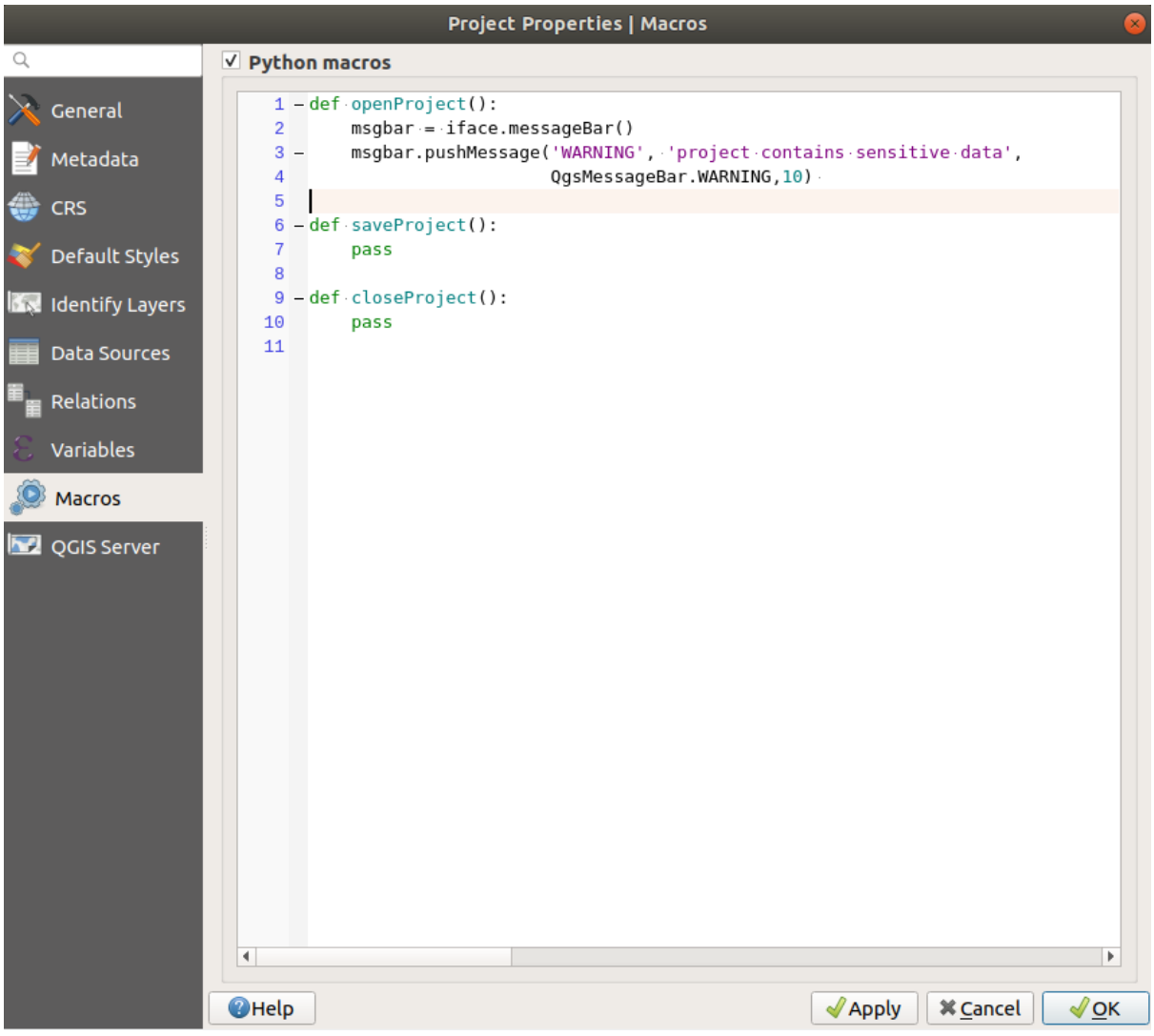


Fig. 5.38: Macro settings

5.3.12 QGIS Server Properties

The *QGIS Server* tab allows you to configure your project in order to publish it online. Here you can define information about the QGIS Server WMS and WFS capabilities, extent and CRS restrictions. More information available in section Creatingwmsfromproject and subsequent.

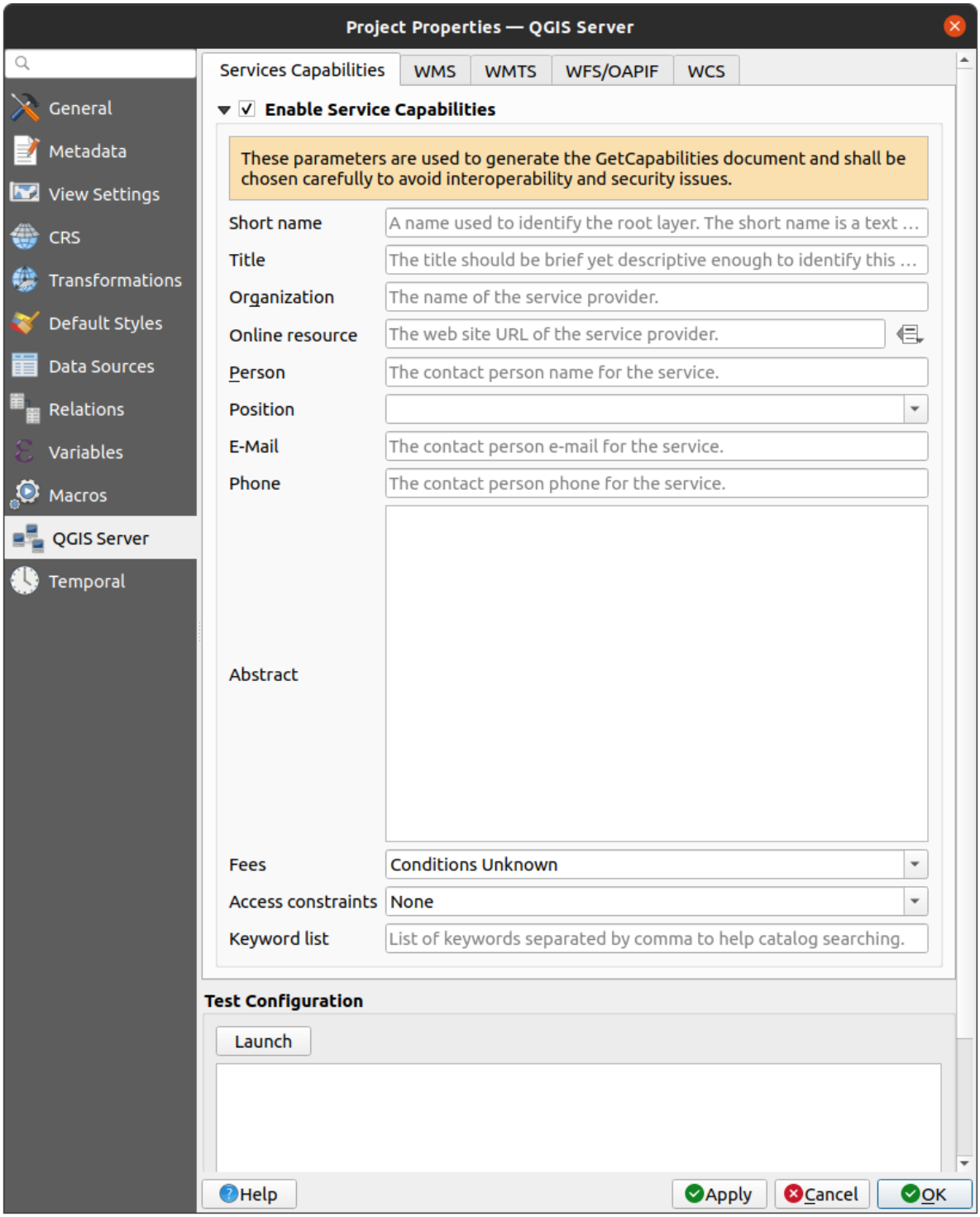


Fig. 5.39: QGIS Server settings

5.3.13 Temporal Properties

The *Temporal* tab is used to set the temporal range of your project, either by using manual *Start date* and *End date* inputs or by calculating it from the current project temporal layers. The project time range can then be used in the *Temporal controller panel* to manage the map canvas *temporal navigation*.

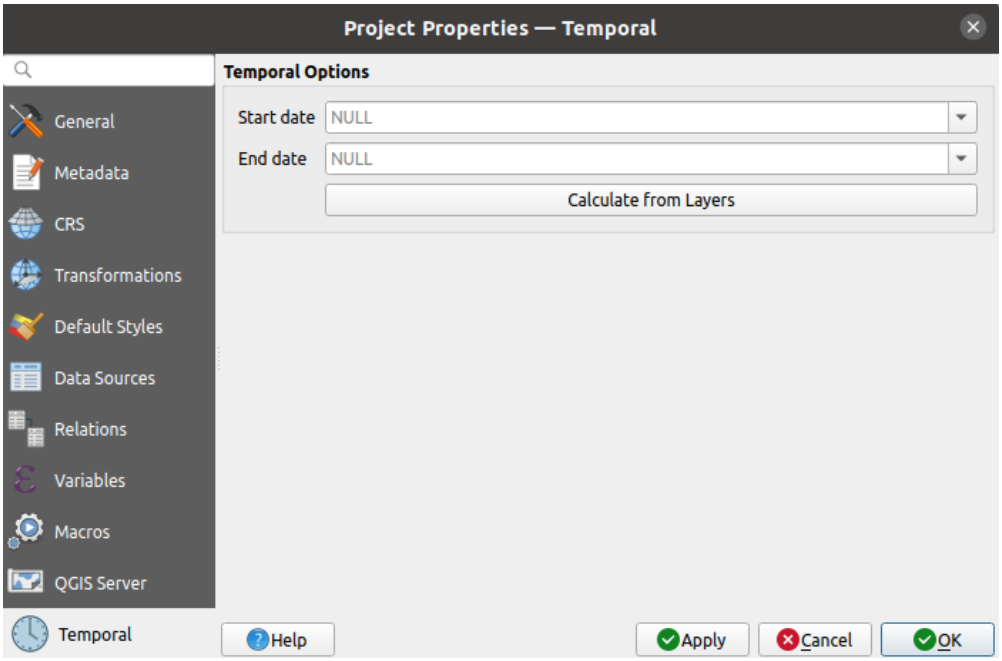


Fig. 5.40: Project Temporal tab

5.3.14 Elevation Properties

The *Elevation* tab helps you configure default settings for the terrain and elevation. When any new *3d map* is created in the project, the map will default to using the same terrain settings as are defined for the project. The project elevation settings will also be respected by the *elevation profile* tool.

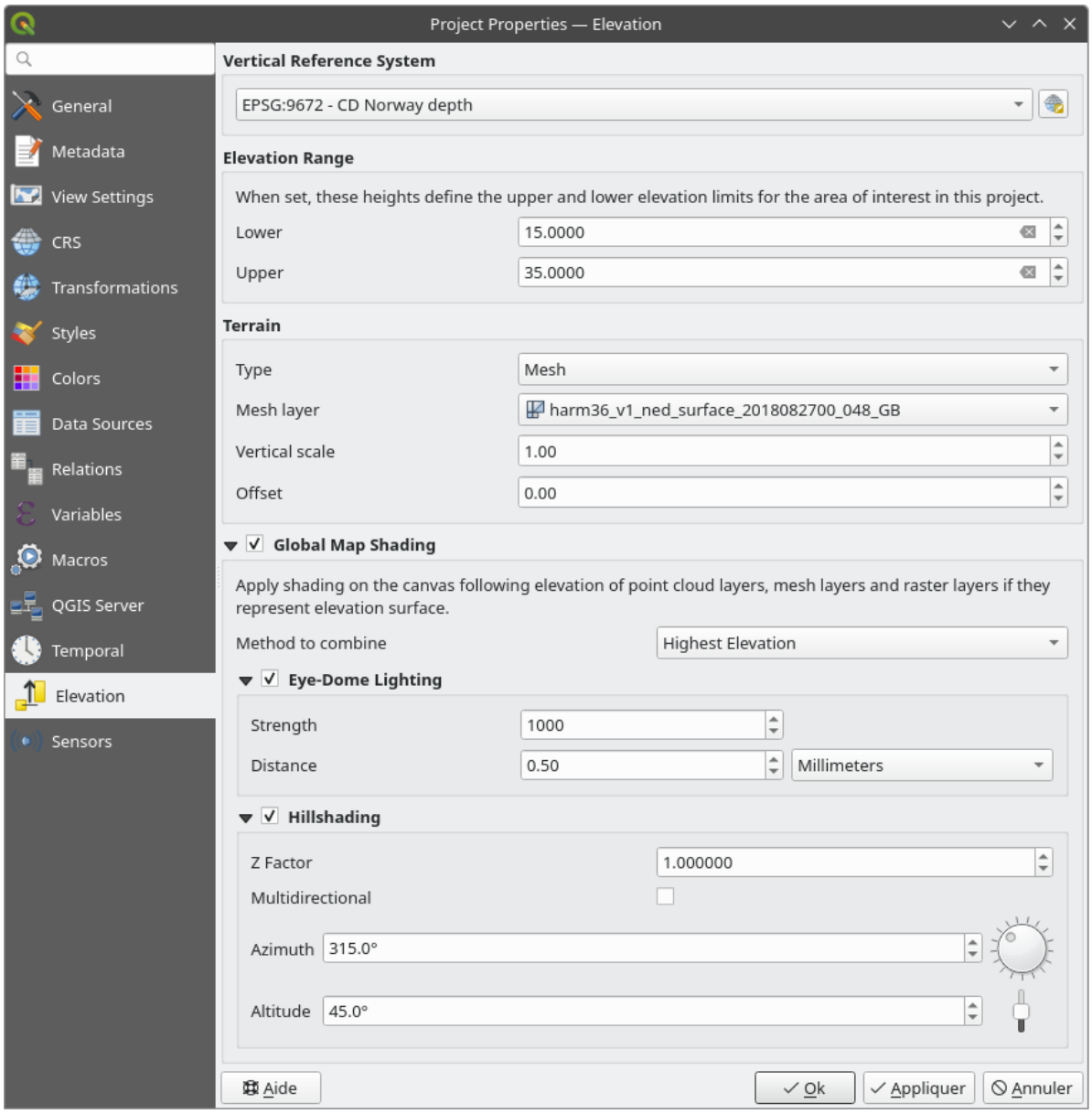


Fig. 5.41: Project Elevation tab

- *Vertical Reference System*: If the CRS of your project is compound one (including a Z dimension), then the vertical CRS used for the project will be the vertical component of the project CRS. If your project CRS is horizontal, then you can select a specific vertical CRS by clicking on the Select CRS.
- *Elevation Range*: helps you define the *Lower* and *Upper* limits for the *elevation controller* in the project.
- *Terrain* and elevation options are available for:
  - *Flat terrain* with *Terrain height* setting
  - *DEM (Raster Layer)*: with setting for defining the *Raster layer*, a *Vertical scale* factor to apply to band values and a vertical *Offset*
  - *Mesh*: with setting for defining the *Mesh layer*, a *Vertical scale* factor to apply to vertices Z value and a vertical *Offset*

  These settings can be overwritten from the 3D map *configuration dialog*.
- With *Global map shading* settings, you apply a global shading effect to the map, based on the elevation of

all the active layers that have elevation properties enabled, including:

- *raster layers*
- *mesh layers*
- *point cloud layers*

Method to combine: the elevation value at any particular position for all the relevant layers are combined, and the considered elevation is chosen depending on one of these methods:

- *Highest elevation* value
- *Based on layer's order*: the elevation on the topmost layer in the layer tree is considered.

Supported shading options are:

- *Eye-dome lighting*: applies shading effects to the map canvas for a better depth rendering. Following parameters can be controlled:
  - *Strength*: increases the contrast, allowing for better depth perception
  - *Distance*: represents the distance of the used pixels off the center pixel and has the effect of making edges thicker.
- *Hillshading*, shaping some reliefs on the map using shading (levels of gray):
  - *Z Factor*: Scaling factor for the elevation value
  - *Multidirectional*: Specify if multidirectional hillshading is to be used
  - *Azimuth*: The azimuth of the light source
  - *Altitude*: The elevation angle of the light source

Note

A shortcut to *Global map shading* properties is available through the *Layer Styling* panel.

5.3.15 Sensors Properties

The *Sensors* tab is used to configure sensors and toggle their connection status. When active, sensors will passively collect data in the background and make available their latest data to expressions and python scripts.

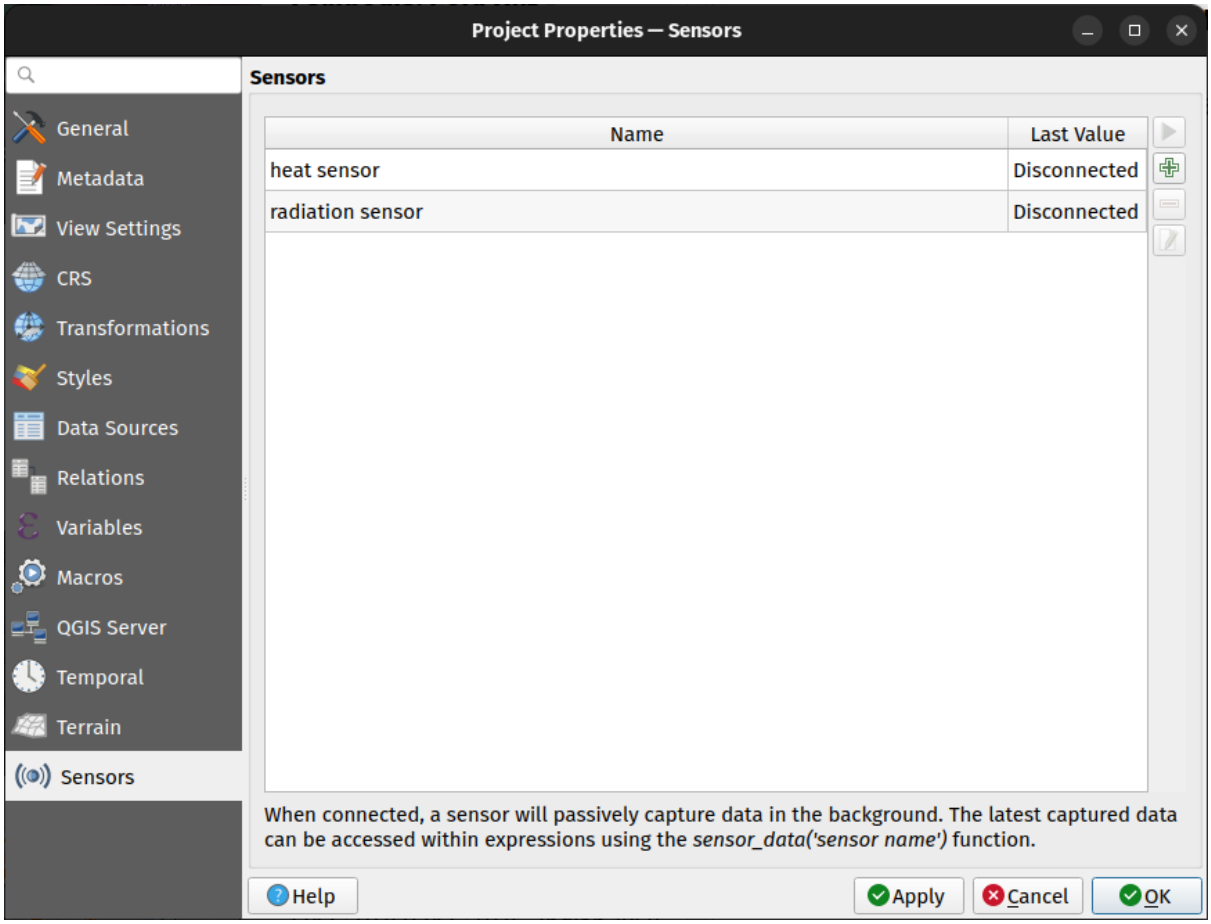


Fig. 5.42: Project Sensors tab

To add a new sensor, click on the button. A settings sub-panel will open and allow you to configure:

- the *Sensor name*: used to retrieve sensor values in expressions and python scripts;
- the *Sensor type*: TCP, UDP, serial port, etc.; and
- additional type-specific details (e.g., host name and port).

  For serial port sensors, comboboxes with the system's available serial ports and connection baud rate are available for their selection. A *Data frame delimiter* can be used to parse returned data. In case multiple data frames are found in the last chunk of data pushed by the serial port, all the data frames are returned. You can then use an expression to further split the returned sensor last data value into individual frames (e.g., using *string_to_array* function).

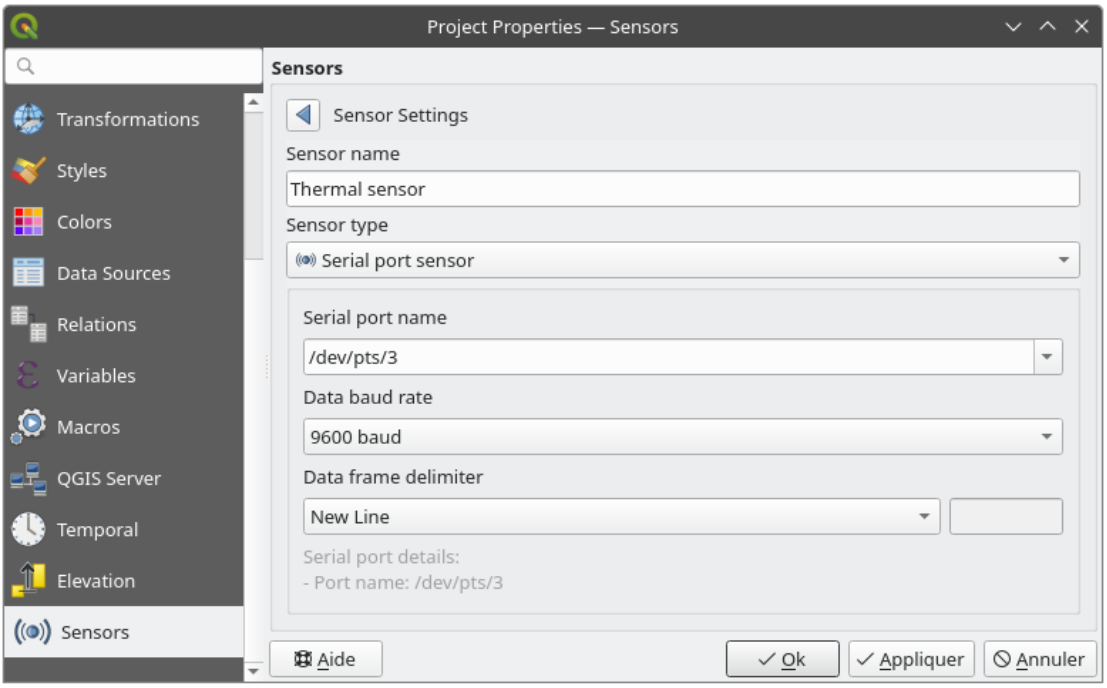


Fig. 5.43: Sensor Settings sub-panel

Once a sensor is configured, you can use the Start button to connect the sensor. Once active, the last collected data will be displayed in the sensors table's *Last value* column.

5.4 Customization

The *Customization* dialog lets you (de)activate almost every element in the QGIS user interface. This can be very useful if you want to provide your end-users with a 'light' version of QGIS, containing only the icons, menus or panels they need.

Note

Before your changes are applied, you need to restart QGIS.

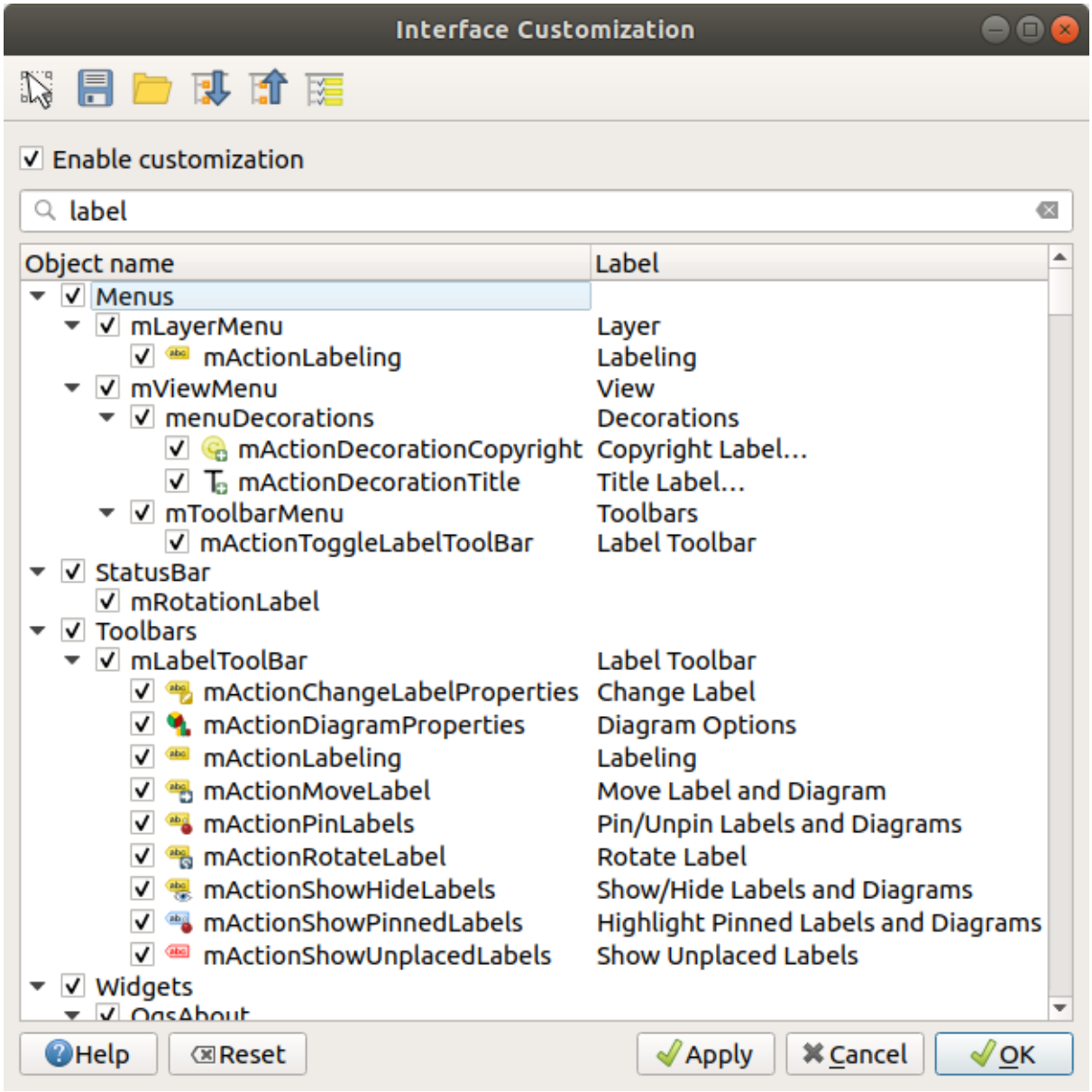


Fig. 5.44: The Customization dialog

Ticking the *Enable customization* checkbox is the first step on the way to QGIS customization. This enables the toolbar and the widget panel from which you can uncheck and thus disable some GUI items.

The configurable item can be:

- a **Menu** or some of its sub-menus from the *Menu Bar*
- a whole **Panel** (see *Panels and Toolbars*)
- the **Status bar** described in *Status Bar* or some of its items
- a **Toolbar**: the whole bar or some of its icons
- or any **widget** from any dialog in QGIS: label, button, combobox…

With Switch to catching widgets in main application, you can click on an item in QGIS interface that you want to be hidden and QGIS automatically unchecks the corresponding entry in the Customization dialog. You can also use the *Search* box to find items by their name or label.

Once you setup your configuration, click *Apply* or *OK* to validate your changes. This configuration becomes the one used by default by QGIS at the next startup.

The modifications can also be saved in a `.ini` file using  Save To File button. This is a handy way to share a common QGIS interface among multiple users. Just click on  Load from File from the destination computer in order to import the `.ini` file. You can also run *command line tools* and save various setups for different use cases as well.

Tip

Easily restore predefined QGIS

The initial QGIS GUI configuration can be restored by one of the methods below:

- unchecking  *Enable customization* option in the Customization dialog or click the  Check All button
- pressing the *Reset* button in the **Settings** frame under *Settings* ► *Options* menu, *System* tab
- launching QGIS at a command prompt with the following command line `qgis --nocustomization`
- setting to `false` the value of *UI* ► *Customization* ► *Enabled* variable under *Settings* ► *Options* menu, *Advanced* tab (see the *warning*).

In most cases, you need to restart QGIS in order to have the change applied.

5.5 Keyboard shortcuts

QGIS provides default keyboard shortcuts for many features. You can find them in section *Menu Bar*. Additionally, the menu option *Settings* ►  *Keyboard Shortcuts...* allows you to change the default keyboard shortcuts and add new ones to QGIS features.

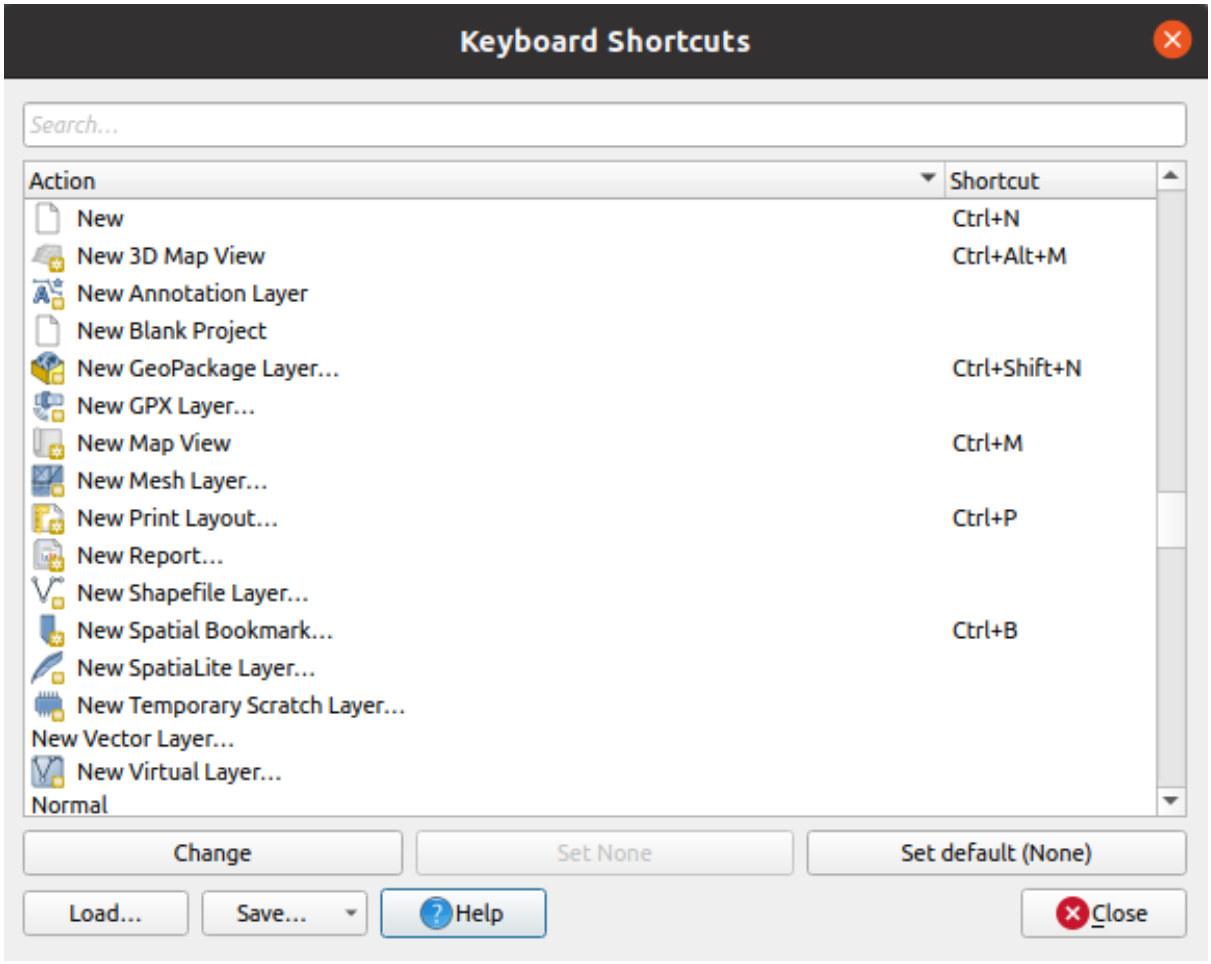


Fig. 5.45: Define shortcut options

Configuration is very simple. Use the search box at the top of the dialog to find a particular action, select it from the list and click on :

- *Change* and press the new combination you want to assign as new shortcut
- *Set None* to clear any assigned shortcut
- or *Set Default* to backup the shortcut to its original and default value.

Proceed as above for any other tools you wish to customize. Once you have finished your configuration, simply *Close* the dialog to have your changes applied. You can also *Save* the changes either as an `.XML` file with only the User Shortcuts or with all Shortcuts or as an `.PDF` file with all Shortcuts and *Load* them into another QGIS installation.

5.6 Running QGIS with advanced settings

5.6.1 Command line and environment variables

We've seen that *launching QGIS* is done as for any application on your OS. QGIS provides command line options for more advanced use cases (in some cases you can use an environment variable instead of the command line option). To get a list of the options, enter `qgis --help` on the command line, which returns:

```
QGIS is a user friendly Open Source Geographic Information System.
Usage: /usr/bin/qgis.bin [OPTION] [FILE]
  OPTION:
```

(continues on next page)

(continued from previous page)

```
        [-v, --version]     display version information and exit
        [-s, --snapshot filename]   emit snapshot of loaded datasets to given file
        [-w, --width width] width of snapshot to emit
        [-h, --height height]       height of snapshot to emit
        [-l, --lang language]       use language for interface text (changes␣
↪existing override)
        [-p, --project projectfile] load the given QGIS project
        [-e, --extent xmin,ymin,xmax,ymax]  set initial map extent
        [-n, --nologo]      hide splash screen
        [-V, --noversioncheck]      don't check for new version of QGIS at startup
        [-P, --noplugins]   don't restore plugins on startup
        [-B, --skipbadlayers]     don't prompt for missing layers
        [-C, --nocustomization]     don't apply GUI customization
        [-z, --customizationfile path]      use the given ini file as GUI␣
↪customization
        [-g, --globalsettingsfile path]     use the given ini file as Global␣
↪Settings (defaults)
        [-a, --authdbdirectory path] use the given directory for authentication␣
↪database
        [-f, --code path]   run the given python file on load
        [-F, --py-args arguments]   arguments for python. These arguments will be␣
↪available for each python execution via 'sys.argv' including the file specified␣
↪by '--code'. All arguments till '--' are passed to python and ignored by QGIS.
        [-d, --defaultui]   start by resetting user ui settings to default
        [--hide-browser]        hide the browser widget
        [--dxf-export filename.dxf]     emit dxf output of loaded datasets to␣
↪given file
        [--dxf-extent xmin,ymin,xmax,ymax]      set extent to export to dxf
        [--dxf-symbology-mode none|symbollayer|feature] symbology mode for dxf␣
↪output
        [--dxf-scale-denom scale]       scale for dxf output
        [--dxf-encoding encoding]       encoding to use for dxf output
        [--dxf-map-theme maptheme]      map theme to use for dxf output
        [--take-screenshots output_path]        take screen shots for the user␣
↪documentation
        [--screenshots-categories categories]   specify the categories of␣
↪screenshot to be used (see QgsAppScreenShots::Categories).
        [--profile name]        load a named profile from the user's profiles␣
↪folder.
        [-S, --profiles-path path]  path to store user profile folders. Will␣
↪create profiles inside a {path}\profiles folder
        [--version-migration]   force the settings migration from older version if␣
↪found
        [--openclprogramfolder]         path to the folder containing the sources␣
↪for OpenCL programs.
        [--help]                this text
        [--]            treat all following arguments as FILEs

 FILE:
   Files specified on the command line can include rasters,
   vectors, and QGIS project files (.qgs and .qgz):
    1. Rasters - supported formats include GeoTiff, DEM
       and others supported by GDAL
    2. Vectors - supported formats include ESRI Shapefiles
       and others supported by OGR and PostgreSQL layers using
       the PostGIS extension
```

Tip

Example Using command line arguments

You can start QGIS by specifying one or more data files on the command line. For example, assuming you are in the `qgis_sample_data` directory, you could start QGIS with a vector layer and a raster file set to load on startup using the following command: `qgis ./raster/landcover.img ./gml/lakes.gml`

--version

This option returns QGIS version information.

--snapshot

This option allows you to create a snapshot in PNG format from the current view. This comes in handy when you have many projects and want to generate snapshots from your data, or when you need to create snapshots of the same project with updated data.

Currently, it generates a PNG file with 800x600 pixels. The size can be adjusted using the `--width` and `--height` arguments. The filename can be added after `--snapshot`. For example:

```
qgis --snapshot my_image.png --width 1000 --height 600 --project my_project.qgs
```

--width

This option returns the width of the snapshot to be emitted (used with `--snapshot`).

--height

This option returns the height of the snapshot to be emitted (used with `--snapshot`).

--lang

Based on your locale, QGIS selects the correct localization. If you would like to change your language, you can specify a language code. For example, `qgis --lang it` starts QGIS in Italian localization.

--project

Starting QGIS with an existing project file is also possible. Just add the command line option `--project` followed by your project name and QGIS will open with all layers in the given file loaded.

--extent

To start with a specific map extent use this option. You need to add the bounding box of your extent in the following order separated by a comma:

```
--extent xmin,ymin,xmax,ymax
```

This option probably makes more sense when paired with the `--project` option to open a specific project at the desired extent.

--nologo

This option hides the splash screen when you start QGIS.

--noversioncheck

Skip searching for a new version of QGIS at startup.

--noplugins

If you have trouble at start-up with plugins, you can avoid loading them at start-up with this option. They will still be available from the Plugins Manager afterwards.

--nocustomization

Using this option, any existing *GUI customization* will not be applied at startup. This means that any hidden buttons, menu items, toolbars, and so on, will show up on QGIS start up. This is not a permanent change. The customization will be applied again if QGIS is launched without this option.

This option is useful for temporarily allowing access to tools that have been removed by customization.

--skipbadlayers

Using this option, you can avoid QGIS prompting the *Handle Unavailable Layers* dialog at startup. The project file is loaded, with missing layers kept as unavailable. More details on the topic at *Handling broken file paths*.

--customizationfile

Using this option, you can define a UI customization file, that will be used at startup.

--globalsettingsfile

The equivalent environment variable is `QGIS_GLOBAL_SETTINGS_FILE`.

Using this option, you can specify the path for a Global Settings file (`.ini`), also known as the Default Settings. The settings in the specified file replace the original inline default ones, but the user profiles' settings will be set on top of those.

QGIS looks for the default global settings file in the following order and only the first found file will be used:

- path specified by the commandline parameter
- path defined by the environment variable
- the AppDataLocation folder, where persistent application data can be stored; it is managed by the user or system administrator and is not touched by installer and does not require any additional setup like passing commandline parameters or settings environment variable. Depending on the OS, it is:
  - `$HOME/.local/share/QGIS/QGIS3/`
  - `C:\Users\<username>\%AppData%\Roaming\QGIS\QGIS3\`
  - `$HOME/Library/Application Support/QGIS/QGIS3/`
- the installation directory, i.e., `your_QGIS_package_path/resources/qgis_global_settings.ini`.

Presently, there's no way to specify a file to write settings to; therefore, you can create a copy of an original settings file, rename, and adapt it.

Setting the `qgis_global_setting.ini` file path to a network shared folder, allows a system administrator to change global settings and defaults in several machines by only editing one file.

--authdbdirectory

This option is similar to `--globalsettingsfile`, but defines the path to the directory where the authentication database will be stored and loaded.

--code

This option can be used to run a given python file directly after QGIS has started.

For example, when you have a python file named `load_alaska.py` with following content:

```
from qgis.utils import iface
raster_file = "/home/gisadmin/Documents/qgis_sample_data/raster/landcover.img"
layer_name = "Alaska"
iface.addRasterLayer(raster_file, layer_name)
```

Assuming you are in the directory where the file `load_alaska.py` is located, you can start QGIS, load the raster file `landcover.img` and give the layer the name 'Alaska' using the following command:

```
qgis --code load_alaska.py
```

--py-args

This option allows to pass arguments to the script provided via the *–code* parameter and more generally to each python execution. After `--py-args`, it is not possible to pass any other parameter than – to QGIS. All arguments after `--py-args` till `--` are passed over to the python interpreter and ignored by QGIS.

```
qgis --code /home/user/test.py --py-args --specialScriptArguments "a text arg"
↪'and another arg' -- layer1 layer2
```

In the above code, `test.py` will have this content in `sys.argv`: ['/home/user/test.py', '–specialScriptArguments', 'a text arg', 'and another arg']. `layer1` and `layer2` will be normally handled by QGIS as layers to load.

--defaultui

On load, **permanently resets** the user interface (UI) to the default settings. This option will restore the panels and toolbars visibility, position, and size. Unless it's changed again, the default UI settings will be used in the following sessions.

Notice that this option doesn't have any effect on *GUI customization*. Items hidden by GUI customization (e.g., the status bar) will remain hidden even using the `--defaultui` option. See also the `--nocustomization` option.

--hide-browser

On load, hides the *Browser* panel from the user interface. The panel can be enabled by right-clicking a space in the toolbars or using the *View* ► *Panels* (*Settings* ► *Panels* in  Linux KDE).

Unless it's enabled again, the Browser panel will remain hidden in the following sessions.

--dxf-*

These options can be used to export a QGIS project into a DXF file. Several options are available:

- *–dxf-export*: the DXF filename into which to export the layers;
- *–dxf-extent*: the extent of the final DXF file;
- *–dxf-symbology-mode*: several values can be used here: `none` (no symbology), `symbollayer` (Symbol layer symbology), `feature` (feature symbology);
- *–dxf-scale-denom*: the scale denominator of the symbology;
- *–dxf-encoding*: the file encoding;
- *–dxf-map-theme*: choose a *map theme* from the layer tree configuration.

--take-screenshots

Takes screenshots for the user documentation. Can be used together with `--screenshots-categories` to filter which categories/sections of the documentation screenshots should be created (see QgsAppScreenShots::Categories).

--profile

Loads QGIS using a specific profile from the user's profile folder. If the named profile doesn't exist, it will be created. This option takes precedence over *user profile startup setting*.

--profiles-path

With this option, you can choose a path to load and save the profiles (user settings). It creates profiles inside a `{path}\profiles` folder, which includes settings, installed plugins, processing models and scripts, and so on.

This option allows you to, for instance, carry all your plugins and settings in a flash drive, or, for example, share the settings between different computers using a file sharing service.

The equivalent environment variable is `QGIS_CUSTOM_CONFIG_PATH`.

--version-migration

If settings from an older version are found (*e.g.*, the `.qgis2` folder from QGIS 2.18), this option will import them into the default QGIS profile.

--openclprogramfolder

Using this option, you can specify an alternative path for your OpenCL programs. This is useful for developers while testing new versions of the programs without needing to replace the existing ones.

The equivalent environment variable is `QGIS_OPENCL_PROGRAM_FOLDER`.

--

This option allows to pass following arguments as files to open in QGIS. They can be supported rasters or vectors files, QGIS layer definition files or project files.

```
qgis -- /home/user/project.qgs /home/user/path_to_data/layer1.gpkg
```

The above code will start QGIS, open the `project` file and load `layer1` in it.

5.6.2 Deploying QGIS within an organization

If you need to deploy QGIS within an organization with a custom configuration file, first you need to copy/paste the content of the default settings file located in `your_QGIS_package_path/resources/qgis_global_settings.ini`. This file already contains some default sections identified by a block starting with `[]`. We recommend that you keep these defaults values and add your own sections at the bottom of the file. If a section is duplicated in the file, QGIS will take the last one from top to bottom.

You can change `allowVersionCheck=false` to disable the QGIS version check.

If you do not want to display the migration window after a fresh install, you need the following section:

```
[migration]
fileVersion=2
settings=true
```

If you want to add a custom variable in the global scope:

```
[variables]
organisation="Your organization"
```

A lot of settings can be set using the `.INI` file such as WMS/WMTS, PostgreSQL connections, proxy settings, maptips, default values for algorithms parameters… For example, if you want to configure organization wide default values for some algorithms it would be handy if you could configure it in the prepared ini-file for all users. If the users then open the toolbox to execute an algorithm, the default values will be there. To set the default value, e.g., for `GRID_SIZE` parameter for the `native:intersection` algorithm, you could write the following into the `.INI` file:

```
[Processing]
DefaultGuiParam\native%3Aintersection\GRID_SIZE=0.01
```

Not all settings can be prepared from QGIS GUI, but to discover the possibilities of the settings `.INI` file, we suggest that you set the config you would like in QGIS Desktop and then search for it in your `.INI` file located in your profile using a text editor.

Finally, you need to set the environment variable `QGIS_GLOBAL_SETTINGS_FILE` to the path of your customized file.

In addition, you can also deploy files such as Python macros, color palettes, layout templates, project templates… either in the QGIS system directory or in the QGIS user profile.

- Layout templates must be deployed in the `composer_templates` directory.
- Project templates must be deployed in the `project_templates` directory.
- Custom Python macros must be deployed in the `python` directory.

CHAPTER

SIX

WORKING WITH PROJECTIONS

A Coordinate Reference System, or CRS, is a method of associating numerical coordinates with a position on the surface of the Earth. QGIS has support for approximately 7,000 standard CRSs, each with different use cases, pros and cons! Choosing an appropriate reference system for your QGIS projects and data can be a complex task, but fortunately QGIS helps guide you through this choice, and makes working with different CRSs as transparent and accurate as possible.

6.1 Overview of Projection Support

QGIS has support for approximately 7,000 known CRSs. These standard CRSs are based on those defined by the European Petroleum Search Group (EPSG) and the Institut Geographique National de France (IGNF), and are made available in QGIS through the underlying "Proj" projection library. Commonly, these standard projections are identified through use of an authority:code combination, where the authority is an organisation name such as "EPSG" or "IGNF", and the code is a unique number associated with a specific CRS. For instance, the common WGS 84 latitude/longitude CRS is known by the identifier `EPSG:4326`, and the web mapping standard CRS is `EPSG:3857`.

Custom, user-created CRSs are stored in a user CRS database. See section *Custom Coordinate Reference System* for information on managing your custom coordinate reference systems.

6.2 Layer Coordinate Reference Systems

In order to correctly project data into a specific target CRS, either your data must contain information about its coordinate reference system or you will need to manually assign the correct CRS to the layer. For PostgreSQL layers, QGIS uses the spatial reference identifier that was specified when that PostgreSQL layer was created. For data supported by GDAL, QGIS relies on the presence of a recognized means of specifying the CRS. For instance, for the Shapefile format this is a file containing an ESRI Well-Known Text (WKT) representation of the layer's CRS. This projection file has the same base name as the `.shp` file and a `.prj` extension. For example, `alaska.shp` would have a corresponding projection file named `alaska.prj`.

Whenever a layer is loaded into QGIS, QGIS attempts to automatically determine the correct CRS for that layer. In some cases this is not possible, e.g. when a layer has been provided without retaining this information. You can configure QGIS behavior whenever it cannot automatically determine the correct CRS for a layer:

1. Open *Settings* ► *Options…* ► *CRS*

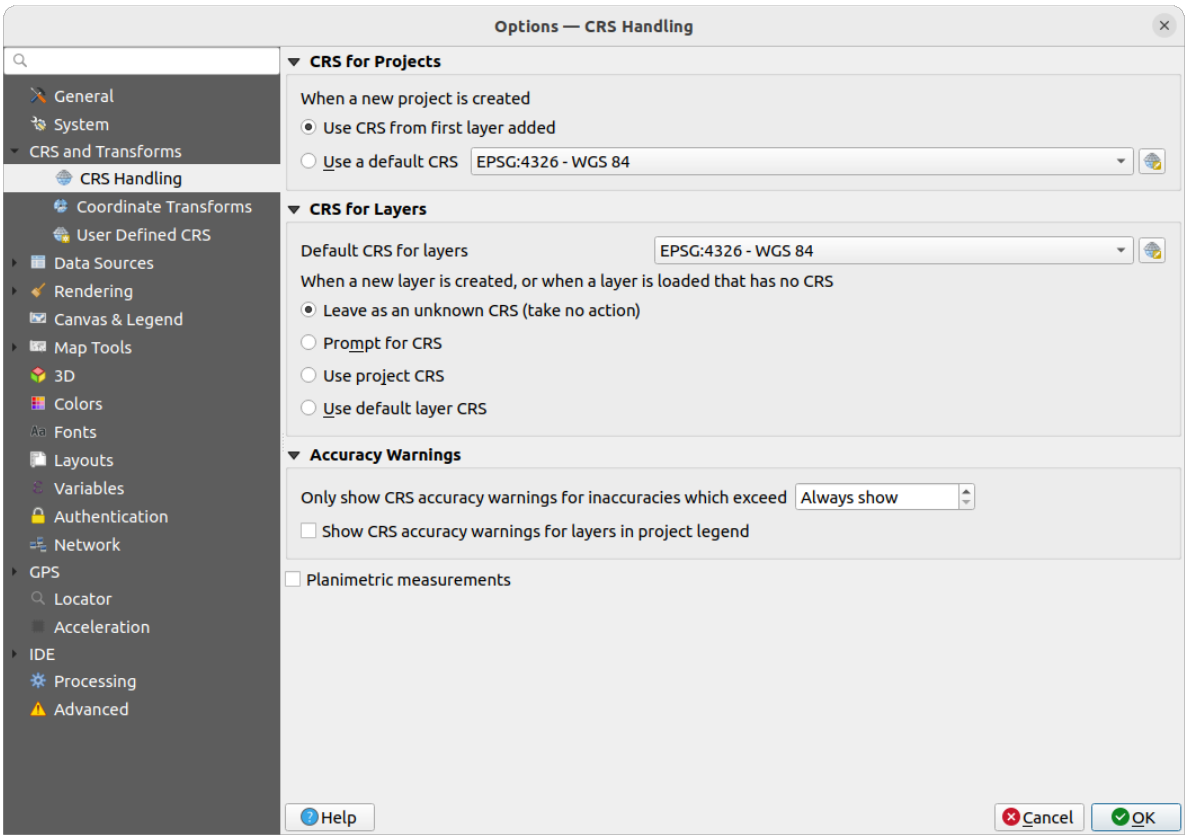


Fig. 6.1: The CRS tab in the QGIS Options Dialog

2. Under the *CRS for layers* group, set the action to do *when a new layer is created, or when a layer is loaded that has no CRS*. One of:

   - *Leave as unknown CRS (take no action)*: there will be no prompt to select a CRS when a layer without CRS is loaded, defering CRS choice to a later time. Convenient when loading a lot of layers at once. Such layers will be identifiable in the *Layers* panel by the icon next to them. They'll also be un-referenced, with coordinates from the layer treated as purely numerical, non-earth values, i.e. the same behavior as all layers get when *a project is set to have no CRS*.
   - *Prompt for CRS*: it will prompt you to manually select the CRS. Selecting the correct choice is crucial, as a wrong choice will place your layer in the wrong position on the Earth's surface! Sometimes, accompanying metadata will describe the correct CRS for a layer, in other cases you will need to contact the original author of the data to determine the correct CRS to use.
   - *Use project CRS*
   - *Use default layer CRS*, as set in the *Default CRS for layers* combobox above.

Tip

To assign the same CRS to multiple layers that have no crs or have a wrong one in one operation:

1. Select the layers in the *Layers* panel
2. Press `Ctrl+Shift+C`. You could also right-click over one of the selected layers or go to *Layer* ► *Set CRS of layer(s)*
3. Find and select the right CRS to use

4. And press *OK*. You can confirm that it has been set correctly in the *Source* tab of the layers' properties dialog.

Note that changing the CRS in this setting does not alter the underlying data source in any way, rather it just changes how QGIS interprets the raw coordinates from the layer in the current QGIS project.

6.3 Project Coordinate Reference Systems

Every project in QGIS also has an associated Coordinate Reference System. The project CRS determines how data is projected from its underlying raw coordinates to the flat map rendered within your QGIS map canvas.

QGIS supports "on the fly" CRS transformation for both raster and vector data. This means that regardless of the underlying CRS of particular map layers in your project, they will always be automatically transformed into the common CRS defined for your project. Behind the scenes, QGIS transparently reprojects all layers contained within your project into the project's CRS, so that they will all be rendered in the correct position with respect to each other!

It is important to make an appropriate choice of CRS for your QGIS projects. Choosing an inappropriate CRS can cause your maps to look distorted, and poorly reflect the real-world relative sizes and positions of features. Usually, while working in smaller geographic areas, there will be a number of standard CRSs used within a particular country or administrative area. It's important to research which CRSs are appropriate or standard choices for the area you are mapping, and ensure that your QGIS project follows these standards.

By default, QGIS starts each new project using a global default projection. This default CRS is `EPSG:4326` (also known as "WGS 84"), and it is a global latitude/longitude based reference system. This default CRS can be changed via the *CRS for New Projects* setting in the *CRS* tab under *Settings* ► *Options...* (see Fig. 6.1). There is an option to automatically set the project's CRS to match the CRS of the first layer loaded into a new project, or alternatively you can select a different default CRS to use for all newly created projects. This choice will be saved for use in subsequent QGIS sessions.

The project CRS can also be set through the *CRS* tab of the *Project* ► *Properties...* dialog. It will also be shown in the lower-right of the QGIS status bar.

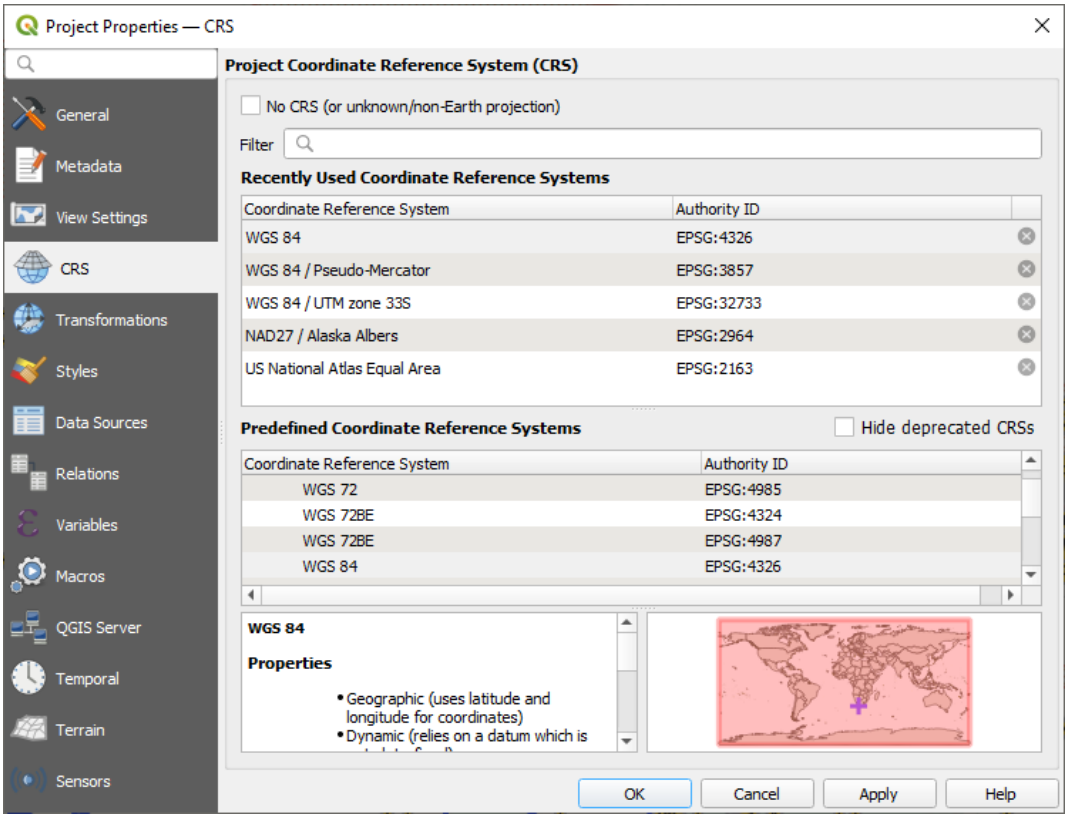


Fig. 6.2: Project Properties Dialog

Available options are:

- *No CRS (or unknown/non-Earth projection)*: Checking this setting will disable ALL projection handling within the QGIS project, causing all layers and map coordinates to be treated as simple 2D Cartesian coordinates, with no relation to positions on the Earth's surface. It can be used to guess a layer CRS (based on its raw coordinates or when using QGIS for non earth uses like role-playing game maps, building mapping or microscopic stuff. In this case:
  - No reprojection is done while rendering the layers: features are just drawn using their raw coordinates.
  - The ellipsoid is locked out and forced to `None/Planimetric`.
  - The distance and area units, and the coordinate display are locked out and forced to "unknown units"; all measurements are done in unknown map units, and no conversion is possible.
- or an existing coordinate reference system that can be *geographic*, *projected* or *user-defined*. A preview of the CRS extent on earth is displayed to help you select the appropriate one. Layers added to the project are translated on-the-fly to this CRS in order to overlay them regardless of their original CRS. Use of units and ellipsoid setting are available and make sense and you can perform calculations accordingly.

If you change your project's CRS and want the units (for distance and area) to match the units defined by the CRS, make sure you have selected *Map units* in the relevant settings, in the *General* tab of the *Project* ► *Properties*... dialog.

Tip

Setting the project CRS from a layer

You can assign a CRS to the project using a layer CRS:

1. In the *Layers* panel, right-click on the layer you want to pick the CRS
2. Select *Set project CRS from Layer*.

The project's CRS is redefined using the layer's CRS. Map canvas extent, coordinates display are updated accordingly and all the layers in the project are on-the-fly translated to the new project CRS.

6.4 Coordinate Reference System Selector

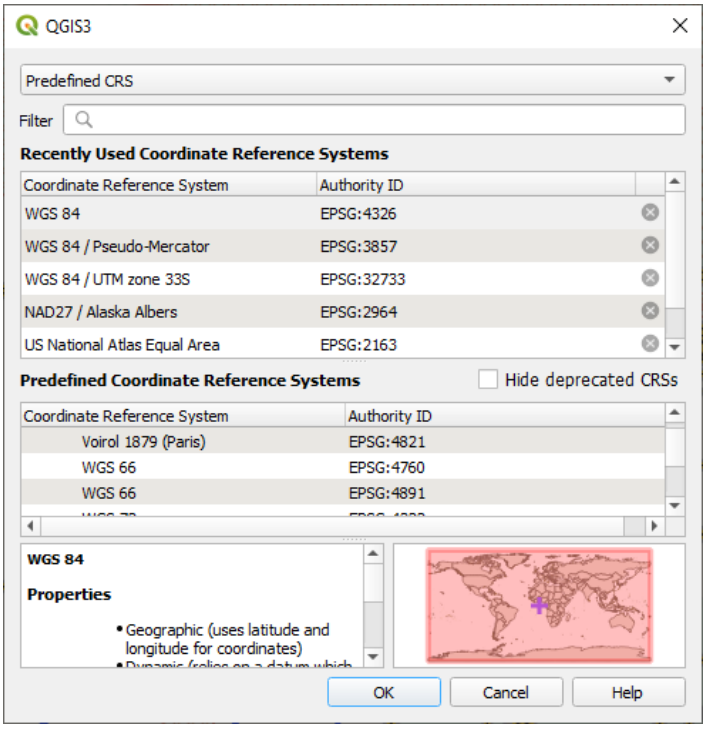


Fig. 6.3: CRS Selector

This dialog helps you assign a Coordinate Reference System to a project or a layer, provided a set of projection databases. Items in the dialog are:

- **Filter**: If you know the EPSG code, the identifier, or the name for a Coordinate Reference System, you can use the search feature to find it. Enter the EPSG code, the identifier or the name.
- **Recently used coordinate reference systems**: If you have certain CRSs that you frequently use in your everyday GIS work, these will be displayed in this list. Click on one of these items to select the associated CRS.
- **Coordinate reference systems of the world**: This is a list of all CRSs supported by QGIS, including Geographic, Projected and Custom coordinate reference systems. To define a CRS, select it from the list by expanding the appropriate node and selecting the CRS. The active CRS is preselected.
- **PROJ text**: This is the CRS string used by the PROJ projection engine. This text is read-only and provided for informational purposes.

If you want to remove a recently used CRS from the list that can be done in several ways:

- Press Remove from recently used CRS button on the right side.
- Press `Del` after selecting CRS that you want to remove.
- Right-click on used CRS and choose *Remove selected CRS from recently used CRS* to remove selected CRS or *Clear all recently used CRS* to remove all used CRS.

The CRS selector also shows a rough preview of the geographic area for which a selected CRS is valid for use. Many CRSs are designed only for use in small geographic areas, and you should not use these outside of the area they were designed for. The preview map shades an approximate area of use whenever a CRS is selected from the list. In addition, this preview map also shows an indicator of the current main canvas map extent.

6.5 Custom Coordinate Reference System

If QGIS does not provide the coordinate reference system you need, you can define a custom CRS. To define a CRS, select *Custom CRS…* from the *Settings* menu. Custom CRSs are stored in your QGIS user database. In addition to your custom CRSs, this database also contains your spatial bookmarks and other custom data.

Defining a custom CRS in QGIS requires a good understanding of the PROJ projection library. To begin, refer to "Cartographic Projection Procedures for the UNIX Environment - A User's Manual" by Gerald I. Evenden, U.S. Geological Survey Open-File Report 90-284, 1990 (available at https://pubs.usgs.gov/of/1990/of90-284/ofr90-284.pdf).

This manual describes the use of `proj` and related command line utilities. The cartographic parameters used with `proj` are described in the user manual and are the same as those used by QGIS.

The *Custom Coordinate Reference System Definition* dialog requires only two parameters to define a user CRS:

1. A descriptive name
2. The cartographic parameters in PROJ or WKT format

To create a new CRS:

1. Click the Add new CRS button
2. Enter a descriptive name
3. Select the format: it can be *Proj String* or *WKT*
4. Add the CRS *Parameters*.

> **Note**
>
> **Prefer storing the CRS definition in WKT format**
>
> Although both `Proj String` and `WKT` formats are supported, it's highly recommended to store projection definitions in the WKT format. Therefore, if the available definition is in the proj format, select that format, enter the parameters and then switch to WKT format. QGIS will convert the definition to the WKT format that you can later save.

5. Click *Validate* to test whether the CRS definition is an acceptable projection definition.

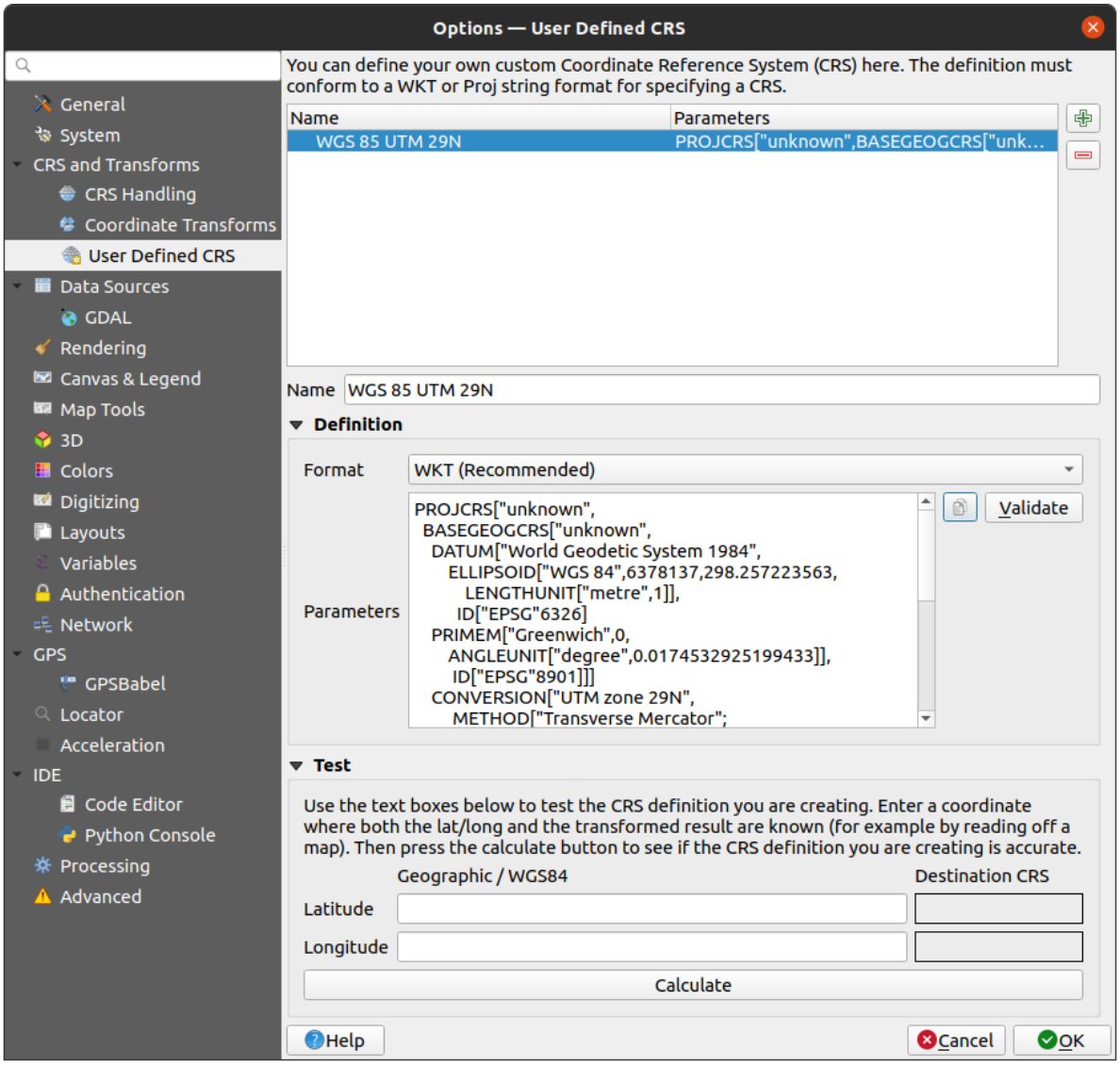


Fig. 6.4: Custom CRS Dialog

You can test your CRS parameters to see if they give sane results. To do this, enter known WGS 84 latitude and longitude values in *North* and *East* fields, respectively. Click on *Calculate*, and compare the results with the known values in your coordinate reference system.

6.5.1 Integrate an NTv2-transformation in QGIS

To integrate an NTv2 transformation file in QGIS you need one more step:

1. Place the NTv2 file (.gsb) in the CRS/Proj folder that QGIS uses (e.g. `C:\OSGeo4W64\share\proj` for windows users)
2. Add **nadgrids** (`+nadgrids=nameofthefile.gsb`) to the Proj definition in the *Parameters* field of the *Custom Coordinate Reference System Definition* (*Settings* ► *Custom Projections…*).

Fig. 6.5: Setting an NTv2 transformation

6.6 Datum Transformations

In QGIS, 'on-the-fly' CRS transformation is enabled by default, meaning that whenever you use layers with different coordinate systems QGIS transparently reprojects them to the project CRS. For some CRS, there are a number of possible transforms available to reproject to the project's CRS!

By default, QGIS will attempt to use the most accurate transformation available. However, in some cases this may not be possible, e.g. whenever additional support files are required to use a transformation. Whenever a more accurate transformation is available, but is not currently usable, QGIS will show an informative warning message advising you of the more accurate transformation and how to enable it on your system. Usually, this requires download of an external package of transformation support files, and extracting these to the `proj` folder under your QGIS *user profile* folder.

If desired, QGIS can also prompt you whenever multiple possible transformations can be made between two CRSs, and allow you to make an informed selection of which is the most appropriate transformation to use for your data.

This customization is done in the *Settings* ► *Options* ► *Transformations* tab menu under the *Default datum transformations* group:

- using *Ask for datum transformation if several are available*: when more than one appropriate datum transformation exist for a source/destination CRS combination, a dialog will automatically be opened prompting users to choose which of these datum transformations to use for the project. If the *Make default* checkbox is ticked when selecting a transformation from this dialog, then the choice is remembered and automatically applied to any newly created QGIS projects.
- or defining a list of appropriate datum transformations to use as defaults when loading a layer to a project or reprojecting a layer.

  Use the button to open the *Select Datum Transformations* dialog. Then:

  1. Choose the *Source CRS* of the layer, using the drop-down menu or the Select CRS widget.
  2. Provide the *Destination CRS* in the same way.
  3. A list of available transformations from source to destination will be shown in the table. Clicking a row shows details on the settings applied and the corresponding accuracy and area of use of the transformation.

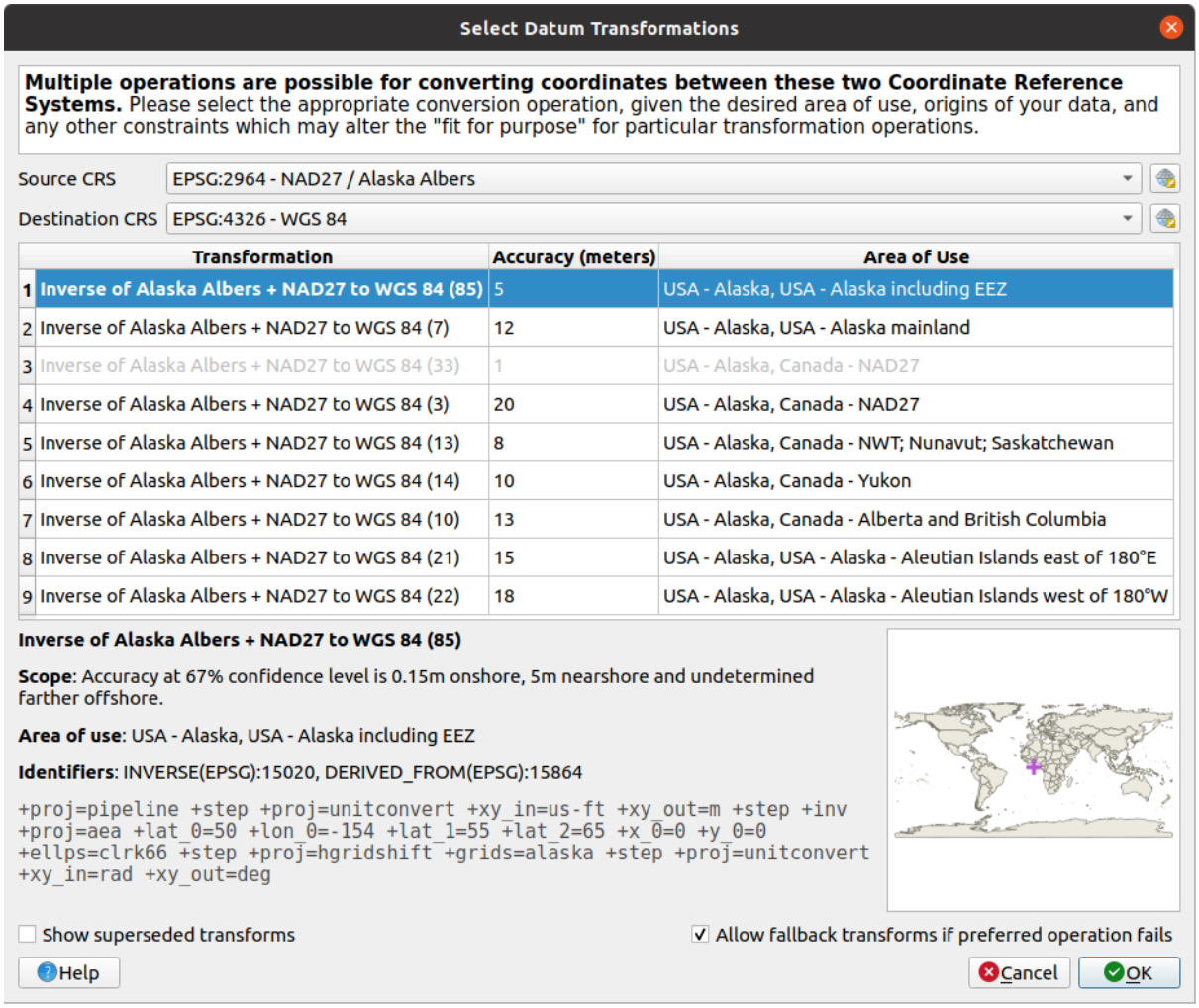


Fig. 6.6: Selecting a preferred default datum transformation

In some cases a transformation may not be available for use on your system. In this case, the transformation will still be shown (greyed) in this list but can not be picked until you install the required package of transformation support. Usually, a button is provided to download and install the corresponding grid, which is then stored under the `proj` folder in the active *user profile* directory.

4. Find your preferred transformation and select it
5. Set whether you *Allow fallback transforms if preferred operation fails*
6. Click *OK*.

   A row is added to the table under *Default Datum Transformations* with information about the *Source CRS*, the *Destination CRS*, the *Operation* applied for the transformation and whether *Allow fallback Transforms* is enabled.

From now, QGIS automatically uses the selected datum transformations for further transformation between these two CRSs until you remove it ( ) from the list or change the entry ( ) in the list.

Datum transformations set in the *Settings* ► *Options* ► *Transformations* tab will be inherited by all new QGIS projects created on the system. Additionally, a particular project may have its own specific set of transformations specified via the *CRS* tab of the *Project properties* dialog (*Project* ► *Properties…*). These settings apply to the current project only.

CHAPTER

SEVEN

VISUALIZING MAPS

Creating maps is one of the “business ends” of QGIS. Map views are canvases to visualize spatial layers loaded in the project, with the various rendering settings applied to them. They can be of *2D* or *3D* type or plotted along a *profile line*, show different scale or extent, or display a different set of the loaded layers thanks to *map themes*.

7.1 2D Map View

The 2D map view (also called **Map canvas**) is the central place where maps are displayed. QGIS opens by default with a single map view (called `main map`), showing layers in 2D, and tightly bound to the *Layers* panel. That window reflects the rendering (symbology, labeling, visibilities…) you applied to the loaded layers.

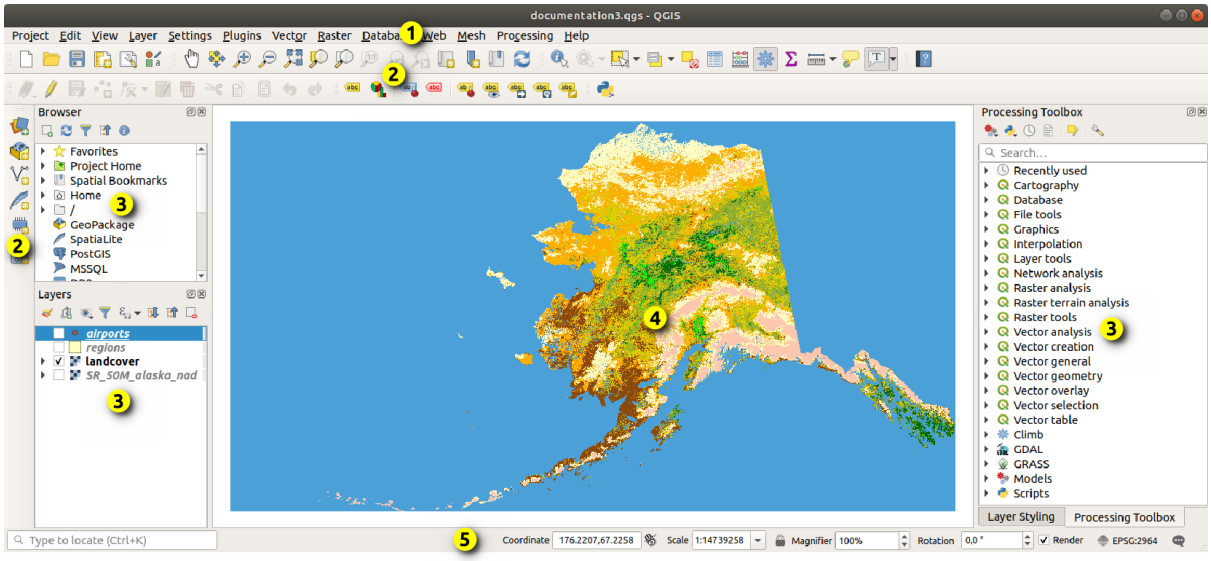


Fig. 7.1: QGIS GUI

7.1.1 Exploring the map view

When you add a layer (see e.g. *Opening Data*), QGIS automatically looks for its CRS. If a different CRS is set by default for the project (see *Project Coordinate Reference Systems*) then the layer extent is "on-the-fly" translated to that CRS, and the map view is zoomed to that extent if you start with a blank QGIS project. If there are already layers in the project, no map canvas resize is performed, so only features falling within the current map canvas extent will be visible.

Click on the map view and you should be able to interact with it, panning or zooming to different areas of the map. Dedicated tools are provided in the *Navigation Toolbar* and in the *View* menu, with handful shortcuts from the keyboard or the mouse buttons.

Table 7.1: Map canvas navigation tools

Tool	Usage
 Pan Map	• Single left click: the map is centered on the clicked point, at the same scale • Hold down the left mouse button and drag the map canvas.
 Zoom In	• Single left click: the map is centered on the clicked point, while the scale gets doubled • Drag a rectangle on the map canvas with the left mouse button to zoom in to an area. • Hold the `Alt` key to switch to the  Zoom Out tool.
 Zoom Out	• Single left click: the map is centered on the clicked point, while the scale gets halved • Drag a rectangle on the map canvas with the left mouse button to zoom out from an area. • Hold the `Alt` key to switch to the  Zoom In tool.
 Pan Map to Selection	Pan the map to the selected features of all the selected layers in the *Layers* panel.
 Zoom To Selection	Zoom to the selected features of all the selected layers in the *Layers* panel. *Also available in the layer contextual menu*
 Zoom To Layer(s)	Zoom to the extent of all the selected layers in the *Layers* panel. *Also available in the layer contextual menu*
 Zoom Full	Zoom to the extent of all the layers in the project or to the *project full extent*.
 Zoom Last	Zoom the map to the previous extent in history.
 Zoom Next	Zoom the map to the next extent in history.
 Zoom to Native Resolution	Zoom the map to a level where one pixel of the active raster layer covers one screen pixel. *Also available in the layer contextual menu*
Mouse wheel	• Pan map: Hold and drag the mouse wheel. • Zoom: Roll the mouse wheel to zoom in or zoom out. With `Ctrl` key pressed while rolling the mouse wheel results in a finer zoom. • Press the back or forward button to browse the map canvas zoom history.

continues on next page

Table 7.1 – continued from previous page

Tool	Usage
Keyboard	• Pan map: Hold down the `Space` key and move the mouse. Press the arrow keys to pan up, down, left and right. • Zoom in: Press `PgUp` or `Ctrl++` • Zoom out: Press `PgDown` or `Ctrl+-` • Zoom to area: When certain map tools are active (Identify, Measure…), hold down `Shift` and drag a rectangle on the map to zoom to that area. Not compatible with active selection or edit tools.

Right-click over the map and you should be able to  *Copy coordinates* of the clicked point in the map CRS, in WGS84 or in a custom CRS. The copied information can then be pasted in an expression, a script, text editor or spreadsheet…

7.1.2 Controlling map rendering

By default, QGIS renders all visible layers whenever the map canvas is refreshed. The events that trigger a refresh of the map canvas include:

- changing the visibility of a layer
- modifying symbology of a visible layer
- adding a layer
- panning or zooming
- resizing the QGIS window

QGIS allows you to control the rendering process in a number of ways.

- at the *global level*
- per layer, using e.g. the *scale dependent rendering*
- or with dedicated tools in the GUI.

To stop the map drawing, press the `Esc` key. This will halt the refresh of the map canvas and leave the map partially drawn. It may however take a bit of time after pressing `Esc` for the map drawing to halt.

To suspend rendering, click the  *Render* checkbox in the bottom-right corner of the status bar. When *Render* is unchecked, QGIS does not redraw the canvas in response to any of the usual triggers mentioned earlier. Examples of when you might want to suspend rendering include:

- adding many layers and symbolizing them prior to drawing
- adding one or more large layers and setting scale dependency before drawing
- adding one or more large layers and zooming to a specific view before drawing
- any combination of the above

Checking the  *Render* checkbox enables rendering and causes an immediate refresh of the map canvas.

7.1.3 Time-based control on the map canvas

QGIS can handle temporal control on loaded layers, i.e. modify the map canvas rendering based on a time variation. To achieve this, you need:

1. Layers that have dynamic temporal properties set. QGIS supports temporal control for different data providers, with custom settings. It's mainly about setting the time range in which the layer would display:
   - *raster layers*: controls whether to display or not the layer.
   - *WMTS layers*: controls whether the data should be rendered based on a static time range or following a dynamic temporal range
   - *vector layers*: features are filtered based on time values associated to their attributes
   - *mesh layers*: displays dynamically the active dataset groups values

   When dynamic temporal options are enabled for a layer, an icon is displayed next to the layer in the *Layers* panel to remind you that the layer is temporally controlled. Click the icon to update the temporal settings.
2. Enable the temporal navigation of the map canvas using the *Temporal controller panel*. The panel is activated:
   - using the Temporal controller panel icon located in the *Map Navigation* toolbar
   - or from the *View* ► *Panels* ► *Temporal controller panel* menu
   - or from the *View* ► *Data Filtering* ► *Temporal controller panel* menu

The temporal controller panel

The *Temporal controller* panel has the following modes:

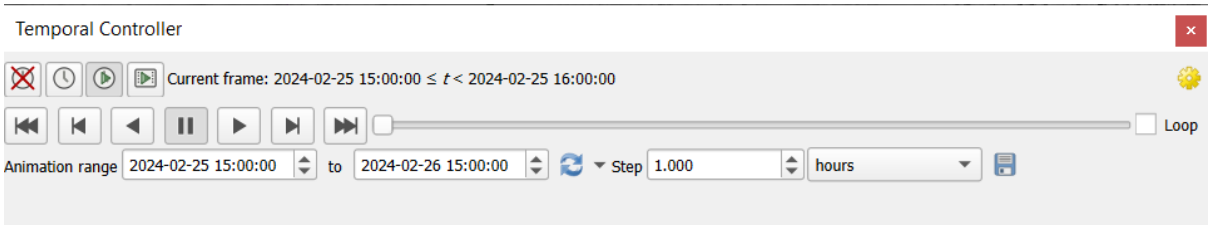


Fig. 7.2: Temporal Controller Panel in navigation mode

- Turn off temporal navigation: all the temporal settings are disabled and visible layers are rendered as usual
- Fixed range temporal navigation: a time range is set and only layers (or features) whose temporal range overlaps with this range are displayed on the map.
- Animated temporal navigation: a time range is set, split into steps, and only layers (or features) whose temporal range overlaps with each frame are displayed on the map
- Animated movie: a total number of frames is set for the animation, with the map settings advancing at each step, displaying the progression frame by frame without any time-based filtering of data.
- Settings for general control of the animation
  - *Frames rate*: number of steps that are shown per second
  - *Cumulative range*: all animation frames will have the same start date-time but different end dates and times. This is useful if you wish to accumulate data in your temporal visualization instead of showing a 'moving time window' across your data.

Animating a temporal navigation

An animation is based on a varying set of visible layers at particular times within a time range. To create a temporal animation:

1. Toggle on the Animated temporal navigation, displaying the animation player widget
2. Enter the *Time range* to consider. Using the button, this can be defined as:
   - *Set to full range* of all the time enabled layers
   - *Set to preset project range* as defined in the *project properties*
   - *Set to single layer's range* taken from a time-enabled layer
3. Fill in the time *Step* to split the time range. Different units are supported, from `seconds` to `centuries`. A `source timestamps` option is also available as step: when selected, this causes the temporal navigation to step between all available time ranges from layers in the project. It's useful when a project contains layers with non-contiguous available times, such as a WMS-T service which provides images that are available at irregular dates. This option will allow you to only step between time ranges where the next available image is shown.
4. Click the button to preview the animation. QGIS will generate scenes using the layers rendering at the set times. Layers display depends on whether they overlap any individual time frame.

   The animation can also be previewed by moving the time slider. Checking the *Loop* checkbox will repeatedly run the animation while clicking stops a running animation. A full set of video player buttons is available.

   Horizontal scrolling using the mouse wheel (where supported) with the cursor on the map canvas will also allow you to navigate, or "scrub", the temporal navigation slider backwards and forwards.
5. Click the Export animation button if you want to generate a series of images representing the scene. They can be later combined in a video editor software:

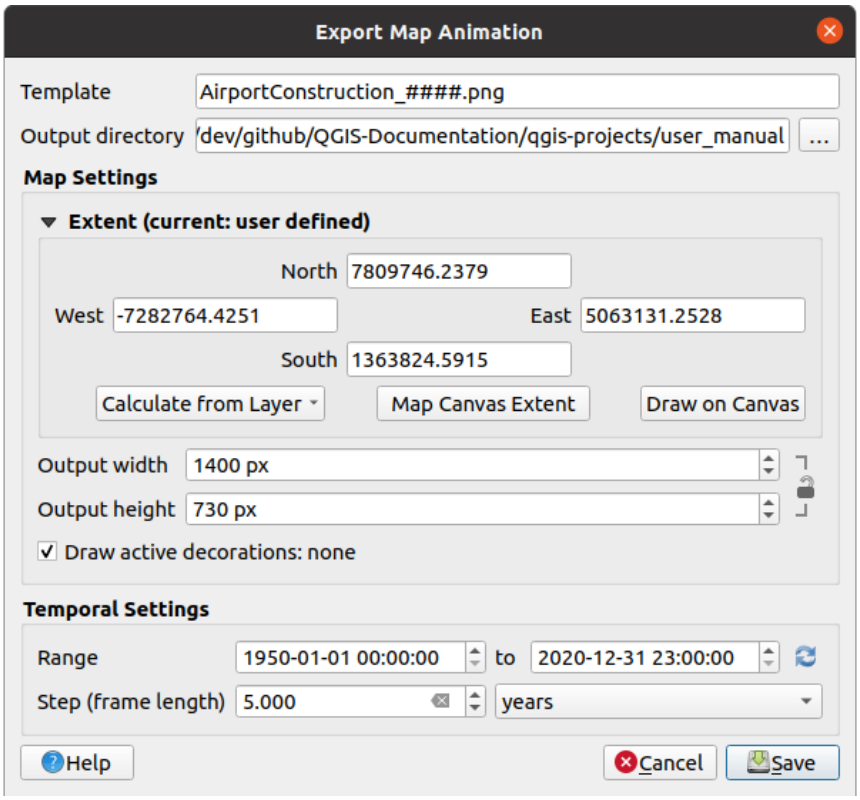


Fig. 7.3: Exporting map canvas animation scenes to images

- The filename *Template*: the `####` are replaced with frame sequence number
- The *Output directory*
- Under *Map settings*, you can:
  - redefine the *spatial extent* to use
  - control the *Resolution* of the image (*Output width* and *Output height*)
  - *Draw active decorations*: whether active *decorations* should be kept in the output
- Under *Temporal settings*, you can redefine:
  - the time *Range* for the animation
  - the *Step (frame length)* in the unit of your choice

7.1.4 The Elevation Controller panel

Elevation Controller allows you to handle elevation z-range data in 2D maps. This option currently supports point cloud layers and raster DEMs. Use the panel to visualize and interact with elevation data in the map canvas.

Activate the *Elevation Controller* panel in the map canvas by clicking on the *Data Filtering* in the *View* menu. The *Elevation Controller* appears as a range slider on the left side of the map canvas. At the top of the slider, there is a Settings menu where you can:

- Set the *Fixed Range Size* value to lock the elevation range to a specific size. This setting is saved within the project and retained across sessions, ensuring consistency for your elevation filter. Use the *Invert Direction* option to reverse the elevation filter slider, so that it moves from high values at the bottom to low values at the top.
- *Set Elevation Range* to open the project elevation tab, where you can set dedicated lower and upper bounds for the elevation range
- *Disable Elevation Filter* to remove the elevation filter from the map canvas

7.1.5 Bookmarking extents on the map

Spatial Bookmarks allow you to "bookmark" a geographic location and return to it later. By default, bookmarks are saved in the user's profile (as *User Bookmarks*), meaning that they are available from any project the user opens. They can also be saved for a single project (named *Project Bookmarks*) and stored within the project file, which can be helpful if the project is to be shared with other users.

Creating a Bookmark

To create a bookmark:

1. Zoom and pan to the area of interest.
2. Select the menu option *View* ► *New Spatial Bookmark…*, press `Ctrl+B` or right-click the *Spatial Bookmarks* entry in the *Browser* panel and select *New Spatial Bookmark*. The *Bookmark Editor* dialog opens.

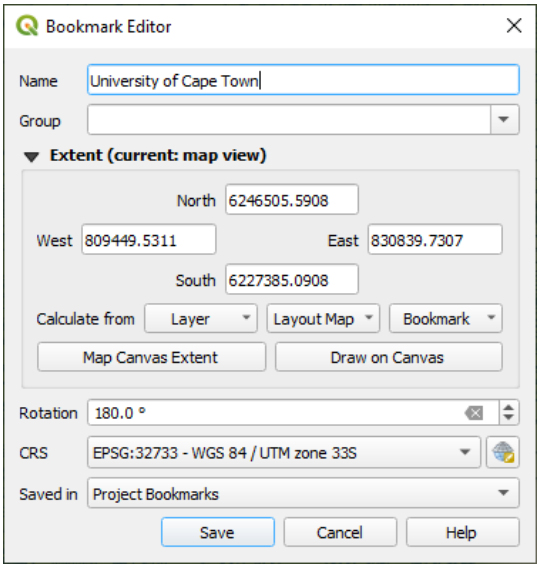


Fig. 7.4: The Bookmark Editor Dialog

3. Enter a descriptive name for the bookmark
4. Enter or select a group name in which to store related bookmarks
5. Select the extent of the area you wish to save, using the *extent selector* widget
6. Change the map *Rotation*
7. Indicate the *CRS* to use for the extent
8. Select whether the bookmark will be *Saved in User Bookmarks* or *Project Bookmarks* (by default, this drop-down list is set to *User Bookmarks*)
9. Press *Save* to add the bookmark to the list

Note that you can have multiple bookmarks with the same name.

Working with Bookmarks

To use and manage bookmarks, you can either use the *Spatial Bookmarks* panel or *Browser*.

Select *View* ► *Show Spatial Bookmark Manager* or press `Ctrl+7` to open the *Spatial Bookmarks Manager* panel.
Select *View* ► *Show Bookmarks* or `Ctrl+Shift+B` to show the *Spatial Bookmarks* entry in the *Browser* panel.

You can perform the following tasks:

Table 7.2: Managing bookmark actions

Task	Spatial Bookmark Manager	Browser
Zoom to a Bookmark	Double-click on it, or select the bookmark and press the *Zoom to bookmark* button.	Double-click on it, drag and drop it to the map canvas, or right-click the bookmark and select *Zoom to Bookmark*.
Delete a bookmark	Select the bookmark and click the *Delete bookmark* button. Confirm your choice.	Right-click the bookmark and select *Delete Spatial Bookmark*. Confirm your choice.

continues on next page

Table 7.2 – continued from previous page

Task	Spatial Bookmark Manager	Browser
Export bookmarks to XML	Click the *Import/Export Bookmarks* button and select *Export*. All the bookmarks (user or project) are saved in an xml file.	Select one or more folders (user or project) or subfolders (groups), then right-click and select *Export Spatial Bookmarks…*. The selected bookmark subset is saved.
Import bookmarks from XML	Click the *Import/Export Bookmarks* button and select *Import*. All bookmarks in the XML file are imported as user bookmarks.	Right-click the *Spatial Bookmarks* entry or one of its folders (user or project) or subfolders (groups) to determine where to import the bookmarks, then select *Import Spatial Bookmarks*. If performed on the *Spatial Bookmarks* entry, the bookmarks are added to *User Bookmarks*.
Edit bookmark	You can change a bookmark by changing the values in the table. You can edit the name, the group, the extent and if it is stored in the project or not.	Right-click the desired bookmark and select *Edit Spatial Bookmark…*. The *Bookmark Editor* will open, allowing you to redefine every aspect of the bookmark as if you were creating it for the first time. You can also drag and drop the bookmark between folders (user and project) and subfolders (groups).

You can manage bookmark actions by right-click on the desired bookmark in the *Spatial Bookmarks Manager*. You can also zoom to bookmarks by typing the bookmark name in the *locator*.

7.1.6 Decorating the map

Decorations include Grid, Title Label, Copyright Label, Image, North Arrow, Scale Bar and Layout Extents. They are used to 'decorate' the map by adding cartographic elements.

Grid

Grid allows you to add a coordinate grid and coordinate annotations to the map canvas.

1. Select menu option *View* ► *Decorations* ► *Grid…* to open the dialog.

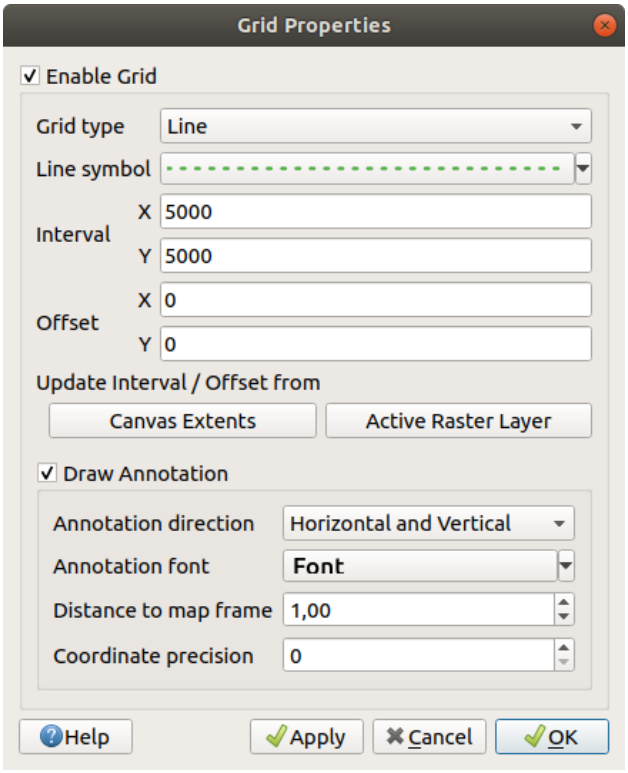


Fig. 7.5: The Grid Dialog

2. Tick *Enable grid* and set grid definitions according to the layers loaded in the map canvas:
   - The *Grid type*: it can be *Line* or *Marker*
   - The associated *Line symbol* or *marker symbol* used to represent the grid marks
   - The *Interval X* and *Interval Y* between the grid marks, in map units
   - An *Offset X* and *Offset Y* distance of the grid marks from the bottom left corner of the map canvas, in map units
   - The interval and offset parameters can be set based on the:
     - *Canvas Extents*: generates a grid with an interval that is approximatively 1/5 of the canvas width
     - *Active Raster Layer* resolution
3. Tick *Draw annotations* to display the coordinates of the grid marks and set:
   - The *Annotation direction*, ie how the labels would be placed relative to their grid line. It can be:
     - *Horizontal* or *Vertical* for all the labels
     - *Horizontal and Vertical*, ie each label is parallel to the grid mark it refers to
     - *Boundary direction*, ie each label follows the canvas boundary, and is perpendicular to the grid mark it refers to
   - The *Annotation font* (text formatting, buffer, shadow…) using the *font selector widget*
   - The *Distance to map frame*, margin between annotations and map canvas limits. Convenient when *exporting the map canvas* eg to an image format or PDF, and avoid annotations to be on the "paper" limits.
   - The *Coordinate precision*
4. Click *Apply* to verify that it looks as expected or *OK* if you're satisfied.

Title Label

Title Label allows you to decorate your map with a **Title**.

To add a Title Label decoration:

1. Select menu option *View* ► *Decorations* ► *Title Label…* to open the dialog.

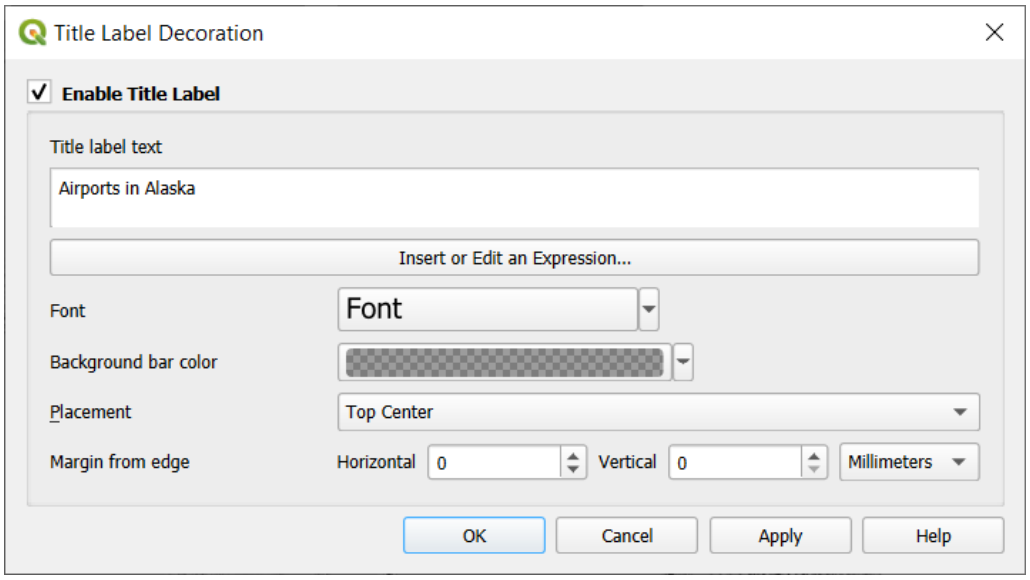


Fig. 7.6: The Title Decoration Dialog

2. Make sure *Enable Title Label* is checked
3. Enter the title text you want to place on the map. You can make it dynamic using the *Insert or Edit an Expression…* button.
4. Choose the *Font* for the label using the *font selector widget* with full access to QGIS *text formatting* options. Quickly set the font color and opacity by clicking the black arrow to the right of the font combo box.
5. Select the *color* to apply to the title's *Background bar color*.
6. Choose the *Placement* of the label in the canvas: options are *Top left*, *Top Center* (default), *Top Right*, *Bottom left*, *Bottom Center* and *Bottom Right*.
7. Refine the placement of the item by setting a horizontal and/or vertical *Margin from Edge*. These values can be in **Millimeters** or **Pixels** or set as a **Percentage** of the width or height of the map canvas.
8. Click *Apply* to verify that it looks as expected or *OK* if you're satisfied.

Copyright Label

Copyright Label can be used to decorate your map with a **Copyright** label.

To add this decoration:

1. Select menu option *View* ► *Decorations* ► *Copyright Label…* to open the dialog.

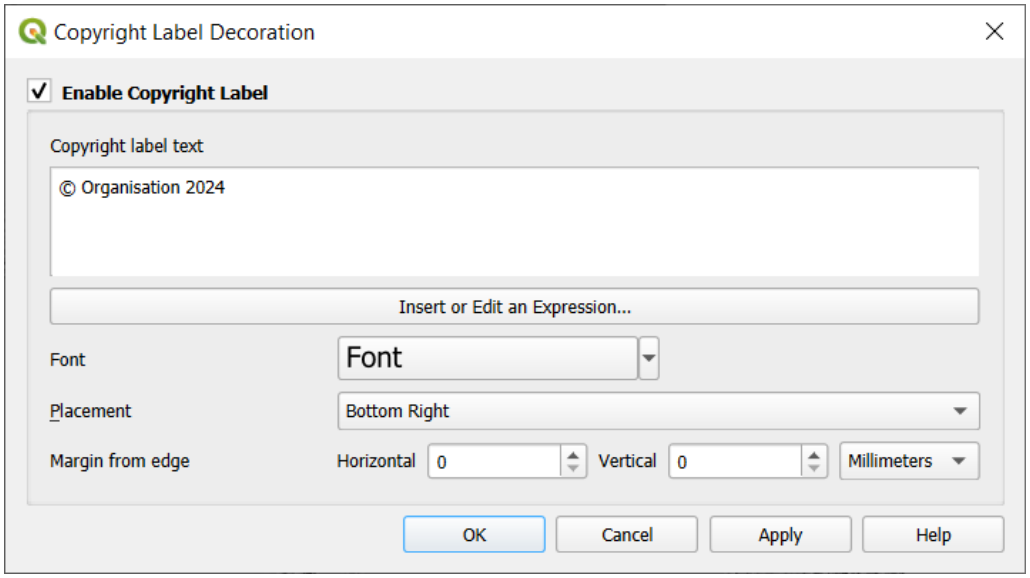


Fig. 7.7: The Copyright Decoration Dialog

2. Make sure *Enable Copyright Label* is checked
3. Enter the copyright text you want to place on the map. You can make it dynamic using the *Insert or Edit an Expression…* button.
4. Choose the *Font* for the label using the *font selector widget* with full access to QGIS *text formatting* options. Quickly set the font color and opacity by clicking the black arrow to the right of the font combo box.
5. Choose the *Placement* of the label in the canvas: options are *Top left*, *Top Center*, *Top Right*, *Bottom left*, *Bottom Center*, and *Bottom Right* (default for Copyright decoration)
6. Refine the placement of the item by setting a horizontal and/or vertical *Margin from Edge*. These values can be in **Millimeters** or **Pixels** or set as a **Percentage** of the width or height of the map canvas.
7. Click *Apply* to verify that it looks as expected or *OK* if you're satisfied.

Image Decoration

Image allows you to add an image (logo, legend, ..) on the map canvas.

To add an image:

1. Select menu option *View* ► *Decorations* ► *Image…* to open the dialog.

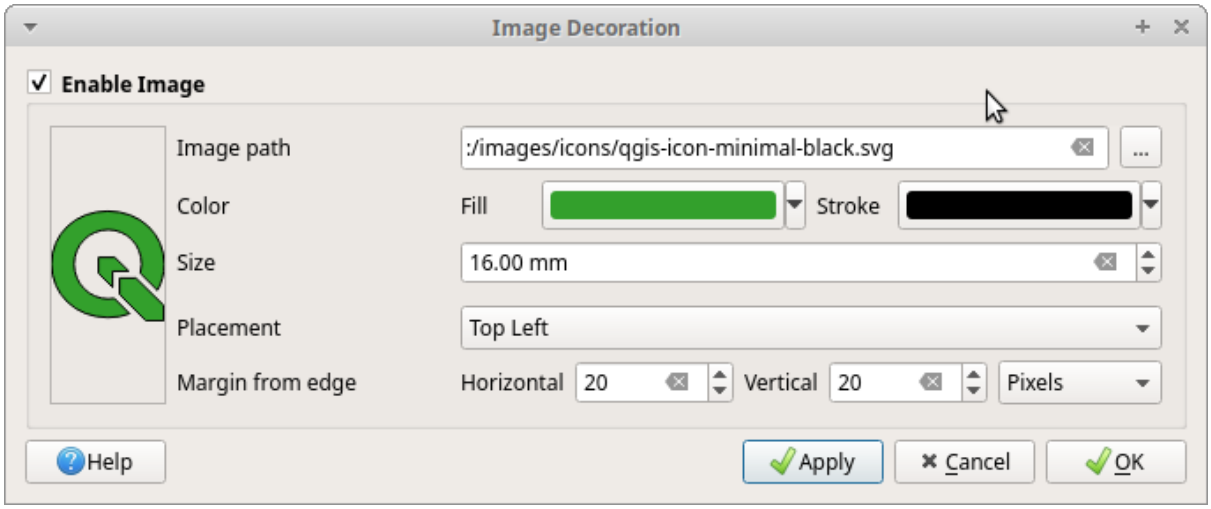


Fig. 7.8: The Image Decoration Dialog

2. Make sure *Enable Image* is checked
3. Select a bitmap (e.g. png or jpg) or SVG image using the … Browse button
4. If you have chosen a parameter enabled SVG then you can also set a *Fill* or *Stroke* (outline) color. For bitmap images, the color settings are disabled.
5. Set a *Size* of the image in mm. The width of selected image is used to resize it to given *Size*.
6. Choose where you want to place the image on the map canvas with the *Placement* combo box. The default position is *Top Left*.
7. Set the *Horizontal* and *Vertical Margin from (Canvas) Edge*. These values can be set in **Millimeters**, **Pixels** or as a **Percentage** of the width or height of the map canvas.
8. Click *Apply* to verify that it looks as expected and *OK* if you're satisfied.

North Arrow

North Arrow allows you to add a north arrow on the map canvas.

To add a north arrow:

1. Select menu option *View* ► *Decorations* ► *North Arrow*… to open the dialog.

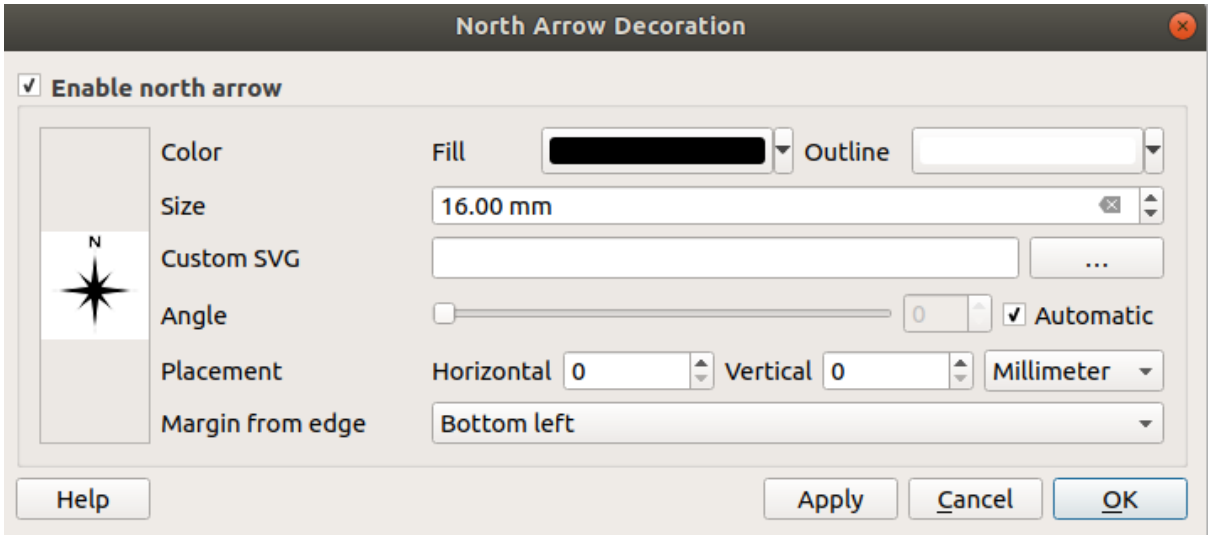


Fig. 7.9: The North Arrow Dialog

2. Make sure *Enable north arrow* is checked
3. Optionally change the color and size, or choose a custom SVG
4. Optionally change the angle or choose **Automatic** to let QGIS determine the direction
5. Optionally choose the placement from the Placement combo box
6. Optionally refine the placement of the arrow by setting a horizontal and/or vertical *Margin from (Canvas) Edge*. These values can be in **Millimeters** or **Pixels** or set as a **Percentage** of the width or height of the map canvas.
7. Click *Apply* to verify that it looks as expected and *OK* if you're satisfied.

Scale Bar

Scale Bar adds a simple scale bar to the map canvas. You can control the style and placement, as well as the labelling of the bar. The scale bar respects the active *project's distance unit* as defined in *Project properties* ► *General* ► *Units for distance measurement*.

To add a scale bar:

1. Select menu option *View* ► *Decorations* ► *Scale Bar…* to open the dialog

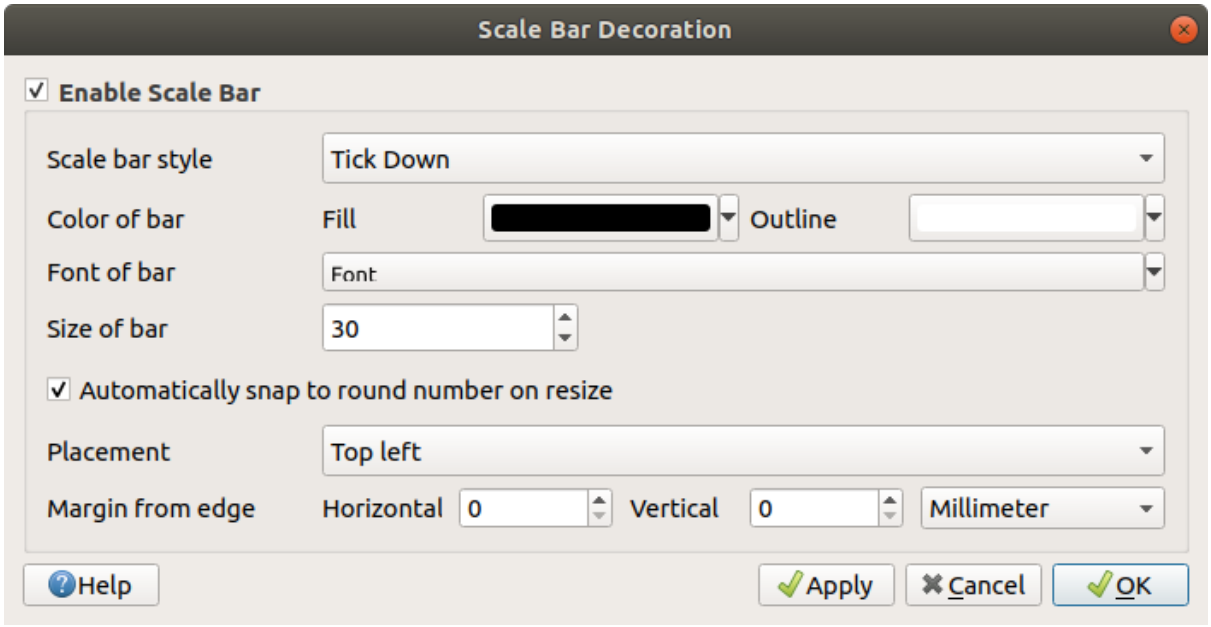


Fig. 7.10: The Scale Bar Dialog

2. Make sure *Enable scale bar* is checked
3. Choose a style from the *Scale bar style* combo box
4. Select the *Color of bar* by choosing a fill color (default: black) and an outline color (default: white). The scale bar fill and outline can be made opaque by clicking on the down arrow to the right of the color input.
5. Select the font for the scale bar from the *Font of bar* combo box
6. Set the *Size of bar* in the active unit
7. Optionally check *Automatically snap to round number on resize* to display easy-to-read values
8. Choose the placement from the *Placement* combo box
9. You can refine the placement of the item by setting a horizontal and/or vertical *Margin from (Canvas) Edge*. These values can be in **Millimeters** or **Pixels** or set as a **Percentage** of the width or height of the map canvas.
10. Click *Apply* to verify that it looks as expected or *OK* if you're satisfied.

Layout Extents

Layout Extents adds the extents of *map item(s)* in print layout(s) to the canvas. When enabled, the extents of all map items within all print layouts are shown using a lightly dotted border labeled with the name of the print layout and map item. You can control the style and labeling of the displayed layout extents. This decoration is useful when you are tweaking the positioning of map elements such as labels, and need to know the actual visible region of print layouts.



Fig. 7.11: Example of layout extents displayed in a QGIS project with two print layouts. The print layout named 'Sights' contains two map items, while the other print layout contains one map item.

To add layout extent(s):

1. Select *View* ► *Decorations* ► *Layout Extents* to open the dialog

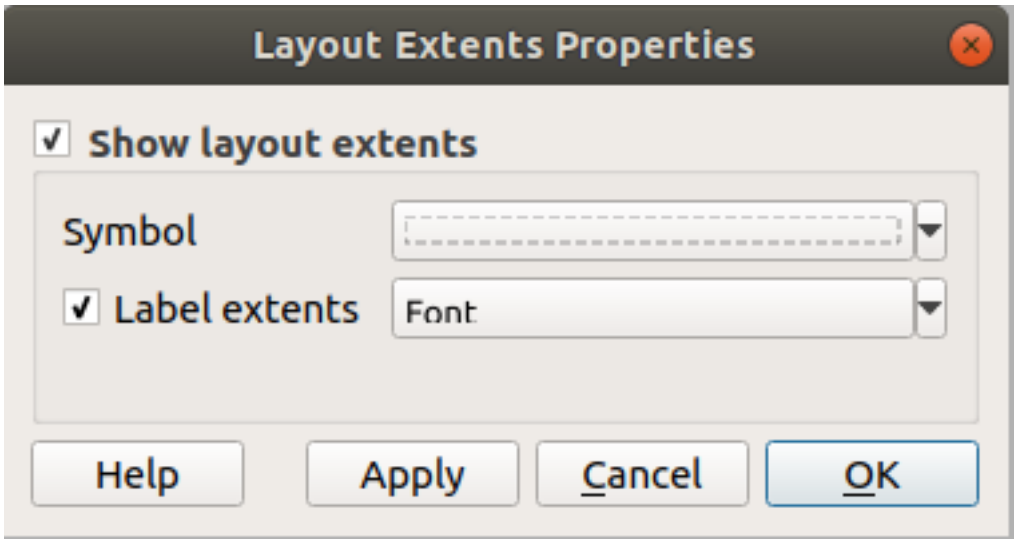


Fig. 7.12: The Layout Extents Dialog

2. Make sure  *Show layout extents* is checked.

3. Optionally change the symbol and labeling of the extents.
4. Click *Apply* to verify that it looks as expected and *OK* if you're satisfied.

> **Tip**
>
> **Decorations Settings**
>
> When you save a QGIS project file, any changes you have made to Grid, North Arrow, Scale Bar, Copyright and Layout Extents will be saved in the project and restored the next time you load the project.

7.1.7 Annotation Tools

Annotations are another type of elements added onto the map canvas to provide additional information that can not be depicted by the rendered layers. Unlike *labels* that rely on attribute values stored in vector layers, annotations are independent details, stored within the project file itself.

Two families of annotations are available in QGIS:

- **Feature annotations**: they are actual georeferenced features of text, marker, line or polygon type stored within a special layer type called "annotation layer". They are tied to a particular geographic location, meaning that moving your map, changing the scale or changing projection won't cause your annotations to jump around the map. Rather, they'll be locked in place to the location you've drawn them.
- **Balloon annotations**: these are individual HTML or form annotations inside a bubble. They can be associated to any layer for their visibility and drawn on top of the map canvas. The size is dependent from the map canvas scale, and its position can be anchored.

> **Tip**
>
> **Layout the map with annotations**
>
> You can print or export annotations with your map to various formats using:
>
> - map canvas export tools available in the *Project* menu
> - *print layout*, in which case you need to check *Draw map canvas items* in the corresponding map item properties

The *Annotations Toolbar* provides a set of tools to create and interact with both families of annotations.

Table 7.3: The Annotations Toolbar actions

Tool	Usage	Scope
New Annotation Layer	Create a new layer to store annotations	Feature annotations
Main Annotation Layer Properties	Control settings of the Main Annotation Layer	
Modify Annotations	Select, move, resize and modify symbology properties of annotations	
Create Polygon Annotation	Create an annotation as a polygon feature	
Create Line Annotation	Create an annotation as a polyline feature	
Create Marker Annotation	Create an annotation as a point feature	
Create Text Annotation at Point	Create an annotation as a text label	
Create Text Annotation along Line	Create an annotation as a curved text along a line feature	
Create Text Annotation In Rectangle	Create a text formatted annotation bound within a rectangle	
Create Picture Annotation	Create an annotation showing an `.SVG` or raster image	
HTML Annotation	Select and create annotation with an `HTML` file's content	Balloon annotations
Form Annotation	Select and create annotation showing attributes of a vector layer in a custom form file	

Note

Starting with QGIS 3.40, it is no longer possible to create balloon annotations of SVG or text type. When loading old projects, any of these annotations will automatically be converted to the newer picture and rectangular text annotation item types.

Feature Annotations

Feature annotations are stored in **annotation layers**. Unlike conventional layers, an annotation layer is available only in the current project and can contain features of different types (text, marker, line, polygon, picture). The layer has no attributes and no symbology associated, but instead each feature can be symbolized on an item-by-item basis, through *Layer Styling* panel.

Two types of annotation layer are available in QGIS:

- A common *Annotation Layer*: you can create one using the New Annotation Layer tool. It is listed in the *Layers* panel, allowing you to control its features' visibility, move it to show above or below particular layers in your map, like any common layer. Double-click the layer and you can access its properties.
- The *Main Annotation Layer*: By default, this is where annotations are stored when no annotation layer is available in the project or is selected at creation time. This layer is always drawn on the very top of your map and you won't see it listed in the *Layers* panel alongside the other layers in your project, meaning that its features are always visible. The *Main Annotation Layer Properties* entry on the *Annotations* toolbar helps you open its properties dialog.

Layer Properties

The properties dialog of an annotation layer provides the following tabs:

- *Information*: a read-only dialog representing an interesting place to quickly grab summarized information and metadata on the current layer. This may include the layer extent, count of items per annotation type and total count, CRS details, …
- *Source*: defines general settings for the annotation layer. You can:
  - Set a *Layer name* that will be used to identify the layer in the project (in the *Layers Panel*, with expressions, …)
  - Display the layer's *Assigned Coordinate Reference System (CRS)*: you can change the layer's CRS, selecting a recently used one in the drop-down list or clicking on  Select CRS button (see *Coordinate Reference System Selector*). Use this process only if the CRS applied to the layer is a wrong one or if none was applied.
- *Rendering*:
  - You can set the *Maximum (inclusive)* and *Minimum (exclusive)* scale, defining a range of scale in which features will be visible. Out of this range, they are hidden. The  Set to current canvas scale button helps you use the current map canvas scale as boundary of the range visibility. See *Visibility Scale Selector* for more information.
  - *Linked layer*: allows you to optionally set a linked visibility layer for the annotation layer. If set, then the annotations will only be drawn when the linked layer is visible in the map.
  - *Opacity*: You can make the underlying layer in the map canvas visible with this tool. Use the slider to adapt the visibility of your vector layer to your needs. You can also make a precise definition of the percentage of visibility in the menu beside the slider.
  - *Blending mode* at the *Layer* level: You can achieve special rendering effects with these tools that you may previously only know from graphics programs. The pixels of your overlaying and underlying layers are mixed through the settings described in *Blending Modes*.
  - Apply *paint effects* on all the layer features with the *Draw Effects* button.

  Some of these options are accessible from the feature annotation *Symbology* properties.

Interacting with features

The Feature annotations have dedicated tools for creation depending on their type.

Table 7.4: Creating feature annotation

Family / based on	Annotation tool	Procedure
Geometry	 Create Polygon Annotation	This is like digitizing a point, line or polygon vector feature. All the usual shortcuts for creating features apply when creating annotation items. A line or polygon annotation is drawn by left-clicking once for each vertex, with a final right mouse click to complete the shape. Snapping can be enabled while you draw, you can use the *Advanced Digitizing Tools* to precisely place vertices, and even switch the *drawing tools* to for example the streaming mode for completely free-form shapes.
	 Create Line Annotation	
	 Create Marker Annotation	
Text	 Create Text	Left-click on the map canvas.
	 Create Text	Left-click to add vertices to the text basement line geometry, and right-click to end. As above, digitizing features capabilities are also available.
	 Create Text	Draw the rectangle box of the text, by left-clicking to add a first corner and left-clicking again to add the opposite one.
Picture	 Create Pictu	Draw the rectangle box of the picture, by left-clicking to add a first corner and left-clicking again to add the opposite one. Then pick an image or SVG file from the file explorer.

Unlike common layers, an annotation layer does not need to be active before you select its features. Simply grab the  Modify Annotations tool and you will be able to interact with any feature annotation:

- **Selection**: left-click on the annotation. By default, annotations are rendered in the order of creation, meaning that recent annotations will be placed on top of older. You may need to play with the Z-index property of features in order to select ones they would sit above.
- **Moving**: Left click on a selected annotation item to start moving it. A right-click or pressing `Esc` key cancels the move, while a second left click will confirm the move. The displacement can also be controlled pressing the cursor keys:
  - `Shift+key` for big movement
  - `Alt+key` for `1 px` movement
- **Geometry modification**: move over an annotation and purple squares are displayed on nodes of its underlying geometry. Left-click on the square, move and click again. For linear or polygonal annotations, double-clicking a segment will add a new vertex.
- **Delete**: Pressing the `Del` or `Backspace` key while an annotation is selected will delete that annotation
- *Change feature symbology*

Feature symbology

A selected annotation will display its *Symbology* properties in the *Layer styling* panel.

- For polygon, polyline and marker annotations, you can modify the appearance using full capabilities of the *symbol properties*
- For text-based annotations, an area with text editing tools helps you construct the string to display. It can be multiline, use HTML formatting and rely on QGIS expression functions. You can moreover modify the appearance using full capabilities of the *text format properties*.

  Depending on the type of annotation, additional options are available.

  - For text annotation at point, it is also possible to:
    - set the text *Alignment* (left, center or right of the anchor point)
    - configure whether the text should *Ignore map rotation* or *Rotate with map*. In both cases, a custom *Angle* can be set for the feature orientation.
  - For text annotation along a line, configure an *Offset from line* in the unit of your choice
  - For text annotation in rectangle, it is also possible to:
    - set the text *Horizontal alignment* (left, center, right or justify) and *Vertical alignment* (top, vertical center or bottom) in the rectangle box
    - configure for the rectangle, the *Margins* distance in the units of your choice, the *Frame* and *Background* colors.

- Picture annotations allow you to display a *Raster image* or an *SVG image*, using the *Remote or embedded file selector*. The placement and size of the picture can be set as:
  - *Scale dependent size*: the picture scales with the map, appearing larger when zooming in and smaller when zooming out. The size is defined by a single reference size in map units (e.g. meters or degrees, depending on the layer's CRS). With *Lock aspect ratio*, you can keep the annotation size proportional to the embedded picture size, or stretch the picture to fill the annotation extent.
  - *Fixed size*: the picture annotation maintains a constant size in screen or output units (e.g. pixels or millimeters), regardless of the map scale. The *Width* and *Height* of the annotation can be filled separatey or adjusted proportionally to the embedded picture size.
  - *Relative to map*: the picture annotation will always be rendered at the same position relative to the map canvas bounds, regardless of the map scale. The *Width* and *Height* of the annotation can be filled separatey or adjusted proportionally to the embedded picture size.

  Moreover, you can enable display of the *Frame* and *Background* of the rectangle box, and configure them using *fill symbols*.

- For text at point, text inside rectangle and picture annotations, you can *Show callout* when the picture or text anchor point is offset from its default placement. To create a callout for an annotation:
  1. Select the annotation item
  2. Move the anchor point, i.e., click the central and green `X` node and click at the new placement
  3. In the annotation properties, press the … button to configure the *callout properties*.

  Tip

  Applying a balloon callout to a text or picture annotation will make you get their old-style balloon annotation display.

- Configure a *Reference scale*: indicates the map scale at which symbol or text sizes which use paper-based units (such as millimeters or points) relate to. The sizes will be scaled accordingly whenever the map is viewed at a different scale. For instance, a line feature wide of 2mm at 1:2000 *Reference scale* will be rendered using 4mm when the map is viewed at 1:1000.
- Set a *Z-index*: a feature with a higher index is placed on top of features with lower index. A convenient setting for both feature display and selection.
- Modify some of the *Layer rendering* settings

Balloon annotations

You can add balloon annotations through *Edit* ► *Add Annotation* ► menu or from the *Annotations Toolbar*:

- HTML Annotation to place the content of an `html` file
- Form Annotation: useful to display attributes of a vector layer in a customized `ui` file (see Fig. 7.13). This is similar to the *custom attribute forms*, but displayed in an annotation item. Also watch *this video <https://www.youtube.com/watch?v=0pDBuSbQ02o&feature=youtu.be&t=2m25s>* from Tim Sutton for more information.

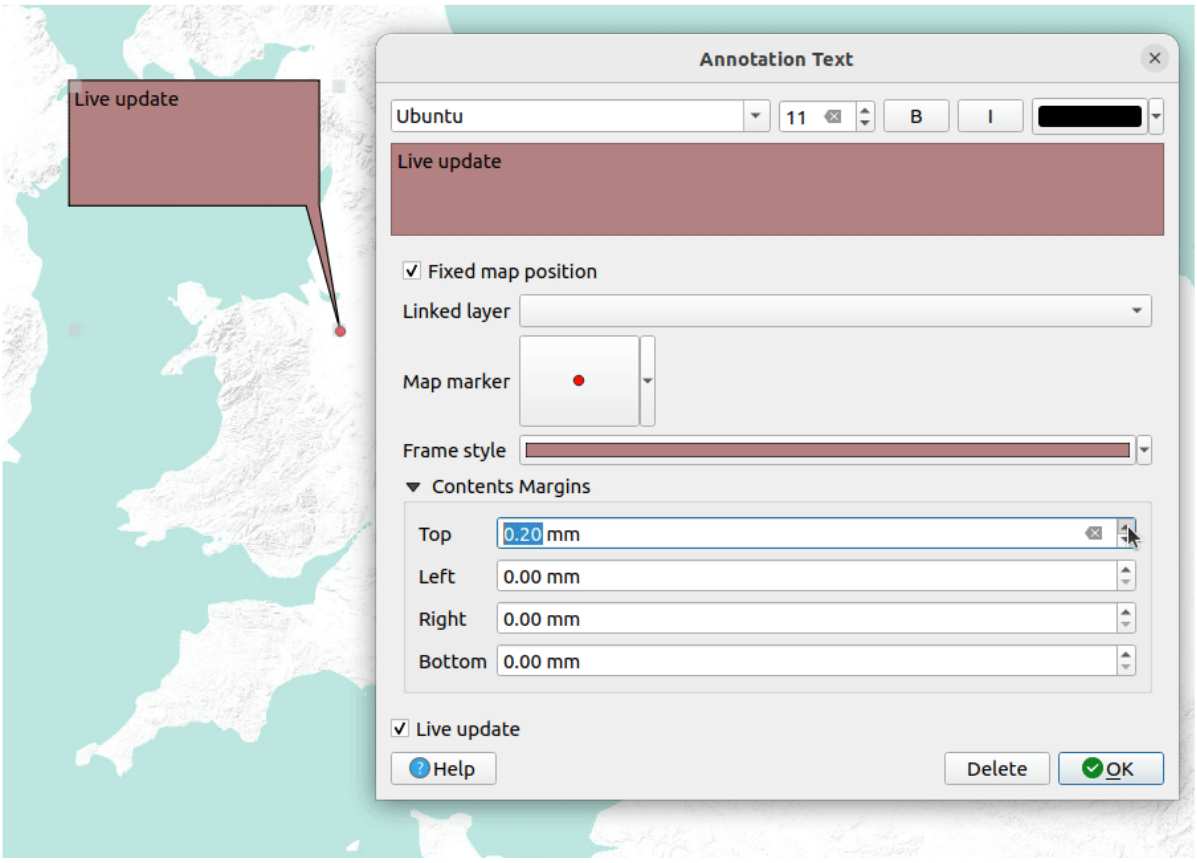


Fig. 7.13: Examples of balloon annotations

To add a balloon annotation, select the corresponding tool and click on the map canvas. An empty balloon is added. Double-click on it and a dialog opens with various options. This dialog is almost the same for all the annotation types:

- At the top, a file selector to fill with the path to an `html` or `ui` file depending on the type of annotation.
- *Fixed map position*: when unchecked, the balloon placement is based on a screen position (instead of the map), meaning that it's always shown regardless the map canvas extent.
- *Linked layer*: associates the annotation with a map layer, making it visible only when that layer is visible.
- *Map marker*: using *QGIS symbols*, sets the symbol to display at the balloon anchor position (shown only when *Fixed map position* is checked).
- *Frame style*: sets the frame background color, transparency, stroke color or width of the balloon using QGIS symbols.
- *Contents margins*: sets interior margins of the annotation frame.
- *Live update* allows you to live preview your changes.

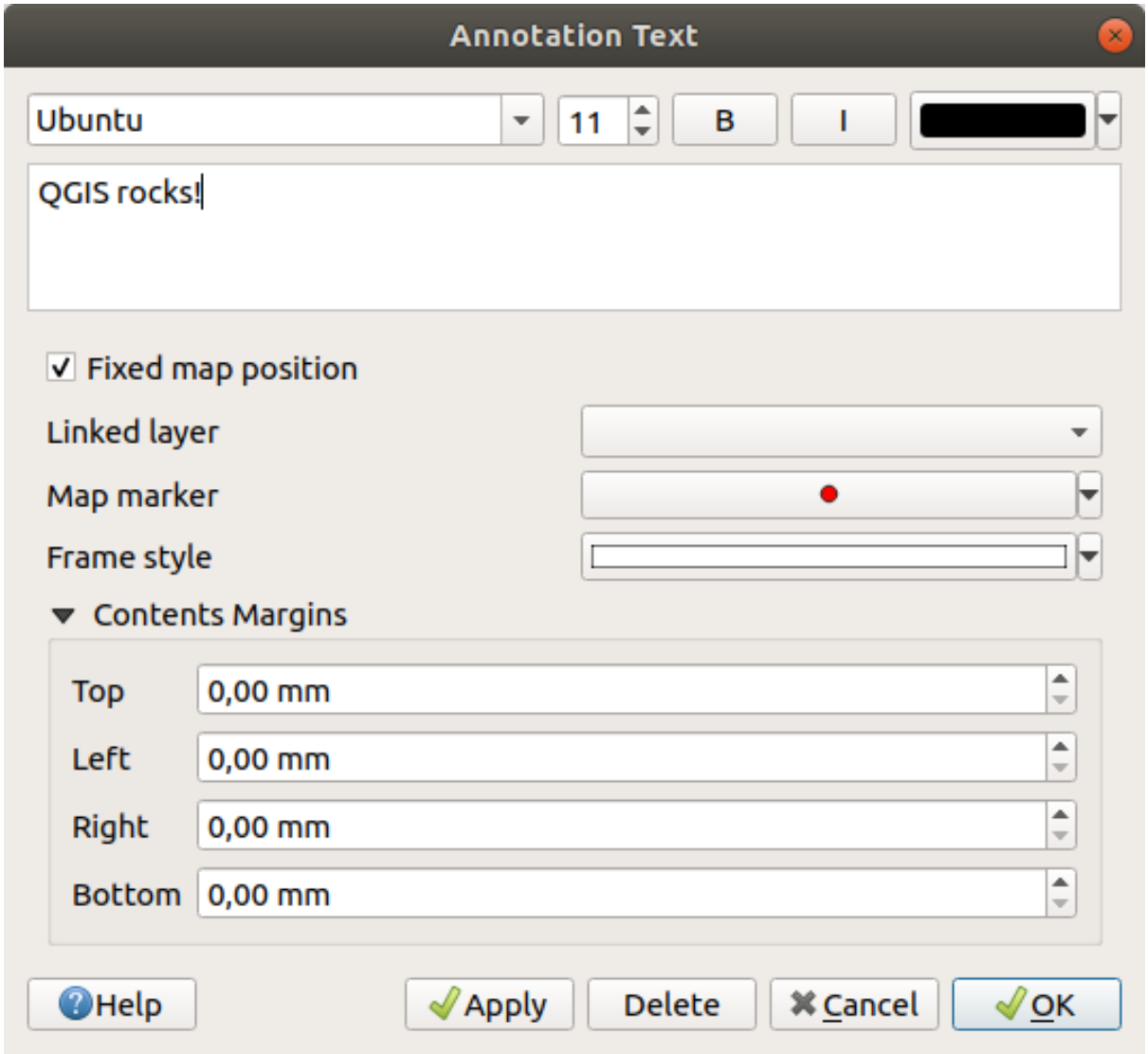


Fig. 7.14: A ballon annotation text dialog

When a balloon annotation tool is active, you can also:

- Select an annotation
- Resize an annotation
- Move an annotation by map position (by dragging the map marker) or by moving only the balloon.
- Delete an annotation: select it and either press the `Del` or `Backspace` button, or double-click it and press the *Delete* button in the properties dialog.
- Right-click and in the contextual menu:
  - *Copy coordinate* of the annotation's map marker in various CRS
  - *Edit* properties of the annotation. Same as double-clicking the annotation.
  - *Delete* the annotation

7.1.8 Measuring

General information

QGIS provides four means of measuring geometries:

- interactive measurement tools
- measuring in the Field Calculator
- derived measurements in the *Identifying Features* tool
- the vector analysis tool: *Vector* ► *Geometry Tools* ► *Export/Add Geometry Columns*

Measuring works within projected coordinate systems (e.g., UTM) and unprojected data. The first three measuring tools behave equally to global project settings:

- Unlike most other GIS, the default measurement metric is ellipsoidal, using the ellipsoid defined in *Project* ► *Properties…* ► *General*. This is true both when geographic and projected coordinate systems are defined for the project.
- If you want to calculate the projected/planimetric area or distance using cartesian maths, the measurement ellipsoid has to be set to "None/Planimetric" (*Project* ► *Properties…* ► *General*). However, with a geographic (ie unprojected) CRS defined for the data and project, area and distance measurement will be ellipsoidal.

However, neither the identify tool nor the field calculator will transform your data to the project CRS before measuring. If you want to achieve this, you have to use the vector analysis tool: *Vector* ► *Geometry Tools* ► *Add Geometry Attributes…*. Here, measurement is planimetric, unless you choose the ellipsoidal measurement.

Measure length, areas, bearings and angles interactively

Click the icon in the Attribute toolbar to begin measurements. The down arrow near the icon switches between length, area, bearing or angle. The default unit used in the dialog is the one set in *Project* ► *Properties…* ► *General* menu.

For the *Measure Line* and the *Measure Area* the measurements can be done in *Cartesian* or *Ellipsoidal* measure.

> **Note**
>
> **Configuring the measure tool**
>
> While measuring length or area, clicking the *Configuration* button at the bottom of the widget opens the *Settings* ► *Options* ► *Map Tools* menu, where you can select the rubberband color, the precision of the measurements and the unit behavior. You can also choose your preferred measurement or angle units, but keep in mind that those values are overridden in the current project by the selection made in the *Project* ► *Properties…* ► *General* menu, and by the selection made in the measurement widget.

All measuring modules use the snapping settings from the digitizing module (see section *Setting the snapping tolerance and search radius*). So, if you want to measure exactly along a line feature, or around a polygon feature, first set its layer snapping tolerance. Now, when using the measuring tools, each mouse click (within the tolerance setting) will snap to that layer.

The Measure Line measures distances between given points. The tool then allows you to click points on the map. Each segment length, as well as the total, shows up in the measure window. In the measure window, you will see coordinates for all your points and distances. Keep in mind that the first row will contain only coordinates, as it represents your starting point. Now it is possible to copy all your line measurements at once to the clipboard using the *Copy* button. Clicking the *Configuration* button you will access to *Measure Tool Copy Settings* where you can set up *copy options*. To stop measuring, click the right mouse button.

Note that you can use the drop-down list near the total to change the *measurement units* interactively while working with the measure tool. This unit is retained for the widget until a new project is created or another project is opened.

The *Info* section in the dialog explains how calculations are made according to the CRS settings available.

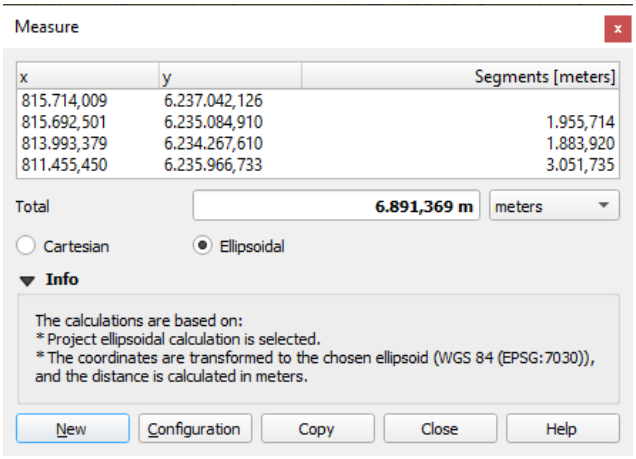


Fig. 7.15: Measure Distance

Measure Area: Areas can also be measured. In the measure window, the accumulated area size appears. Right-click to stop drawing. The Info section is also available as well as the ability to switch between different *area units*.

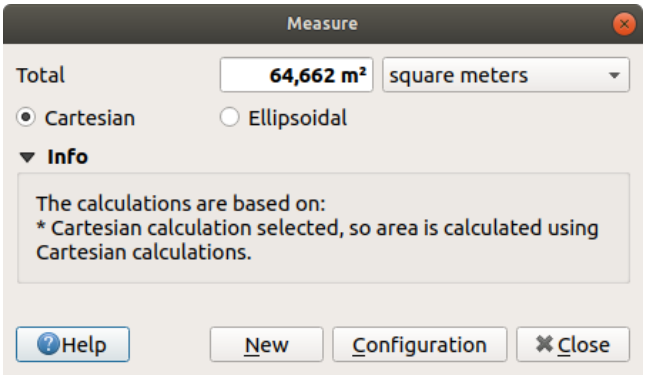


Fig. 7.16: Measure Area

Measure Bearing: You can also measure bearings. The cursor becomes cross-shaped. Click to draw the first point of the bearing, then move the cursor to draw the second point. The measurement is displayed in a pop-up dialog.

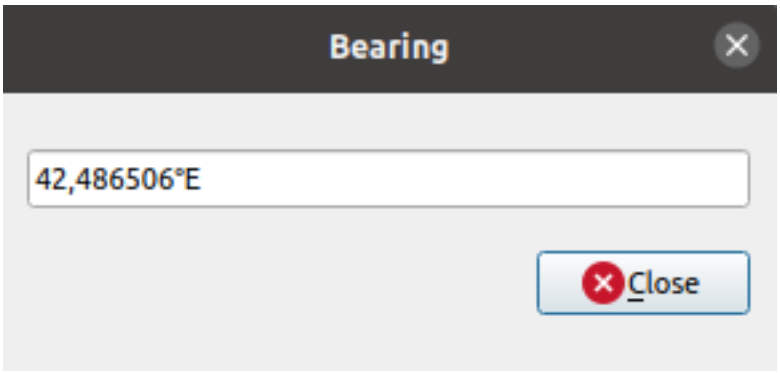


Fig. 7.17: Measure Bearing

Measure Angle: You can also measure angles. The cursor becomes cross-shaped. Click to draw the first segment of the angle you wish to measure, then move the cursor to draw the desired angle. The measurement is displayed in a pop-up dialog.

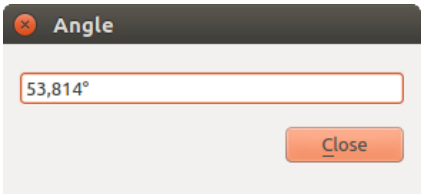


Fig. 7.18: Measure Angle

7.1.9 Setting additional map views

It is also possible to open additional map views whose content could diverge from the *Layers* panel current state. To add a new map view, go to *View* ► *New Map View*. A new floating widget, mimicking the main map view's rendering, is added to QGIS. You can add as many map views as you need. They can be kept floating, placed side by side or stacked on top of each other.

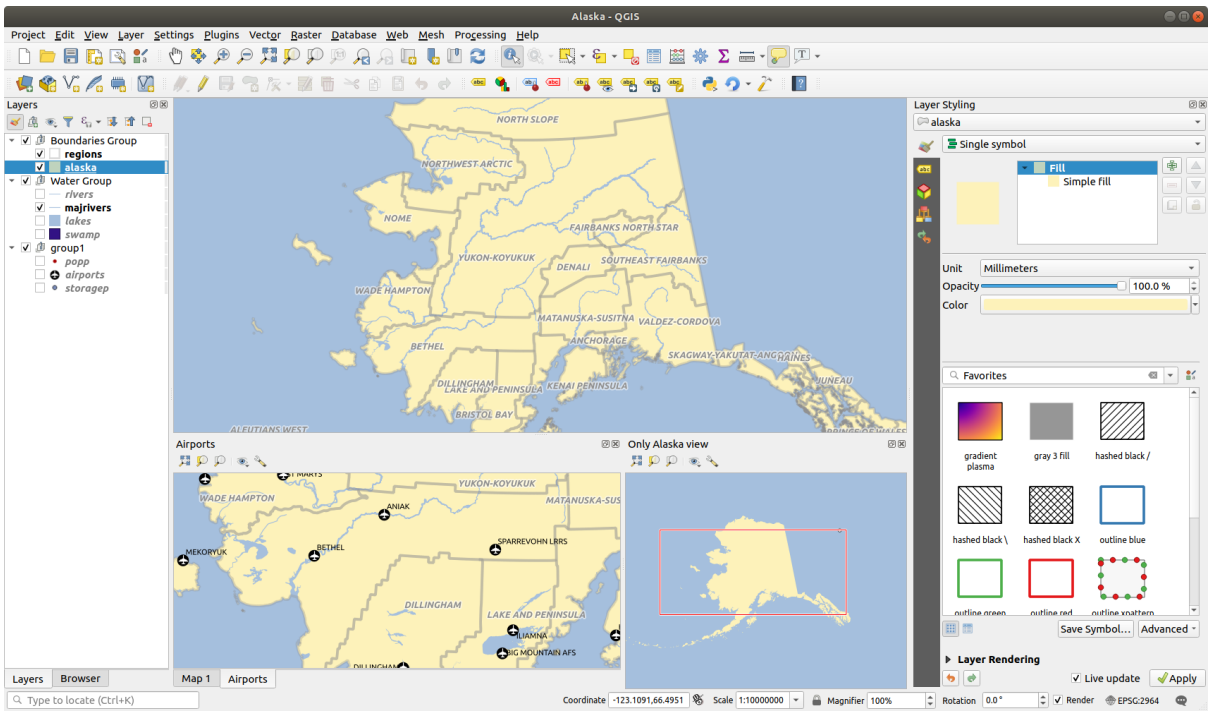


Fig. 7.19: Multiple map views with different settings

At the top of an additional map canvas, there's a toolbar with the following capabilities:

- Zoom Full, Zoom to Selection and Zoom to Layer(s) to navigate within the view
- Set View Theme to select the *map theme* to display in the map view. If set to `(none)`, the view will follow the *Layers* panel changes.
- View settings to configure the map view:
  - *Synchronize view center with main map*: syncs the center of the map views without changing the scale. This allows you to have an overview style or magnified map which follows the main canvas center.

- *Synchronize view to selection*: same as zoom to selection
- *Scale*
- *Rotation*
- *Magnification*
- *Synchronize scale* with the main map scale. A *Scale factor* can then be applied, allowing you to have a view which is e.g. always 2x the scale of the main canvas.
- *Show annotations*
- *Show cursor position*
- *Show main canvas extent*
- *Show labels*: allows to hide labels regardless they are set in the displayed layers' properties
- *Change map CRS…*
- *Rename view…*

7.1.10 Exporting the map view

Maps you make can be layout and exported to various formats using the advanced capabilities of the *print layout or report*. It's also possible to directly export the current rendering, without a layout. This quick "screenshot" of the map view has some convenient features.

To export the map canvas with the current rendering:

1. Go to *Project* ► *Import/Export*
2. Depending on your output format, select either
   - *Export Map to Image…*
   - or *Export Map to PDF…*

The two tools provide you with a common set of options. In the dialog that opens:

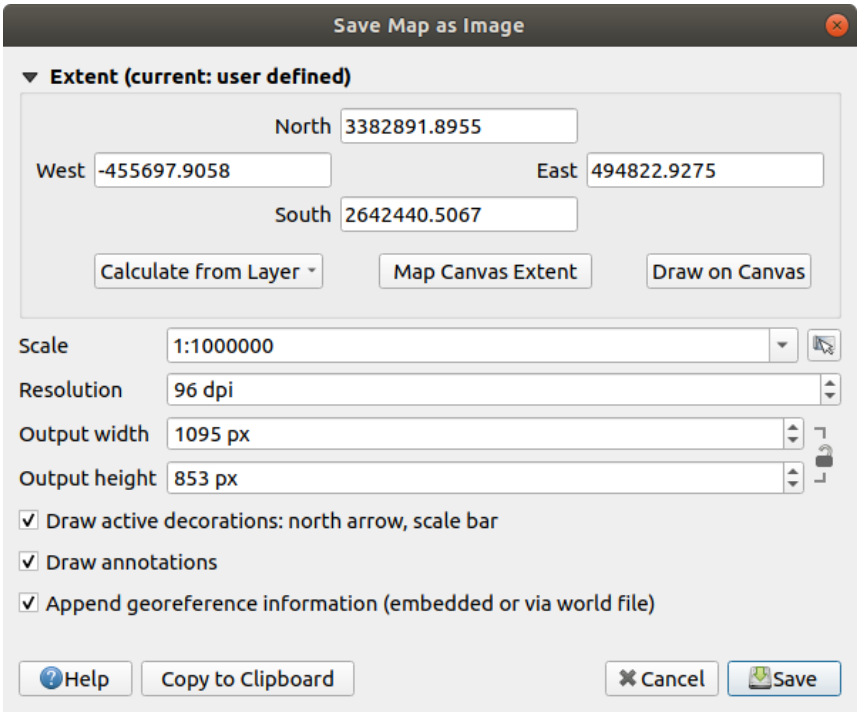


Fig. 7.20: The Save Map as Image dialog

1. Choose the *Extent* to export: it can be the current view extent (the default), the extent of a layer or a custom extent drawn over the map canvas. Coordinates of the selected area are displayed and manually editable.
2. Enter the *Scale* of the map or select it from the *predefined scales*: changing the scale will resize the extent to export (from the center).
3. Set the *Resolution* of the output
4. Control the *Output width* and *Output height* in pixels of the image: based by default on the current resolution and extent, they can be customized and will resize the map extent (from the center). The size ratio can be locked, which may be particularly convenient when drawing the extent on the canvas.
5. ☑ *Draw active decorations*: in use *decorations* (scale bar, title, grid, north arrow…) are exported with the map
6. ☑ *Draw annotations* to export any *annotation*
7. ☑ *Append georeference information (embedded or via world file)*: depending on the output format, a world file of the same name (with extension `PNGW` for `PNG` images, `JPGW` for `JPG`, …) is saved in the same folder as your image. The `PDF` format embeds the information in the PDF file.
8. When exporting to PDF, more options are available in the *Save map as PDF…* dialog:

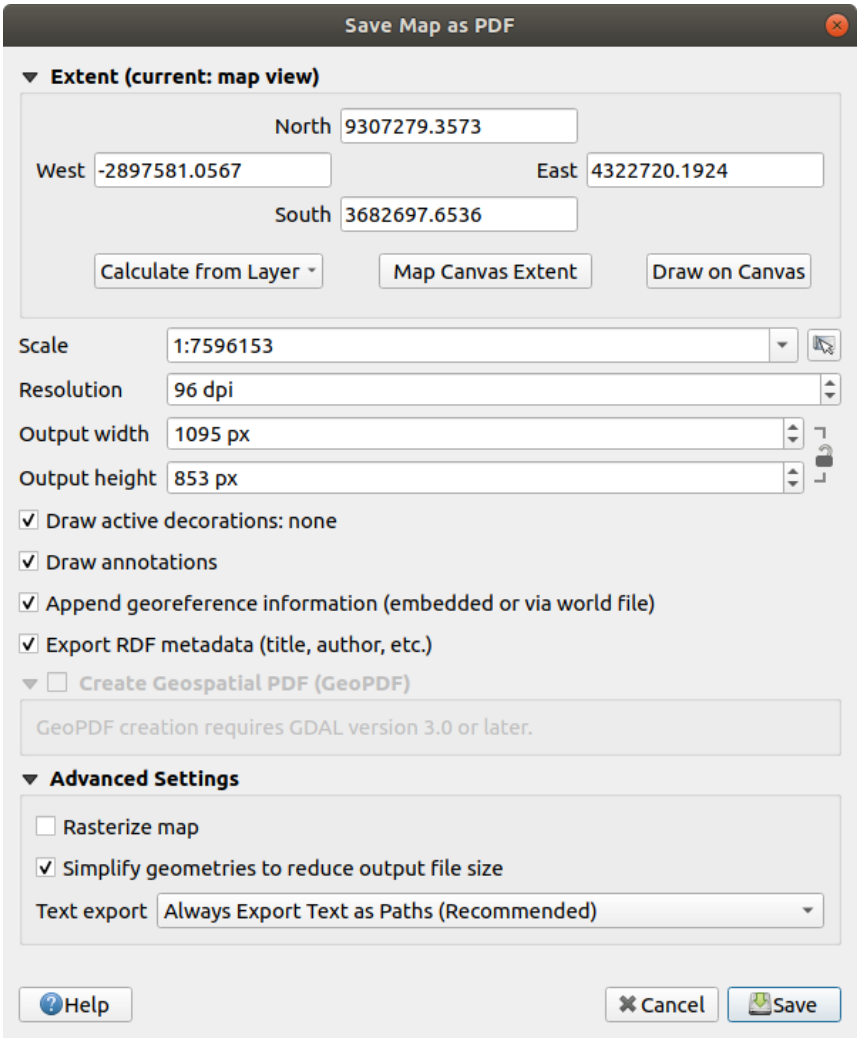


Fig. 7.21: The Save Map as PDF dialog

- *Export RDF metadata* of the document such as the title, author, date, description…
- *Create Geospatial PDF*: Generate a georeferenced PDF file. You can:
  - Choose the Geospatial PDF *Format*
  - *Include vector feature information* in the Geospatial PDF file: will include all the geometry and attribute information from features visible within the map in the output Geospatial PDF file.

> **Note**
>
> A Geospatial PDF file can also be used as a data source. For more on Geospatial PDF support in QGIS, see https://north-road.com/2019/09/03/qgis-3-10-loves-geospatialpdf/.

- *Rasterize map*
- *Simplify geometries to reduce output file size*: Geometries will be simplified while exporting the map by removing vertices that are not discernibly different at the export resolution (e.g. if the export resolution is `300 dpi`, vertices that are less than `1/600 inch` apart will be removed). This can reduce the size and complexity of the export file (very large files can fail to load in other applications).
- Set the *Text export*: controls whether text labels are always or preferably exported as *text or outline objects*.

9. Click *Save* to select file location, name and format.

   When exporting to image, it's also possible to *Copy to clipboard* the expected result of the above settings and paste the map in another application such as LibreOffice, GIMP…

7.2 3D Map View

3D visualization support is offered through the 3D map view. You can create, manage and open 3D map views via *View* ► *3D Map Views* ► menu:

1. By clicking on *New 3D Map View* you can create a new 3D map view. A floating and dockable QGIS panel will appear (see *The 3D Map View dialog*). It has the same extent and view as the 2D main map canvas and provides a set of navigation tools to turn the view into 3D.
2. By clicking on *Manage 3D Map Views* you get in the 3D Map Views Manager. Here you get the ability to open, duplicate, remove and rename 3D map views.
3. If you created one or more 3D map views, you see them listed in *3D Map Views*. You can turn them on and off by clicking on. They will be saved by saving the project, even if they are turned off.

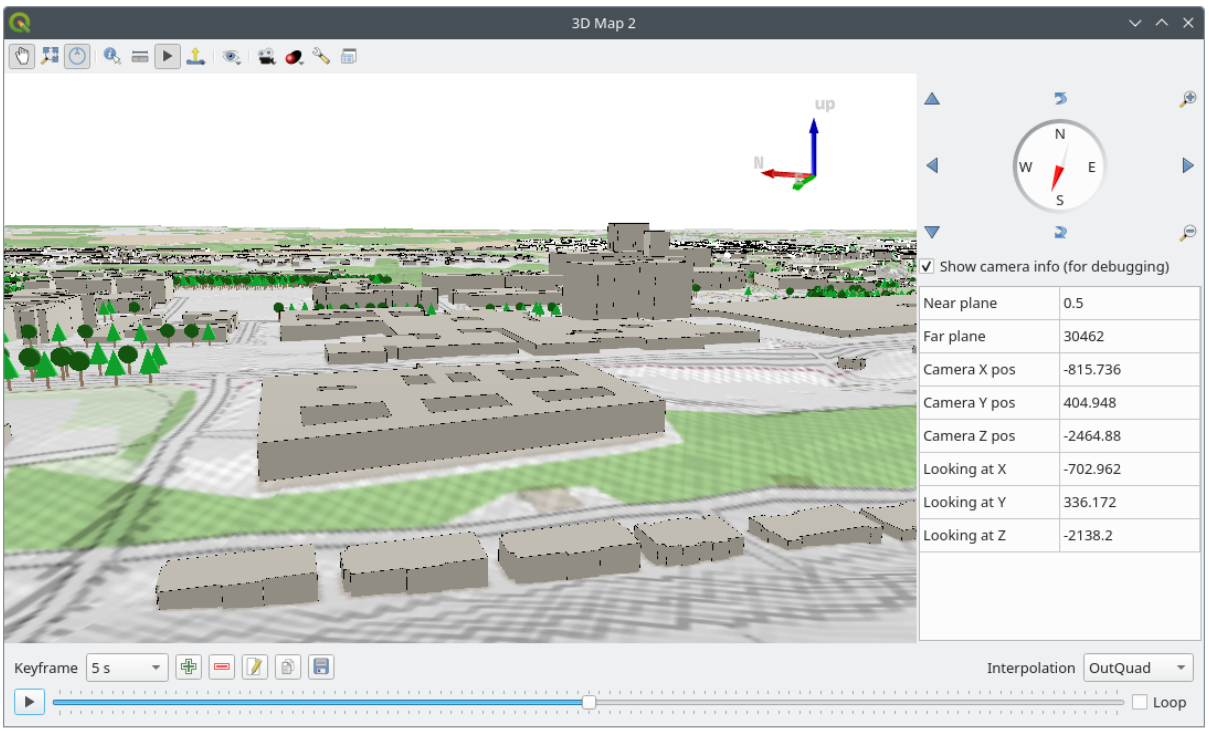


Fig. 7.22: The 3D Map View dialog

The following tools are provided at the top of the 3D map view panel:

- Camera Control: moves the view, keeping the same angle and direction of the camera
- Zoom Full: resizes the view to the whole layers' extent, or the *reference extent* if set
- Toggle On-Screen Notification: shows/hides the navigation widget (that is meant to ease controlling of the map view)

- Identify: returns information on the clicked point of the terrain or the clicked 3D feature(s) – More details at *Identifying Features*
- Measurement Line: measures the horizontal, vertical, and total 3D distance between points
  - Click to add points
  - Right-click to finish the measurement
  - Right-clicking twice or pressing *New* will reset the table
- Animations: shows/hides the *animation player* widget
- Export menu allows to export the scene to various formats:
  - Save as Image…: exports the current view to an image file format
  - Export 3D Scene: exports the current view as a 3D scene (`.obj` file), allowing post-processing in applications like Blender… The terrain and vector features are exported as 3D objects. The export settings, overriding the layers *properties* or map view *configuration*, include:
    - *Scene name* and destination *Folder*
    - *Terrain resolution*
    - *Terrain texture resolution*
    - *Model scale*
    - *Smooth edges*
    - *Export normals*
    - *Export textures*
- Set View Theme: Allows you to select the set of layers to display in the map view from predefined *map themes*.
- The Camera menu helps you control relation between the 2D and 3D views:
  - Synchronize the views (*2D map view follows 3D camera* and/or *3D camera follows 2D Map view*)
  - *Show visible camera area in 2D map view*
  - *Set 3D scene on 2D map view*: allows to clip the 3D scene and display only the terrain and features intersecting an extent drawn on the 2D map canvas. More options are available in the *General configuration* tab.
  - Cross Section Tool creates a cross section in 3D scene by defining an area on the terrain in the 2D map canvas. Click to set the start point, move the mouse to define the line direction and click again to set the end point. You can adjust the size of the section by moving the mouse after selecting the start and end points. Press `Esc` or right-click to reset the selection.
  - Use Disable Cross Section to remove the cross section.
- Effects adds visual effects to the 3D rendering, such as showing *shadows*, *eye dome lighting* or *ambient occlusion*.
- The Options button opens the dialog to configure the 3D map view *settings*.
- Dock 3D Map View: switch from docked widget to top level window

7.2.1 Scene Configuration

The 3D map view opens with some default settings you can customize. To do so, expand the Options menu at the top of the 3D canvas panel and press the *Configure* button to open the *3D configuration* window.

In the 3D Configuration window there are various options to fine-tune the 3D scene:

General

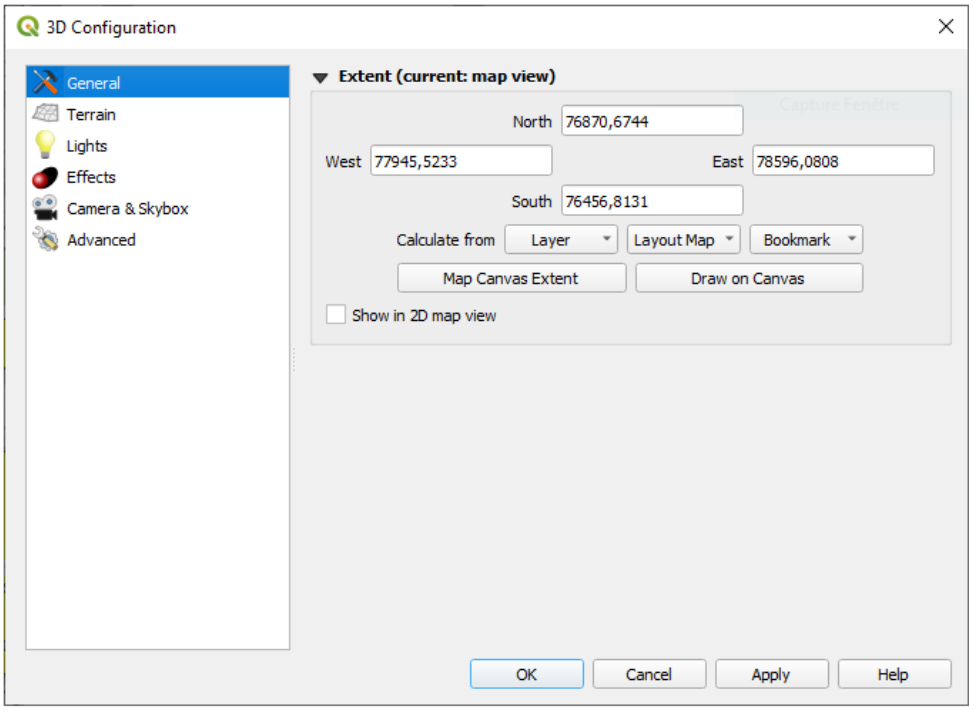


Fig. 7.23: The 3D Map General Configuration dialog

Under the *General* tab, you can:

- Limit the 3D scene's contents to a 2D map extent, using the *spatial extent selector*: terrain and features are clipped at the specified extent and only the parts falling within the extent are loaded in the scene. Elevation range of the terrain, mesh and pointcloud layers is also taken into account so that the camera is not positioned below the scene's contents, particularly when using the terrain's vertical scale setting to exaggerate the elevation differences.

  The set extent is also used as reference extent when pressing the Zoom full button in 3D map view.

- Check *Show in 2D map view* to display in the main map canvas a rubberband corresponding to the current extent of the 3D scene.

Terrain

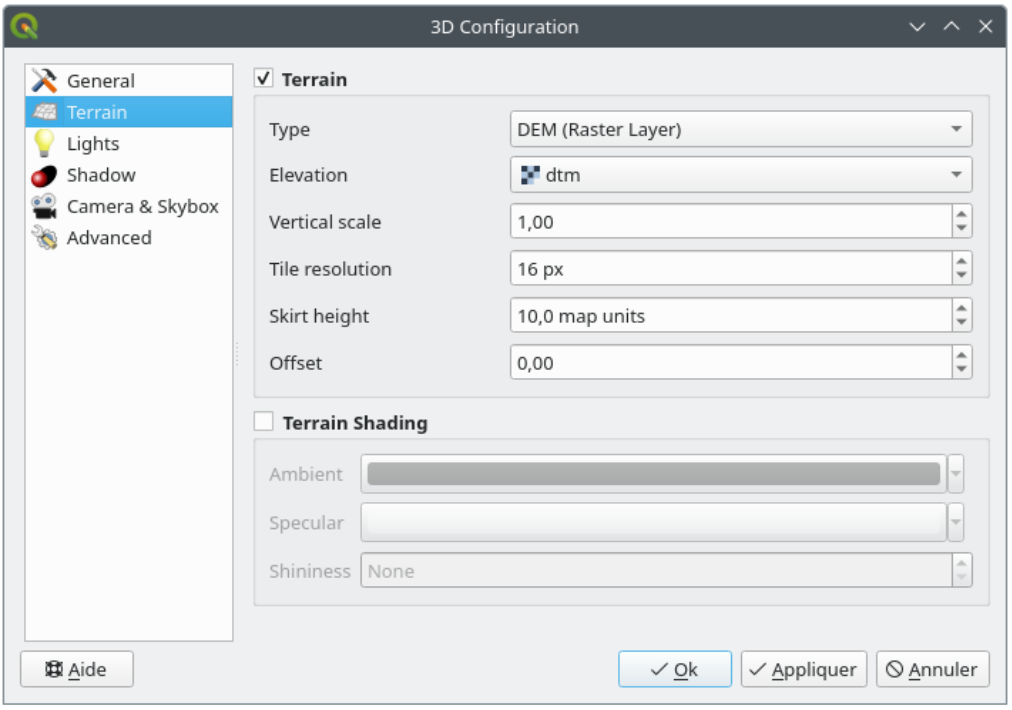


Fig. 7.24: The 3D Map Terrain Configuration dialog

- *Terrain*: Before diving into the details, it is worth noting that the terrain in a 3D view is represented by a hierarchy of terrain tiles and as the camera moves closer to the terrain, existing tiles that do not have sufficient details are replaced by smaller tiles with more details. Each tile has mesh geometry derived from the elevation raster layer and texture from 2D map layers.
  - The elevation terrain *Type* can be:
    - a *Flat terrain*
    - a loaded *DEM (Raster Layer)*
    - an *Online* service, loading elevation tiles produced by Mapzen tools – more details at https://registry.opendata.aws/terrain-tiles/
    - a loaded *Mesh* dataset
    - a *Quantized Mesh* layer from the project
  - *Elevation*: Raster or mesh layer to be used for generation of the terrain. The raster layer must contain a band that represents elevation. For a mesh layer, the Z values of the vertices are used.
  - *Vertical scale*: Scale factor for vertical axis. Increasing the scale will exaggerate the height of the landforms.
  - *Tile resolution*: How many samples from the terrain raster layer to use for each tile. A value of 16px means that the geometry of each tile will consist of 16x16 elevation samples. Higher numbers create more detailed terrain tiles at the expense of increased rendering complexity.
  - *Skirt height*: Sometimes it is possible to see small cracks between tiles of the terrain. Raising this value will add vertical walls ("skirts") around terrain tiles to hide the cracks.
  - *Offset*: moves the terrain up or down, e.g. to adjust its elevation with respect to the ground level of other objects in the scene.

    This can be useful when there is a discrepancy between the height of the terrain and the height of layers in your scene (e.g. point clouds which use a relative vertical height only). In this case adjusting the terrain

elevation manually to coincide with the elevation of objects in your scene can improve the navigation experience.

- When a mesh layer is used as terrain, you can configure the *Triangles settings* (wireframe display, smooth triangles, level of detail) and the *Rendering colors settings* (as a uniform color or *color ramp based*). More details in the *Mesh layer 3D properties* section.
- *Terrain shading*: Allows you to choose how the terrain should be rendered:
  - Shading disabled - terrain color is determined only from map texture
  - Shading enabled - terrain color is determined using Phong's shading model, taking into account map texture, the terrain normal vector, scene light(s) and the terrain material's *Ambient* and *Specular* colors and *Shininess*

Lights

From the *Lights* tab, press the menu to add

- up to eight *Point lights*: emits light in all directions, like a sphere of light filling an area. Objects closer to the light will be brighter, and objects further away will be darker. A point light has a set position (*X*, *Y* and *Z*), a *Color*, an *Intensity* and an *Attenuation*
- up to four *Directional lights*: mimics the lighting that you would get from a giant flash light very far away from your objects, always centered and that never dies off (e.g. the sun). It emits parallel light rays in a single direction but the light reaches out into infinity. A directional light can be rotated given an *Azimuth*, have an *Altitude*, a *Color* and an *Intensity*.

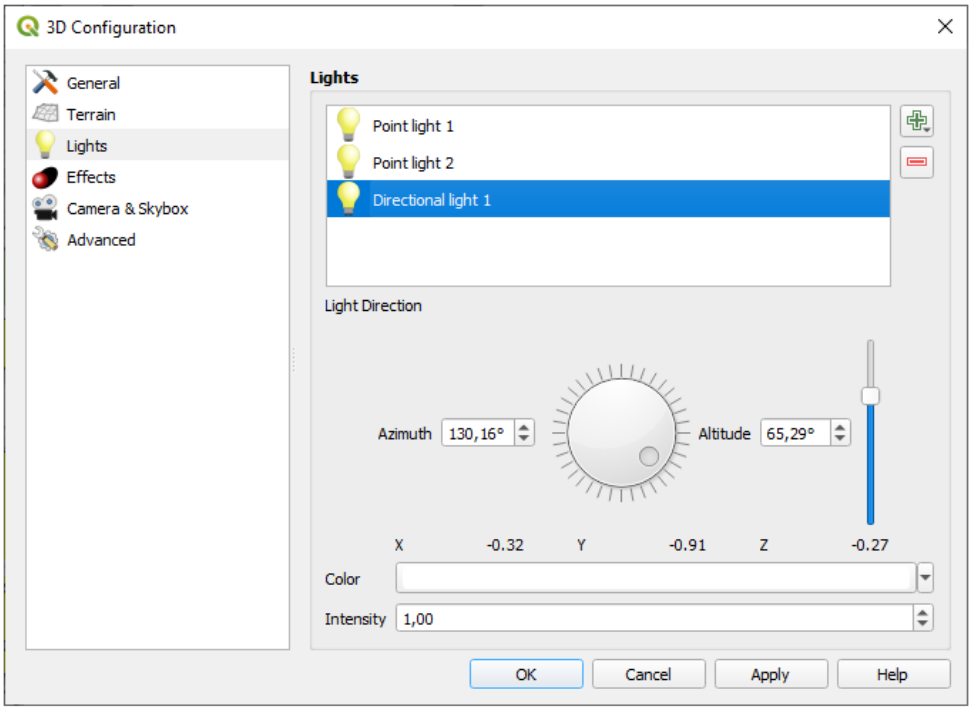


Fig. 7.25: The 3D Map Lights Configuration dialog

Effects

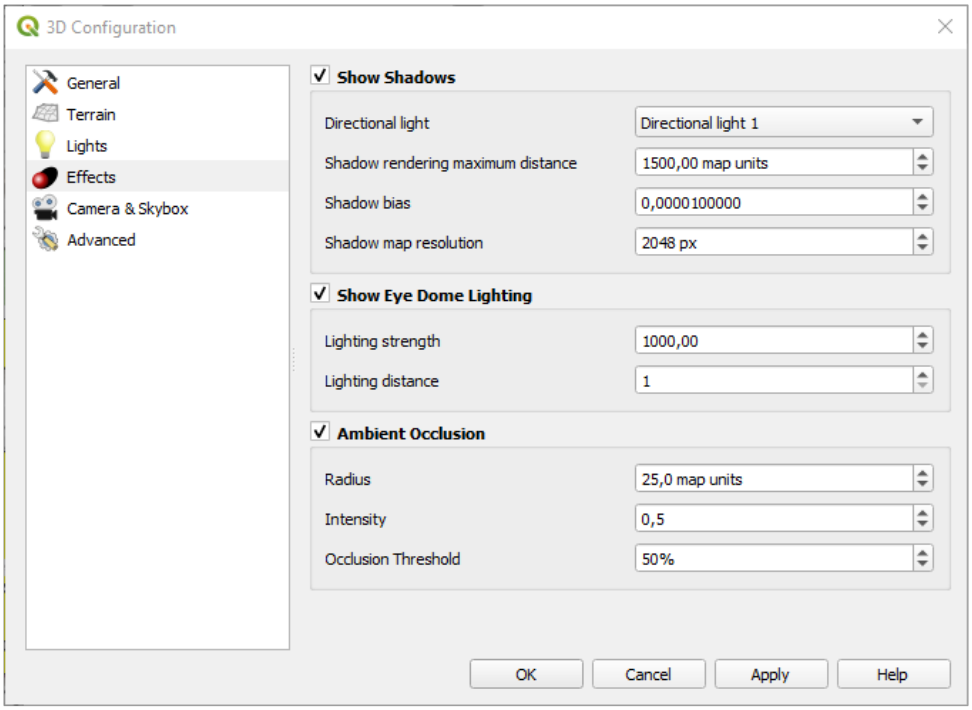


Fig. 7.26: The 3D Map Effects Configuration dialog

- Check *Show shadows* to display shadows within your scene, given:
  - a *Directional light*
  - a *Shadow rendering maximum distance*: to avoid rendering shadow of too distant objects, particularly when the camera looks up along the horizon
  - a *Shadow bias*: to avoid self-shadowing effects that could make some areas darker than others, due to differences between map sizes. The lower the better
  - a *Shadow map resolution*: to make shadows look sharper. It may result in less performance if the resolution parameter is too high.
- *Show Eye Dome Lighting* (EDL): a post processing effect which enhances depth perception. Each pixel's depth (distance off the camera) is compared to its neighboring pixels' depth and gets highlighted according to that depth difference, making the edges stand out. Affects the whole scene and can be combined with *Screen Space Ambient Occlusion*. Following parameters can be controlled:
  - *Lighting strength*: increases the contrast, allowing for better depth perception
  - *Lighting distance*: represents the distance of the used pixels off the center pixel and has the effect of making edges thicker.
- Add screen-space *Ambient Occlusion* (SSAO): a post processing effect which also enhances depth perception by applying a darker shading to areas which are less exposed to ambient lighting. Affects the whole scene and can be combined with *Eye dome Lighting*. Following parameters can be controlled:
  - *Radius*: how far we will reach to calculate ambient occlusion
  - *Intensity*: how strong the effect should be (higher values make things darker)
  - *Occlusion threshold*: how many neighboring points need to be occluded for the effect to appear (lower values than 50% will make the output darker, but possibly providing greater range of occlusion)

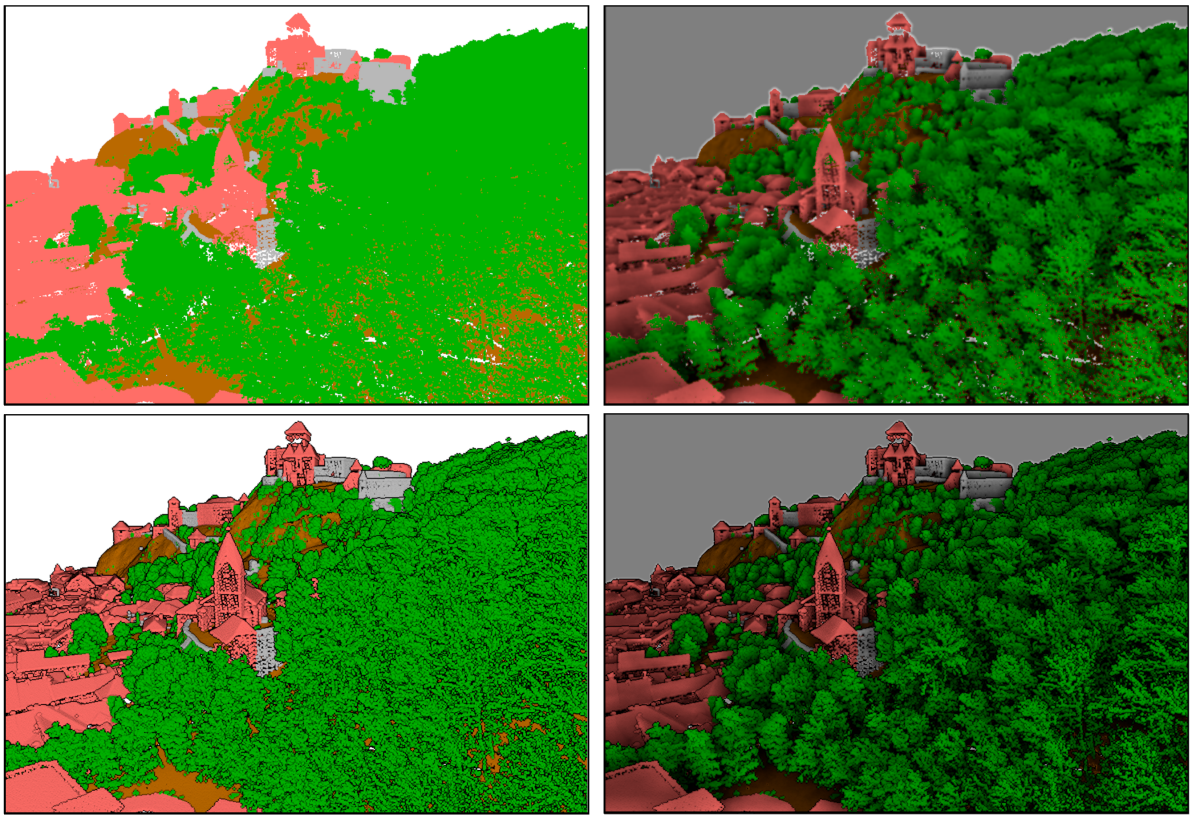


Fig. 7.27: Rendering Point clouds in 3D map using Eye Dome Lighting (EDL) and/or Screen-Space Ambient Occlusion (SSAO)

From top, left to right: No effect – SSAO only – EDL only – SSAO and EDL

Camera & Skybox

In this tab, you can control different parameters like camera, 3D axis, navigation synchronization and skybox.

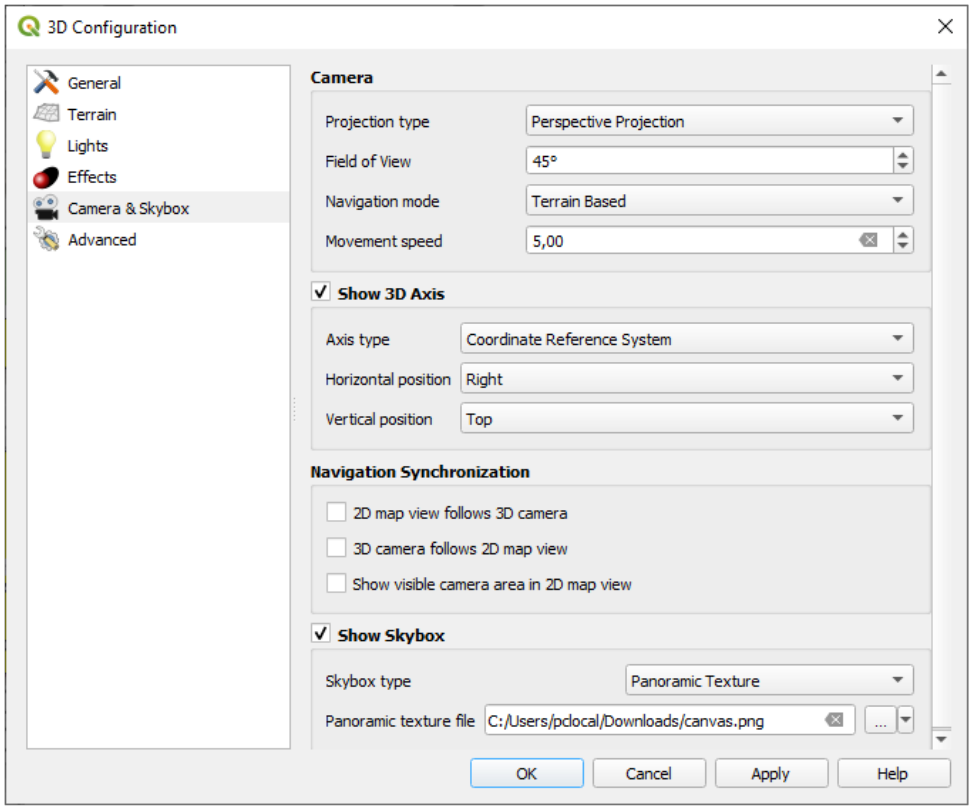


Fig. 7.28: The 3D Map Camera Configuration dialog

- The *Camera* parameter group overrides some *default camera settings* made in the *Settings* ► *Options* ► *3D* dialog.
- Check *Show 3D Axis* to enable 3D axis tool. This parameter group allows to set the axis type and its position.
  - With the *Coordinate Reference System* type an orthogonal axis will be represented.
  - With the *Cube* type, a 3D cube will be represented. The cube faces can be used to change the camera view: for example, click on the *north* face to set the camera to see from the north.

Tip

Right-click the 3D axis to quickly set its position and type, and the camera view.

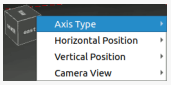


Fig. 7.29: The 3D Axis context menu

- The *Navigation Synchronization* parameter group adds options to synchronize 2D view with 3D camera position or 3D camera position with 2D view or bi directional synchronization. The last option displays the extent visible from the 3D camera over the 2D map view.
- Check *Show skybox* to enable skybox rendering in the scene. The skybox type can be:
  - *Panoramic texture*, with a single file providing sight on 360°
  - *Distinct faces*, with a texture file for each of the six sides of a box containing the scene

Texture image files of the skybox can be files on the disk, remote URLs or embedded in the project (*more details*).

Advanced

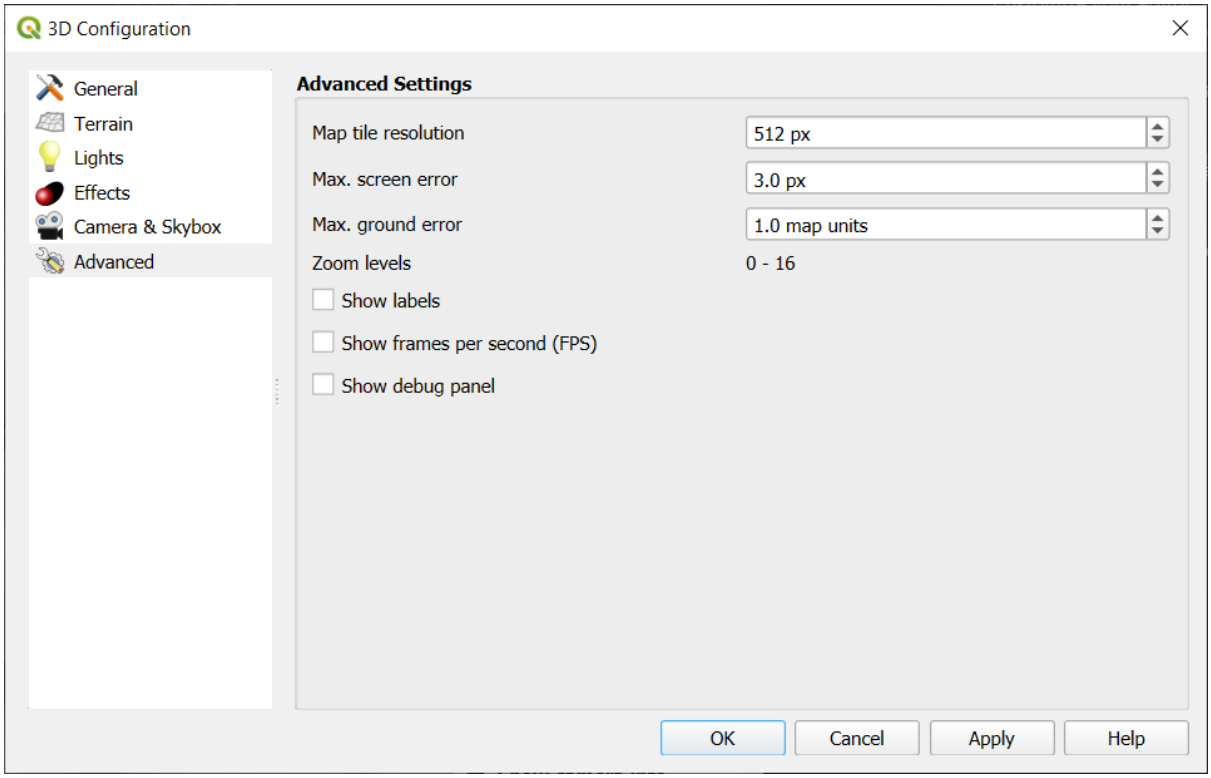


Fig. 7.30: The 3D Map Advanced Configuration dialog

- *Map tile resolution*: Width and height of the 2D map images used as textures for the terrain tiles. 256px means that each tile will be rendered into an image of 256x256 pixels. Higher numbers create more detailed terrain tiles at the expense of increased rendering complexity.
- *Max. screen error*: Determines the threshold for swapping terrain tiles with more detailed ones (and vice versa) - i.e. how soon the 3D view will use higher quality tiles. Lower numbers mean more details in the scene at the expense of increased rendering complexity.
- *Max. ground error*: The resolution of the terrain tiles at which dividing tiles into more detailed ones will stop (splitting them would not introduce any extra detail anyway). This value limits the depth of the hierarchy of tiles: lower values make the hierarchy deep, increasing rendering complexity.
- *Zoom levels*: Shows the number of zoom levels (depends on the map tile resolution and max. ground error).
- ☐ *Show labels*: Toggles map labels on/off
- ☐ *Show frames per second (FPS)*
- ☐ *Show debug panel*: side panel which displays all debug information about the 3D map view:
  - ☐ *Show map tile info*: Include border and tile numbers for the terrain tiles (useful for troubleshooting terrain issues)
  - ☐ *Show bounding boxes*: Show 3D bounding boxes of the terrain tiles (useful for troubleshooting terrain issues)
  - ☐ *Show camera's view center*

  - *Show camera's rotation center*
  - *Show light sources*: shows a sphere at light source origins, allowing easier repositioning and placement of light sources relative to the scene contents
  - *Stop scene updates*
  - *Show debug overlay*: visual overlay which displays some useful debugging and profiling information. This allows in particular to quickly see the frame graph and the scene graph
  - *Debug Shadow Map*: renders the scene as a red-black image from the point of view of the light used for shadows (for troubleshooting). The widget is set with a proportional *Size* to the 3D map view's, and docked in a *Corner*.
  - *Debug Depth Map*: renders the scene's depth map as an image with nearer pixels being darker (for troubleshooting). The widget is set with a proportional *Size* to the 3D map view's, and docked in a *Corner*.
  - *Show camera info*:
    * *Far plane*: controls how far from the camera rendering stops
    * *Near plane*: controls how close to the camera rendering starts
    * *Camera X/Y/Z pos*: sets the camera's position in 3D space
    * *Looking at X/Y/Z*: sets the target point the camera is looking at

Note

When your 3D map view is open in a standalone window, you can use `CTRL + SHIFT + d` keyboard shortcut to access the debug panel.

7.2.2 Navigation options

To explore the map view in 3D:

- Tilt the terrain (rotating it around a horizontal axis that goes through the center of the window)
  - Press the Tilt up and Tilt down tools
  - Press `Shift` and use the up/down keys
  - Drag the mouse forward/backward with the middle mouse button pressed
  - Press `Shift` and drag the mouse forward/backward with the left mouse button pressed
- Rotate the terrain (around a vertical axis that goes through the center of the window)
  - Turn the compass of the navigation widget to the watching direction
  - Press `Shift` and use the left/right keys
  - Drag the mouse right/left with the middle mouse button pressed
  - Press `Shift` and drag the mouse right/left with the left mouse button pressed
- Change the camera position (and the view center), moving it around in a horizontal plan
  - Drag the mouse with the left mouse button pressed, and the Camera control button enabled
  - Press the directional arrows of the navigation widget
  - Use the up/down/left/right keys to move the camera forward, backward, right and left, respectively
- Change the camera altitude: press the `Page Up`/`Page Down` keys

- Change the camera orientation (the camera is kept at its position but the view center point moves)
  - Press `Ctrl` and use the arrow keys to turn the camera up, down, left and right
  - Press `Ctrl` and drag the mouse with the left mouse button pressed
- Zoom in and out
  - Press the corresponding  Zoom In and  Zoom Out tools of the navigation widget
  - Scroll the mouse wheel (keep `Ctrl` pressed results in finer zooms)
  - Drag the mouse with the right mouse button pressed to zoom in (drag down) and out (drag up)

To reset the camera view, click the  Zoom Full button on the top of the 3D canvas panel.

7.2.3 Creating an animation

An animation is based on a set of keyframes - camera positions at particular times. To create an animation:

1. Toggle on the  Animations tool, displaying the animation player widget
2. Click the  Add keyframe button and enter a *Keyframe time* in seconds. The *Keyframe* combo box now displays the time set.
3. Using the navigation tools, move the camera to the position to associate with the current keyframe time.
4. Repeat the previous steps to add as many keyframes (with time and position) as necessary.
5. Click the  button to preview the animation. QGIS will generate scenes using the camera positions/rotations at set times, and interpolating them in between these keyframes. Various *Interpolation* modes for animations are available (eg, linear, inQuad, outQuad, inCirc… – more details at https://doc.qt.io/qt-5/qeasingcurve.html#EasingFunction-typedef).

   The animation can also be previewed by moving the time slider. Keeping the *Loop* box checked will repeatedly run the animation while clicking stops a running animation.

Click  Export animation frames to generate a series of images representing the scene. Other than the filename *Template* and the *Output directory*, you can set the number of *Frames per second*, the *Output width* and *Output height*.

7.2.4 3D vector layers

A vector layer with elevation values can be shown in the 3D map view by checking *Enable 3D Renderer* in the *3D View* section of the vector layer properties. A number of options are available for controlling the rendering of the 3D vector layer.

7.3 Elevation Profile View

The *Elevation Profile* panel is a plotting tool for side view, for visualizing elevation data along a line. It supports vector, raster, mesh and point cloud layers. Data can be of 2D or 3D type.

To add an elevation profile view, go to *View* ►  *Elevation Profile* menu. You can add as many profile views as you want, and they can be docked, piled on top of each other, or floating.

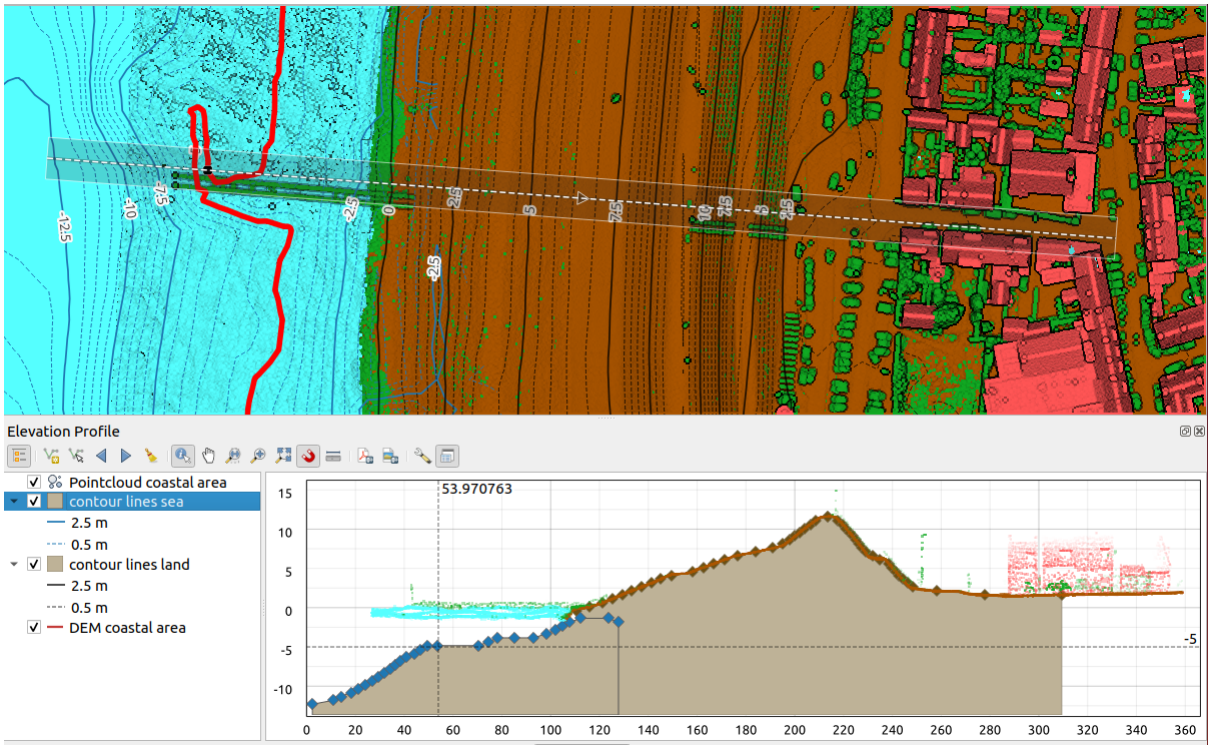


Fig. 7.31: Elevation Profile dialog embedded below main map canvas
Figure shows a coastline with a town behind a sea-dike. A larger tolerance applied to the elevation profile line returns more point cloud features.

7.3.1 The interface

At the top of the *Elevation Profile* panel, a toolbar provides you with the following tools:

Table 7.5: Elevation Profile View toolbar

Tool	Short-cut	Description
Add Layers	hold `Ctrl` while drag-and-drop	Allows you to add new layers to a plot.
Show Layer Tree		Shows or hides a list of project layers to configure rendering in the profile view.
Capture Curve		Draws interactively a line over the map canvas to represent the profile curve.
Capture Curve From Feature		Picks an existing line feature on the map canvas and generate a profile curve along that line.
Nudge Left	`Ctrl+A]`	Allows you to slowly move the capture line across the map to the left (e.g. to find the optimal profile line based on the elevation).
Nudge Right	`Ctrl+A]`	Allows you to slowly move the capture line across the map to the right (e.g. to find the optimal profile line based on the elevation).
Clear		Removes the profile line and any plot displayed in the *Elevation Profile* view

continues on next page

Table 7.5 – continued from previous page

Tool	Short-cut	Description
Identify Features		Identifies features in the plot canvas via either a single click, or click-and-drag rectangle. Results are shown in the standard *Identify Results* dock.
Pan	`Space`	Click and drag to pan the plot canvas. Can also be done with the middle mouse button.
Zoom X Axis		Zooms in/out along the horizontal axis, keeping the vertical ratio
zoom	`Ctrl+Sp`	Click or click-and-drag a rectangle over the plot to zoom in. Press `Alt` and click to instead zoom out.
Zoom Full		Zooms the *Elevation Profile* view to the extent of the capture line
Enable Snapping		Allows to snap to the edge or vertex of the plot features in the profile view. Convenient for accurate retrieval of coordinates or distance measurements.
Measure Distances		Measures horizontal and vertical distances
Export As PDF		Exports plots to PDF (as high quality vector objects)
Export As Image		Exports plots to several image formats
Export Results		Exports plots as 3D features, 2D Profile and Distance/Elevation Table. (The results can be saved as DXF files, CSV files or any of the standard writable vector spatial formats). More details at *Exporting from the elevation profile*.
Options ►		Provides access to configuration settings of the profile elevation line.
► *Lock distance/elevation scales*		Ensures that the horizontal and vertical scales are always kept equal (so that e.g. a 45° slope will appear as a 45° slope in the profile view).
► *Distance units*		Allows to render distances in the profile chart with units other than the map canvas units.
► *Tolerance*		Sets how far from the actual profile line a feature (vector point, line or polygon, point cloud) can reside to be included in the results. Uses the map units and is ignored by other layer or geometry types. For some layer types (e.g. points, lines, polygons), a custom tolerance can be set in the layer's Elevation properties tab, which replaces the global tolerance and allows more control over feature inclusion based on distance to the profile line.
► *Rename Profile…*		Allows to rename the profile view.
Dock Elevation Profile View		Switch between docked and floating status of the view

In the bottom left, a copy of the *Layers* panel can be displayed pushing the *Show Layer Tree* button. This is however an independent widget, with its own set of visible layers, in a custom stack order. It allows you to control layers rendering and behavior within the plot canvas:

- Tick the box next to the layer name to set whether it should be rendered in the plot canvas
- Drag-and-drop layers up or down to change the order of the layers
- style rendering of layers in the profile view: double-click a layer or right-click and select *Properties…* to open the layer's *Elevation* properties tab for configuration. A summary of elevation settings is displayed as tooltip when hovering over the layer.

On the right of the layer tree, the plot canvas is the main place you can preview the elevation profile of the enabled layers. It is based on a graduated grid in which the horizontal axis displays the length of the profile line and the vertical axis displays the Z elevation of the observed features. It also allows a set of interactions such as zooming, panning, measuring, identifying features, … using the tools at the top.

7.3.2 Creating an elevation profile

To create a profile view, you can:

1. Go to *View* ►  *Elevation Profile* menu. The *Elevation profile* panel opens.
2. Create the profile line along which the terrain and the features will be rendered. Select a drawing tool:
   -  Capture Curve: click left over the main map canvas to add vertices and click right to finalize a line that will be used as profile line
   - or  Capture Curve From Feature: click a line feature on the map canvas to select it as the profile line. If multiple features are present at the clicked point then a popup menu will appear allowing you to select among them.

   All the line digitizing capabilities such as the *snapping options*, *tracing*, *digitizing techniques* or the advanced *digitizing panel* are available for use.

   The plot canvas may start rendering some features.
3. The next step is to configure the elevation properties of the layers you want to visualize.
   1. Push on the  Show Layer Tree button to display the list of layers. By default, not all the layers of the project are loaded and referenced in the profile tool; e.g., raster layers which are not marked as having elevation data in their properties are ignored. Clicking the  Add layers button will show a filtered list of possible layers which can be added to the plot, but which currently aren't in the plot. Applying the dialog with selected layers will automatically mark them as having elevation data and immediately add them to the plot.

      Tip

      Holding `Ctrl` key, it is possible to drag and drop additional layers from the main layer tree to one in the elevation profile tool.

   2. Toggle visibility of the layers you are interested in. These are the only ones rendered in the profile view and selected layers can be different from the main *Layers* panel's.
   3. Double-click a layer name or right-click and select *Properties*. The *Elevation* properties tab of the layer opens. This is the place you configure how each feature or terrain should render on the profile view. Available properties depend on the layer type:
      - *Raster Elevation Properties*
      - *Vector Elevation Properties*
      - *Point Cloud Elevation Properties*
      - *Mesh Elevation Properties*

      The profile view starts rendering terrain or features of active layers crossing the given profile curve, as soon as they have elevation properties configured.
4. Under  Options drop-down menu, you can set the *Tolerance* value. This value is used to create a flat buffer around the elevation profile line, visible in the main map canvas. Any visible feature (point, line, polygon, etc.) overlapping that buffer will be captured in the plot canvas.

Note

Limitations with polygon extrusion

Geometry extrusion can be set in the  *Elevation* properties of a layer, and rendered in the profile view. When tolerance is enabled, it is however not trivial to render extruded polygons, thus, for now, polygon extrusion is ignored.

7.3.3 Interacting with the profile Views

When an elevation profile line is created, the plot canvas zooms to its full extent. On the X-axis you can see the length of the profile and, on the Y-axis the height range between minimum and maximum height captured, both in map units.

When you move the mouse pointer in the elevation profile view, you will see two crossing dot lines:

- the vertical line shows the height information
- the horizontal line shows the distance from the beginning of the elevation profile Line

When you move the mouse pointer in the elevation profile canvas, you can also see a black dot move along the elevation profile line on the main map canvas. At the middle of the profile line, an arrow indicates its direction.

As for the main map canvas, QGIS provides means to navigate on the plot canvas:

-  Pan is used to move the elevation profile extent in any direction you want. Holding `Space` key while moving the mouse also shifts the plot canvas extent.
-  Zoom X Axis is used to zoom in along the horizontal axis, keeping the ratio of the vertical axis (the elevation) the same. Left click to stretch out the plot along the X axis, with the clicked point at the center of the axis, or drag a rectangle to stretch out the plot along the X axis to the rectangle width. Hold `Alt` while using  Zoom X Axis to zoom out along that axis.
-  Zoom is used to zoom in on a certain point (using left click), or to a certain extent (dragging a rectangle on the area). Hold `Alt` while using  Zoom to instead zoom out. In combination with the `Ctrl` key you can zoom in or out more smoothly.
-  Zoom Full is the default zoom level used at the beginning and shows the full extent of the profile line, with all returned features. Use it to reset the zoom level.

It is also possible to interact with the elements displayed in the plot canvas:

- Press  Enable snapping button to accurately catch points, vertices or edges of the features, for an accurate measurement or coordinates report.
-  Identify Features is used to identify features of the visible layers in the layer tree. You can drag a rectangle across several features in the profile view to query all of them. When compatible with the format (e.g. vector, point cloud), these features will be highlighted in the main map canvas.
-  Measurement Distances: click or select two points in the plot canvas to report the horizontal *Distance*, the *Elevation* and the *Total distance* between them, in map units.
-  Nudge Left and  Nudge right are used to shift the position of the elevation profile line in the map canvas to its left or right. The plot canvas will be redrawn, showing features and terrain overlapping the profile line buffer. The line is moved sideways using the *Tolerance* value in  Options menu.

Warning

Closing an elevation profile view or the project currently removes the view from the project.

For more details, give a look to QGIS elevation profile/cross section tool – a deep dive!, a presentation done by Nyall Dawson.

7.3.4 Exporting from the elevation profile

The top toolbar of the *Elevation profile* dialog provides a variety of tools to export an elevation profile, allowing you to reuse the plots in another software:

-  Export As PDF: Exports plots to PDF (as high quality vector objects). Following options are requested for generating the output file:
  - The *Page size*, selecting a page from a predefined list or setting custom dimensions, and the page orientation
  - The *Chart ranges* to control the extent of the chart to export:
    - on the X axis, the *Minimum distance* and *Maximum distance* from the profile curve starting point
    - On the Y axis, the *Minimum elevation* and *Maximum elevation*
  - The *Distance axis settings*: helps you configure the horizontal graduation of the grid over which the elevation profile is rendered. You can customize the *Label interval*, the *Major grid line interval* and the *Minor grid line interval*.
  - The *Elevation axis settings*, same as above, for vertical graduation
-  Export As Image: Exports plots to an image file format. Specific *Export options* such as the *Image width* and *Image height* are requested, along with the *Chart ranges*, *Distance axis settings* and *Elevation axis settings* previously mentioned.
-  Export results: Exports the profile results as a set of features of a vector layer (`.DXF`, `.CSV`, `.SHP`, `.GPKG`, …). Depending on the output format, multiple layers may be returned when the geometry types of exported features differ.
  - *Export 3D features*: Exports profile lines as 3D features, with elevation values taken from the elevation slices and stored in exported geometry Z values.
  - *Export 2D profile*: Exports profiles as 2D profile lines, with elevation stored in exported geometry Y dimension and distance in X dimension (as shown in the elevation profile widget).
  - *Export distance/elevation table*: Exports profiles as a table of sampled distance vs elevation values.

CHAPTER

EIGHT

GENERAL TOOLS

8.1 Context help

Whenever you need help on a specific topic, you can access the corresponding page in the current User Manual via the *Help* button available in most dialogs — please note that third-party plugins can point to dedicated web pages.

8.2 Panels

By default, QGIS provides many panels to work with. Some of these panels are described below while others may be found in different parts of the document. A complete list of default panels provided by QGIS is available via the *View* ► *Panels* ► menu and mentioned at *Panels*.

8.2.1 Layers Panel

The *Layers* panel (also called the `map legend`) lists all the layers in the project and helps you manage their visibility and shape the map. You can show or hide the panel by pressing `Ctrl+1`.

QGIS provides a variety of ways to add layers to a project:

- using the *Add* button from the dedicated data provider tab in the *Data source manager* dialog
- from QGIS *Browser panel* or *DB Manager*: double-click, drag-and-drop files and layers onto QGIS or use the contextual menu
- drag-and-drop files from the Operating System files explorer onto QGIS

In all these scenarios, you can open one or many layers at a time. New layers are added to the *Layers* panel:

1. if dropped over the *Layers* panel, at the exact location they are released
2. in the other cases, at a location that respects the global *behavior used when adding new layers* setting
3. and in case of multiple layers, they are sorted in a way that increases the chance of their stacking being logical and features being visible as most as possible, using the following logic (top to bottom):
    - vector point layers
    - vector line layers
    - vector polygon layers
    - point cloud layers
    - mesh layers
    - raster layers

At the top of the *Layers* panel, a toolbar allows you to:

-  Open the layer styling dock (F7): toggle the *Layer Styling* panel on and off.

- Add new group: see *Interact with groups and layers*
- Manage Map Themes: control visibility of layers and arrange them in different *map themes*.
- filter layers in the legend tree:
  - *Filter Legend by Map Content*: only the layers that are set visible and whose features intersect the current map canvas have their style rendered in the layers panel. Otherwise, a generic NULL symbol is applied to the layer. Based on the layer symbology, this is a convenient way to identify which kind of features from which layers cover your area of interest.
  - *Show Private Layers*: a convenient shortcut to display and interact with *private layers* in the *Layers* panel without modifying the project settings.
  - *Show Broken Layers Only*: only layers with broken data sources are displayed.
- Filter Legend by Expression: apply an expression to remove styles from the selected layer tree that have no feature satisfying the condition. This can be used to highlight features that are within a given area/feature of another layer. From the drop-down list, you can edit and clear the expression currently applied.
- Expand All or Collapse All layers and groups in the layers panel.
- Remove Layer/Group currently selected.

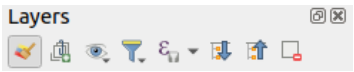


Fig. 8.1: Layer Toolbar in Layers Panel

> **Note**
>
> Tools to manage the layers panel are also available for map and legend items in print layouts

Configuring map themes

The Manage Map Themes drop-down button provides access to convenient shortcuts to manipulate visibility of the layers in the *Layers* panel:

- *Show All Layers*
- *Hide All Layers*
- *Show Selected Layers*
- *Hide Selected Layers*
- *Toggle Selected Layers*: changes the visibility of the first selected layer in the panel, and applies that state to the other selected layers. Also accesible through `Space` shortcut.
- *Toggle Selected Layers Independently*: changes the visibility status of each selected layer
- *Hide Deselected Layers*

Beyond the simple control of layer visibility, the Manage Map Themes menu allows you to configure **Map Themes** in the legend and switch from one map theme to another. A map theme is a **snapshot** of the current map legend that records:

- the layers set as visible in the *Layers* panel
- **and** for each visible layer:
  - the reference to the *style* applied to the layer
  - the visible classes of the style, ie the layer checked node items in the *Layers panel*. This applies to *symbologies* other than the single symbol rendering
  - the collapsed/expanded state of the layer node(s) and the group(s) it's placed inside

To create a map theme:

1. Check a layer you want to show
2. Configure the layer properties (symbology, diagram, labels…) as usual
3. Expand the *Style* ► menu at the bottom and click on *Add…* to store the settings as *a new style embedded in the project*

   Note

   A map theme does not remember the current details of the properties: only a reference to the style name is saved, so whenever you apply modifications to the layer while this style is enabled (eg change the symbology rendering), the map theme is updated with new information.

4. Repeat the previous steps as necessary for the other layers
5. If applicable, expand or collapse groups or visible layer nodes in the *Layers* panel
6. Click on the Manage Map Themes button on top of the panel, and *Add Theme…*
7. Enter the map theme's name and click *OK*

The new theme is listed in the lower part of the drop-down menu.

You can create as many map themes as you need: whenever the current combination in the map legend (visible layers, their active style, the map legend nodes) does not match any existing map theme contents as defined above, click on *Add Theme…* to create a new map theme, or use *Replace Theme* ► to update a map theme. You can rename the active map theme with *Rename Current Theme…* or use the *Remove Current Theme* button to delete it.

Map themes are helpful to switch quickly between different preconfigured combinations: select a map theme in the list to restore its combination. All configured themes are also accessible in the print layout, allowing you to create different map items based on specific themes and independent of the current main canvas rendering (see *Map item layers*).

Overview of the context menu of the Layers panel

At the bottom of the toolbar, the main component of the Layers panel is the frame listing all the layers added to the project, optionally organized in groups. A layer with a checked box next to it displays its contents overlapping the map canvas extent, unless a *scale-based visibility* is set. A layer can be selected and dragged up or down in the legend to change the Z-ordering. Z-ordering means that layers listed nearer the top of the legend are drawn over layers listed lower down in the legend. Also a layer or a group of layers can be dragged across several QGIS instances.

Note

The Z-ordering behavior can be overridden by the *Layer Order* panel.

Depending on the item selected in the panel, a right-click shows a dedicated set of options presented below. Some of these entries are not available when multiple selections are made (e.g., *Filter*, *Rename*, *Properties*).

Table 8.1: Contextual menus from *Layers* panel items

Option	Group	Vector Layer	Raster Layer	Mesh Layer	Point Cloud Layer	3D Lay
Zoom to Layer(s)/Group	✓	✓	✓	✓	✓	✓
Zoom to Selection		✓				
Show in Overview		✓	✓	✓	✓	✓
Show Feature Count		✓				
Show Label		✓		✓		
Copy Layer/Group	✓	✓	✓	✓	✓	✓
Rename Layer/Group	✓	✓	✓	✓	✓	✓
Zoom to Native Resolution (100%)			✓			
Stretch Using Current Extent			✓			
Update SQL Layer…		✓				
Execute SQL…		✓				
Edit Virtual Layer…		✓				
Add Group	✓					
Duplicate Layer		✓	✓	✓	✓	✓
Remove Layer/Group…	✓	✓	✓	✓	✓	✓
Move Out of Group		✓	✓	✓	✓	
Move to Top	✓	✓	✓	✓	✓	
Move to Bottom	✓	✓	✓	✓	✓	
Check and all its Parents		✓	✓	✓	✓	
Group Selected		✓	✓	✓	✓	
Open Attribute Table		✓				
Toggle Editing		✓		✓		
Current Edits ►		✓		✓		
Filter…		✓	✓		✓	
Change Data Source…		✓	✓	✓	✓	
Repair Data Source…		✓	✓	✓	✓	
Actions on selections ► (in edit mode)		✓				
► *Duplicate Feature*		✓				
► *Duplicate Feature and Digitize*		✓				
Set Layer Scale Visibility…		✓	✓	✓	✓	✓
Zoom to Visible Scale		✓	✓	✓	✓	
Layer CRS ►		✓	✓	✓	✓	✓
► *Set Project CRS from Layer*		✓	✓	✓	✓	✓
► *Set to..* (recent CRSs)				✓	✓	✓
► *Set Layer CRS…*		✓	✓	✓	✓	✓
Set Group CRS…	✓					
Set Group WMS Data…	✓					

continues on next pa

Table 8.1 – continued from previous page

Option	Group	Vector Layer	Raster Layer	Mesh Layer	Point Cloud Layer	3D Lay
Mutually Exclusive Group	☑					
Check and all its children (Ctrl-click)	☑					
Uncheck and all its children (Ctrl-click)	☑					
Make Permanent		☑				
Export ►	☑	☑	☑	☑	☑	☑
► *Save As…*			☑			
► *Save Features As…*		☑				
► *Save Selected Features As…*		☑				
► *Save As Layer Definition File…*	☑	☑	☑	☑	☑	☑
► *Save As QGIS Layer Style File…*		☑	☑	☑	☑	☑
Styles ►		☑	☑	☑	☑	☑
► *Copy Style*		☑	☑	☑	☑	☑
► *Paste Style*	☑	☑	☑	☑	☑	
► *Add…*		☑	☑	☑	☑	☑
► *Rename Current…*		☑	☑	☑	☑	☑
► *Edit symbol…*		☑				
► *Copy Symbol*		☑				
► *Paste Symbol*		☑				
Add Layer Notes…		☑	☑	☑	☑	☑
Edit Layer Notes…		☑	☑	☑	☑	
Remove Layer Notes		☑	☑	☑	☑	
Properties…		☑	☑	☑	☑	☑

For GRASS vector layers, Toggle editing is not available. See section *Digitizing and editing a GRASS vector layer* for information on editing GRASS vector layers.

Interact with groups and layers

Layers in the legend window can be organized into groups. There are different ways to do this:

1. Press the icon to add a new group. Type in a name for the group and press `Enter`. Now click on an existing layer and drag it onto the group.
2. Select more than one layer, then press the icon. The selected layers are automatically inserted into the new group.
3. Select some layers, right-click in the legend window and choose *Group Selected*. The selected layers will automatically be placed in a new group.

To move a layer out of a group, drag it out, or right-click on it and choose *Move Out of Group*: the layer is moved from the group and placed above it. Groups can also be nested inside other groups. If a layer is placed in a nested group, *Move Out of Group* will move the layer out of all nested groups.

To move a group or layer to the top of the layer panel, either drag it to the top, or choose *Move to Top*. If you use this option on a layer nested in a group, the layer is moved to the top in its current group. The *Move to Bottom* option follows the same logic to move layers and groups down.

The checkbox for a group will show or hide the checked layers in the group with one click. With `Ctrl` pressed, the checkbox will also turn on or off all the layers in the group and its sub-groups.

`Ctrl`-click on a checked / unchecked layer will uncheck / check the layer and all its parents.

Enabling the **Mutually Exclusive Group** option means you can make a group have only one layer visible at the same time. Whenever a layer within the group is set visible the others will be toggled not visible.

It is possible to select more than one layer or group at the same time by holding down the `Ctrl` key while clicking additional layers. You can then move all selected layers to a new group at the same time.

You may also delete more than one layer or group at once by selecting several items with the `Ctrl` key and then pressing `Ctrl+D`: all selected layers or groups will be removed from the layers list.

More information on layers and groups using indicator icon

In some circumstances, formatting or icons next to the layer or group in the *Layers* panel change to give more information about the layer/group. These elements are:

-  to indicate that the layer is in edit mode and you can modify the data
-  to indicate that the layer being edited has some unsaved changes
-  to indicate *a filter* applied to the layer. Hover over the icon to see the filter expression and double-click to update the query
-  to identify layers that are *required* in the project, hence non removable
-  to identify an *embedded group or layer* and the path to their original project file
-  to identify a layer whose data source was not available at the project file opening (see *Handling broken file paths*). Click the icon to update the source path or select *Repair Data Source…* entry from the layer contextual menu.
-  to remind you that the layer is a *temporary scratch layer* and its content will be discarded when you close this project. To avoid data loss and make the layer permanent, click the icon to store the layer in any of the GDAL vector formats supported by QGIS.
-  to identify a layer used in *offline editing mode*.
-  to identify a layer that has no/unknown CRS
-  for layers with coordinates stored in a coordinate reference system which is inherently low accuracy (requires the *corresponding setting* to be enabled)
-  to identify a temporal layer controlled by canvas animation
-  to identify a layer that has *notes* associated
- A grayed name, when the map canvas current scale is outside the layer's visibility scale range (as set in its *Rendering* properties). Select the contextual menu *Zoom to Visible Scale* option to zoom the map to the layer's nearest visibility scale bound.

Control layers rendering through grouping

Groups are a means of structuring layers within a tree in the project but they can also impact how their component layers are rendered, namely as a single flattened object during map renders.

The option for such a rendering is available within the *Layer Styling* panel whenever a group is selected. Under the Symbology tab, check *Render Layers as a Group* to enable a set of options to control the appearance of the child layers as a whole, instead of individual layers:

- *Opacity*: Features from child layers which are obscured by other child layers remain obscured, and the opacity applies to the "whole of group" only.

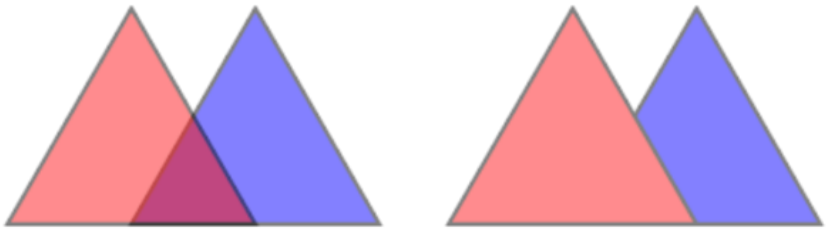


Fig. 8.2: Setting opacity on layers vs on a group
The image on the left shows two layers being rendered at 50% opacity (underlying features are visible, but semi-masked by the 50% red feature on top). The second image shows the result of setting the opacity on the group (parts of the blue underlying child layer is completely obscured by the red layer on top and then the result is rendered at 50% opacity).

- *Blend modes*: Just like opacity, setting a *blend mode* (like multiply, overlay, …) for an entire group results first in flattening features of child layers, with upper ones obscuring lower. The rendering is then obtained by blending the flat group and the layers sitting below the group.
  - When the child layers have blend modes assigned, it is applied before flattening but the scope is restricted to only affecting other child layers from that group, and not other layers sitting below the whole group.
  - Some more *blending modes* options are available for child layers in groups, in their *Symbology* tab which perform "clipping" style operations on other child layers during the render. You can e.g. clip the render of one layer's content by the content in a second "mask" layer.
- *Layer effects*: applies *effects* only to the flattened render of the child layers; So e.g. a drop shadow effect applied to the group would not be visible for obscured child layers.

When a group is set to *Render layers as a group*, then only the group will be shown in the *Layer Order* panel list. Group children will not be visible in this order list, as their ordering is determined by the placement of the group layer.

Editing layer style

From the *Layers* panel, you have shortcuts to change the layer rendering quickly and easily.

Right-click on a layer and select *Styles* ► in the list in order to:

- see the *styles* currently available for the layer. If you defined many styles for the layer, you can switch from one to another and your layer rendering will automatically be updated on the map canvas.
- copy part or all of the current style, and when applicable, paste a copied style from another layer
- *Rename current…* style
- *Add* a new style (which is actually a copy of the current one)
- or *Remove current* style (only when multiple styles are available).

> **Tip**
>
> **Quickly share a layer style**
>
> From the context menu, copy the style of a layer and paste it to a group or a selection of layers: the style is applied to all the layers that are of the same type (vector, raster, mesh, point cloud, …) as the original layer and, for vector layers, have the same geometry type (point, line or polygon).

When using a symbology based on features classification (e.g. *categorized*, *graduated* or *rule-based* for vector layers, or *classification* for point clouds), right-clicking a class entry in the *Layers* panels makes it possible to edit the visibility of the classes (and their features) and avoid (un)checking them one by one:

-  *Toggle Items*
-  *Show All Items*
-  *Hide All Items*

With vector layer, the contextual menu of a class leaf entry also gives access to:

-  *Select features*: selects in the layer all the features matching that class
-  *Show in attribute table*: opens an attribute table filtered to only the features matching that class
- update the *symbol color* using a **Color Wheel**. For convenience, the recently used colors are also available at the bottom of the color wheel.
- *Edit Symbol…*: opens the *Symbol Selector* dialog to change feature symbol (symbol, size, color…).
- *Copy Symbol*
- *Paste Symbol*

> **Tip**
>
> Double-clicking a class leaf entry also opens the *Symbol Selector* dialog.

8.2.2 Layer Styling Panel

The *Layer Styling* panel (also enabled with `Ctrl+3`) is a shortcut to some of the functionalities of the *Layer Properties* dialog. It provides a quick and easy way to define the rendering and the behavior of a layer, and to visualize its effects without having to open the layer properties dialog.

In addition to avoiding the blocking (or "modal") layer properties dialog, the layer styling panel also avoids cluttering the screen with dialogs, and contains most style functions (color selector, effects properties, rule edit, label substitution…): e.g., clicking color buttons inside the layer style panel causes the color selector dialog to be opened inside the layer style panel itself rather than as a separate dialog.

From a drop-down list of current layers in the layer panel, select an item and:

- Depending on the active item, set:
  -  *Symbology* for groups (see *Control layers rendering through grouping*)
  -  *Symbology*,  *Transparency*, and  *Histogram* properties for raster layer. These options are the same as in the *Raster Properties Dialog*.

  - *Symbology*, *Labels*, *Mask* and *3D View* properties for vector layer. These options are the same as in the *The Vector Properties Dialog* and can be extended by custom properties introduced by third-party plugins.
  - *Symbology* and *3D View* properties for mesh layer. These options are the same as in the *Mesh Dataset Properties*.
  - *Symbology*, *3D View* and *Elevation* properties for point cloud layer. These options are the same as in the *Point Clouds Properties*.
- Enable and configure *global map shading* properties
- Manage the associated style(s) in the *Style Manager* (more details at *Managing Custom Styles*).
- See the *History* of changes you applied to the layer style in the current project: you can therefore cancel or restore to any state by selecting it in the list and clicking *Apply*.

Another powerful feature of this panel is the *Live update* checkbox. Tick it to render your changes immediately on the map canvas: you no longer need to click the *Apply* button.

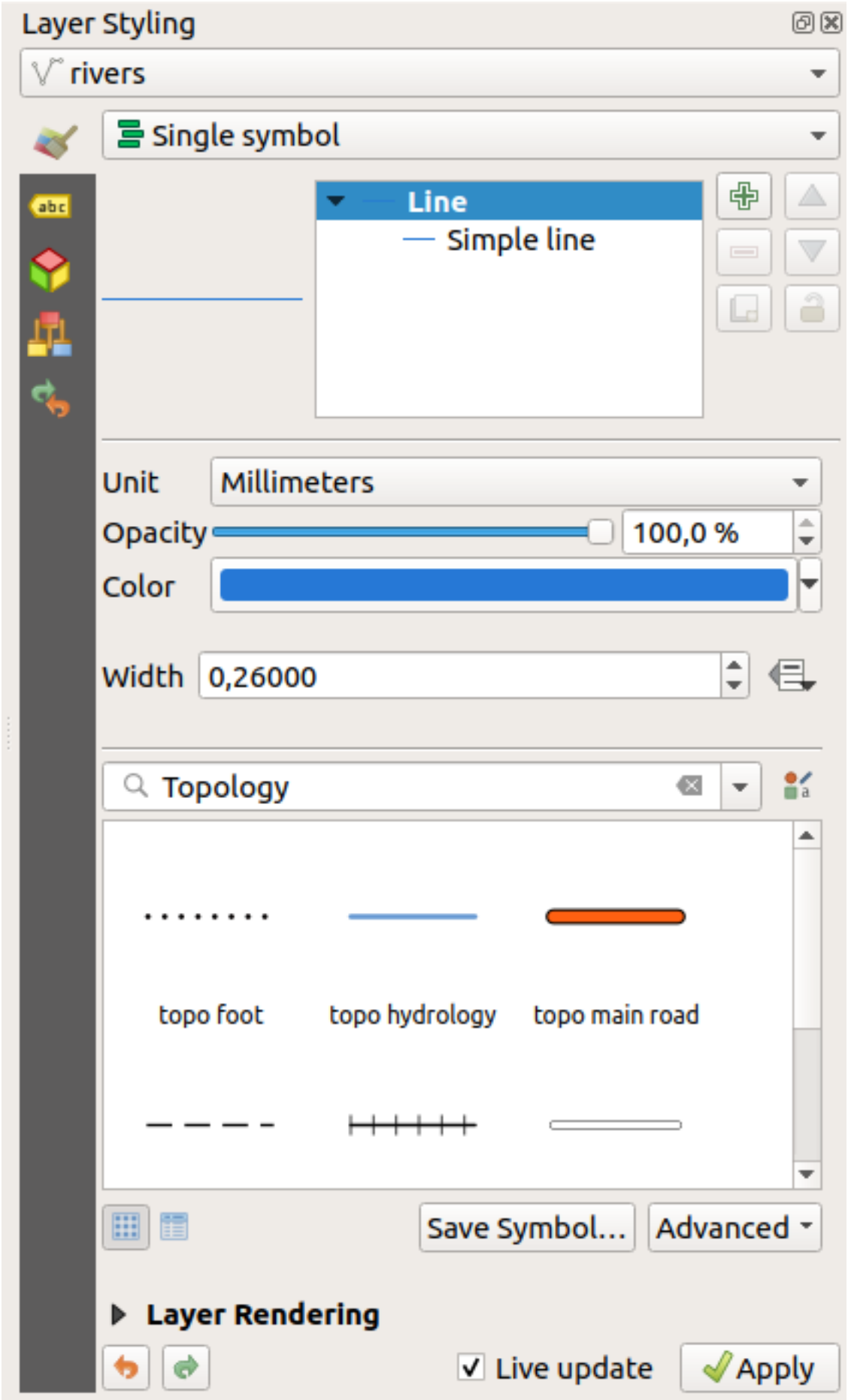


Fig. 8.3: Defining a layer's symbology from the layer styling panel

8.2.3 Layer Order Panel

By default, layers shown on the QGIS map canvas are drawn following their order in the *Layers* panel: the higher a layer is in the panel, the higher (hence, more visible) it'll be in the map view.

You can define a drawing order for the layers independent of the order in the layers panel with the *Layer Order* panel enabled in *View* ► *Panels* ► menu or with `Ctrl+9`. Check *Control rendering order* underneath the list of layers and reorganize the layers in the panel as you want. This order becomes the one applied to the map canvas. For example, in Fig. 8.4, you can see that the `airports` features are displayed over the `alaska` polygon despite those layers' respective placement in the Layers panel.

Unchecking *Control rendering order* will revert to default behavior.

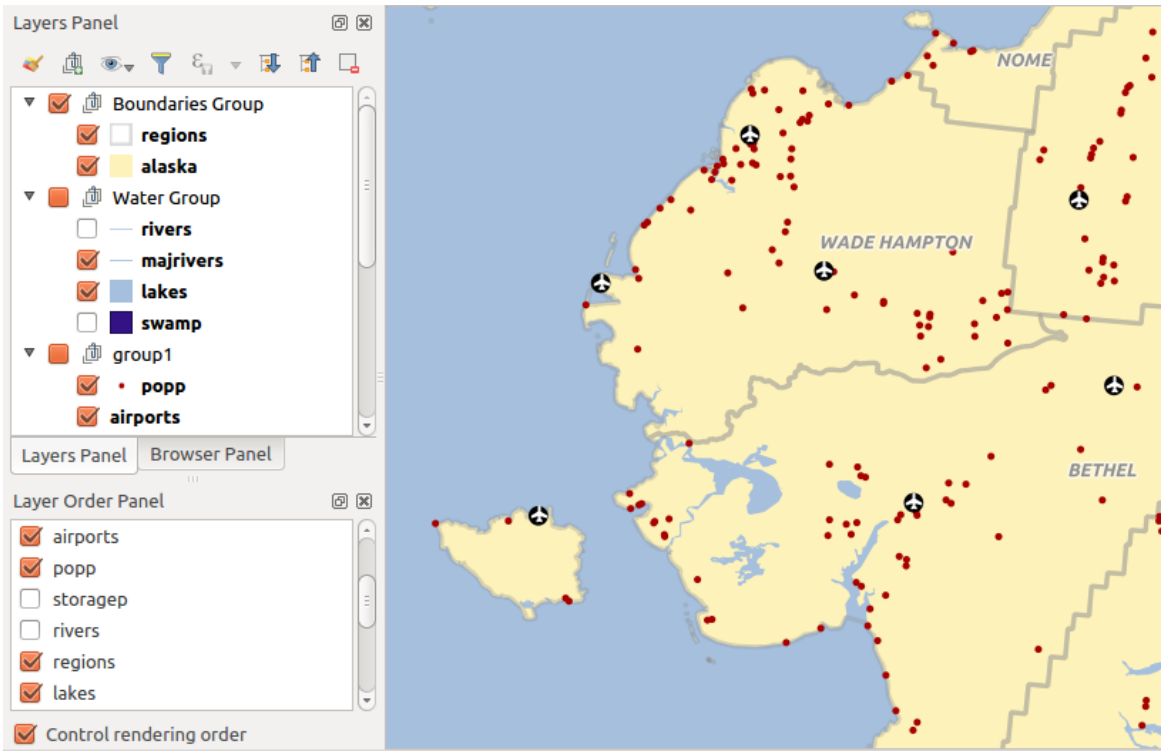


Fig. 8.4: Define a layer order independent of the legend

8.2.4 Overview Panel

The *Overview* panel (`Ctrl+8`) displays a map with a full extent view of some of the layers. The Overview map is filled with layers using the *Show in Overview* option from the *Layer* menu or in the layer contextual menu. Within the view, a red rectangle shows the current map canvas extent, helping you quickly to determine which area of the whole map you are currently viewing. If you click-and-drag the red rectangle in the overview frame, the main map view extent will update accordingly.

Note that labels are not rendered to the map overview even if the layers used in the map overview have been set up for labeling.

8.2.5 Log Messages Panel

When loading or processing some operations, you can track and follow messages that appear in different tabs using the *Log Messages Panel*. It can be activated using the most right icon in the bottom status bar.

8.2.6 Undo/Redo Panel

For each layer being edited, the *Undo/Redo* (`Ctrl+5`) panel shows the list of actions carried out, allowing you quickly to undo a set of actions by selecting the action listed above. More details at *Undo and Redo edits*.

8.2.7 Statistical Summary Panel

The *Statistics* panel (`Ctrl+6`) provides summarized information on any vector layer. This panel allows you to select:

- the vector layer to compute the statistics on: it can be selected from the top drop-down menu, or synced with the active layer in the *Layers* panel using the *Follow selected layer* checkbox at the bottom of the statistics drop-down list
- the field or *expression* to use: for each layer, the last entry is remembered and automatically computed upon layer reselection.
- the statistics to return using the drop-down button at the bottom-right of the dialog. Depending on the field's (or expression's values) type, available statistics are:

Table 8.2: Statistics available for each field type

Statistics	String	Integer	Float	Date
Count	☑	☑	☑	☑
Count Distinct Value	☑			☑
Count Missing value	☑	☑	☑	☑
Sum		☑	☑	
Mean		☑	☑	☑
Standard Deviation		☑	☑	
Standard Deviation on Sample		☑	☑	
Minimal value	☑	☑	☑	☑
Maximal value	☑	☑	☑	☑
Range		☑	☑	☑
Minority	☑	☑	☑	
Majority	☑	☑	☑	
Variety		☑	☑	
First Quartile		☑	☑	
Third Quartile		☑	☑	
Inter Quartile Range		☑	☑	
Minimum Length	☑			
Maximum Length	☑			
Mean Length	☑			

The statistical summary can be:

- returned for the whole layer or *Selected features only*
- copied to the clipboard and pasted as a table in another application
- recalculated using the button when the underlying data source changes (eg, new or removed features/fields, attribute modification)

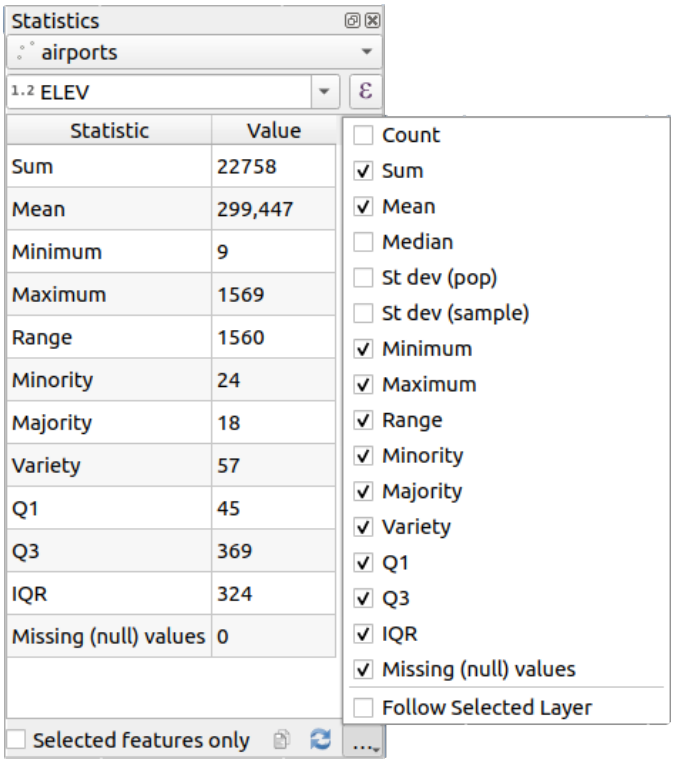


Fig. 8.5: Show statistics on a field

8.2.8 Debugging/Development Tools Panel

The *Debugging/Development Tools* panel (`F12`) provides a unified place for handling and debugging actions within QGIS. Available tools are organized under following tabs:

- *API Documentation*: embeds developer oriented documentation
- *Network Logger*: records and displays network requests and interactions
- *Query Logger*: logs SQL commands issued by data providers and duration of execution
- *Profiler*: provides load times for every actions in order to detect causes of slow down

> **Note**
>
> Plugin authors can extend the panel with custom tabs for debugging and developping their own plugins. This is done using `registerDevToolWidgetFactory` method.

API Documentation

The *API Documentation* tab can display either the PyQGIS API documentation or the Qt API documentation. These are embedded in the widget, allowing you to browse and search the docs without leaving QGIS. It is also the place you can display specific help page for objects or classes in your code sample; from the Python interactive console or text editor dialog, select or hover a text and press `F1` or right-click and select Contextual help to get its API help. If you wish to read the active page in the system web browser, then press Open in web browser.

Read *contextual help* for details on the widget configuration.

Network Logger

The *Network Logger* tab helps you record and display network requests, along with a whole load of useful details like request and reply status, header, errors, SSL configuration errors, timeouts, cache status, etc.

From its top toolbar, you can:

- *Record Log*: which will start or stop the logging.
- *Clear Log*: will clear the log history.
- *Save Log…*: will first show a big warning that the log is sensitive and should be treated as confidential and then allow you to save the log.
- Press the *Settings* drop-down menu to select whether to *Show Successful Requests*, *Show Timeouts* and *Show Replies Served from Cache*.
- *Disable cache*: will disable the cache so that every request has to be performed.
- *Filter requests* based on URL string subsets or request status

By right clicking on a request you can:

- *Open URL* which will open the URL in your default browser.
- *Copy URL*
- *Copy As cURL* to use it in the terminal.
- *Copy as JSON*: copies the tree values as a json string to the clipboard, for easy pasting in bug reports or for remote assistance.

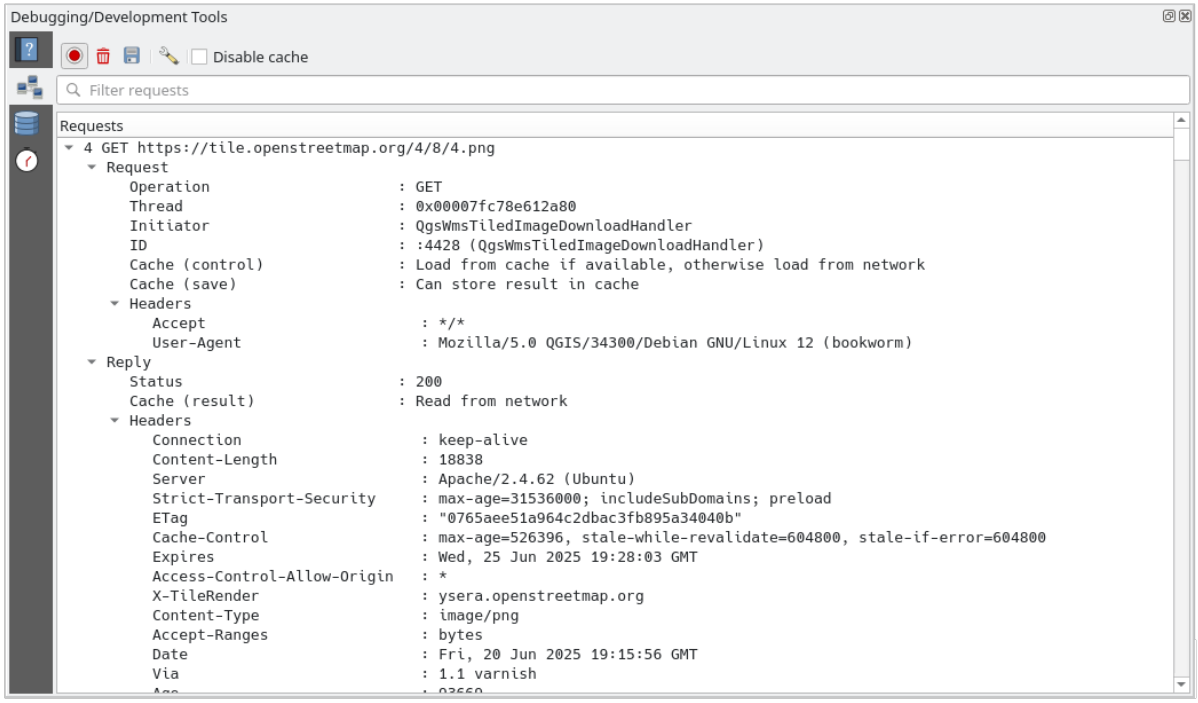


Fig. 8.6: Network Logger output for GET Request

Query Logger

The *Query Logger* is the place to log the SQL commands sent by the data provider and the connections API to the backend database, together with their execution time as measured by QGIS (i.e. in the client that sent the commands). This can be useful when investigating performances of a particular layer during debugging or development of a QGIS algorithm or plugin.

From its top toolbar, you can:

- *Record Log*: which will start or stop the logging.
- *Clear Log*: will clear the log history.
- *Save Log…*: will first show a big warning that the log is sensitive and should be treated as confidential and then allow you to save the log.
- *Filter queries* based on the query string subsets or details such as provider type, start time, initiator, …

Right-click on a reported query, and you can:

- *Copy SQL* command called by QGIS on the database
- *Copy as JSON*: copies the tree values as a json string to the clipboard, for easy pasting in bug reports or for remote assistance.

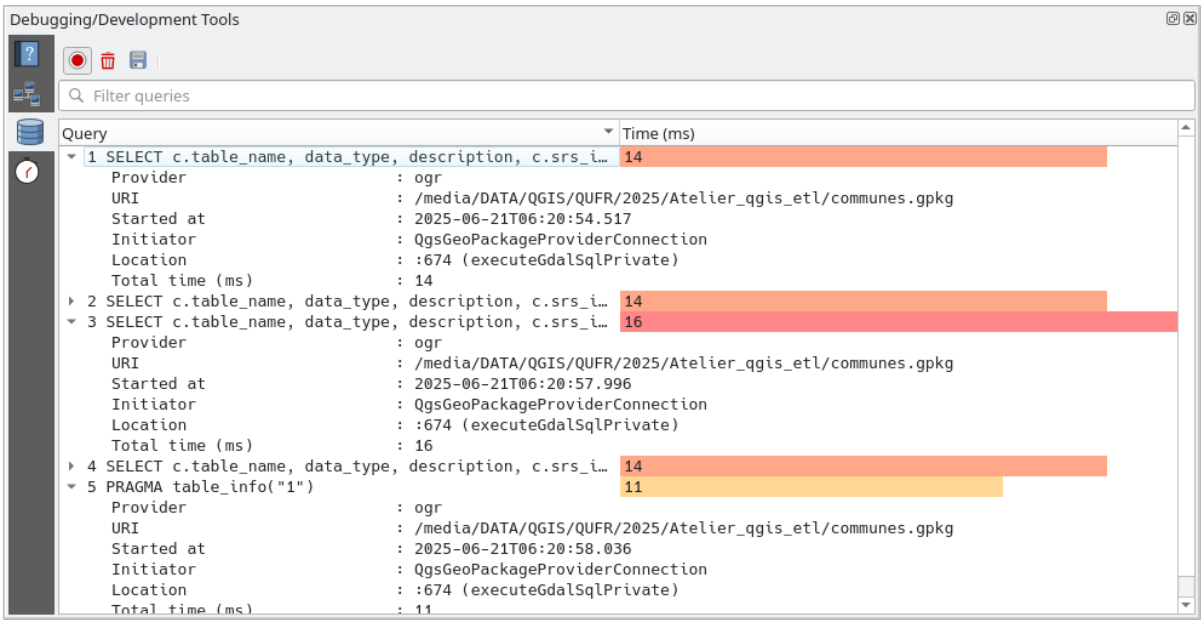


Fig. 8.7: Query Logger output

Profiler

The *Profiler* tab allows to get load times for every single operation involved in actions requested by the user. Depending on the context, these operations can be settings reading, menu, map canvas or 3D views creation, resolving map layers reference, bookmark or layout loading, … This helps identify causes of slow load times.

Default supported actions can be selected from the *Category* drop-down menu:

- *Startup* helps you diagnose (and fix) occasional long startup times of QGIS.
- *Project Load* allows you to get a breakdown of the various stages of project load, in order to identify the causes of slow project load times.
- *Map Render* tool allows you to identify pain points in your map rendering and track down exactly which layers are causing long map redraws.

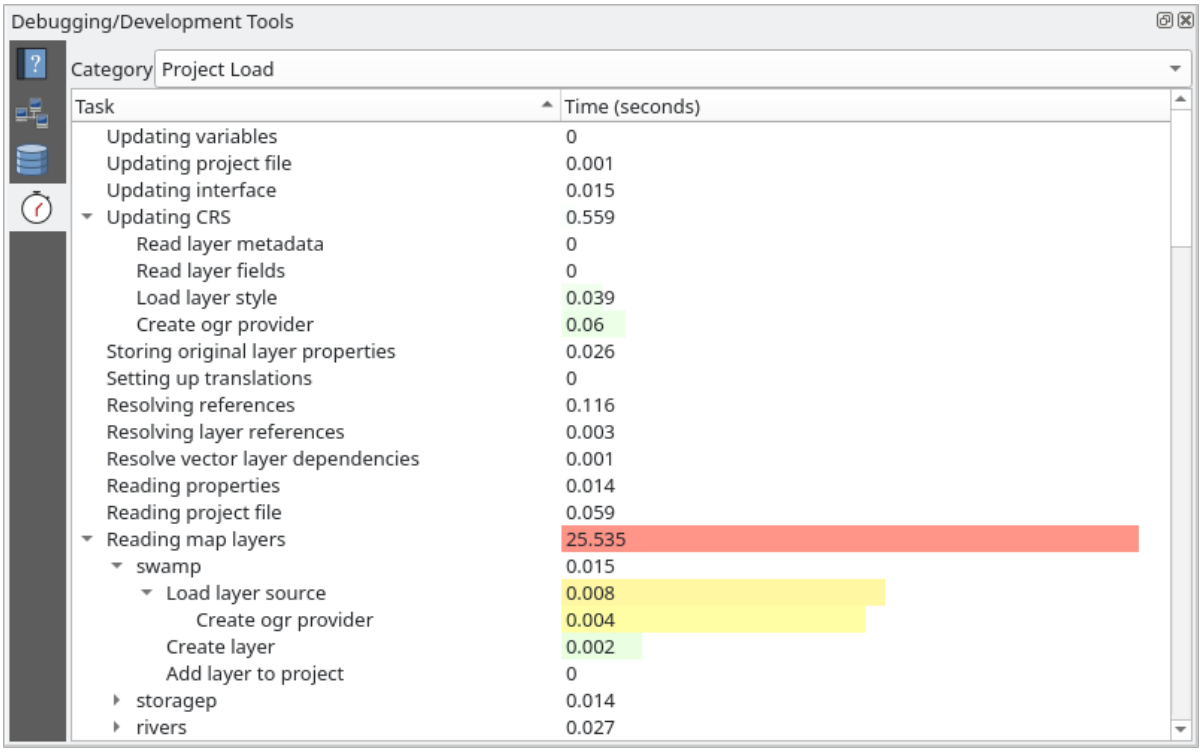


Fig. 8.8: Profiler for QGIS Startup

8.3 Embedding layers from external projects

Sometimes, you'd like to keep some layers in different projects, but with the same style. You can either create a *default style* for these layers or embed them from another project to save time and effort.

Embed layers and groups from an existing project has some advantages over styling:

- All types of layers (vector or raster, local or online…) can be added
- Fetching groups and layers, you can keep the same tree structure of the "background" layers in your different projects
- While the embedded layers are editable, you can't change their properties such as symbology, labels, forms, default values and actions, ensuring consistency across projects
- Modify the items in the original project and changes are propagated to all the other projects

If you want to embed content from other project files into your project, select *Layer* ► *Embed Layers and Groups*:

1. Click the … button to look for a project: you can see the content of the project (see Fig. 8.9)
2. Hold down `Ctrl` (or `Cmd`) and click on the layers and groups you wish to retrieve
3. Click *OK*

The selected layers and groups are embedded in the *Layers* panel and displayed on the map canvas. An icon is added next to their name for recognition and hovering over displays a tooltip with the original project file path.

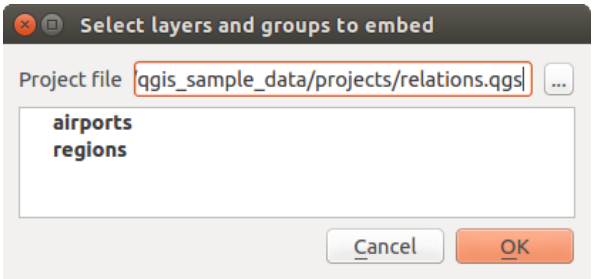


Fig. 8.9: Select layers and groups to embed

Like any other layer, an embedded layer can be removed from the project by right-clicking on the layer and clicking Remove.

Tip

Change rendering of an embedded layer

It's not possible to change the rendering of an embedded layer, unless you make the changes in the original project file. However, right-clicking on a layer and selecting *Duplicate* creates a layer which is fully-featured and not dependent on the original project. You can then safely remove the linked layer.

8.4 Interacting with features

8.4.1 Selecting features

QGIS provides several tools to select features on the map canvas. Selection tools are available in the *Edit* ► *Select* menu or in the *Selection Toolbar*.

Note

Selection tools work with the currently active layer.

Selecting manually on the map canvas

To select one or more features with the mouse, you can use one of the following tools:

- Select Features by area or single click
- Select Features by Polygon
- Select Features by Freehand
- Select Features by Radius

Note

Other than Select Features by Polygon, these manual selection tools allow you to select feature(s) on the map canvas with a single click.

Note

Use the  Select Features by Polygon tool to use an existing polygon feature (from any layer) to select overlapping features in the active layer. Right-click in the polygon and choose it from the context menu that shows a list of all the polygons that contain the clicked point. All the overlapping features from the active layer are selected.

Tip

Use the *Edit* ► *Select* ► *Reselect Features* tool to redo your latest selection. Very useful when you have painstakingly made a selection, and then click somewhere else accidentally and clear your selection.

While using the  *Select Feature(s)* tool, holding `Shift` or `Ctrl` (`Cmd` on macOS) toggles whether a feature is selected (i.e., either adds to the current selection or removes from it).

For the other tools, different behaviors can be performed by holding down:

- `Shift`: add features to the current selection
- `Ctrl`/`Cmd`: subtract features from the current selection
- `Ctrl+Shift`/`Cmd+Shift`: intersect with the current selection, i.e., only keep overlapping features from the current selection
- `Alt`: select features that are totally within the selection shape. Combined with `Shift` or `Ctrl`/`Cmd` keys, you can add or subtract features to/from the current selection.

Automatic selection

The other selection tools, most of them available from the *Attribute table*, perform a selection based on a feature's attribute or its selection state (note that attribute table and map canvas show the same information, so if you select one feature in the attribute table, it will be selected on the map canvas too):

-  Select Features By Expression… or press `Ctrl+F3` to select features using the expression dialog
-  Select Features By Value… or press `F3`
-  Deselect Features from All Layers or press `Ctrl+Alt+A` to deselect all selected features in all layers
-  Deselect Features from the Current Active Layer or press `Ctrl+Shift+A`
-  Select All Features or press `Ctrl+A` to select all features in the current layer
-  Invert Feature Selection to invert the selection in the current layer
-  Select by Location to select the features based on their spatial relationship with other features (in the same or another layer - see *Select by location*)
-  Select within distance to select features wherever they are within the specified maximum distance from referenced features - see *Select within distance*)

For example, if you want to find regions that are boroughs from `regions.shp` of the QGIS sample data, you can:

1. Use the  Select features using an Expression icon
2. Expand the *Fields and Values* group
3. Double-click the field that you want to query ("TYPE_2")

4. Click *All Unique* in the panel that shows up on the right
5. From the list, double-click 'Borough'. In the *Expression* editor field, write the following query:

```
"TYPE_2"  =  'Borough'
```

6. Click *Select Features*

From the expression builder dialog, you can also use *Function list* ► *Recent (Selection)* to make a selection that you have used before. The dialog remembers the last 20 expressions used. See *Expressions* for more information and examples.

Tip

Save your selection into a new file

Users can save selected features into a **New Temporary Scratch Layer** or a **New Vector Layer** using *Edit* ► *Copy Features* and *Edit* ► *Paste Features as* in the desired format.

Select Features By Value

This selection tool opens the layer's feature form allowing the user to choose which value to look for for each field, whether the search should be case-sensitive, and the operation that should be used. The tool has also autocompletes, automatically filling the search box with existing values.

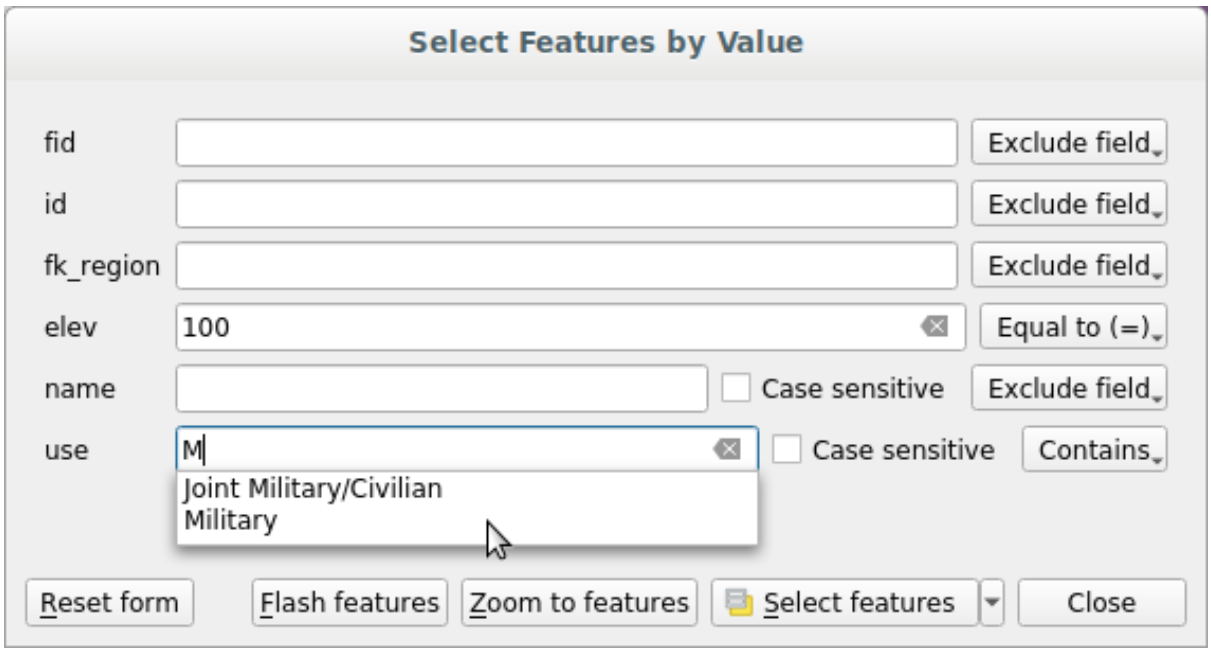


Fig. 8.10: Filter/Select features using form dialog

Alongside each field, there is a drop-down list with options to control the search behaviour:

Table 8.3: Query operators per data type

Field search option	String	Numeric	Date
Exclude Field from the search	☑	☑	☑
Equal to (=)	☑	☑	☑
Not equal to (≠)	☑	☑	☑
Greater than (>)		☑	☑
Less than (<)		☑	☑
Greater than or equal to (≥)		☑	☑
Less than or equal to (≤)		☑	☑
Between (inclusive)		☑	☑
Not between (inclusive)		☑	☑
Contains	☑		
Does not contain	☑		
Is missing (null)	☑	☑	☑
Is not missing (not null)	☑	☑	☑
Starts with	☑		
Ends with	☑		

For string comparisons, it is also possible to use the ☑ *Case sensitive* option.

After setting all search options, click *Select features* to select the matching features. The drop-down options are:

- *Select features*
- *Add to current selection*
- *Remove from current selection*
- *Filter current selection*

You can also clear all search options using the *Reset form* button.

Once the conditions are set, you can also either:

- *Zoom to features* on the map canvas without the need of a preselection
- *Flash features*, highlighting the matching features. This is a handy way to identify a feature without selection or using the Identify tool. Note that the flash does not alter the map canvas extent and would be visible only if the feature is within the bounds of the current map canvas.

8.4.2 Identifying Features

The  Identify Features tool allows you to interact with the map canvas and get information on features or pixels in a pop-up window. It can be used to query most of the layer types supported by QGIS (vector, raster, mesh, point cloud, wms, wfs, …). To identify an element, use either:

- *View* ► *Identify Features*
- `Ctrl+Shift+I` (or ✗ `Cmd+Shift+I`),
-  Identify Features button on the *Attributes* toolbar

Then click on a feature or pixel of the active layer. The identified item gets highlighted in the map canvas while the *Identify Results* dialog opens with detailed information on it. The dialog also shows a set of buttons for advanced configuration.

The Identify Results dialog

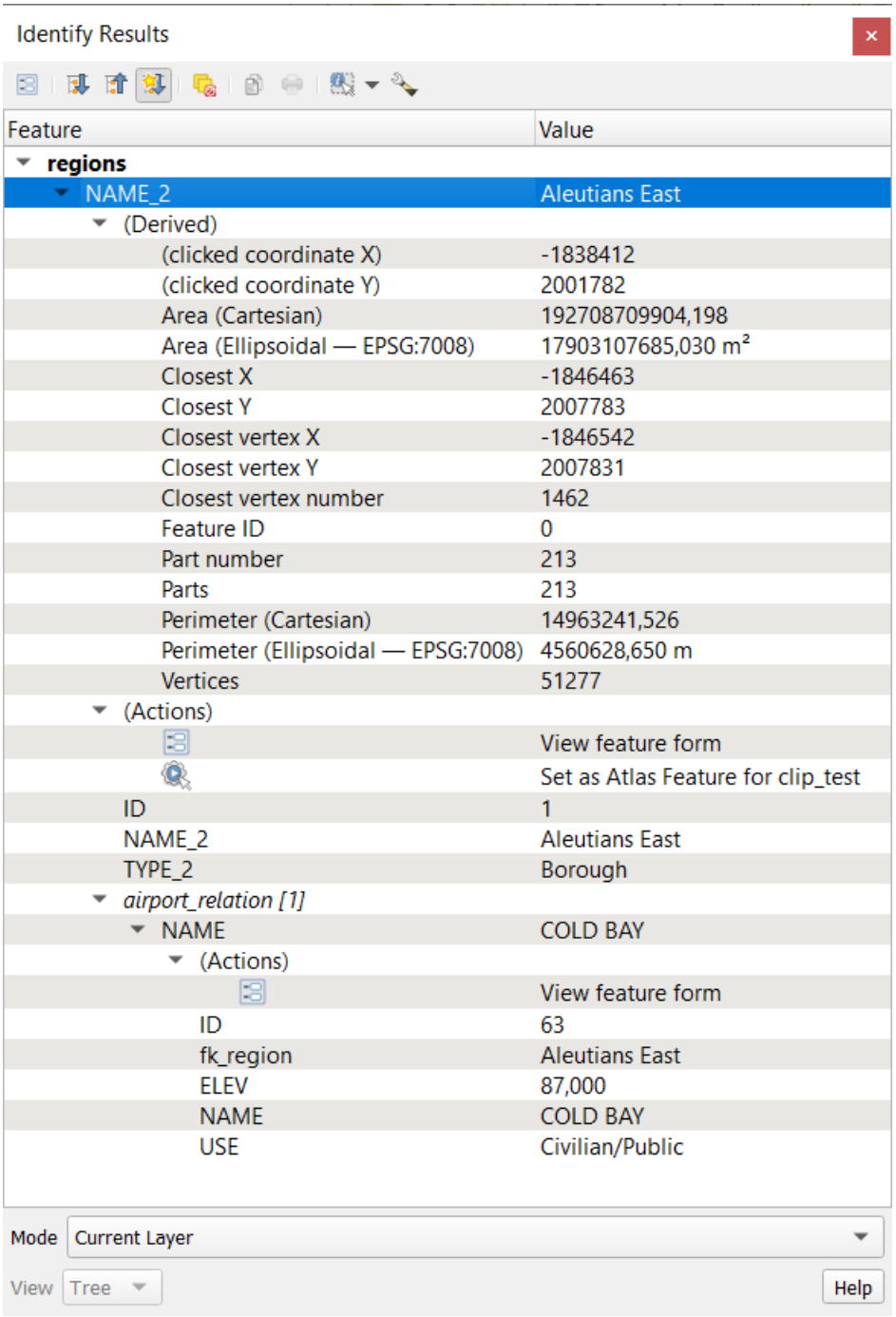


Fig. 8.11: Identify Results dialog

From bottom to top:

- The *View* controls the general aspect of the dialog and the formatting of the results; it can be set as:
  - **Tree**: this is the default view, and returns the results in a tree-structure

  - **Table**: available only for raster-based layers, it allows to display the results as a table whose columns are `Layer`, `FID`, `Attribute` and `Value`
  - or **Graph**: available only for raster-based layers
- The *Mode* helps you select the layers from which results could be returned. These layers should be set visible, displaying data in the map canvas, and set *identifiable* from the *Project properties* ► *Data Sources* ► *Layers capabilities*. Available modes are:
  - **Current layer**: only the layer(s) selected in the *Layers* panel return results. If a group is selected, then results are picked from its leaf layers.
  - **Top down, stop at first**: results are from the layer of the top most feature or pixel under the mouse.
  - **Top down**: results are from the layers with feature or pixel under the mouse.
  - **Layer selection**: opens a contextual menu where the user selects the layer to identify features from. If only a single feature is under the mouse, then the results are automatically displayed.
- In the upper part of the *Identify Results* dialog, a frame shows the *information* returned by features as a table, a graph or a tree, depending on the *selected view*. When in a tree view, you have a handful of tools above the results:
  -  Open Form of the current feature
  -  Expand tree
  -  Collapse tree
  -  Expand New Results by Default to define whether the next identified feature's information should be collapsed or expanded
  -  Clear Results
  -  Copy the identified feature to clipboard, suitable for pasting in a spreadsheet.
  -  Print selected HTML response: a text-based formatting of the results to print on paper or save as a `.PDF` file
  - the *interactive identifying tools*: a drop-down menu with tools for selecting on the map canvas features or pixels to identify
  - Under  Identify Settings, you can activate whether to:
    -  *Auto open form for single feature results*: If checked, each time a single feature is identified, a form opens showing its attributes. This is a handy way to quickly edit a feature's attributes.
    -  *Hide derived attributes from results* to only show fields actually defined in the layer
    -  *Hide NULL values from results*
  -  Help to access the current documentation

Using the Identify Features tool

In its default display (*View: Tree*), the *Identify Results* panel offers several tools to interact with the layers to query. A smart combination of these tools with the *target layers selector* may greatly improve identification operations:

-  Identify Feature(s) by single click or click-and-drag
  - single click or click-and-drag: overlaying features in the target layers are returned
  - right-click: overlaying features from target layers are listed in the contextual menu, grouped by layers. You can then choose to:
    - display the result for a specific feature,

    * display the result for all the features of a specific layer,
    * for vector layers, it is also possible to open its attribute table filtered to the returned features
    * or show all of the returned features.

-  Identify Features on Mouse over: move over the map canvas and hovered items in the target layers get highlighted and returned in the results panel.
-  Identify Features by Polygon: returns items overlapping a drawn or selected polygon.
  - Draw a polygon (left click to add point, right click to close the polygon) and all the overlaying features from target layers are highlighted and returned in the results panel.
  - Right-click and you get the list of all visible polygon features in the project under the click. Pick an entry and QGIS will return all the features from the target layers that overlap the selected polygon.
-  Identify Features by Freehand: returns items overlapping a polygon drawn by freehand. Draw a polygon (left-click to start, move the pointer to shape the area and right-click to close the polygon). All the overlaying features from target layers are highlighted and returned in the results panel.
-  Identify Features by Radius returns items overlapping a drawn circle. Draw a cercle (left-click to indicate the center point, move the pointer to shape the area or enter the radius in the pop-up text box and left-click or press `Enter` to validate the circle). All the overlaying features from target layers are highlighted and returned in the results panel.

> **Tip**
>
> **Filter the layers to query with the Identify Features tool**
>
> Under *Project* ► *Properties...* ► *Data Sources* ► *Layer Capabilities*, uncheck the *Identifiable* column next to a layer to avoid it being queried when using the  Identify Features tool. This is a handy way to return features from only layers that are of interest to you.

Feature information

When you identify a data in the map canvas, the *Identify Results* dialog will list information about the items clicked (or hovered over, depending on the tool in use). The default view is a tree view in which the first item is the name of the layer and its children are its identified feature(s). Each feature is described by the name of a field along with its value. This field is the one set in *Layer Properties* ► *Display*. All the other information about the feature follows.

The feature information displayed by the identify tool will depend on the type of layer you have selected, whether it is a vector layer (including vector tiles or point cloud data) or raster layer. If your layer is raster, clicking on a location on the map canvas with identify tool will highlight the identified raster pixel. The Identify Results dialog can be customized to display custom fields, but by default it will display the following information:

- The feature *display name*;
- **Actions**: Actions can be added to the identify feature windows. The action is run by clicking on the action label. By default, only one action is added, namely `View feature form` for editing. You can define more actions in the layer's properties dialog (see *Actions Properties*).
- **Derived**: This information is calculated or derived from other information. It includes:
  - general information about the feature's geometry:
    * depending on the geometry type, the cartesian measurements of length, perimeter or area in the layer's CRS units. For 3D line vectors the cartesian line length is available.
    * depending on the geometry type and if an ellipsoid is set in the project properties dialog for *Measurements*, the ellipsoidal values of length, perimeter or area using the specified units

        * the count of geometry parts in the feature and the number of the part clicked
        * the count of vertices in the feature
    - coordinate information, using the project properties *Coordinates display* settings:
        * `X` and `Y` coordinate values of the point clicked
        * the number of the closest vertex to the point clicked
        * `X` and `Y` coordinate values of the closest vertex (and `Z`/`M` if applicable)
        * if you click on a curved segment, the radius of that section is also displayed.
        * if both the vector layer and the project have vertical datums set and they differ, the `Z` value will be displayed for both datums.
- **Data attributes**: This is the list of attribute fields and values for the feature that has been clicked.
- information about the related child feature if you defined a *relation*:
    - the name of the relation
    - the entry in reference field, e.g. the name of the related child feature
    - **Actions**: lists actions defined in the layer's properties dialog (see *Actions Properties*) and the default action is `View feature form`.
    - **Data attributes**: This is the list of attributes fields and values of the related child feature.

> **Note**
>
> Links in the feature's attributes are clickable from the *Identify Results* panel and will open in your default web browser.

Results contextual menu

Other functions can be found in the context menu of the identified item. For example, from the context menu you can:

- View the feature form
- Zoom to feature
- Copy feature: Copy all feature geometry and attributes
- Toggle feature selection: Add identified feature to selection
- Copy attribute value: Copy only the value of the attribute that you click on
- Copy feature attributes: Copy the attributes of the feature
- Select features by attribute value: Select all features in the layer that match the selected attribute
- Clear result: Remove results in the window
- Clear highlights: Remove features highlighted on the map
- Highlight all
- Highlight layer
- Activate layer: Choose a layer to be activated
- Layer properties: Open layer properties window
- Expand all
- Collapse all

8.5 Save and Share Layer Properties

8.5.1 Managing Custom Styles

When a vector layer is added to the map canvas, QGIS by default uses a random symbol/color to render its features. However, you can set a default symbol in *Project* ► *Properties…* ► *Default styles* that will be applied to each newly added layer according to its geometry type.

Most of the time, though, you'd rather have a custom and more complex style that can be applied automatically or manually to the layers (with less effort). You can achieve this by using the *Style* menu at the bottom of the Layer Properties dialog. This menu provides you with functions to create, load and manage styles.

A style stores various information set in the layer or project properties dialog to render or interact with the layer. This includes:

Table 8.4: Components of stored style and their references

Category	Vector	Raster
Layer Configuration	*Layers capabilities*	
	Display Properties	*Display Properties*
Symbology	*Symbology Properties*	*Symbology Properties* [1]
3D Symbology	*3D View Properties*	N/A
Labels	*Labels Properties*	N/A
Fields	*Fields Properties* and *Constraints*	N/A
Attributes Form	*Attributes Form Properties*	N/A
Actions	*Actions Properties*	N/A
Map Tips	*Display Properties*	*Display Properties*
Diagrams	*Diagrams Properties*	N/A
Attribute Table Configuration	*Working with the Attribute Table*	
Rendering	*Rendering Properties*	*Rendering Properties* [1]
Custom Properties	*Information Properties*	*Information Properties*
Geometry Options	*Digitizing Properties*	N/A
Relations	*Setting relations between multiple layers*	N/A
Temporal Properties	*Temporal Properties*	*Temporal Properties*
Legend Settings	*Legend Properties*	N/A
Elevation Properties	*Elevation Properties*	*Elevation Properties*
Notes	*Layer notes*	

[1]: For raster layers, symbology and rendering items are melted together, meaning that saving/loading one would also save/load items of/from the other.

By default, the style applied to a loaded layer is named `default`. Once you have got the ideal and appropriate rendering for your layer, you can save it by clicking the  *Style* combo box and choosing:

- **Rename Current**: The active style is renamed and updated with the current options
- **Add**: A new style is created using the current options. By default, it will be saved in the QGIS project file. See below to save the style in another file or a database
- **Remove**: Delete unwanted style, in case you have more than one style defined for the layer.

At the bottom of the Style drop-down list, you can see the styles set for the layer with the active one checked.

Note that each time you validate the layer properties dialog, the active style is updated with the changes you've made.

You can create as many styles as you wish for a layer but only one can be active at a time. In combination with *Map Themes*, this offers a quick and powerful way to manage complex projects without the need to duplicate any layer in the map legend.

Note

Given that whenever you apply modifications to the layer properties, changes are stored in the active style, always ensure you are editing the right style to avoid mistakenly altering a style used in a *map theme*.

Tip

Manage styles from layer context menu

Right-click on the layer in the *Layers* panel to copy, paste, add or rename layer styles. For both vector and raster layers, you can choose which style categories to copy/paste. Choose *All Style Categories* or one of the available categories, depending on the layer type. Note that for raster layers, the *Symbology* and *Rendering* categories are always copied together.

8.5.2 Storing Styles in a File or a Database

While styles created from the *Style* combo box are by default saved inside the project and can be copied and pasted from layer to layer in the project, it's also possible to save them outside the project so that they can be loaded in another project.

Save as text file

Clicking the  *Style* ► *Save Style*, you can save the style as a:

- QGIS layer style file (`.qml`)
- SLD file (`.sld`), only available for vector layers

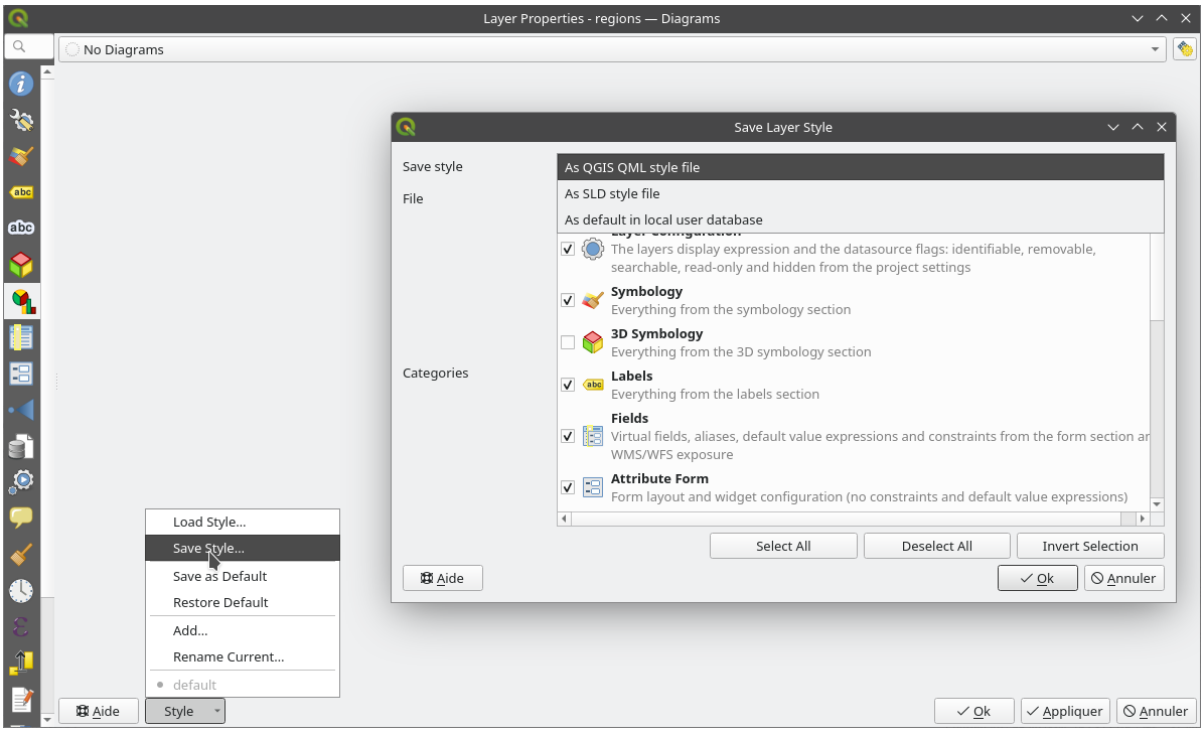


Fig. 8.12: Vector layer style combo box options

Used on file-based format layers (`.shp`, `.tab`...), *Save as Default* generates a `.qml` file for the layer (with the same name). SLDs can be exported from any type of renderer – single symbol, categorized, graduated or rule-based – but when importing an SLD, either a single symbol or rule-based renderer is created. This means that categorized or graduated styles are converted to rule-based. If you want to preserve those renderers, you have to use the QML format. On the other hand, it can be very handy sometimes to have this easy way of converting styles to rule-based.

Save in database

Vector layer styles can also be stored in a database if the layer datasource is a database provider. Supported formats are PostgreSQL, GeoPackage, SpatiaLite, MS SQL Server and Oracle. The layer style is saved inside a table (named `layer_styles`) in the database. Click on *Save Style...* ► *Save in database* then fill in the dialog to define a style name, add a description, a `.ui` file if applicable and to check if the style should be the default style.

You can save several styles for a single table in the database. However, each table can have only one default style. Default styles can be saved in the layer database or in `qgis.db`, a local SQLite database in the active *user profile* directory.

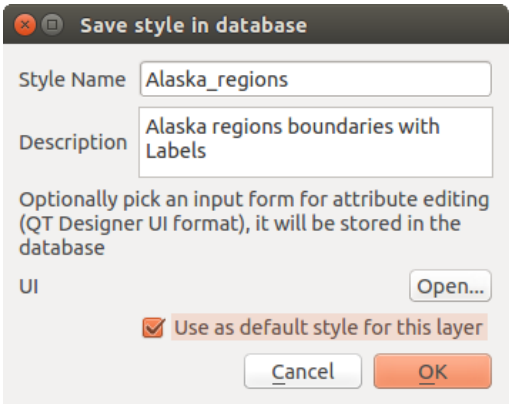


Fig. 8.13: Save Style in database Dialog

Tip

Sharing style files between databases

You can only save your style in a database if the layer comes from such a database. You can't mix databases (layer in Oracle and style in MS SQL Server for instance). Use instead a plain text file if you want the style to be shared among databases.

Note

You may encounter issues restoring the `layer_styles` table from a PostgreSQL database backup. Follow *QGIS layer_style table and database backup* to fix that.

Load style

When loading a layer in QGIS, if a default style already exists for this layer, QGIS loads the layer with this style. Also *Style* ► *Restore Default* looks for and loads that file, replacing the layer's current style.

Style ► *Load Style* helps you apply any saved style to a layer. While text-file styles (`.sld` or `.qml`) can be applied to any layer whatever its format, loading styles stored in a database is only possible if the layer is from the same database or the style is stored in the QGIS local database.

The *Database Styles Manager* dialog displays a list of styles related to the layer found in the database and all the other styles saved in it, with name and description.

Tip

Quickly share a layer style within the project

You can also share layer styles within a project without importing a file or database style: right-click on the layer in the *Layers Panel* and, from the *Styles* combo box , copy the style of a layer and paste it to a group or a selection of layers: the style is applied to all the layers that are of the same type (vector vs raster) as the original layer and, in the case of vector layers, have the same geometry type (point, line or polygon).

8.5.3 Layer definition file

Layer definitions can be saved as a `Layer Definition File` (`.qlr`) using *Export* ► *Save As Layer Definition File…* in the active layers' context menu. A layer definition file (`.qlr`) includes references to the data source of the layers and their styles. `.qlr` files are shown in the Browser Panel and can be used to add the layers (with the saved style) to the Layers Panel. You can also drag and drop `.qlr` files from the system file manager into the map canvas.

8.6 Documenting your data

In addition to displaying and symbolizing the data in the layers, QGIS allows you to fill:

- **metadata**: information to help people find and understand the dataset, how they can access and use it… these are properties of the datasource and can live out of the QGIS project.
- **notes**: instructions and comments regarding the layer in the current project

8.6.1 Metadata

In the layer properties dialog, the  *Metadata* tab provides you with options to create and edit a metadata report on your layer.

Information to fill concern:

- the data *Identification*: basic attribution of the dataset (parent, identifier, title, abstract, language…);
- the *Categories* the data belongs to. Alongside the **ISO** categories, you can add custom ones;
- the *Keywords* to retrieve the data and associated concepts following a standard based vocabulary;
- the *Access* to the dataset (licenses, rights, fees, and constraints);
- the *Extent* of the dataset, either spatial one (CRS, map extent, altitudes) or temporal;
- the *Contact* of the owner(s) of the dataset;
- the *Links* to ancillary resources and related information;
- the *History* of the dataset.

A summary of the filled information is provided in the *Validation* tab and helps you identify potential issues related to the form. You can then either fix them or ignore them.

Metadata are saved in the project file by default, the *Metadata* drop-down offers options for loading/saving metadata from `.qmd` file and for loading/saving metadata in the "Default" location.

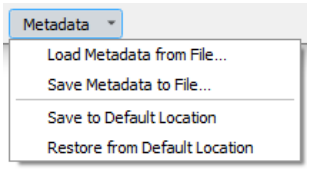


Fig. 8.14: Metadata load/save options

The "Default" location used by *Save to Default Location* and *Restore from Default Location* changes depending on the underlying data source and on its configuration:

- For PostgreSQL data sources if the configuration option *Allow saving/loading QGIS layer metadata in the database* is checked the metadata are stored inside a dedicated table in the database.
- For GeoPackage data sources *Save to Default Location* always saves the metadata in the internal metadata tables of the GeoPackage.

  When metadata are saved into the internal tables of PostgreSQL or GeoPackage they become available for search and filtering in the browser and in the *layer metadata search panel*.
- For all other file based data sources *Save to Default Location* saves the metadata in a `.qmd` file alongside the file.
- In all other cases *Save to Default Location* saves the metadata in a local `.sqlite` database.

8.6.2 Layer notes

Layer notes allow you to document the layer within the current project. They can be place to store important messages for users of the project like to do lists, instructions, warnings, …

From the layer's contextual menu in *Layers* panel, select *Add layer notes…* and fill the open dialog with necessary texts.

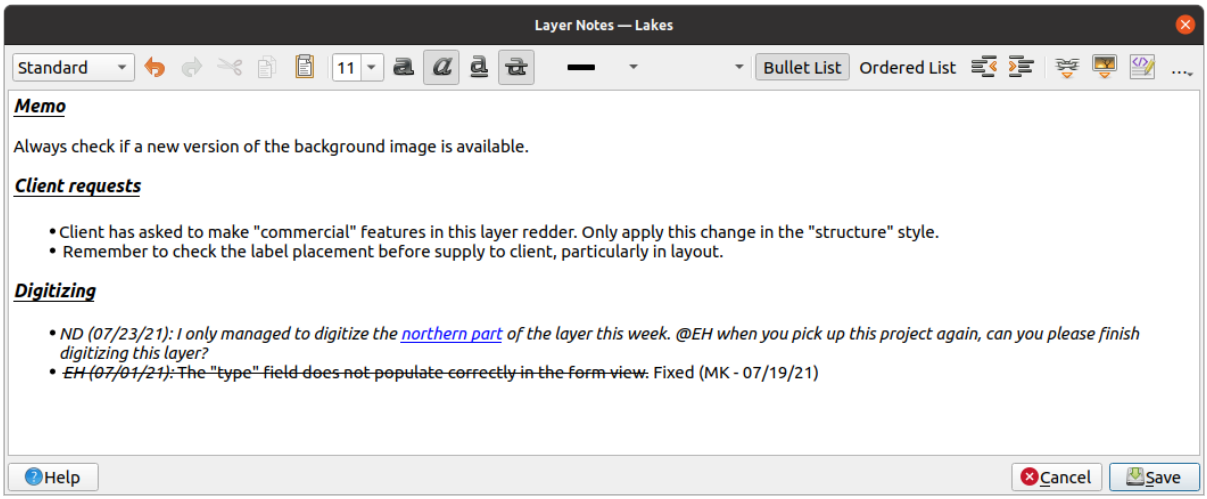


Fig. 8.15: Adding notes to a layer

The *Add layer notes* dialog provides a html-based multiline text box with a complete set of tools for:

- text manipulation: cut, copy, paste, undo, redo
- characters formatting, applied to all or parts of the contents: font size and color, bold, italic, underline, strikethrough, background color, URL highlighting
- paragraph structuring: bullet and numbered lists, indentation, predefined headings
- file insertion, even with drag-and-drop
- editing with HTML coding

From the … drop-down at the far right of the toolbar, you can:

- *Remove all formatting*
- *Remove character formatting*
- *Clear all content*

In the *Layers* panel, a layer with a note is assigned the  icon which, upon hover, displays the note. Click the icon to edit the note. You can as well right-click the layer and *Edit layer note…* or *Remove layer note*.

Note

Notes are part of the *layer style* and can be saved in the `.qml` or `.qlr` file. They can also be transferred from one layer to another while copy-pasting the layer style.

8.7 Storing values in Variables

In QGIS, you can use variables to store useful recurrent values (e.g. the project's title, or the user's full name) that can be used in expressions. Variables can be defined at the application's global level, project level, layer level, processing modeler level, layout level, and layout item's level. Just like CSS cascading rules, variables can be overwritten - e.g., a project level variable will overwrite any application global level variables set with the same name. You can use these variables to build text strings or other custom expressions using the `@` character before the variable name. For example in print layout creating a label with this content:

```
This map was made using QGIS [% @qgis_version %]. The project file for this
map is: [% @project_path %]
```

Will render the label like this:

```
This map was made using QGIS 3.4.4-Madeira. The project file for this map is:
/gis/qgis-user-conference-2019.qgs
```

Besides the *preset read-only variables*, you can define your own custom variables for any of the levels mentioned above. You can manage:

- **global variables** from the *Settings* ► *Options* menu
- **project variables** from the *Project Properties* dialog (see *Project Properties*)
- **vector layer variables** from the *Layer Properties* dialog (see *The Vector Properties Dialog*);
- **modeler variables** from the *Model Designer* dialog (see *The model designer*);
- **layout variables** from the *Layout* panel in the Print layout (see *The Layout Panel*);
- and **layout item variables** from the *Item Properties* panel in the Print layout (see *Layout Items Common Options*).

To differentiate from editable variables, read-only variable names and values are displayed in italic. On the other hand, higher level variables overwritten by lower level ones are strike through.

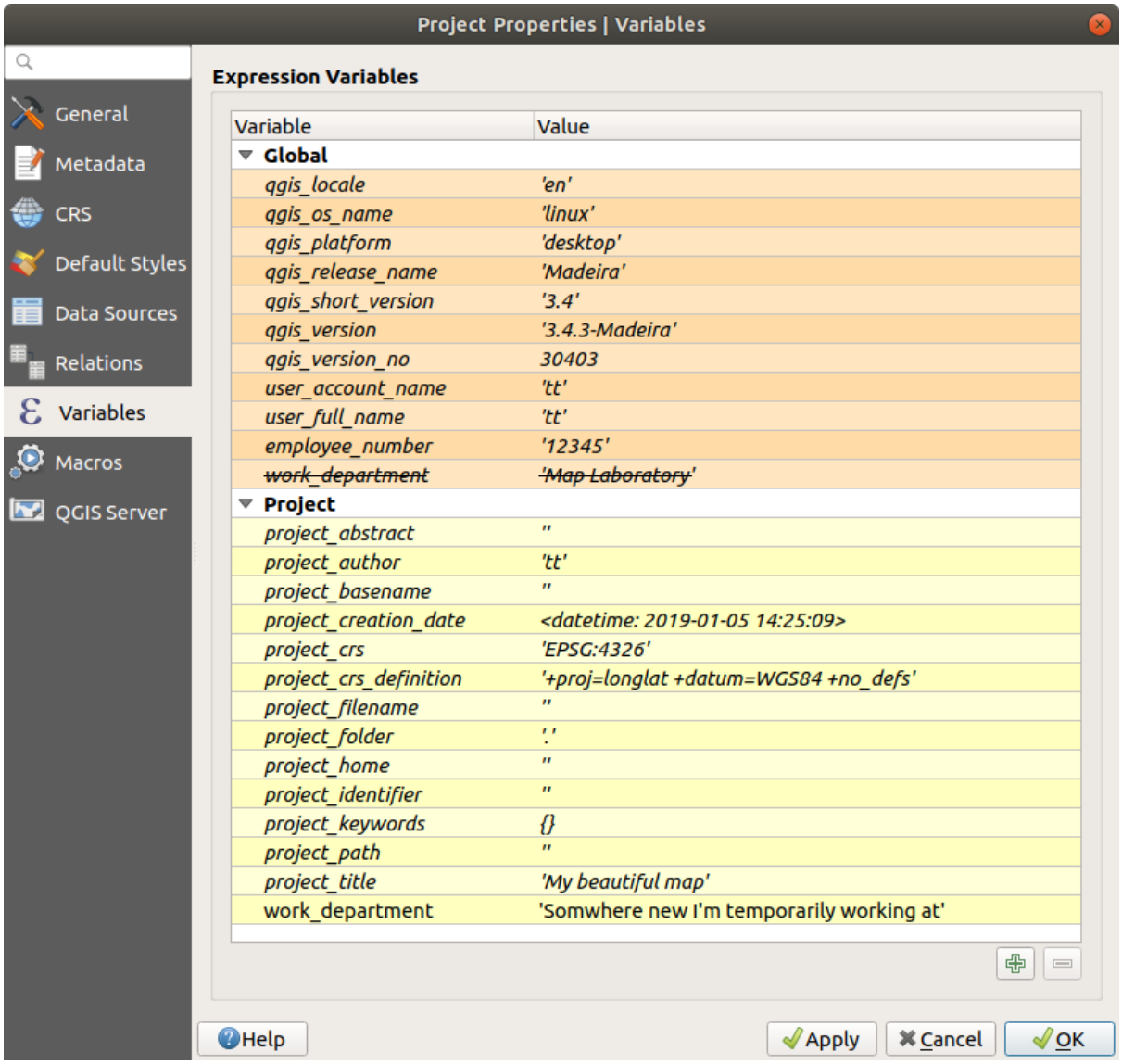


Fig. 8.16: Variables editor at the project level

Note

You can read more about variables and find some examples in Nyall Dawson's Exploring variables in QGIS 2.12, part 1, part 2 and part 3 blog posts.

8.8 Authentication

QGIS has the facility to store/retrieve authentication credentials in a secure manner. Users can securely save credentials into authentication configurations, which are stored in a portable database, can be applied to server or database connections, and are safely referenced by their ID tokens in project or settings files. For more information see *Authentication System*.

A master password needs to be set up when initializing the authentication system and its portable database.

8.9 Common widgets

In QGIS, there are some options you'll often have to work with. For convenience, QGIS provides you with special widgets that are presented below.

8.9.1 Color Selector

The color dialog

The *Select Color* dialog will appear whenever you click the  icon to choose a color. The features of this dialog depend on the state of the *Use native color chooser dialogs* parameter checkbox in *Settings* ► *Options…* ► *General*. When checked, the color dialog used is the native one of the OS on which QGIS is running. Otherwise, the QGIS custom color chooser is used.

The custom color chooser dialog has four different tabs which allow you to select colors by  Color ramp,  Color wheel,  Color swatches or  Color picker. With the first two tabs, you can browse to all possible color combinations and apply your choice to the item.

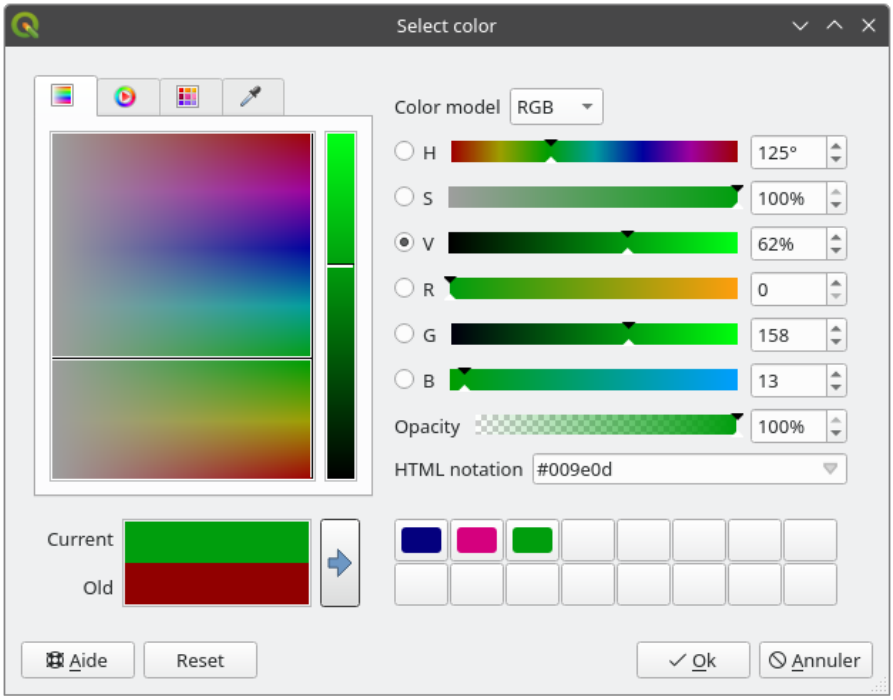


Fig. 8.17: Color selector ramp tab

In the  Color swatches tab, you can choose from a list of color palettes (see *Colors settings* for details). All but the *Recent colors* palette can be modified with the  Add current color and  Remove selected color buttons at the bottom of the frame.

The … button next to the palette combo box also offers several options to:

- copy, paste, import or export colors
- create, import or remove color palettes
- add the custom palette to the color selector widget with the *Show in Color Buttons* item (see Fig. 8.19)

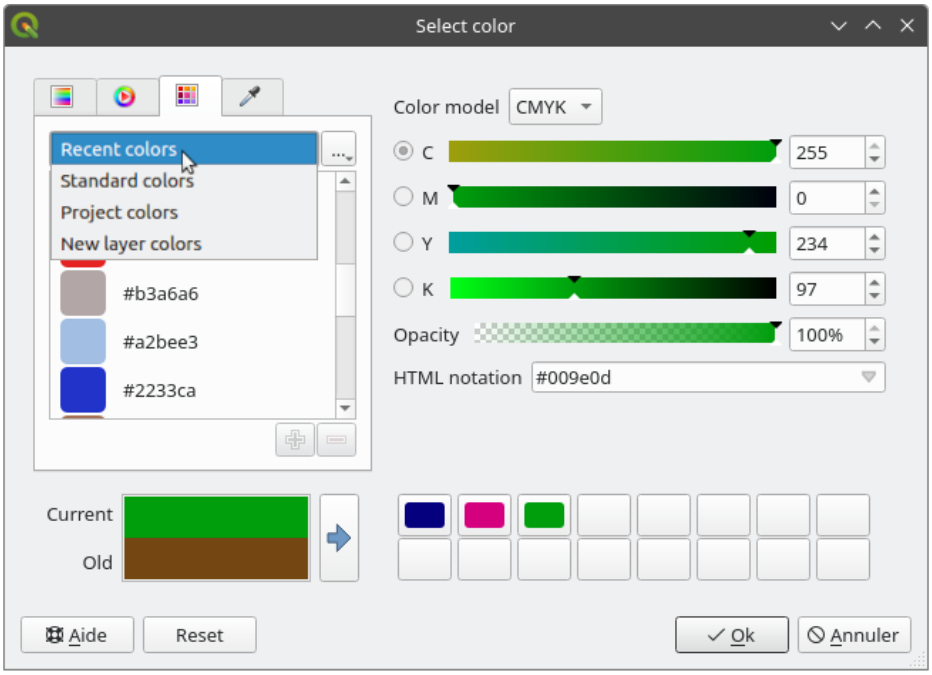


Fig. 8.18: Color selector swatches tab

Another option is to use the  Color picker which allows you to sample a color from under your mouse cursor at any part of the QGIS UI or even from another application: press the space bar while the tab is active, move the mouse over the desired color and click on it or press the space bar again. You can also click the *Sample Color* button to activate the picker.

Whatever method you use, the selected color is always described through options of a *Color model* that defaults to the *project color model*:

- *RGB*: color is set using sliders for `HSV` (Hue, Saturation, Value) or `RGB` (Red, Green, Blue) values.
- *CMYK* (Cyan, Magenta, Yellow, Black)

Modifying a color is as simple as clicking on the color wheel or ramp or on any of the color parameters sliders. You can adjust such parameters with the spinbox beside or by scrolling the mouse wheel over the corresponding slider. You can also type the color in HTML notation. Finally, there is an *Opacity* slider to set transparency level.

The dialog also provides a visual comparison between the *Old* color (applied to object) and the *Current* one (being selected). Using drag-and-drop or pressing the  Add color to swatch button, any of these colors can be saved in a slot for easy access.

Tip

Quick color modification

Drag-and-drop a color selector widget onto another one to apply its color.

The color drop-down shortcut

Click the drop-down arrow to the right of the  color button to display a widget for quick color selection. This shortcut provides access to:

- a color wheel to pick a color from
- an alpha slider to change color opacity
- the color palettes previously set to *Show in Color Buttons*
- copy the current color and paste it into another widget
- pick a color from anywhere on your computer display
- choose a color from the color selector dialog
- drag-and-drop the color from one widget to another for quick modification

Tip

Scroll the mouse wheel over a color selector widget to quickly modify the opacity of the associated color.

Note

When the color widget is set to a *project color* through the data-defined override properties, the above functions for changing the color are unavailable. You'd first need to *Unlink color* or *Clear* the definition.

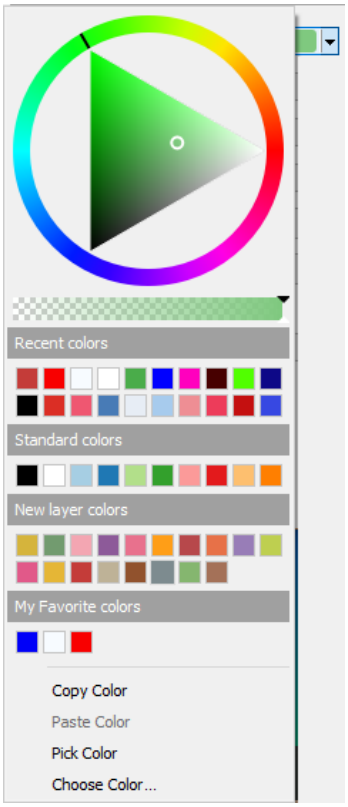


Fig. 8.19: Quick color selector menu

The color ramp drop-down shortcut

Color ramps are a practical way to apply a set of colors to one or many features. Their creation is described in the *Setting a Color Ramp* section. As for the colors, pressing the  color ramp button opens the corresponding color ramp type dialog allowing you to change its properties.

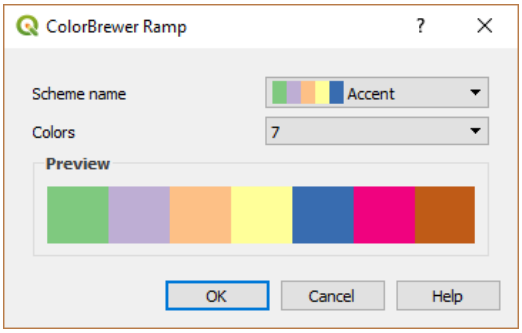


Fig. 8.20: Customizing a colorbrewer ramp

The drop-down menu to the right of the button gives quick access to a wider set of color ramps and options:

- *Invert Color Ramp*
- *Clear Current Ramp* to unset any assigned color ramp to the widget (available only in some contexts)
- *Random Colors*: available only in some contexts (e.g., when a color ramp is being used for a layer symbology), checking this entry creates and applies a color ramp with random colors. It also enables a *Shuffle random colors* entry to regenerate a new random color ramp if the current one is not satisfactory.
- a preview of the `gradient` or `catalog: cpt-city` color ramps flagged as **Favorites** in the *Style Manager* dialog
- *All Color Ramps* to access the compatible color ramps database
- *Create New Color Ramp…* of any supported type that could be used in the current widget (note that this color ramp will not be available elsewhere unless you save it in the library)
- *Edit Color Ramp…*, the same as clicking the whole color ramp button
- *Save Color Ramp…*, to save the current color ramp with its customizations in the style library

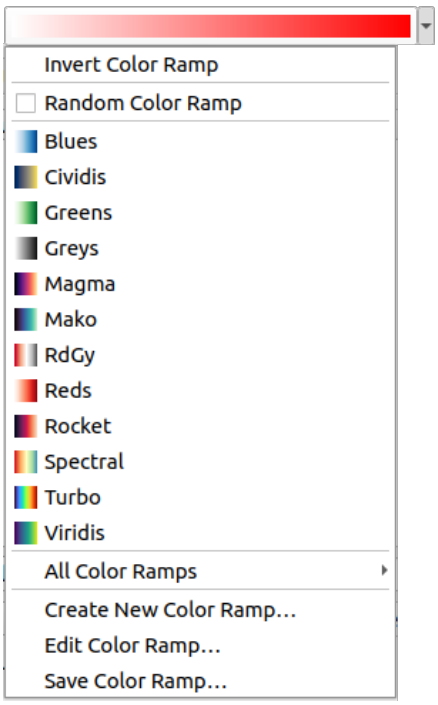


Fig. 8.21: Quick color ramp selection widget

8.9.2 Symbol Widget

The *Symbol* selector widget is a convenient shortcut when you want to set symbol properties of a feature. Clicking the drop-down arrow shows the following symbol options, together with the features of the *color drop-down widget*:

- *Configure Symbol...*: the same as pressing the symbol selector widget. It opens a dialog to set the *symbol parameters*.
- *Copy Symbol* from the current item
- *Paste Symbol* to the current item, speeding configuration
- *Clear Current Symbol* to unset any assigned symbol to the widget (available only in some contexts)

Tip

Scroll the mouse wheel over a marker or line symbol widget to quickly modify the size of the associated symbol.

8.9.3 Remote or embedded file selector

Along with the file selector widget, the ... button will sometimes show a drop-down arrow. This is usually available when using:

- an SVG file in a symbol or a label
- a raster image to customize symbols, labels, textures or decorations

Pressing the arrow will provide you with a menu to:

- *Select File...*, to load the file from the file system. The file is identified through the file path and QGIS needs to resolve the path in order to display the corresponding image

- *From URL…*, to load the file from a remote URL. As above, the image will only be loaded on successful retrieval of the remote resource
- *Embed File…*, to embed the file inside the current project, style database, or print layout template. The file is then always rendered as part of the item. This is a convenient way to create self-contained projects with custom symbols which can be easily shared amongst different users and installations of QGIS.
- *Extract Embedded File…*, to extract the embedded file from the widget and save it on disk.

8.9.4 Visibility Scale Selector

The visibility scale selector provides options to control the scales at which an element will be made visible in the map canvas. Out of the specified range of scales, the elements are not displayed. It can be applied e.g. to layers, labels or diagrams, from their *Rendering* properties tab.

1. Tick the *Scale dependent visibility* box
2. Fill the *Minimum (exclusive)* box with the most zoomed out desired scale, typing the value or selecting it from the *predefined scales*
3. and/or fill the *Maximum (inclusive)* box with the most zoomed in desired scale

   The Set to current canvas scale button next to the scale boxes sets the current map canvas scale as boundary of the range visibility. Press the arrow next to the button to access scales from layouts' maps and reuse them to fill the box.

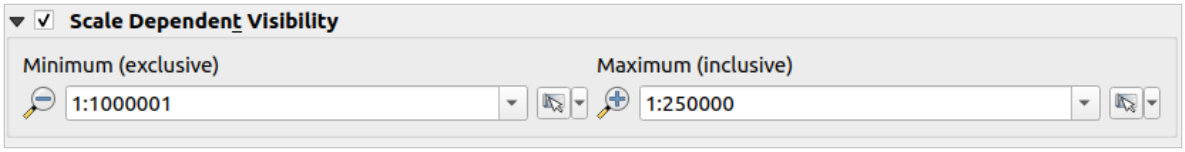


Fig. 8.22: Visibility scale selector widget

8.9.5 Spatial Extent Selector

The *Extent* selector widget is a convenient shortcut when you want to select a spatial extent to assign to a layer or to limit the actions to run on. Depending on the context, it offers selection between:

- *Current Layer Extent*: e.g. when exporting a layer
- *Calculate from Layer* ►: uses extent of a layer loaded in the current project
- Use current *Map Canvas Extent*
- *Draw on Canvas*: a rectangle whose coordinates are then used
- *Calculate from Bookmark*: uses extent of a saved *bookmark*
- *Calculate from Layout Map*: uses extent of a *layout map*
- Enter or edit the coordinates as `xmin, xmax, ymin, ymax`

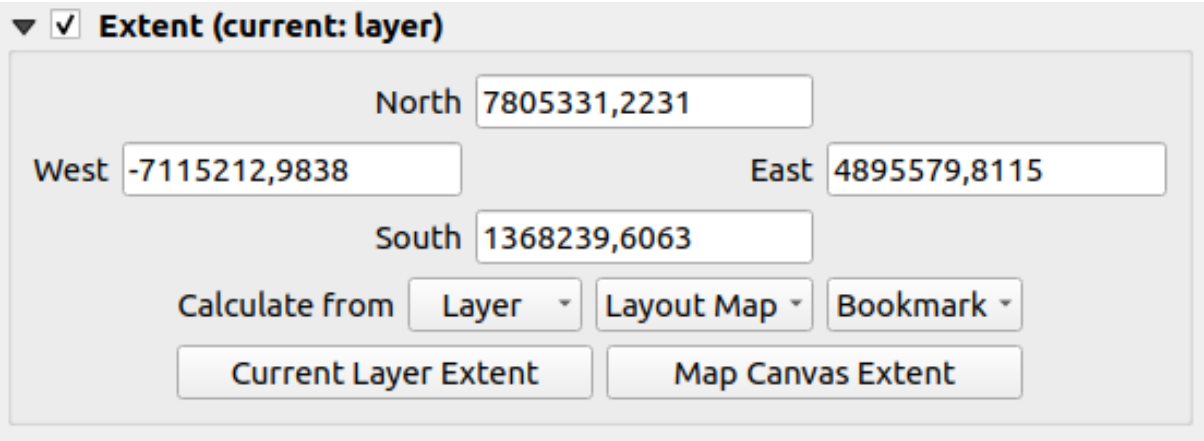


Fig. 8.23: Extent selector widget

8.9.6 Font Selector

The *Font* selector widget is a convenient shortcut when you want to set font properties for textual information (feature labels, decoration labels, map legend text, …). Clicking the drop-down arrow shows some or all of the following options:



Fig. 8.24: Font selector drop-down menu

- *Clear Current Text Format* to unset any assigned text format to the widget (available only in some contexts)
- *Font Size* in the associated unit
- *Recent Fonts* ► menu with the active font checked (at the top)
- *Configure Format...*: same as pressing the font selector widget. It opens the *Text format* dialog, providing advanced formatting options such as color, opacity, orientation, HTML notation, buffer, background, shadow, …

- *Copy Format* of the text
- *Paste Format* to the text, speeding configuration
- the *color widget* for quick color setting

> **Tip**
>
> Scroll the mouse wheel over a font selector widget to quickly modify the font size of the associated text.

8.9.7 Unit Selector

Size properties of the items (labels, symbols, layout elements, …) in QGIS are not necessarily bound to either the project units or the units of a particular layer. For a large set of properties, the *Unit* selector drop-down menu allows you to tweak their values according to the rendering you want (based on screen resolution, paper size, or the terrain). Available units are:

- *Millimeters*
- *Points*
- *Pixels*
- *Inches*
- *Percentage*: allows you to set some properties as a percent of another one. For example, this is useful for creation of text formats where the components (buffer size, shadow radius…) nicely scale as the text size is changed, instead of having constant buffer/shadow sizes. So you don't need to adjust those sizes, when the text size changes.
- *Meters at Scale*: This allows you to always set the size in meters, regardless of what the underlying map units are (e.g. they can be in inches, feet, geographic degrees, …). The size in meters is calculated based on the current project ellipsoid setting and a projection of the distances in meters at the center of the current map extent. For maps in a projected coordinate system this is calculated using projected units. For maps in a geographic (latitude/longitude) based system the size is approximated by calculating meter sizes using ellipsoidal calculations for the vertical scale of the map.
- and *Map Units*: The size is scaled according to the map view scale. Because this can lead to too big or too small values, use the button next to the entry to constrain the size to a range of values based on:
  - The *Minimum scale* and the *Maximum scale*: The value is scaled based on the map view scale until you reach any of these scale limits. Out of the range of scale, the value at the nearest scale limit is kept.
  - and/or The *Minimum size* and the *Maximum size* in `mm`: The value is scaled based on the map view scale until it reaches any of these limits; Then the limit size is kept.

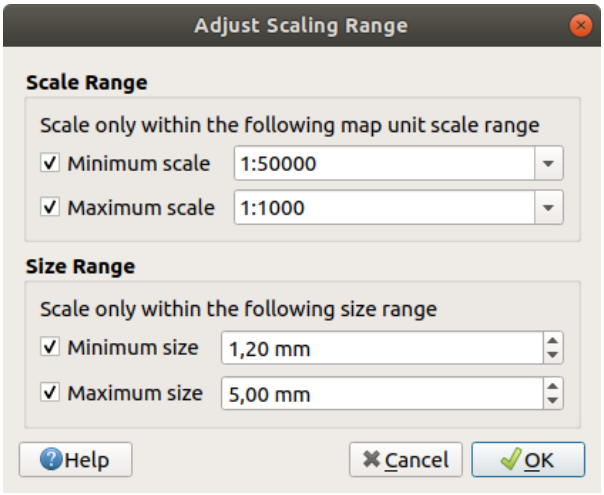


Fig. 8.25: Adjust scaling range dialog

8.9.8 Number Formatting

Numeric formatters allow formatting of numeric values for display, using a variety of different formatting techniques (for instance scientific notation, currency values, percentage values, custom formatting, etc). One use of this is to set text in a layout scale bar or table, a color ramp legend, an elevation profile plot, …

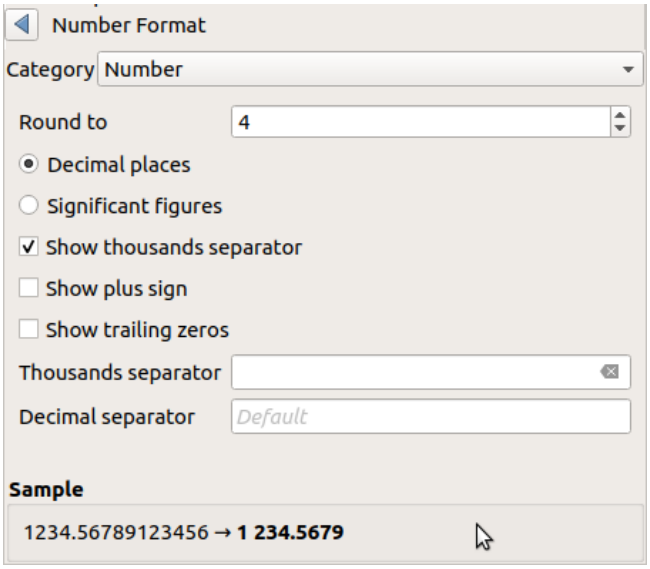


Fig. 8.26: Formatting numeric value

Different categories of formats are supported. For most of them, you can set part or all of the following numeric options:

- *Show thousands separator*
- *Show plus sign*
- *Show trailing zeros*

But they can also have their custom settings. Provided categories are:

- *General*, the default category: has no setting and displays values as set in the parent widget properties or using the global settings.
- *Custom expression*: allows you to craft a custom *QGIS expression* to format numbers. The expression can use the `@value` variable to retrieve the value to be formatted, and then use any standard QGIS expression function to format this as desired.
- *Number*
  - The value can be *Round to* a self defined number of *Decimal places* or their *Significant figures*
  - customize the *Thousands separator* and *Decimal separator*
- *Bearing* for a text representation of a direction/bearing using:
  - *Format*: possible ranges of values are `0 to 180°, with E/W suffix`, `-180 to +180°` and `0 to 360°`
  - number of *Decimal places*
- *Currency* for a text representation of a currency value.
  - *Prefix*
  - *Suffix*
  - number of *Decimal places*
- *Fraction* for a vulgar fractional representation of a decimal value (e.g. *1/2* instead of *0.5*)
  - ☐ *Use unicode super/subscript* to show. For example $^{1}/_{2}$ instead of 1/2
  - ☐ *Use dedicated Unicode characters*
  - customize the *Thousands separator*
- *Percentage* - appends `%` to the values, with setting of:
  - number of *Decimal places*
  - *Scaling* to indicate whether the actual values already represent percentages (then they will be kept as is) or fractions (then they are converted)
- *Scientific* notation in the form `2.56e+03`. The number of *Decimal places* can be set.

A live preview of the settings is displayed under the *Sample* section.

8.9.9 Blending Modes

QGIS offers different options for special rendering effects with these tools that you may previously only know from graphics programs. Blending modes can be applied on layers and features, and also on print layout items:

- *Normal*: This is the standard blend mode, which uses the alpha channel of the top pixel to blend with the pixel beneath it. The colors aren't mixed.
- *Lighten*: This selects the maximum of each component from the foreground and background pixels. Be aware that the results tend to be jagged and harsh.
- *Screen*: Light pixels from the source are painted over the destination, while dark pixels are not. This mode is most useful for mixing the texture of one item with another item (such as using a hillshade to texture another layer).
- *Dodge*: Brighten and saturate underlying pixels based on the lightness of the top pixel. Brighter top pixels cause the saturation and brightness of the underlying pixels to increase. This works best if the top pixels aren't too bright. Otherwise the effect is too extreme.
- *Addition*: Adds pixel values of one item to the other. In case of values above the maximum value (in the case of RGB), white is displayed. This mode is suitable for highlighting features.

- *Darken*: Retains the lowest values of each component of the foreground and background pixels. Like lighten, the results tend to be jagged and harsh.
- *Multiply*: Pixel values of the top item are multiplied with the corresponding values for the bottom item. The results are darker.
- *Burn*: Darker colors in the top item cause the underlying items to darken. Burn can be used to tweak and colorize underlying layers.
- *Overlay*: Combines multiply and screen blending modes. Light parts become lighter and dark parts become darker.
- *Soft light*: Very similar to overlay, but instead of using multiply/screen it uses color burn/dodge. This is supposed to emulate shining a soft light onto an image.
- *Hard light*: Hard light is also very similar to the overlay mode. It's supposed to emulate projecting a very intense light onto an image.
- *Difference*: Subtracts the top pixel from the bottom pixel, or the other way around, in order always to get a positive value. Blending with black produces no change, as the difference with all colors is zero.
- *Subtract*: Subtracts pixel values of one item from the other. In the case of negative values, black is displayed.

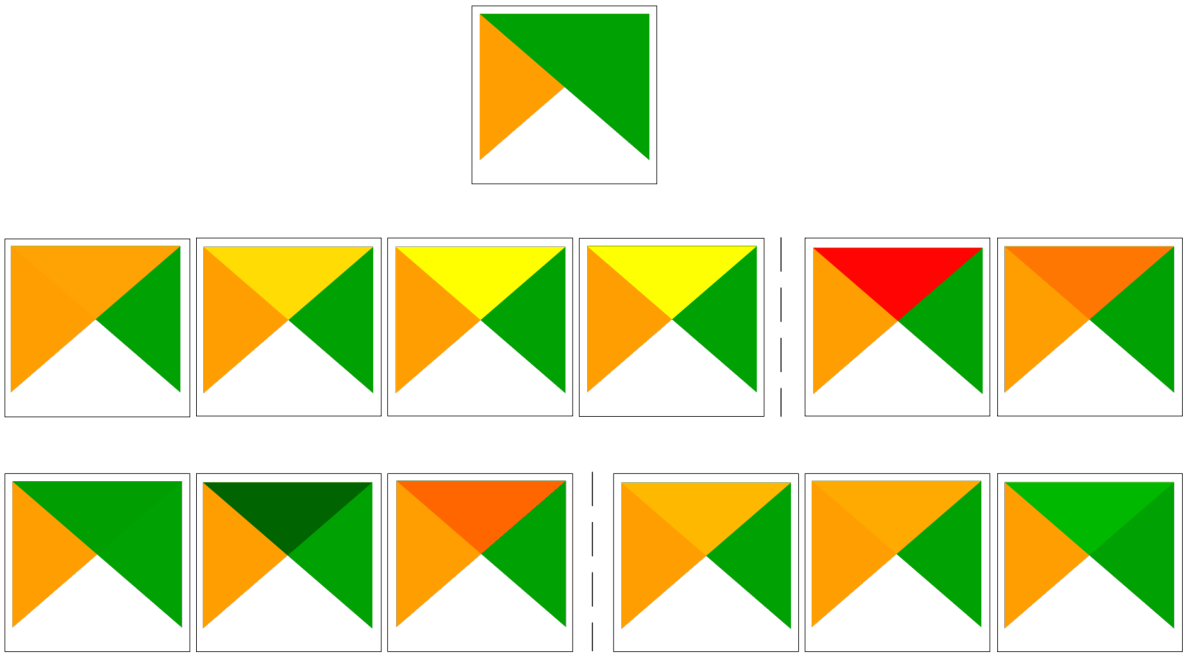


Fig. 8.27: Examples of blend modes applied to a green feature over an orange
From top to bottom, left to right: Normal – Lighten, Screen, Dodge, Addition – Difference, Subtract – Darken, Multiply, Burn – Overlay, Soft light, Hard light

When a layer is part of a group that *renders layers as a group*, additional blending modes are available for the rendering. They provide methods to clip the render of one layer's content by the content in a second "mask" layer.

- *Masked By Below*: The output is the top pixel, where the opacity is reduced by that of the bottom pixel.
- *Mask Below*: The output is the bottom pixel, where the opacity is reduced by that of the top pixel.
- *Inverse Masked By Below*: The output is the top pixel, where the opacity is reduced by the inverse of the bottom pixel.
- *Inverse Mask Below*: The output is the bottom pixel, where the opacity is reduced by the inverse of the top pixel.

- *Paint Inside Below*: The top pixel is blended on top of the bottom pixel, with the opacity of the top pixel reduced by the opacity of the bottom pixel.
- *Paint Below Inside*: The bottom pixel is blended on top of the top pixel, with the opacity of the bottom pixel reduced by the opacity of the top pixel.

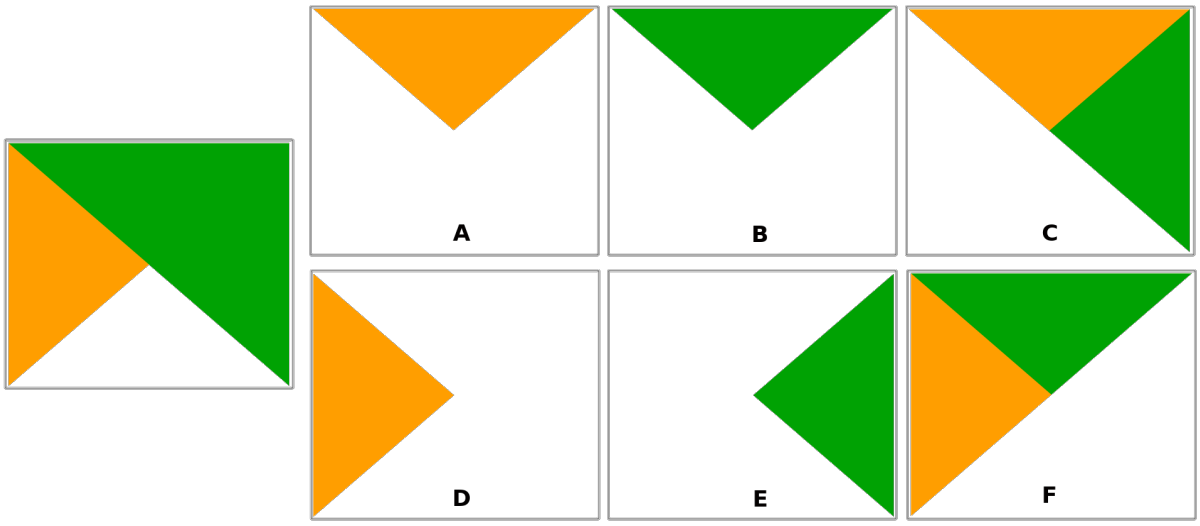


Fig. 8.28: Examples of blend clipping mode applied to top green layer in a group
A: Mask Below **B**: Masked By Below **C**: Paint Below Inside **D**: Inverse Mask Below **E**: Inverse Masked By Below **F**: Paint Inside Below

8.9.10 Data defined override setup

Next to many options in the vector layer properties dialog or settings in the print layout, you will find a Data defined override icon. Using *expressions* based on layer attributes or item settings, prebuilt or custom functions and *variables*, this tool allows you to set dynamic values for parameters. When enabled, the value returned by this widget is applied to the parameter regardless of its normal value (checkbox, textbox, slider…).

The data defined override widget

Clicking the Data defined override icon shows the following entries:

- *Description…* that indicates if the option is enabled, which input is expected, the valid input type and the current definition. Hovering over the widget also pops up this information.
- *Store data in the project*: a button allowing the property to be stored using the *Auxiliary Storage Properties* mechanism.
- *Field type*: an entry to select from the layer's fields that match the valid input type.
- *Color*: when the widget is linked to a color property, this menu gives access to the colors defined as part of the current *project's colors* scheme.
- *Variable*: a menu to access the available user-defined *variables*
- *Edit…* button to create or edit the expression to apply, using the *Expression String Builder* dialog. To help you correctly fill in the expression, a reminder of the expected output's format is provided in the dialog.
- *Paste* and *Copy* buttons.
- *Clear* button to remove the setup.

- For numeric and color properties, *Assistant…* to rescale how the feature data is applied to the property (more details *below*)

> **Tip**
>
> **Use right-click to (de)activate the data override**
>
> When the data-defined override option is set up correctly the icon is yellow or . If it is broken, the icon is red or .
>
> You can enable or disable a configured Data-defined override button by simply clicking the widget with the right mouse button.

> **Tip**
>
> **Use middle-click to create or edit the expression to apply**
>
> You can directly open the *Expression String Builder* dialog to create or edit the expression to apply by simply clicking the Data-defined override widget with the middle mouse button.

Using the data-defined assistant interface

When the Data-defined override button is associated with a size, a rotation, an opacity or a color property, it has an *Assistant…* option that helps you change how the data is applied to the parameter for each feature. The assistant allows you to:

- Define the *Input* data, ie:
  - *Source*: the attribute to represent, using a field or an *expression*
  - the range of values to represent: you can manually enter the values or use the Fetch value range from layer button to fill these fields automatically with the minimum and maximum values returned by the *Source* expression applied to your data
- *Apply transform curve*: by default, output values (see below for setting) are applied to input features following a linear scale. You can override this logic: enable the transform option, click on the graphic to add break point(s) and drag the point(s) to apply a custom distribution.
- Define the *Output* values: the options vary according to the parameter to define. You can globally set:
  - for a color setting, the *color ramp* to apply to values and the single color to use for NULL values
  - for the others, the minimum and maximum values to apply to the selected property as well as the size/angle/opacity value for ignored or NULL source features
  - for size properties, the *Scale method* of representation which can be **Flannery**, **Exponential**, **Surface**, **Radius** or **Linear**
  - the *Exponent* to use for data scaling when the *Scale method* is of exponential type or when tweaking the opacity

When compatible with the property, a live-update preview is displayed in the right-hand side of the dialog to help you control the value scaling.

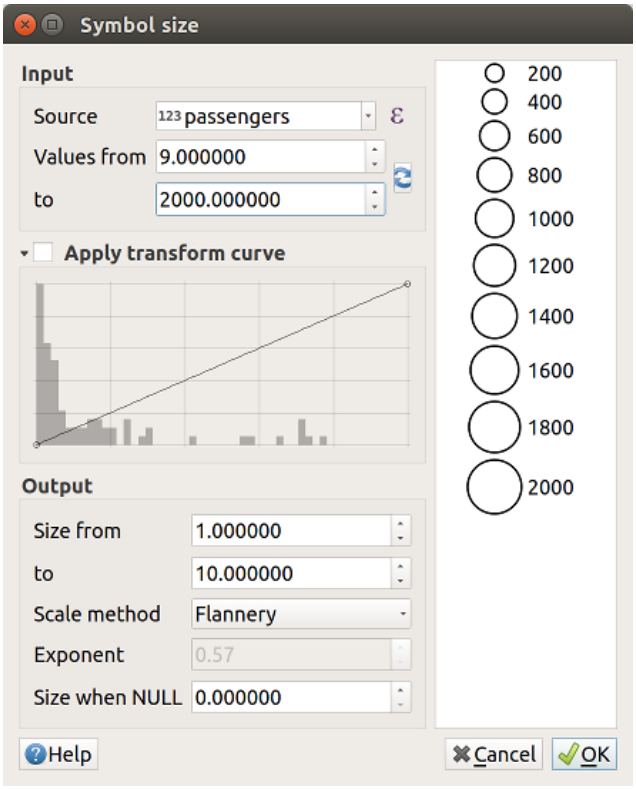


Fig. 8.29: Scaling feature size based on passengers field's value

The values presented in the varying size assistant above will set the size 'Data-defined override' with:

```
coalesce(scale_exp("passengers", 9, 2000, 1, 10, 0.57), 0)
```

CHAPTER

NINE

LEVEL UP WITH EXPRESSIONS

9.1 Expressions

Based on layer data and prebuilt or user defined functions, **Expressions** offer a powerful way to manipulate attribute value, geometry and variables in order to dynamically change the geometry style, the content or position of the label, the value for diagram, the height of a layout item, select some features, create virtual field, …

Note

A list of the default functions and variables for writing expressions can be found at *List of functions*, with detailed information and examples.

9.1.1 The Expression string builder

Main dialog to build expressions, the *Expression string builder* is available from many parts in QGIS and, can particularly be accessed when:

- clicking the ε button;
- *selecting features* with the Select By Expression… tool;
- *editing attributes* with e.g. the Field calculator tool;
- manipulating symbology, label or layout item parameters with the Data defined override tool (see *Data defined override setup*);
- building a *geometry generator* symbol layer;
- doing some *geoprocessing*.

The Expression builder dialog offers access to the:

- *Expression* tab which, thanks to a list of *predefined functions*, helps to write and check the expression to use;
- *Function Editor tab* which helps to extend the list of functions by creating custom ones.

The Interface

The *Expression* tab provides the main interface to write expressions using functions, layer fields and values. It contains the following widgets:

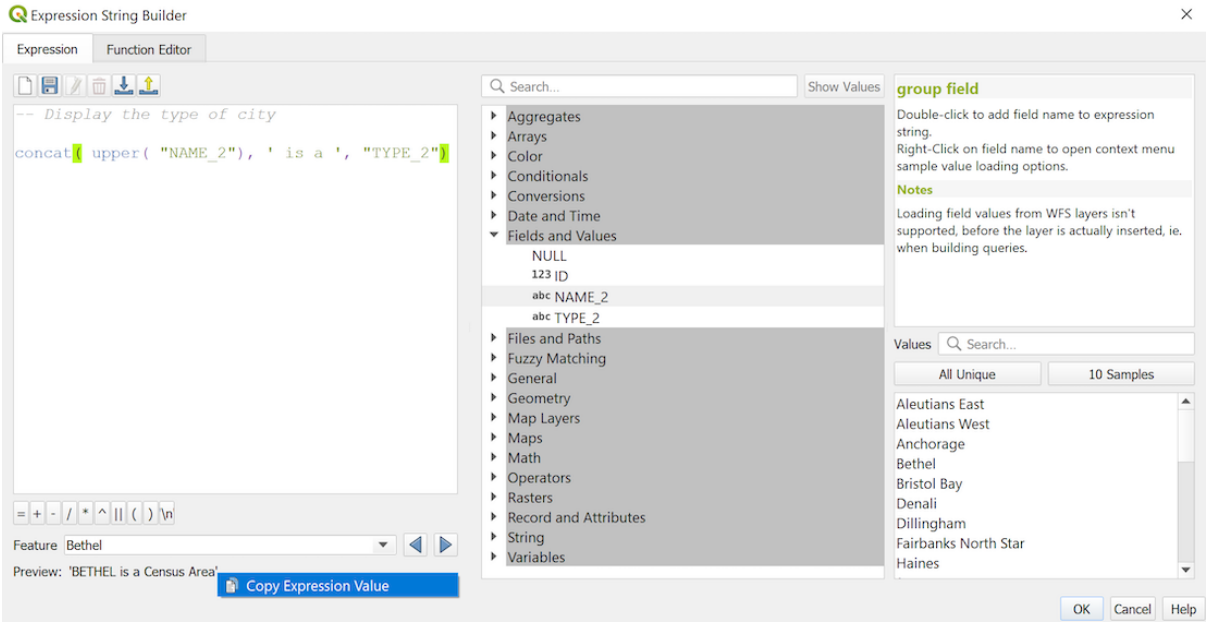


Fig. 9.1: The Expression tab

- An expression editor area for typing or pasting expressions. Autocompletion is available to speed expression writing:
  - Corresponding variables, function names and field names to the input text are shown below: use the `Up` and `Down` arrows to browse the items and press `Tab` to insert in the expression or simply click on the wished item.
  - Function parameters are shown while filling them.

  QGIS also checks the expression rightness and highlights all the errors using:
  - *Underline*: for unknown functions, wrong or invalid arguments;
  - *Marker*: for every other error (eg, missing parenthesis, unexpected character) at a single location.

> **Tip**
>
> **Document your expression with comments**
>
> When using complex expression, it is good practice to add text either as a multiline comment or inline comments to help you remember.

```
/*
Labels each region with its highest (in altitude) airport(s)
and altitude, eg 'AMBLER : 264m' for the 'Northwest Artic' region
*/
with_variable(
  'airport_alti', -- stores the highest altitude of the region
  aggregate(
    'airports',
    'max',
    "ELEV", -- the field containing the altitude
    -- and limit the airports to the region they are within
    filter := within( @geometry, geometry( @parent ) )
```

```
  ),
    aggregate( -- finds airports at the same altitude in the region
      'airports',
      'concatenate',
      "NAME",
      filter := within( @geometry, geometry( @parent ) )
        and "ELEV" = @airport_alti
    )
    || ' : ' || @airport_alti || 'm'
    -- using || allows regions without airports to be skipped
)
```

- Above the expression editor, a set of tools helps you:
  -  Clear the expression editor
  - Create and manage *user expressions*
- Under the expression editor, you find:
  - A set of basic operators to help you build the expression
  - An indication of the expected format of output when you are data-defining feature properties
  - A live *Output preview* of the expression (up to 60 characters), evaluated on the first feature of the Layer by default. To view output preview text exceeding 60 characters, you can hover your cursor over the text to display a tooltip pop-up containing the entire output preview. To copy the output preview text onto your clipboard, right-click on the output preview text and select  *Copy Expression Value*.

    You can browse and evaluate other features of the layer using the *Feature* combobox (the values are taken from the *display name* property of the layer).

    In case of error, it indicates it and you can access the details with the provided hyperlink.
- A function selector displays the list of functions, variables, fields… organized in groups. A search box is available to filter the list and quickly find a particular function or field. Double-clicking an item adds it to the expression editor.
- A help panel displays help for each selected item in the function selector.

> **Tip**
>
> Press `Ctrl+Click` when hovering a function name in an expression to automatically display its help in the dialog.

A field's values widget shown when a field is selected in the function selector helps to fetch features attributes:

- Look for a particular field value
- Display the list of *All Unique* or *10 Samples* values. Also available from right-click.

  When the field is mapped with another layer or a set of values, i.e. if the *field widget* is of *RelationReference*, *ValueRelation* or *ValueMap* type, it's possible to list all the values of the mapped field (from the referenced layer, table or list). Moreover, you can filter this list to  *Only show values in use* in the current field.

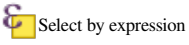
Double-clicking a field value in the widget adds it to the expression editor.

> **Tip**

The right panel, showing functions help or field values, can be collapsed (invisible) in the dialog. Press the *Show Values* or *Show Help* button to get it back.

Writing an expression

QGIS expressions are used to select features or set values. Writing an expression in QGIS follows some rules:

1. **The dialog defines the context**: if you are used to SQL, you probably know queries of the type *select features from layer where condition* or *update layer set field = new_value where condition*. A QGIS expression also needs all these information but the tool you use to open the expression builder dialog provides parts of them. For example, giving a layer (`buildings`) with a field (`height`):
   - pressing the  tool means that you want to "select features from buildings". The **condition** is the only information you need to provide in the expression text widget, e.g. type `"height" > 20` to select buildings that are higher than 20.
   - with this selection made, pressing the  button and choosing "height" as *Update existing field*, you already provide the command "update buildings set height = ??? where height > 20". The only remaining bits you have to provide in this case is the **new value**, e.g. just enter `50` in the expression editor textbox to set the height of the previously selected buildings.
2. **Pay attention to quotes**: single quotes return a literal, so a text placed between single quotes (`'145'`) is interpreted as a string. Double quotes will give you the value of that text so use them for fields (`"myfield"`). Fields can also be used without quotes (`myfield`). No quotes for numbers (`3.16`).

Note

Functions normally take as argument a string for field name. Do:

```
attribute( @atlas_feature, 'height' ) -- returns the value stored in the
↪"height" attribute of the current atlas feature
```

And not:

```
attribute( @atlas_feature, "height" ) -- fetches the value of the attribute␣
↪named "height" (e.g. 100), and use that value as a field
                                      -- from which to return the atlas␣
↪feature value. Probably wrong as a field named "100" may not exist.
```

Tip

Use named parameters to ease expression reading

Some functions require many parameters to be set. The expression engine supports the use of named parameters. This means that instead of writing the cryptic expression `clamp( 1, 2, 9)`, you can use `clamp( min:=1, value:=2, max:=9)`. This also allows arguments to be switched, e.g. `clamp( value:=2, max:=9, min:=1)`. Using named parameters helps clarify what the arguments for an expression function refer to, which is helpful when you are trying to interpret an expression later!

Some use cases of expressions

- From the Field Calculator, calculate a "pop_density" field using the existing "total_pop" and "area_km2" fields:

```
"total_pop" / "area_km2"
```

- Label or categorize features based on their area:

```
CASE WHEN $area > 10 000 THEN 'Larger' ELSE 'Smaller' END
```

- Update the field "density_level" with categories according to the "pop_density" values:

```
CASE WHEN "pop_density" < 50 THEN 'Low population density'
     WHEN "pop_density" >= 50 and "pop_density" < 150 THEN 'Medium population␣
↪density'
     WHEN "pop_density" >= 150 THEN 'High population density'
END
```

- Apply a categorized style to all the features according to whether their average house price is smaller or higher than 10000€ per square metre:

```
"price_m2" > 10000
```

- Using the "Select By Expression…" tool, select all the features representing areas of "High population density" and whose average house price is higher than 10000€ per square metre:

```
"density_level" = 'High population density' and "price_m2" > 10000
```

  The previous expression could also be used to define which features to label or show on the map.

- Select features that overlap a natural zone from the "lands" layer:

```
overlay_intersects( layer:='lands', filter:="zone_type"='Natural' )
```

- Count for each feature the number of buildings they contain:

```
array_length( overlay_contains( layer:='buildings', expression:=@id ) )
```

- Create a different symbol (type) for the layer, using the geometry generator:

```
point_on_surface( @geometry )
```

- Given a point feature, generate a closed line (using `make_line`) around its geometry:

```
make_line(
  -- using an array of points placed around the original
  array_foreach(
    -- list of angles for placing the projected points (every 90°)
    array:=generate_series( 0, 360, 90 ),
    -- translate the point 20 units in the given direction (angle)
    expression:=project( @geometry, distance:=20, azimuth:=radians( @element )␣
↪)
  )
)
```

- In a print layout label, display the name of the "airports" features that are within the layout "Map 1" item:

```
with_variable( 'extent',
               map_get( item_variables( 'Map 1' ), 'map_extent' ),
               aggregate( 'airports', 'concatenate', "NAME",
                          intersects( @geometry, @extent ), ' ,'
                        )
             )
```

Saving Expressions

Using the Add current expression to user expressions button above the expression editor frame, you can save important expressions you want to have quick access to. These are available from the **User expressions** group in the middle panel. They are saved under the *user profile* (`<userprofile>/QGIS/QGIS3.ini` file) and available in all expression dialogs inside all projects of the current user profile.

A set of tools available above the expression editor frame helps you manage the user expressions:

- Add the current expression to user expressions: store the expression in the user profile. A label and a help text can be added for easy identification.
- Edit selected expression from user expressions, as well as their help and label
- Remove selected expression from user expressions
- Import user expressions from a `.json` file into the active user profile folder
- Export user expressions as a `.json` file; all the user expressions in the user profile `QGIS3.ini` file are shared

9.1.2 Function Editor

With the *Function Editor* tab, you are able to write your own functions in Python language. This provides a handy and comfortable way to address particular needs that would not be covered by the predefined functions.

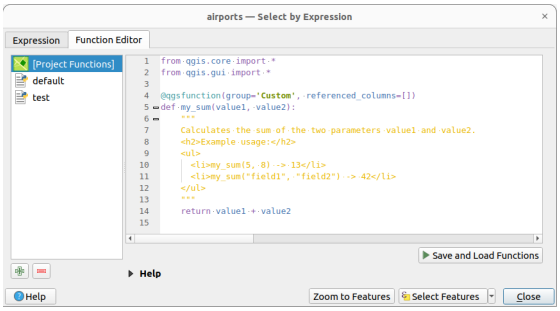


Fig. 9.2: The Function Editor tab

To create a new function:

1. Press the New File button.
2. Select where to store the new function. You can choose to save it either in a *Function file* or in *Project functions*.
   - If you select *Function file*, you need to enter a name for the Python `.py` file, which is based on a QGIS template file and is stored in the `/python/expressions` folder under the active *user profile* directory, thus available for all projects you may open with that profile.

     Custom Python functions stored as Function files are auto loaded whenever you start a new project under the user profile. Be aware that for sharing such Python functions, you need to share the `.py` file in the `/python/expressions` folder.
   - If you select *Project functions*, the new function is stored in the project file and only available for that project.

     Custom Python functions stored as Project functions can be easily shared by sharing the project file where they were saved. When opening a project with project functions, QGIS can load or ignore them depending on whether *embedded Python code is enabled*.

   A new item is added in the left panel of the *Function Editor* tab.

3. The right panel displays the content of the file: a python script template.
4. Uncomment the code block and start editing it according to your needs. You can rely on code autocompletion, code formatting and typing helpers, as defined in *Python settings*. Online API documentation for Qt and PyQGIS objects is also accessible pressing the `F1` key.

   You will find in the *Help* widget general information for setting the @qgsfunction decorator.
5. Press the  *Save and Load Functions* button. The function you wrote is added to the functions tree in the *Expression* tab, by default under the `Custom` group.
6. Enjoy your new function.
7. If the function requires improvements, enable the *Function Editor* tab, do the changes and press again the  *Save and Load Functions* button to make them available in the file, hence in any expression tab.

To delete a custom function:

1. Enable the *Function Editor* tab
2. Select the function in the list
3. Press the  Remove selected function. The function is removed from the list and, depending on the storage, the corresponding `.py` file is deleted from the user profile folder, or the Python functions are removed from the project file.

> **Note**
>
> When a project file with Project functions is closed, the corresponding Project functions are unloaded and are no longer available in the QGIS session.

Example

Here's a short example on how to create your own `my_sum` function that will operate with two values.

```
from qgis.core import *
from qgis.gui import *

@qgsfunction(group='Custom', referenced_columns=[])
def my_sum(value1, value2):
    """
    Calculates the sum of the two parameters value1 and value2.
    <h2>Example usage:</h2>
    <ul>
      <li>my_sum(5, 8) -> 13</li>
      <li>my_sum("field1", "field2") -> 42</li>
    </ul>
    """
    return value1 + value2
```

The `@qgsfunction` decorator accepts the following arguments:

- The `group` argument indicates the group in which the function should be listed in the Expression dialog.
- `usesgeometry`: Set this to `True` if the expression requires access to the features geometry. By default `False`.
- `handlesnull`: Set this to `True` if the expression has custom handling for NULL values. If `False` (default), the result will always be NULL as soon as any parameter is NULL.
- `referenced_columns`: An array of attribute names that are required to run the function. Defaults to `[QgsFeatureRequest.ALL_ATTRIBUTES]`. Specifying a subset of fields or an empty list will result in a faster execution.

- `params_as_list`: Set this to `True` to pass the function parameters as a list. If `False` (default), the function will receive the parameters as individual arguments.

The function itself takes any number and type of parameters you want to pass to your function. These should be set before the following optional arguments:

- `feature`: the current feature
- `parent`: the `QgsExpression` object
- `context`: If there is an argument called `context` found at the last position, this variable will contain a `QgsExpressionContext` object, that gives access to various additional information like expression variables. E.g. `context.variable( 'layer_id' )`

If those arguments are present in the function signature, they will be automatically passed to the function, without the need to specify them in the expression.

The previous example function can then be used in expressions:

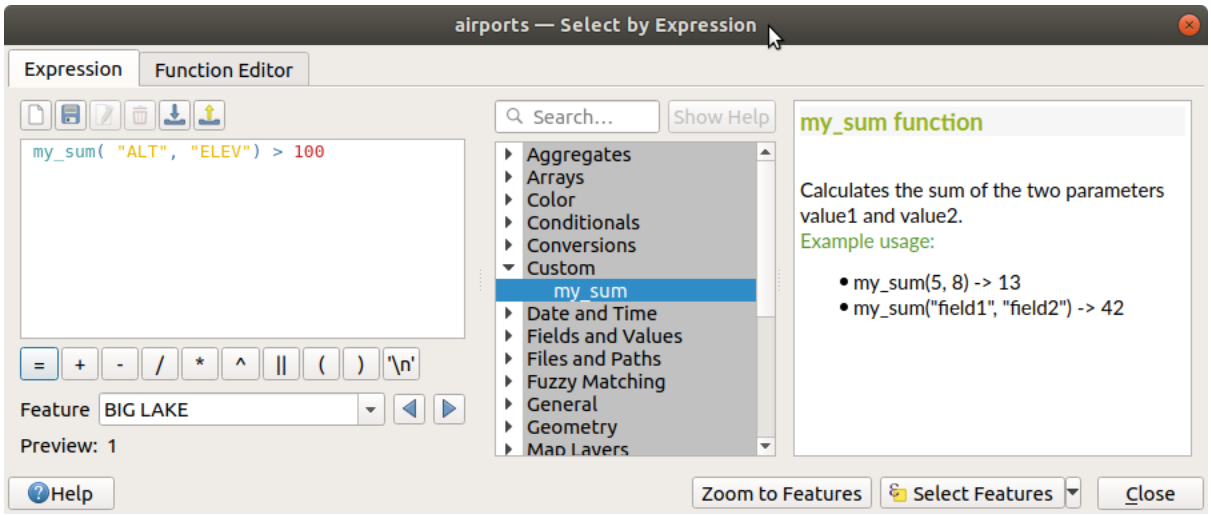


Fig. 9.3: Custom Function added to the Expression tab

Find more examples and additional arguments for custom functions at `qgsfunction`. Further information about creating Python code can be found in the PyQGIS-Developer-Cookbook.

9.2 List of functions

The functions, operators and variables available in QGIS are listed below, grouped by categories.

9.2.1 Aggregates Functions

This group contains functions which aggregate values over layers and fields.

aggregate

Returns an aggregate value calculated using features from another layer.

Syntax	aggregate(layer, aggregate, expression, [filter], [concatenator=''], [order_by]) [] marks optional arguments
Arguments	• **layer** - a string, representing either a layer name or layer ID • **aggregate** - a string corresponding to the aggregate to calculate. Valid options are: – count – count_distinct – count_missing – minimum or min – maximum or max – sum – mean – median – stdev – stdevsample – range – minority – majority – q1: first quartile – q3: third quartile – iqr: inter quartile range – min_length: minimum string length – max_length: maximum string length – concatenate: join strings with a concatenator – concatenate_unique: join unique strings with a concatenator – collect: create an aggregated multipart geometry – array_agg: create an array of aggregated values • **expression** - sub expression or field name to aggregate • **filter** - optional filter expression to limit the features used for calculating the aggregate. Fields and geometry are from the features on the joined layer. The source feature can be accessed with the variable @parent. • **concatenator** - optional string to use to join values for 'concatenate' and 'concatenate_unique' aggregates • **order_by** - optional filter expression to order the features used for calculating the aggregate. Fields and geometry are from the features on the joined layer. By default, the features will be returned in an unspecified order.
Examples	• `aggregate(layer:='rail_stations',aggregate:='sum', expression:="passengers")` → sum of all values from the passengers field in the rail_stations layer • `aggregate('rail_stations','sum', "passengers"/7)` → calculates a daily average of "passengers" by dividing the "passengers" field by 7 before summing the values • `aggregate(layer:='rail_stations',aggregate:='sum', expression:="passengers",filter:="class">3)` → sums up all values from the "passengers" field from features where the "class" attribute is greater than 3 only • `aggregate(layer:='rail_stations',aggregate:='concatenate', expression:="name", concatenator:=',')` → comma separated list of the name field for all features in the rail_stations layer • `aggregate(layer:='countries', aggregate:='max', expression:="code", filter:=intersects( @geometry, geometry(@parent) ) )` → The country code of an intersecting country on the layer 'countries' • `aggregate(layer:='rail_stations',aggregate:='sum', expression:="passengers",filter:=contains( @atlas_geometry, @geometry ) )` → sum of all values from the passengers field in the rail_stations within the current atlas feature • `aggregate(layer:='rail_stations', aggregate:='collect', expression:=centroid(@geometry), filter:=attribute(@parent,'name') )` → aggregates centroid geometries of the rail_stations of the same region as current feature

array_agg

Returns an array of aggregated values from a field or expression.

Syntax	array_agg(expression, [group_by], [filter], [order_by]) [] marks optional arguments
Arguments	• **expression** - sub expression of field to aggregate • **group_by** - optional expression to use to group aggregate calculations • **filter** - optional expression to use to filter features used to calculate aggregate • **order_by** - optional expression to use to order features used to calculate aggregate. By default, the features will be returned in an unspecified order.
Examples	• `array_agg("name",group_by:="state")` → list of name values, grouped by state field

collect

Returns the multipart geometry of aggregated geometries from an expression

Syntax	collect(expression, [group_by], [filter]) [] marks optional arguments
Arguments	• **expression** - geometry expression to aggregate • **group_by** - optional expression to use to group aggregate calculations • **filter** - optional expression to use to filter features used to calculate aggregate
Examples	• `collect( @geometry )` → multipart geometry of aggregated geometries • `collect( centroid(@geometry), group_by:="region", filter:= "use" = 'civilian' )` → aggregated centroids of the civilian features based on their region value

concatenate

Returns all aggregated strings from a field or expression joined by a delimiter.

Syntax	concatenate(expression, [group_by], [filter], [concatenator], [order_by]) [] marks optional arguments
Arguments	• **expression** - sub expression of field to aggregate • **group_by** - optional expression to use to group aggregate calculations • **filter** - optional expression to use to filter features used to calculate aggregate • **concatenator** - optional string to use to join values. Empty by default. • **order_by** - optional expression to use to order features used to calculate aggregate. By default, the features will be returned in an unspecified order.
Examples	• `concatenate("town_name",group_by:="state",concatenator:=', ')` → comma separated list of town_names, grouped by state field

concatenate_unique

Returns all unique strings from a field or expression joined by a delimiter.

Syntax	concatenate_unique(expression, [group_by], [filter], [concatenator], [order_by]) [] marks optional arguments
Arguments	• **expression** - sub expression of field to aggregate • **group_by** - optional expression to use to group aggregate calculations • **filter** - optional expression to use to filter features used to calculate aggregate • **concatenator** - optional string to use to join values. Empty by default. • **order_by** - optional expression to use to order features used to calculate aggregate. By default, the features will be returned in an unspecified order.
Examples	• `concatenate_unique("town_name",group_by:="state", concatenator:=',')` → comma separated list of unique town_names, grouped by state field

count

Returns the count of matching features.

Syntax	count(expression, [group_by], [filter]) [] marks optional arguments
Arguments	• **expression** - sub expression of field to aggregate • **group_by** - optional expression to use to group aggregate calculations • **filter** - optional expression to use to filter features used to calculate aggregate
Examples	• `count("stations",group_by:="state")` → count of stations, grouped by state field

count_distinct

Returns the count of distinct values.

Syntax	count_distinct(expression, [group_by], [filter]) [] marks optional arguments
Arguments	• **expression** - sub expression of field to aggregate • **group_by** - optional expression to use to group aggregate calculations • **filter** - optional expression to use to filter features used to calculate aggregate
Examples	• `count_distinct("stations",group_by:="state")` → count of distinct stations values, grouped by state field

count_missing

Returns the count of missing (NULL) values.

Syntax	count_missing(expression, [group_by], [filter]) [] marks optional arguments
Arguments	• **expression** - sub expression of field to aggregate • **group_by** - optional expression to use to group aggregate calculations • **filter** - optional expression to use to filter features used to calculate aggregate
Examples	• `count_missing("stations",group_by:="state")` → count of missing (NULL) station values, grouped by state field

iqr

Returns the calculated inter quartile range from a field or expression.

Syntax	iqr(expression, [group_by], [filter]) [] marks optional arguments
Arguments	• **expression** - sub expression of field to aggregate • **group_by** - optional expression to use to group aggregate calculations • **filter** - optional expression to use to filter features used to calculate aggregate
Examples	• `iqr("population",group_by:="state")` → inter quartile range of population value, grouped by state field

majority

Returns the aggregate majority of values (most commonly occurring value) from a field or expression.

Syntax	majority(expression, [group_by], [filter]) [] marks optional arguments
Arguments	• **expression** - sub expression of field to aggregate • **group_by** - optional expression to use to group aggregate calculations • **filter** - optional expression to use to filter features used to calculate aggregate
Examples	• `majority("class",group_by:="state")` → most commonly occurring class value, grouped by state field

max_length

Returns the maximum length of strings from a field or expression.

Syntax	max_length(expression, [group_by], [filter]) [] marks optional arguments
Arguments	• **expression** - sub expression of field to aggregate • **group_by** - optional expression to use to group aggregate calculations • **filter** - optional expression to use to filter features used to calculate aggregate
Examples	• `max_length("town_name",group_by:="state")` → maximum length of town_name, grouped by state field

maximum

Returns the aggregate maximum value from a field or expression.

Syntax	maximum(expression, [group_by], [filter]) [] marks optional arguments
Arguments	• **expression** - sub expression of field to aggregate • **group_by** - optional expression to use to group aggregate calculations • **filter** - optional expression to use to filter features used to calculate aggregate
Examples	• `maximum("population",group_by:="state")` → maximum population value, grouped by state field

mean

Returns the aggregate mean value from a field or expression.

Syntax	mean(expression, [group_by], [filter]) [] marks optional arguments
Arguments	• **expression** - sub expression of field to aggregate • **group_by** - optional expression to use to group aggregate calculations • **filter** - optional expression to use to filter features used to calculate aggregate
Examples	• `mean("population",group_by:="state")` → mean population value, grouped by state field

median

Returns the aggregate median value from a field or expression.

Syntax	median(expression, [group_by], [filter]) [] marks optional arguments
Arguments	• **expression** - sub expression of field to aggregate • **group_by** - optional expression to use to group aggregate calculations • **filter** - optional expression to use to filter features used to calculate aggregate
Examples	• `median("population",group_by:="state")` → median population value, grouped by state field

min_length

Returns the minimum length of strings from a field or expression.

Syntax	min_length(expression, [group_by], [filter]) [] marks optional arguments
Arguments	• **expression** - sub expression of field to aggregate • **group_by** - optional expression to use to group aggregate calculations • **filter** - optional expression to use to filter features used to calculate aggregate
Examples	• `min_length("town_name",group_by:="state")` → minimum length of town_name, grouped by state field

minimum

Returns the aggregate minimum value from a field or expression.

Syntax	minimum(expression, [group_by], [filter]) [] marks optional arguments
Arguments	• **expression** - sub expression of field to aggregate • **group_by** - optional expression to use to group aggregate calculations • **filter** - optional expression to use to filter features used to calculate aggregate
Examples	• `minimum("population",group_by:="state")` → minimum population value, grouped by state field

minority

Returns the aggregate minority of values (least occurring value) from a field or expression.

Syntax	minority(expression, [group_by], [filter]) [] marks optional arguments
Arguments	• **expression** - sub expression of field to aggregate • **group_by** - optional expression to use to group aggregate calculations • **filter** - optional expression to use to filter features used to calculate aggregate
Examples	• `minority("class",group_by:="state")` → least occurring class value, grouped by state field

q1

Returns the calculated first quartile from a field or expression.

Syntax	q1(expression, [group_by], [filter]) [] marks optional arguments
Arguments	• **expression** - sub expression of field to aggregate • **group_by** - optional expression to use to group aggregate calculations • **filter** - optional expression to use to filter features used to calculate aggregate
Examples	• `q1("population",group_by:="state")` → first quartile of population value, grouped by state field

q3

Returns the calculated third quartile from a field or expression.

Syntax	q3(expression, [group_by], [filter]) [] marks optional arguments
Arguments	• **expression** - sub expression of field to aggregate • **group_by** - optional expression to use to group aggregate calculations • **filter** - optional expression to use to filter features used to calculate aggregate
Examples	• `q3("population",group_by:="state")` → third quartile of population value, grouped by state field

range

Returns the aggregate range of values (maximum - minimum) from a field or expression.

Syntax	range(expression, [group_by], [filter]) [] marks optional arguments
Arguments	• **expression** - sub expression of field to aggregate • **group_by** - optional expression to use to group aggregate calculations • **filter** - optional expression to use to filter features used to calculate aggregate
Examples	• `range("population",group_by:="state")` → range of population values, grouped by state field

relation_aggregate

Returns an aggregate value calculated using all matching child features from a layer relation.

Syntax	relation_aggregate(relation, aggregate, expression, [concatenator=''], [order_by]) [] marks optional arguments
Arguments	• **relation** - a string, representing a relation ID • **aggregate** - a string corresponding to the aggregate to calculate. Valid options are: – count – count_distinct – count_missing – minimum or min – maximum or max – sum – mean – median – stdev – stdevsample – range – minority – majority – q1: first quartile – q3: third quartile – iqr: inter quartile range – min_length: minimum string length – max_length: maximum string length – concatenate: join strings with a concatenator – concatenate_unique: join unique strings with a concatenator – collect: create an aggregated multipart geometry – array_agg: create an array of aggregated values • **expression** - sub expression or field name to aggregate • **concatenator** - optional string to use to join values for 'concatenate' aggregate • **order_by** - optional expression to order the features used for calculating the aggregate. Fields and geometry are from the features on the joined layer. By default, the features will be returned in an unspecified order.
Examples	• `relation_aggregate(relation:='my_relation', aggregate:='mean',expression:="passengers")` → mean value of all matching child features using the 'my_relation' relation • `relation_aggregate('my_relation','sum', "passengers"/7)` → sum of the passengers field divided by 7 for all matching child features using the 'my_relation' relation • `relation_aggregate('my_relation','concatenate', "towns", concatenator:=',')` → comma separated list of the towns field for all matching child features using the 'my_relation' relation • `relation_aggregate('my_relation','array_agg', "id")` → array of the id field from all matching child features using the 'my_relation' relation

Further reading: *Setting relations between multiple layers*

stdev

Returns the aggregate standard deviation value from a field or expression.

Syntax	stdev(expression, [group_by], [filter]) [] marks optional arguments
Arguments	• **expression** - sub expression of field to aggregate • **group_by** - optional expression to use to group aggregate calculations • **filter** - optional expression to use to filter features used to calculate aggregate
Examples	• `stdev("population",group_by:="state")` → standard deviation of population value, grouped by state field

sum

Returns the aggregate summed value from a field or expression.

Syntax	sum(expression, [group_by], [filter]) [] marks optional arguments
Arguments	• **expression** - sub expression of field to aggregate • **group_by** - optional expression to use to group aggregate calculations • **filter** - optional expression to use to filter features used to calculate aggregate
Examples	• `sum("population",group_by:="state")` → summed population value, grouped by state field

9.2.2 Array Functions

This group contains functions to create and manipulate arrays (also known as list data structures). The order of values within the array matters, unlike the *'map' data structure*, where the order of key-value pairs is irrelevant and values are identified by their keys.

array

Returns an array containing all the values passed as parameter.

Syntax	array(value1, value2, ...)
Arguments	• **value** - a value
Examples	• `array(2,10)` → [2, 10] • `array(2,10)[0]` → 2

array_all

Returns TRUE if an array contains all the values of a given array.

Syntax	array_all(array_a, array_b)
Arguments	• **array_a** - an array • **array_b** - the array of values to search
Examples	• `array_all(array(1,2,3),array(2,3))` → TRUE • `array_all(array(1,2,3),array(1,2,4))` → FALSE

array_append

Returns an array with the given value added at the end.

Syntax	array_append(array, value)
Arguments	• **array** - an array • **value** - the value to add
Examples	• `array_append(array(1,2,3),4)` → [1, 2, 3, 4]

array_cat

Returns an array containing all the given arrays concatenated.

Syntax	array_cat(array1, array2, …)
Arguments	• **array** - an array
Examples	• `array_cat(array(1,2),array(2,3))` → [1, 2, 2, 3]

array_contains

Returns TRUE if an array contains the given value.

Syntax	array_contains(array, value)
Arguments	• **array** - an array • **value** - the value to search
Examples	• `array_contains(array(1,2,3),2)` → TRUE

array_count

Counts the number of occurrences of a given value in an array.

Syntax	array_count(array, value)
Arguments	• **array** - an array • **value** - the value to count
Examples	• `array_count(array('a', 'b', 'c', 'b'), 'b')` → 2

array_distinct

Returns an array containing distinct values of the given array.

Syntax	array_distinct(array)
Arguments	• **array** - an array
Examples	• `array_distinct(array(1,2,3,2,1))` → [1, 2, 3]

array_filter

Returns an array with only the items for which the expression evaluates to true.

Syntax	array_filter(array, expression, [limit=0]) [] marks optional arguments
Arguments	• **array** - an array • **expression** - an expression to evaluate on each item. The variable *@element* will be replaced by the current value. • **limit** - maximum number of elements to be returned. Use 0 to return all values.
Examples	• `array_filter(array(1,2,3),@element < 3)` → [1, 2] • `array_filter(array(1,2,3),@element < 3, 1)` → [1]

array_find

Returns the lowest index (0 for the first one) of a value within an array. Returns -1 if the value is not found.

Syntax	array_find(array, value)
Arguments	• **array** - an array • **value** - the value to search
Examples	• `array_find(array('a', 'b', 'c'), 'b')` → 1 • `array_find(array('a', 'b', 'c', 'b'), 'b')` → 1

array_first

Returns the first value of an array.

Syntax	array_first(array)
Arguments	• **array** - an array
Examples	• `array_first(array('a','b','c'))` → 'a'

array_foreach

Returns an array with the given expression evaluated on each item.

Syntax	array_foreach(array, expression)
Arguments	• **array** - an array • **expression** - an expression to evaluate on each item. The variable *@element* will be replaced by the current value and the variable *@counter* by the current index (starting with 0).
Examples	• `array_foreach(array('a','b','c'),upper(@element))` → ['A', 'B', 'C'] • `array_foreach(array(1,2,3),@element + 10)` → [11, 12, 13] • `array_foreach(array(1,2,3),@element + @counter)` → [1, 3, 5]

array_get

Returns the Nth value (0 for the first one) or the last -Nth value (-1 for the last one) of an array.

Syntax	array_get(array, pos)
Arguments	• **array** - an array • **pos** - the index to get (0 based)
Examples	• `array_get(array('a','b','c'),1)` → 'b' • `array_get(array('a','b','c'),-1)` → 'c'

Hint

You can also use the *index operator ([])* to get a value from an array.

array_insert

Returns an array with the given value added at the given position.

Syntax	array_insert(array, pos, value)
Arguments	• **array** - an array • **pos** - the position where to add (0 based) • **value** - the value to add
Examples	• `array_insert(array(1,2,3),1,100)` → [1, 100, 2, 3]

array_intersect

Returns TRUE if at least one element of array1 exists in array2.

Syntax	array_intersect(array1, array2)
Arguments	• **array1** - an array • **array2** - another array
Examples	• `array_intersect(array(1,2,3,4),array(4,0,2,5))` → TRUE

array_last

Returns the last value of an array.

Syntax	array_last(array)
Arguments	• **array** - an array
Examples	• `array_last(array('a','b','c'))` → 'c'

array_length

Returns the number of elements of an array.

Syntax	array_length(array)
Arguments	• **array** - an array
Examples	• `array_length(array(1,2,3))` → 3

array_majority

Returns the most common values in an array.

Syntax	array_majority(array, [option='all']) [] marks optional arguments
Arguments	• **array** - an array • **option='all'** - a string specifying the return values handling. Valid options are: – all: Default, all most common values are returned in an array. – any: Returns one of the most common values. – median: Returns the median of the most common values. Non arithmetic values are ignored. – real_majority: Returns the value which occurs more than half the size of the array.
Examples	• `array_majority(array(0,1,42,42,43), 'all')` → [42] • `array_majority(array(0,1,42,42,43,1), 'all')` → [42, 1] • `array_majority(array(0,1,42,42,43,1), 'any')` → 1 or 42 • `array_majority(array(0,1,1,2,2), 'median')` → 1.5 • `array_majority(array(0,1,42,42,43), 'real_majority')` → NULL • `array_majority(array(0,1,42,42,43,42), 'real_majority')` → NULL • `array_majority(array(0,1,42,42,43,42,42), 'real_majority')` → 42

array_max

Returns the maximum value of an array.

Syntax	array_max(array)
Arguments	• **array** - an array
Examples	• `array_max(array(0,42,4,2))` → 42

array_mean

Returns the mean of arithmetic values in an array. Non numeric values in the array are ignored.

Syntax	array_mean(array)
Arguments	• **array** - an array
Examples	• `array_mean(array(0,1,7,66.6,135.4))` → 42 • `array_mean(array(0,84,'a','b','c'))` → 42

array_median

Returns the median of arithmetic values in an array. Non arithmetic values in the array are ignored.

Syntax	array_median(array)
Arguments	• **array** - an array
Examples	• `array_median(array(0,1,42,42,43))` → 42 • `array_median(array(0,1,2,42,'a','b'))` → 1.5

array_min

Returns the minimum value of an array.

Syntax	array_min(array)
Arguments	• **array** - an array
Examples	• `array_min(array(43,42,54))` → 42

array_minority

Returns the less common values in an array.

Syntax	array_minority(array, [option='all']) [] marks optional arguments
Arguments	• **array** - an array • **option='all'** - a string specifying the return values handling. Valid options are: – all: Default, all less common values are returned in an array. – any: Returns one of the less common values. – median: Returns the median of the less common values. Non arithmetic values are ignored. – real_minority: Returns values which occur less than half the size of the array.
Examples	• `array_minority(array(0,42,42), 'all')` → [0] • `array_minority(array(0,1,42,42), 'all')` → [0, 1] • `array_minority(array(0,1,42,42,43,1), 'any')` → 0 or 43 • `array_minority(array(1,2,3,3), 'median')` → 1.5 • `array_minority(array(0,1,42,42,43), 'real_minority')` → [42, 43, 0, 1] • `array_minority(array(0,1,42,42,43,42), 'real_minority')` → [42, 43, 0, 1] • `array_minority(array(0,1,42,42,43,42,42), 'real_minority')` → [43, 0, 1]

array_prepend

Returns an array with the given value added at the beginning.

Syntax	array_prepend(array, value)
Arguments	• **array** - an array • **value** - the value to add
Examples	• `array_prepend(array(1,2,3),0)` → [0, 1, 2, 3]

array_prioritize

Returns an array sorted using the ordering specified in another array. Values which are present in the first array but are missing from the second array will be added to the end of the result.

Syntax	array_prioritize(array, array_prioritize)
Arguments	• **array** - an array • **array_prioritize** - an array with values ordered by priority
Examples	• `array_prioritize(array(1, 8, 2, 5), array(5, 4, 2, 1, 3, 8))` → [5, 2, 1, 8] • `array_prioritize(array(5, 4, 2, 1, 3, 8), array(1, 8, 6, 5))` → [1, 8, 5, 4, 2, 3]

array_remove_all

Returns an array with all the entries of the given value removed.

Syntax	array_remove_all(array, value)
Arguments	• **array** - an array • **value** - the values to remove
Examples	• `array_remove_all(array('a','b','c','b'),'b')` → ['a', 'c']

array_remove_at

Returns an array with the item at the given index removed. Supports positive (0 for the first element) and negative (the last -Nth value, -1 for the last element) index.

Syntax	array_remove_at(array, pos)
Arguments	• **array** - an array • **pos** - the position to remove (0 based)
Examples	• `array_remove_at(array(1, 2, 3), 1)` → [1, 3] • `array_remove_at(array(1, 2, 3), -1)` → [1, 2]

array_replace

Returns an array with the supplied value, array, or map of values replaced.

Value & array variant

Returns an array with the supplied value or array of values replaced by another value or an array of values.

Syntax	array_replace(array, before, after)
Arguments	• **array** - the input array • **before** - the value or array of values to replace • **after** - the value or array of values to use as a replacement
Examples	• `array_replace(array('QGIS','SHOULD','ROCK'),'SHOULD', 'DOES')` → ['QGIS', 'DOES', 'ROCK'] • `array_replace(array(3,2,1),array(1,2,3),array(7,8,9))` → [9, 8, 7] • `array_replace(array('Q','G','I','S'),array('Q','S'),'-')` → ['-', 'G', 'I', '-']

Map variant

Returns an array with the supplied map keys replaced by their paired values.

Syntax	array_replace(array, map)
Arguments	• **array** - the input array • **map** - the map containing keys and values
Examples	• `array_replace(array('APP', 'SHOULD', 'ROCK'),map('APP', 'QGIS','SHOULD','DOES'))` → ['QGIS', 'DOES', 'ROCK']

Further reading: *replace*, *regexp_replace*

array_reverse

Returns the given array with array values in reversed order.

Syntax	array_reverse(array)
Arguments	• **array** - an array
Examples	• `array_reverse(array(2,4,0,10))` → [10, 0, 4, 2]

array_slice

Returns a portion of the array. The slice is defined by the start_pos and end_pos arguments.

Syntax	array_slice(array, start_pos, end_pos)
Arguments	• **array** - an array • **start_pos** - the index of the start position of the slice (0 based). The start_pos index is included in the slice. If you use a negative start_pos, the index is counted from the end of the list (-1 based). • **end_pos** - the index of the end position of the slice (0 based). The end_pos index is included in the slice. If you use a negative end_pos, the index is counted from the end of the list (-1 based).
Examples	• `array_slice(array(1,2,3,4,5),0,3)` → [1, 2, 3, 4] • `array_slice(array(1,2,3,4,5),0,-1)` → [1, 2, 3, 4, 5] • `array_slice(array(1,2,3,4,5),-5,-1)` → [1, 2, 3, 4, 5] • `array_slice(array(1,2,3,4,5),0,0)` → [1] • `array_slice(array(1,2,3,4,5),-2,-1)` → [4, 5] • `array_slice(array(1,2,3,4,5),-1,-1)` → [5] • `array_slice(array('Dufour','Valmiera','Chugiak', 'Brighton'),1,2)` → ['Valmiera', 'Chugiak'] • `array_slice(array('Dufour','Valmiera','Chugiak', 'Brighton'),-2,-1)` → ['Chugiak', 'Brighton']

array_sort

Returns the provided array with its elements sorted.

Syntax	array_sort(array, [ascending=true]) [] marks optional arguments
Arguments	• **array** - an array • **ascending** - set this parameter to false to sort the array in descending order
Examples	• `array_sort(array(3,2,1))` → [1, 2, 3]

array_sum

Returns the sum of arithmetic values in an array. Non numeric values in the array are ignored.

Syntax	array_sum(array)
Arguments	• **array** - an array
Examples	• `array_sum(array(0,1,39.4,1.6,'a'))` → 42.0

array_to_string

Concatenates array elements into a string separated by a delimiter and using optional string for empty values.

Syntax	array_to_string(array, [delimiter=','], [empty_value='']) [] marks optional arguments
Arguments	• **array** - the input array • **delimiter** - the string delimiter used to separate concatenated array elements • **empty_value** - the optional string to use as replacement for empty (zero length) matches
Examples	• `array_to_string(array('1','2','3'))` → '1,2,3' • `array_to_string(array(1,2,3),'-')` → '1-2-3' • `array_to_string(array('1','','3'),',','0')` → '1,0,3'

Further reading: *string_to_array*

generate_series

Creates an array containing a sequence of numbers.

Syntax	generate_series(start, stop, [step=1]) [] marks optional arguments
Arguments	• **start** - first value of the sequence • **stop** - value that ends the sequence once reached • **step** - value used as the increment between values
Examples	• `generate_series(1,5)` → [1, 2, 3, 4, 5] • `generate_series(5,1,-1)` → [5, 4, 3, 2, 1]

geometries_to_array

Splits a geometry into simpler geometries in an array.

Syntax	geometries_to_array(geometry)
Arguments	• **geometry** - the input geometry
Examples	• `geometries_to_array(geom_from_wkt('MultiPoint (1 2, 5 21)'))` → An array containing 'Point (1 2)' and 'Point (5 21)' • `geometries_to_array(geom_from_wkt('GeometryCollection (Polygon ((5 8, 4 1, 3 2, 5 8)),LineString (3 2, 4 2))'))` → an array of a polygon and a line geometries • `geom_to_wkt(geometries_to_array(geom_from_wkt('GeometryCollection (Polygon ((5 8, 4 1, 3 2, 5 8)),LineString (3 2, 4 2))'))[0])` → 'Polygon ((5 8, 4 1, 3 2, 5 8))' • `geometries_to_array(geom_from_wkt('MULTIPOLYGON(((5 5,0 0, 0 10,5 5)),((5 5,10 10,10 0,5 5))'))` → an array of two polygon geometries

regexp_matches

Returns an array of all strings captured by capturing groups, in the order the groups themselves appear in the supplied regular expression against a string.

Syntax	regexp_matches(string, regex, [empty_value='']) [] marks optional arguments
Arguments	• **string** - the string to capture groups from against the regular expression • **regex** - the regular expression used to capture groups • **empty_value** - the optional string to use as replacement for empty (zero length) matches
Examples	• `regexp_matches('QGIS=>rocks','(.*)=>(.*)')` → ['QGIS', 'rocks'] • `regexp_matches('key=>','(.*)=>(.*)','empty value')` → ['key', 'empty value']

Further reading: *substr*, *regexp_substr*

string_to_array

Splits string into an array using supplied delimiter and optional string for empty values.

Syntax	string_to_array(string, [delimiter=','], [empty_value='']) [] marks optional arguments
Arguments	• **string** - the input string • **delimiter** - the string delimiter used to split the input string • **empty_value** - the optional string to use as replacement for empty (zero length) matches
Examples	• `string_to_array('1,2,3',',')` → ['1', '2', '3'] • `string_to_array('1,,3',',','0')` → ['1', '0', '3']

Further reading: *array_to_string*

9.2.3 Color Functions

This group contains functions for manipulating colors.

color_cmyk

Returns a string representation of a color based on its cyan, magenta, yellow and black components

Syntax	color_cmyk(cyan, magenta, yellow, black)
Arguments	• **cyan** - cyan component of the color, as a percentage integer value from 0 to 100 • **magenta** - magenta component of the color, as a percentage integer value from 0 to 100 • **yellow** - yellow component of the color, as a percentage integer value from 0 to 100 • **black** - black component of the color, as a percentage integer value from 0 to 100
Examples	• `color_cmyk(100,50,0,10)` → '0,115,230'

color_cmyka

Returns a string representation of a color based on its cyan, magenta, yellow, black and alpha (transparency) components

Syntax	color_cmyka(cyan, magenta, yellow, black, alpha)
Arguments	• **cyan** - cyan component of the color, as a percentage integer value from 0 to 100 • **magenta** - magenta component of the color, as a percentage integer value from 0 to 100 • **yellow** - yellow component of the color, as a percentage integer value from 0 to 100 • **black** - black component of the color, as a percentage integer value from 0 to 100 • **alpha** - alpha component as an integer value from 0 (completely transparent) to 255 (opaque).
Examples	• `color_cmyka(100,50,0,10,200)` → '0,115,230,200'

color_cmykf

Returns a color object based on its cyan, magenta, yellow, black and alpha components.

Syntax	color_cmykf(cyan, magenta, yellow, black, [alpha=1.0]) [] marks optional arguments
Arguments	• **cyan** - cyan component as a float value from 0.0 to 1.0 • **magenta** - magenta component as a float value from 0.0 to 1.0 • **yellow** - yellow component as a float value from 0.0 to 1.0 • **black** - black component as a float value from 0.0 to 1.0 • **alpha** - alpha component as a float value from 0.0 to 1.0
Examples	• `color_cmykf(1,0.9,0.81,0.62)` → CMYKA: 1.00,0.90,0.81,0.62,1.00

color_grayscale_average

Applies a grayscale filter to a color and returns it. Returned type is the same as color argument, i.e. a color string representation or a color object.

Syntax	color_grayscale_average(color)
Arguments	• **color** - a color string representation or a color object
Examples	• `color_grayscale_average('255,100,50')` → '135,135,135,255' • `color_grayscale_average(color_cmykf(0.6,0.5,0.1,0.8))` → CMYKA: 0.40,0.40,0.40,0.80,1.00

color_hsl

Returns a string representation of a color based on its hue, saturation, and lightness attributes.

Syntax	color_hsl(hue, saturation, lightness)
Arguments	• **hue** - hue of the color, as an integer value from 0 to 360 • **saturation** - saturation percentage of the color as an integer value from 0 to 100 • **lightness** - lightness percentage of the color as an integer value from 0 to 100
Examples	• `color_hsl(100,50,70)` → '166,217,140'

color_hsla

Returns a string representation of a color based on its hue, saturation, lightness and alpha (transparency) attributes

Syntax	color_hsla(hue, saturation, lightness, alpha)
Arguments	• **hue** - hue of the color, as an integer value from 0 to 360 • **saturation** - saturation percentage of the color as an integer value from 0 to 100 • **lightness** - lightness percentage of the color as an integer value from 0 to 100 • **alpha** - alpha component as an integer value from 0 (completely transparent) to 255 (opaque).
Examples	• `color_hsla(100,50,70,200)` → '166,217,140,200'

color_hslf

Returns a color object based on its hue, saturation, and lightness attributes.

Syntax	color_hslf(hue, saturation, lightness, [alpha=1.0]) [] marks optional arguments
Arguments	• **hue** - hue of the color, as a float value from 0.0 to 1.0 • **saturation** - saturation of the color as a float value from 0.0 to 1.0 • **lightness** - lightness of the color as a float value from 0.0 to 1.0 • **alpha** - alpha component as a float value from 0.0 to 1.0
Examples	• `color_hslf(0.3,0.52,0.7)` → HSLA: 0.30,0.52,0.70,1.00

color_hsv

Returns a string representation of a color based on its hue, saturation, and value attributes.

Syntax	color_hsv(hue, saturation, value)
Arguments	• **hue** - hue of the color, as an integer value from 0 to 360 • **saturation** - saturation percentage of the color as an integer value from 0 to 100 • **value** - value percentage of the color as an integer from 0 to 100
Examples	• `color_hsv(40,100,100)` → '255,170,0'

color_hsva

Returns a string representation of a color based on its hue, saturation, value and alpha (transparency) attributes.

Syntax	color_hsva(hue, saturation, value, alpha)
Arguments	• **hue** - hue of the color, as an integer value from 0 to 360 • **saturation** - saturation percentage of the color as an integer value from 0 to 100 • **value** - value percentage of the color as an integer from 0 to 100 • **alpha** - alpha component as an integer value from 0 (completely transparent) to 255 (opaque)
Examples	• `color_hsva(40,100,100,200)` → '255,170,0,200'

color_hsvf

Returns a color object based on its hue, saturation, and value attributes.

Syntax	color_hsvf(hue, saturation, value, [alpha=1.0]) [] marks optional arguments
Arguments	• **hue** - hue of the color, as a float value from 0.0 to 1.0 • **saturation** - saturation of the color as a float value from 0.0 to 1.0 • **value** - value of the color as as a float value from 0.0 to 1.0 • **alpha** - alpha component as a float value from 0.0 to 1.0
Examples	• `color_hsvf(0.4,1,0.6)` → HSVA: 0.40,1.00,0.60,1.00

color_mix

Returns a color mixing the red, green, blue, and alpha values of two provided colors based on a given ratio. Returned type is the same as color arguments, i.e. a color string representation or a color object.

Syntax	color_mix(color1, color2, ratio)
Arguments	• **color1** - a color string or a color object • **color2** - a color string or a color object • **ratio** - a ratio
Examples	• `color_mix_rgb('0,0,0','255,255,255',0.5)` → '127,127,127,255' • `color_mix(color_cmykf(0.9,0.9,0.9,0.9),color_cmykf(0.1,0.1,0.1,0.1),0.5)` → CMYKA: 0.50,0.50,0.50,0.50,1.00

color_mix_rgb

Returns a string representing a color mixing the red, green, blue, and alpha values of two provided colors based on a given ratio.

Syntax	color_mix_rgb(color1, color2, ratio)
Arguments	• **color1** - a color string • **color2** - a color string • **ratio** - a ratio
Examples	• `color_mix_rgb('0,0,0','255,255,255',0.5)` → '127,127,127,255'

color_part

Returns a specific component from a color string or color object, e.g., the red component or alpha component.

Syntax	color_part(color, component)
Arguments	• **color** - a color string or a color object • **component** - a string corresponding to the color component to return. Valid options are: – red: RGB red component (0-255) – green: RGB green component (0-255) – blue: RGB blue component (0-255) – alpha: alpha (transparency) value (0-255) – hue: HSV hue (0-360) – saturation: HSV saturation (0-100) – value: HSV value (0-100) – hsl_hue: HSL hue (0-360) – hsl_saturation: HSL saturation (0-100) – lightness: HSL lightness (0-100) – cyan: CMYK cyan component (0-100) – magenta: CMYK magenta component (0-100) – yellow: CMYK yellow component (0-100) – black: CMYK black component (0-100)
Examples	• `color_part('200,10,30','green')` → 10 • `to_int(color_part(color_cmykf(0.1,0.2,0.3,0.9),'black'))` → 90

color_rgb

Returns a string representation of a color based on its red, green, and blue components.

Syntax	color_rgb(red, green, blue)
Arguments	• **red** - red component as an integer value from 0 to 255 • **green** - green component as an integer value from 0 to 255 • **blue** - blue component as an integer value from 0 to 255
Examples	• `color_rgb(255,127,0)` → '255,127,0'

color_rgba

Returns a string representation of a color based on its red, green, blue, and alpha (transparency) components.

Syntax	color_rgba(red, green, blue, alpha)
Arguments	• **red** - red component as an integer value from 0 to 255 • **green** - green component as an integer value from 0 to 255 • **blue** - blue component as an integer value from 0 to 255 • **alpha** - alpha component as an integer value from 0 (completely transparent) to 255 (opaque).
Examples	• `color_rgba(255,127,0,200)` → '255,127,0,200'

color_rgbf

Returns a color object based on its red, green, blue and alpha components.

Syntax	color_rgbf(red, green, blue, [alpha=1.0]) [] marks optional arguments
Arguments	• **red** - red component as a float value from 0.0 to 1.0 • **green** - green component as a float value from 0.0 to 1.0 • **blue** - blue component as a float value from 0.0 to 1.0 • **alpha** - alpha component as a float value from 0.0 to 1.0
Examples	• `color_rgbf(1.0,0.5,0)` → RGBA: 1.00,0.50,0.00,1.00

create_ramp

Returns a gradient ramp from a map of color strings and steps.

Syntax	create_ramp(map, [discrete=false]) [] marks optional arguments
Arguments	• **map** - a map of color strings and steps • **discrete** - set this parameter to true to create a discrete color ramp
Examples	• `ramp_color(create_ramp(map(0,'0,0,0',1,'255,0,0')),1)` → '255,0,0,255'

darker

Returns a darker (or lighter) color. Returned type is the same as color arguments, i.e. a color string representation or a color object.

Syntax	darker(color, factor)
Arguments	• **color** - a color string or a color object • **factor** - an integer corresponding to the darkening factor: – if the factor is greater than 100, this function returns a darker color (e.g., setting factor to 200 returns a color that is half the brightness); – if the factor is less than 100, the return color is lighter, but using the lighter() function for this purpose is recommended; – if the factor is 0 or negative, the return value is unspecified.
Examples	• `darker('200,10,30', 200)` → '100,5,15,255'

Further reading: *lighter*

lighter

Returns a lighter (or darker) color. Returned type is the same as color arguments, i.e. a color string representation or a color object.

Syntax	lighter(color, factor)
Arguments	• **color** - a color string or a color object • **factor** - an integer corresponding to the lightening factor: – if the factor is greater than 100, this function returns a lighter color (e.g., setting factor to 150 returns a color that is 50% brighter); – if the factor is less than 100, the return color is darker, but using the darker() function for this purpose is recommended; – if the factor is 0 or negative, the return value is unspecified.
Examples	• `lighter('200,10,30', 200)` → '255,158,168,255'

Further reading: *darker*

project_color

Returns a color from the project's color scheme.

Syntax	project_color(name)
Arguments	• **name** - a color name
Examples	• `project_color('Logo color')` → '20,140,50'

Further reading: *setting project colors*

project_color_object

Returns a color from the project's color scheme. Contrary to project_color which returns a color string representation, project_color_object returns a color object.

Syntax	project_color_object(name)
Arguments	• **name** - a color name
Examples	• `project_color_object('Logo color')` → RGBA: 0.08,0.55,0.20,1.00

ramp_color

Returns a string representing a color from a color ramp.

Saved ramp variant

Returns a string representing a color from a saved ramp

Syntax	ramp_color(ramp_name, value)
Arguments	• **ramp_name** - the name of the color ramp as a string, for example 'Spectral' • **value** - the position on the ramp to select the color from as a real number between 0 and 1
Examples	• `ramp_color('Spectral',0.3)` → '253,190,115,255'

Note

The color ramps available vary between QGIS installations. This function may not give the expected results if you move your QGIS project between installations.

Expression-created ramp variant

Returns a string representing a color from an expression-created ramp

Syntax	ramp_color(ramp, value)
Arguments	• **ramp** - the color ramp • **value** - the position on the ramp to select the color from as a real number between 0 and 1
Examples	• `ramp_color(create_ramp(map(0,'0,0,0',1,'255,0,0')),1)` → '255,0,0,255'

Further reading: *Setting a Color Ramp*, *The color ramp drop-down shortcut*

ramp_color_object

Returns a color object from a color ramp. Contrary to ramp_color which returns a color string representation, ramp_color_object returns a color object.

Saved ramp variant

Returns a color object from a saved ramp

Syntax	ramp_color_object(ramp_name, value)
Arguments	• **ramp_name** - the name of the color ramp as a string, for example 'Spectral' • **value** - the position on the ramp to select the color from as a real number between 0 and 1
Examples	• `ramp_color_object('Spectral',0.3)` → RGBA: 0.99,0.75,0.45,1.00

> **Note**
>
> The color ramps available vary between QGIS installations. This function may not give the expected results if you move your QGIS project between installations.

Expression-created ramp variant

Returns a color object from an expression-created ramp

Syntax	ramp_color_object(ramp, value)
Arguments	• **ramp** - the color ramp • **value** - the position on the ramp to select the color from as a real number between 0 and 1
Examples	• `ramp_color_object(create_ramp(map(0,color_rgbf(0,0,0),1, color_rgbf(1,0,0))),1)` → RGBA: 1.00,0.00,0.00,1.00

set_color_part

Sets a specific color component for a color string or a color object, e.g., the red component or alpha component.

Syntax	set_color_part(color, component, value)
Arguments	• **color** - a color string or a color object • **component** - a string corresponding to the color component to set. Valid options are: – red: RGB red component (0-255) – green: RGB green component (0-255) – blue: RGB blue component (0-255) – alpha: alpha (transparency) value (0-255) – hue: HSV hue (0-360) – saturation: HSV saturation (0-100) – value: HSV value (0-100) – hsl_hue: HSL hue (0-360) – hsl_saturation: HSL saturation (0-100) – lightness: HSL lightness (0-100) – cyan: CMYK cyan component (0-100) – magenta: CMYK magenta component (0-100) – yellow: CMYK yellow component (0-100) – black: CMYK black component (0-100) • **value** - new value for color component, respecting the ranges listed above
Examples	• `set_color_part('200,10,30','green',50)` → '200,50,30,255' • `set_color_part(color_cmykf(0.21,0,0.92,0.70),'black',100)` → CMYKA: 0.21,0.00,0.92,1.00,1.00

9.2.4 Conditional Functions

This group contains functions to handle conditional checks in expressions.

CASE

CASE is used to evaluate a series of conditions and return a result for the first condition met. The conditions are evaluated sequentially, and if a condition is true, the evaluation stops, and the corresponding result is returned. If none of the conditions are true, the value in the ELSE clause is returned. Furthermore, if no ELSE clause is set and none of the conditions are met, NULL is returned.

CASE

WHEN *condition* THEN *result*

[…n]

[ELSE *result*]

END

[ ] marks optional components

Arguments	• **WHEN condition** - A condition expression to evaluate • **THEN result** - If *condition* evaluates to True then *result* is evaluated and returned. • **ELSE result** - If none of the above conditions evaluated to True then *result* is evaluated and returned.
Examples	• `CASE WHEN "name" IS NULL THEN 'None' END` → Returns the string 'None' if the "name" field is NULL • `CASE WHEN $area > 10000 THEN 'Big property' WHEN $area > 5000 THEN 'Medium property' ELSE 'Small property' END` → Returns the string 'Big property' if the area is bigger than 10000, 'Medium property' if the area is between 5000 and 10000, and 'Small property' for others

coalesce

Returns the first non-NULL value from the expression list.

This function can take any number of arguments.

Syntax	coalesce(expression1, expression2, …)
Arguments	• **expression** - any valid expression or value, regardless of type.
Examples	• `coalesce(NULL, 2)` → 2 • `coalesce(NULL, 2, 3)` → 2 • `coalesce(7, NULL, 3*2)` → 7 • `coalesce("fieldA", "fallbackField", 'ERROR')` → value of fieldA if it is non-NULL else the value of "fallbackField" or the string 'ERROR' if both are NULL

if

Tests a condition and returns a different result depending on the conditional check.

Syntax	if(condition, result_when_true, result_when_false)
Arguments	• **condition** - the condition which should be checked • **result_when_true** - the result which will be returned when the condition is true or another value that does not convert to false. • **result_when_false** - the result which will be returned when the condition is false or another value that converts to false like 0 or ''. NULL will also be converted to false.
Examples	• `if( 1+1=2, 'Yes', 'No' )` → 'Yes' • `if( 1+1=3, 'Yes', 'No' )` → 'No' • `if( 5 > 3, 1, 0)` → 1 • `if( '', 'It is true (not empty)', 'It is false (empty)' )` → 'It is false (empty)' • `if( ' ', 'It is true (not empty)', 'It is false (empty)' )` → 'It is true (not empty)' • `if( 0, 'One', 'Zero' )` → 'Zero' • `if( 10, 'One', 'Zero' )` → 'One'

nullif

Returns a NULL value if value1 equals value2; otherwise it returns value1. This can be used to conditionally substitute values with NULL.

Syntax	nullif(value1, value2)
Arguments	• **value1** - The value that should either be used or substituted with NULL. • **value2** - The control value that will trigger the NULL substitution.
Examples	• `nullif('(none)', '(none)')` → NULL • `nullif('text', '(none)')` → 'text' • `nullif("name", '')` → NULL, if name is an empty string (or already NULL), the name in any other case.

regexp_match

Return the first matching position matching a regular expression within an unicode string, or 0 if the substring is not found.

Syntax	regexp_match(input_string, regex)
Arguments	• **input_string** - the string to test against the regular expression • **regex** - The regular expression to test against. Backslash characters must be double escaped (e.g., "\\s" to match a white space character or "\\b" to match a word boundary).
Examples	• `regexp_match('QGIS ROCKS','\\sROCKS')` → 5 • `regexp_match('Budač','udač\\b')` → 2

try

Tries an expression and returns its value if error-free. If the expression returns an error, an alternative value will be returned when provided otherwise the function will return NULL.

Syntax	try(expression, [alternative]) [] marks optional arguments
Arguments	• **expression** - the expression which should be run • **alternative** - the result which will be returned if the expression returns an error.
Examples	• `try( to_int( '1' ), 0 )` → 1 • `try( to_int( 'a' ), 0 )` → 0 • `try( to_date( 'invalid_date' ) )` → NULL

9.2.5 Conversions Functions

This group contains functions to convert one data type to another (e.g., string from/to integer, binary from/to string, string to date, …).

convert_timezone

Converts a datetime object to a different timezone.

Syntax	convert_timezone(datetime, timezone)
Arguments	• **datetime** - datetime value • **timezone** - target timezone
Examples	• `convert_timezone(to_datetime('2012-05-04 12:50:00+3'), timezone_from_id('UTC+10'))` → datetime 2012-05-04 19:50:00 (UTC+10) • `convert_timezone("DATE_FIELD", timezone_from_id('Australia/Darwin'))` → Datetime from DATE_FIELD, converted to the 'Australia/Darwin' timezone

from_base64

Decodes a string in the Base64 encoding into a binary value.

Syntax	from_base64(string)
Arguments	• **string** - the string to decode
Examples	• `from_base64('UUdJUw==')` → 'QGIS'

get_timezone

Returns the timezone object associated with a datetime value.

Syntax	get_timezone(datetime)
Arguments	• **datetime** - datetime with timezone
Examples	• `timezone_id(get_timezone(to_datetime('2012-05-04 12:50:00+3')))` → 'UTC+03' • `timezone_id(get_timezone("DATE_FIELD"))` → ID of timezone associated with the DATE_FIELD value

hash

Creates a hash from a string with a given method. One byte (8 bits) is represented with two hex "digits", so 'md4' (16 bytes) produces a 16 * 2 = 32 character long hex string and 'keccak_512' (64 bytes) produces a 64 * 2 = 128 character long hex string.

Syntax	hash(string, method)
Arguments	• **string** - the string to hash • **method** - The hash method among 'md4', 'md5', 'sha1', 'sha224', 'sha384', 'sha512', 'sha3_224', 'sha3_256', 'sha3_384', 'sha3_512', 'keccak_224', 'keccak_256', 'keccak_384', 'keccak_512'
Examples	• `hash('QGIS', 'md4')` → 'c0fc71c241cdebb6e888cbac0e2b68eb' • `hash('QGIS', 'md5')` → '57470aaa9e22adaefac7f5f342f1c6da' • `hash('QGIS', 'sha1')` → 'f87cfb2b74cdd5867db913237024e7001e62b114' • `hash('QGIS', 'sha224')` → '4093a619ada631c770f44bc643ead18fb393b93d6a6af1861fcfece0' • `hash('QGIS', 'sha256')` → 'eb045cba7a797aaa06ac58830846e40c8e8c780bc0676d3393605fae50c • `hash('QGIS', 'sha384')` → '91c1de038cc3d09fdd512e99f9dd9922efadc39ed21d3922e69a4305cc2 • `hash('QGIS', 'sha512')` → 'c2c092f2ab743bf8edbeb6d028a745f30fc720408465ed369421f0a4e20 • `hash('QGIS', 'sha3_224')` → '467f49a5039e7280d5d42fd433e80d203439e338eaabd701f0d6c17d • `hash('QGIS', 'sha3_256')` → '540f7354b6b8a6e735f2845250f15f4f3ba4f666c55574d9e9354575 • `hash('QGIS', 'sha3_384')` → '96052da1e77679e9a65f60d7ead961b28797782314478638 6eb4364 • `hash('QGIS', 'sha3_512')` → '900d079dc69761da113980253aa8ac0414a8bd6d09879a916228f874 • `hash('QGIS', 'keccak_224')` → '5b0ce6acef8b0a121d4ac4f3eaa8503c799ad4e26a3392d1fb2014 • `hash('QGIS', 'keccak_256')` → '991c520aa6815392de24087f61b2ae0fd56abbfeee4a8ca019c101 • `hash('QGIS', 'keccak_384')` → 'c57a3aed9d856fa04e5eeee9b62b6e027cca81ba574116d3cc1f0d4

md5

Creates a md5 hash from a string.

Syntax	md5(string)
Arguments	• **string** - the string to hash
Examples	• `md5('QGIS')` → '57470aaa9e22adaefac7f5f342f1c6da'

set_timezone

Sets the timezone object associated with a datetime value, without changing the date or time components. This function can be used to replace the timezone for a datetime.

Syntax	set_timezone(datetime, timezone)
Arguments	• **datetime** - datetime value • **timezone** - new timezone for datetime
Examples	• `set_timezone(to_datetime('2012-05-04 12:50:00+3'), time-zone_from_id('UTC+10'))` → datetime 2012-05-04 12:50:00 (UTC+10) • `set_timezone(make_datetime(2020,1,1,10,0,0), time-zone_from_id('Australia/Darwin'))` → Datetime of 2020-01-01 10:00:00 with associated timezone 'Australia/Darwin'

sha256

Creates a sha256 hash from a string.

Syntax	sha256(string)
Arguments	• **string** - the string to hash
Examples	• `sha256('QGIS')` → 'eb045cba7a797aaa06ac58830846e40c8e8c780bc0676d3393605fae50c05309'

timezone_from_id

Creates a timezone object from a string ID (from the IANA timezone database). The ID must be one of the available system IDs or a valid UTC-with-offset ID.

Syntax	timezone_from_id(id)
Arguments	• **id** - string containing the time zone ID
Examples	• `timezone_from_id('Australia/Brisbane')` → AEST timezone object • `timezone_from_id('UTC+10:30')` → UTC+10:30 timezone object • `timezone_from_id('UTC-3')` → UTC-03 timezone object

timezone_id

Returns the ID string for a timezone object, using IDs from the IANA timezone database.

Syntax	timezone_id(timezone)
Arguments	• **timezone** - a valid timezone object
Examples	• `timezone_id(timezone_from_id('Australia/Brisbane'))` → 'Australia/Brisbane'

to_base64

Encodes a binary value into a string, using the Base64 encoding.

Syntax	to_base64(value)
Arguments	• **value** - the binary value to encode
Examples	• `to_base64('QGIS')` → 'UUdJUw=='

to_bool

Converts a given value to a boolean. The function will return false if the value is NULL, an empty string, an empty list, or 0.

Syntax	to_bool(value)
Arguments	• **value** - value to convert to boolean
Examples	• `to_bool('')` → false • `to_bool('123')` → true • `to_bool('false')` → true • `to_bool(0)` → false • `to_bool(1)` → true • `to_bool(null)` → false

to_date

Converts a string into a date object. An optional format string can be provided to parse the string; see QDate::fromString or the documentation of the format_date function for additional documentation on the format. By default the current QGIS user locale is used.

Syntax	to_date(string, [format], [language]) [] marks optional arguments
Arguments	• **string** - string representing a date value • **format** - format used to convert the string into a date • **language** - language (lowercase, two- or three-letter, ISO 639 language code) used to convert the string into a date. By default the current QGIS user locale is used.
Examples	• `to_date('2012-05-04')` → 2012-05-04 • `to_date('June 29, 2019','MMMM d, yyyy')` → 2019-06-29, if the current locale uses the name 'June' for the sixth month, otherwise an error occurs • `to_date('29 juin, 2019','d MMMM, yyyy','fr')` → 2019-06-29

Further reading: *format_date*

to_datetime

Converts a string into a datetime object. An optional format string can be provided to parse the string; see QDate::fromString, QTime::fromString or the documentation of the format_date function for additional documentation on the format. By default the current QGIS user locale is used.

Syntax	to_datetime(string, [format], [language]) [] marks optional arguments
Arguments	• **string** - string representing a datetime value • **format** - format used to convert the string into a datetime • **language** - language (lowercase, two- or three-letter, ISO 639 language code) used to convert the string into a datetime. By default the current QGIS user locale is used.
Examples	• `to_datetime('2012-05-04 12:50:00')` → 2012-05-04T12:50:00 • `to_datetime('June 29, 2019 @ 12:34','MMMM d, yyyy @ HH:mm')` → 2019-06-29T12:34, if the current locale uses the name 'June' for the sixth month, otherwise an error occurs • `to_datetime('29 juin, 2019 @ 12:34','d MMMM, yyyy @ HH:mm','fr')` → 2019-06-29T12:34

Further reading: *format_date*

to_decimal

Converts a degree, minute, second coordinate to its decimal equivalent.

Syntax	to_decimal(value)
Arguments	• **value** - A degree, minute, second string.
Examples	• `to_decimal('6°21\'16.445')` → 6.3545680555

to_dm

Converts a coordinate to degree, minute.

Syntax	to_dm(coordinate, axis, precision, [formatting=]) [] marks optional arguments
Arguments	• **coordinate** - A latitude or longitude value. • **axis** - The axis of the coordinate. Either 'x' or 'y'. • **precision** - Number of decimals. • **formatting** - Designates the formatting type. Acceptable values are NULL (default), 'aligned' or 'suffix'.
Examples	• `to_dm(6.1545681, 'x', 3)` → 6°9.274′ • `to_dm(6.1545681, 'y', 4, 'aligned')` → 6°09.2741′N • `to_dm(6.1545681, 'y', 4, 'suffix')` → 6°9.2741′N

to_dms

Converts a coordinate to degree, minute, second.

Syntax	to_dms(coordinate, axis, precision, [formatting=]) [] marks optional arguments
Arguments	• **coordinate** - A latitude or longitude value. • **axis** - The axis of the coordinate. Either 'x' or 'y'. • **precision** - Number of decimals. • **formatting** - Designates the formatting type. Acceptable values are NULL (default), 'aligned' or 'suffix'.
Examples	• `to_dms(6.1545681, 'x', 3)` → 6°9′16.445″ • `to_dms(6.1545681, 'y', 4, 'aligned')` → 6°09′16.4452″N • `to_dms(6.1545681, 'y', 4, 'suffix')` → 6°9′16.4452″N

to_int

Converts a string to integer number. If a value cannot be converted to integer the expression is invalid (e.g '123asd' is invalid).

Syntax	to_int(string)
Arguments	• **string** - string to convert to integer number
Examples	• `to_int('123')` → 123

to_interval

Converts a string to an interval type. Can be used to take days, hours, month, etc of a date.

Syntax	to_interval(string)
Arguments	• **string** - a string representing an interval. Allowable formats include {n} days {n} hours {n} months.
Examples	• `to_interval('1 day 2 hours')` → interval: 1.08333 days • `to_interval( '0.5 hours' )` → interval: 30 minutes • `to_datetime('2012-05-05 12:00:00') - to_interval('1 day 2 hours')` → 2012-05-04T10:00:00

to_real

Converts a string to a real number. If a value cannot be converted to real the expression is invalid (e.g '123.56asd' is invalid). Numbers are rounded after saving changes if the precision is smaller than the result of the conversion.

Syntax	to_real(string)
Arguments	• **string** - string to convert to real number
Examples	• `to_real('123.45')` → 123.45

to_string

Converts a number to string. The conversion is not locale-aware, see 'format_number' for a locale-aware alternative.

Syntax	to_string(number)
Arguments	• **number** - Integer or real value. The number to convert to string.
Examples	• `to_string(1.23)` → '1.23'

Further reading: *format_number*

to_time

Converts a string into a time object. An optional format string can be provided to parse the string; see QTime::fromString for additional documentation on the format.

Syntax	to_time(string, [format], [language]) [] marks optional arguments
Arguments	• **string** - string representing a time value • **format** - format used to convert the string into a time • **language** - language (lowercase, two- or three-letter, ISO 639 language code) used to convert the string into a time
Examples	• `to_time('12:30:01')` → 12:30:01 • `to_time('12:34','HH:mm')` → 12:34:00 • `to_time('12:34','HH:mm','fr')` → 12:34:00

Further reading: *format_date*

9.2.6 CRS Functions

This group contains functions to operate on coordinate reference system objects.

crs_from_text

Creates a coordinate reference system from a string definition. The string can represent an authority ID, a WKT definition, or a PROJ string definition of the CRS.

Syntax	crs_from_text(definition)
Arguments	• **definition** - CRS definition
Examples	• `crs_from_text('EPSG:3857')` → crs value 'EPSG:3857 - WGS 84 / Pseudo-Mercator' • `crs_from_text('PROJ:+proj=merc +a=6378137 +b=6378137 +lat_ts=0 +lon_0=0 +x_0=0 +y_0=0 +k=1 +units=m +nadgrids=@null +wktext +no_defs +type=crs')` → crs value 'EPSG:3857 - WGS 84 / Pseudo-Mercator'

crs_to_authid

Returns the authority ID string for a coordinate reference system.

Syntax	crs_to_authid(crs)
Arguments	• **crs** - crs value
Examples	• `crs_to_authid(crs_from_text('PROJ:+proj=merc +a=6378137 +b=6378137 +lat_ts=0 +lon_0=0 +x_0=0 +y_0=0 +k=1 +units=m +nadgrids=@null +wktext +no_defs +type=crs'))` → 'EPSG:3857'

9.2.7 Custom Functions

This group contains functions created by the user. See *Function Editor* for more details.

9.2.8 Date and Time Functions

This group contains functions for handling date and time data. This group shares several functions with the *Conversions Functions* (to_date, to_time, to_datetime, to_interval) and *String Functions* (format_date) groups.

> **Note**
>
> **Storing date, datetime and intervals on fields**
>
> The ability to store *date*, *time* and *datetime* values directly on fields depends on the data source's provider (e.g., Shapefile accepts *date* format, but not *datetime* or *time* format). The following are some suggestions to overcome this limitation:
>
> - *date*, *datetime* and *time* can be converted and stored in text type fields using the *format_date()* function.
> - *Intervals* can be stored in integer or decimal type fields after using one of the date extraction functions (e.g., *day()* to get the interval expressed in days)

age

Returns the difference between two dates or datetimes.

The difference is returned as an `Interval` and needs to be used with one of the following functions in order to extract useful information:

- `year`
- `month`
- `week`
- `day`
- `hour`
- `minute`
- `second`

Syntax	age(datetime1, datetime2)
Arguments	• **datetime1** - a string, date or datetime representing the later date • **datetime2** - a string, date or datetime representing the earlier date
Examples	• `day(age('2012-05-12','2012-05-02'))` → 10 • `hour(age('2012-05-12','2012-05-02'))` → 240

convert_timezone

Converts a datetime object to a different timezone.

Syntax	convert_timezone(datetime, timezone)
Arguments	• **datetime** - datetime value • **timezone** - target timezone
Examples	• `convert_timezone(to_datetime('2012-05-04 12:50:00+3'), timezone_from_id('UTC+10'))` → datetime 2012-05-04 19:50:00 (UTC+10) • `convert_timezone("DATE_FIELD", timezone_from_id('Australia/Darwin'))` → Datetime from DATE_FIELD, converted to the 'Australia/Darwin' timezone

datetime_from_epoch

Returns a datetime whose date and time are the number of milliseconds, msecs, that have passed since 1970-01-01T00:00:00.000, Coordinated Universal Time (Qt.UTC), and converted to Qt.LocalTime.

Syntax	datetime_from_epoch(int)
Arguments	• **int** - number (milliseconds)
Examples	• `datetime_from_epoch(1483225200000)` → 2017-01-01T00:00:00

day

Extracts the day from a date, or the number of days from an interval.

Date variant

Extracts the day from a date or datetime.

Syntax	day(date)
Arguments	• **date** - a date or datetime value
Examples	• `day('2012-05-12')` → 12

Interval variant

Calculates the length in days of an interval.

Syntax	day(interval)
Arguments	• **interval** - interval value to return number of days from
Examples	• `day(to_interval('3 days'))` → 3 • `day(to_interval('3 weeks 2 days'))` → 23 • `day(age('2012-01-01','2010-01-01'))` → 730

day_of_week

Returns the day of the week for a specified date or datetime. The returned value ranges from 0 to 6, where 0 corresponds to a Sunday and 6 to a Saturday.

Syntax	day_of_week(date)
Arguments	• **date** - date or datetime value
Examples	• `day_of_week(to_date('2015-09-21'))` → 1

epoch

Returns the interval in milliseconds between the unix epoch and a given date value.

Syntax	epoch(date)
Arguments	• **date** - a date or datetime value
Examples	• `epoch(to_date('2017-01-01'))` → 1483203600000

format_date

Formats a date type or string into a custom string format. Uses Qt date/time format strings. See QDateTime::toString.

Syntax	format_date(datetime, format, [language]) [] marks optional arguments
Arguments	• **datetime** - date, time or datetime value • **format** - String template used to format the string.

Expression	Output
d	the day as number without a leading zero (1 to 31)
dd	the day as number with a leading zero (01 to 31)
ddd	the abbreviated localized day name (e.g. 'Mon' to 'Sun')
dddd	the long localized day name (e.g. 'Monday' to 'Sunday')
M	the month as number without a leading zero (1-12)
MM	the month as number with a leading zero (01-12)
MMM	the abbreviated localized month name (e.g. 'Jan' to 'Dec')
MMMM	the long localized month name (e.g. 'January' to 'December')
yy	the year as two digit number (00-99)
yyyy	the year as four digit number

These expressions may be used for the time part of the format string:

Expression	Output
h	the hour without a leading zero (0 to 23 or 1 to 12 if AM/PM display)
hh	the hour with a leading zero (00 to 23 or 01 to 12 if AM/PM display)
H	the hour without a leading zero (0 to 23, even with AM/PM display)
HH	the hour with a leading zero (00 to 23, even with AM/PM display)
m	the minute without a leading zero (0 to 59)
mm	the minute with a leading zero (00 to 59)
s	the second without a leading zero (0 to 59)
ss	the second with a leading zero (00 to 59)
z	the milliseconds without trailing zeroes (0 to 999)
zzz	the milliseconds with trailing zeroes (000 to 999)
AP or A	interpret as an AM/PM time. *AP* must be either 'AM' or 'PM'.
ap or a	Interpret as an AM/PM time. *ap* must be either 'am' or 'pm'.

- **language** - language (lowercase, two- or three-letter, ISO 639 language code) used to format the date into a custom string. By default the current QGIS user locale is used.

Examples

- `format_date('2012-05-15','dd.MM.yyyy')` → '15.05.2012'
- `format_date('2012-05-15','d MMMM yyyy','fr')` → '15 mai 2012'
- `format_date('2012-05-15','dddd')` → 'Tuesday', if the current locale is an English variant
- `format_date('2012-05-15 13:54:20','dd.MM.yy')` → '15.05.12'
- `format_date('13:54:20','hh:mm AP')` → '01:54 PM'

get_timezone

Returns the timezone object associated with a datetime value.

Syntax	get_timezone(datetime)
Arguments	• **datetime** - datetime with timezone
Examples	• `timezone_id(get_timezone(to_datetime('2012-05-04 12:50:00+3')))` → 'UTC+03' • `timezone_id(get_timezone("DATE_FIELD"))` → ID of timezone associated with the DATE_FIELD value

hour

Extracts the hour part from a datetime or time, or the number of hours from an interval.

Time variant

Extracts the hour part from a time or datetime.

Syntax	hour(datetime)
Arguments	• **datetime** - a time or datetime value
Examples	• `hour( to_datetime('2012-07-22 13:24:57') )` → 13

Interval variant

Calculates the length in hours of an interval.

Syntax	hour(interval)
Arguments	• **interval** - interval value to return number of hours from
Examples	• `hour(to_interval('3 hours'))` → 3 • `hour(age('2012-07-22T13:00:00','2012-07-22T10:00:00'))` → 3 • `hour(age('2012-01-01','2010-01-01'))` → 17520

make_date

Creates a date value from year, month and day numbers.

Syntax	make_date(year, month, day)
Arguments	• **year** - Year number. Years 1 to 99 are interpreted as is. Year 0 is invalid. • **month** - Month number, where 1=January • **day** - Day number, beginning with 1 for the first day in the month
Examples	• `make_date(2020,5,4)` → date value 2020-05-04

make_datetime

Creates a datetime value from year, month, day, hour, minute and second numbers.

Syntax	make_datetime(year, month, day, hour, minute, second)
Arguments	• **year** - Year number. Years 1 to 99 are interpreted as is. Year 0 is invalid. • **month** - Month number, where 1=January • **day** - Day number, beginning with 1 for the first day in the month • **hour** - Hour number • **minute** - Minutes • **second** - Seconds (fractional values include milliseconds)
Examples	• `make_datetime(2020,5,4,13,45,30.5)` → datetime value 2020-05-04 13:45:30.500

make_interval

Creates an interval value from year, month, weeks, days, hours, minute and seconds values.

Syntax	make_interval([years=0], [months=0], [weeks=0], [days=0], [hours=0], [minutes=0], [seconds=0]) [] marks optional arguments
Arguments	• **years** - Number of years (assumes a 365.25 day year length). • **months** - Number of months (assumes a 30 day month length) • **weeks** - Number of weeks • **days** - Number of days • **hours** - Number of hours • **minutes** - Number of minutes • **seconds** - Number of seconds
Examples	• `make_interval(hours:=3)` → interval: 3 hours • `make_interval(days:=2, hours:=3)` → interval: 2.125 days • `make_interval(minutes:=0.5, seconds:=5)` → interval: 35 seconds

make_time

Creates a time value from hour, minute and second numbers.

Syntax	make_time(hour, minute, second)
Arguments	• **hour** - Hour number • **minute** - Minutes • **second** - Seconds (fractional values include milliseconds)
Examples	• `make_time(13,45,30.5)` → time value 13:45:30.500

minute

Extracts the minutes part from a datetime or time, or the number of minutes from an interval.

Time variant

Extracts the minutes part from a time or datetime.

Syntax	minute(datetime)
Arguments	• **datetime** - a time or datetime value
Examples	• `minute( to_datetime('2012-07-22 13:24:57') )` → 24

Interval variant

Calculates the length in minutes of an interval.

Syntax	minute(interval)
Arguments	• **interval** - interval value to return number of minutes from
Examples	• `minute(to_interval('3 minutes'))` → 3 • `minute(age('2012-07-22T00:20:00','2012-07-22T00:00:00'))` → 20 • `minute(age('2012-01-01','2010-01-01'))` → 1051200

month

Extracts the month part from a date, or the number of months from an interval.

Date variant

Extracts the month part from a date or datetime.

Syntax	month(date)
Arguments	• **date** - a date or datetime value
Examples	• `month('2012-05-12')` → 05

Interval variant

Calculates the length in months of an interval.

Syntax	month(interval)
Arguments	• **interval** - interval value to return number of months from
Examples	• `month(to_interval('3 months'))` → 3 • `month(age('2012-01-01','2010-01-01'))` → 4.03333

now

Returns the current date and time. The function is static and will return consistent results while evaluating. The time returned is the time when the expression is prepared.

Syntax	now()
Examples	• `now()` → 2012-07-22T13:24:57

second

Extracts the seconds part from a datetime or time, or the number of seconds from an interval.

Time variant

Extracts the seconds part from a time or datetime.

Syntax	second(datetime)
Arguments	• **datetime** - a time or datetime value
Examples	• `second( to_datetime('2012-07-22 13:24:57') )` → 57

Interval variant

Calculates the length in seconds of an interval.

Syntax	second(interval)
Arguments	• **interval** - interval value to return number of seconds from
Examples	• `second(to_interval('3 minutes'))` → 180 • `second(age('2012-07-22T00:20:00','2012-07-22T00:00:00'))` → 1200 • `second(age('2012-01-01','2010-01-01'))` → 63072000

set_timezone

Sets the timezone object associated with a datetime value, without changing the date or time components. This function can be used to replace the timezone for a datetime.

Syntax	set_timezone(datetime, timezone)
Arguments	• **datetime** - datetime value • **timezone** - new timezone for datetime
Examples	• `set_timezone(to_datetime('2012-05-04 12:50:00+3'), time-zone_from_id('UTC+10'))` → datetime 2012-05-04 12:50:00 (UTC+10) • `set_timezone(make_datetime(2020,1,1,10,0,0), time-zone_from_id('Australia/Darwin'))` → Datetime of 2020-01-01 10:00:00 with associated timezone 'Australia/Darwin'

timezone_from_id

Creates a timezone object from a string ID (from the IANA timezone database). The ID must be one of the available system IDs or a valid UTC-with-offset ID.

Syntax	timezone_from_id(id)
Arguments	• **id** - string containing the time zone ID
Examples	• `timezone_from_id('Australia/Brisbane')` → AEST timezone object • `timezone_from_id('UTC+10:30')` → UTC+10:30 timezone object • `timezone_from_id('UTC-3')` → UTC-03 timezone object

timezone_id

Returns the ID string for a timezone object, using IDs from the IANA timezone database.

Syntax	timezone_id(timezone)
Arguments	• **timezone** - a valid timezone object
Examples	• `timezone_id(timezone_from_id('Australia/Brisbane'))` → 'Australia/Brisbane'

to_date

Converts a string into a date object. An optional format string can be provided to parse the string; see QDate::fromString or the documentation of the format_date function for additional documentation on the format. By default the current QGIS user locale is used.

Syntax	to_date(string, [format], [language]) [] marks optional arguments
Arguments	• **string** - string representing a date value • **format** - format used to convert the string into a date • **language** - language (lowercase, two- or three-letter, ISO 639 language code) used to convert the string into a date. By default the current QGIS user locale is used.
Examples	• `to_date('2012-05-04')` → 2012-05-04 • `to_date('June 29, 2019','MMMM d, yyyy')` → 2019-06-29, if the current locale uses the name 'June' for the sixth month, otherwise an error occurs • `to_date('29 juin, 2019','d MMMM, yyyy','fr')` → 2019-06-29

Further reading: *format_date*

to_datetime

Converts a string into a datetime object. An optional format string can be provided to parse the string; see QDate::fromString, QTime::fromString or the documentation of the format_date function for additional documentation on the format. By default the current QGIS user locale is used.

Syntax	to_datetime(string, [format], [language]) [] marks optional arguments
Arguments	• **string** - string representing a datetime value • **format** - format used to convert the string into a datetime • **language** - language (lowercase, two- or three-letter, ISO 639 language code) used to convert the string into a datetime. By default the current QGIS user locale is used.
Examples	• `to_datetime('2012-05-04 12:50:00')` → 2012-05-04T12:50:00 • `to_datetime('June 29, 2019 @ 12:34','MMMM d, yyyy @ HH:mm')` → 2019-06-29T12:34, if the current locale uses the name 'June' for the sixth month, otherwise an error occurs • `to_datetime('29 juin, 2019 @ 12:34','d MMMM, yyyy @ HH:mm', 'fr')` → 2019-06-29T12:34

Further reading: *format_date*

to_interval

Converts a string to an interval type. Can be used to take days, hours, month, etc of a date.

Syntax	to_interval(string)
Arguments	• **string** - a string representing an interval. Allowable formats include {n} days {n} hours {n} months.
Examples	• `to_interval('1 day 2 hours')` → interval: 1.08333 days • `to_interval( '0.5 hours' )` → interval: 30 minutes • `to_datetime('2012-05-05 12:00:00') - to_interval('1 day 2 hours')` → 2012-05-04T10:00:00

to_time

Converts a string into a time object. An optional format string can be provided to parse the string; see QTime::fromString for additional documentation on the format.

Syntax	to_time(string, [format], [language]) [] marks optional arguments
Arguments	• **string** - string representing a time value • **format** - format used to convert the string into a time • **language** - language (lowercase, two- or three-letter, ISO 639 language code) used to convert the string into a time
Examples	• `to_time('12:30:01')` → 12:30:01 • `to_time('12:34','HH:mm')` → 12:34:00 • `to_time('12:34','HH:mm','fr')` → 12:34:00

Further reading: *format_date*

week

Extracts the week number from a date, or the number of weeks from an interval.

Date variant

Extracts the week number from a date or datetime.

Syntax	week(date)
Arguments	• **date** - a date or datetime value
Examples	• `week('2012-05-12')` → 19

Interval variant

Calculates the length in weeks of an interval.

Syntax	week(interval)
Arguments	• **interval** - interval value to return number of months from
Examples	• `week(to_interval('3 weeks'))` → 3 • `week(age('2012-01-01','2010-01-01'))` → 104.285

year

Extracts the year part from a date, or the number of years from an interval.

Date variant

Extracts the year part from a date or datetime.

Syntax	year(date)
Arguments	• **date** - a date or datetime value
Examples	• `year('2012-05-12')` → 2012

Interval variant

Calculates the length in years of an interval.

Syntax	year(interval)
Arguments	• **interval** - interval value to return number of years from
Examples	• `year(to_interval('3 years'))` → 3 • `year(age('2012-01-01','2010-01-01'))` → 1.9986

Some examples:

Besides these functions, subtracting dates, datetimes or times using the − (minus) operator will return an interval.

Adding or subtracting an interval to dates, datetimes or times, using the + (plus) and − (minus) operators, will return a datetime.

- Get the number of days until QGIS 3.0 release:

```
to_date('2017-09-29') - to_date(now())
-- Returns <interval: 203 days>
```

- The same with time:

```
to_datetime('2017-09-29 12:00:00') - now()
-- Returns <interval: 202.49 days>
```

- Get the datetime of 100 days from now:

```
now() + to_interval('100 days')
-- Returns <datetime: 2017-06-18 01:00:00>
```

9.2.9 Fields and Values

Contains a list of fields from the active layer, and special values. Fields list includes the ones stored in the dataset, *virtual* and *auxiliary* ones as well as from *joins*.

Double-click a field name to have it added to your expression. You can also type the field name (preferably inside double quotes) or its *alias*.

To retrieve fields values to use in an expression, select the appropriate field and, in the shown widget, choose between *10 Samples* and *All Unique*. Requested values are then displayed and you can use the *Search* box at the top of the list to filter the result. Sample values can also be accessed via right-clicking on a field.

To add a value to the expression you are writing, double-click on it in the list. If the value is of a string type, it should be simple quoted, otherwise no quote is needed.

NULL

Equates to a NULL value.

Syntax	NULL
Examples	• `NULL` → a NULL value

Note

To test for NULL use an *IS NULL* or *IS NOT NULL* expression.

9.2.10 Files and Paths Functions

This group contains functions which manipulate file and path names.

base_file_name

Returns the base name of the file without the directory or file suffix.

Syntax	base_file_name(path)
Arguments	• **path** - a file path or a map layer value. If a map layer value is specified then the file source of the layer will be used.
Examples	• `base_file_name('/home/qgis/data/country_boundaries.shp')` → 'country_boundaries'

exif

Retrieves exif tag values from an image file.

Syntax	exif(path, [tag]) [] marks optional arguments
Arguments	• **path** - An image file path or a map layer value. If a map layer value is specified then the file source of the layer will be used. • **tag** - The tag to return. If empty, a map with all exif tag values will be returned.
Examples	• `exif('/my/photo.jpg','Exif.Image.Orientation')` → 0

file_exists

Returns TRUE if a file path exists.

Syntax	file_exists(path)
Arguments	• **path** - a file path or a map layer value. If a map layer value is specified then the file source of the layer will be used.
Examples	• `file_exists('/home/qgis/data/country_boundaries.shp')` → TRUE

file_name

Returns the name of a file (including the file extension), excluding the directory.

Syntax	file_name(path)
Arguments	• **path** - a file path or a map layer value. If a map layer value is specified then the file source of the layer will be used.
Examples	• `file_name('/home/qgis/data/country_boundaries.shp')` → 'country_boundaries.shp'

file_path

Returns the directory component of a file path. This does not include the file name.

Syntax	file_path(path)
Arguments	• **path** - a file path or a map layer value. If a map layer value is specified then the file source of the layer will be used.
Examples	• `file_path('/home/qgis/data/country_boundaries.shp')` → '/home/qgis/data'

file_size

Returns the size (in bytes) of a file.

Syntax	file_size(path)
Arguments	• **path** - a file path or a map layer value. If a map layer value is specified then the file source of the layer will be used.
Examples	• `file_size('/home/qgis/data/country_boundaries.geojson')` → 5674

file_suffix

Returns the file suffix (extension) from a file path.

Syntax	file_suffix(path)
Arguments	• **path** - a file path or a map layer value. If a map layer value is specified then the file source of the layer will be used.
Examples	• `file_suffix('/home/qgis/data/country_boundaries.shp')` → 'shp'

is_directory

Returns TRUE if a path corresponds to a directory.

Syntax	is_directory(path)
Arguments	• **path** - a file path or a map layer value. If a map layer value is specified then the file source of the layer will be used.
Examples	• `is_directory('/home/qgis/data/country_boundaries.shp')` → FALSE • `is_directory('/home/qgis/data/')` → TRUE

is_file

Returns TRUE if a path corresponds to a file.

Syntax	is_file(path)
Arguments	• **path** - a file path or a map layer value. If a map layer value is specified then the file source of the layer will be used.
Examples	• `is_file('/home/qgis/data/country_boundaries.shp')` → TRUE • `is_file('/home/qgis/data/')` → FALSE

9.2.11 Form Functions

This group contains functions that operate exclusively under the attribute form context. For example, in *field's widgets* settings.

current_parent_value

Only usable in an embedded form context, this function returns the current, unsaved value of a field in the parent form currently being edited. This will differ from the parent feature's actual attribute values for features which are currently being edited or have not yet been added to a parent layer. When used in a value-relation widget filter expression, this function should be wrapped into a 'coalesce()' that can retrieve the actual parent feature from the layer when the form is not used in an embedded context.

Syntax	current_parent_value(field_name)
Arguments	• **field_name** - a field name in the current parent form
Examples	• `current_parent_value( 'FIELD_NAME' )` → The current value of a field 'FIELD_NAME' in the parent form.

current_value

Returns the current, unsaved value of a field in the form or table row currently being edited. This will differ from the feature's actual attribute values for features which are currently being edited or have not yet been added to a layer.

Syntax	current_value(field_name)
Arguments	• **field_name** - a field name in the current form or table row
Examples	• `current_value( 'FIELD_NAME' )` → The current value of field 'FIELD_NAME'.

9.2.12 Fuzzy Matching Functions

This group contains functions for fuzzy comparisons between values.

hamming_distance

Returns the Hamming distance between two strings. This equates to the number of characters at corresponding positions within the input strings where the characters are different. The input strings must be the same length, and the comparison is case-sensitive.

Syntax	hamming_distance(string1, string2)
Arguments	• **string1** - a string • **string2** - a string
Examples	• `hamming_distance('abc','xec')` → 2 • `hamming_distance('abc','ABc')` → 2 • `hamming_distance(upper('abc'),upper('ABC'))` → 0 • `hamming_distance('abc','abcd')` → NULL

levenshtein

Returns the Levenshtein edit distance between two strings. This equates to the minimum number of character edits (insertions, deletions or substitutions) required to change one string to another.

The Levenshtein distance is a measure of the similarity between two strings. Smaller distances mean the strings are more similar, and larger distances indicate more different strings. The distance is case sensitive.

Syntax	levenshtein(string1, string2)
Arguments	• **string1** - a string • **string2** - a string
Examples	• `levenshtein('kittens','mitten')` → 2 • `levenshtein('Kitten','kitten')` → 1 • `levenshtein(upper('Kitten'),upper('kitten'))` → 0

longest_common_substring

Returns the longest common substring between two strings. This substring is the longest string that is a substring of the two input strings. For example, the longest common substring of "ABABC" and "BABCA" is "BABC". The substring is case sensitive.

Syntax	longest_common_substring(string1, string2)
Arguments	• **string1** - a string • **string2** - a string
Examples	• `longest_common_substring('ABABC','BABCA')` → 'BABC' • `longest_common_substring('abcDeF','abcdef')` → 'abc' • `longest_common_substring(upper('abcDeF'),upper('abcdex'))` → 'ABCDE'

soundex

Returns the Soundex representation of a string. Soundex is a phonetic matching algorithm, so strings with similar sounds should be represented by the same Soundex code.

Syntax	soundex(string)
Arguments	• **string** - a string
Examples	• `soundex('robert')` → 'R163' • `soundex('rupert')` → 'R163' • `soundex('rubin')` → 'R150'

9.2.13 General Functions

This group contains general assorted functions.

env

Gets an environment variable and returns its content as a string. If the variable is not found, NULL will be returned. This is handy to inject system specific configuration like drive letters or path prefixes. Definition of environment variables depends on the operating system, please check with your system administrator or the operating system documentation how this can be set.

Syntax	env(name)
Arguments	• **name** - The name of the environment variable which should be retrieved.
Examples	• `env( 'LANG' )` → 'en_US.UTF-8' • `env( 'MY_OWN_PREFIX_VAR' )` → 'Z:' • `env( 'I_DO_NOT_EXIST' )` → NULL

eval

Evaluates an expression which is passed in a string. Useful to expand dynamic parameters passed as context variables or fields.

Syntax	eval(expression)
Arguments	• **expression** - an expression string
Examples	• `eval('\'nice\'')` → 'nice' • `eval(@expression_var)` → [whatever the result of evaluating @expression_var might be…]

eval_template

Evaluates a template which is passed in a string. Useful to expand dynamic parameters passed as context variables or fields.

Syntax	eval_template(template)
Arguments	• **template** - a template string
Examples	• `eval_template('QGIS [% upper(\'rocks\') %]')` → QGIS ROCKS

is_layer_visible

Returns TRUE if a specified layer is visible.

Syntax	is_layer_visible(layer)
Arguments	• **layer** - a string, representing either a layer name or layer ID
Examples	• `is_layer_visible('baseraster')` → TRUE

mime_type

Returns the mime type of the binary data.

Syntax	mime_type(bytes)
Arguments	• **bytes** - the binary data
Examples	• `mime_type('<html><body></body></html>')` → text/html • `mime_type(from_base64('R0lGODlhAQABAAAAACH5BAEKAAEALAAAAAABAAEAAAIAOw==` → image/gif

var

Returns the value stored within a specified variable.

Syntax	var(name)
Arguments	• **name** - a variable name
Examples	• `var('qgis_version')` → '2.12'

Further reading: List of default *variables*

with_variable

This function sets a variable for any expression code that will be provided as 3rd argument. This is only useful for complicated expressions, where the same calculated value needs to be used in different places.

Syntax	with_variable(name, value, expression)
Arguments	• **name** - the name of the variable to set • **value** - the value to set • **expression** - the expression for which the variable will be available
Examples	• `with_variable('my_sum', 1 + 2 + 3, @my_sum * 2 + @my_sum * 5)` → 42

9.2.14 Geometry Functions

This group contains functions that operate on geometry objects (e.g. buffer, transform, $area).

affine_transform

Returns the geometry after an affine transformation. Calculations are in the Spatial Reference System of this geometry. The operations are performed in a scale, rotation, translation order. If there is a Z or M offset but the coordinate is not present in the geometry, it will be added.

Syntax	affine_transform(geometry, delta_x, delta_y, rotation_z, scale_x, scale_y, [delta_z=0], [delta_m=0], [scale_z=1], [scale_m=1]) [] marks optional arguments
Arguments	• **geometry** - a geometry • **delta_x** - x-axis translation • **delta_y** - y-axis translation • **rotation_z** - rotation around z-axis in degrees counter-clockwise • **scale_x** - x-axis scale factor • **scale_y** - y-axis scale factor • **delta_z** - z-axis translation • **delta_m** - m-axis translation • **scale_z** - z-axis scale factor • **scale_m** - m-axis scale factor
Examples	• `geom_to_wkt(affine_transform(geom_from_wkt('LINESTRING(1 1, 2 2)'), 2, 2, 0, 1, 1))` → 'LineString (3 3, 4 4)' • `geom_to_wkt(affine_transform(geom_from_wkt('POLYGON((0 0, 0 3, 2 2, 0 0))'), 0, 0, -90, 1, 2))` → 'Polygon ((0 0, 6 0, 4 -2, 0 0))' • `geom_to_wkt(affine_transform(geom_from_wkt('POINT(3 1)'), 0, 0, 0, 1, 1, 5, 0))` → 'PointZ (3 1 5)'

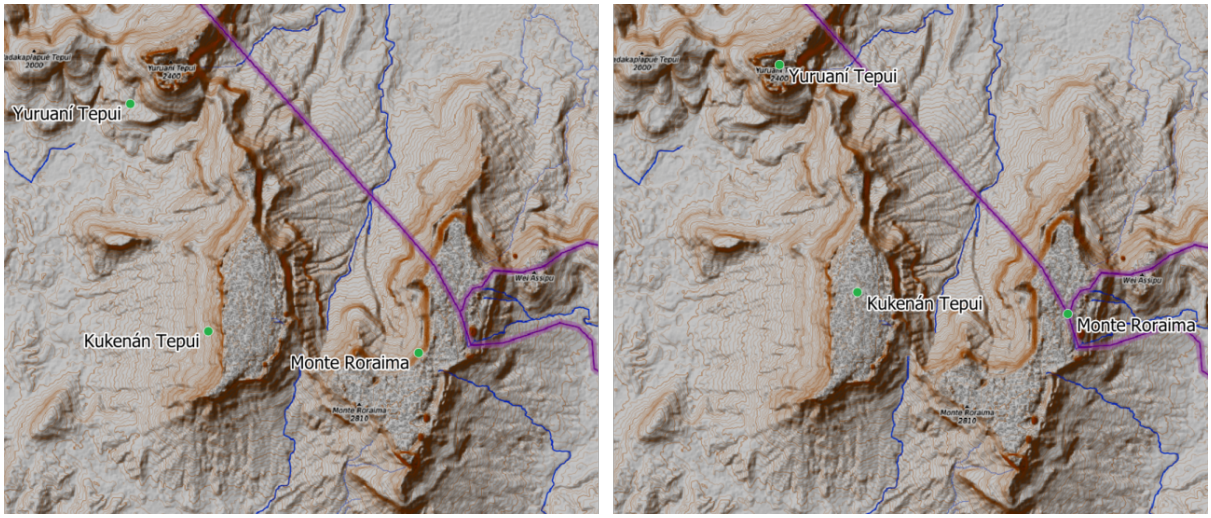


Fig. 9.4: Vector point layer (green dots) before (left), and after (right) an affine transformation (translation).

angle_at_vertex

Returns the bisector angle (average angle) to the geometry for a specified vertex on a linestring geometry. Angles are in degrees clockwise from north.

Syntax	angle_at_vertex(geometry, vertex)
Arguments	• **geometry** - a linestring geometry • **vertex** - vertex index, starting from 0; if the value is negative, the selected vertex index will be its total count minus the absolute value
Examples	• `angle_at_vertex(geometry:=geom_from_wkt('LineString(0 0, 10 0, 10 10)'),vertex:=1)` → 45.0

apply_dash_pattern

Applies a dash pattern to a geometry, returning a MultiLineString geometry which is the input geometry stroked along each line/ring with the specified pattern.

Syntax	apply_dash_pattern(geometry, pattern, [start_rule=no_rule], [end_rule=no_rule], [adjustment=both], [pattern_offset=0]) [] marks optional arguments
Arguments	• **geometry** - a geometry (accepts (multi)linestrings or (multi)polygons). • **pattern** - dash pattern, as an array of numbers representing dash and gap lengths. Must contain an even number of elements. • **start_rule** - optional rule for constraining the start of the pattern. Valid values are 'no_rule', 'full_dash', 'half_dash', 'full_gap', 'half_gap'. • **end_rule** - optional rule for constraining the end of the pattern. Valid values are 'no_rule', 'full_dash', 'half_dash', 'full_gap', 'half_gap'. • **adjustment** - optional rule for specifying which part of patterns are adjusted to fit the desired pattern rules. Valid values are 'both', 'dash', 'gap'. • **pattern_offset** - Optional distance specifying a specific distance along the pattern to commence at.
Examples	• `geom_to_wkt(apply_dash_pattern(geom_from_wkt('LINESTRING(1 1, 10 1)'), array(3, 1)))` → MultiLineString ((1 1, 4 1),(5 1, 8 1),(9 1, 10 1, 10 1)) • `geom_to_wkt(apply_dash_pattern(geom_from_wkt('LINESTRING(1 1, 10 1)'), array(3, 1), start_rule:='half_dash'))` → MultiLineString ((1 1, 2.5 1),(3.5 1, 6.5 1),(7.5 1, 10 1, 10 1))

$area

Returns the area of the current feature. The area calculated by this function respects both the current project's ellipsoid setting and area unit settings. For example, if an ellipsoid has been set for the project then the calculated area will be ellipsoidal, and if no ellipsoid is set then the calculated area will be planimetric.

Syntax	$area
Examples	• `$area` → 42

area

Returns the area of a geometry polygon object. Calculations are always planimetric in the Spatial Reference System (SRS) of this geometry, and the units of the returned area will match the units for the SRS. This differs from the calculations performed by the $area function, which will perform ellipsoidal calculations based on the project's ellipsoid and area unit settings.

Syntax	area(geometry)
Arguments	• **geometry** - polygon geometry object
Examples	• `area(@geometry)` → area of the current feature's geometry • `area(geom_from_wkt('POLYGON((0 0, 4 0, 4 2, 0 2, 0 0))'))` → 8.0

azimuth

Returns the north-based azimuth as the angle in radians measured clockwise from the vertical on point_a to point_b.

Syntax	azimuth(point_a, point_b)
Arguments	• **point_a** - point geometry • **point_b** - point geometry
Examples	• `degrees( azimuth( make_point(25, 45), make_point(75, 100) ) )` → 42.273689 • `degrees( azimuth( make_point(75, 100), make_point(25,45) ) )` → 222.273689

bearing

Returns the north-based bearing as the angle in radians measured clockwise on the ellipsoid from the vertical on point_a to point_b.

Syntax	bearing(point_a, point_b, [source_crs], [ellipsoid]) [] marks optional arguments
Arguments	• **point_a** - point geometry • **point_b** - point geometry • **source_crs** - an optional string or CRS object representing the source CRS of the points. By default the current layer's CRS is used. • **ellipsoid** - an optional string representing the acronym or the authority ID (e.g., 'EPSG:7030') of the ellipsoid on which the bearing should be measured. By default the current project's ellipsoid setting is used.
Examples	• `degrees( bearing( make_point(16198544, -4534850), make_point(18736872, -1877769), 'EPSG:3857', 'EPSG:7030') )` → 49.980071 • `degrees( bearing( make_point(18736872, -1877769), make_point(16198544, -4534850), 'EPSG:3857', 'WGS84') )` → 219.282386

boundary

Returns the closure of the combinatorial boundary of the geometry (ie the topological boundary of the geometry). For instance, a polygon geometry will have a boundary consisting of the linestrings for each ring in the polygon. Some geometry types do not have a defined boundary, e.g., points or geometry collections, and will return NULL.

Syntax	boundary(geometry)
Arguments	• **geometry** - a geometry
Examples	• `boundary(@geometry)` → boundary of the current feature's geometry • `geom_to_wkt(boundary(geom_from_wkt('Polygon((1 1, 0 0, -1 1, 1 1))')))` → 'LineString(1 1,0 0,-1 1,1 1)' • `geom_to_wkt(boundary(geom_from_wkt('LineString(1 1,0 0,-1 1)')))` → 'MultiPoint ((1 1),(-1 1))'

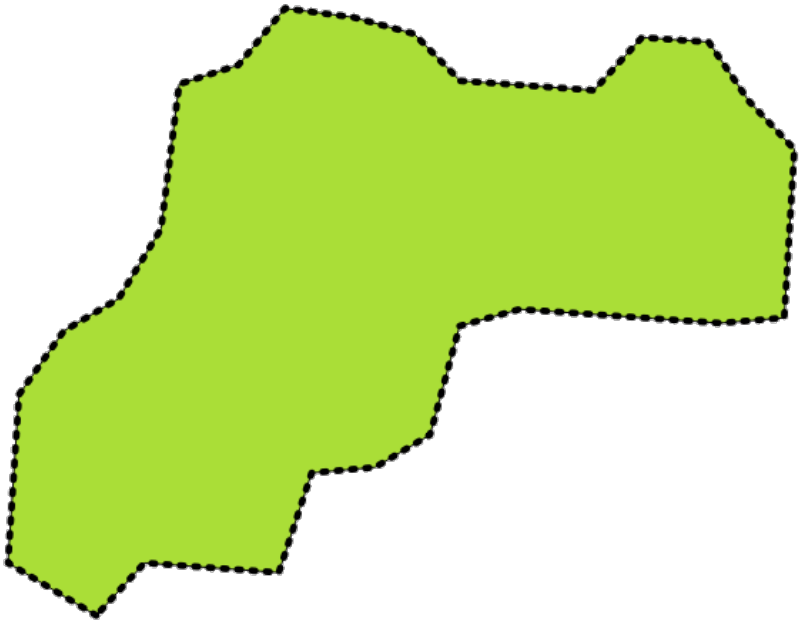


Fig. 9.5: Boundary (black dashed line) of the source polygon layer

Further reading: *Boundary* algorithm

bounds

Returns a geometry which represents the bounding box of an input geometry. Calculations are in the Spatial Reference System of this geometry.

Syntax	bounds(geometry)
Arguments	• **geometry** - a geometry
Examples	• `bounds(@geometry)` → bounding box of the current feature's geometry • `geom_to_wkt(bounds(geom_from_wkt('Polygon((1 1, 0 0, -1 1, 1 1))')))` → 'Polygon ((-1 0, 1 0, 1 1, -1 1, -1 0))'

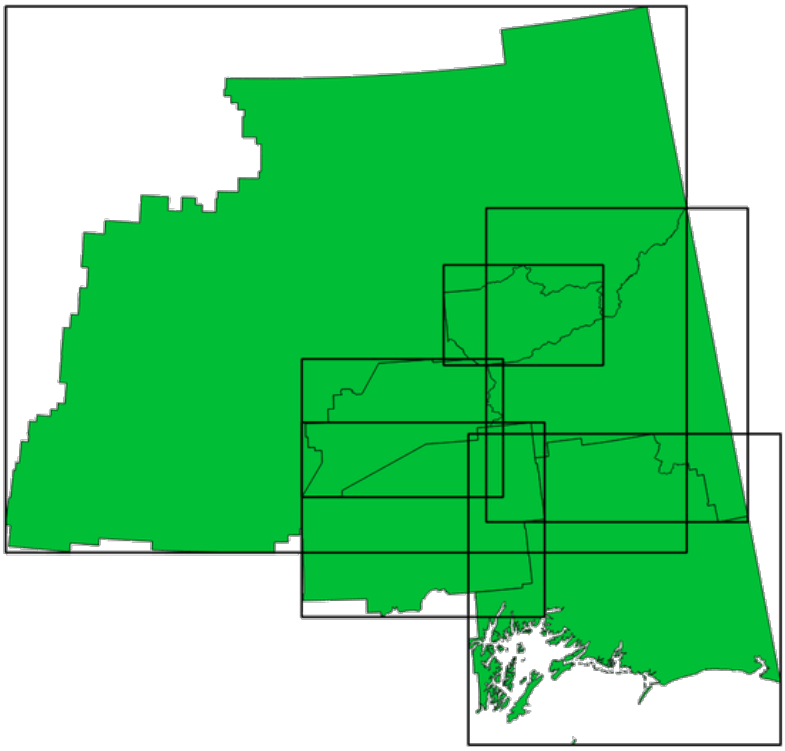


Fig. 9.6: Black lines represent the bounding boxes of each polygon feature

Further reading: *Bounding boxes* algorithm

bounds_height

Returns the height of the bounding box of a geometry. Calculations are in the Spatial Reference System of this geometry.

Syntax	bounds_height(geometry)
Arguments	• **geometry** - a geometry
Examples	• `bounds_height(@geometry)` → height of bounding box of the current feature's geometry • `bounds_height(geom_from_wkt('Polygon((1 1, 0 0, -1 1, 1 1))'))` → 1

bounds_width

Returns the width of the bounding box of a geometry. Calculations are in the Spatial Reference System of this geometry.

Syntax	bounds_width(geometry)
Arguments	• **geometry** - a geometry
Examples	• `bounds_width(@geometry)` → width of bounding box of the current feature's geometry • `bounds_width(geom_from_wkt('Polygon((1 1, 0 0, -1 1, 1 1))'))` → 2

buffer

Returns a geometry that represents all points whose distance from this geometry is less than or equal to distance. Calculations are in the Spatial Reference System of this geometry.

Syntax	buffer(geometry, distance, [segments=8], [cap='round'], [join='round'], [miter_limit=2]) [] marks optional arguments
Arguments	• **geometry** - a geometry • **distance** - buffer distance in layer units • **segments** - number of segments to use to represent a quarter circle when a round join style is used. A larger number results in a smoother buffer with more nodes. • **cap** - end cap style for buffer. Valid values are 'round', 'flat' or 'square' • **join** - join style for buffer. Valid values are 'round', 'bevel' or 'miter'. • **miter_limit** - miter distance limit, for use when the join style is set to 'miter'
Examples	• `buffer(@geometry, 10.5)` → polygon of the current feature's geometry buffered by 10.5 units

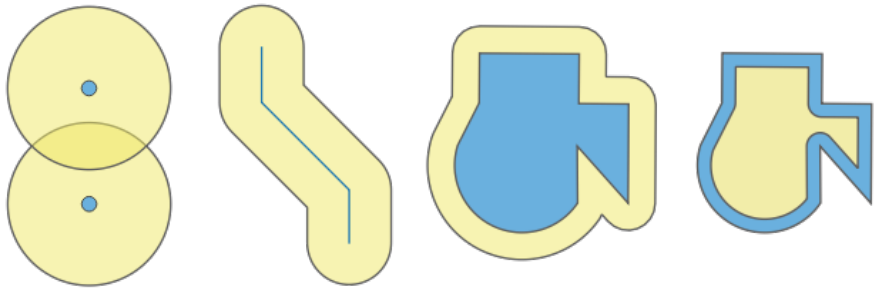


Fig. 9.7: Buffer (in yellow) of points, line, polygon with positive buffer, and polygon with negative buffer

Further reading: *Buffer* algorithm

buffer_by_m

Creates a buffer along a line geometry where the buffer diameter varies according to the m-values at the line vertices.

Syntax	buffer_by_m(geometry, [segments=8]) [] marks optional arguments
Arguments	• **geometry** - input geometry. Must be a (multi)line geometry with m values. • **segments** - number of segments to approximate quarter-circle curves in the buffer.
Examples	• `buffer_by_m(geometry:=geom_from_wkt('LINESTRINGM(1 2 0.5, 4 2 0.2)'),segments:=8)` → A variable width buffer starting with a diameter of 0.5 and ending with a diameter of 0.2 along the linestring geometry.

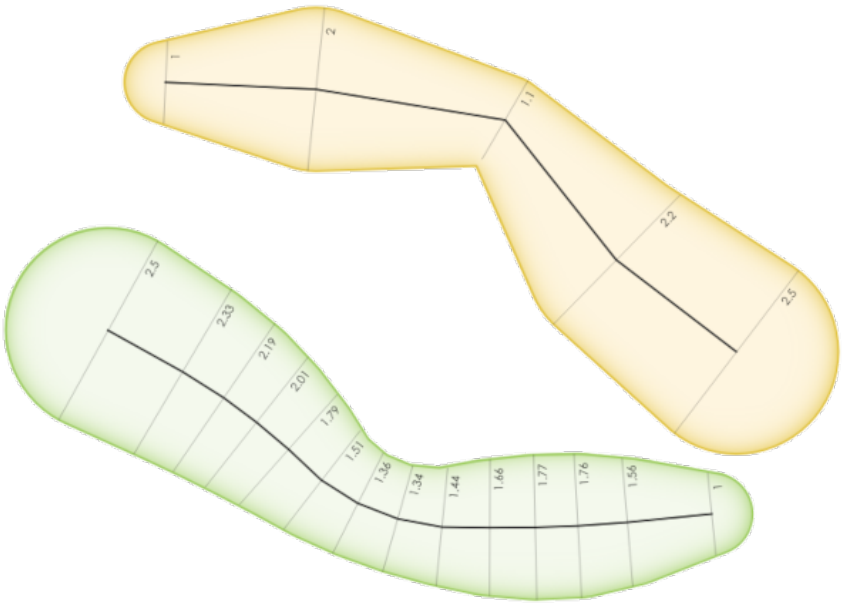


Fig. 9.8: Buffering line features using the m value on the vertices

Further reading: *Variable width buffer (by M value)* algorithm

centroid

Returns the geometric center of a geometry.

Syntax	centroid(geometry)
Arguments	• **geometry** - a geometry
Examples	• `centroid(@geometry)` → a point geometry

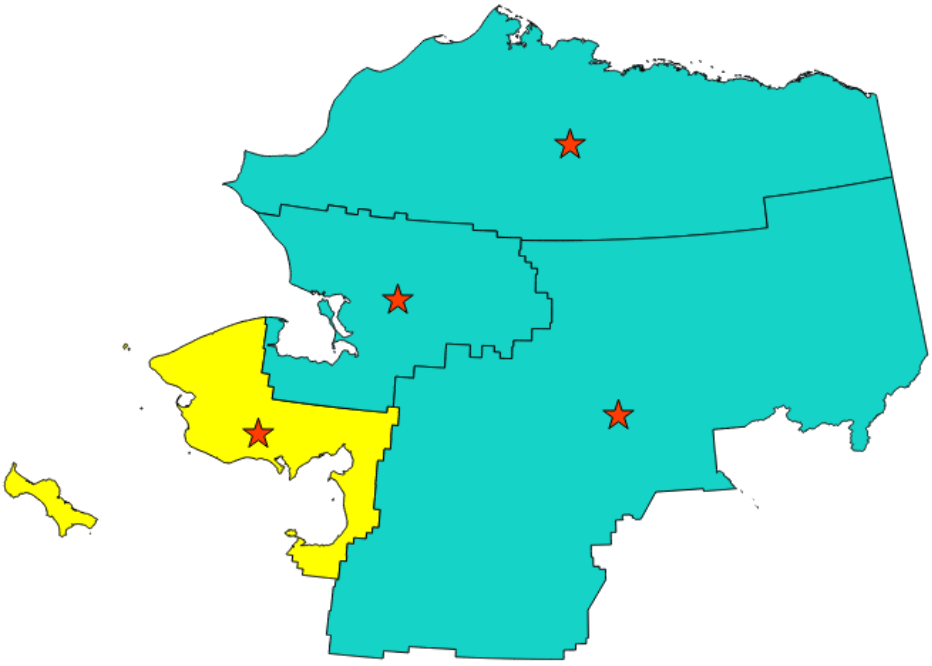


Fig. 9.9: The red stars represent the centroids of the features of the input layer.

Further reading: *Centroids* algorithm

close_line

Returns a closed line string of the input line string by appending the first point to the end of the line, if it is not already closed. If the geometry is not a line string or multi line string then the result will be NULL.

Syntax	close_line(geometry)
Arguments	• **geometry** - a line string geometry
Examples	• `geom_to_wkt(close_line(geom_from_wkt('LINESTRING(0 0, 1 0, 1 1)')))` → 'LineString (0 0, 1 0, 1 1, 0 0)' • `geom_to_wkt(close_line(geom_from_wkt('LINESTRING(0 0, 1 0, 1 1, 0 0)')))` → 'LineString (0 0, 1 0, 1 1, 0 0)'

closest_point

Returns the point on geometry1 that is closest to geometry2.

Syntax	closest_point(geometry1, geometry2)
Arguments	• **geometry1** - geometry to find closest point on • **geometry2** - geometry to find closest point to
Examples	• `geom_to_wkt(closest_point(geom_from_wkt('LINESTRING (20 80, 98 190, 110 180, 50 75 )'),geom_from_wkt('POINT(100 100)')))` → 'Point(73.0769 115.384)'

collect_geometries

Collects a set of geometries into a multi-part geometry object.

List of arguments variant

Geometry parts are specified as separate arguments to the function.

Syntax	collect_geometries(geometry1, geometry2, …)
Arguments	• **geometry** - a geometry
Examples	• `geom_to_wkt(collect_geometries(make_point(1,2), make_point(3,4), make_point(5,6)))` → 'MultiPoint ((1 2),(3 4),(5 6))'

Array variant

Geometry parts are specified as an array of geometry parts.

Syntax	collect_geometries(array)
Arguments	• **array** - array of geometry objects
Examples	• `geom_to_wkt(collect_geometries(array(make_point(1,2), make_point(3,4), make_point(5,6))))` → 'MultiPoint ((1 2),(3 4),(5 6))'

Further reading: *Collect geometries* algorithm

combine

Returns the combination of two geometries.

Syntax	combine(geometry1, geometry2)
Arguments	• **geometry1** - a geometry • **geometry2** - a geometry
Examples	• `geom_to_wkt( combine( geom_from_wkt( 'LINESTRING(3 3, 4 4, 5 5)' ), geom_from_wkt( 'LINESTRING(3 3, 4 4, 2 1)' ) ) )` → 'MULTILINESTRING((4 4, 2 1), (3 3, 4 4), (4 4, 5 5))' • `geom_to_wkt( combine( geom_from_wkt( 'LINESTRING(3 3, 4 4)' ), geom_from_wkt( 'LINESTRING(3 3, 6 6, 2 1)' ) ) )` → 'LINESTRING(3 3, 4 4, 6 6, 2 1)'

concave_hull

Returns a possibly concave polygon that contains all the points in the geometry

Syntax	concave_hull(geometry, target_percent, [allow_holes=False]) [] marks optional arguments
Arguments	• **geometry** - a geometry • **target_percent** - the percentage of area of the convex hull the solution tries to approach. A target_percent of 1 gives the same result as the convex hull. A target_percent between 0 and 0.99 produces a result that should have a smaller area than the convex hull. • **allow_holes** - optional argument specifying whether to allow holes within the output geometry. Defaults to FALSE, set to TRUE to allow including holes in the output geometry.
Examples	• `geom_to_wkt(concave_hull(geom_from_wkt('MULTILINESTRING((106 164,30 112,74 70,82 112,130 94,130 62,122 40,156 32,162 76,172 88),(132 178,134 148,128 136,96 128,132 108,150 130,170 142,174 110,156 96,158 90,158 88),(22 64,66 28,94 38,94 68,114 76,112 30,132 10,168 18,178 34,186 52,184 74,190 100,190 122,182 148,178 170,176 184,156 164,146 178,132 186,92 182,56 158,36 150,62 150,76 128,88 118))'), 0.99))` → 'Polygon ((30 112, 36 150, 92 182, 132 186, 176 184, 190 122, 190 100, 186 52, 178 34, 168 18, 132 10, 112 30, 66 28, 22 64, 30 112))'

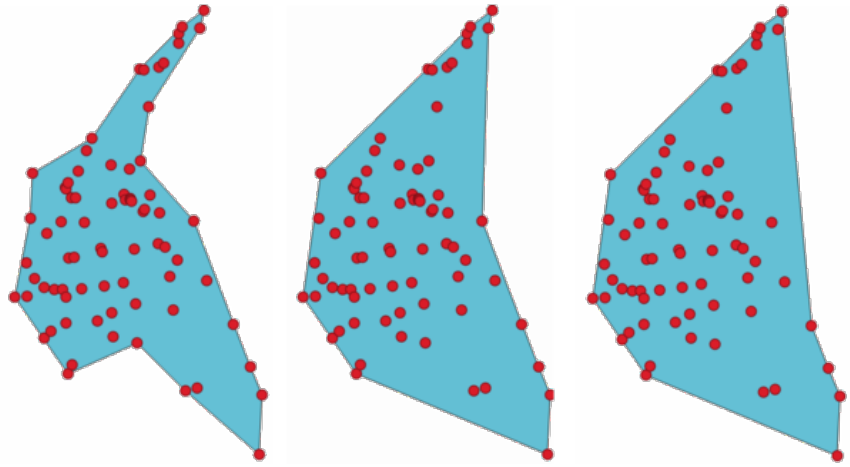


Fig. 9.10: Concave hulls with increasing target_percent parameter

Further reading: *convex_hull*, *Concave hull* algorithm

contains

Tests whether a geometry contains another. Returns TRUE if and only if no points of geometry2 lie in the exterior of geometry1, and at least one point of the interior of geometry2 lies in the interior of geometry1.

Syntax	contains(geometry1, geometry2)
Arguments	• **geometry1** - a geometry • **geometry2** - a geometry
Examples	• `contains( geom_from_wkt( 'POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))' ), geom_from_wkt( 'POINT(0.5 0.5 )' ) )` → TRUE • `contains( geom_from_wkt( 'POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))' ), geom_from_wkt( 'LINESTRING(3 3, 4 4, 5 5)' ) )` → FALSE

Further reading: *overlay_contains*

convex_hull

Returns the convex hull of a geometry. It represents the minimum convex geometry that encloses all geometries within the set.

Syntax	convex_hull(geometry)
Arguments	• **geometry** - a geometry
Examples	• `convex_hull(@geometry)` → convex hull of the current feature's geometry • `geom_to_wkt( convex_hull( geom_from_wkt( 'LINESTRING(3 3, 4 4, 4 10)' ) ) )` → 'POLYGON((3 3, 4 10, 4 4, 3 3))'

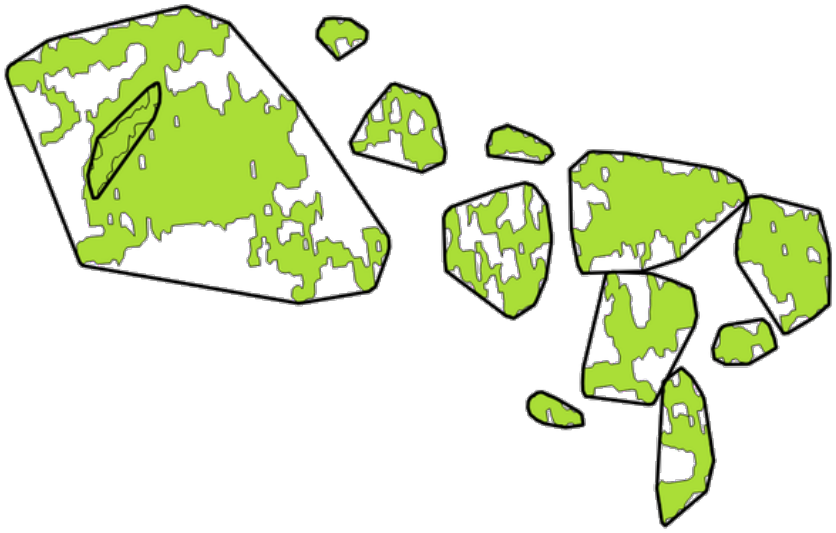


Fig. 9.11: Black lines identify the convex hull for each feature

Further reading: *concave_hull*, *Convex hull* algorithm

crosses

Tests whether a geometry crosses another. Returns TRUE if the supplied geometries have some, but not all, interior points in common.

Syntax	crosses(geometry1, geometry2)
Arguments	• **geometry1** - a geometry • **geometry2** - a geometry
Examples	• `crosses( geom_from_wkt( 'LINESTRING(3 5, 4 4, 5 3)' ), geom_from_wkt( 'LINESTRING(3 3, 4 4, 5 5)' ) )` → TRUE • `crosses( geom_from_wkt( 'POINT(4 5)' ), geom_from_wkt( 'LINESTRING(3 3, 4 4, 5 5)' ) )` → FALSE

Further reading: *overlay_crosses*

densify_by_count

Takes a polygon or line layer geometry and generates a new one in which the geometries have a larger number of vertices than the original one.

Syntax	densify_by_count(geometry, vertices)
Arguments	• **geometry** - a geometry (accepts (multi)linestrings or (multi)polygons). • **vertices** - number of vertices to add (per segment)
Examples	• `geom_to_wkt(densify_by_count(geom_from_wkt('LINESTRING(1 1, 10 1)'), 3))` → LineString (1 1, 3.25 1, 5.5 1, 7.75 1, 10 1)

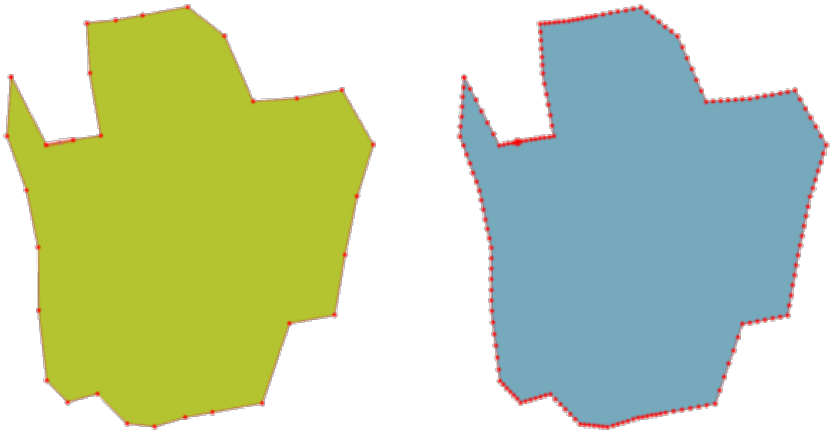


Fig. 9.12: Red points show the vertices before and after the densify

Further reading: *Densify by count* algorithm

densify_by_distance

Takes a polygon or line layer geometry and generates a new one in which the geometries are densified by adding additional vertices on edges that have a maximum distance of the specified interval distance.

Syntax	densify_by_distance(geometry, distance)
Arguments	• **geometry** - a geometry (accepts (multi)linestrings or (multi)polygons). • **distance** - maximum interval distance between vertices in output geometry
Examples	• `geom_to_wkt(densify_by_distance(geom_from_wkt('LINESTRING(1 1, 10 1)'), 4))` → LineString (1 1, 4 1, 7 1, 10 1)

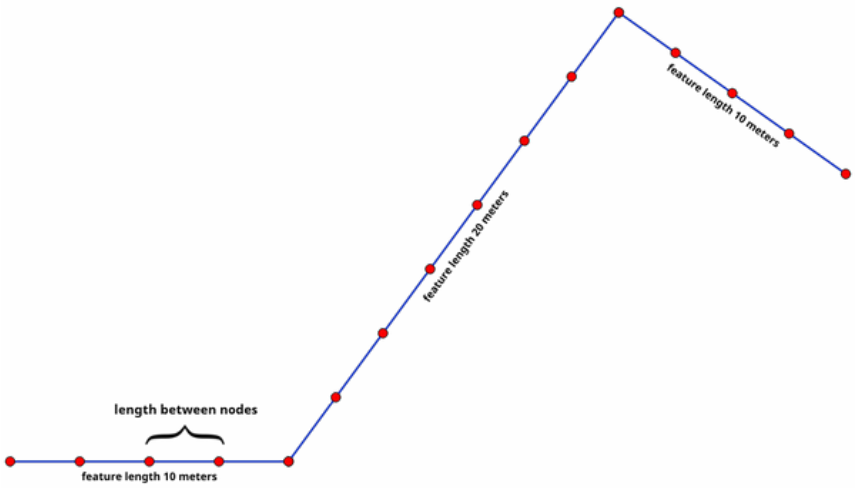


Fig. 9.13: Densify geometry at a given interval

Further reading: *Densify by interval* algorithm

difference

Returns a geometry that represents that part of geometry1 that does not intersect with geometry2.

Syntax	difference(geometry1, geometry2)
Arguments	• **geometry1** - a geometry • **geometry2** - a geometry
Examples	• `geom_to_wkt( difference( geom_from_wkt( 'LINESTRING(3 3, 4 4, 5 5)' ), geom_from_wkt( 'LINESTRING(3 3, 4 4)' ) ) )` → 'LINESTRING(4 4, 5 5)'

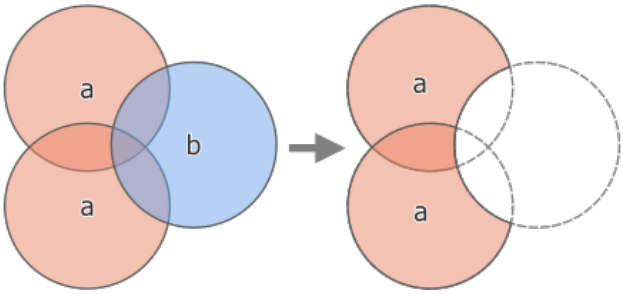


Fig. 9.14: Difference operation between a two-features input layer 'a' and a single feature overlay layer 'b' (left) - resulting in a new layer with the modified 'a' features (right)

Further reading: *Difference* algorithm

disjoint

Tests whether geometries do not spatially intersect. Returns TRUE if the geometries do not share any space together.

Syntax	disjoint(geometry1, geometry2)
Arguments	• **geometry1** - a geometry • **geometry2** - a geometry
Examples	• `disjoint( geom_from_wkt( 'POLYGON((0 0, 0 1, 1 1, 1 0, 0 0 ))' ), geom_from_wkt( 'LINESTRING(3 3, 4 4, 5 5)' ) )` → TRUE • `disjoint( geom_from_wkt( 'LINESTRING(3 3, 4 4, 5 5)' ), geom_from_wkt( 'POINT(4 4)' ))` → FALSE

Further reading: *overlay_disjoint*

distance

Returns the minimum distance (based on spatial reference) between two geometries in projected units.

Syntax	distance(geometry1, geometry2)
Arguments	• **geometry1** - a geometry • **geometry2** - a geometry
Examples	• `distance( geom_from_wkt( 'POINT(4 4)' ), geom_from_wkt( 'POINT(4 8)' ) )` → 4

distance_to_vertex

Returns the distance along the geometry to a specified vertex.

Syntax	distance_to_vertex(geometry, vertex)
Arguments	• **geometry** - a linestring geometry • **vertex** - vertex index, starting from 0; if the value is negative, the selected vertex index will be its total count minus the absolute value
Examples	• `distance_to_vertex(geometry:=geom_from_wkt('LineString(0 0, 10 0, 10 10)'),vertex:=1)` → 10.0

end_point

Returns the last node from a geometry.

Syntax	end_point(geometry)
Arguments	• **geometry** - geometry object
Examples	• `end_point(@geometry)` → end point of the current feature's geometry • `geom_to_wkt(end_point(geom_from_wkt('LINESTRING(4 0, 4 2, 0 2)')))` → 'Point (0 2)'

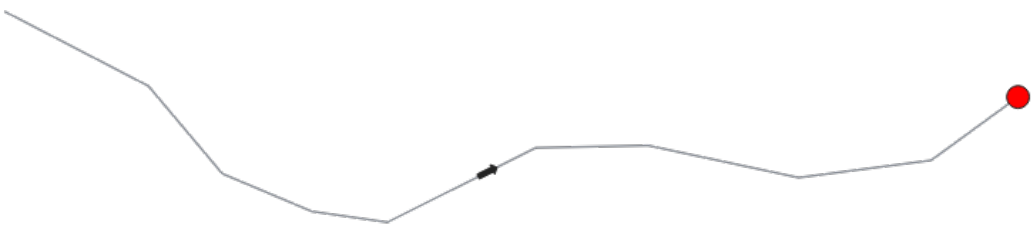


Fig. 9.15: End point of a line feature

Further reading: *start_point*, *Extract specific vertices* algorithm

exif_geotag

Creates a point geometry from the exif geotags of an image file.

Syntax	exif_geotag(path)
Arguments	• **path** - An image file path or a map layer value. If a map layer value is specified then the file source of the layer will be used.
Examples	• `geom_to_wkt(exif_geotag('/my/photo.jpg'))` → 'Point (2 4)'

extend

Extends the start and end of a linestring geometry by a specified amount. Lines are extended using the bearing of the first and last segment in the line. For a multilinestring, all the parts are extended. Distances are in the Spatial Reference System of this geometry.

Syntax	extend(geometry, start_distance, end_distance)
Arguments	• **geometry** - a (multi)linestring geometry • **start_distance** - distance to extend the start of the line • **end_distance** - distance to extend the end of the line.
Examples	• `geom_to_wkt(extend(geom_from_wkt('LineString(0 0, 1 0, 1 1)'),1,2))` → 'LineString (-1 0, 1 0, 1 3)' • `geom_to_wkt(extend(geom_from_wkt('MultiLineString((0 0, 1 0, 1 1), (2 2, 0 2, 0 5))'),1,2))` → 'MultiLineString ((-1 0, 1 0, 1 3),(3 2, 0 2, 0 7))'

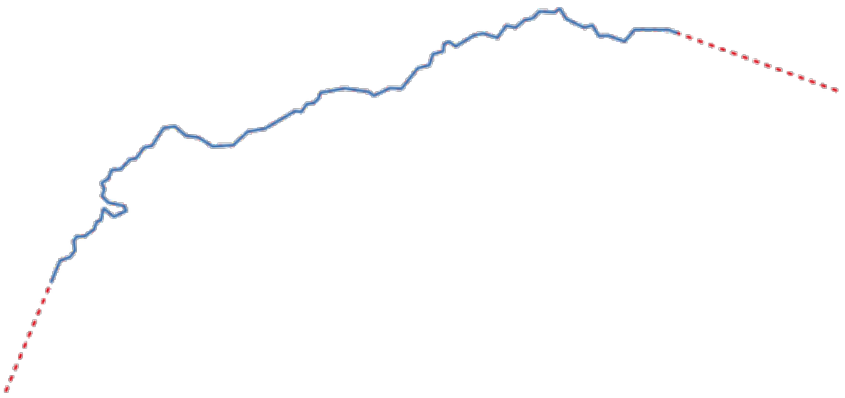


Fig. 9.16: The red dashes represent the initial and final extension of the original layer

Further reading: *Extend lines* algorithm

exterior_ring

Returns a line string representing the exterior ring of a polygon geometry. If the geometry is not a polygon then the result will be NULL.

Syntax	exterior_ring(geometry)
Arguments	• **geometry** - a polygon geometry
Examples	• `geom_to_wkt(exterior_ring(geom_from_wkt('POLYGON((-1 -1, 4 0, 4 2, 0 2, -1 -1),( 0.1 0.1, 0.1 0.2, 0.2 0.2, 0.2, 0.1, 0.1 0.1))')))` → 'LineString (-1 -1, 4 0, 4 2, 0 2, -1 -1)'

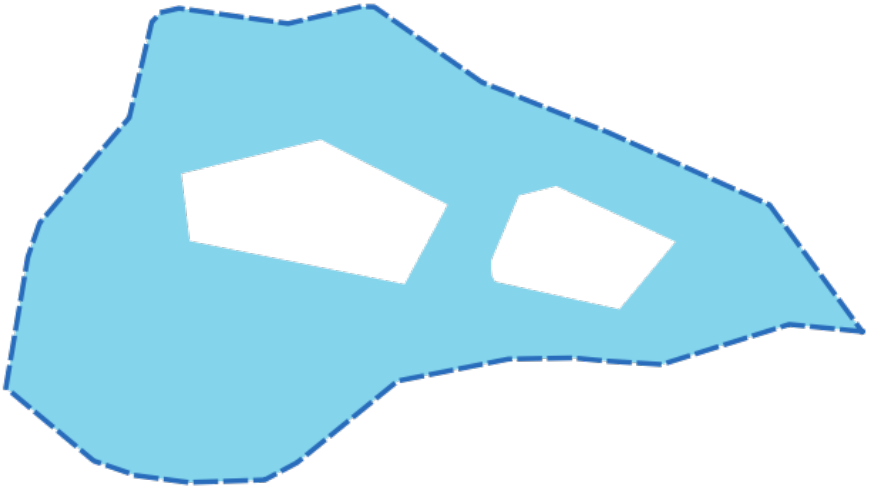


Fig. 9.17: The dashed line represents the exterior ring of the polygon

Further reading: *Boundary* algorithm, *interior_ring_n*

extrude

Returns an extruded version of the input (Multi-)Curve or (Multi-)Linestring geometry with an extension specified by x and y.

Syntax	extrude(geometry, x, y)
Arguments	• **geometry** - a curve or linestring geometry • **x** - x extension, numeric value • **y** - y extension, numeric value
Examples	• `geom_to_wkt(extrude(geom_from_wkt('LineString(1 2, 3 2, 4 3)'), 1, 2))` → 'Polygon ((1 2, 3 2, 4 3, 5 5, 4 4, 2 4, 1 2))' • `geom_to_wkt(extrude(geom_from_wkt('MultiLineString((1 2, 3 2), (4 3, 8 3))'), 1, 2))` → 'MultiPolygon (((1 2, 3 2, 4 4, 2 4, 1 2)),((4 3, 8 3, 9 5, 5 5, 4 3)))'

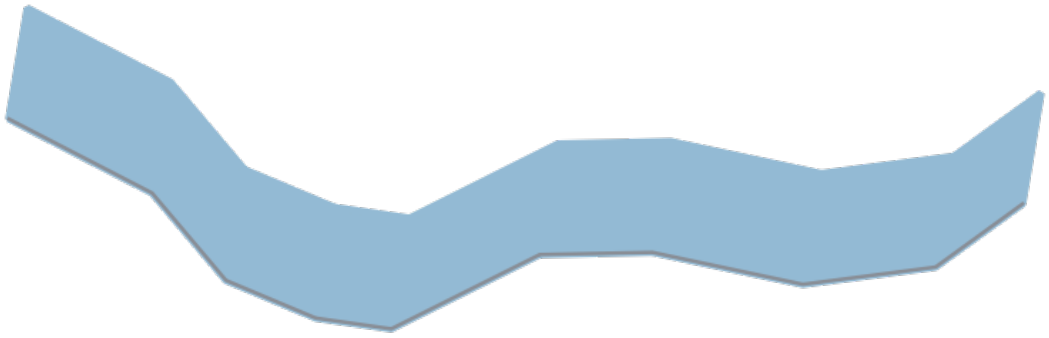


Fig. 9.18: Generating a polygon by extruding a line with offset in x and y directions

flip_coordinates

Returns a copy of the geometry with the x and y coordinates swapped. Useful for repairing geometries which have had their latitude and longitude values reversed.

Syntax	flip_coordinates(geometry)
Arguments	• **geometry** - a geometry
Examples	• `geom_to_wkt(flip_coordinates(make_point(1, 2)))` → 'Point (2 1)' • `geom_to_wkt(flip_coordinates(geom_from_wkt('LineString(0 2, 1 0, 1 6)')))` → 'LineString (2 0, 0 1, 6 1)'

Further reading: *Swap X and Y coordinates* algorithm

force_polygon_ccw

Forces a geometry to respect the convention where exterior rings are counter-clockwise, interior rings are clockwise.

Syntax	force_polygon_ccw(geometry)
Arguments	• **geometry** - a geometry. Any non-polygon geometries are returned unchanged.
Examples	• `geom_to_wkt(force_polygon_ccw(geometry:=geom_from_wkt('Polygon ((-1 -1, 0 2, 4 2, 4 0, -1 -1))')))` → 'Polygon ((-1 -1, 4 0, 4 2, 0 2, -1 -1))'

Further reading: *force_polygon_cw*, *force_rhr*

force_polygon_cw

Forces a geometry to respect the convention where exterior rings are clockwise, interior rings are counter-clockwise.

Syntax	force_polygon_cw(geometry)
Arguments	• **geometry** - a geometry. Any non-polygon geometries are returned unchanged.
Examples	• `geom_to_wkt(force_polygon_cw(geometry:=geom_from_wkt('POLYGON((-1 -1, 4 0, 4 2, 0 2, -1 -1))')))` → 'Polygon ((-1 -1, 0 2, 4 2, 4 0, -1 -1))'

Further reading: *force_polygon_ccw*, *force_rhr*

force_rhr

Forces a geometry to respect the Right-Hand-Rule, in which the area that is bounded by a polygon is to the right of the boundary. In particular, the exterior ring is oriented in a clockwise direction and the interior rings in a counter-clockwise direction. Due to the inconsistency in the definition of the Right-Hand-Rule in some contexts it is recommended to use the explicit force_polygon_cw function instead.

Syntax	force_rhr(geometry)
Arguments	• **geometry** - a geometry. Any non-polygon geometries are returned unchanged.
Examples	• `geom_to_wkt(force_rhr(geometry:=geom_from_wkt('POLYGON((-1 -1, 4 0, 4 2, 0 2, -1 -1))')))` → 'Polygon ((-1 -1, 0 2, 4 2, 4 0, -1 -1))'

Further reading: *Force right-hand-rule* algorithm, *force_polygon_ccw*, *force_polygon_cw*

geom_from_gml

Returns a geometry from a GML representation of geometry.

Syntax	geom_from_gml(gml)
Arguments	• **gml** - GML representation of a geometry as a string
Examples	• `geom_from_gml('<gml:LineString srsName="EPSG:4326"><gml:coordinates>4, 4 5,5 6,6</gml:coordinates></gml:LineString>')` → a line geometry object

geom_from_wkb

Returns a geometry created from a Well-Known Binary (WKB) representation.

Syntax	geom_from_wkb(binary)
Arguments	• **binary** - Well-Known Binary (WKB) representation of a geometry (as a binary blob)
Examples	• `geom_from_wkb( geom_to_wkb( make_point(4,5) ) )` → a point geometry object

geom_from_wkt

Returns a geometry created from a Well-Known Text (WKT) representation.

Syntax	geom_from_wkt(text)
Arguments	• **text** - Well-Known Text (WKT) representation of a geometry
Examples	• `geom_from_wkt( 'POINT(4 5)' )` → a geometry object

geom_to_wkb

Returns the Well-Known Binary (WKB) representation of a geometry

Syntax	geom_to_wkb(geometry)
Arguments	• **geometry** - a geometry
Examples	• `geom_to_wkb( @geometry )` → binary blob of the current feature's geometry

geom_to_wkt

Returns the Well-Known Text (WKT) representation of the geometry without SRID metadata.

Syntax	geom_to_wkt(geometry, [precision=8]) [] marks optional arguments
Arguments	• **geometry** - a geometry • **precision** - numeric precision
Examples	• `geom_to_wkt(@geometry)` → well-known text representation of the current feature's geometry • `geom_to_wkt( make_point(6, 50) )` → 'POINT(6 50)' • `geom_to_wkt(centroid(geom_from_wkt('Polygon((1 1, 0 0, -1 1, 1 1))')))` → 'POINT(0 0.66666667)' • `geom_to_wkt(centroid(geom_from_wkt('Polygon((1 1, 0 0, -1 1, 1 1))')), 2)` → 'POINT(0 0.67)'

$geometry

Returns the geometry of the current feature. Can be used for processing with other functions. **WARNING: This function is deprecated. It is recommended to use the replacement @geometry variable instead.**

Syntax	$geometry
Examples	• `geom_to_wkt( $geometry )` → 'POINT(6 50)'

geometry

Returns a feature's geometry.

Syntax	geometry(feature)
Arguments	• **feature** - a feature object
Examples	• `geometry( @feature )` → the geometry of the current feature. Prefer using @geometry. • `geom_to_wkt( geometry( get_feature_by_id( 'streets', 1 ) ) )` → the geometry in WKT of the feature with the id 1 on the layer "streets", e.g. 'POINT(6 50)' • `intersects( @geometry, geometry( get_feature( 'streets', 'name', 'Main St.' ) ) )` → TRUE if the current feature spatially intersects the 'Main St.' named feature in the "streets" layer

geometry_n

Returns a specific geometry from a geometry collection, or NULL if the input geometry is not a collection. Also returns a part from a multipart geometry.

Syntax	geometry_n(geometry, index)
Arguments	• **geometry** - geometry collection • **index** - index of geometry to return, where 1 is the first geometry in the collection
Examples	• `geom_to_wkt(geometry_n(geom_from_wkt('GEOMETRYCOLLECTION(POINT(0 1), POINT(0 0), POINT(1 0), POINT(1 1))'),3))` → 'Point (1 0)'

geometry_type

Returns a string value describing the type of a geometry (Point, Line or Polygon)

Syntax	geometry_type(geometry)
Arguments	• **geometry** - a geometry
Examples	• `geometry_type( geom_from_wkt( 'LINESTRING(2 5, 3 6, 4 8)') )` → 'Line' • `geometry_type( geom_from_wkt( 'MULTILINESTRING((2 5, 3 6, 4 8), (1 1, 0 0))') )` → 'Line' • `geometry_type( geom_from_wkt( 'POINT(2 5)') )` → 'Point' • `geometry_type( geom_from_wkt( 'POLYGON((-1 -1, 4 0, 4 2, 0 2, -1 -1))') )` → 'Polygon'

hausdorff_distance

Returns the Hausdorff distance between two geometries. This is basically a measure of how similar or dissimilar 2 geometries are, with a lower distance indicating more similar geometries.

The function can be executed with an optional densify fraction argument. If not specified, an approximation to the standard Hausdorff distance is used. This approximation is exact or close enough for a large subset of useful cases. Examples of these are:

- computing distance between Linestrings that are roughly parallel to each other, and roughly equal in length. This occurs in matching linear networks.
- Testing similarity of geometries.

If the default approximate provided by this method is insufficient, specify the optional densify fraction argument. Specifying this argument performs a segment densification before computing the discrete Hausdorff distance. The parameter sets the fraction by which to densify each segment. Each segment will be split into a number of equal-length subsegments, whose fraction of the total length is closest to the given fraction. Decreasing the densify fraction parameter will make the distance returned approach the true Hausdorff distance for the geometries.

Syntax	hausdorff_distance(geometry1, geometry2, [densify_fraction]) [] marks optional arguments
Arguments	• **geometry1** - a geometry • **geometry2** - a geometry • **densify_fraction** - densify fraction amount
Examples	• `hausdorff_distance( geometry1:= geom_from_wkt('LINESTRING (0 0, 2 1)'),geometry2:=geom_from_wkt('LINESTRING (0 0, 2 0)'))` → 2 • `hausdorff_distance( geom_from_wkt('LINESTRING (130 0, 0 0, 0 150)'),geom_from_wkt('LINESTRING (10 10, 10 150, 130 10)'))` → 14.142135623 • `hausdorff_distance( geom_from_wkt('LINESTRING (130 0, 0 0, 0 150)'),geom_from_wkt('LINESTRING (10 10, 10 150, 130 10)'), 0.5)` → 70.0

inclination

Returns the inclination measured from the zenith (0) to the nadir (180) on point_a to point_b.

Syntax	inclination(point_a, point_b)
Arguments	• **point_a** - point geometry • **point_b** - point geometry
Examples	• `inclination( make_point( 5, 10, 0 ), make_point( 5, 10, 5 ) )` → 0.0 • `inclination( make_point( 5, 10, 0 ), make_point( 5, 10, 0 ) )` → 90.0 • `inclination( make_point( 5, 10, 0 ), make_point( 50, 100, 0 ) )` → 90.0 • `inclination( make_point( 5, 10, 0 ), make_point( 5, 10, -5 ) )` → 180.0

interior_ring_n

Returns a specific interior ring from a polygon geometry, or NULL if the geometry is not a polygon.

Syntax	interior_ring_n(geometry, index)
Arguments	• **geometry** - polygon geometry • **index** - index of interior to return, where 1 is the first interior ring
Examples	• `geom_to_wkt(interior_ring_n(geom_from_wkt('POLYGON((-1 -1, 4 0, 4 2, 0 2, -1 -1),(-0.1 -0.1, 0.4 0, 0.4 0.2, 0 0.2, -0.1 -0.1),(-1 -1, 4 0, 4 2, 0 2, -1 -1))'),1))` → 'LineString (-0.1 -0.1, 0.4 0, 0.4 0.2, 0 0.2, -0.1 -0.1)'

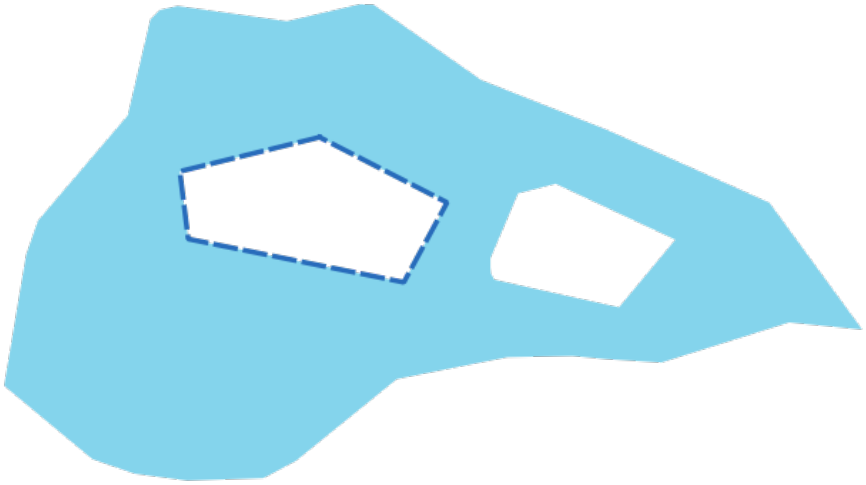


Fig. 9.19: The dashed line represents a specific interior ring of the polygon

Further reading: *exterior_ring*

intersection

Returns a geometry that represents the shared portion of two geometries.

Syntax	intersection(geometry1, geometry2)
Arguments	• **geometry1** - a geometry • **geometry2** - a geometry
Examples	• `geom_to_wkt( intersection( geom_from_wkt( 'LINESTRING(3 3, 4 4, 5 5)' ), geom_from_wkt( 'LINESTRING(3 3, 4 4)' ) ) )` → 'LINESTRING(3 3, 4 4)' • `geom_to_wkt( intersection( geom_from_wkt( 'LINESTRING(3 3, 4 4, 5 5)' ), geom_from_wkt( 'MULTIPOINT(3.5 3.5, 4 5)' ) ) )` → 'POINT(3.5 3.5)'

Further reading: *Intersection* algorithm

intersects

Tests whether a geometry intersects another. Returns TRUE if the geometries spatially intersect (share any portion of space) and false if they do not.

Syntax	intersects(geometry1, geometry2)
Arguments	• **geometry1** - a geometry • **geometry2** - a geometry
Examples	• `intersects( geom_from_wkt( 'POINT(4 4)' ), geom_from_wkt( 'LINESTRING(3 3, 4 4, 5 5)' ) )` → TRUE • `intersects( geom_from_wkt( 'POINT(4 5)' ), geom_from_wkt( 'POINT(5 5)' ) )` → FALSE

Further reading: *overlay_intersects*

intersects_bbox

Tests whether a geometry's bounding box overlaps another geometry's bounding box. Returns TRUE if the geometries spatially intersect the bounding box defined and false if they do not.

Syntax	intersects_bbox(geometry1, geometry2)
Arguments	• **geometry1** - a geometry • **geometry2** - a geometry
Examples	• `intersects_bbox( geom_from_wkt( 'POINT(4 5)' ), geom_from_wkt( 'LINESTRING(3 3, 4 4, 5 5)' ) )` → TRUE • `intersects_bbox( geom_from_wkt( 'POINT(6 5)' ), geom_from_wkt( 'POLYGON((3 3, 4 4, 5 5, 3 3))' ) )` → FALSE

is_closed

Returns TRUE if a line string is closed (start and end points are coincident), or false if a line string is not closed. If the geometry is not a line string then the result will be NULL.

Syntax	is_closed(geometry)
Arguments	• **geometry** - a line string geometry
Examples	• `is_closed(geom_from_wkt('LINESTRING(0 0, 1 1, 2 2)'))` → FALSE • `is_closed(geom_from_wkt('LINESTRING(0 0, 1 1, 2 2, 0 0)'))` → TRUE

is_empty

Returns TRUE if a geometry is empty (without coordinates), false if the geometry is not empty and NULL if there is no geometry. See also is_empty_or_null.

Syntax	is_empty(geometry)
Arguments	• **geometry** - a geometry
Examples	• `is_empty(geom_from_wkt('LINESTRING(0 0, 1 1, 2 2)'))` → FALSE • `is_empty(geom_from_wkt('LINESTRING EMPTY'))` → TRUE • `is_empty(geom_from_wkt('POINT(7 4)'))` → FALSE • `is_empty(geom_from_wkt('POINT EMPTY'))` → TRUE

Further reading: *is_empty_or_null*

is_empty_or_null

Returns TRUE if a geometry is NULL or empty (without coordinates) or false otherwise. This function is like the expression '@geometry IS NULL or is_empty(@geometry)'

Syntax	is_empty_or_null(geometry)
Arguments	• **geometry** - a geometry
Examples	• `is_empty_or_null(NULL)` → TRUE • `is_empty_or_null(geom_from_wkt('LINESTRING(0 0, 1 1, 2 2)'))` → FALSE • `is_empty_or_null(geom_from_wkt('LINESTRING EMPTY'))` → TRUE • `is_empty_or_null(geom_from_wkt('POINT(7 4)'))` → FALSE • `is_empty_or_null(geom_from_wkt('POINT EMPTY'))` → TRUE

Further reading: *is_empty*, *NULL*

is_multipart

Returns TRUE if the geometry is of Multi type.

Syntax	is_multipart(geometry)
Arguments	• **geometry** - a geometry
Examples	• `is_multipart(@geometry)` → checks if the current feature's geometry is multipart • `is_multipart(geom_from_wkt('MULTIPOINT ((0 0),(1 1),(2 2))'))` → TRUE • `is_multipart(geom_from_wkt('POINT (0 0)'))` → FALSE

is_valid

Returns TRUE if a geometry is valid; if it is well-formed in 2D according to the OGC rules.

Syntax	is_valid(geometry)
Arguments	• **geometry** - a geometry
Examples	• `is_valid(@geometry)` → checks if the current feature's geometry is valid • `is_valid(geom_from_wkt('LINESTRING(0 0, 1 1, 2 2, 0 0)'))` → TRUE • `is_valid(geom_from_wkt('LINESTRING(0 0)'))` → FALSE

Further reading: *make_valid*, *Check validity* algorithm

$length

Returns the length of a linestring. If you need the length of a border of a polygon, use $perimeter instead. The length calculated by this function respects both the current project's ellipsoid setting and distance unit settings. For example, if an ellipsoid has been set for the project then the calculated length will be ellipsoidal, and if no ellipsoid is set then the calculated length will be planimetric.

Syntax	$length
Examples	• `$length` → 42.4711

length

Returns the number of characters in a string or the length of a geometry linestring.

String variant

Returns the number of characters in a string.

Syntax	length(string)
Arguments	• **string** - string to count length of
Examples	• `length('hello')` → 5

Geometry variant

Calculate the length of a geometry line object. Calculations are always planimetric in the Spatial Reference System (SRS) of this geometry, and the units of the returned length will match the units for the SRS. This differs from the calculations performed by the $length function, which will perform ellipsoidal calculations based on the project's ellipsoid and distance unit settings.

Syntax	length(geometry)
Arguments	• **geometry** - line geometry object
Examples	• `length(@geometry)` → length of the current feature's geometry • `length(geom_from_wkt('LINESTRING(0 0, 4 0)'))` → 4.0

Further reading: *straight_distance_2d*

length3D

Calculates the 3D length of a geometry line object. If the geometry is not a 3D line object, it returns its 2D length. Calculations are always planimetric in the Spatial Reference System (SRS) of this geometry, and the units of the returned length will match the units for the SRS. This differs from the calculations performed by the $length function, which will perform ellipsoidal calculations based on the project's ellipsoid and distance unit settings.

Syntax	length3D(geometry)
Arguments	• **geometry** - line geometry object
Examples	• `length3D(@geometry)` → 3D length of the current feature's geometry • `length3D(geom_from_wkt('LINESTRINGZ(0 0 0, 3 0 4)'))` → 5.0

line_interpolate_angle

Returns the angle parallel to the geometry at a specified distance along a linestring geometry. Angles are in degrees clockwise from north.

Syntax	line_interpolate_angle(geometry, distance)
Arguments	• **geometry** - a linestring geometry • **distance** - distance along line to interpolate angle at
Examples	• `line_interpolate_angle(geometry:=geom_from_wkt('LineString(0 0, 10 0)'),distance:=5)` → 90.0

line_interpolate_point

Returns the point interpolated by a specified distance along a linestring geometry.

Syntax	line_interpolate_point(geometry, distance)
Arguments	• **geometry** - a linestring geometry • **distance** - distance along line to interpolate
Examples	• `geom_to_wkt(line_interpolate_point(geometry:=geom_from_wkt('LineString(0, 8 0)'), distance:=5))` → 'Point (5 0)' • `geom_to_wkt(line_interpolate_point(geometry:=geom_from_wkt('LineString(0, 1 1, 2 0)'), distance:=2.1))` → 'Point (1.48492424 0.51507576)' • `geom_to_wkt(line_interpolate_point(geometry:=geom_from_wkt('LineString(0, 1 0)'), distance:=2))` → NULL

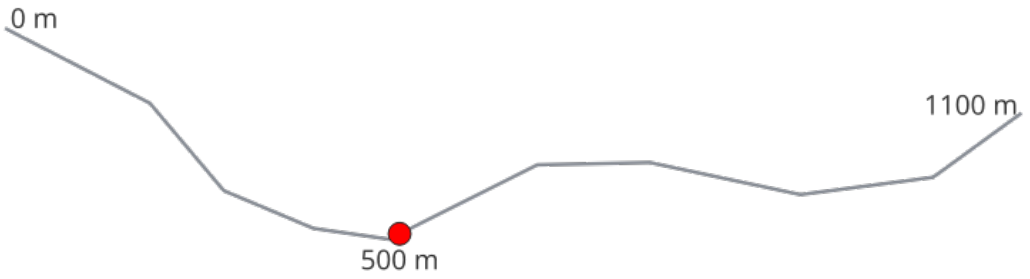


Fig. 9.20: Interpolated point at 500m of the beginning of the line

Further reading: *Interpolate point on line* algorithm

line_interpolate_point_by_m

Returns the point interpolated by a matching M value along a linestring geometry.

Syntax	line_interpolate_point_by_m(geometry, m, [use_3d_distance=false]) [] marks optional arguments
Arguments	• **geometry** - a linestring geometry • **m** - an M value • **use_3d_distance** - controls whether 2D or 3D distances between vertices should be used during interpolation (this option is only considered for lines with z values)
Examples	• `geom_to_wkt(line_interpolate_point_by_m(geom_from_wkt('LineStringM(0 0 0, 10 10 10)'), m:=5))` → 'Point (5 5)'

line_locate_m

Returns the distance along a linestring corresponding to the first matching interpolated M value.

Syntax	line_locate_m(geometry, m, [use_3d_distance=false]) [] marks optional arguments
Arguments	• **geometry** - a linestring geometry • **m** - an M value • **use_3d_distance** - controls whether 2D or 3D distances between vertices should be used during interpolation (this option is only considered for lines with z values)
Examples	• `line_locate_m(geometry:=geom_from_wkt('LineStringM(0 0 0, 10 10 10)'),m:=5)` → 7.07106

line_locate_point

Returns the distance along a linestring corresponding to the closest position the linestring comes to a specified point geometry.

Syntax	line_locate_point(geometry, point)
Arguments	• **geometry** - a linestring geometry • **point** - point geometry to locate closest position on linestring to
Examples	• `line_locate_point(geometry:=geom_from_wkt('LineString(0 0, 10 0)'),point:=geom_from_wkt('Point(5 0)'))` → 5.0

line_merge

Returns a LineString or MultiLineString geometry, where any connected LineStrings from the input geometry have been merged into a single linestring. This function will return NULL if passed a geometry which is not a LineString/MultiLineString.

Syntax	line_merge(geometry)
Arguments	• **geometry** - a LineString/MultiLineString geometry
Examples	• `geom_to_wkt(line_merge(geom_from_wkt('MULTILINESTRING((0 0, 1 1),(1 1, 2 2))')))` → 'LineString(0 0,1 1,2 2)' • `geom_to_wkt(line_merge(geom_from_wkt('MULTILINESTRING((0 0, 1 1),(11 1, 21 2))')))` → 'MultiLineString((0 0, 1 1),(11 1, 21 2)'

line_substring

Returns the portion of a line (or curve) geometry which falls between the specified start and end distances (measured from the beginning of the line). Z and M values are linearly interpolated from existing values.

Syntax	line_substring(geometry, start_distance, end_distance)
Arguments	• **geometry** - a linestring or curve geometry • **start_distance** - distance to start of substring • **end_distance** - distance to end of substring
Examples	• `geom_to_wkt(line_substring(geometry:=geom_from_wkt('LineString(0 0, 10 0)'),start_distance:=2,end_distance:=6))` → 'LineString (2 0,6 0)'



Fig. 9.21: Substring line with starting distance set at 0 meters and the ending distance at 250 meters.

Further reading: *Line substring* algorithm

m

Returns the m (measure) value of a point geometry.

Syntax	m(geometry)
Arguments	• **geometry** - a point geometry
Examples	• `m( geom_from_wkt( 'POINTM(2 5 4)' ) )` → 4

m_at

Retrieves a m coordinate of the geometry, or NULL if the geometry has no m value.

Syntax	m_at(geometry, vertex)
Arguments	• **geometry** - geometry object • **vertex** - index of the vertex of the geometry (indices start at 0; negative values apply from the last index, starting at -1)
Examples	• `m_at(geom_from_wkt('LineStringZM(0 0 0 0, 10 10 0 5, 10 10 0 0)'), 1)` → 5

m_max

Returns the maximum m (measure) value of a geometry.

Syntax	m_max(geometry)
Arguments	• **geometry** - a geometry containing m values
Examples	• `m_max( make_point_m( 0,0,1 ) )` → 1 • `m_max(make_line( make_point_m( 0,0,1 ), make_point_m( -1,-1,2 ), make_point_m( -2,-2,0 ) ) )` → 2

m_min

Returns the minimum m (measure) value of a geometry.

Syntax	m_min(geometry)
Arguments	• **geometry** - a geometry containing m values
Examples	• `m_min( make_point_m( 0,0,1 ) )` → 1 • `m_min(make_line( make_point_m( 0,0,1 ), make_point_m( -1,-1,2 ), make_point_m( -2,-2,0 ) ) )` → 0

main_angle

Returns the angle of the long axis (clockwise, in degrees from North) of the oriented minimal bounding rectangle, which completely covers the geometry.

Syntax	main_angle(geometry)
Arguments	• **geometry** - a geometry
Examples	• `main_angle(geom_from_wkt('Polygon ((321577 129614, 321581 129618, 321585 129615, 321581 129610, 321577 129614))'))` → 38.66

make_circle

Creates a circular polygon.

Syntax	make_circle(center, radius, [segments=36]) [] marks optional arguments
Arguments	• **center** - center point of the circle • **radius** - radius of the circle • **segments** - optional argument for polygon segmentation. By default this value is 36
Examples	• `geom_to_wkt(make_circle(make_point(10,10), 5, 4))` → 'Polygon ((10 15, 15 10, 10 5, 5 10, 10 15))' • `geom_to_wkt(make_circle(make_point(10,10,5), 5, 4))` → 'PolygonZ ((10 15 5, 15 10 5, 10 5 5, 5 10 5, 10 15 5))' • `geom_to_wkt(make_circle(make_point(10,10,5,30), 5, 4))` → 'PolygonZM ((10 15 5 30, 15 10 5 30, 10 5 5 30, 5 10 5 30, 10 15 5 30))'

make_ellipse

Creates an elliptical polygon.

Syntax	make_ellipse(center, semi_major_axis, semi_minor_axis, azimuth, [segments=36]) [] marks optional arguments
Arguments	• **center** - center point of the ellipse • **semi_major_axis** - semi-major axis of the ellipse • **semi_minor_axis** - semi-minor axis of the ellipse • **azimuth** - orientation of the ellipse • **segments** - optional argument for polygon segmentation. By default this value is 36
Examples	• `geom_to_wkt(make_ellipse(make_point(10,10), 5, 2, 90, 4))` → 'Polygon ((15 10, 10 8, 5 10, 10 12, 15 10))' • `geom_to_wkt(make_ellipse(make_point(10,10,5), 5, 2, 90, 4))` → 'PolygonZ ((15 10 5, 10 8 5, 5 10 5, 10 12 5, 15 10 5))' • `geom_to_wkt(make_ellipse(make_point(10,10,5,30), 5, 2, 90, 4))` → 'PolygonZM ((15 10 5 30, 10 8 5 30, 5 10 5 30, 10 12 5 30, 15 10 5 30))'

make_line

Creates a line geometry from a series of point geometries.

List of arguments variant

Line vertices are specified as separate arguments to the function.

Syntax	make_line(point1, point2, …)
Arguments	• **point** - a point geometry (or array of points)
Examples	• `geom_to_wkt(make_line(make_point(2,4),make_point(3,5)))` → 'LineString (2 4, 3 5)' • `geom_to_wkt(make_line(make_point(2,4),make_point(3,5),make_point(9,7)))` → 'LineString (2 4, 3 5, 9 7)'

Array variant

Line vertices are specified as an array of points.

Syntax	make_line(array)
Arguments	• **array** - array of points
Examples	• `geom_to_wkt(make_line(array(make_point(2,4),make_point(3,5),make_point(9,7))))` → 'LineString (2 4, 3 5, 9 7)'

make_point

Creates a point geometry from an x and y (and optional z and m) value.

Syntax	make_point(x, y, [z], [m]) [] marks optional arguments
Arguments	• **x** - x coordinate of point • **y** - y coordinate of point • **z** - optional z coordinate of point • **m** - optional m value of point
Examples	• `geom_to_wkt(make_point(2,4))` → 'Point (2 4)' • `geom_to_wkt(make_point(2,4,6))` → 'PointZ (2 4 6)' • `geom_to_wkt(make_point(2,4,6,8))` → 'PointZM (2 4 6 8)'

make_point_m

Creates a point geometry from an x, y coordinate and m value.

Syntax	make_point_m(x, y, m)
Arguments	• **x** - x coordinate of point • **y** - y coordinate of point • **m** - m value of point
Examples	• `geom_to_wkt(make_point_m(2,4,6))` → 'PointM (2 4 6)'

make_polygon

Creates a polygon geometry from an outer ring and optional series of inner ring geometries.

Syntax	make_polygon(outerRing, [innerRing1], [innerRing2], …) [] marks optional arguments
Arguments	• **outerRing** - closed line geometry for polygon's outer ring • **innerRing** - optional closed line geometry for inner ring
Examples	• `geom_to_wkt(make_polygon(geom_from_wkt('LINESTRING( 0 0, 0 1, 1 1, 1 0, 0 0 )')))` → 'Polygon ((0 0, 0 1, 1 1, 1 0, 0 0))' • `geom_to_wkt(make_polygon(geom_from_wkt('LINESTRING( 0 0, 0 1, 1 1, 1 0, 0 0 )'),geom_from_wkt('LINESTRING( 0.1 0.1, 0.1 0.2, 0.2 0.2, 0.2 0.1, 0.1 0.1 )'), geom_from_wkt('LINESTRING( 0.8 0.8, 0.8 0.9, 0.9 0.9, 0.9 0.8, 0.8 0.8 )')))` → 'Polygon ((0 0, 0 1, 1 1, 1 0, 0 0),(0.1 0.1, 0.1 0.2, 0.2 0.2, 0.2 0.1, 0.1 0.1),(0.8 0.8, 0.8 0.9, 0.9 0.9, 0.9 0.8, 0.8 0.8))'

make_rectangle_3points

Creates a rectangle from 3 points.

Syntax	make_rectangle_3points(point1, point2, point3, [option=0]) [] marks optional arguments
Arguments	• **point1** - First point. • **point2** - Second point. • **point3** - Third point. • **option** - An optional argument to construct the rectangle. By default this value is 0. Value can be 0 (distance) or 1 (projected). Option distance: Second distance is equal to the distance between 2nd and 3rd point. Option projected: Second distance is equal to the distance of the perpendicular projection of the 3rd point on the segment or its extension.
Examples	• `geom_to_wkt(make_rectangle_3points(make_point(0, 0), make_point(0,5), make_point(5, 5), 0))` → 'Polygon ((0 0, 0 5, 5 5, 5 0, 0 0))' • `geom_to_wkt(make_rectangle_3points(make_point(0, 0), make_point(0,5), make_point(5, 3), 1))` → 'Polygon ((0 0, 0 5, 5 5, 5 0, 0 0))'

make_regular_polygon

Creates a regular polygon.

Syntax	make_regular_polygon(center, radius, number_sides, [circle=0]) [] marks optional arguments
Arguments	• **center** - center of the regular polygon • **radius** - second point. The first if the regular polygon is inscribed. The midpoint of the first side if the regular polygon is circumscribed. • **number_sides** - Number of sides/edges of the regular polygon • **circle** - Optional argument to construct the regular polygon. By default this value is 0. Value can be 0 (inscribed) or 1 (circumscribed)
Examples	• `geom_to_wkt(make_regular_polygon(make_point(0,0), make_point(0,5), 5))` → 'Polygon ((0 5, 4.76 1.55, 2.94 -4.05, -2.94 -4.05, -4.76 1.55, 0 5))' • `geom_to_wkt(make_regular_polygon(make_point(0,0), project(make_point(0,0), 4.0451, radians(36)), 5))` → 'Polygon ((0 5, 4.76 1.55, 2.94 -4.05, -2.94 -4.05, -4.76 1.55, 0 5))'

make_square

Creates a square from a diagonal.

Syntax	make_square(point1, point2)
Arguments	• **point1** - First point of the diagonal • **point2** - Last point of the diagonal
Examples	• `geom_to_wkt(make_square( make_point(0,0), make_point(5, 5)))` → 'Polygon ((0 0, -0 5, 5 5, 5 0, 0 0))' • `geom_to_wkt(make_square( make_point(5,0), make_point(5, 5)))` → 'Polygon ((5 0, 2.5 2.5, 5 5, 7.5 2.5, 5 0))'

make_triangle

Creates a triangle polygon.

Syntax	make_triangle(point1, point2, point3)
Arguments	• **point1** - first point of the triangle • **point2** - second point of the triangle • **point3** - third point of the triangle
Examples	• `geom_to_wkt(make_triangle(make_point(0,0), make_point(5, 5), make_point(0,10)))` → 'Triangle ((0 0, 5 5, 0 10, 0 0))' • `geom_to_wkt(boundary(make_triangle(make_point(0,0), make_point(5,5), make_point(0,10))))` → 'LineString (0 0, 5 5, 0 10, 0 0)'

make_valid

Returns a valid geometry or an empty geometry if the geometry could not be made valid.

Syntax	make_valid(geometry, [method=structure], [keep_collapsed=false]) [] marks optional arguments
Arguments	• **geometry** - a geometry • **method** - repair algorithm. May be either 'structure' or 'linework'. The 'linework' option combines all rings into a set of noded lines and then extracts valid polygons from that linework. The 'structure' method first makes all rings valid and then merges shells and subtracts holes from shells to generate valid result. Assumes that holes and shells are correctly categorized. • **keep_collapsed** - if set to true, then components that have collapsed into a lower dimensionality will be kept. For example, a ring collapsing to a line, or a line collapsing to a point.
Examples	• `geom_to_wkt(make_valid(geom_from_wkt('POLYGON((3 2, 4 1, 5 8, 3 2, 4 2))')))` → 'Polygon ((3 2, 5 8, 4 1, 3 2))' • `geom_to_wkt(make_valid(geom_from_wkt('POLYGON((3 2, 4 1, 5 8, 3 2, 4 2))'), 'linework'))` → 'GeometryCollection (Polygon ((5 8, 4 1, 3 2, 5 8)),LineString (3 2, 4 2))' • `geom_to_wkt(make_valid(geom_from_wkt('POLYGON((3 2, 4 1, 5 8))'), method:='linework'))` → 'Polygon ((3 2, 4 1, 5 8, 3 2))' • `make_valid(geom_from_wkt('LINESTRING(0 0)'))` → An empty geometry

Further reading: *is_valid*, *Fix geometries* algorithm

minimal_circle

Returns the minimal enclosing circle of a geometry. It represents the minimum circle that encloses all geometries within the set.

Syntax	minimal_circle(geometry, [segments=36]) [] marks optional arguments
Arguments	• **geometry** - a geometry • **segments** - optional argument for polygon segmentation. By default this value is 36
Examples	• `geom_to_wkt( minimal_circle( geom_from_wkt( 'LINESTRING(0 5, 0 -5, 2 1)' ), 4 ) )` → 'Polygon ((0 5, 5 -0, -0 -5, -5 0, 0 5))' • `geom_to_wkt( minimal_circle( geom_from_wkt( 'MULTIPOINT(1 2, 3 4, 3 2)' ), 4 ) )` → 'Polygon ((3 4, 3 2, 1 2, 1 4, 3 4))'

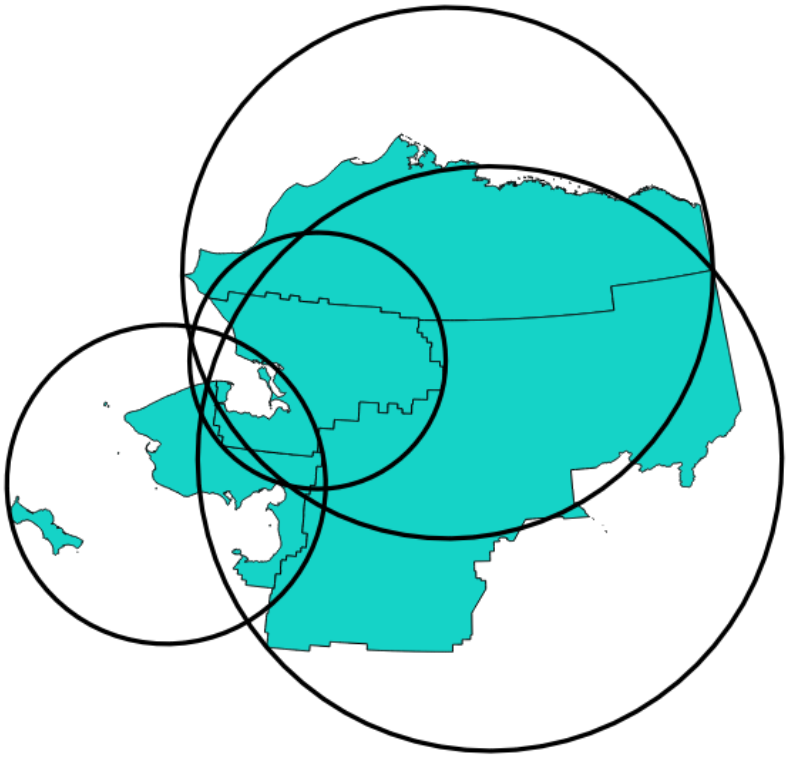


Fig. 9.22: Minimal enclosing circle of each feature

Further reading: *Minimum enclosing circles* algorithm

nodes_to_points

Returns a multipoint geometry consisting of every node in the input geometry.

Syntax	nodes_to_points(geometry, [ignore_closing_nodes=false]) [] marks optional arguments
Arguments	• **geometry** - geometry object • **ignore_closing_nodes** - optional argument specifying whether to include duplicate nodes which close lines or polygons rings. Defaults to false, set to true to avoid including these duplicate nodes in the output collection.
Examples	• `geom_to_wkt(nodes_to_points(geom_from_wkt('LINESTRING(0 0, 1 1, 2 2)')))` → 'MultiPoint ((0 0),(1 1),(2 2))' • `geom_to_wkt(nodes_to_points(geom_from_wkt('POLYGON((-1 -1, 4 0, 4 2, 0 2, -1 -1))'),true))` → 'MultiPoint ((-1 -1),(4 0),(4 2),(0 2))'

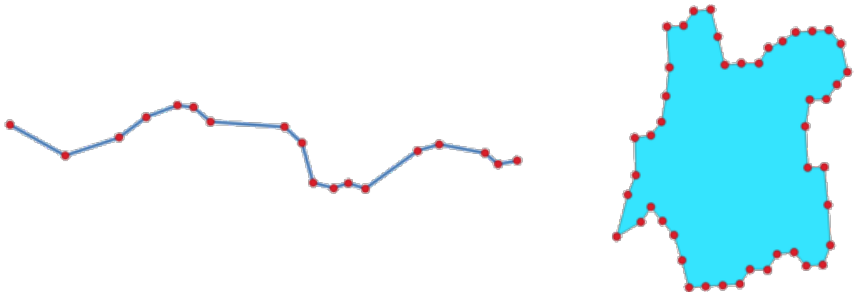


Fig. 9.23: Multi-point feature extracted from vertices

Further reading: *Extract vertices* algorithm

num_geometries

Returns the number of geometries in a geometry collection, or the number of parts in a multi-part geometry. The function returns NULL if the input geometry is not a collection.

Syntax	num_geometries(geometry)
Arguments	• **geometry** - geometry collection or multi-part geometry
Examples	• `num_geometries(geom_from_wkt('GEOMETRYCOLLECTION(POINT(0 1), POINT(0 0), POINT(1 0), POINT(1 1))'))` → 4 • `num_geometries(geom_from_wkt('MULTIPOINT((0 1), (0 0), (1 0))'))` → 3

num_interior_rings

Returns the number of interior rings in a polygon or geometry collection, or NULL if the input geometry is not a polygon or collection.

Syntax	num_interior_rings(geometry)
Arguments	• **geometry** - input geometry
Examples	• `num_interior_rings(geom_from_wkt('POLYGON((-1 -1, 4 0, 4 2, 0 2, -1 -1),(-0.1 -0.1, 0.4 0, 0.4 0.2, 0 0.2, -0.1 -0.1))'))` → 1

num_points

Returns the number of vertices in a geometry.

Syntax	num_points(geometry)
Arguments	• **geometry** - a geometry
Examples	• `num_points(@geometry)` → number of vertices in the current feature's geometry

num_rings

Returns the number of rings (including exterior rings) in a polygon or geometry collection, or NULL if the input geometry is not a polygon or collection.

Syntax	num_rings(geometry)
Arguments	• **geometry** - input geometry
Examples	• `num_rings(geom_from_wkt('POLYGON((-1 -1, 4 0, 4 2, 0 2, -1 -1),(-0.1 -0.1, 0.4 0, 0.4 0.2, 0 0.2, -0.1 -0.1))'))` → 2

offset_curve

Returns a geometry formed by offsetting a linestring geometry to the side. Distances are in the Spatial Reference System of this geometry.

Syntax	offset_curve(geometry, distance, [segments=8], [join=1], [miter_limit=2.0]) [] marks optional arguments
Arguments	• **geometry** - a (multi)linestring geometry • **distance** - offset distance. Positive values will be buffered to the left of lines, negative values to the right • **segments** - number of segments to use to represent a quarter circle when a round join style is used. A larger number results in a smoother line with more nodes. • **join** - join style for corners, where 1 = round, 2 = miter and 3 = bevel • **miter_limit** - limit on the miter ratio used for very sharp corners (when using miter joins only)
Examples	• `offset_curve(@geometry, 10.5)` → line offset to the left by 10.5 units • `offset_curve(@geometry, -10.5)` → line offset to the right by 10.5 units • `offset_curve(@geometry, 10.5, segments:=16, join:=1)` → line offset to the left by 10.5 units, using more segments to result in a smoother curve • `offset_curve(@geometry, 10.5, join:=3)` → line offset to the left by 10.5 units, using a beveled join

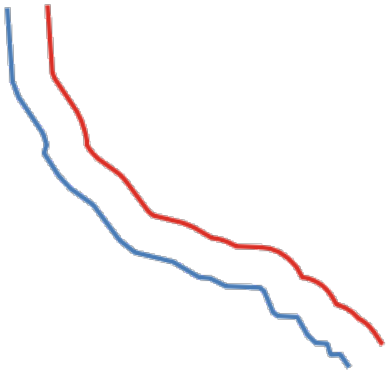


Fig. 9.24: In blue the source layer, in red the offset one

Further reading: *Offset lines* algorithm

order_parts

Orders the parts of a MultiGeometry by a given criteria

Syntax	order_parts(geometry, orderby, [ascending=true]) [] marks optional arguments
Arguments	• **geometry** - a multi-type geometry • **orderby** - an expression string defining the order criteria • **ascending** - boolean, True for ascending, False for descending
Examples	• `geom_to_wkt(order_parts(geom_from_wkt('MultiPolygon (((1 1, 5 1, 5 5, 1 5, 1 1)),((1 1, 9 1, 9 9, 1 9, 1 1)))'), 'area(@geometry)', False))` → 'MultiPolygon (((1 1, 9 1, 9 9, 1 9, 1 1)),((1 1, 5 1, 5 5, 1 5, 1 1)))' • `geom_to_wkt(order_parts(geom_from_wkt('LineString(1 2, 3 2, 4 3)'), '1', True))` → 'LineString(1 2, 3 2, 4 3)'

oriented_bbox

Returns a geometry which represents the minimal oriented bounding box of an input geometry.

Syntax	oriented_bbox(geometry)
Arguments	• **geometry** - a geometry
Examples	• `geom_to_wkt( oriented_bbox( geom_from_wkt( 'MULTIPOINT(1 2, 3 4, 3 2)' ) ) )` → 'Polygon ((3 2, 3 4, 1 4, 1 2, 3 2))'

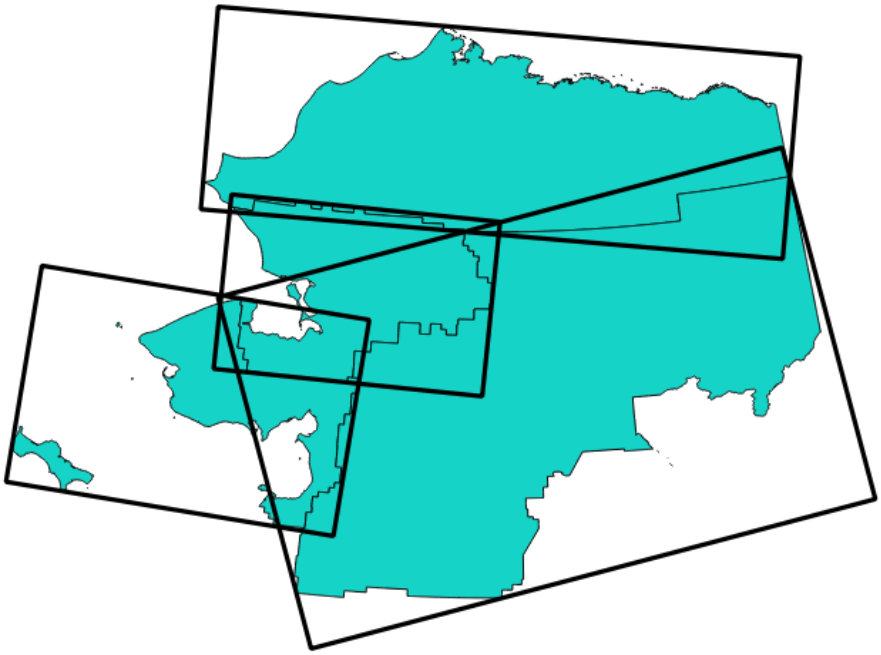


Fig. 9.25: Oriented minimum bounding box

Further reading: *Oriented minimum bounding box* algorithm

overlaps

Tests whether a geometry overlaps another. Returns TRUE if the geometries share space, are of the same dimension, but are not completely contained by each other.

Syntax	overlaps(geometry1, geometry2)
Arguments	• **geometry1** - a geometry • **geometry2** - a geometry
Examples	• `overlaps( geom_from_wkt( 'LINESTRING(3 5, 4 4, 5 5, 5 3)' ), geom_from_wkt( 'LINESTRING(3 3, 4 4, 5 5)' ) )` → TRUE • `overlaps( geom_from_wkt( 'LINESTRING(0 0, 1 1)' ), geom_from_wkt( 'LINESTRING(3 3, 4 4, 5 5)' ) )` → FALSE

overlay_contains

Returns whether the current feature spatially contains at least one feature from a target layer, or an array of expression-based results for the features in the target layer contained in the current feature.

Read more on the underlying GEOS “Contains” predicate, as described in PostGIS ST_Contains function.

Syntax	overlay_contains(layer, [expression], [filter], [limit], [cache=false]) [] marks optional arguments
Arguments	• **layer** - the layer whose overlay is checked • **expression** - an optional expression to evaluate on the features from the target layer. If not set, the function will just return a boolean indicating whether there is at least one match. • **filter** - an optional expression to filter the target features to check. If not set, all the features will be checked. • **limit** - an optional integer to limit the number of matching features. If not set, all the matching features will be returned. • **cache** - set this to true to build a local spatial index (most of the time, this is unwanted, unless you are working with a particularly slow data provider)
Examples	• `overlay_contains('regions')` → TRUE if the current feature spatially contains a region • `overlay_contains('regions', filter:= population > 10000)` → TRUE if the current feature spatially contains a region with a population greater than 10000 • `overlay_contains('regions', name)` → an array of names, for the regions contained in the current feature • `array_to_string(overlay_contains('regions', name))` → a string as a comma separated list of names, for the regions contained in the current feature • `array_sort(overlay_contains(layer:='regions', expression:="name", filter:= population > 10000))` → an ordered array of names, for the regions contained in the current feature and with a population greater than 10000 • `overlay_contains(layer:='regions', expression:= geom_to_wkt(@geometry), limit:=2)` → an array of geometries (in WKT), for up to two regions contained in the current feature

Further reading: *contains*, *array manipulation*, *Select by location* algorithm

overlay_crosses

Returns whether the current feature spatially crosses at least one feature from a target layer, or an array of expression-based results for the features in the target layer crossed by the current feature.

Read more on the underlying GEOS "Crosses" predicate, as described in PostGIS ST_Crosses function.

Syntax	overlay_crosses(layer, [expression], [filter], [limit], [cache=false]) [] marks optional arguments
Arguments	• **layer** - the layer whose overlay is checked • **expression** - an optional expression to evaluate on the features from the target layer. If not set, the function will just return a boolean indicating whether there is at least one match. • **filter** - an optional expression to filter the target features to check. If not set, all the features will be checked. • **limit** - an optional integer to limit the number of matching features. If not set, all the matching features will be returned. • **cache** - set this to true to build a local spatial index (most of the time, this is unwanted, unless you are working with a particularly slow data provider)
Examples	• `overlay_crosses('regions')` → TRUE if the current feature spatially crosses a region • `overlay_crosses('regions', filter:= population > 10000)` → TRUE if the current feature spatially crosses a region with a population greater than 10000 • `overlay_crosses('regions', name)` → an array of names, for the regions crossed by the current feature • `array_to_string(overlay_crosses('regions', name))` → a string as a comma separated list of names, for the regions crossed by the current feature • `array_sort(overlay_crosses(layer:='regions', expression:="name", filter:= population > 10000))` → an ordered array of names, for the regions crossed by the current feature and with a population greater than 10000 • `overlay_crosses(layer:='regions', expression:= geom_to_wkt(@geometry), limit:=2)` → an array of geometries (in WKT), for up to two regions crossed by the current feature

Further reading: *crosses*, *array manipulation*, *Select by location* algorithm

overlay_disjoint

Returns whether the current feature is spatially disjoint from all the features of a target layer, or an array of expression-based results for the features in the target layer that are disjoint from the current feature.

Read more on the underlying GEOS "Disjoint" predicate, as described in PostGIS ST_Disjoint function.

Syntax	overlay_disjoint(layer, [expression], [filter], [limit], [cache=false]) [] marks optional arguments
Arguments	• **layer** - the layer whose overlay is checked • **expression** - an optional expression to evaluate on the features from the target layer. If not set, the function will just return a boolean indicating whether there is at least one match. • **filter** - an optional expression to filter the target features to check. If not set, all the features will be checked. • **limit** - an optional integer to limit the number of matching features. If not set, all the matching features will be returned. • **cache** - set this to true to build a local spatial index (most of the time, this is unwanted, unless you are working with a particularly slow data provider)
Examples	• `overlay_disjoint('regions')` → TRUE if the current feature is spatially disjoint from all the regions • `overlay_disjoint('regions', filter:= population > 10000)` → TRUE if the current feature is spatially disjoint from all the regions with a population greater than 10000 • `overlay_disjoint('regions', name)` → an array of names, for the regions spatially disjoint from the current feature • `array_to_string(overlay_disjoint('regions', name))` → a string as a comma separated list of names, for the regions spatially disjoint from the current feature • `array_sort(overlay_disjoint(layer:='regions', expression:="name", filter:= population > 10000))` → an ordered array of names, for the regions spatially disjoint from the current feature and with a population greater than 10000 • `overlay_disjoint(layer:='regions', expression:= geom_to_wkt(@geometry), limit:=2)` → an array of geometries (in WKT), for up to two regions spatially disjoint from the current feature

Further reading: *disjoint*, *array manipulation*, *Select by location* algorithm

overlay_equals

Returns whether the current feature spatially equals to at least one feature from a target layer, or an array of expression-based results for the features in the target layer that are spatially equal to the current feature.

Read more on the underlying GEOS "Equals" predicate, as described in PostGIS ST_Equals function.

Syntax	overlay_equals(layer, [expression], [filter], [limit], [cache=false]) [] marks optional arguments
Arguments	• **layer** - the layer whose overlay is checked • **expression** - an optional expression to evaluate on the features from the target layer. If not set, the function will just return a boolean indicating whether there is at least one match. • **filter** - an optional expression to filter the target features to check. If not set, all the features will be checked. • **limit** - an optional integer to limit the number of matching features. If not set, all the matching features will be returned. • **cache** - set this to true to build a local spatial index (most of the time, this is unwanted, unless you are working with a particularly slow data provider)
Examples	• `overlay_equals('regions')` → TRUE if the current feature is spatially equal to a region • `overlay_equals('regions', filter:= population > 10000)` → TRUE if the current feature is spatially equal to a region with a population greater than 10000 • `overlay_equals('regions', name)` → an array of names, for the regions spatially equal to the current feature • `array_to_string(overlay_equals('regions', name))` → a string as a comma separated list of names, for the regions spatially equal to the current feature • `array_sort(overlay_equals(layer:='regions', expression:="name", filter:= population > 10000))` → an ordered array of names, for the regions spatially equal to the current feature and with a population greater than 10000 • `overlay_equals(layer:='regions', expression:= geom_to_wkt(@geometry), limit:=2)` → an array of geometries (in WKT), for up to two regions spatially equal to the current feature

Further reading: *array manipulation*, *Select by location* algorithm

overlay_intersects

Returns whether the current feature spatially intersects at least one feature from a target layer, or an array of expression-based results for the features in the target layer intersected by the current feature.

Read more on the underlying GEOS "Intersects" predicate, as described in PostGIS ST_Intersects function.

Syntax	overlay_intersects(layer, [expression], [filter], [limit], [cache=false], [min_overlap], [min_inscribed_circle_radius], [return_details], [sort_by_intersection_size]) [] marks optional arguments
Arguments	• **layer** - the layer whose overlay is checked • **expression** - an optional expression to evaluate on the features from the target layer. If not set, the function will just return a boolean indicating whether there is at least one match. • **filter** - an optional expression to filter the target features to check. If not set, all the features will be checked. • **limit** - an optional integer to limit the number of matching features. If not set, all the matching features will be returned. • **cache** - set this to true to build a local spatial index (most of the time, this is unwanted, unless you are working with a particularly slow data provider) • **min_overlap** - defines an optional exclusion filter: – for polygons, a minimum area in current feature squared units for the intersection. If the intersection results in multiple polygons the intersection will be returned if at least one of the polygons has an area greater or equal to the value – for lines, a minimum length in current feature units. If the intersection results in multiple lines the intersection will be returned if at least one of the lines has a length greater or equal to the value. • **min_inscribed_circle_radius** - defines an optional exclusion filter (for polygons only): minimum radius in current feature units for the maximum inscribed circle of the intersection. If the intersection results in multiple polygons the intersection will be returned if at least one of the polygons has a radius for the maximum inscribed circle greater or equal to the value. Read more on the underlying GEOS predicate, as described in PostGIS ST_MaximumInscribedCircle function. This argument requires GEOS >= 3.9. • **return_details** - Set this to true to return a list of maps containing (key names in quotes) the feature 'id', the expression 'result' and the 'overlap' value (of the largest element in case of multipart). The 'radius' of the maximum inscribed circle is also returned when the target layer is a polygon. Only valid when used with the expression parameter • **sort_by_intersection_size** - only valid when used with an expression, set this to 'des' to return the results ordered by the overlap value in descending order or set this to 'asc' for ascending order.
Examples	• `overlay_intersects('regions')` → TRUE if the current feature spatially intersects a region • `overlay_intersects('regions', filter:= population > 10000)` → TRUE if the current feature spatially intersects a region with a population greater than 10000 • `overlay_intersects('regions', name)` → an array of names, for the regions intersected by the current feature • `array_to_string(overlay_intersects('regions', name))` → a string as a comma separated list of names, for the regions intersected by the current feature • `array_sort(overlay_intersects(layer:='regions', expression:="name", filter:= population > 10000))` → an ordered array of names, for the regions intersected by the current feature and with a population greater than 10000 • `overlay_intersects(layer:='regions', expression:= geom_to_wkt(@geometry), limit:=2)` → an array of geometries (in WKT), for up to two regions intersected by the current feature • `overlay_intersects(layer:='regions', min_overlap:=0.54)` → TRUE if the current feature spatially intersects a region and the intersection area (of at least one of the parts in case of multipolygons) is greater or equal to 0.54 • `overlay_intersects(layer:='regions', min_inscribed_circle_radius:=0.54)` → TRUE if the current feature spatially intersects a region and the intersection area maximum inscribed circle's radius (of at least one of the parts in case of multipart) is greater or equal to 0.54 • `overlay_intersects(layer:='regions', expression:= geom_to_wkt(@geometry), return_details:=true)` → an array of maps containing 'id', 'result', 'overlap' and 'radius'

Further reading: *intersects*, *array manipulation*, *Select by location* algorithm

overlay_nearest

Returns whether the current feature has feature(s) from a target layer within a given distance, or an array of expression-based results for the features in the target layer within a distance from the current feature.

Note: This function can be slow and consume a lot of memory for large layers.

Syntax	overlay_nearest(layer, [expression], [filter], [limit=1], [max_distance], [cache=false]) [] marks optional arguments
Arguments	• **layer** - the target layer • **expression** - an optional expression to evaluate on the features from the target layer. If not set, the function will just return a boolean indicating whether there is at least one match. • **filter** - an optional expression to filter the target features to check. If not set, all the features in the target layer will be used. • **limit** - an optional integer to limit the number of matching features. If not set, only the nearest feature will be returned. If set to -1, returns all the matching features. • **max_distance** - an optional distance to limit the search of matching features. If not set, all the features in the target layer will be used. • **cache** - set this to true to build a local spatial index (most of the time, this is unwanted, unless you are working with a particularly slow data provider)
Examples	• `overlay_nearest('airports')` → TRUE if the "airports" layer has at least one feature • `overlay_nearest('airports', max_distance:= 5000)` → TRUE if there is an airport within a distance of 5000 map units from the current feature • `overlay_nearest('airports', name)` → the name of the closest airport to the current feature, as an array • `array_to_string(overlay_nearest('airports', name))` → the name of the closest airport to the current feature, as a string • `overlay_nearest(layer:='airports', expression:= name, max_distance:= 5000)` → the name of the closest airport within a distance of 5000 map units from the current feature, as an array • `overlay_nearest(layer:='airports', expression:="name", filter:= "Use"='Civilian', limit:=3)` → an array of names, for up to the three closest civilian airports ordered by distance • `overlay_nearest(layer:='airports', expression:="name", limit:= -1, max_distance:= 5000)` → an array of names, for all the airports within a distance of 5000 map units from the current feature, ordered by distance

Further reading: *array manipulation*, *Join attributes by nearest* algorithm

overlay_touches

Returns whether the current feature spatially touches at least one feature from a target layer, or an array of expression-based results for the features in the target layer touched by the current feature.

Read more on the underlying GEOS "Touches" predicate, as described in PostGIS ST_Touches function.

Syntax	overlay_touches(layer, [expression], [filter], [limit], [cache=false]) [] marks optional arguments
Arguments	• **layer** - the layer whose overlay is checked • **expression** - an optional expression to evaluate on the features from the target layer. If not set, the function will just return a boolean indicating whether there is at least one match. • **filter** - an optional expression to filter the target features to check. If not set, all the features will be checked. • **limit** - an optional integer to limit the number of matching features. If not set, all the matching features will be returned. • **cache** - set this to true to build a local spatial index (most of the time, this is unwanted, unless you are working with a particularly slow data provider)
Examples	• `overlay_touches('regions')` → TRUE if the current feature spatially touches a region • `overlay_touches('regions', filter:= population > 10000)` → TRUE if the current feature spatially touches a region with a population greater than 10000 • `overlay_touches('regions', name)` → an array of names, for the regions touched by the current feature • `string_to_array(overlay_touches('regions', name))` → a string as a comma separated list of names, for the regions touched by the current feature • `array_sort(overlay_touches(layer:='regions', expression:="name", filter:= population > 10000))` → an ordered array of names, for the regions touched by the current feature and with a population greater than 10000 • `overlay_touches(layer:='regions', expression:= geom_to_wkt(@geometry), limit:=2)` → an array of geometries (in WKT), for up to two regions touched by the current feature

Further reading: *touches*, *array manipulation*, *Select by location* algorithm

overlay_within

Returns whether the current feature is spatially within at least one feature from a target layer, or an array of expression-based results for the features in the target layer that contain the current feature.

Read more on the underlying GEOS "Within" predicate, as described in PostGIS ST_Within function.

Syntax	overlay_within(layer, [expression], [filter], [limit], [cache=false]) [] marks optional arguments
Arguments	• **layer** - the layer whose overlay is checked • **expression** - an optional expression to evaluate on the features from the target layer. If not set, the function will just return a boolean indicating whether there is at least one match. • **filter** - an optional expression to filter the target features to check. If not set, all the features will be checked. • **limit** - an optional integer to limit the number of matching features. If not set, all the matching features will be returned. • **cache** - set this to true to build a local spatial index (most of the time, this is unwanted, unless you are working with a particularly slow data provider)
Examples	• `overlay_within('regions')` → TRUE if the current feature is spatially within a region • `overlay_within('regions', filter:= population > 10000)` → TRUE if the current feature is spatially within a region with a population greater than 10000 • `overlay_within('regions', name)` → an array of names, for the regions containing the current feature • `array_to_string(overlay_within('regions', name))` → a string as a comma separated list of names, for the regions containing the current feature • `array_sort(overlay_within(layer:='regions', expression:="name", filter:= population > 10000))` → an ordered array of names, for the regions containing the current feature and with a population greater than 10000 • `overlay_within(layer:='regions', expression:= geom_to_wkt(@geometry), limit:=2)` → an array of geometries (in WKT), for up to two regions containing the current feature

Further reading: *within*, *array manipulation*, *Select by location* algorithm

$perimeter

Returns the perimeter length of the current feature. The perimeter calculated by this function respects both the current project's ellipsoid setting and distance unit settings. For example, if an ellipsoid has been set for the project then the calculated perimeter will be ellipsoidal, and if no ellipsoid is set then the calculated perimeter will be planimetric.

Syntax	$perimeter
Examples	• `$perimeter` → 42

perimeter

Returns the perimeter of a geometry polygon object. Calculations are always planimetric in the Spatial Reference System (SRS) of this geometry, and the units of the returned perimeter will match the units for the SRS. This differs from the calculations performed by the $perimeter function, which will perform ellipsoidal calculations based on the project's ellipsoid and distance unit settings.

Syntax	perimeter(geometry)
Arguments	• **geometry** - polygon geometry object
Examples	• `perimeter(@geometry)` → perimeter of the current feature's geometry • `perimeter(geom_from_wkt('POLYGON((0 0, 4 0, 4 2, 0 2, 0 0))'))` → 12.0

point_n

Returns a specific node from a geometry.

Syntax	point_n(geometry, index)
Arguments	• **geometry** - geometry object • **index** - index of node to return, where 1 is the first node; if the value is negative, the selected vertex index will be its total count minus the absolute value
Examples	• `geom_to_wkt(point_n(geom_from_wkt('POLYGON((0 0, 4 0, 4 2, 0 2, 0 0))'),2))` → 'Point (4 0)'

Further reading: *Extract specific vertices* algorithm

point_on_surface

Returns a point guaranteed to lie on the surface of a geometry.

Syntax	point_on_surface(geometry)
Arguments	• **geometry** - a geometry
Examples	• `point_on_surface(@geometry)` → a point geometry

Further reading: *Point on Surface* algorithm

pole_of_inaccessibility

Calculates the approximate pole of inaccessibility for a surface, which is the most distant internal point from the boundary of the surface. This function uses the 'polylabel' algorithm (Vladimir Agafonkin, 2016), which is an iterative approach guaranteed to find the true pole of inaccessibility within a specified tolerance. More precise tolerances require more iterations and will take longer to calculate.

Syntax	pole_of_inaccessibility(geometry, tolerance)
Arguments	• **geometry** - a geometry • **tolerance** - maximum distance between the returned point and the true pole location
Examples	• `geom_to_wkt(pole_of_inaccessibility( geom_from_wkt('POLYGON((0 1, 0 9, 3 10, 3 3, 10 3, 10 1, 0 1))'), 0.1))` → 'Point(1.546875 2.546875)'



Fig. 9.26: Pole of inaccessibility

Further reading: *Pole of inaccessibility* algorithm

project

Returns a point projected from a start point using a distance, a bearing (azimuth) and an elevation in radians.

Syntax	project(point, distance, azimuth, [elevation]) [] marks optional arguments
Arguments	• **point** - start point • **distance** - distance to project • **azimuth** - azimuth in radians clockwise, where 0 corresponds to north • **elevation** - angle of inclination in radians
Examples	• `geom_to_wkt(project(make_point(1, 2), 3, radians(270)))` → 'Point(-2, 2)'

Further reading: *Project points (Cartesian)* algorithm

relate

Tests the Dimensional Extended 9 Intersection Model (DE-9IM) representation of the relationship between two geometries.

Relationship variant

Returns the Dimensional Extended 9 Intersection Model (DE-9IM) representation of the relationship between two geometries.

Syntax	relate(geometry, geometry)
Arguments	• **geometry** - a geometry • **geometry** - a geometry
Examples	• `relate( geom_from_wkt( 'LINESTRING(40 40,120 120)' ), geom_from_wkt( 'LINESTRING(40 40,60 120)' ) )` → 'FF1F00102'

Pattern match variant

Tests whether the DE-9IM relationship between two geometries matches a specified pattern.

Syntax	relate(geometry, geometry, pattern)
Arguments	• **geometry** - a geometry • **geometry** - a geometry • **pattern** - DE-9IM pattern to match
Examples	• `relate( geom_from_wkt( 'LINESTRING(40 40,120 120)' ), geom_from_wkt( 'LINESTRING(40 40,60 120)' ), '**1F001**' )` → TRUE

reverse

Reverses the direction of a line string or reverses a string of text.

String variant

Reverses the order of characters in a string.

Syntax	reverse(string)
Arguments	• **string** - string to reverse
Examples	• `reverse('hello')` → 'olleh'

Geometry variant

Reverses the direction of a line string by reversing the order of its vertices.

Syntax	reverse(geometry)
Arguments	• **geometry** - a geometry
Examples	• `geom_to_wkt(reverse(geom_from_wkt('LINESTRING(0 0, 1 1, 2 2)')))` → 'LINESTRING(2 2, 1 1, 0 0)'

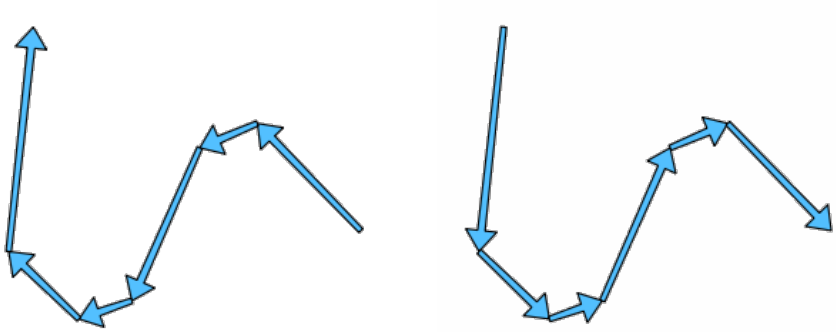


Fig. 9.27: Reversing line direction

Further reading: *Reverse line direction* algorithm

rotate

Returns a rotated version of a geometry. Calculations are in the Spatial Reference System of this geometry.

Syntax	rotate(geometry, rotation, [center=NULL], [per_part=false]) [] marks optional arguments
Arguments	• **geometry** - a geometry • **rotation** - clockwise rotation in degrees • **center** - rotation center point. If not specified, the center of the geometry's bounding box is used. • **per_part** - apply rotation per part. If true, then rotation will apply around the center of each part's bounding box when the input geometry is multipart and an explicit rotation center point is not specified.
Examples	• `rotate(@geometry, 45, make_point(4, 5))` → geometry rotated 45 degrees clockwise around the (4, 5) point • `rotate(@geometry, 45)` → geometry rotated 45 degrees clockwise around the center of its bounding box

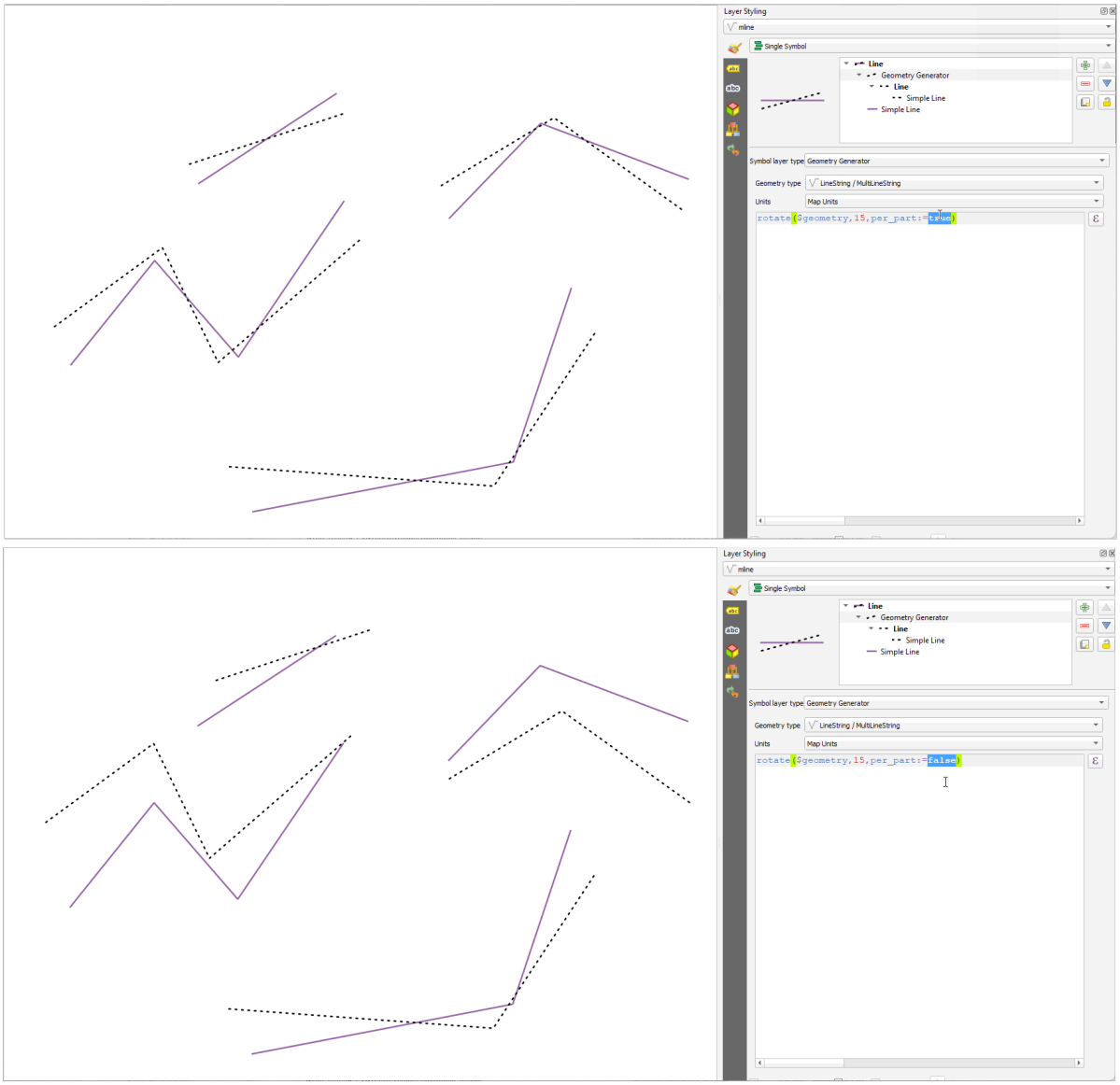


Fig. 9.28: Rotating features

roundness

Calculates how close a polygon shape is to a circle. The function returns 1 when the polygon shape is a perfect circle and 0 when it is completely flat.

Syntax	roundness(geometry)
Arguments	• **geometry** - a polygon
Examples	• `round(roundness(geom_from_wkt('POLYGON(( 0 0, 0 1, 1 1, 1 0, 0 0))')), 3)` → 0.785 • `round(roundness(geom_from_wkt('POLYGON(( 0 0, 0 0.1, 1 0.1, 1 0, 0 0))')), 3)` → 0.260

Further reading: *Roundness* algorithm

scale

Returns a scaled version of a geometry. Calculations are in the Spatial Reference System of this geometry.

Syntax	scale(geometry, x_scale, y_scale, [center]) [] marks optional arguments
Arguments	• **geometry** - a geometry • **x_scale** - x-axis scaling factor • **y_scale** - y-axis scaling factor • **center** - scaling center point. If not specified, the center of the geometry's bounding box is used.
Examples	• `scale(@geometry, 2, 0.5, make_point(4, 5))` → geometry scaled twice horizontally and halved vertically, around the (4, 5) point • `scale(@geometry, 2, 0.5)` → geometry twice horizontally and halved vertically, around the center of its bounding box

segments_to_lines

Returns a multi line geometry consisting of a line for every segment in the input geometry.

Syntax	segments_to_lines(geometry)
Arguments	• **geometry** - geometry object
Examples	• `geom_to_wkt(segments_to_lines(geom_from_wkt('LINESTRING(0 0, 1 1, 2 2)')))` → 'MultiLineString ((0 0, 1 1),(1 1, 2 2))'

Further reading: *Explode lines* algorithm

shared_paths

Returns a collection containing paths shared by the two input geometries. Those going in the same direction are in the first element of the collection, those going in the opposite direction are in the second element. The paths themselves are given in the direction of the first geometry.

Syntax	shared_paths(geometry1, geometry2)
Arguments	• **geometry1** - a LineString/MultiLineString geometry • **geometry2** - a LineString/MultiLineString geometry
Examples	• `geom_to_wkt(shared_paths(geom_from_wkt('MULTILINESTRING((26 125,26 200,126 200,126 125,26 125),(51 150,101 150,76 175,51 150))'),geom_from_wkt('LINESTRING(151 100,126 156.25,126 125,90 161, 76 175)'))` → 'GeometryCollection (MultiLineString ((126 156.25, 126 125),(101 150, 90 161),(90 161, 76 175)),MultiLineString EMPTY)' • `geom_to_wkt(shared_paths(geom_from_wkt('LINESTRING(76 175,90 161,126 125,126 156.25,151 100)'), geom_from_wkt('MULTILINESTRING((26 125,26 200,126 200,126 125,26 125),(51 150,101 150,76 175,51 150))')))` → 'GeometryCollection (MultiLineString EMPTY,MultiLineString ((76 175, 90 161),(90 161, 101 150),(126 125, 126 156.25)))'

shortest_line

Returns the shortest line joining geometry1 to geometry2. The resultant line will start at geometry1 and end at geometry2.

Syntax	shortest_line(geometry1, geometry2)
Arguments	• **geometry1** - geometry to find shortest line from • **geometry2** - geometry to find shortest line to
Examples	• `geom_to_wkt(shortest_line(geom_from_wkt('LINESTRING (20 80, 98 190, 110 180, 50 75 )'),geom_from_wkt('POINT(100 100)')))` → 'LineString(73.0769 115.384, 100 100)'

simplify

Simplifies a geometry by removing nodes using a distance based threshold (ie, the Douglas Peucker algorithm). The algorithm preserves large deviations in geometries and reduces the number of vertices in nearly straight segments.

Syntax	simplify(geometry, tolerance)
Arguments	• **geometry** - a geometry • **tolerance** - maximum deviation from straight segments for points to be removed
Examples	• `geom_to_wkt(simplify(geometry:=geom_from_wkt('LineString(0 0, 5 0.1, 10 0)'),tolerance:=5))` → 'LineString(0 0, 10 0)'



Fig. 9.29: From left to right, source layer and increasing simplification tolerances

Further reading: *Simplify* algorithm

simplify_vw

Simplifies a geometry by removing nodes using an area based threshold (ie, the Visvalingam-Whyatt algorithm). The algorithm removes vertices which create small areas in geometries, e.g., narrow spikes or nearly straight segments.

Syntax	simplify_vw(geometry, tolerance)
Arguments	• **geometry** - a geometry • **tolerance** - a measure of the maximum area created by a node for the node to be removed
Examples	• `geom_to_wkt(simplify_vw(geometry:=geom_from_wkt('LineString(0 0, 5 0, 5.01 10, 5.02 0, 10 0)'),tolerance:=5))` → 'LineString(0 0, 10 0)'

Further reading: *Simplify* algorithm

single_sided_buffer

Returns a geometry formed by buffering out just one side of a linestring geometry. Distances are in the Spatial Reference System of this geometry.

Syntax	single_sided_buffer(geometry, distance, [segments=8], [join=1], [miter_limit=2.0]) [] marks optional arguments
Arguments	• **geometry** - a (multi)linestring geometry • **distance** - buffer distance. Positive values will be buffered to the left of lines, negative values to the right • **segments** - number of segments to use to represent a quarter circle when a round join style is used. A larger number results in a smoother buffer with more nodes. • **join** - join style for corners, where 1 = round, 2 = miter and 3 = bevel • **miter_limit** - limit on the miter ratio used for very sharp corners (when using miter joins only)
Examples	• `single_sided_buffer(@geometry, 10.5)` → line buffered to the left by 10.5 units • `single_sided_buffer(@geometry, -10.5)` → line buffered to the right by 10.5 units • `single_sided_buffer(@geometry, 10.5, segments:=16, join:=1)` → line buffered to the left by 10.5 units, using more segments to result in a smoother buffer • `single_sided_buffer(@geometry, 10.5, join:=3)` → line buffered to the left by 10.5 units, using a beveled join



Fig. 9.30: Left versus right side buffer on the same vector line layer

Further reading: *Single sided buffer* algorithm

sinuosity

Returns the sinuosity of a curve, which is the ratio of the curve length to the straight (2D) distance between its endpoints.

Syntax	sinuosity(geometry)
Arguments	• **geometry** - Input curve (circularstring, linestring)
Examples	• `round(sinuosity(geom_from_wkt('LINESTRING(2 0, 2 2, 3 2, 3 3)')), 3)` → 1.265 • `sinuosity(geom_from_wkt('LINESTRING( 3 1, 5 1)'))` → 1.0

smooth

Smooths a geometry by adding extra nodes which round off corners in the geometry. If input geometries contain Z or M values, these will also be smoothed and the output geometry will retain the same dimensionality as the input geometry.

Syntax	smooth(geometry, [iterations=1], [offset=0.25], [min_length=-1], [max_angle=180]) [] marks optional arguments
Arguments	• **geometry** - a geometry • **iterations** - number of smoothing iterations to apply. Larger numbers result in smoother but more complex geometries. • **offset** - value between 0 and 0.5 which controls how tightly the smoothed geometry follow the original geometry. Smaller values result in a tighter smoothing, larger values result in looser smoothing. • **min_length** - minimum length of segments to apply smoothing to. This parameter can be used to avoid placing excessive additional nodes in shorter segments of the geometry. • **max_angle** - maximum angle at node for smoothing to be applied (0-180). By lowering the maximum angle intentionally sharp corners in the geometry can be preserved. For instance, a value of 80 degrees will retain right angles in the geometry.
Examples	• `geom_to_wkt(smooth(geometry:=geom_from_wkt('LineString(0 0, 5 0, 5 5)'),iterations:=1,offset:=0.2,min_length:=-1, max_angle:=180))` → 'LineString (0 0, 4 0, 5 1, 5 5)'

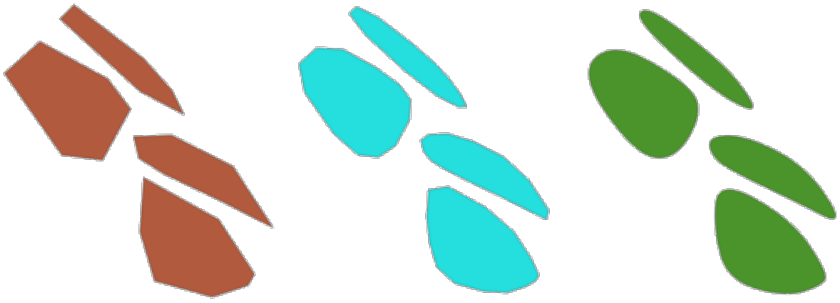


Fig. 9.31: Increasing number of iterations causes smoother geometries

Further reading: *Smooth* algorithm

square_wave

Constructs square/rectangular waves along the boundary of a geometry.

Syntax	square_wave(geometry, wavelength, amplitude, [strict=False]) [] marks optional arguments
Arguments	• **geometry** - a geometry • **wavelength** - wavelength of square waveform • **amplitude** - amplitude of square waveform • **strict** - By default the wavelength argument is treated as a "maximum wavelength", where the actual wavelength will be dynamically adjusted so that an exact number of square waves are created along the boundaries of the geometry. If the strict argument is set to true then the wavelength will be used exactly and an incomplete pattern may be used for the final waveform.
Examples	• `square_wave(geom_from_wkt('LineString(0 0, 10 0)'), 3, 1)` → Square waves with wavelength 3 and amplitude 1 along the linestring

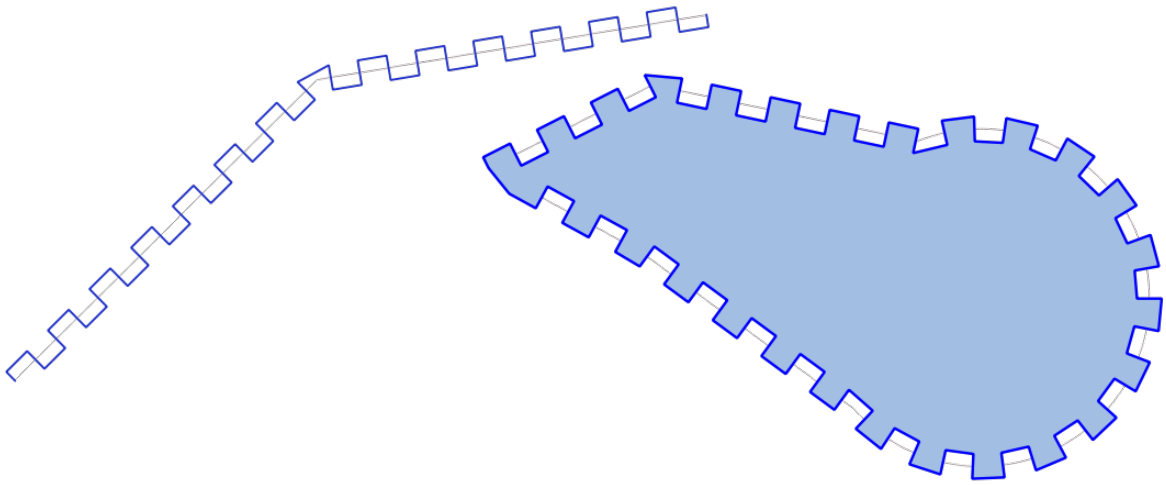


Fig. 9.32: Symbolizing features with square waves

square_wave_randomized

Constructs randomized square/rectangular waves along the boundary of a geometry.

Syntax	square_wave_randomized(geometry, min_wavelength, max_wavelength, min_amplitude, max_amplitude, [seed=0]) [] marks optional arguments
Arguments	• **geometry** - a geometry • **min_wavelength** - minimum wavelength of waves • **max_wavelength** - maximum wavelength of waves • **min_amplitude** - minimum amplitude of waves • **max_amplitude** - maximum amplitude of waves • **seed** - specifies a random seed for generating waves. If the seed is 0, then a completely random set of waves will be generated.
Examples	• `square_wave_randomized(geom_from_wkt('LineString(0 0, 10 0)'), 2, 3, 0.1, 0.2)` → Randomly sized square waves with wavelengths between 2 and 3 and amplitudes between 0.1 and 0.2 along the linestring

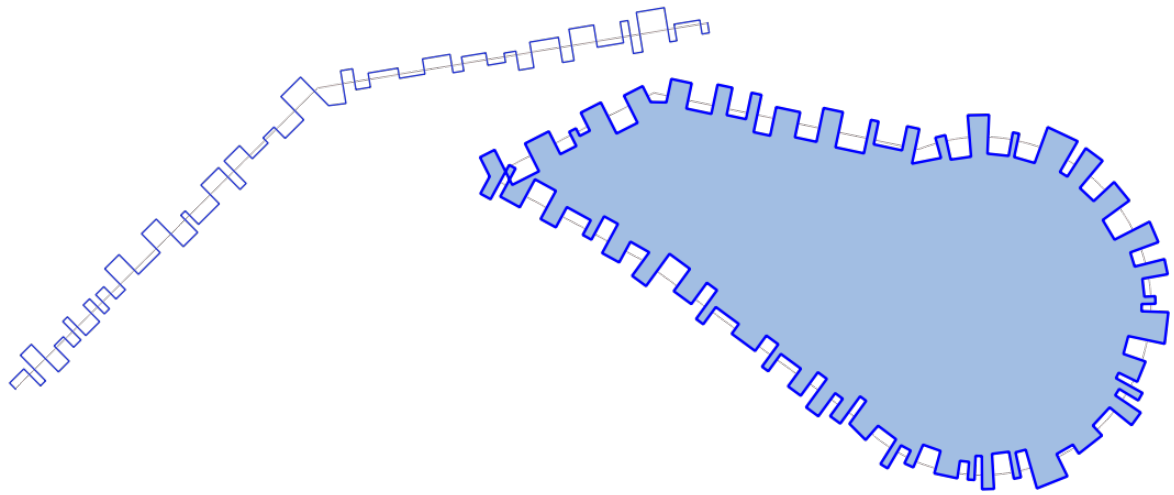


Fig. 9.33: Symbolizing features with square randomized waves

start_point

Returns the first node from a geometry.

Syntax	start_point(geometry)
Arguments	• **geometry** - geometry object
Examples	• `geom_to_wkt(start_point(geom_from_wkt('LINESTRING(4 0, 4 2, 0 2)')))` → 'Point (4 0)'

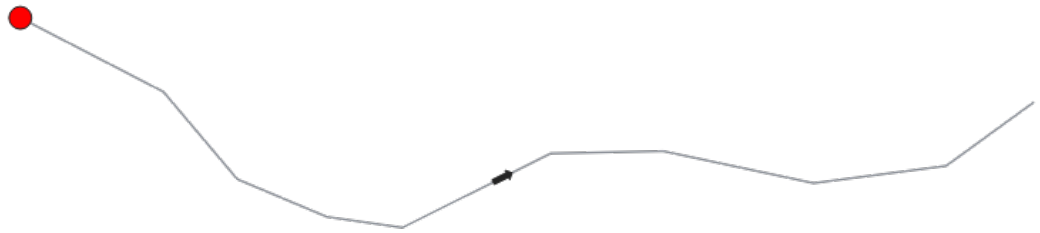


Fig. 9.34: Starting point of a line feature

Further reading: *end_point*, *Extract specific vertices* algorithm

straight_distance_2d

Returns the direct/euclidean distance between the first and last vertex of a geometry. The geometry must be a curve (circularstring, linestring).

Syntax	straight_distance_2d(geometry)
Arguments	• **geometry** - The geometry.
Examples	• `straight_distance_2d(geom_from_wkt('LINESTRING(1 0, 1 1)'))` → 1 • `round(straight_distance_2d(geom_from_wkt('LINESTRING(1 4, 3 5, 5 0)')), 3)` → 5.657

Further reading: *length*

sym_difference

Returns a geometry that represents the portions of two geometries that do not intersect.

Syntax	sym_difference(geometry1, geometry2)
Arguments	• **geometry1** - a geometry • **geometry2** - a geometry
Examples	• `geom_to_wkt( sym_difference( geom_from_wkt( 'LINESTRING(3 3, 4 4, 5 5)' ), geom_from_wkt( 'LINESTRING(3 3, 8 8)' ) ) )` → 'LINESTRING(5 5, 8 8)'

Further reading: *Symmetrical difference* algorithm

tapered_buffer

Creates a buffer along a line geometry where the buffer diameter varies evenly over the length of the line.

Syntax	tapered_buffer(geometry, start_width, end_width, [segments=8]) [] marks optional arguments
Arguments	• **geometry** - input geometry. Must be a (multi)line geometry. • **start_width** - width of buffer at start of line, • **end_width** - width of buffer at end of line. • **segments** - number of segments to approximate quarter-circle curves in the buffer.
Examples	• `tapered_buffer(geometry:=geom_from_wkt('LINESTRING(1 2, 4 2)'),start_width:=1,end_width:=2,segments:=8)` → A tapered buffer starting with a diameter of 1 and ending with a diameter of 2 along the linestring geometry.

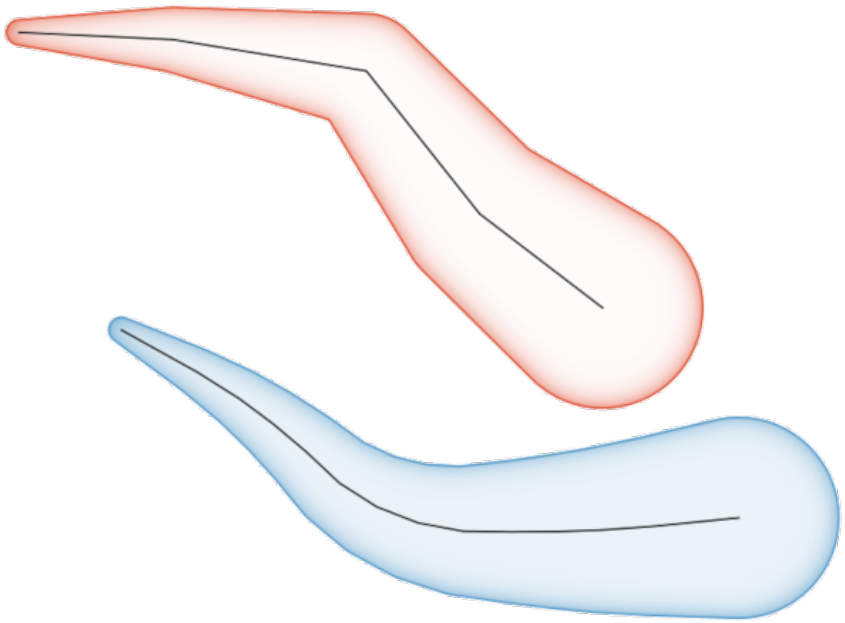


Fig. 9.35: Tapered buffer on line features

Further reading: *Tapered buffers* algorithm

touches

Tests whether a geometry touches another. Returns TRUE if the geometries have at least one point in common, but their interiors do not intersect.

Syntax	touches(geometry1, geometry2)
Arguments	• **geometry1** - a geometry • **geometry2** - a geometry
Examples	• `touches( geom_from_wkt( 'LINESTRING(5 3, 4 4)' ), geom_from_wkt( 'LINESTRING(3 3, 4 4, 5 5)' ) )` → TRUE • `touches( geom_from_wkt( 'POINT(4 4)' ), geom_from_wkt( 'POINT(5 5)' ) )` → FALSE

Further reading: *overlay_touches*

transform

Returns the geometry transformed from a source CRS to a destination CRS.

Syntax	transform(geometry, source_auth_id, dest_auth_id)
Arguments	• **geometry** - a geometry • **source_auth_id** - the source CRS definition or CRS object • **dest_auth_id** - the destination CRS definition or CRS object
Examples	• `geom_to_wkt( transform( make_point(488995.53240249, 7104473.38600835), 'EPSG:2154', 'EPSG:4326' ) )` → 'POINT(0 51)'

Further reading: *Reproject layer* algorithm

translate

Returns a translated version of a geometry. Calculations are in the Spatial Reference System of this geometry.

Syntax	translate(geometry, dx, dy)
Arguments	• **geometry** - a geometry • **dx** - delta x • **dy** - delta y
Examples	• `translate(@geometry, 5, 10)` → a geometry of the same type like the original one

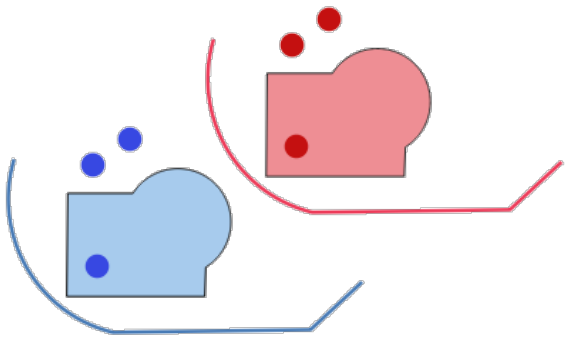


Fig. 9.36: Translating features

Further reading: *Translate* algorithm

triangular_wave

Constructs triangular waves along the boundary of a geometry.

Syntax	triangular_wave(geometry, wavelength, amplitude, [strict=False]) [] marks optional arguments
Arguments	• **geometry** - a geometry • **wavelength** - wavelength of triangular waveform • **amplitude** - amplitude of triangular waveform • **strict** - By default the wavelength argument is treated as a "maximum wavelength", where the actual wavelength will be dynamically adjusted so that an exact number of triangular waves are created along the boundaries of the geometry. If the strict argument is set to true then the wavelength will be used exactly and an incomplete pattern may be used for the final waveform.
Examples	• `triangular_wave(geom_from_wkt('LineString(0 0, 10 0)'), 3, 1)` → Triangular waves with wavelength 3 and amplitude 1 along the linestring

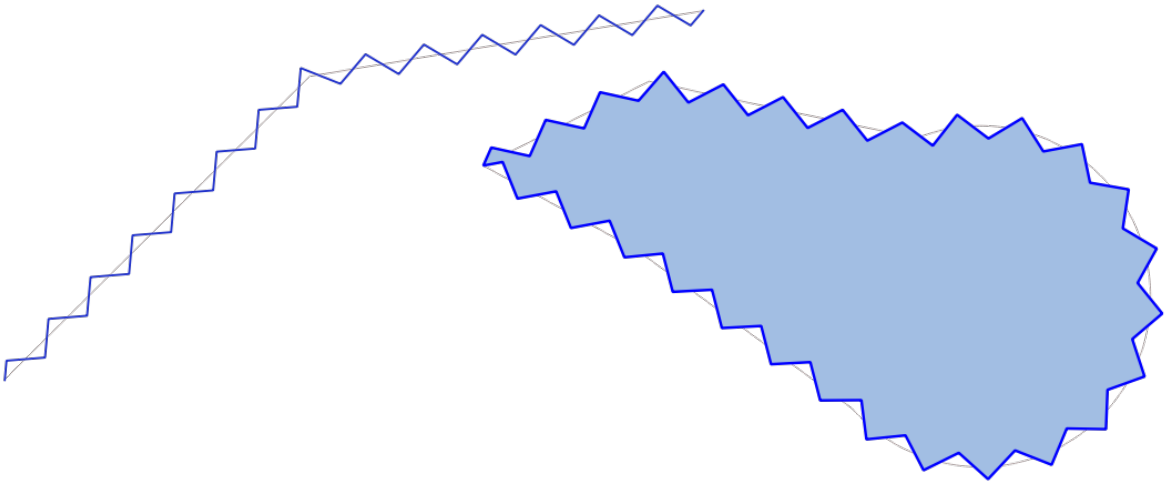


Fig. 9.37: Symbolizing features with triangular waves

triangular_wave_randomized

Constructs randomized triangular waves along the boundary of a geometry.

Syntax	triangular_wave_randomized(geometry, min_wavelength, max_wavelength, min_amplitude, max_amplitude, [seed=0]) [] marks optional arguments
Arguments	• **geometry** - a geometry • **min_wavelength** - minimum wavelength of waves • **max_wavelength** - maximum wavelength of waves • **min_amplitude** - minimum amplitude of waves • **max_amplitude** - maximum amplitude of waves • **seed** - specifies a random seed for generating waves. If the seed is 0, then a completely random set of waves will be generated.
Examples	• `triangular_wave_randomized(geom_from_wkt('LineString(0 0, 10 0)'), 2, 3, 0.1, 0.2)` → Randomly sized triangular waves with wavelengths between 2 and 3 and amplitudes between 0.1 and 0.2 along the linestring

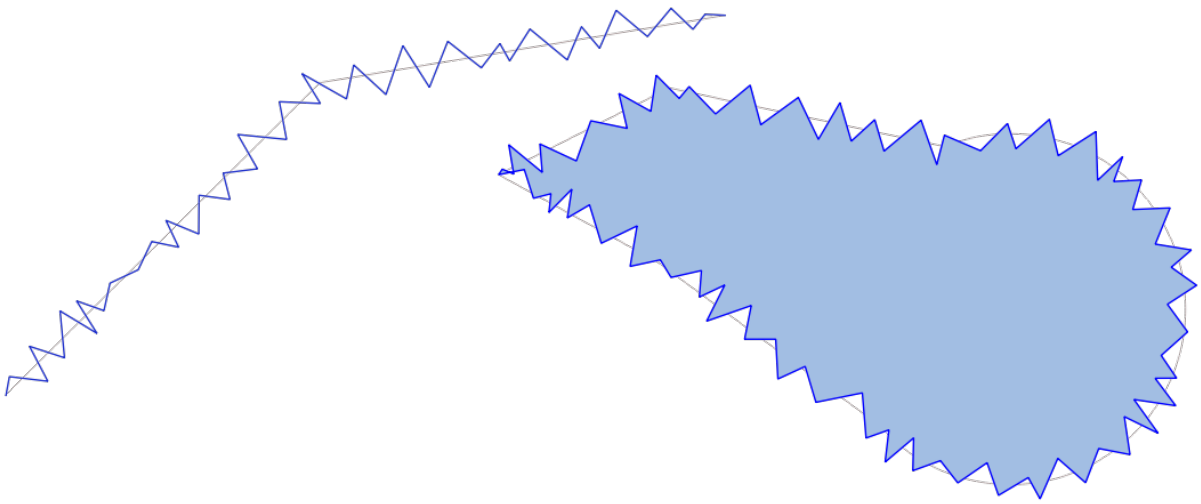


Fig. 9.38: Symbolizing features with triangular randomized waves

union

Returns a geometry that represents the point set union of the geometries.

Syntax	union(geometry1, geometry2)
Arguments	• **geometry1** - a geometry • **geometry2** - a geometry
Examples	• `geom_to_wkt( union( make_point(4, 4), make_point(5, 5) ) )` → 'MULTIPOINT(4 4, 5 5)'

wave

Constructs rounded (sine-like) waves along the boundary of a geometry.

Syntax	wave(geometry, wavelength, amplitude, [strict=False]) [] marks optional arguments
Arguments	• **geometry** - a geometry • **wavelength** - wavelength of sine-like waveform • **amplitude** - amplitude of sine-like waveform • **strict** - By default the wavelength argument is treated as a "maximum wavelength", where the actual wavelength will be dynamically adjusted so that an exact number of waves are created along the boundaries of the geometry. If the strict argument is set to true then the wavelength will be used exactly and an incomplete pattern may be used for the final waveform.
Examples	• `wave(geom_from_wkt('LineString(0 0, 10 0)'), 3, 1)` → Sine-like waves with wavelength 3 and amplitude 1 along the linestring

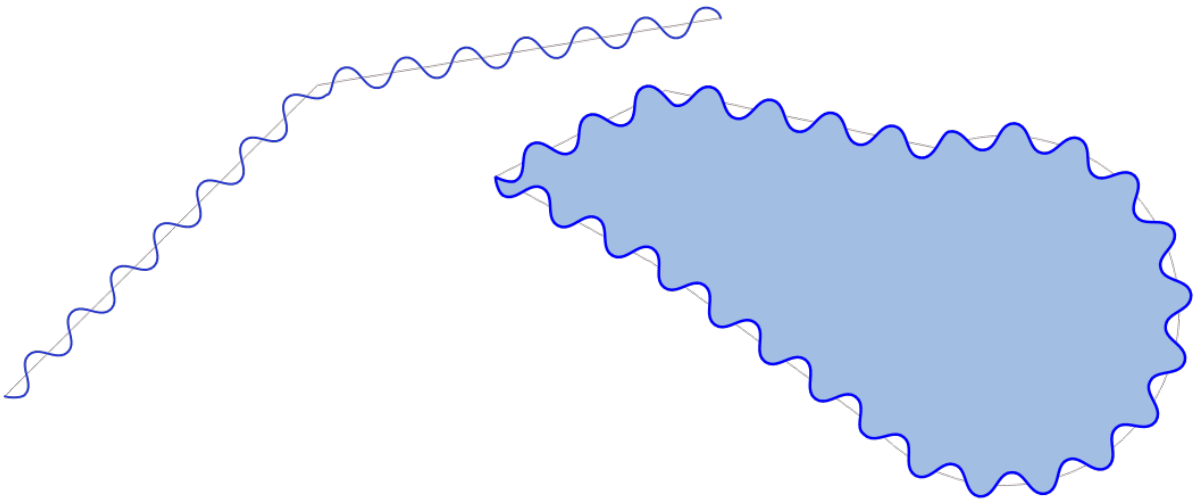


Fig. 9.39: Symbolizing features with waves

wave_randomized

Constructs randomized curved (sine-like) waves along the boundary of a geometry.

Syntax	wave_randomized(geometry, min_wavelength, max_wavelength, min_amplitude, max_amplitude, [seed=0]) [] marks optional arguments
Arguments	• **geometry** - a geometry • **min_wavelength** - minimum wavelength of waves • **max_wavelength** - maximum wavelength of waves • **min_amplitude** - minimum amplitude of waves • **max_amplitude** - maximum amplitude of waves • **seed** - specifies a random seed for generating waves. If the seed is 0, then a completely random set of waves will be generated.
Examples	• `wave_randomized(geom_from_wkt('LineString(0 0, 10 0)'), 2, 3, 0.1, 0.2)` → Randomly sized curved waves with wavelengths between 2 and 3 and amplitudes between 0.1 and 0.2 along the linestring

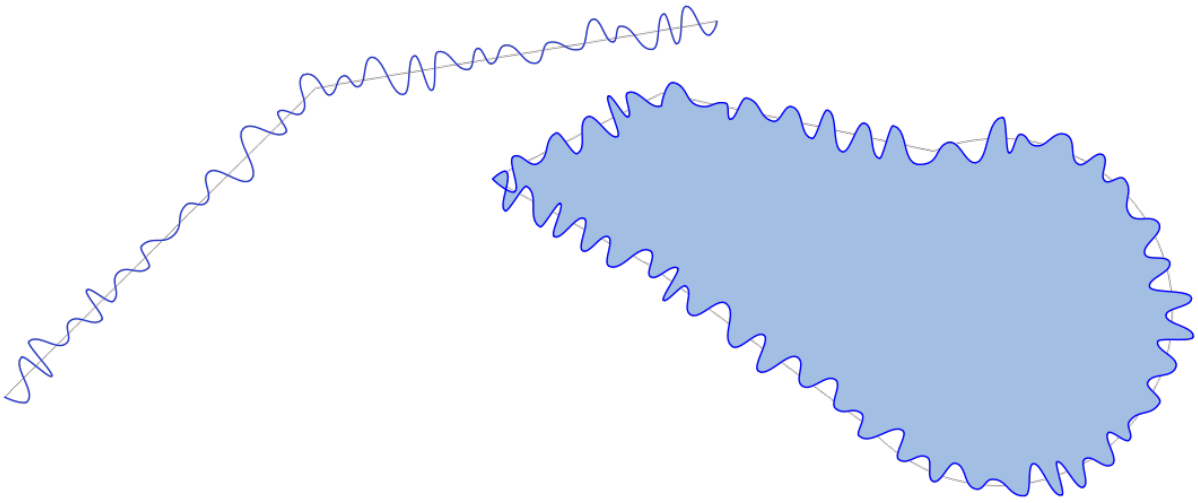


Fig. 9.40: Symbolizing features with randomized waves

wedge_buffer

Returns a wedge shaped buffer originating from a point geometry.

Syntax	wedge_buffer(center, azimuth, width, outer_radius, [inner_radius=0.0]) [] marks optional arguments
Arguments	• **center** - center point (origin) of buffer. Must be a point geometry. • **azimuth** - angle (in degrees) for the middle of the wedge to point. • **width** - buffer width (in degrees). Note that the wedge will extend to half of the angular width either side of the azimuth direction. • **outer_radius** - outer radius for buffers • **inner_radius** - optional inner radius for buffers
Examples	• `wedge_buffer(center:=geom_from_wkt('POINT(1 2)'), azimuth:=90,width:=180,outer_radius:=1)` → A wedge shaped buffer centered on the point (1,2), facing to the East, with a width of 180 degrees and outer radius of 1.

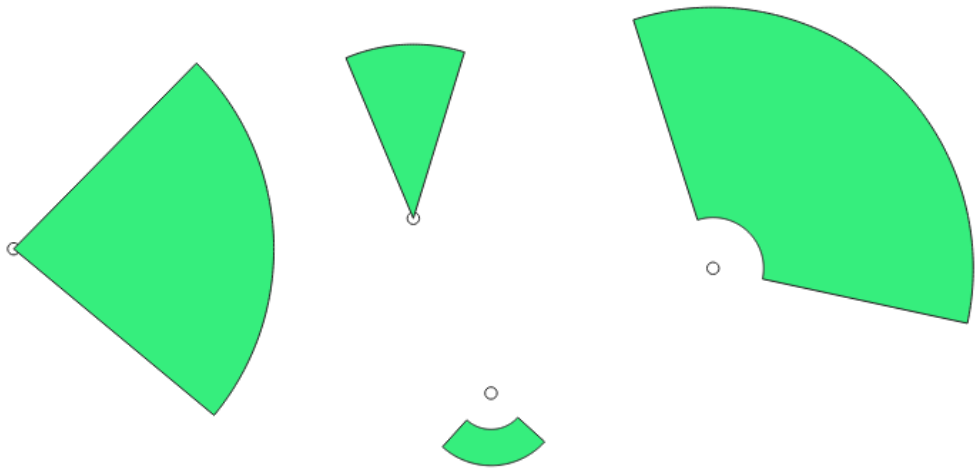


Fig. 9.41: Wedge buffering features

Further reading: *Create wedge buffers* algorithm

within

Tests whether a geometry is within another. Returns TRUE if the geometry1 is completely within geometry2.

Syntax	within(geometry1, geometry2)
Arguments	• **geometry1** - a geometry • **geometry2** - a geometry
Examples	• `within( geom_from_wkt( 'POINT( 0.5 0.5)' ), geom_from_wkt( 'POLYGON((0 0, 0 1, 1 1, 1 0, 0 0))' ) )` → TRUE • `within( geom_from_wkt( 'POINT( 5 5 )' ), geom_from_wkt( 'POLYGON((0 0, 0 1, 1 1, 1 0, 0 0 ))' ) )` → FALSE

Further reading: *overlay_within*

$x

Returns the x coordinate of the current point feature. If the feature is a multipoint feature, then the x-coordinate of the first point will be returned. **WARNING: This function is deprecated. It is recommended to use the replacement x() function with @geometry variable instead.**

Syntax	$x
Examples	• `$x` → 42

Further reading: *x*

x

Returns the x coordinate of a point geometry, or the x coordinate of the centroid for a non-point geometry.

Syntax	x(geometry)
Arguments	• **geometry** - a geometry
Examples	• `x( geom_from_wkt( 'POINT(2 5)' ) )` → 2 • `x( @geometry )` → x coordinate of the current feature's centroid

$x_at

Retrieves a x coordinate of the current feature's geometry. **WARNING: This function is deprecated. It is recommended to use the replacement x_at function with @geometry variable instead.**

Syntax	$x_at(vertex)
Arguments	• **vertex** - index of the vertex of the current geometry (indices start at 0; negative values apply from the last index, starting at -1)
Examples	• `$x_at(1)` → 5

Further reading: *x_at*

x_at

Retrieves a x coordinate of the geometry.

Syntax	x_at(geometry, vertex)
Arguments	• **geometry** - geometry object • **vertex** - index of the vertex of the geometry (indices start at 0; negative values apply from the last index, starting at -1)
Examples	• `x_at( geom_from_wkt( 'POINT(4 5)' ), 0 )` → 4

x_max

Returns the maximum x coordinate of a geometry. Calculations are in the spatial reference system of this geometry.

Syntax	x_max(geometry)
Arguments	• **geometry** - a geometry
Examples	• `x_max(@geometry)` → the maximum x coordinate of the current feature's geometry • `x_max( geom_from_wkt( 'LINESTRING(2 5, 3 6, 4 8)') )` → 4

x_min

Returns the minimum x coordinate of a geometry. Calculations are in the spatial reference system of this geometry.

Syntax	x_min(geometry)
Arguments	• **geometry** - a geometry
Examples	• `x_min(@geometry)` → the minimum x coordinate of the current feature's geometry • `x_min( geom_from_wkt( 'LINESTRING(2 5, 3 6, 4 8)') )` → 2

$y

Returns the y coordinate of the current point feature. If the feature is a multipoint feature, then the y-coordinate of the first point will be returned. **WARNING: This function is deprecated. It is recommended to use the replacement y() function with @geometry variable instead.**

Syntax	$y
Examples	• `$y` → 42

Further reading: *y*

y

Returns the y coordinate of a point geometry, or the y coordinate of the centroid for a non-point geometry.

Syntax	y(geometry)
Arguments	• **geometry** - a geometry
Examples	• `y( geom_from_wkt( 'POINT(2 5)' ) )` → 5 • `y( @geometry )` → y coordinate of the current feature's centroid

$y_at

Retrieves a y coordinate of the current feature's geometry. **WARNING: This function is deprecated. It is recommended to use the replacement y_at function with @geometry variable instead.**

Syntax	$y_at(vertex)
Arguments	• **vertex** - index of the vertex of the current geometry (indices start at 0; negative values apply from the last index, starting at -1)
Examples	• `$y_at(1)` → 2

Further reading: *y_at*

y_at

Retrieves a y coordinate of the geometry.

Syntax	y_at(geometry, vertex)
Arguments	• **geometry** - geometry object • **vertex** - index of the vertex of the geometry (indices start at 0; negative values apply from the last index, starting at -1)
Examples	• `y_at( geom_from_wkt( 'POINT(4 5)' ), 0 )` → 5

y_max

Returns the maximum y coordinate of a geometry. Calculations are in the spatial reference system of this geometry.

Syntax	y_max(geometry)
Arguments	• **geometry** - a geometry
Examples	• `y_max(@geometry)` → the maximum y coordinate of the current feature's geometry • `y_max( geom_from_wkt( 'LINESTRING(2 5, 3 6, 4 8)') )` → 8

y_min

Returns the minimum y coordinate of a geometry. Calculations are in the spatial reference system of this geometry.

Syntax	y_min(geometry)
Arguments	• **geometry** - a geometry
Examples	• `y_min(@geometry)` → the minimum y coordinate of the current feature's geometry • `y_min( geom_from_wkt( 'LINESTRING(2 5, 3 6, 4 8)') )` → 5

$z

Returns the z value of the current point feature if it is 3D. If the feature is a multipoint feature, then the z value of the first point will be returned. **WARNING: This function is deprecated. It is recommended to use the replacement z() function with @geometry variable instead.**

Syntax	$z
Examples	• `$z` → 123

z

Returns the z coordinate of a point geometry, or NULL if the geometry has no z value.

Syntax	z(geometry)
Arguments	• **geometry** - a point geometry
Examples	• `z( geom_from_wkt( 'POINTZ(2 5 7)' ) )` → 7

z_at

Retrieves a z coordinate of the geometry, or NULL if the geometry has no z value.

Syntax	z_at(geometry, vertex)
Arguments	• **geometry** - geometry object • **vertex** - index of the vertex of the geometry (indices start at 0; negative values apply from the last index, starting at -1)
Examples	• `z_at(geom_from_wkt('LineStringZ(0 0 0, 10 10 5, 10 10 0)'), 1)` → 5

z_max

Returns the maximum z coordinate of a geometry, or NULL if the geometry has no z value.

Syntax	z_max(geometry)
Arguments	• **geometry** - a geometry with z coordinate
Examples	• `z_max( geom_from_wkt( 'POINT ( 0 0 1 )' ) )` → 1 • `z_max( geom_from_wkt( 'MULTIPOINT ( 0 0 1 , 1 1 3 )' ) )` → 3 • `z_max( make_line( make_point( 0,0,0 ), make_point( -1,-1,-2 ) ) )` → 0 • `z_max( geom_from_wkt( 'LINESTRING( 0 0 0, 1 0 2, 1 1 -1 )' ) )` → 2 • `z_max( geom_from_wkt( 'POINT ( 0 0 )' ) )` → NULL

z_min

Returns the minimum z coordinate of a geometry, or NULL if the geometry has no z value.

Syntax	z_min(geometry)
Arguments	• **geometry** - a geometry with z coordinate
Examples	• `z_min( geom_from_wkt( 'POINT ( 0 0 1 )' ) )` → 1 • `z_min( geom_from_wkt( 'MULTIPOINT ( 0 0 1 , 1 1 3 )' ) )` → 1 • `z_min( make_line( make_point( 0,0,0 ), make_point( -1,-1,-2 ) ) )` → -2 • `z_min( geom_from_wkt( 'LINESTRING( 0 0 0, 1 0 2, 1 1 -1 )' ) )` → -1 • `z_min( geom_from_wkt( 'POINT ( 0 0 )' ) )` → NULL

9.2.15 Layout Functions

This group contains functions to manipulate print layout items properties.

item_variables

Returns a map of variables from a layout item inside this print layout.

Syntax	item_variables(id)
Arguments	• **id** - layout item ID
Examples	• `map_get( item_variables('Map 0'), 'map_scale')` → scale of the item 'Map 0' in the current print layout

Further reading: List of default *variables*

map_credits

Returns a list of credit (usage rights) strings for the layers shown in a layout, or specific layout map item.

Syntax	map_credits([id], [include_layer_names=false], [layer_name_separator=': ']) [] marks optional arguments
Arguments	• **id** - Map item ID. If not specified, the layers from all maps in the layout will be used. • **include_layer_names** - Set to true to include layer names before their credit strings • **layer_name_separator** - String to insert between layer names and their credit strings, if include_layer_names is true
Examples	• `array_to_string( map_credits() )` → comma separated list of layer credits for all layers shown in all map items in the layout, e.g 'CC-BY-NC, CC-BY-SA' • `array_to_string( map_credits( 'Main Map' ) )` → comma separated list of layer credits for layers shown in the 'Main Map' layout item, e.g 'CC-BY-NC, CC-BY-SA' • `array_to_string( map_credits( 'Main Map', include_layer_names := true, layer_name_separator := ': ' ) )` → comma separated list of layer names and their credits for layers shown in the 'Main Map' layout item, e.g. 'Railway lines: CC-BY-NC, Basemap: CC-BY-SA'

This function requires the *Access metadata properties* of the layers to have been filled.

9.2.16 Map Layers

This group contains a list of the available layers in the current project and, for each layer, their fields (stored in the dataset, virtual or auxiliary ones as well as from joins). The fields can be interacted the same way as mentioned in *Fields and Values*, except that a double-click will add the name as a string (single quoted) to the expression instead of as a field reference given that they do not belong to the active layer. This offers a convenient way to write expressions referring to different layers, such as when performing *aggregates*, *attribute* or *spatial* queries.

It also provides some convenient functions to manipulate layers.

decode_uri

Takes a layer and decodes the uri of the underlying data provider. It depends on the dataprovider, which data is available.

Syntax	decode_uri(layer, [part]) [] marks optional arguments
Arguments	• **layer** - The layer for which the uri should be decoded. • **part** - The part of the uri to return. If unspecified, a map with all uri parts will be returned.
Examples	• `decode_uri(@layer)` → {'layerId': '0', 'layerName': '', 'path': '/home/qgis/shapefile.shp'} • `decode_uri(@layer)` → {'layerId': NULL, 'layerName': 'layer', 'path': '/home/qgis/geopackage.gpkg'} • `decode_uri(@layer, 'path')` → 'C:\my_data\qgis\shape.shp'

layer_property

Returns a matching layer property or metadata value.

Syntax	layer_property(layer, property)
Arguments	• **layer** - a string, representing either a layer name or layer ID • **property** - a string corresponding to the property to return. Valid options are: – name: layer name – id: layer ID – title: metadata title string – abstract: metadata abstract string – keywords: metadata keywords – data_url: metadata URL – attribution: metadata attribution string – attribution_url: metadata attribution URL – source: layer source – min_scale: minimum display scale for layer – max_scale: maximum display scale for layer – is_editable: if layer is in edit mode – crs: layer CRS – crs_definition: layer CRS full definition – crs_description: layer CRS description – crs_ellipsoid: acronym of the layer CRS ellipsoid – extent: layer extent (as a geometry object) – distance_units: layer distance units – type: layer type, e.g., Vector or Raster – storage_type: storage format (vector layers only) – geometry_type: geometry type, e.g., Point (vector layers only) – feature_count: approximate feature count for layer (vector layers only) – path: File path to the layer data source. Only available for file based layers.
Examples	• `layer_property('streets','title')` → 'Basemap Streets' • `layer_property('airports','feature_count')` → 120 • `layer_property('landsat','crs')` → 'EPSG:4326'

Further reading: *vector*, *raster* and *mesh* layer properties

load_layer

Loads a layer by source URI and provider name.

Syntax	load_layer(uri, provider)
Arguments	• **uri** - layer source URI string • **provider** - layer data provider name
Examples	• `layer_property(load_layer('c:/data/roads.shp', 'ogr'), 'feature_count')` → count of features from the c:/data/roads.shp vector layer

9.2.17 Maps Functions

This group contains functions to create or manipulate keys and values of map data structures (also known as dictionary objects, key-value pairs, or associative arrays). Unlike the *list data structure* where values order matters, the order of the key-value pairs in the map object is not relevant and values are identified by their keys.

from_json

Loads a JSON formatted string.

Syntax	from_json(string)
Arguments	• **string** - JSON string
Examples	• `from_json('{"1":"one","2":"two"}')` → { '1': 'one', '2': 'two' } • `from_json('[1,2,3]')` → [1,2,3]

hstore_to_map

Creates a map from a hstore-formatted string.

Syntax	hstore_to_map(string)
Arguments	• **string** - the input string
Examples	• `hstore_to_map('qgis=>rocks')` → { 'qgis': 'rocks' }

map

Returns a map containing all the keys and values passed as pair of parameters.

Syntax	map(key1, value1, key2, value2, ...)
Arguments	• **key** - a key (string) • **value** - a value
Examples	• `map('1','one','2', 'two')` → { '1': 'one', '2': 'two' } • `map('1','one','2', 'two')['1']` → 'one'

map_akeys

Returns all the keys of a map as an array.

Syntax	map_akeys(map)
Arguments	• **map** - a map
Examples	• `map_akeys(map('1','one','2','two'))` → ['1', '2']

map_avals

Returns all the values of a map as an array.

Syntax	map_avals(map)
Arguments	• **map** - a map
Examples	• `map_avals(map('1','one','2','two'))` → ['one', 'two']

map_concat

Returns a map containing all the entries of the given maps. If two maps contain the same key, the value of the second map is taken.

Syntax	map_concat(map1, map2, …)
Arguments	• **map** - a map
Examples	• `map_concat(map('1','one', '2','overridden'),map('2','two', '3','three'))` → { '1': 'one', '2': 'two', '3': 'three' }

map_delete

Returns a map with the given key and its corresponding value deleted.

Syntax	map_delete(map, key)
Arguments	• **map** - a map • **key** - the key to delete
Examples	• `map_delete(map('1','one','2','two'),'2')` → { '1': 'one' }

map_exist

Returns TRUE if the given key exists in the map.

Syntax	map_exist(map, key)
Arguments	• **map** - a map • **key** - the key to lookup
Examples	• `map_exist(map('1','one','2','two'),'3')` → FALSE

map_get

Returns the value of a map, given its key. Returns NULL if the key does not exist.

Syntax	map_get(map, key)
Arguments	• **map** - a map • **key** - the key to lookup
Examples	• `map_get(map('1','one','2','two'),'2')` → 'two' • `map_get( item_variables('Map 0'), 'map_scale')` → scale of the item 'Map 0' (if it exists) in the current print layout

Hint

You can also use the *index operator ([])* to get a value from a map.

map_insert

Returns a map with an added key/value. If the key already exists, its value is overridden.

Syntax	map_insert(map, key, value)
Arguments	• **map** - a map • **key** - the key to add • **value** - the value to add
Examples	• `map_insert(map('1','one'),'3','three')` → { '1': 'one', '3': 'three' } • `map_insert(map('1','one','2','overridden'),'2','two')` → { '1': 'one', '2': 'two' }

map_prefix_keys

Returns a map with all keys prefixed by a given string.

Syntax	map_prefix_keys(map, prefix)
Arguments	• **map** - a map • **prefix** - a string
Examples	• `map_prefix_keys(map('1','one','2','two'), 'prefix-')` → { 'prefix-1': 'one', 'prefix-2': 'two' }

map_to_hstore

Merge map elements into a hstore-formatted string.

Syntax	map_to_hstore(map)
Arguments	• **map** - the input map
Examples	• `map_to_hstore(map('1','one','2','two'))` → '"1"=>"one","2"=>"two"'

map_to_html_dl

Merge map elements into a HTML definition list string.

Syntax	map_to_html_dl(map)
Arguments	• **map** - the input map
Examples	• `map_to_html_dl(map('1','one','2','two'))` → <dl><dt>1</dt><dd>one</dd><dt>2</dt><dd>two</dd></dl>

map_to_html_table

Merge map elements into a HTML table string.

Syntax	map_to_html_table(map)
Arguments	• **map** - the input map
Examples	• `map_to_html_table(map('1','one','2','two'))` → <table><thead><tr><th>1</th><th>2</th></tr></thead><tbody><tr><td>one</td><td>two</td></tr></tbody

to_json

Create a JSON formatted string from a map, array or other value.

Syntax	to_json(value)
Arguments	• **value** - The input value
Examples	• `to_json(map('1','one','2','two'))` → {"1":"one","2":"two"} • `to_json(array(1,2,3))` → [1,2,3]

url_encode

Returns an URL encoded string from a map. Transforms all characters in their properly-encoded form producing a fully-compliant query string.

Note that the plus sign '+' is not converted.

Syntax	url_encode(map)
Arguments	• **map** - a map.
Examples	• `url_encode(map('a&+b', 'a and plus b', 'a=b', 'a equals b'))` → 'a%26+b=a%20and%20plus%20b&a%3Db=a%20equals%20b'

9.2.18 Mathematical Functions

This group contains math functions (e.g., square root, sin and cos).

abs

Returns the absolute value of a number.

Syntax	abs(value)
Arguments	• **value** - a number
Examples	• `abs(-2)` → 2

acos

Returns the inverse cosine of a value in radians.

Syntax	acos(value)
Arguments	• **value** - cosine of an angle in radians
Examples	• `acos(0.5)` → 1.0471975511966

asin

Returns the inverse sine of a value in radians.

Syntax	asin(value)
Arguments	• **value** - sine of an angle in radians
Examples	• `asin(1.0)` → 1.5707963267949

atan

Returns the inverse tangent of a value in radians.

Syntax	atan(value)
Arguments	• **value** - tan of an angle in radians
Examples	• `atan(0.5)` → 0.463647609000806

atan2

Returns the inverse tangent of dy/dx by using the signs of the two arguments to determine the quadrant of the result.

Syntax	atan2(dy, dx)
Arguments	• **dy** - y coordinate difference • **dx** - x coordinate difference
Examples	• `atan2(1.0, 1.732)` → 0.523611477769969

ceil

Rounds a number upwards.

Syntax	ceil(value)
Arguments	• **value** - a number
Examples	• `ceil(4.9)` → 5 • `ceil(-4.9)` → -4

clamp

Restricts an input value to a specified range.

Syntax	clamp(minimum, input, maximum)
Arguments	• **minimum** - the smallest value *input* is allowed to take. • **input** - a value which will be restricted to the range specified by *minimum* and *maximum* • **maximum** - the largest value *input* is allowed to take
Examples	• `clamp(1,5,10)` → 5 *input* is between 1 and 10 so is returned unchanged • `clamp(1,0,10)` → 1 *input* is less than minimum value of 1, so function returns 1 • `clamp(1,11,10)` → 10 *input* is greater than maximum value of 10, so function returns 10

cos

Returns cosine of an angle.

Syntax	cos(angle)
Arguments	• **angle** - angle in radians
Examples	• `cos(1.571)` → 0.000796326710733263

degrees

Converts from radians to degrees.

Syntax	degrees(radians)
Arguments	• **radians** - numeric value
Examples	• `degrees(3.14159)` → 180 • `degrees(1)` → 57.2958

exp

Returns exponential of an value.

Syntax	exp(value)
Arguments	• **value** - number to return exponent of
Examples	• `exp(1.0)` → 2.71828182845905

floor

Rounds a number downwards.

Syntax	floor(value)
Arguments	• **value** - a number
Examples	• `floor(4.9)` → 4 • `floor(-4.9)` → -5

ln

Returns the natural logarithm of a value.

Syntax	ln(value)
Arguments	• **value** - numeric value
Examples	• `ln(1)` → 0 • `ln(2.7182818284590452354)` → 1

log

Returns the value of the logarithm of the passed value and base.

Syntax	log(base, value)
Arguments	• **base** - any positive number • **value** - any positive number
Examples	• `log(2, 32)` → 5 • `log(0.5, 32)` → -5

log10

Returns the value of the base 10 logarithm of the passed expression.

Syntax	log10(value)
Arguments	• **value** - any positive number
Examples	• `log10(1)` → 0 • `log10(100)` → 2

max

Returns the largest value in a set of values.

Syntax	max(value1, value2, …)
Arguments	• **value** - a number
Examples	• `max(2,10.2,5.5)` → 10.2 • `max(20.5,NULL,6.2)` → 20.5

min

Returns the smallest value in a set of values.

Syntax	min(value1, value2, …)
Arguments	• **value** - a number
Examples	• `min(20.5,10,6.2)` → 6.2 • `min(2,-10.3,NULL)` → -10.3

pi

Returns value of pi for calculations.

Syntax	pi()
Examples	• `pi()` → 3.14159265358979

radians

Converts from degrees to radians.

Syntax	radians(degrees)
Arguments	• **degrees** - numeric value
Examples	• `radians(180)` → 3.14159 • `radians(57.2958)` → 1

rand

Returns a random integer within the range specified by the minimum and maximum argument (inclusive). If a seed is provided, the returned will always be the same, depending on the seed.

Syntax	rand(min, max, [seed=NULL]) [] marks optional arguments
Arguments	• **min** - an integer representing the smallest possible random number desired • **max** - an integer representing the largest possible random number desired • **seed** - any value to use as seed
Examples	• `rand(1, 10)` → 8

randf

Returns a random float within the range specified by the minimum and maximum argument (inclusive). If a seed is provided, the returned will always be the same, depending on the seed.

Syntax	randf([min=0.0], [max=1.0], [seed=NULL]) [] marks optional arguments
Arguments	• **min** - an float representing the smallest possible random number desired • **max** - an float representing the largest possible random number desired • **seed** - any value to use as seed
Examples	• `randf(1, 10)` → 4.59258286403147

round

Rounds a number to number of decimal places. The rounding is done to the nearest number. The ceil or floor functions can be used if rounding up or down is required.

Syntax	round(value, [places=0]) [] marks optional arguments
Arguments	• **value** - decimal number to be rounded • **places** - Optional integer representing number of places to round decimals to. For negative integers the rounding is done to the left of the decimal point.
Examples	• `round(1234.567, 2)` → 1234.57 • `round(1234.567)` → 1235 • `round(1234.567, -1)` → 1230 • `round(1234.567, -2)` → 1200

scale_exponential

Transforms a given value from an input domain to an output range using an exponential curve. This function can be used to ease values in or out of the specified output range.

Syntax	scale_exponential(value, domain_min, domain_max, range_min, range_max, exponent)
Arguments	• **value** - A value in the input domain. The function will return a corresponding scaled value in the output range. • **domain_min** - Specifies the minimum value in the input domain, the smallest value the input value should take. • **domain_max** - Specifies the maximum value in the input domain, the largest value the input value should take. • **range_min** - Specifies the minimum value in the output range, the smallest value which should be output by the function. • **range_max** - Specifies the maximum value in the output range, the largest value which should be output by the function. • **exponent** - A positive value (greater than 0), which dictates the way input values are mapped to the output range. Large exponents will cause the output values to 'ease in', starting slowly before accelerating as the input values approach the domain maximum. Smaller exponents (less than 1) will cause output values to 'ease out', where the mapping starts quickly but slows as it approaches the domain maximum.
Examples	• `scale_exponential(5,0,10,0,100,2)` → 3.030 easing in, using an exponent of 2 • `scale_exponential(3,0,10,0,100,0.5)` → 87.585 easing out, using an exponent of 0.5

scale_linear

Transforms a given value from an input domain to an output range using linear interpolation.

Syntax	scale_linear(value, domain_min, domain_max, range_min, range_max)
Arguments	• **value** - A value in the input domain. The function will return a corresponding scaled value in the output range. • **domain_min** - Specifies the minimum value in the input domain, the smallest value the input value should take. • **domain_max** - Specifies the maximum value in the input domain, the largest value the input value should take. • **range_min** - Specifies the minimum value in the output range, the smallest value which should be output by the function. • **range_max** - Specifies the maximum value in the output range, the largest value which should be output by the function.
Examples	• `scale_linear(5,0,10,0,100)` → 50 • `scale_linear(0.2,0,1,0,360)` → 72 scaling a value between 0 and 1 to an angle between 0 and 360 • `scale_linear(1500,1000,10000,9,20)` → 9.6111111 scaling a population which varies between 1000 and 10000 to a font size between 9 and 20

scale_polynomial

Transforms a given value from an input domain to an output range using a polynomial curve. This function can be used to ease values in or out of the specified output range.

Syntax	scale_polynomial(value, domain_min, domain_max, range_min, range_max, exponent)
Arguments	• **value** - A value in the input domain. The function will return a corresponding scaled value in the output range. • **domain_min** - Specifies the minimum value in the input domain, the smallest value the input value should take. • **domain_max** - Specifies the maximum value in the input domain, the largest value the input value should take. • **range_min** - Specifies the minimum value in the output range, the smallest value which should be output by the function. • **range_max** - Specifies the maximum value in the output range, the largest value which should be output by the function. • **exponent** - A positive value (greater than 0), which dictates the way input values are mapped to the output range. Large exponents will cause the output values to 'ease in', starting slowly before accelerating as the input values approach the domain maximum. Smaller exponents (less than 1) will cause output values to 'ease out', where the mapping starts quickly but slows as it approaches the domain maximum.
Examples	• `scale_polynomial(5,0,10,0,100,2)` → 25 easing in, using an exponent of 2 • `scale_polynomial(3,0,10,0,100,0.5)` → 54.772 easing out, using an exponent of 0.5

sin

Returns the sine of an angle.

Syntax	sin(angle)
Arguments	• **angle** - angle in radians
Examples	• `sin(1.571)` → 0.999999682931835

sqrt

Returns square root of a value.

Syntax	sqrt(value)
Arguments	• **value** - a number
Examples	• `sqrt(9)` → 3

tan

Returns the tangent of an angle.

Syntax	tan(angle)
Arguments	• **angle** - angle in radians
Examples	• `tan(1.0)` → 1.5574077246549

9.2.19 Meshes Functions

This group contains functions which calculate or return mesh related values.

$face_area

Returns the area of the current mesh face. The area calculated by this function respects both the current project's ellipsoid setting and area unit settings. For example, if an ellipsoid has been set for the project then the calculated area will be ellipsoidal, and if no ellipsoid is set then the calculated area will be planimetric.

Syntax	$face_area
Examples	• `$face_area` → 42

$face_index

Returns the index of the current mesh face.

Syntax	$face_index
Examples	• `$face_index` → 4581

$vertex_as_point

Returns the current vertex as a point geometry.

Syntax	$vertex_as_point
Examples	• `geom_to_wkt( $vertex_as_point )` → 'POINT(800 1500 41)'

$vertex_index

Returns the index of the current mesh vertex.

Syntax	$vertex_index
Examples	• `$vertex_index` → 9874

$vertex_x

Returns the X coordinate of the current mesh vertex.

Syntax	$vertex_x
Examples	• `$vertex_x` → 42.12

$vertex_y

Returns the Y coordinate of the current mesh vertex.

Syntax	$vertex_y
Examples	• `$vertex_y` → 12.24

$vertex_z

Returns the Z value of the current mesh vertex.

Syntax	$vertex_z
Examples	• `$vertex_z` → 42

9.2.20 Operators

This group contains operators (e.g., +, -, *). Note that for most of the mathematical functions below, if one of the inputs is NULL then the result is NULL.

%

Remainder of division. Takes the sign of the dividend.

Syntax	a % b
Arguments	• **a** - value • **b** - value
Examples	• `9 % 2` → 1 • `9 % -2` → 1 • `-9 % 2` → -1 • `5 % NULL` → NULL

*

Multiplication of two values

Syntax	a * b
Arguments	• **a** - value • **b** - value
Examples	• `5 * 4` → 20 • `5 * NULL` → NULL

+

Addition of two values. If one of the values is NULL the result will be NULL.

Syntax	a + b
Arguments	• **a** - value • **b** - value
Examples	• `5 + 4` → 9 • `5 + NULL` → NULL • `'QGIS ' + 'ROCKS'` → 'QGIS ROCKS' • `to_datetime('2020-08-01 12:00:00') + '1 day 2 hours'` → 2020-08-02T14:00:00

Further reading: *concat*, ||

-

Subtraction of two values. If one of the values is NULL the result will be NULL.

Syntax	a - b
Arguments	• **a** - value • **b** - value
Examples	• `5 - 4` → 1 • `5 - NULL` → NULL • `to_datetime('2012-05-05 12:00:00') - to_interval('1 day 2 hours')` → 2012-05-04T10:00:00

/

Division of two values

Syntax	a / b
Arguments	• **a** - value • **b** - value
Examples	• `5 / 4` → 1.25 • `5 / NULL` → NULL

//

Floor division of two values

Syntax	a // b
Arguments	• **a** - value • **b** - value
Examples	• `9 // 2` → 4

<

Compares two values and evaluates to 1 if the left value is less than the right value.

Syntax	a < b
Arguments	• **a** - value • **b** - value
Examples	• `5 < 4` → FALSE • `5 < 5` → FALSE • `4 < 5` → TRUE

<=

Compares two values and evaluates to 1 if the left value is less or equal than the right value.

Syntax	a <= b
Arguments	• **a** - value • **b** - value
Examples	• `5 <= 4` → FALSE • `5 <= 5` → TRUE • `4 <= 5` → TRUE

<>

Compares two values and evaluates to 1 if they are not equal.

Syntax	a <> b
Arguments	• **a** - value • **b** - value
Examples	• `5 <> 4` → TRUE • `4 <> 4` → FALSE • `5 <> NULL` → NULL • `NULL <> NULL` → NULL

=

Compares two values and evaluates to 1 if they are equal.

Syntax	a = b
Arguments	• **a** - value • **b** - value
Examples	• `5 = 4` → FALSE • `4 = 4` → TRUE • `5 = NULL` → NULL • `NULL = NULL` → NULL

>

Compares two values and evaluates to 1 if the left value is greater than the right value.

Syntax	a > b
Arguments	• **a** - value • **b** - value
Examples	• `5 > 4` → TRUE • `5 > 5` → FALSE • `4 > 5` → FALSE

>=

Compares two values and evaluates to 1 if the left value is greater or equal than the right value.

Syntax	a >= b
Arguments	• **a** - value • **b** - value
Examples	• `5 >= 4` → TRUE • `5 >= 5` → TRUE • `4 >= 5` → FALSE

AND

Returns TRUE when conditions a and b are true.

Syntax	a AND b
Arguments	• **a** - condition • **b** - condition
Examples	• `TRUE AND TRUE` → TRUE • `TRUE AND FALSE` → FALSE • `4 = 2+2 AND 1 = 1` → TRUE • `4 = 2+2 AND 1 = 2` → FALSE

BETWEEN

Returns TRUE if value is within the specified range. The range is considered inclusive of the bounds. To test for exclusion NOT BETWEEN can be used.

Syntax	value BETWEEN lower_bound AND higher_bound
Arguments	• **value** - the value to compare with a range. It can be a string, a number or a date. • **lower_bound AND higher_bound** - range bounds
Examples	• `'B' BETWEEN 'A' AND 'C'` → TRUE • `2 BETWEEN 1 AND 3` → TRUE • `2 BETWEEN 2 AND 3` → TRUE • `'B' BETWEEN 'a' AND 'c'` → FALSE • `lower('B') BETWEEN 'a' AND 'b'` → TRUE

Note

value BETWEEN lower_bound AND higher_bound is the same as "*value >= lower_bound AND value <= higher_bound*".

Further reading: *NOT BETWEEN*

ILIKE

Returns TRUE if the first parameter matches case-insensitive the supplied pattern. LIKE can be used instead of ILIKE to make the match case-sensitive. Works with numbers also.

Syntax	string/number ILIKE pattern
Arguments	• **string/number** - string to search • **pattern** - pattern to find, you can use '%' as a wildcard, '_' as a single char and '\\' to escape these special characters.
Examples	• `'A' ILIKE 'A'` → TRUE • `'A' ILIKE 'a'` → TRUE • `'A' ILIKE 'B'` → FALSE • `'ABC' ILIKE 'b'` → FALSE • `'ABC' ILIKE 'B'` → FALSE • `'ABC' ILIKE '_b_'` → TRUE • `'ABC' ILIKE '_B_'` → TRUE • `'ABCD' ILIKE '_b_'` → FALSE • `'ABCD' ILIKE '_B_'` → FALSE • `'ABCD' ILIKE '_b%'` → TRUE • `'ABCD' ILIKE '_B%'` → TRUE • `'ABCD' ILIKE '%b%'` → TRUE • `'ABCD' ILIKE '%B%'` → TRUE • `'ABCD%' ILIKE 'abcd\\%'` → TRUE • `'ABCD' ILIKE '%B\\%'` → FALSE

IN

Returns TRUE if value is found within a list of values.

Syntax	a IN b
Arguments	• **a** - value • **b** - list of values
Examples	• `'A' IN ('A','B')` → TRUE • `'A' IN ('C','B')` → FALSE

IS

Returns TRUE if a is the same as b.

Syntax	a IS b
Arguments	• **a** - any value • **b** - any value
Examples	• `'A' IS 'A'` → TRUE • `'A' IS 'a'` → FALSE • `4 IS 4` → TRUE • `4 IS 2+2` → TRUE • `4 IS 2` → FALSE • `@geometry IS NULL` → 0, if your geometry is not NULL

IS NOT

Returns TRUE if a is not the same as b.

Syntax	a IS NOT b
Arguments	• **a** - value • **b** - value
Examples	• `'a' IS NOT 'b'` → TRUE • `'a' IS NOT 'a'` → FALSE • `4 IS NOT 2+2` → FALSE

LIKE

Returns TRUE if the first parameter matches the supplied pattern. Works with numbers also.

Syntax	string/number LIKE pattern
Arguments	• **string/number** - value • **pattern** - pattern to compare value with, you can use '%' as a wildcard, '_' as a single char and '\\' to escape these special characters.
Examples	• `'A' LIKE 'A'` → TRUE • `'A' LIKE 'a'` → FALSE • `'A' LIKE 'B'` → FALSE • `'ABC' LIKE 'B'` → FALSE • `'ABC' LIKE '_B_'` → TRUE • `'ABCD' LIKE '_B_'` → FALSE • `'ABCD' LIKE '_B%'` → TRUE • `'ABCD' LIKE '%B%'` → TRUE • `'1%' LIKE '1\\%'` → TRUE • `'1_' LIKE '1\\%'` → FALSE

NOT

Negates a condition.

Syntax	NOT a
Arguments	• **a** - condition
Examples	• `NOT 1` → FALSE • `NOT 0` → TRUE

NOT BETWEEN

Returns TRUE if value is not within the specified range. The range is considered inclusive of the bounds.

Syntax	value NOT BETWEEN lower_bound AND higher_bound
Arguments	• **value** - the value to compare with a range. It can be a string, a number or a date. • **lower_bound AND higher_bound** - range bounds
Examples	• `'B' NOT BETWEEN 'A' AND 'C'` → FALSE • `1.0 NOT BETWEEN 1.1 AND 1.2` → TRUE • `2 NOT BETWEEN 2 AND 3` → FALSE • `'B' NOT BETWEEN 'a' AND 'c'` → TRUE • `lower('B') NOT BETWEEN 'a' AND 'b'` → FALSE

Note

value NOT BETWEEN lower_bound AND higher_bound is the same as "*value* < *lower_bound OR value* > *higher_bound*".

Further reading: *BETWEEN*

OR

Returns TRUE when condition a or b is true.

Syntax	a OR b
Arguments	• **a** - condition • **b** - condition
Examples	• `4 = 2+2 OR 1 = 1` → TRUE • `4 = 2+2 OR 1 = 2` → TRUE • `4 = 2 OR 1 = 2` → FALSE

[]

Index operator. Returns an element from an array or map value.

Syntax	[index]
Arguments	• **index** - array index or map key value
Examples	• `array(1,2,3)[0]` → 1 • `array(1,2,3)[2]` → 3 • `array(1,2,3)[-1]` → 3 • `map('a',1,'b',2)['a']` → 1 • `map('a',1,'b',2)['b']` → 2

Further reading: *array_get*, *map_get*

^

Power of two values.

Syntax	a ^ b
Arguments	• **a** - value • **b** - value
Examples	• `5 ^ 4` → 625 • `5 ^ NULL` → NULL

||

Joins two values together into a string.

If one of the values is NULL the result will be NULL. See the CONCAT function for a different behavior.

Syntax	a || b
Arguments	• **a** - value • **b** - value
Examples	• `'Here' || ' and ' || 'there'` → 'Here and there' • `'Nothing' || NULL` → NULL • `'Dia: ' || "Diameter"` → 'Dia: 25' • `1 || 2` → '12'

Further reading: *concat*, *+*

~

Performs a regular expression match on a string value. Backslash characters must be double escaped (e.g., "\\s" to match a white space character).

Syntax	string ~ regex
Arguments	• **string** - A string value • **regex** - A regular expression. Slashes must be escaped, eg \\d.
Examples	• `'hello' ~ 'll'` → TRUE • `'hello' ~ '^ll'` → FALSE • `'hello' ~ 'llo$'` → TRUE • `'abc123' ~ '\\d+'` → TRUE

Further reading: *regexp_match*

9.2.21 Processing Functions

This group contains functions that operate on processing algorithms.

parameter

Returns the value of a processing algorithm input parameter.

Syntax	parameter(name)
Arguments	• **name** - name of the corresponding input parameter
Examples	• `parameter('BUFFER_SIZE')` → 5.6

9.2.22 Rasters Functions

This group contains functions to operate on raster layer.

raster_attributes

Returns a map with the fields names as keys and the raster attribute table values as values from the attribute table entry that matches the given raster value.

Syntax	raster_attributes(layer, band, value)
Arguments	• **layer** - the name or id of a raster layer • **band** - the band number for the associated attribute table lookup. • **value** - raster value
Examples	• `raster_attributes('vegetation', 1, raster_value('vegetation', 1, make_point(1,1)))` → {'class': 'Vegetated', 'subclass': 'Trees'} • `raster_attributes('vegetation', 1, raster_value('vegetation', 1, @layer_cursor_point))` → {'class': 'Vegetated', 'subclass': 'Trees'}

raster_statistic

Returns statistics from a raster layer.

Syntax	raster_statistic(layer, band, property)

Arguments	• **layer** - a string, representing either a raster layer name or layer ID
	• **band** - integer representing the band number from the raster layer, starting at 1
	• **property** - a string corresponding to the property to return. Valid options are:
	  – min: minimum value
	  – max: maximum value
	  – avg: average (mean) value
	  – stdev: standard deviation of values
	  – range: range of values (max - min)
	  – sum: sum of all values from raster

Examples	• `raster_statistic('lc',1,'avg')` → Average value from band 1 from 'lc' raster layer • `raster_statistic('ac2010',3,'min')` → Minimum value from band 3 from 'ac2010' raster layer

raster_value

Returns the raster value found at the provided point.

Syntax	raster_value(layer, band, point)
Arguments	• **layer** - the name or id of a raster layer • **band** - the band number to sample the value from. • **point** - point geometry (for multipart geometries having more than one part, a NULL value will be returned)
Examples	• `raster_value('dem', 1, make_point(1,1))` → 25 • `raster_value('ndvi', 2, @layer_cursor_point)` → 25

9.2.23 Record and Attributes Functions

This group contains functions that operate on record identifiers.

attribute

Returns an attribute from a feature.

Variant 1

Returns the value of an attribute from the current feature.

Syntax	attribute(attribute_name)
Arguments	• **attribute_name** - name of attribute to be returned
Examples	• `attribute( 'name' )` → value stored in 'name' attribute for the current feature

Variant 2

Allows the target feature and attribute name to be specified.

Syntax	attribute(feature, attribute_name)
Arguments	• **feature** - a feature • **attribute_name** - name of attribute to be returned
Examples	• `attribute( @atlas_feature, 'name' )` → value stored in 'name' attribute for the current atlas feature

attributes

Returns a map containing all attributes from a feature, with field names as map keys.

Variant 1

Returns a map of all attributes from the current feature.

Syntax	attributes()
Examples	• `attributes()['name']` → value stored in 'name' attribute for the current feature

Variant 2

Allows the target feature to be specified.

Syntax	attributes(feature)
Arguments	• **feature** - a feature
Examples	• `attributes( @atlas_feature )['name']` → value stored in 'name' attribute for the current atlas feature

Further reading: *Maps Functions*

$currentfeature

Returns the current feature being evaluated. This can be used with the 'attribute' function to evaluate attribute values from the current feature. **WARNING: This function is deprecated. It is recommended to use the replacement @feature variable instead.**

Syntax	$currentfeature
Examples	• `attribute( $currentfeature, 'name' )` → value stored in 'name' attribute for the current feature

display_expression

Returns the display expression for a given feature in a layer. The expression is evaluated by default. Can be used with zero, one or more arguments, see below for details.

No parameters

If called with no parameters, the function will evaluate the display expression of the current feature in the current layer.

Syntax	display_expression()
Examples	• `display_expression()` → The display expression of the current feature in the current layer.

One 'feature' parameter

If called with a 'feature' parameter only, the function will evaluate the specified feature from the current layer.

Syntax	display_expression(feature)
Arguments	• **feature** - The feature which should be evaluated.
Examples	• `display_expression(@atlas_feature)` → The display expression of the current atlas feature.

Layer and feature parameters

If the function is called with both a layer and a feature, it will evaluate the specified feature from the specified layer.

Syntax	display_expression(layer, feature, [evaluate=true]) [] marks optional arguments
Arguments	• **layer** - The layer (or its ID or name) • **feature** - The feature which should be evaluated. • **evaluate** - If the expression must be evaluated. If false, the expression will be returned as a string literal only (which could potentially be later evaluated using the 'eval' function).
Examples	• `display_expression( 'streets', get_feature_by_id('streets', 1))` → The display expression of the feature with the ID 1 on the layer 'streets'. • `display_expression('a_layer_id', @feature, 'False')` → The display expression of the given feature not evaluated.

feature_id

Returns a feature's unique ID, or NULL if the feature is not valid.

Syntax	feature_id(feature)
Arguments	• **feature** - a feature object
Examples	• `feature_id( @feature )` → the ID of the current feature

Further reading: *get_feature_by_id*

get_feature

Returns the first feature of a layer matching a given attribute value.

Single value variant

Along with the layer ID, a single column and value are specified.

Syntax	get_feature(layer, attribute, value)
Arguments	• **layer** - layer name or ID • **attribute** - attribute name to use for the match • **value** - attribute value to match
Examples	• `get_feature('streets','name','main  st')` → first feature found in "streets" layer with "main st" value in the "name" field

Map variant

Along with the layer ID, a map containing the columns (key) and their respective value to be used.

Syntax	get_feature(layer, attribute)
Arguments	• **layer** - layer name or ID • **attribute** - Map containing the column and value pairs to use
Examples	• `get_feature('streets',map('name','main st','lane_num','4'))` → first feature found in "streets" layer with "main st" value in the "name" field and "4" value in the "lane_num" field

get_feature_by_id

Returns the feature with an id on a layer.

Syntax	get_feature_by_id(layer, feature_id)
Arguments	• **layer** - layer, layer name or layer id • **feature_id** - the id of the feature which should be returned
Examples	• `get_feature_by_id('streets', 1)` → the feature with the id 1 on the layer "streets"

Further reading: *feature_id*

$id

Returns the feature id of the current row. **WARNING: This function is deprecated. It is recommended to use the replacement @id variable instead.**

Syntax	$id
Examples	• `$id` → 42

Further reading: *feature_id*, *get_feature_by_id*

is_attribute_valid

Returns TRUE if a specific feature attribute meets all constraints.

Syntax	is_attribute_valid(attribute, [feature], [layer], [strength]) [] marks optional arguments
Arguments	• **attribute** - an attribute name • **feature** - A feature. If not set, the current feature will be used. • **layer** - A vector layer. If not set, the current layer will be used. • **strength** - Set to 'hard' or 'soft' to narrow down to a specific constraint type. If not set, the function will return FALSE if either a hard or a soft constraint fails.
Examples	• `is_attribute_valid('HECTARES')` → TRUE if the current feature's value in the "HECTARES" field meets all constraints. • `is_attribute_valid('HOUSES',get_feature('my_layer', 'FID', 10), 'my_layer')` → FALSE if the value in the "HOUSES" field from the feature with "FID"=10 in 'my_layer' fails to meet all constraints.

Further reading: *Constraints*

is_feature_valid

Returns TRUE if a feature meets all field constraints.

Syntax	is_feature_valid([feature], [layer], [strength]) [] marks optional arguments
Arguments	• **feature** - A feature. If not set, the current feature will be used. • **layer** - A vector layer. If not set, the current layer will be used. • **strength** - Set to 'hard' or 'soft' to narrow down to a specific constraint type. If not set, the function will return FALSE if either a hard or a soft constraint fails.
Examples	• `is_feature_valid(strength:='hard')` → TRUE if all fields from the current feature meet their hard constraints. • `is_feature_valid(get_feature('my_layer', 'FID', 10), 'my_layer')` → FALSE if all fields from feature with "FID"=10 in 'my_layer' fails to meet all constraints.

Further reading: *Constraints*

is_selected

Returns TRUE if a feature is selected. Can be used with zero, one or two arguments, see below for details.

No parameters

If called with no parameters, the function will return TRUE if the current feature in the current layer is selected.

Syntax	is_selected()
Examples	• `is_selected()` → TRUE if the current feature in the current layer is selected.

One 'feature' parameter

If called with a 'feature' parameter only, the function returns TRUE if the specified feature from the current layer is selected.

Syntax	is_selected(feature)
Arguments	• **feature** - The feature which should be checked for selection.
Examples	• `is_selected(@atlas_feature)` → TRUE if the current atlas feature is selected. • `is_selected(get_feature('streets', 'name', 'Main St.'))` → TRUE if the unique named "Main St." feature on the active "streets" layer is selected. • `is_selected(get_feature_by_id('streets', 1))` → TRUE if the feature with the id 1 on the active "streets" layer is selected.

Two parameters

If the function is called with both a layer and a feature, it will return TRUE if the specified feature from the specified layer is selected.

Syntax	is_selected(layer, feature)
Arguments	• **layer** - The layer (its ID or name) on which the selection will be checked. • **feature** - The feature which should be checked for selection.
Examples	• `is_selected( 'streets', get_feature('streets', 'name', "street_name"))` → TRUE if the current building's street is selected (assuming the building layer has a field named 'street_name' and the 'streets' layer has a field called 'name' with unique values). • `is_selected( 'streets', get_feature_by_id('streets', 1))` → TRUE if the feature with the id 1 on the "streets" layer is selected.

maptip

Returns the maptip for a given feature in a layer. The expression is evaluated by default. Can be used with zero, one or more arguments, see below for details.

No parameters

If called with no parameters, the function will evaluate the maptip of the current feature in the current layer.

Syntax	maptip()
Examples	• `maptip()` → The maptip of the current feature in the current layer.

One 'feature' parameter

If called with a 'feature' parameter only, the function will evaluate the specified feature from the current layer.

Syntax	maptip(feature)
Arguments	• **feature** - The feature which should be evaluated.
Examples	• `maptip(@atlas_feature)` → The maptip of the current atlas feature.

Layer and feature parameters

If the function is called with both a layer and a feature, it will evaluate the specified feature from the specified layer.

Syntax	maptip(layer, feature, [evaluate=true]) [] marks optional arguments
Arguments	• **layer** - The layer (or its ID or name) • **feature** - The feature which should be evaluated. • **evaluate** - If the expression must be evaluated. If false, the expression will be returned as a string literal only (which could potentially be later evaluated using the 'eval_template' function).
Examples	• `maptip('streets', get_feature_by_id('streets', 1))` → The maptip of the feature with the ID 1 on the layer 'streets'. • `maptip('a_layer_id', @feature, 'False')` → The maptip of the given feature not evaluated.

num_selected

Returns the number of selected features on a given layer. By default works on the layer on which the expression is evaluated.

Syntax	num_selected([layer=current layer]) [] marks optional arguments
Arguments	• **layer** - The layer (or its id or name) on which the selection will be checked.
Examples	• `num_selected()` → The number of selected features on the current layer. • `num_selected('streets')` → The number of selected features on the layer streets

represent_attributes

Returns a map with the attribute names as keys and the configured representation values as values. The representation value for the attributes depends on the configured widget type for each attribute. Can be used with zero, one or more arguments, see below for details.

No parameters

If called with no parameters, the function will return the representation of the attributes of the current feature in the current layer.

Syntax	represent_attributes()
Examples	• `represent_attributes()` → The representation of the attributes for the current feature.

One 'feature' parameter

If called with a 'feature' parameter only, the function will return the representation of the attributes of the specified feature from the current layer.

Syntax	represent_attributes(feature)
Arguments	• **feature** - The feature which should be evaluated.
Examples	• `represent_attributes(@atlas_feature)` → The representation of the attributes for the specified feature from the current layer.

Layer and feature parameters

If called with a 'layer' and a 'feature' parameter, the function will return the representation of the attributes of the specified feature from the specified layer.

Syntax	represent_attributes(layer, feature)
Arguments	• **layer** - The layer (or its ID or name). • **feature** - The feature which should be evaluated.
Examples	• `represent_attributes('atlas_layer', @atlas_feature)` → The representation of the attributes for the specified feature from the specified layer.

Further reading: *represent_value*

represent_value

Returns the configured representation value for a field value. It depends on the configured widget type. Often, this is useful for 'Value Map' widgets.

Syntax	represent_value(value, [fieldName]) [] marks optional arguments
Arguments	• **value** - The value which should be resolved. Most likely a field. • **fieldName** - The field name for which the widget configuration should be loaded.
Examples	• `represent_value("field_with_value_map")` → Description for value • `represent_value('static value', 'field_name')` → Description for static value

Further reading: *widget types*, *represent_attributes*

sqlite_fetch_and_increment

Manage autoincrementing values in sqlite databases.

SQlite default values can only be applied on insert and not prefetched.

This makes it impossible to acquire an incremented primary key via AUTO_INCREMENT before creating the row in the database. Sidenote: with postgres, this works via the option *evaluate default values*.

When adding new features with relations, it is really nice to be able to already add children for a parent, while the parents form is still open and hence the parent feature uncommitted.

To get around this limitation, this function can be used to manage sequence values in a separate table on sqlite based formats like gpkg.

The sequence table will be filtered for a sequence id (filter_attribute and filter_value) and the current value of the id_field will be incremented by 1 and the incremented value returned.

If additional columns require values to be specified, the default_values map can be used for this purpose.

Note

This function modifies the target sqlite table. It is intended for usage with default value configurations for attributes.

When the database parameter is a layer and the layer is in transaction mode, the value will only be retrieved once during the lifetime of a transaction and cached and incremented. This makes it unsafe to work on the same database from several processes in parallel.

Syntax	sqlite_fetch_and_increment(database, table, id_field, filter_attribute, filter_value, [default_values]) [] marks optional arguments
Arguments	• **database** - Path to the sqlite file or geopackage layer • **table** - Name of the table that manages the sequences • **id_field** - Name of the field that contains the current value • **filter_attribute** - Name the field that contains a unique identifier for this sequence. Must have a UNIQUE index. • **filter_value** - Name of the sequence to use. • **default_values** - Map with default values for additional columns on the table. The values need to be fully quoted. Functions are allowed.
Examples	• `sqlite_fetch_and_increment(@layer, 'sequence_table', 'last_unique_id', 'sequence_id', 'global', map('last_change', 'date(''now'')', 'user', '''' || @user_account_name || '''')) → 0` • `sqlite_fetch_and_increment(layer_property(@layer, 'path'), 'sequence_table', 'last_unique_id', 'sequence_id', 'global', map('last_change', 'date(''now'')', 'user', '''' || @user_account_name || '''')) → 0`

Further reading: *Data Sources Properties*, *Setting relations between multiple layers*

uuid

Generates a Universally Unique Identifier (UUID) for each row using the Qt QUuid::createUuid method.

Syntax	uuid([format='WithBraces']) [] marks optional arguments
Arguments	• **format** - The format, as the UUID will be formatted. 'WithBraces', 'WithoutBraces' or 'Id128'.
Examples	• `uuid()` → '{0bd2f60f-f157-4a6d-96af-d4ba4cb366a1}' • `uuid('WithoutBraces')` → '0bd2f60f-f157-4a6d-96af-d4ba4cb366a1' • `uuid('Id128')` → '0bd2f60ff1574a6d96afd4ba4cb366a1'

9.2.24 Relations

This group contains the list of the *relations* available in the current project, with their description. It provides a quick access to the relation ID for writing an expression (with e.g. the *relation_aggregate* function) or customizing a form.

9.2.25 Sensors Functions

This group contains functions to interact with sensors.

sensor_data

Returns the last captured value (or values as a map for sensors which report multiple values) from a registered sensor.

Syntax	sensor_data(name, [expiration]) [] marks optional arguments
Arguments	• **name** - the sensor name • **expiration** - maximum millisecond since last captured value allowed
Examples	• `sensor_data('geiger_1')` → '2000'

9.2.26 String Functions

This group contains functions that operate on strings (e.g., that replace, convert to upper case).

ascii

Returns the unicode code associated with the first character of a string.

Syntax	ascii(string)
Arguments	• **string** - the string to convert to unicode code
Examples	• `ascii('Q')` → 81

char

Returns the character associated with a unicode code.

Syntax	char(code)
Arguments	• **code** - a unicode code number
Examples	• `char(81)` → 'Q'

concat

Concatenates several strings to one. NULL values are converted to empty strings. Other values (like numbers) are converted to strings.

Syntax	concat(string1, string2, …)
Arguments	• **string** - a string value
Examples	• `concat('sun', 'set')` → 'sunset' • `concat('a','b','c','d','e')` → 'abcde' • `concat('Anno ', 1984)` → 'Anno 1984' • `concat('The Wall', NULL)` → 'The Wall'

About fields concatenation

You can also concatenate strings or field values using either `||` or + operators, with some special characteristics:

- The + operator also means sum up expression, so if you have an integer (field or numeric value) operand, this can be error prone and you better use the others:

```
'My feature id is: ' + "gid" => triggers an error as gid returns an integer
```

- When any of the arguments is a NULL value, either `||` or + will return a NULL value. To return the other arguments regardless the NULL value, you may want to use the `concat` or `array_to_string` function:

```
'My feature id is: ' + NULL ==> NULL
'My feature id is: ' || NULL => NULL
concat('My feature id is: ', NULL) => 'My feature id is: '
array_to_string( array('My feature id is: ', NULL) ) => 'My feature id is: '
```

further reading: *array_to_string*, ||, +

format

Format a string using supplied arguments.

Syntax	format(string, arg1, arg2, …)
Arguments	• **string** - A string with placeholders %1, %2, etc., for the arguments. Placeholders can be repeated. The lowest numbered placeholder is replaced by arg1, the next by arg2, etc. • **arg** - any type. Any number of arguments.
Examples	• `format('This %1 a %2','is', 'test')` → 'This is a test' • `format('This is %2','a bit unexpected but 2 is lowest number in string','normal')` → 'This is a bit unexpected but 2 is lowest number in string'

format_date

Formats a date type or string into a custom string format. Uses Qt date/time format strings. See QDateTime::toString.

Syntax	format_date(datetime, format, [language]) [] marks optional arguments
Arguments	• **datetime** - date, time or datetime value • **format** - String template used to format the string.

Expression	Output
d	the day as number without a leading zero (1 to 31)
dd	the day as number with a leading zero (01 to 31)
ddd	the abbreviated localized day name (e.g. 'Mon' to 'Sun')
dddd	the long localized day name (e.g. 'Monday' to 'Sunday')
M	the month as number without a leading zero (1-12)
MM	the month as number with a leading zero (01-12)
MMM	the abbreviated localized month name (e.g. 'Jan' to 'Dec')
MMMM	the long localized month name (e.g. 'January' to 'December')
yy	the year as two digit number (00-99)
yyyy	the year as four digit number

These expressions may be used for the time part of the format string:

Expression	Output
h	the hour without a leading zero (0 to 23 or 1 to 12 if AM/PM display)
hh	the hour with a leading zero (00 to 23 or 01 to 12 if AM/PM display)
H	the hour without a leading zero (0 to 23, even with AM/PM display)
HH	the hour with a leading zero (00 to 23, even with AM/PM display)
m	the minute without a leading zero (0 to 59)
mm	the minute with a leading zero (00 to 59)
s	the second without a leading zero (0 to 59)
ss	the second with a leading zero (00 to 59)
z	the milliseconds without trailing zeroes (0 to 999)
zzz	the milliseconds with trailing zeroes (000 to 999)
AP or A	interpret as an AM/PM time. *AP* must be either 'AM' or 'PM'.
ap or a	Interpret as an AM/PM time. *ap* must be either 'am' or 'pm'.

- **language** - language (lowercase, two- or three-letter, ISO 639 language code) used to format the date into a custom string. By default the current QGIS user locale is used.

Examples

- `format_date('2012-05-15','dd.MM.yyyy')` → '15.05.2012'
- `format_date('2012-05-15','d MMMM yyyy','fr')` → '15 mai 2012'
- `format_date('2012-05-15','dddd')` → 'Tuesday', if the current locale is an English variant
- `format_date('2012-05-15 13:54:20','dd.MM.yy')` → '15.05.12'
- `format_date('13:54:20','hh:mm AP')` → '01:54 PM'

format_number

Returns a number formatted with the locale separator for thousands. By default the current QGIS user locale is used. Also truncates the decimal places to the number of supplied places.

Syntax	format_number(number, [places=0], [language], [omit_group_separators=false], [trim_trailing_zeroes=false]) [] marks optional arguments
Arguments	• **number** - number to be formatted • **places** - integer representing the number of decimal places to truncate the string to. • **language** - language (lowercase, two- or three-letter, ISO 639 language code) used to format the number into a string. By default the current QGIS user locale is used. • **omit_group_separators** - if set to true then group separators will not be included in the string • **trim_trailing_zeroes** - if set to true then trailing zeros following the decimal point will be trimmed from the string
Examples	• `format_number(10000000.332,2)` → '10,000,000.33' if e.g. the current locale is an English variant • `format_number(10000000.332,2,'fr')` → '10 000 000,33'

left

Returns a substring that contains the *n* leftmost characters of the string.

Syntax	left(string, length)
Arguments	• **string** - a string • **length** - integer. The number of characters from the left of the string to return.
Examples	• `left('Hello World',5)` → 'Hello'

Further reading: *right*

length

Returns the number of characters in a string or the length of a geometry linestring.

String variant

Returns the number of characters in a string.

Syntax	length(string)
Arguments	• **string** - string to count length of
Examples	• `length('hello')` → 5

Geometry variant

Calculate the length of a geometry line object. Calculations are always planimetric in the Spatial Reference System (SRS) of this geometry, and the units of the returned length will match the units for the SRS. This differs from the calculations performed by the $length function, which will perform ellipsoidal calculations based on the project's ellipsoid and distance unit settings.

Syntax	length(geometry)
Arguments	• **geometry** - line geometry object
Examples	• `length(@geometry)` → length of the current feature's geometry • `length(geom_from_wkt('LINESTRING(0 0, 4 0)'))` → 4.0

lower

Converts a string to lower case letters.

Syntax	lower(string)
Arguments	• **string** - the string to convert to lower case
Examples	• `lower('HELLO World')` → 'hello world'

Further reading: *upper*, *title*

lpad

Returns a string padded on the left to the specified width, using a fill character. If the target width is smaller than the string's length, the string is truncated.

Syntax	lpad(string, width, fill)
Arguments	• **string** - string to pad • **width** - length of new string • **fill** - character to pad the remaining space with
Examples	• `lpad('Hello', 10, 'x')` → 'xxxxxHello' • `lpad('Hello', 3, 'x')` → 'Hel'

Further reading: *rpad*

ltrim

Removes the longest string containing only the specified characters (a space by default) from the start of string.

Syntax	ltrim(string, [characters=' ']) [] marks optional arguments
Arguments	• **string** - string to trim • **characters** - characters to trim
Examples	• `ltrim(' hello world ')` → 'hello world ' • `ltrim('zzzytest', 'xyz')` → 'test'

Further reading: *rtrim*, *trim*

regexp_match

Return the first matching position matching a regular expression within an unicode string, or 0 if the substring is not found.

Syntax	regexp_match(input_string, regex)
Arguments	• **input_string** - the string to test against the regular expression • **regex** - The regular expression to test against. Backslash characters must be double escaped (e.g., "\\s" to match a white space character or "\\b" to match a word boundary).
Examples	• `regexp_match('QGIS ROCKS','\\sROCKS')` → 5 • `regexp_match('Budač','udač\\b')` → 2

Further reading: *strpos*

regexp_replace

Returns a string with the supplied regular expression replaced.

Syntax	regexp_replace(input_string, regex, replacement)
Arguments	• **input_string** - the string to replace matches in • **regex** - The regular expression to replace. Backslash characters must be double escaped (e.g., "\\s" to match a white space character). • **replacement** - The string that will replace any matching occurrences of the supplied regular expression. Captured groups can be inserted into the replacement string using \\1, \\2, etc.
Examples	• `regexp_replace('QGIS SHOULD ROCK','\\sSHOULD\\s',' DOES ')` → 'QGIS DOES ROCK' • `regexp_replace('ABC123','\\d+','')` → 'ABC' • `regexp_replace('my name is John','(.*) is (.*)','\\2 is \\1')` → 'John is my name'

Further reading: *replace*, *array_replace*

regexp_substr

Returns the portion of a string which matches a supplied regular expression.

Syntax	regexp_substr(input_string, regex)
Arguments	• **input_string** - the string to find matches in • **regex** - The regular expression to match against. Backslash characters must be double escaped (e.g., "\\s" to match a white space character).
Examples	• `regexp_substr('abc123','(\\d+)')` → '123'

Further reading: *substr*, *regexp_matches*

repeat

Repeats a string a specified number of times.

Syntax	repeat(string, number)
Arguments	• **string** - string to repeat • **number** - number of times to repeat string
Examples	• `repeat('Hello', 3)` → 'HelloHelloHello'

replace

Returns a string with the supplied string, array, or map of strings replaced.

String & array variant

Returns a string with the supplied string or array of strings replaced by a string or an array of strings.

Syntax	replace(string, before, after)
Arguments	• **string** - the input string • **before** - the string or array of strings to replace • **after** - the string or array of strings to use as a replacement
Examples	• `replace('QGIS SHOULD ROCK','SHOULD','DOES')` → 'QGIS DOES ROCK' • `replace('QGIS ABC',array('A','B','C'),array('X','Y','Z'))` → 'QGIS XYZ' • `replace('QGIS',array('Q','S'),'')` → 'GI'

Map variant

Returns a string with the supplied map keys replaced by paired values. Longer map keys are evaluated first.

Syntax	replace(string, map)
Arguments	• **string** - the input string • **map** - the map containing keys and values
Examples	• `replace('APP SHOULD ROCK',map('APP','QGIS','SHOULD','DOES'))` → 'QGIS DOES ROCK' • `replace('forty two',map('for','4','two','2','forty two','42'))` → '42'

Further reading: *regexp_replace*, *array_replace*

reverse

Reverses the direction of a line string or reverses a string of text.

String variant

Reverses the order of characters in a string.

Syntax	reverse(string)
Arguments	• **string** - string to reverse
Examples	• `reverse('hello')` → 'olleh'

Geometry variant

Reverses the direction of a line string by reversing the order of its vertices.

Syntax	reverse(geometry)
Arguments	• **geometry** - a geometry
Examples	• `geom_to_wkt(reverse(geom_from_wkt('LINESTRING(0 0, 1 1, 2 2)')))` → 'LINESTRING(2 2, 1 1, 0 0)'

right

Returns a substring that contains the *n* rightmost characters of the string.

Syntax	right(string, length)
Arguments	• **string** - a string • **length** - integer. The number of characters from the right of the string to return.
Examples	• `right('Hello World',5)` → 'World'

Further reading: *left*

rpad

Returns a string padded on the right to the specified width, using a fill character. If the target width is smaller than the string's length, the string is truncated.

Syntax	rpad(string, width, fill)
Arguments	• **string** - string to pad • **width** - length of new string • **fill** - character to pad the remaining space with
Examples	• `rpad('Hello', 10, 'x')` → 'Helloxxxxx' • `rpad('Hello', 3, 'x')` → 'Hel'

Further reading: *lpad*

rtrim

Removes the longest string containing only the specified characters (a space by default) from the end of string.

Syntax	rtrim(string, [characters=' ']) [] marks optional arguments
Arguments	• **string** - string to trim • **characters** - characters to trim
Examples	• `rtrim(' hello world ')` → ' hello world' • `rtrim('testxxzx', 'xyz')` → 'test'

Further reading: *ltrim*, *trim*

strpos

Return the first matching position of a substring within another string, or 0 if the substring is not found.

Syntax	strpos(haystack, needle)
Arguments	• **haystack** - string that is to be searched • **needle** - string to search for
Examples	• `strpos('HELLO WORLD','WORLD')` → 7 • `strpos('HELLO WORLD','GOODBYE')` → 0

Further reading: *regexp_match*

substr

Returns a part of a string.

Syntax	substr(string, start, [length]) [] marks optional arguments
Arguments	• **string** - the full input string • **start** - integer representing start position to extract beginning with 1; if start is negative, the return string will begin at the end of the string minus the start value • **length** - integer representing length of string to extract; if length is negative, the return string will omit the given length of characters from the end of the string
Examples	• `substr('HELLO WORLD',3,5)` → 'LLO W' • `substr('HELLO WORLD',6)` → ' WORLD' • `substr('HELLO WORLD',-5)` → 'WORLD' • `substr('HELLO',3,-1)` → 'LL' • `substr('HELLO WORLD',-5,2)` → 'WO' • `substr('HELLO WORLD',-5,-1)` → 'WORL'

Further reading: *regexp_substr*, *regexp_matches*

title

Converts all words of a string to title case (all words lower case with leading capital letter).

Syntax	title(string)
Arguments	• **string** - the string to convert to title case
Examples	• `title('hello WOrld')` → 'Hello World'

Further reading: *lower*, *upper*

to_string

Converts a number to string. The conversion is not locale-aware, see 'format_number' for a locale-aware alternative.

Syntax	to_string(number)
Arguments	• **number** - Integer or real value. The number to convert to string.
Examples	• `to_string(1.23)` → '1.23'

Further reading: *format_number*

trim

Removes all leading and trailing whitespace (spaces, tabs, etc) from a string.

Syntax	trim(string)
Arguments	• **string** - string to trim
Examples	• `trim(' hello world ')` → 'hello world'

Further reading: *ltrim*, *rtrim*

upper

Converts a string to upper case letters.

Syntax	upper(string)
Arguments	• **string** - the string to convert to upper case
Examples	• `upper('hello WOrld')` → 'HELLO WORLD'

Further reading: *lower*, *title*

wordwrap

Returns a string wrapped to a maximum/minimum number of characters.

Syntax	wordwrap(string, wrap_length, [delimiter_string]) [] marks optional arguments
Arguments	• **string** - the string to be wrapped • **wrap_length** - an integer. If wrap_length is positive the number represents the ideal maximum number of characters to wrap; if negative, the number represents the minimum number of characters to wrap. • **delimiter_string** - Optional delimiter string to wrap to a new line.
Examples	• `wordwrap('UNIVERSITY OF QGIS',13)` → 'UNIVERSITY OF QGIS' • `wordwrap('UNIVERSITY OF QGIS',-3)` → 'UNIVERSITY OF QGIS'

9.2.27 User Expressions

This group contains the expressions saved as *user expressions*.

9.2.28 Variables

This group contains dynamic variables related to the application, the project file and other settings. The availability of variables depends on the context:

- from the  Select by expression dialog
- from the  Field calculator dialog
- from the layer properties dialog
- from the print layout

To use these variables in an expression, they should be preceded by the `@` character (e.g, `@row_number`).

Variable	Description
algorithm_id	The unique ID of an algorithm
animation_end_time	End of the animation's overall temporal time range (as a datetime value)
animation_interval	Duration of the animation's overall temporal time range (as an interval value)
animation_start_time	Start of the animation's overall temporal time range (as a datetime value)
atlas_feature	The current atlas feature (as feature object)
atlas_featureid	The current atlas feature ID
atlas_featurenumber	The current atlas feature number in the layout
atlas_filename	The current atlas file name
atlas_geometry	The current atlas feature geometry
atlas_layerid	The current atlas coverage layer ID
atlas_layername	The current atlas coverage layer name
atlas_pagename	The current atlas page name
atlas_totalfeatures	The total number of features in atlas
band	The number of the band in the raster layer
band_description	The description of the band in the raster layer
band_name	The name of the band in the raster layer
canvas_cursor_point	The last cursor position on the canvas in the project's geographical coordinates
cluster_color	The color of symbols within a cluster, or NULL if symbols have mixed colors
cluster_size	The number of symbols contained within a cluster
current_feature	The feature currently being edited in the attribute form or table row
current_geometry	The geometry of the feature currently being edited in the form or the table row
current_parent_feature	represents the feature currently being edited in the parent form. Only usable in an embedded form context.
current_parent_geometry	represents the geometry of the feature currently being edited in the parent form. Only usable in an embedded form context.
form_mode	What the form is used for, like AddFeatureMode, SingleEditMode, MultiEditMode, SearchMode, AggregateSearchMode or IdentifyMode as string.
feature	The current feature being evaluated. This can be used with the 'attribute' function to evaluate attribute values from the current feature.
frame_duration	Temporal duration of each animation frame (as an interval value)
frame_number	Current frame number during animation playback
frame_rate	Number of frames per second during animation playback
fullextent_maxx	Maximum x value from full canvas extent (including all layers)
fullextent_maxy	Maximum y value from full canvas extent (including all layers)
fullextent_minx	Minimum x value from full canvas extent (including all layers)
fullextent_miny	Minimum y value from full canvas extent (including all layers)
geometry	The geometry of the current feature being evaluated

continues on next page

Table 9.1 – continued from previous page

Variable	Description
geometry_part_count	The number of parts in rendered feature's geometry
geometry_part_num	The current geometry part number for feature being rendered
geometry_point_count	The number of points in the rendered geometry's part
geometry_point_num	The current point number in the rendered geometry's part
geometry_ring_num	Current geometry ring number for feature being rendered (for polygon features only). The exterior ring has a value of 0.
grid_axis	The current grid annotation axis (eg, 'x' for longitude, 'y' for latitude)
grid_number	The current grid annotation value
id	The ID of the current feature being evaluated
item_id	The layout item user ID (not necessarily unique)
item_uuid	The layout item unique ID
layer	The current layer
layer_crs	The Coordinate Reference System Authority ID of the current layer
layer_crs_ellipsoid	The ellipsoid Authority ID of the current layer CRS
layer_cursor_point	Point geometry under the mouse position in map canvas (or the GetFeatureInfo position in context of QGIS Server), in active layer's CRS
layer_id	The ID of current layer
layer_ids	The IDs of all the map layers in the current project as a list
layer_name	The name of current layer
layer_vertical_crs	The Identifier for the vertical coordinate reference system of the layer (e.g., 'EPSG:5703')
layer_vertical_crs_definition	The full definition of the vertical Coordinate reference system of the layer
layer_vertical_crs_descriptio	The name of the vertical Coordinate reference system of the layer
layer_vertical_crs_wkt	The WKT definition of the vertical Coordinate reference system of the current layer
layers	All the map layers in the current project as a list
layout_dpi	The composition resolution (DPI)
layout_name	The layout name
layout_numpages	The number of pages in the layout
layout_page	The page number of the current item in the layout
layout_pageheight	The active page height in the layout (in mm for standard paper sizes, or whatever unit was used for custom paper size)
layout_pageoffsets	Array of Y coordinate of the top of each page. Allows to dynamically position items on pages in a context where page sizes may change
layout_pagewidth	The active page width in the layout (in mm for standard paper sizes, or whatever unit was used for custom paper size)
legend_column_count	The number of columns in the legend
legend_filter_by_map	Indicates if the content of the legend is filtered by the map
legend_filter_out_atlas	Indicates if the atlas is filtered out of the legend
legend_split_layers	Indicates if layers can be split in the legend
legend_title	The title of the legend
legend_wrap_string	The character(s) used to wrap the legend text
map_crs	The Coordinate reference system of the current map
map_crs_acronym	The acronym of the Coordinate reference system of the current map
map_crs_definition	The full definition of the Coordinate reference system of the current map
map_crs_description	The name of the Coordinate reference system of the current map
map_crs_ellipsoid	The acronym of the ellipsoid of the Coordinate reference system of the current map
map_crs_proj4	The Proj4 definition of the Coordinate reference system of the current map
map_crs_projection	The descriptive name of the projection method used by the Coordinate reference system of the map (e.g. 'Albers Equal Area')
map_crs_wkt	The WKT definition of the Coordinate reference system of the current map
map_end_time	The end of the map's temporal time range (as a datetime value)
map_extent	The geometry representing the current extent of the map
map_extent_center	The point feature at the center of the map
map_extent_height	The current height of the map
map_extent_width	The current width of the map

continues on next page

Table 9.1 – continued from previous page

Variable	Description
map_id	The ID of current map destination. This will be 'canvas' for canvas renders, and the item ID for layout map renders
map_interval	The duration of the map's temporal time range (as an interval value)
map_layer_ids	The list of map layer IDs visible in the map
map_layers	The list of map layers visible in the map
map_rotation	The current rotation of the map
map_scale	The current scale of the map
map_start_time	The start of the map's temporal time range (as a datetime value)
map_units	The units of map measurements
map_z_range_lower	Lower elevation of the map's elevation range
map_z_range_upper	Upper elevation of the map's elevation range
model_path	Full path (including file name) of current model (or project path if model is embedded in a project).
model_folder	Folder containing current model (or project folder if model is embedded in a project).
model_name	Name of current model
model_group	Group for current model
notification_message	Content of the notification message sent by the provider (available only for actions triggered by provider notifications).
parent	Refers to the current feature in the parent layer, providing access to its attributes and geometry when filtering an *aggregate* function
plot_axis	The associated plot axis, e.g. 'x' or 'y'.
plot_axis_value	The current value for the plot axis.
project_abstract	The project abstract, taken from project metadata
project_area_units	The area unit for the current project, used when calculating areas of geometries
project_author	The project author, taken from project metadata
project_basename	The basename of current project's filename (without path and extension)
project_creation_date	The project creation date, taken from project metadata
project_crs	Identifier for the coordinate reference system of the project (e.g., 'EPSG:4326')
project_crs_arconym	The acronym of the Coordinate reference system of the project
project_crs_definition	The full definition of the Coordinate reference system of the project
project_crs_description	The description of the Coordinate reference system of the project
project_crs_ellipsoid	The ellipsoid of the Coordinate reference system of the project
project_crs_proj4	The Proj4 representation of the Coordinate reference system of the project
project_crs_wkt	The WKT (well known text) representation of the coordinate reference system of the project
project_distance_units	The distance unit for the current project, used when calculating lengths of geometries and distances
project_ellipsoid	The name of the ellipsoid of the current project, used when calculating geodetic areas or lengths of geometries
project_filename	The filename of the current project
project_folder	The folder of the current project
project_home	The home path of the current project
project_identifier	The project identifier, taken from the project's metadata
project_keywords	The project keywords, taken from the project's metadata
project_last_saved	Date/time when project was last saved.
project_path	The full path (including file name) of the current project
project_title	The title of current project
project_units	The units of the project's CRS
project_vertical_crs	Identifier for the vertical Coordinate reference system of the project (e.g., 'EPSG:5703')
project_vertical_crs_definitic	The full definition of the vertical coordinate reference system of the project
project_vertical_crs_descript	The description of the vertical coordinate reference system of the project
project_vertical_crs_wkt	The WKT (well known text) representation of the vertical coordinate reference system of the project

continues on next page

Table 9.1 – continued from previous page

Variable	Description
qgis_locale	The current language of QGIS
qgis_os_name	The current Operating system name, eg 'windows', 'linux' or 'osx'
qgis_platform	The QGIS platform, eg 'desktop' or 'server'
qgis_release_name	The current QGIS release name
qgis_short_version	The current QGIS version short string
qgis_version	The current QGIS version string
qgis_version_no	The current QGIS version number
row_number	Stores the number of the current row
snapping_results	Gives access to snapping results while digitizing a feature (only available in add feature)
scale_value	The current scale bar distance value
selected_file_path	Selected file path from file widget selector when uploading a file with an external storage system
symbol_angle	The angle of the symbol used to render the feature (valid for marker symbols only)
symbol_color	The color of the symbol used to render the feature
symbol_count	The number of features represented by the symbol (in the layout legend)
symbol_frame	The frame number (for animated symbols only)
symbol_id	The Internal ID of the symbol (in the layout legend)
symbol_label	The label for the symbol (either a user defined label or the default autogenerated label - in the layout legend)
symbol_layer_count	Total number of symbol layers in the symbol
symbol_layer_index	Current symbol layer index
symbol_marker_column	Column number for marker (valid for point pattern fills only).
symbol_marker_row	Row number for marker (valid for point pattern fills only).
user_account_name	The current user's operating system account name
user_full_name	The current user's operating system user name
value	The current value
vector_tile_zoom	Exact vector tile zoom level of the map that is being rendered (derived from the current map scale). Normally in interval [0, 20]. Unlike @zoom_level, this variable is a floating point value which can be used to interpolate values between two integer zoom levels.
with_variable	Allows setting a variable for usage within an expression and avoid recalculating the same value repeatedly
zoom_level	Vector tile zoom level of the map that is being rendered (derived from the current map scale). Normally in interval [0, 20].

Some examples:

- Return the X coordinate of a map item center in layout:

```
x( map_get( item_variables( 'map1'), 'map_extent_center' ) )
```

- Return, for each feature in the current layer, the number of overlapping airport features:

```
aggregate( layer:='airport', aggregate:='count', expression:="code",
              filter:=intersects( $geometry, geometry( @parent ) ) )
```

- Get the object_id of the first snapped point of a line:

```
with_variable(
  'first_snapped_point',
  array_first( @snapping_results ),
  attribute(
    get_feature_by_id(
      map_get( @first_snapped_point, 'layer' ),
      map_get( @first_snapped_point, 'feature_id' )
```

(continues on next page)

(continued from previous page)

```
    ),
    'object_id'
  )
)
```

9.2.29 Recent Functions

This group contains recently used functions. Depending on the context of its usage (feature selection, field calculator, generic), recently applied expressions are added to the corresponding list (up to ten expressions), sorted from more to less recent. This makes it easy to quickly retrieve and reapply previously used expressions.

CHAPTER

TEN

THE STYLE LIBRARY

10.1 The Style Manager

10.1.1 The Style Manager dialog

The *Style Manager* is the place where you can manage and create generic style items. These are symbols, color ramps, text formats or label settings that can be used to symbolize features, layers or print layouts. They are stored in the `symbology-style.db` database under the active *user profile* and shared with all the project files opened with that profile. Style items can also be shared with others thanks to the export/import capabilities of the *Style Manager* dialog.

You can open that modeless dialog either:

- from the *Settings* ► *Style Manager…* menu
- with the Style Manager button from the Project toolbar
- or with the Style Manager button from a vector *Layer Properties* ► menu (while *configuring a symbol* or *formatting a text*).

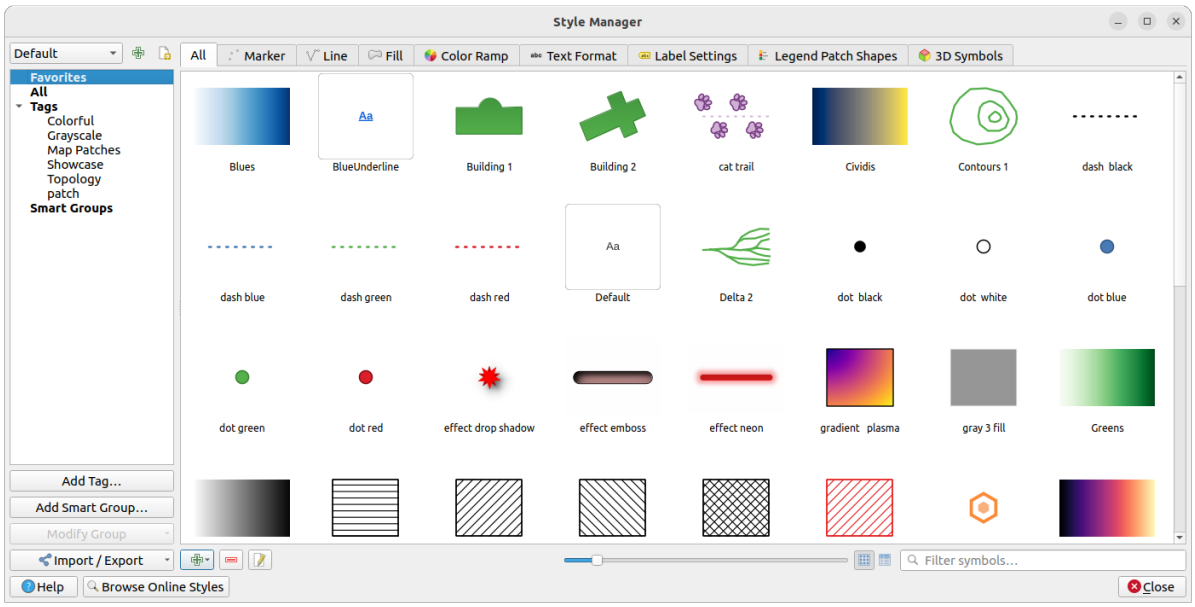


Fig. 10.1: The Style Manager

Organizing style items

In the upper left corner of the *Style Manager* dialog, within the drop-down menu, you can choose the style database you want to connect to. If you choose *Default* you will be connected to the default style database where you can find all available default style items and the ones that you saved in this database. Choosing *Project Styles* will connect you to the Project Style database where you can find only symbols that you saved in this database. If you created more style databases, they will be listed in the drop-down menu. There are also options to Add existing style database to project and Create new style database (see more *Style database*).

For each style database, you can organize the elements into different categories, listed in the panel on the left:

- **Favorites**: displayed by default when configuring an item, it shows an extensible set of items;
- **All**: lists all the available items for the active type;
- **Tags**: shows a list of labels you can use to identify the items. An item can be tagged more than once. Select a tag in the list and the tabs are updated to show only their items that belong to it. To create a new tag you could later attach to a set of items, use the *Add Tag…* button or select the *Add Tag…* from any tag contextual menu;
- **Smart Group**: a smart group dynamically fetches its symbols according to conditions set (see eg, Fig. 10.2). Click the *Add Smart Group…* button to create smart groups. The dialog box allows you to enter an expression to filter the items to select (has a particular tag, have a string in its name, etc.). Any symbol, color ramp, text format or label setting that satisfies the entered condition(s) is automatically added to the smart group.

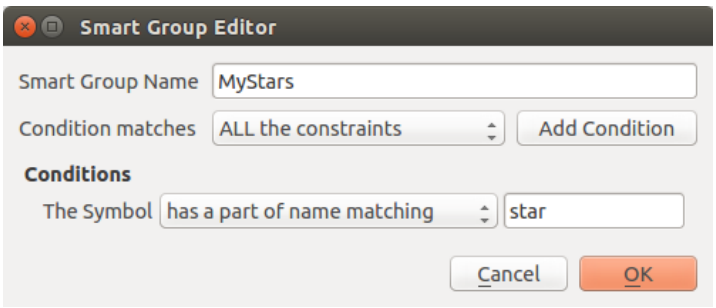


Fig. 10.2: Creating a Smart Group

Tags and smart groups are not mutually exclusive: they are simply two different ways to organize your style elements. Unlike the smart groups that automatically fetch their belonged items based on the input constraints, tags are filled by the user. To edit any of those categories, you can either:

- select the items, right-click and choose *Add to Tag* ► and then select the tag name or create a new tag;
- select the tag and press *Modify group…* ► *Attach Selected Tag to Symbols*. A checkbox appears next to each item to help you select or deselect it. When selection is finished, press *Modify group…* ► *Finish Tagging*.
- select the smart group, press *Modify group…* ► *Edit smart group…* and configure a new set of constraints in the *Smart Group Editor* dialog. This option is also available in the contextual menu of the smart group.

To remove a tag or a smart group, right-click on it and select the *Remove* button. Note that this does not delete the items grouped in the category.

The *Style Manager* dialog displays in its center a frame with previewed items organized into tabs:

- *All* for a complete collection of point, linear and surface symbols and label settings as well as predefined color ramps and text formats;
- *Marker* for point symbols only;
- *Line* for linear symbols only;

- *Fill* for surface symbols only;
- *Color ramp*;
- *Text format* to manage *text formats*, which store the font, color, buffers, shadows, and backgrounds of texts (i.e. all the formatting parts of the label settings, which for instance can be used in layouts);
- *Label settings* to manage *label settings*, which include the text formats and some layer-type specific settings such as label placement, priority, callouts, rendering…
- *Legend Patch Shapes* to manage custom legend patch shapes, which include *Marker*, *Line* and *Fill* geometries.
- *3D Symbols* to configure symbols with *3D properties* (extrusion, shading, altitude, …) for the features to render in a *3D Map view*

You can arrange the Styles in *Icon View* or in *List View* on the bottom right side. In both views the tooltip shows a larger instance of the style. The thumbnail size slider at the left of the icons helps you adjust the actual thumbnail sizes in the dialog, for a better preview of the symbols!

Adding, editing or removing an item

As seen earlier, style elements are listed under different tabs whose contents depend on the active category (tag, smart group, favorites…). When a tab is enabled, you can:

- Add new items: press the Add item button and configure the item following *symbols*, *color ramps* or *text format and label* builder description.
- Modify an existing item: select an item and press Edit item button and configure as mentioned above.
- Delete existing items: to delete an element you no longer need, select it and click Remove item (also available through right-click). The item will be deleted from the local database.

Note that the *All* tab provides access to these options for every type of item.

Right-clicking over a selection of items also allows you to:

- *Add to Favorites*;
- *Remove from Favorites*;
- *Add to Tag* ► and select the appropriate tag or create a new one to use; the currently assigned tags are checked;
- *Clear Tags*: detaching the symbols from any tag;
- *Remove Item(s)*;
- *Edit Item*: applies to the item you right-click over;
- *Copy Item*;
- *Paste Item* …: pasting to one of the categories of the style manager or elsewhere in QGIS (symbol or color buttons)
- *Export Selected Symbol(s) as PNG…* (only available with symbols);
- *Export Selected Symbol(s) as SVG…* (only available with symbols);

Sharing style items

The *Import/Export* tool, at the left bottom of the Style Manager dialog, offers options to easily share symbols, color ramps, text formats and label settings with others. These options are also available through right-click over the items.

Exporting items

You can export a set of items to an `.XML` file:

1. Expand the *Import/Export* drop-down menu and select *Export Item(s)...*
2. Choose the items you'd like to integrate. Selection can be done with the mouse or using a tag or a group previously set.
3. Press *Export* when ready. You'll be prompted to indicate the destination of the saved file. The XML format generates a single file containing all the selected items. This file can then be imported in another user's style library.

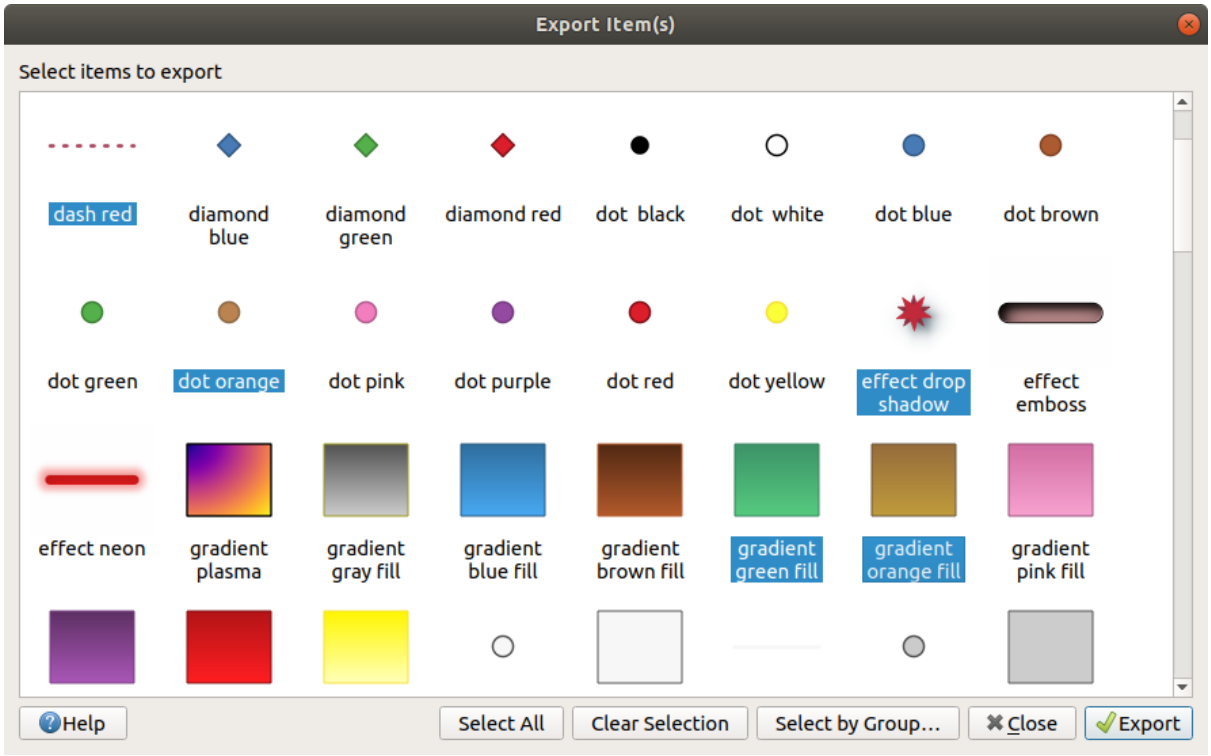


Fig. 10.3: Exporting style items

When symbols are selected, you can also export them to `.PNG` or `.SVG`. Exporting to `.PNG` or `.SVG` (both not available for other style item types) creates a file for each selected symbol in a given folder. The SVG folder can be added to the *SVG paths* in *Settings* ► *Options* ► *System* menu of another user, allowing him direct access to all these symbols.

Importing items

You can extend your style library by importing new items:

1. Expand the *Import/Export* drop-down menu and select *Import Item(s)* at the left bottom of the dialog.
2. In the new dialog, indicate the source of the style items (it can be an `.xml` file on the disk or a url).
3. Set whether to *Add to favorites* the items to import.
4. Check *Do not import embedded tags* to avoid the import of tags associated to the items being imported.
5. Give the name of any *Additional tag(s)* to apply to the new items.
6. Select from the preview the symbols you want to add to your library.
7. And press *Import*.

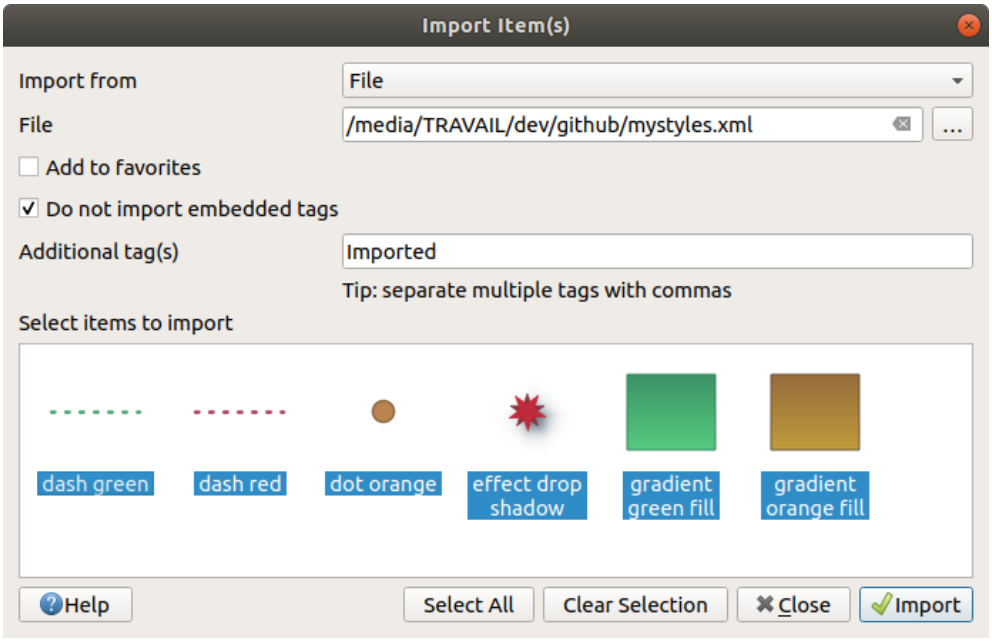


Fig. 10.4: Importing style items

Using the Browser panel

It's also possible to import style items into the active user profile style database directly from the *Browser* panel:

1. Select the style `.xml` file in the browser
2. Drag-and-drop it over the map canvas or right-click and select *Import Style…*
3. Fill the *Import Items* dialog following *Importing items*
4. Press *Import* and the selected style items are added to the style database

Double-clicking the style file in the browser opens the *Style Manager* dialog showing the items in the file. You can select them and press *Copy to Default Style…* to import them into the active style database. Tags can be assigned to items. Also available through right-click, *Open Style…* command.

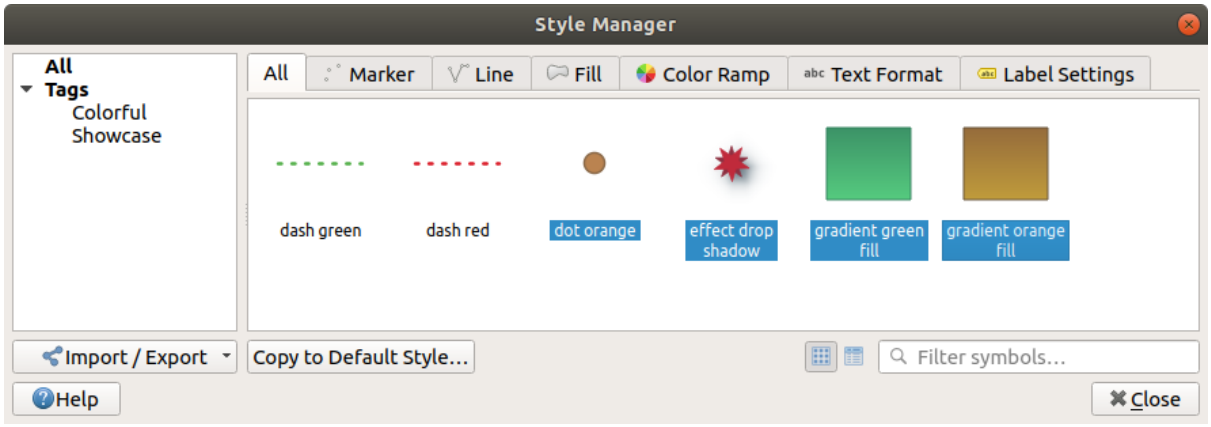


Fig. 10.5: Opening a style items file

The dialog also allows to export single symbols as `.PNG` or `.SVG` files.

Using the online repository

The QGIS project maintains a repository with a collection of styles shared by QGIS users. This is available at https://hub.qgis.org/styles and can be accessed from the *Style Manager* dialog, pressing the *Browse Online Styles* button at the bottom.

From that repository, you can:

1. Browse and search for any style items, based on their type or name
2. Download the style file and unzip it
3. Load the `.xml` based file into your style database in QGIS, using any of the aforementioned import methods.

10.1.2 Setting a Color Ramp

The Color ramp tab in the *Style Manager* dialog helps you preview different color ramps based on the category selected in the left panel.

To create a custom color ramp, activate the Color ramp tab and click the Add item button. The button reveals a drop-down list to choose the ramp type:

- *Gradient*: given a start and end colors, generates a color ramp which can be **continuous** or **discrete**. With double-clicking the ramp preview, you can add as many intermediate color stops as you want. Click on the color stop indicator and under *Gradient stop* you can:
  - adjust its *Relative position* from the color ramp start. Also possible dragging the indicator with the mouse, or pressing the arrow keys (combine with `Shift` key for a larger move)
  - specify the color model to use when interpolating between colors: it can be *RGB*, *HSL* or *HSV*. In some circumstances, this option can help avoid desaturated mid tones, resulting in more visually pleasing gradients.
  - set the direction which the interpolation should follow for the **Hue** component of a *HSL* or *HSV* color specification. It can be *Clockwise* or *Counterclockwise*.
  - set the *color properties*
  - remove the color stop pressing *Delete stop* or `DEL`

  The *Plots* group provides another graphical way to design the color ramp, changing the position or the opacity and HSL components of the color stops.

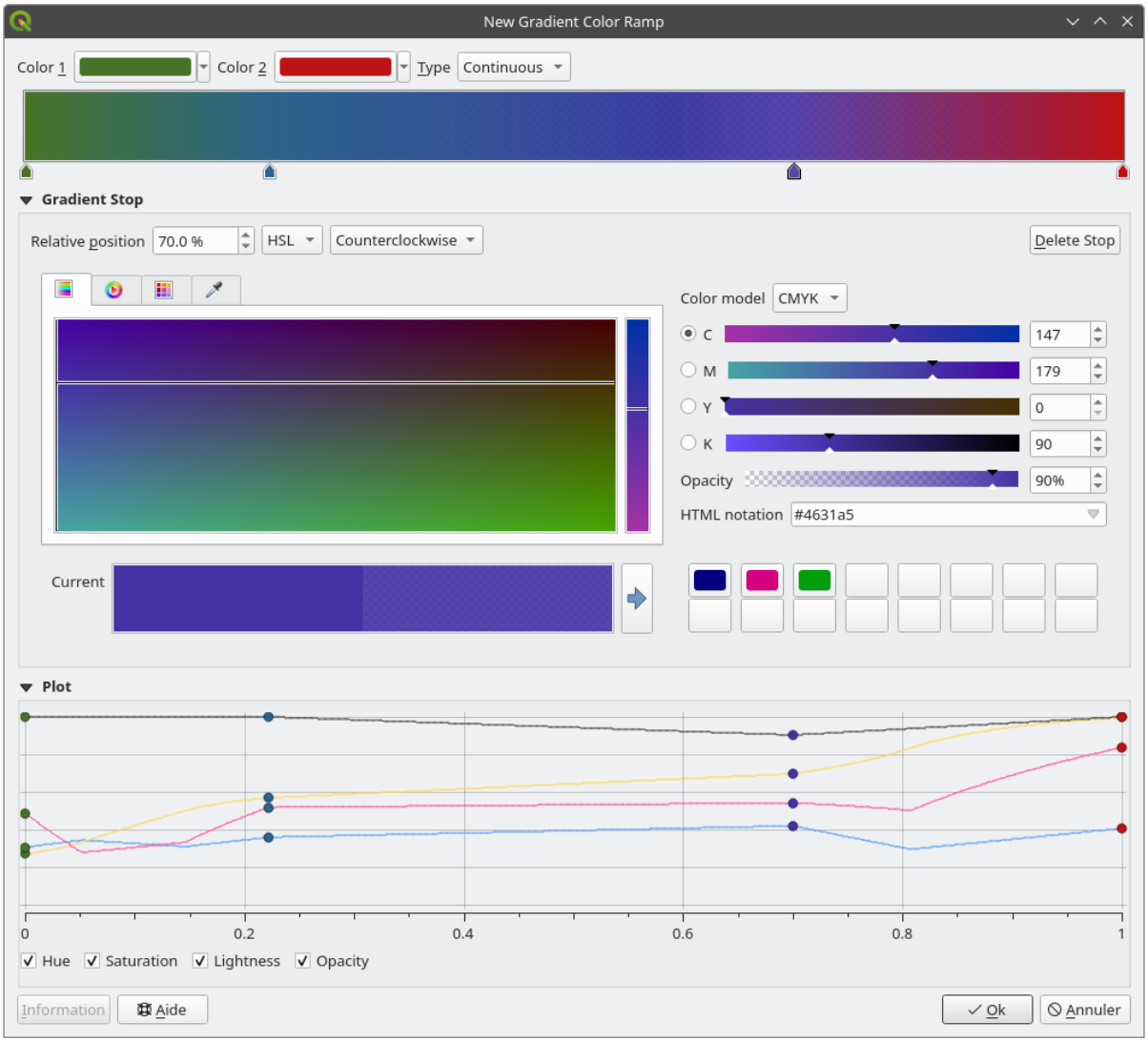


Fig. 10.6: Example of custom gradient color ramp with multiple stops

> **Hint**
>
> Drag-and-drop a color from a color spot onto the gradient ramp preview adds a new color stop.

- *Color presets*: allows to create a color ramp consisting of a list of colors selected by the user;
- *Random*: creates a random set of colors based on range of values for *Hue*, *Saturation*, *Value* and *Opacity* and a number of colors (*Classes*);
- *Catalog: ColorBrewer*: a set of predefined discrete color gradients you can customize the number of colors in the ramp;
- or *Catalog: cpt-city*: an access to a whole catalog of color gradients to locally *save as standard gradient*. The cpt-city option opens a new dialog with hundreds of themes included 'out of the box'.

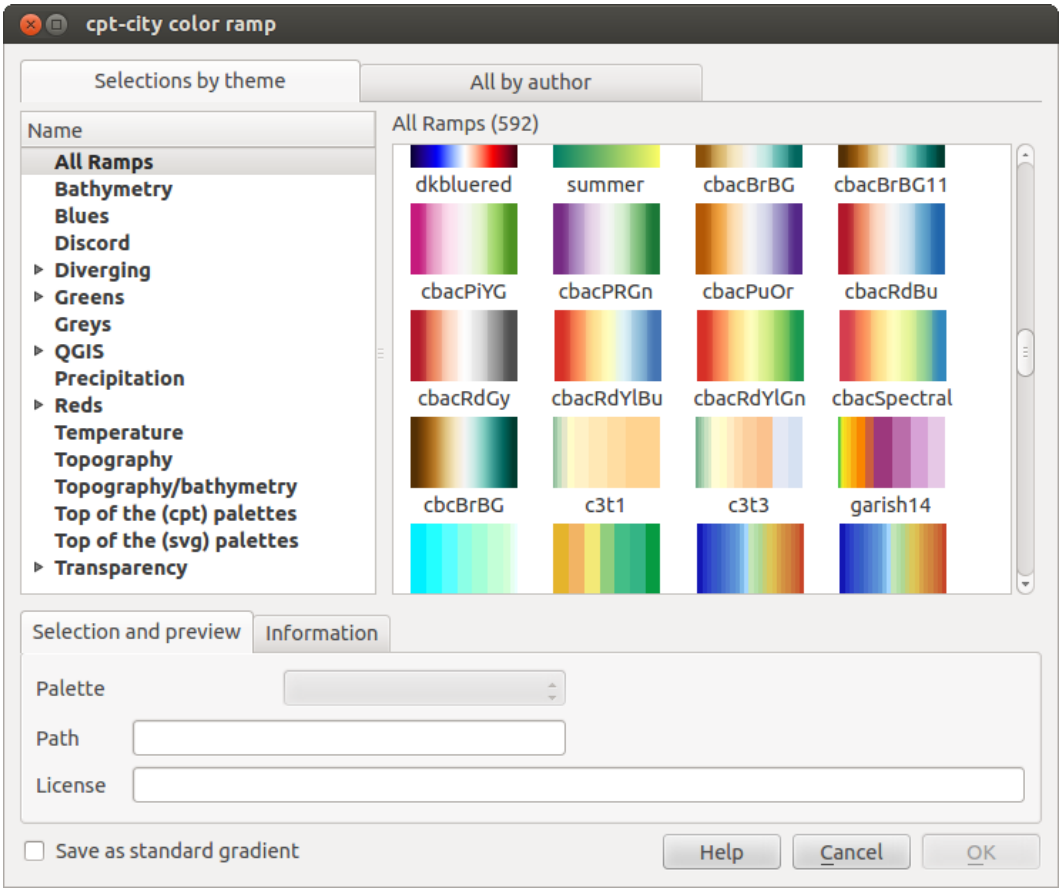


Fig. 10.7: cpt-city dialog with hundreds of color ramps

10.1.3 Creating a Legend Patch Shape

To create a new Legend Patch Shape, activate the *Legend Patch Shapes* tab and click the  Add item button. The button reveals a drop-down list to choose the geometry type:

- *Marker Legend Patch Shape…*: to use with point geometries.
- *Line Legend Patch Shape…*: to use with line geometries.
- *Fill Legend Patch Shape…*: to use with polygon geometries.

All three options will show the same dialog.

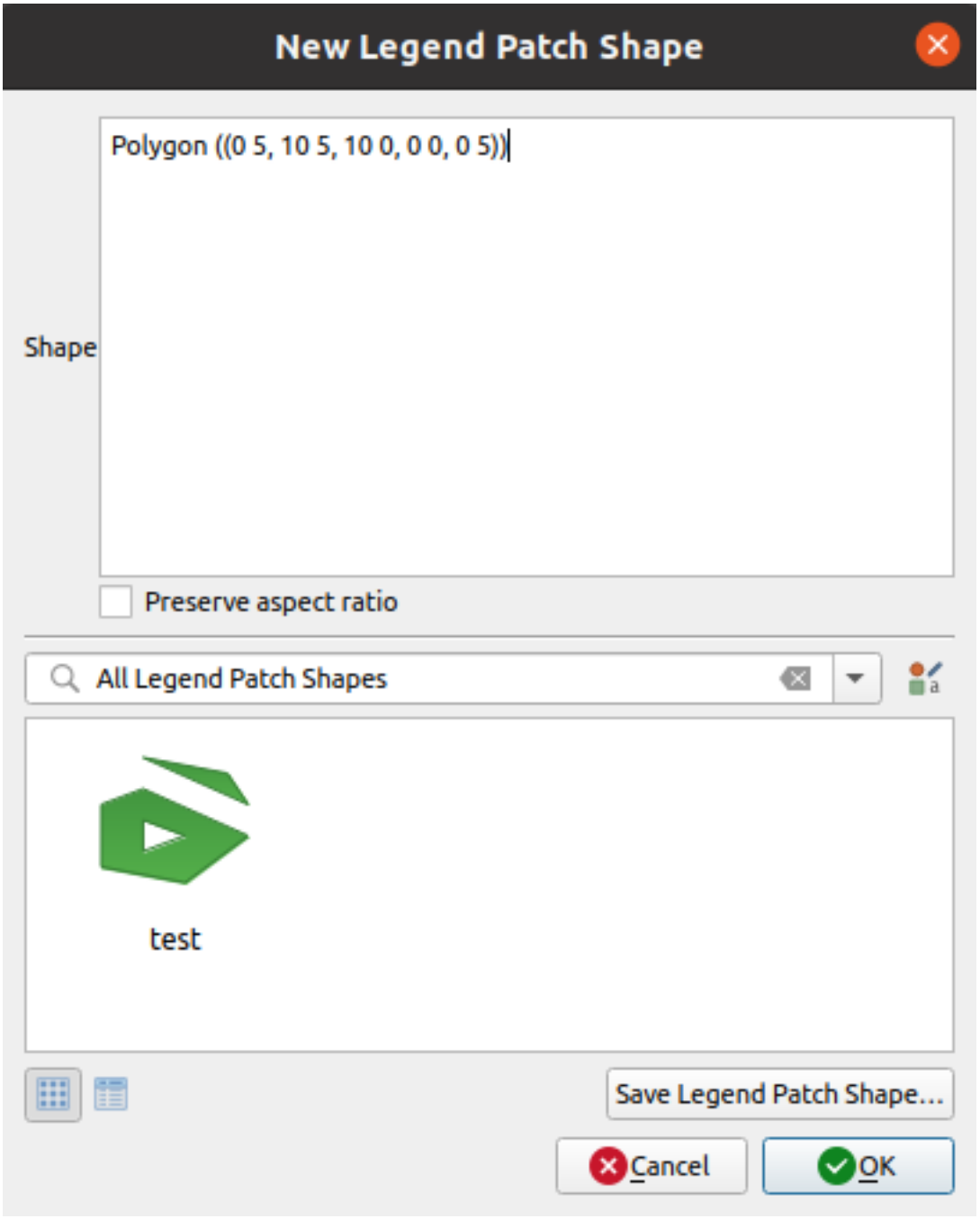


Fig. 10.8: Create a new Legend Patch Shape

Only the shape type and displayed legend patch shapes will differ regarding to the chosen geometry type. The following options will be available:

- *Shape*: define the shape of the legend patch shape as a WKT string. Single and multipart geometries may be used, but no GeometryCollection.
- *Preserve aspect ratio*
- *Icon View* or *List View* of available legend patch shapes, filtered by tags.

When the new Shape is defined you can *Save Legend Patch Shape…* or press *OK*, which will both lead to the same dialog.

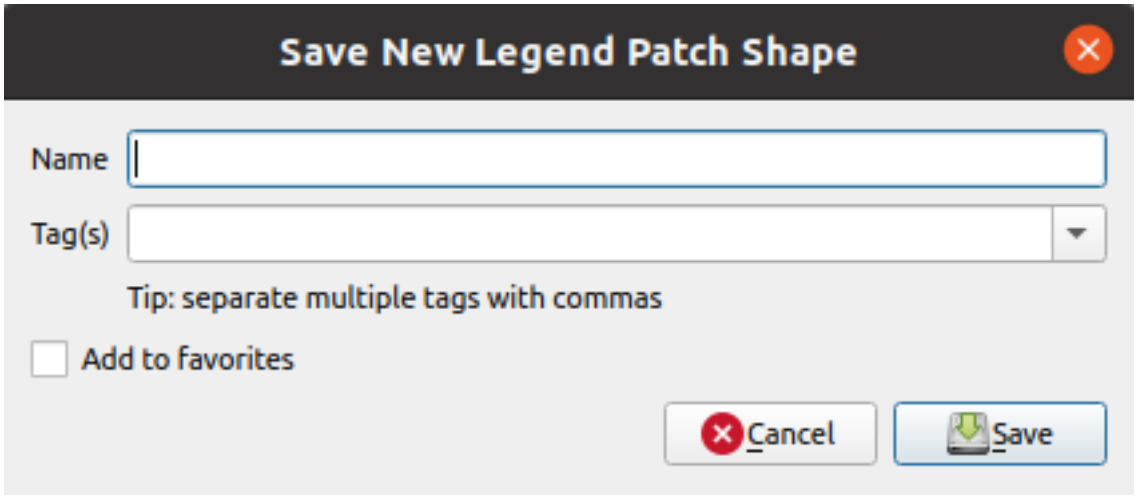


Fig. 10.9: Save a new Legend Patch Shape

Here you have to choose a name, tags to describe the shape and if it should be added to favorites.

If you press *Save…*, the shape is added to the list and you are directed back to the *New Legend Patch Shape* dialog to keep creating new shapes.

10.2 The Symbol Selector

The Symbol selector is the main dialog to design a symbol. You can create or edit Marker, Line or Fill Symbols.

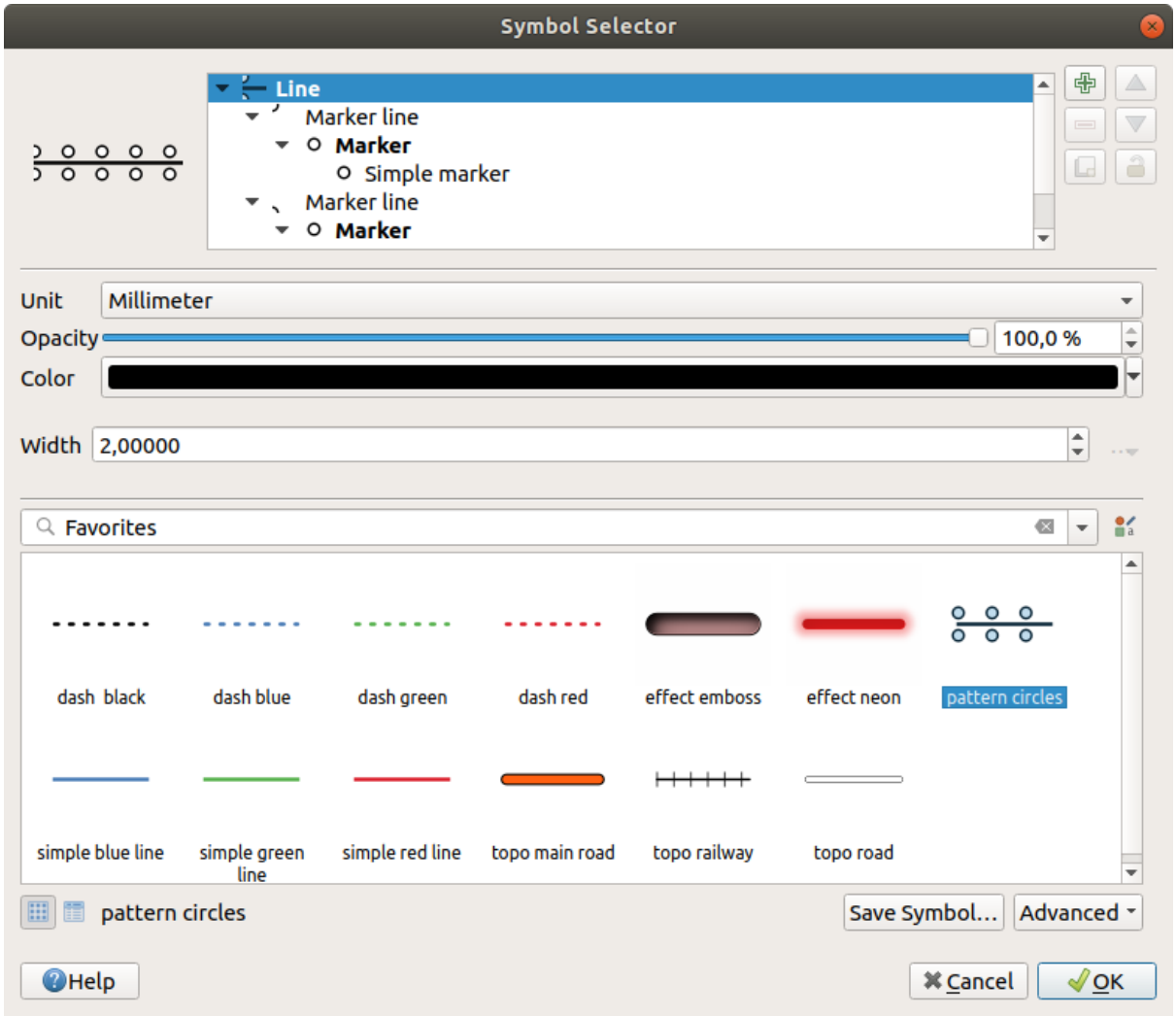


Fig. 10.10: Designing a Line symbol

Two main components structure the symbol selector dialog:

- the symbol tree, showing symbol layers that are combined afterwards to shape a new global symbol
- and settings to configure the selected symbol layer in the tree.

10.2.1 The symbol layer tree

A symbol can consist of several *Symbol layers*. The symbol tree shows the overlay of these symbol layers that are combined afterwards to shape a new global symbol. Besides, a dynamic symbol representation is updated as soon as symbol properties change.

Depending on the level selected in the symbol tree items, various tools are made available to help you manage the tree:

- add new symbol layer: you can stack as many symbols as you want
- remove the selected symbol layer
- lock colors of symbol layer: a locked color stays unchanged when user changes the color at the global (or upper) symbol level
- duplicate a (group of) symbol layer(s)

- move up or down the symbol layer

10.2.2 Configuring a symbol

In QGIS, configuring a symbol is done in two steps: the symbol and then the symbol layer.

The symbol

At the top level of the tree, it depends on the layer geometry and can be of **Marker**, **Line** or **Fill** type. Each symbol can embed one or more symbols (including, of any other type) or symbol layers.

You can setup some parameters that apply to the global symbol:

- *Unit*: it can be **Millimeters**, **Points**, **Pixels**, **Meters at Scale**, **Map units** or **Inches** (see *Unit Selector* for more details)
- *Opacity*
- *Color*: when this parameter is changed by the user, its value is echoed to all unlocked sub-symbols color
- *Size* and *Rotation* for marker symbols
- *Width* for line symbols

> **Tip**
>
> Use the *Size* (for marker symbols) or the *Width* (for line symbols) properties at the symbol level to proportionally resize all of its embedded *symbol layers* dimensions.

> **Note**
>
> The *Data-defined override* button next to the width, size or rotation parameters is inactive when setting the symbol from the Style manager dialog. When the symbol is connected to a map layer, this button helps you create *proportional or multivariate analysis* rendering.

- A preview of the *symbols library*: Symbols of the same type are shown and, through the editable drop-down list just above, can be filtered by free-form text or by *categories*. You can also update the list of symbols using the  Style Manager button and open the eponym dialog. There, you can use any capabilities as exposed in *The Style Manager* section.

  The symbols are displayed either:

  - in an icon list (with thumbnail, name and associated tags) using the  List View button below the frame;
  - or as icon preview using the  Icon View button.
- Press the *Save Symbol* button to open the *Save New Symbol* dialog. Here, you can choose the *Destination* where you want to add the symbol being edited, give it a *Name* and add *Tag(s)*. You also have the option to  Add to favorites your new symbol.
- With the *Advanced*  option, you can:
  - for line and fill symbols, *Clip features to canvas extent*.
  - for fill symbols, *Force right-hand rule orientation*: allows forcing rendered fill symbols to follow the standard “right hand rule” for ring orientation (i.e, polygons where the exterior ring is clockwise, and the interior rings are all counter-clockwise).

The orientation fix is applied while rendering only, and the original feature geometry is unchanged. This allows for creation of fill symbols with consistent appearance, regardless of the dataset being rendered and the ring orientation of individual features.

- for marker symbols, the *Buffer settings…* enables addition of a halo effect around the marker symbol in order to make it more readable against different backgrounds. The buffer is calculated and drawn using the shape of ALL the symbol layers in the marker. You can adjust the *Size*, *Join style* and *Symbol* of display.
- for all symbol types, enable and configure a distance for the map canvas *Extent buffer*, or the *Animation settings*.
- Depending on the *symbology* of the layer the symbol is being applied to, additional settings are available in the *Advanced* menu:
  - *Symbol levels…* to define the order of symbols rendering
  - *Data-defined Size Legend*
  - *Match to Saved Symbols…* and *Match to Symbols from File…* to automatically *assign symbols to classes*

The symbol layer

At a lower level of the tree, you can customize the symbol layers. The available symbol layer types depend on the upper symbol type. You can apply on the symbol layer  *paint effects* to enhance its rendering.

Because describing all the options of all the symbol layer types would not be possible, only particular and significant ones are mentioned below.

Common parameters

Some common options and widgets are available to build a symbol layer, regardless it's of marker, line or fill sub-type:

- the *color selector* widget to ease color manipulation
- *Units*: it can be **Millimeters**, **Points**, **Pixels**, **Meters at Scale**, **Map units** or **Inches** (see *Unit Selector* for more details)
- the  Data-defined override widget near almost all options, extending capabilities of customizing each symbol (see *Data defined override setup* for more information)
- the  *Enable symbol layer* option controls the symbol layer's visibility. Disabled symbol layers are not drawn when rendering the symbol but are saved in the symbol. Being able to hide symbol layers is convenient when looking for the best design of your symbol as you don't need to remove any for the testing. The data-defined override then makes it possible to hide or display different symbol layers based on expressions (using, for instance, feature attributes).
- the  *Draw effects* button for *effects rendering*.

> **Note**
>
> While the description below assumes that the symbol layer type is bound to the feature geometry, keep in mind that you can embed symbol layers in each others. In that case, the lower level symbol layer parameter (placement, offset…) might be bound to the upper-level symbol, and not to the feature geometry itself.

Marker Symbols

Appropriate for point geometry features, marker symbols have several *Symbol layer types*:

- **Simple marker** (default)

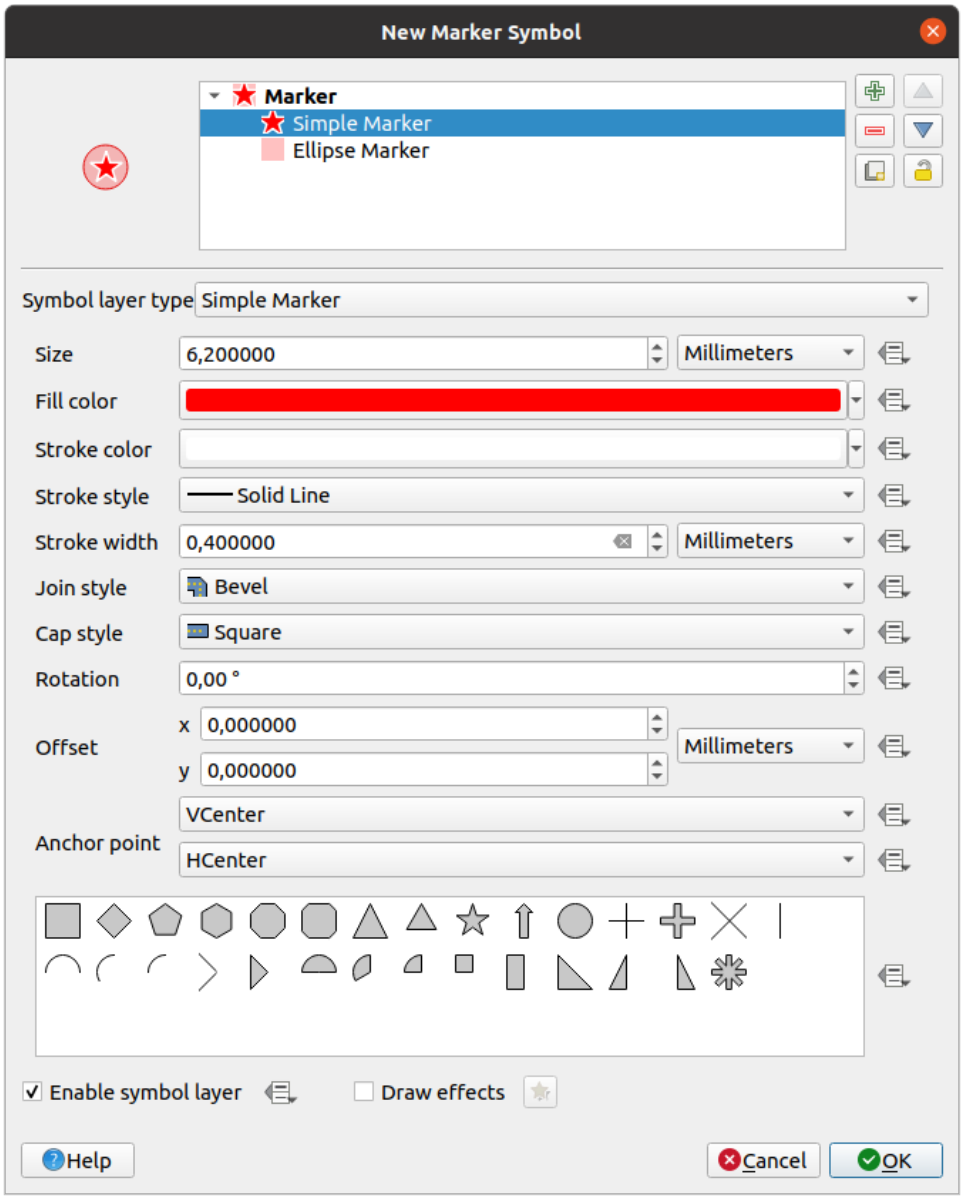


Fig. 10.11: Designing a Simple Marker Symbol

The simple marker symbol layer type has the following properties:

- *Size* in various supported units
- *Fill color*
- *Stroke color*, *Stroke style* from a predefined list and *Stroke size*
- *Join style*: it can be **Bevel**, **Miter** or **Round**
- *Cap style*: it can be **Square**, **Flat** or **Round**
- *Rotation*

  - *Offset* in *X* and *Y* directions from the feature
  - *Anchor point*: defining the quadrant point on the symbol to settle as placement origin. This is the point the *Offset* is applied on.

- **Animated marker** (see *Animated marker*)
- **Ellipse marker**: a simple marker symbol layer, with customizable width and height
- **Filled marker**: similar to the simple marker symbol layer, except that it uses a *fill sub symbol* to render the marker. This allows use of all the existing QGIS fill (and stroke) styles for rendering markers, e.g. gradient or shapeburst fills.
- **Font marker**: similar to the simple marker symbol layer, except that it uses installed fonts to render the marker. Its additional properties are:
  - *Font family*
  - *Font style*
  - *Character(s)*, representing the text to display as symbol. They can be typed in or selected from the font characters collection widget and you can live *Preview* them with the selected settings.
- **Geometry generator** (see *The Geometry Generator*)
- **Mask**: its sub-symbol defines a mask shape whose color property will be ignored and only the opacity will be used. This is convenient when the marker symbol overlaps with labels or other symbols whose colors are close, making it hard to decipher. More details at *Masks Properties*.
- **Raster image marker**: use an image (`PNG`, `JPG`, `BMP` ...) as marker symbol. The image can be a file on the disk, a remote URL, embedded in the style database (*more details*) or it can be encoded as a base64 string.

  Width and height of the image can be set independently or using the  Lock aspect ratio. The size can be set using any of the *common units* or as a percentage of the image's original size (scaled by the width).
- **Vector Field marker** (see *The Vector Field Marker*)
- **SVG marker**: provides you with images from your SVG paths (set in *Settings* ► *Options...* ► *System* menu) to render as marker symbol. Width and height of the symbol can be set independently or using the  Lock aspect ratio. Each SVG file colors and stroke can also be adapted. The image can be a file on the disk, a remote URL, embedded in the style database (*more details*) or it can be encoded as a base64 string.

  The symbol can also be set with *Dynamic SVG parameters*. See *Parametrizable SVG* section to parametrize an SVG symbol.

  Note

  SVG version requirements

  QGIS renders SVG files that follow the SVG Tiny 1.2 profile, intended for implementation on a range of devices, from cellphones and PDAs to laptop and desktop computers, and thus includes a subset of the features included in SVG 1.1 Full, along with new features to extend the capabilities of SVG.

  Some features not included in these specifications might not be rendered correctly in QGIS.

Line Symbols

Appropriate for line geometry features, line symbols have the following symbol layer types:

- **Simple line** (default)

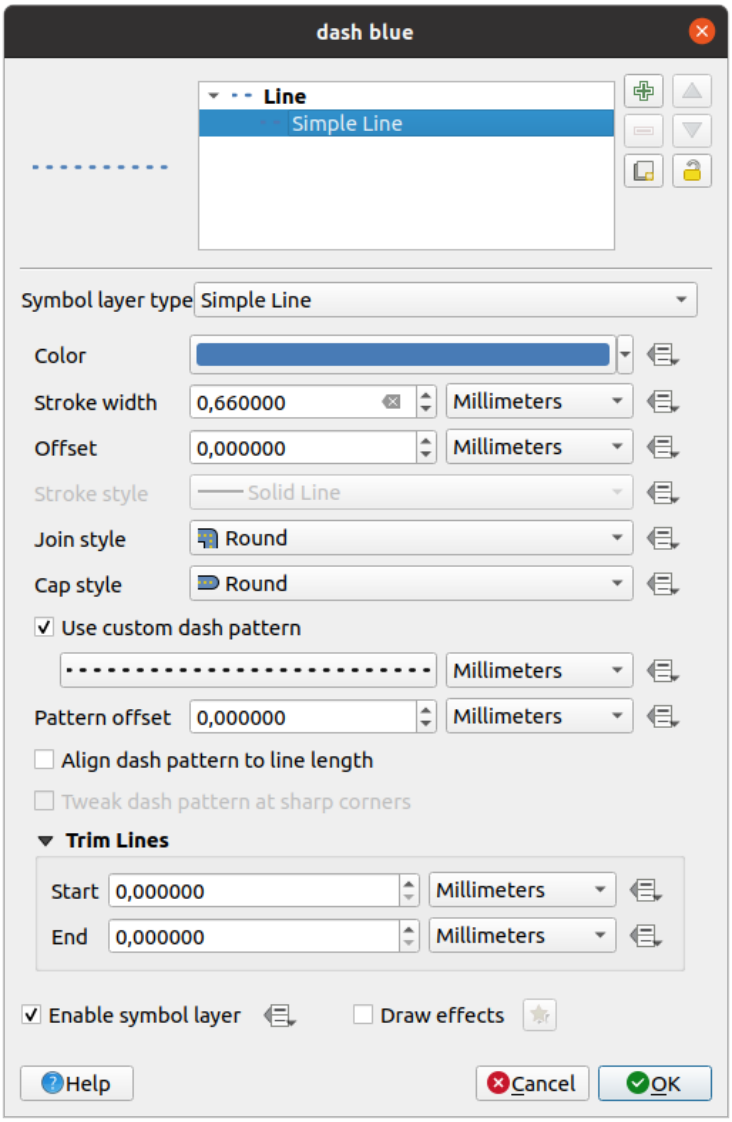


Fig. 10.12: Designing a Simple Line Symbol

The simple line symbol layer type has many of the same properties as the *simple marker symbol*, and in addition:

- *Use custom dash pattern*: overrides the *Stroke style* setting with a custom dash. You would need to define length of consecutive dashes and spaces shaping the model, in the chosen unit. The total length of the pattern is displayed at the bottom of the dialog.
- *Pattern offset*: the positioning of the dashes/spaces in the line can be tweaked, so that they can be placed at nicer positions to account for corners in the line (also can be used potentially to "align" adjacent dash pattern borders)
- *Align dash pattern to line length*: the dash pattern length will be adjusted so that the line will end with a complete dash element, instead of a gap.
- *Tweak dash pattern at sharp corners*: dynamically adjusts the dash pattern placement so that sharp

corners are represented by a full dash element coming into and out of the sharp corner. Dependent on *Align dash pattern to line length*.

  - *Trim lines* from *Start* and/or *End*: allows for the line rendering to trim off the first x mm and last y mm from the actual line string when drawing the line. It supports a range of *units*, including percentage of the overall line length, and can be data defined for extra control. The start/end trim distance can be used e.g. when creating complex symbols where a line layer should not overlap marker symbol layers placed at the start and end of the line.

- **Arrow**: draws lines as curved (or not) arrows with a single or a double head with configurable (and data-defined):
  - *Head type*
  - *Arrow type*
  - *Arrow width*
  - *Arrow width at start*
  - *Head length*
  - *Head thickness*
  - *Offset*

  It is possible to create *Curved arrows* (the line feature must have at least three vertices) and *Repeat arrow on each segment*. It also uses a *fill symbol* such as gradients or shapeburst to render the arrow body. Combined with the geometry generator, this type of layer symbol helps you representing flow maps.
- **Geometry generator** (see *The Geometry Generator*)
- **Interpolated line**: allows to render a line whose *Stroke width* and/or *Color* may be constant (given a *Fixed width* and *Single color* parameters) or vary along the geometry. When varying, necessary inputs are:
  - *Start value* and *End value*: Values that will be used for interpolation at the extremities of the features geometry. They can be fixed values, feature's attributes or based on an expression.
  - *Min. value* and *Max. value*: Values between which the interpolation is performed. Press the Load button to automatically fill them based on the minimum and maximum start/end values applied to the layer.
  - Only available for the stroke option:
    * *Min. width* and *Max. width*: define the range of the varying width. *Min. width* is assigned to the *Min. value* and *Max. width* to the *Max. value*. A *unit* can be associated.
    * *Use absolute value*: only consider absolute value for interpolation (negative values are used as positive).
    * *Ignore out of range*: by default, when the `[start value - end value]` range of a feature is not included in the `[min. value - max. value]` range, the out-of-bounds parts of the feature's geometry are rendered with the min or max width. Check this option to not render them at all.
  - For varying color, you can use any of the interpolation methods of *color ramp classification*

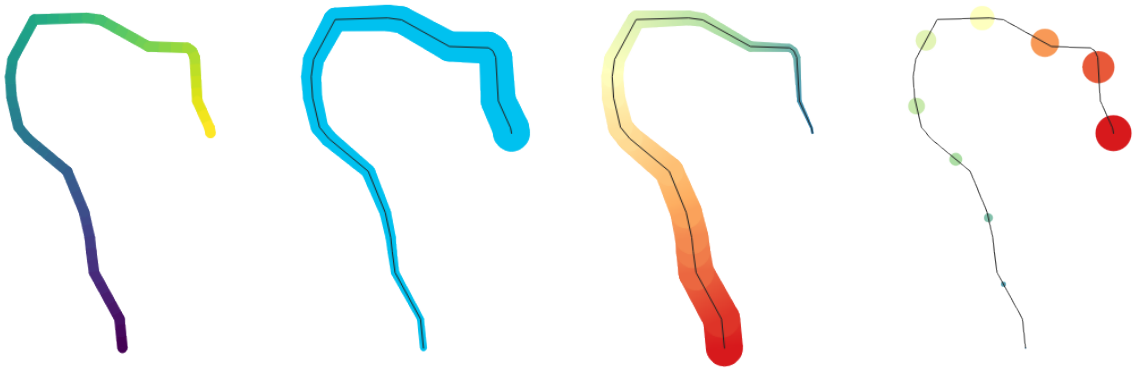


Fig. 10.13: Examples of interpolated lines

- **Marker line**: repeats a *marker symbol* over the length of a line.
  - The *Marker placement* can be set using a regular interval setting (starting from the first vertex) and/or the line geometry property (on first or last vertex, inner vertices, the central point of the line or of each segment, or on every curve point).
  - When first or last vertex placement is enabled, the *Place on every part extremity* option will make the markers render also at the first or last vertex for every part of multipart geometries.
  - *Offset along line*: the markers placement can also be given an offset along the line, in the ref:*unit <unit_selector>* of your choice (millimeters, points, map unit, meters at scale, percentage, …):
    - A positive value offsets the markers symbols in the line direction (with *On first vertex* and *With interval* placements) and backwards (with *On last vertex* placement).
    - A negative value on a not closed line will result in no offset (for *On first vertex* and *On last vertex* placements) or backwards offset of the symbols (from the last vertex).
    - With a closed ring, QGIS treats the offset as continuing to loop around the ring (forward or backward). E.g. setting the offset to 150% (resp. -10% or -110%) results in the offset being treated as 50% (resp. 90%) of the length of the closed ring.
  - The *Rotate marker to follow line direction* option sets whether each marker symbol should be oriented relative to the line direction or not.

    Because a line is often a succession of segments of different directions, the rotation of the marker is calculated by averaging over a specified distance along the line. For example, setting the *Average angle over* property to `4mm` means that the two points along the line that are `2mm` before and after the symbol placement are used to calculate the line angle for that marker symbol. This has the effect of smoothing (or removing) any tiny local deviations from the overall line direction, resulting in much nicer visual orientations of the marker line symbols.
  - *Line offset*: the marker symbols can also be offset from the line feature.
- **Hashed line**: repeats a line segment (a hash) over the length of a line symbol, with a line sub-symbol used to render each individual segment. In other words, a hashed line is like a marker line in which marker symbols are replaced with segments. As such, the hashed lines have the *same properties* as marker line symbols, along with:
  - *Hash length*
  - *Hash rotation*

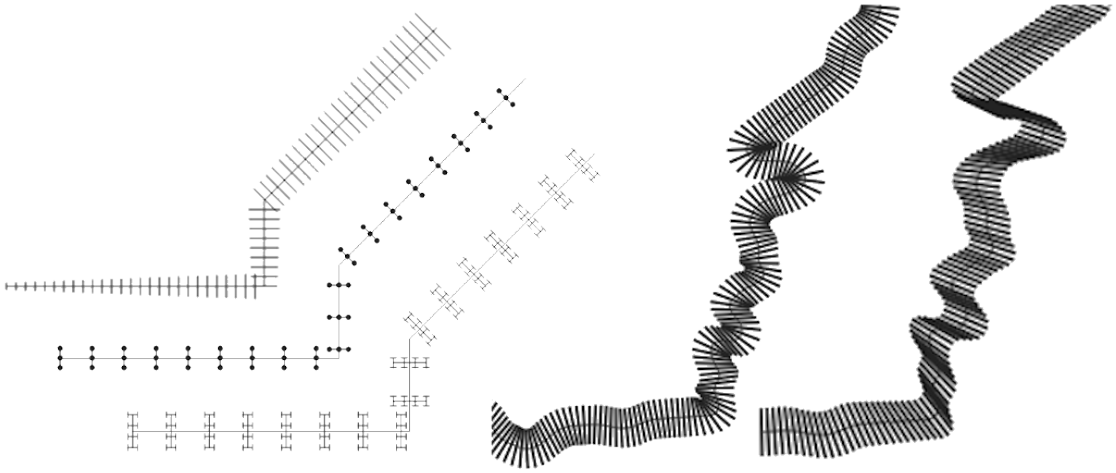


Fig. 10.14: Examples of hashed lines

- **Raster line**: renders and repeats a raster image following the length of a line feature shape. The *Stroke width*, *Offset*, *Join style*, *Cap style* and *Opacity* can be adjusted.

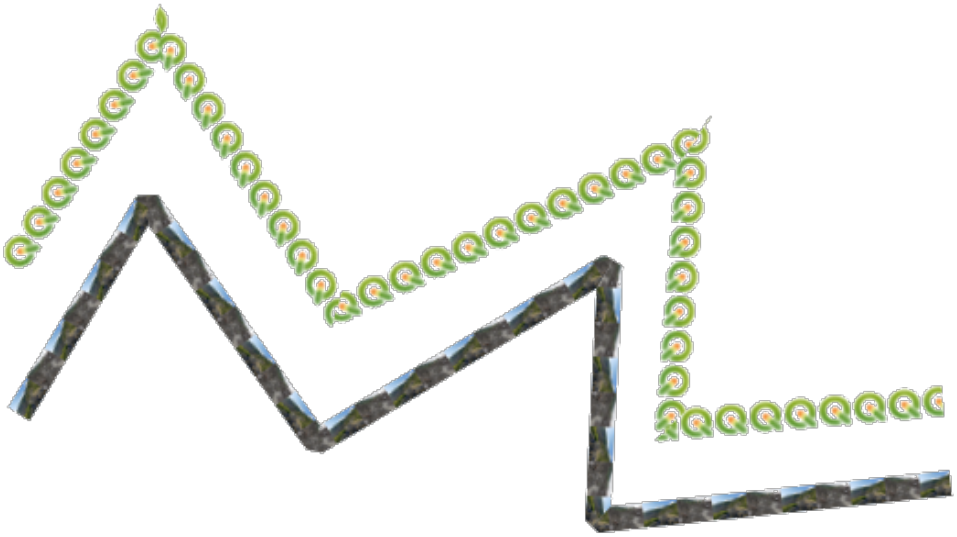


Fig. 10.15: Examples of raster lines

- **Linear referencing**: allows placing text labels at regular intervals along a line, or at positions corresponding to existing vertices. Positions can be calculated using Cartesian distances, or interpolated from z/m values. Parameters include:
  - *Measure placement*: Labels can be placed at regular cartesian 2d distances, at regular linearly interpolated spacing calculated using the Z or M values in geometries, or at existing vertices. A fixed or data-defined *Interval* is given.
  - *Quantity*: Label displayed at each position represent the running distance along the line, or the linearly interpolated Z or M value. Labels are rendered using the full range of *text* and *number* formatting functionalities available in QGIS.

> **Note**
>
> When using the distance-based placement or labels, the distances are calculated using 2D only. Cartesian calculations are based on the original layer CRS.

- *Skip multiples of*: If set, labels which are a multiple of this value will be skipped over. This allows construction of complex referencing labels, e.g. where a symbol has two linear referencing symbol layers, one set to label every 100m in a small font, skipping multiples of 1000, and a second set to label every 1000m in a big bold font.
- *Average angle over*: Labels are rendered using an angle calculated by averaging the linestring, so sharp tiny jaggies don't result in unslightly label rotation.
- *Show marker symbols*, at referenced points in the line feature, using a full QGIS marker symbol. This allows e.g. showing a cross-hatch at the labeled point, for a "ruler" style line.

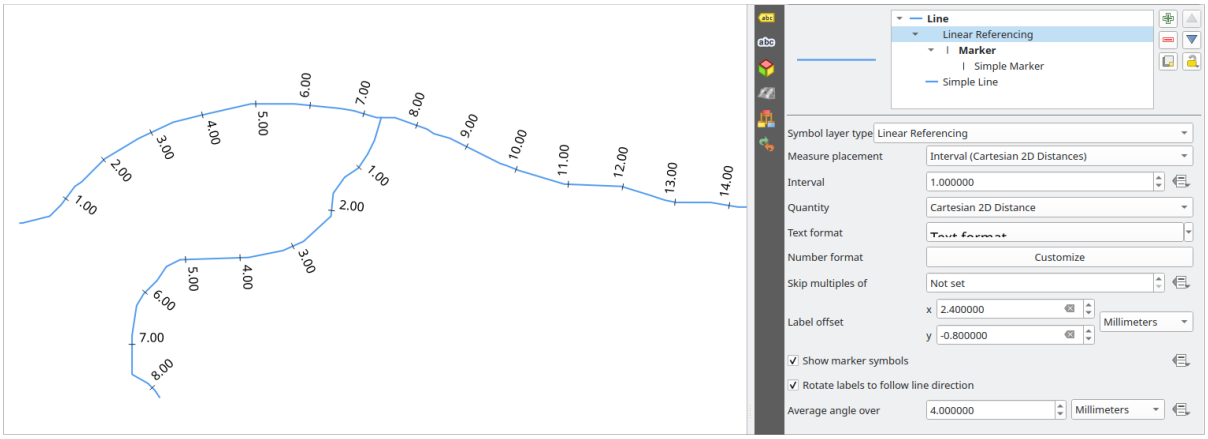


Fig. 10.16: Drawing linear labels along line feature

- **Lineburst**: renders a gradient along the width of a line. You can choose between *Two color* or *Color ramp* and the *Stroke width*, *Offset*, *Join style*, *Cap style* can be adjusted.



Fig. 10.17: Examples of lineburst lines

- **Filled line**: renders the interior of the lines using a fill symbol allowing for lines filled with gradients, line hatches, etc. The *Stroke width*, *Offset*, *Join style*, *Cap style* can be adjusted.

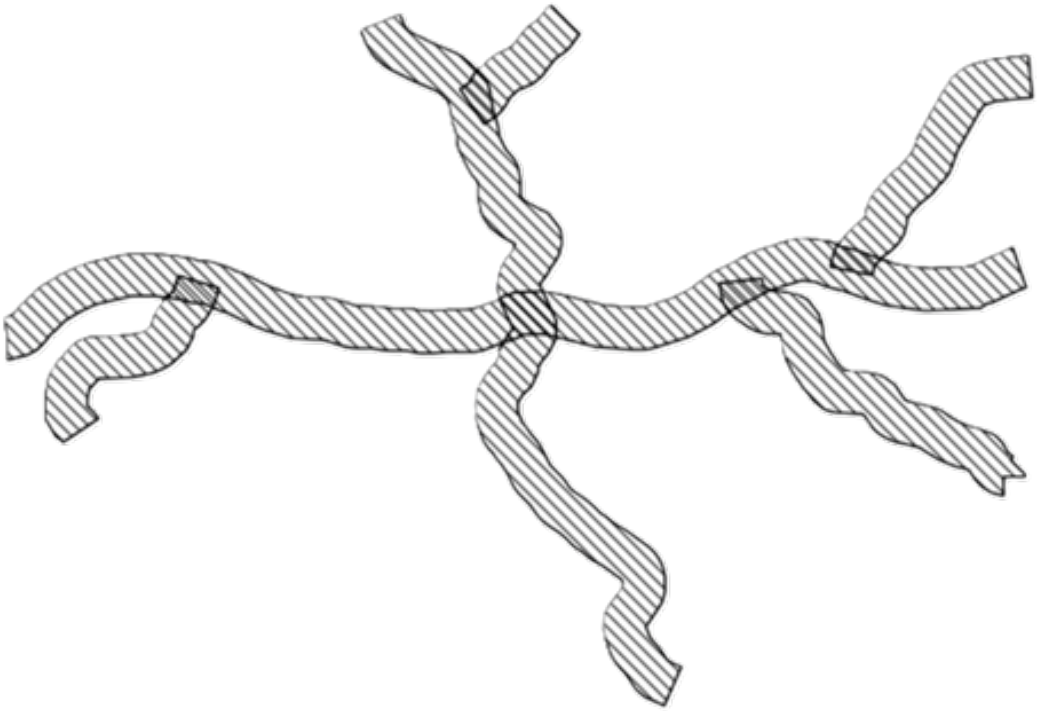


Fig. 10.18: Examples of filled lines

Fill Symbols

Appropriate for polygon geometry features, fill symbols have also several symbol layer types:

- **Simple fill** (default): fills a polygon with a uniform color

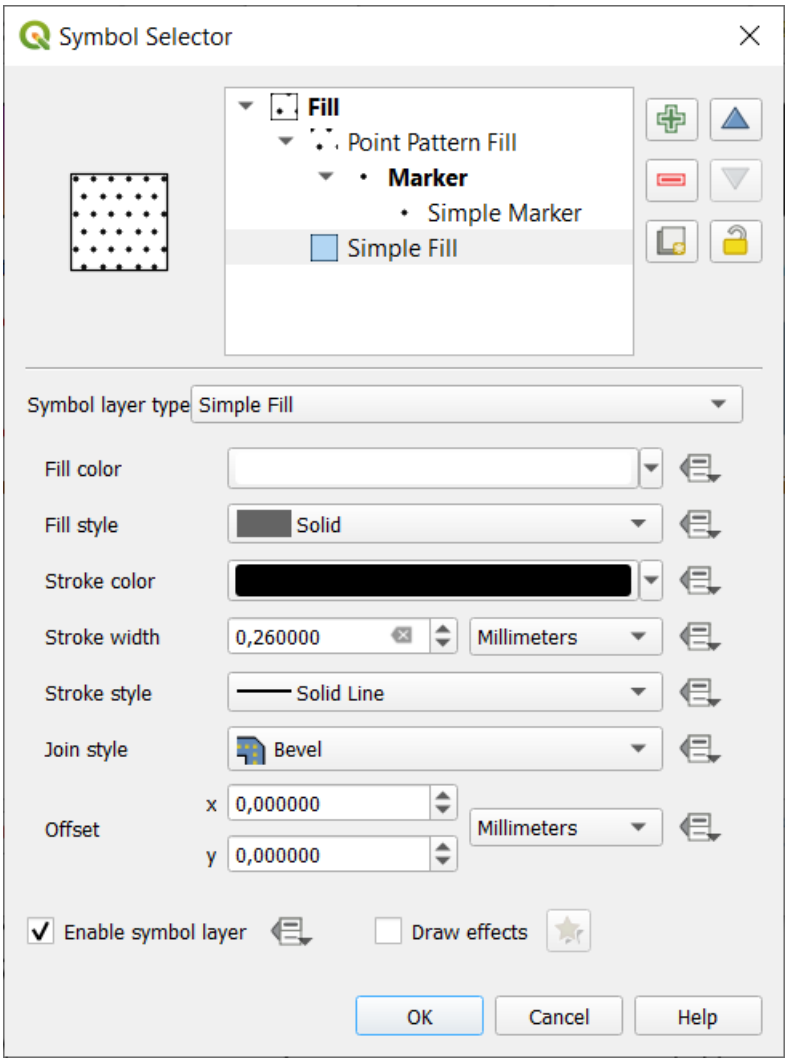


Fig. 10.19: Designing a Simple Fill Symbol

- **Centroid fill**: places a *marker symbol* at the centroid of the visible feature. The position of the marker may not be the real centroid of the feature, because calculation takes into account the polygon(s) clipped to area visible in map canvas for rendering and ignores holes. Use the *geometry generator symbol* if you want the exact centroid.

  You can:

  - *Force placement of markers inside polygons*
  - *Draw markers on every part of multi-part features* or place the point only on its biggest part
  - display the marker symbol(s) in whole or in part, keeping parts overlapping the current feature geometry (*Clip markers to polygon boundary*) or the geometry part the symbol belongs to (*Clip markers to current part boundary only*)

- **Geometry generator** (see *The Geometry Generator*)
- **Gradient fill**: uses a radial, linear or conical gradient, based on either simple two color gradients or a predefined *gradient color ramp* to fill polygons. The gradient can be rotated and applied on a single feature basis or across

the whole map extent. Also start and end points can be set via coordinates or using the centroid (of feature or map). A data-defined offset can be defined.

- **Line pattern fill**: fills the polygon with a hatching pattern of *line symbol layer*. You can set:
  - *Alignment*: defines how the pattern is positioned relative to the feature(s):
    - *Align pattern to feature*: lines are rendered within each feature
    - *Align pattern to map extent*: a pattern is rendered over the whole map extent, allowing lines to align nicely across features
  - *Rotation* of the lines, counter-clockwise
  - *Spacing*: distance between consecutive lines
  - *Offset* distance of the lines from the feature boundary
  - *Clipping*: allows to control how lines in the fill should be clipped to the polygon shape. Options are:
    - *Clip During Render Only*: lines are created covering the whole bounding box of the feature and then clipped while drawing. Line extremities (beginning and end) will not be visible.
    - *Clip Lines Before Render*: lines are clipped to the exact shape of the polygon prior to rendering. Line extremities (including cap styles, start/end marker line objects, …) will be visible, and may sometimes extend outside of the polygon (depending on the line symbol settings).
    - *No Clipping*: no clipping at all is done - lines will cover the whole bounding box of the feature
- **Point pattern fill**: fills the polygon with a grid pattern of *marker symbol*. You can set:
  - *Alignment*: defines how the pattern is positioned relative to the feature(s):
    - *Align pattern to feature*: marker lines are rendered within each feature
    - *Align pattern to map extent*: a pattern is rendered over the whole map extent, allowing markers to align nicely across features

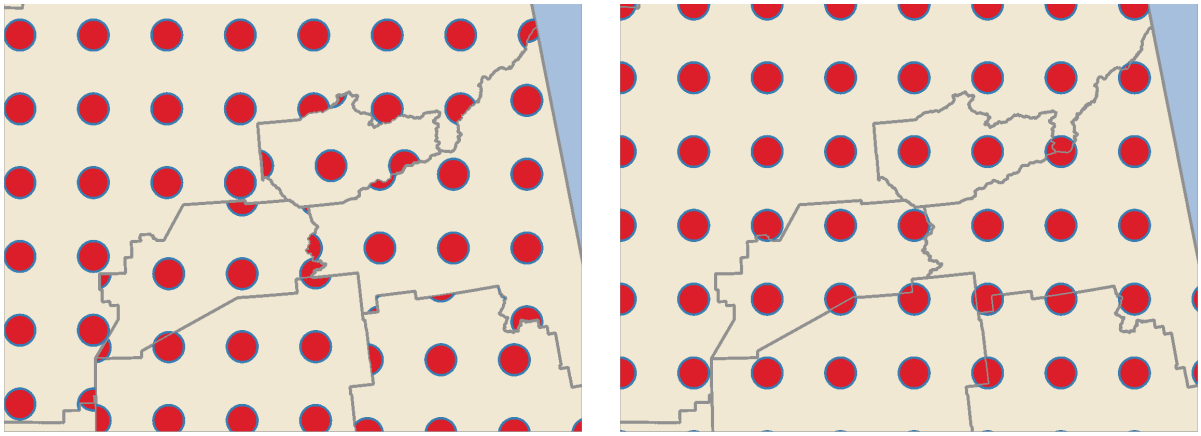


Fig. 10.20: Aligning point pattern to feature (left) and to map extent (right)

  - *Distance*: *Horizontal* and *Vertical* distances between consecutive markers
  - *Displacement*: a *Horizontal* (resp. *Vertical*) offset of alignment between consecutive markers in a column (resp. in a row)
  - *Offset*: *Horizontal* and *Vertical* distances from the feature boundary
  - *Clipping*: allows to control how markers in the fill should be clipped to the polygon shape. Options are:
    - *Clip to shape*: markers are clipped so that only the portions inside the polygon are visible
    - *Marker centroid within shape*: only markers where the center of the marker falls inside the polygon are drawn, but these markers won't be clipped to the outside of the polygon

    * *Marker completely within shape*: only markers which fall completely within the polygon are shown
    * *No clipping*: any marker which intersects at all with the polygon will be completely rendered (strictly speaking its the "intersects with the bounding box of the marker")

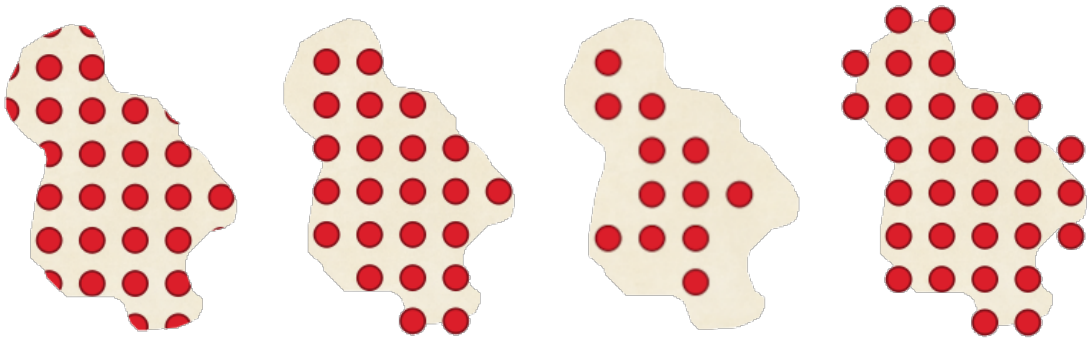


Fig. 10.21: Clipping markers in fill - From left to right: Clip to shape, Marker centroid within shape, Marker completely within shape, No clipping

  - *Rotation* of the whole pattern, clockwise
  - The *Randomize pattern* group setting allows each point in a point pattern fill to be randomly shifted up to the specified maximum distance *Horizontally* or *Vertically*. You can specify the maximum offset in any supported units, such as millimeters, points, map units, or even "percentage" (where percentage is relative to the pattern width or height).

    You can set an optional random number seed to avoid the symbol patterns "jumping" around between map refreshes. Data defined overrides are also supported.

    Note

    The main difference between the *Randomize pattern* and the *random marker fill* symbol type is that the random offset with a point pattern allows for quasi-"regular" placement of markers – because the points in the pattern are effectively constrained to a grid, this allows creation of semi-random fills which don't have empty areas or overlapping markers. (As opposed to the random marker fill, which will always place points completely randomly… sometimes resulting in visual clusters of points or unwanted empty areas).

- **Random marker fill**: fills the polygon with a *marker symbol* placed at random locations within the polygon boundary. You can set:
  - *Count method*: whether the number of marker symbols to render is considered as an absolute count or density-based
  - *Point count*: the number of marker symbols to render,
  - an optional random number *seed*, to give consistent placement
  - *Density area*: in case of density-based count method, ensures the fill density of markers remains the same on different scale / zoom levels of markers whenever maps are refreshed (also allows random placement to play nice with QGIS server and tile-based rendering)
  - *Clip markers to polygon boundary*: whether markers rendered near the edges of polygons should be clipped to the polygon boundary or not
- **Raster image fill**: fills the polygon with tiles from a raster image (`PNG` `JPG`, `BMP` …). The image can be a file on the disk, a remote URL or an embedded file encoded as a string (*more details*). Options include (data defined) opacity, size, coordinate mode (object or viewport), rotation and offset. Under the option Size you can independently adjust the width and height of the fill pattern, enabling stretched raster fills in either the

horizontal or vertical directions. The image width and height can be set using any of the *common units* or as a percentage of the original size.

- **SVG fill**: fills the polygon using *SVG markers* of a given size (*Texture width*).
- **Shapeburst fill**: buffers a gradient fill, where a gradient is drawn from the boundary of a polygon towards the polygon's centre. Configurable parameters include distance from the boundary to shade, use of color ramps or simple two color gradients, optional blurring of the fill and offsets.
- **Outline: Arrow**: uses a line *arrow symbol* layer to represent the polygon boundary. The settings for the outline arrow are the same as for arrow line symbols.
- **Outline: Hashed line**: uses a *hash line symbol* layer to represent the polygon boundary (*Rings*) which can be the interior rings only, the exterior ring only or all the rings). The other settings for the outline hashed line are the same as for hashed line symbols.
- **Outline: Marker line**: uses a *marker line symbol* layer to represent the polygon boundary (*Rings*) which can be the interior rings only, the exterior ring only or all the rings). The other settings for the outline marker line are same as for marker line symbols.
- **Outline: simple line**: uses a *simple line symbol* layer to represent the polygon boundary (*Rings*) which can be the interior rings only, the exterior ring only or all the rings). The *Draw line only inside polygon* option displays the polygon borders inside the polygon and can be useful to clearly represent adjacent polygon boundaries. The other settings for the outline simple line are the same as for simple line symbols.

> **Note**
>
> When geometry type is polygon, you can choose to disable the automatic clipping of lines/polygons to the canvas extent. In some cases this clipping results in unfavourable symbology (e.g. centroid fills where the centroid must always be the actual feature's centroid).

Parametrizable SVG

You have the possibility to change the colors of a *SVG marker*. You have to add the placeholders `param(fill)` for fill color, `param(fill-opacity)` for fill opacity, `param(outline)` and `param(outline-opacity)` for stroke color and opacity respectively, and `param(outline-width)` for stroke width. These placeholders can optionally be followed by a default value, e.g.:

```
<svg width="100%" height="100%">
<rect fill="param(fill) #ff0000" fill-opacity="param(fill-opacity) 1" stroke=
↪"param(outline) #00ff00" stroke-opacity="param(outline-opacity) 1" stroke-width=
↪"param(outline-width) 10" width="100" height="100">
</rect>
</svg>
```

More generally, SVG can be freely parametrized using `param(param_name)`. This param can either be used as an attribute value or a node text:

```
<g stroke-width=".265" text-anchor="middle" alignment-baseline="param(align)">
  <text x="98" y="147.5" font-size="6px">param(text1)</text>
  <text x="98" y="156.3" font-size="4.5px">param(text2)</text>
</g>
```

The parameters can then be defined as expressions in the *Dynamic SVG parameters* table.

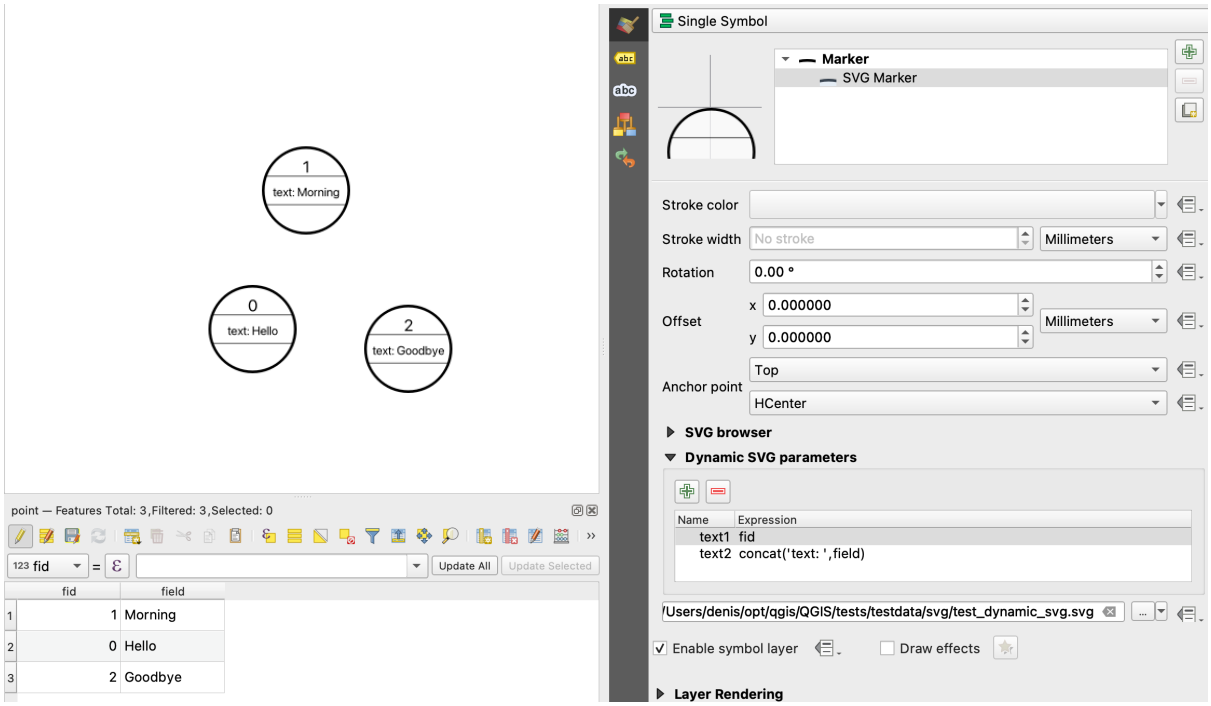


Fig. 10.22: Dynamic SVG parameters table

Note

QGIS is looking for a complete SVG node. So if your parameter is within a more complex node, you need to inject the complete node with the expression. For instance, `transform="rotate(param(angle)"` will not work. Instead, you need to do `transform="param(rotation)"` and `rotation` parameter will be defined with the expression `'rotate(' || coalesce(my_field, 0) || ')'`.

The Geometry Generator

Available with all types of symbols, the *geometry generator* symbol layer allows to use *expression syntax* to generate a geometry on the fly during the rendering process. The resulting geometry does not have to match with the original *Geometry type* and you can add several differently modified symbol layers on top of each other.

A *Units* property can be set: when the geometry generator symbol is not applied to a layer (e.g., it is used on a layout item), this allows more control over the generated output.

Some examples:

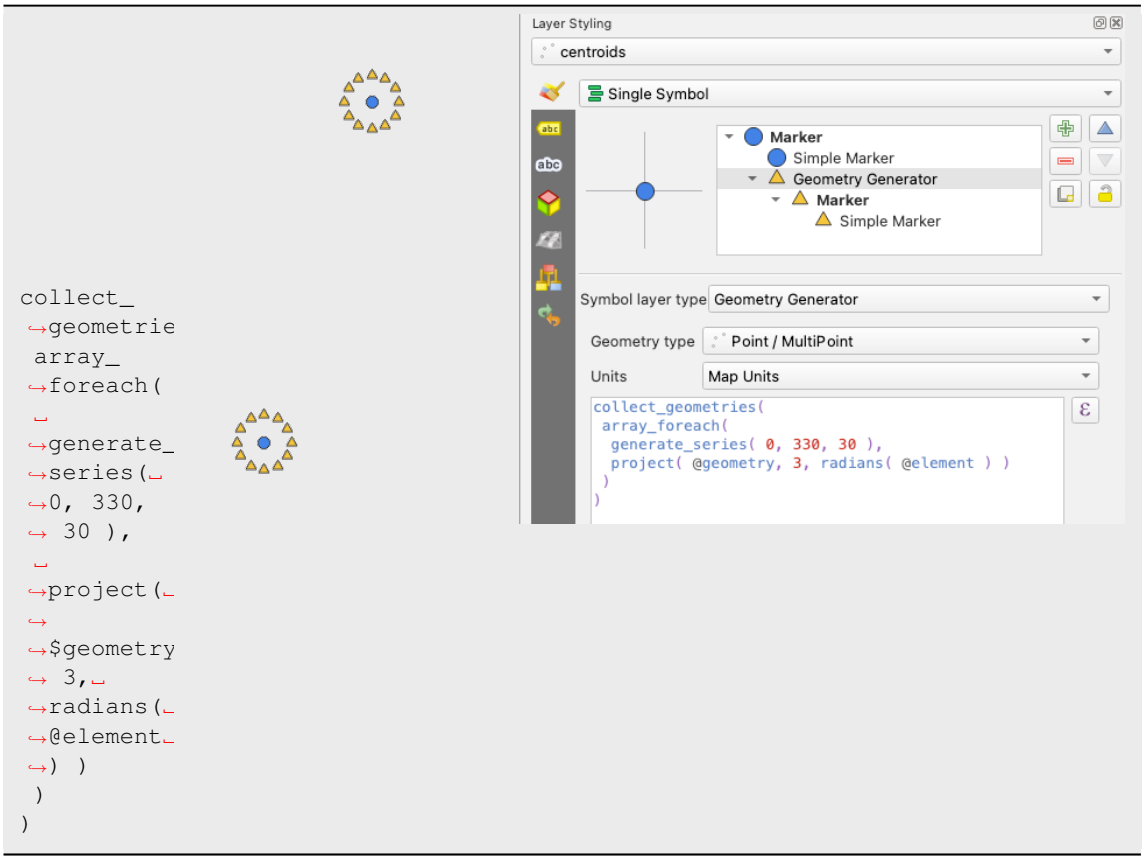
- Render symbol as the centroid of a feature

```
centroid( $geometry )
```

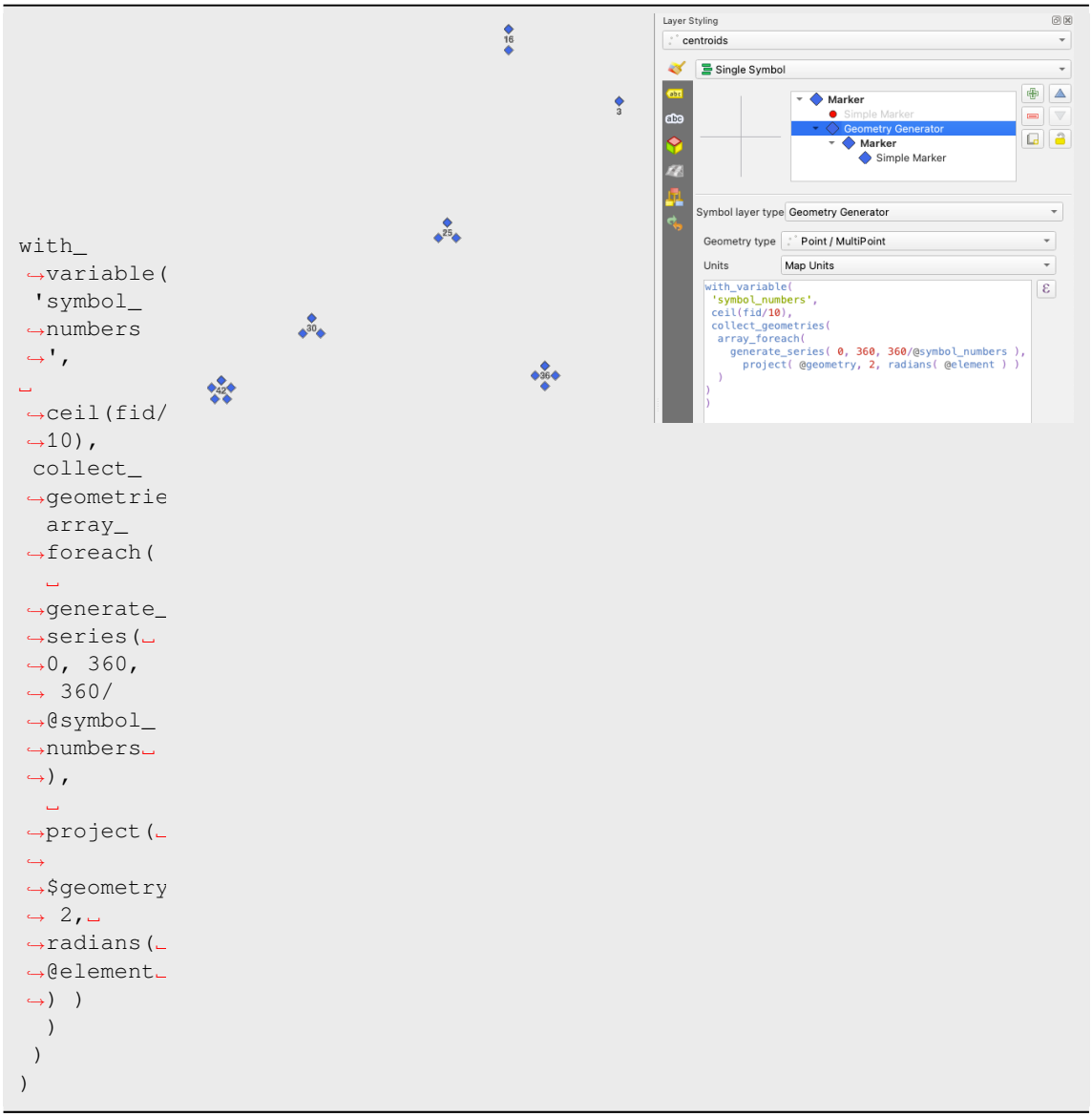
- Visually overlap features within a 100 map units distance from a point feature, i.e generate a 100m buffer around the point

```
buffer( $geometry, 100 )
```

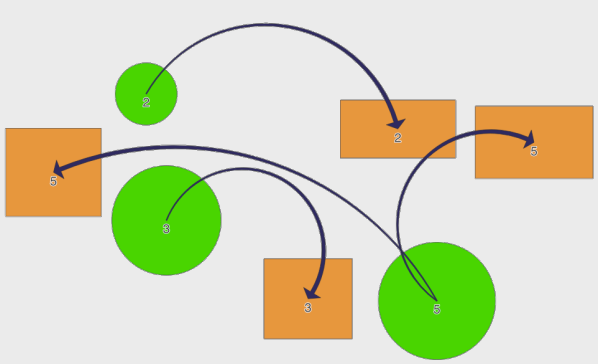
- Create a radial effect of points surrounding the central feature point when used as a MultiPoint geometry generator

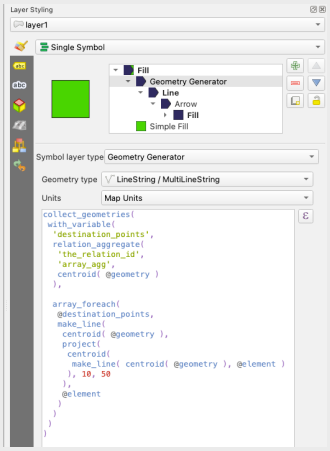


- Create a radial effect of points surrounding the central feature point. The number of points varies based on a field.



- Create a curved arrow line connecting features of two layers based on their *relation*






```
collect_
↪geometrie
 with_
↪variable(

↪'destinat
↪points
↪',
 ␣
↪relation_
↪aggregate
   'the_
↪relation_
↪id',

↪'array_
↪agg',
  ␣
↪centroid(
↪
↪$geometry
↪)
  ),
  array_
↪foreach(
  ␣
↪@destinat
↪points,
   make_
↪line(
   ␣
↪centroid(
↪@geometry
↪),
   ␣
↪project(
    ␣
↪centroid(
     ␣
↪make_
↪line(␣
↪centroid(
↪@geometry
↪),␣
↪@element␣
↪)
     ),
     10,␣
↪50
    ),
   ␣
↪@element
   )
  )
 )
)
```

The Vector Field Marker

The vector field marker is used to display vector field data such as earth deformation, tidal flows, and the like. It displays the vectors as lines (preferably arrows) that are scaled and oriented according to selected attributes of data points. It can only be used to render point data; line and polygon layers are not drawn by this symbology.

The vector field is defined by attributes in the data, which can represent the field either by:

- **cartesian** components (`x` and `y` components of the field)
- or **polar** coordinates: in this case, attributes define `Length` and `Angle`. The angle may be measured either clockwise from north, or Counterclockwise from east, and may be either in degrees or radians.
- or as **height only** data, which displays a vertical arrow scaled using an attribute of the data. This is appropriate for displaying the vertical component of deformation, for example.

The magnitude of field can be scaled up or down to an appropriate size for viewing the field.

Animated marker

Animated marker symbol type allows you to use a `.GIF`, `.WebP`, `.MNG`, etc. animation file to represent points on your map. You can specify:

- *File* path,
- *Frame rate*: number of steps that are shown per second, indicating how fast the animation is played,
- *Size* in any *supported unit*,
- *Opacity*,
- *Rotation*,
- *Offset* in *x* and *y* directions from the marker position,
- *Anchor point*

There are two ways to handle animated symbols:

- **When your map is not configured as an animation** (i.e. it's a standard QGIS project without animations), the frame for the animated markers will be determined solely by the current timestamp.
- **When your map is** *configured as an animation*, the animated markers will sync with the animation's timeline. This means that animated markers will pause when the animation is paused, progress with the animation, and so forth. The map will also be redrawn according to the frame rate established for temporal animation. This mode is also applied when exporting an animation using the temporal controller.

10.3 Setting a label

Labels are textual information you can display on maps. They add details you could not necessarily represent using symbols, and may refer to vector features, raster cells, mesh elements, or simple annotations on the map… Two types of text-related items are available in QGIS:

- *Text Format*: defines the appearance of the text, including *font, size, colors*, *shadow*, *background*, *buffer*, …

  They can be used to render texts over the map (layout/map title, decorations, scale bar, …), usually through the *font* widget.

  To create a *Text Format* item:

  1. Open the  *Style Manager* dialog

2. Activate the *Text format* tab

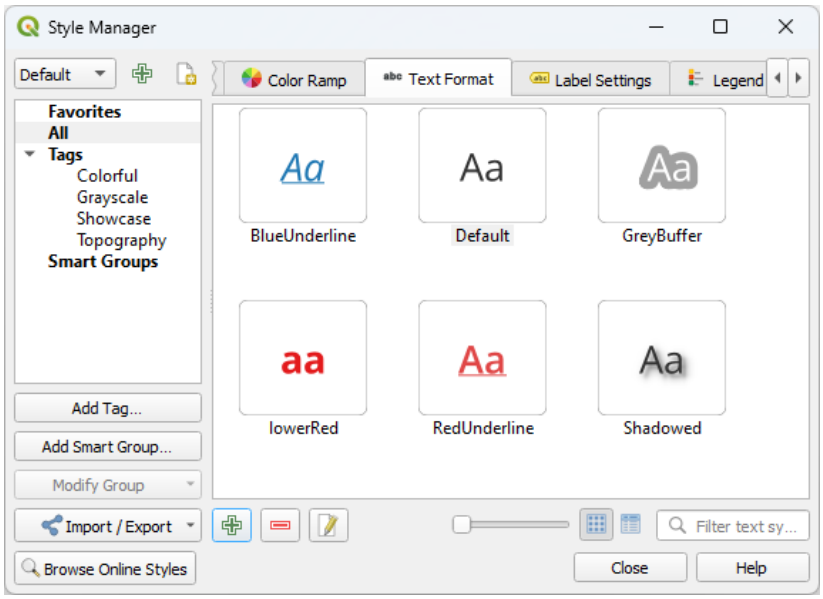


Fig. 10.23: Text formats in Style Manager dialog

3. Press the Add item button. The *Text Format* dialog opens for *configuration*. As usual, these properties are *data-definable*.

- *Label Settings*: extend the text format settings with properties related to the location or the interaction with other texts or features (*callouts*, *placement*, *overlay, scale visibility*, *mask* …).

  They are used to configure smart labelling for vector and mesh layers through the *Labels* tab of the vector or mesh *Layer Properties* dialog or *Layer Styling* panel or using the Layer Labeling Options button of the *Label toolbar*.

  To create a *Label Settings* item:

  1. Open the *Style Manager* dialog
  2. Activate the *Label Settings* tab

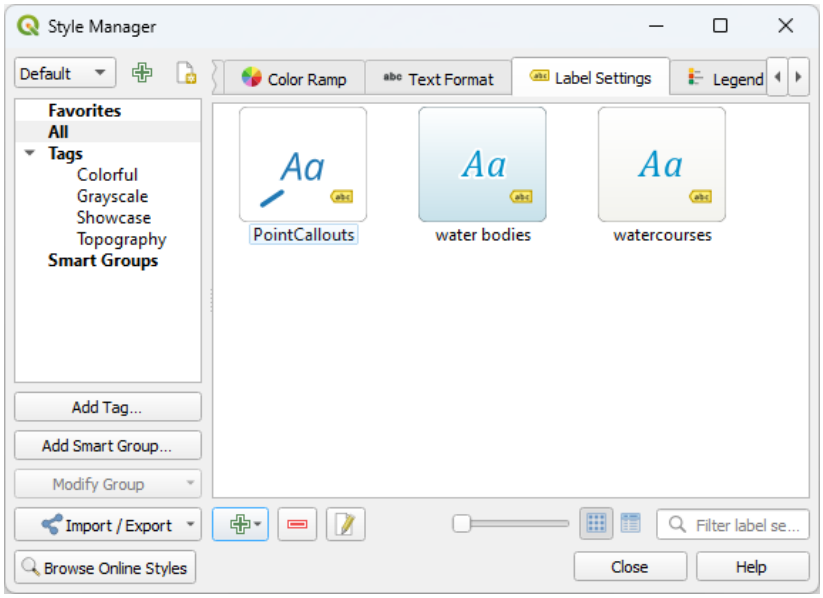


Fig. 10.24: Label Settings in Style Manager dialog

3. Press the Add item menu and select the entry corresponding to the geometry type of the features you want to label.

The *Label Settings* dialog opens with the following properties. As usual, these properties are *data-definable*.

10.3.1 Formatting the label text

Whether you are configuring a *Text Format* or *Label Settings* item, and depending on the type of layer you are configuring, you will be given the following options:

Properties tab	Text format	Label settings	Vector or mesh layer	raster layer
Text	☑	☑	☑	☑
Formatting	☑	☑	☑	☑
Buffer	☑	☑	☑	☑
Mask		☑	☑	
Background	☑	☑	☑	☑
Shadow	☑	☑	☑	☑
Callout		☑	☑	
Placement		☑	☑	☑
Rendering		☑	☑	☑

Attention

While for legibility, “feature” is the name used below to indicate the item being labeled, depending on the underlying layer type, it can be replaced by “pixel”, “face” or “vertex”.

Text tab

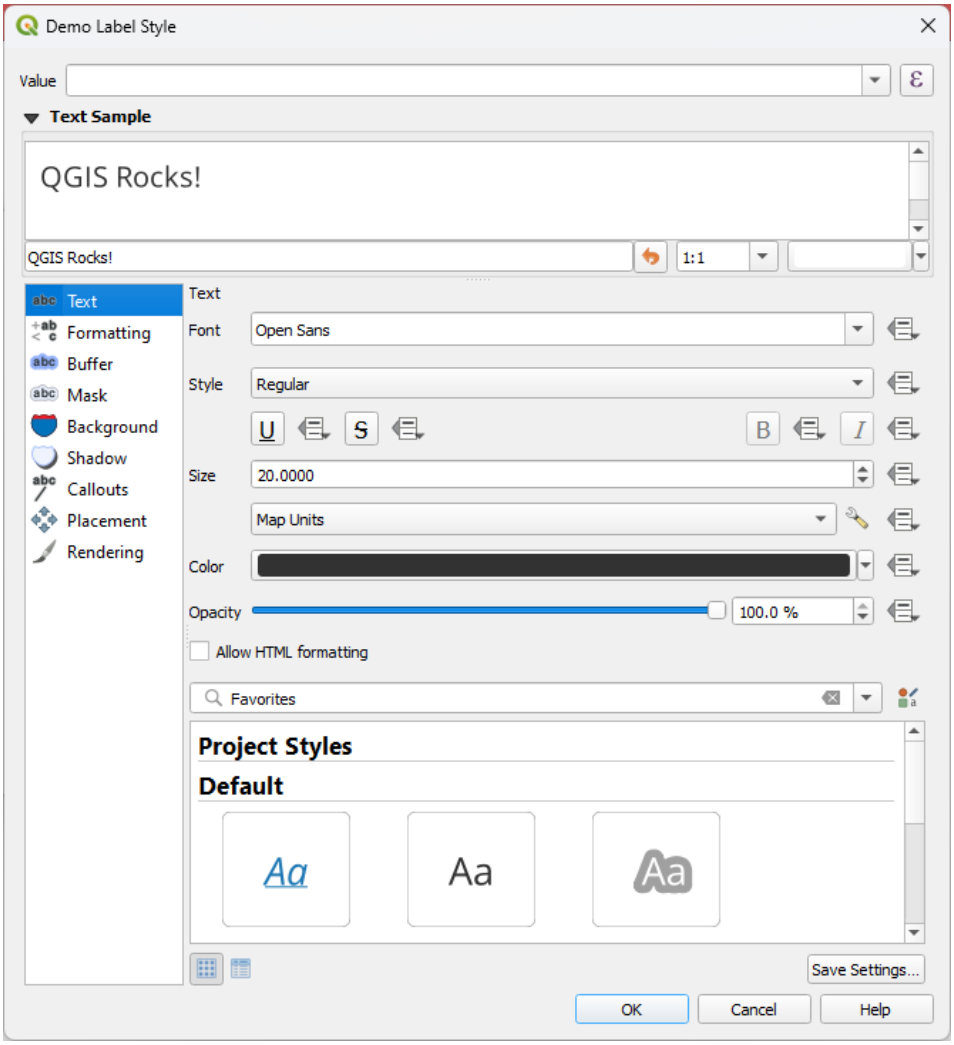


Fig. 10.25: Labels settings - Text tab

In the abc *Text* tab, you can set:

- the *Font*, from the ones available on your machine
- the *Style*: along with the common styles of the font, you can set whether the text should be underlined or striked through
- the *Size* in any *supported unit*
- the *Color*
- the *Opacity*
- and *Allow HTML Formatting* enables the use of a subset of HTML tags and CSS rules to customize the label.

At the bottom of the tab, a widget shows a filterable list of compatible items stored in your *style manager database*. This allows you to easily configure the current text format or label setting based on an existing one, and also save a new item to the style database: Press the *Save format…* or *Save settings…* button and provide a name and tag(s).

Note

When configuring a *Label Settings* item, text format items are also available in this widget. Select one to quickly overwrite the current *textual properties* of the label. Likewise, you can create/overwrite a text format from there.

Allow HTML Formatting

With *Allow HTML Formatting* enabled, you need to provide the HTML code in the *Value* field. Use whitespaces instead of tabs for any kind of indentation. The expression is parsed and any supported HTML tag overrides its corresponding setting in the labels properties. Because it is impossible to list and detail every HTML tag and CSS property that QGIS currently supports, we invite you to explore and test in your labels the ones supported by the underlying Qt library.

Examples of supported HTML tags:

- Text formatting, such as italic or bold, e.g.:

```
<i>QGIS</i> <b>rocks!</b>
```

- Superscript and subscript, where the text will be vertically $^{\text{super}}$ or $_{\text{sub}}$ aligned and automatically sized to 2/3 of the parent font size. You can also set a fixed font size for the superscript/subscript by including css rules, e.g.:

```
<sup style="font-size:33pt">my superscript text</sup>
```

- Text horizontal alignment, using either HTML the `align="xxx"` attribute or `center` tag. For HTML5 compatibility, prefer using the CSS property `text-align`.

> **Attention**
>
> Horizontal alignment can not be used for curved labels.

- Structuring a block of text using header tags (such as `h1`, `h2`, …), or paragraphs (with `p`, `div`, `br`):

```
<div class="myDiv">
  <h2>QGIS always rocks!!</h2>
  <p align="center">Let's dive into details of its nice features.</p>
</div>
```

- Image insertion: any image format readable by QGIS can be used in HTML label content. It can be served from local file paths, HTTP links, or base64 embedded content, using the `src="xxx"` attribute. Image sizes can be specified via the `width="##"` and `height="##"` attributes, in `points` unit. If width or height is not specified it will automatically be calculated from the original image size. Images are placed inline only, not as floating images, and not on curved text labels.

```
<img src="qgis.png" width=40 height=60>
```

Examples of supported CSS properties:

- Font properties (`color`, `font-family`, `font-size`, `font-weight`, `font-style`, `word-spacing`). Note that `word-spacing` will always use unit points.
- Text decorations such as underline, overline and line-through (`text-decoration`)
- Text alignment (`vertical-align`, `text-align`). Horizontal alignment can not be used for curved labels.
- Line height, in `points` or `percent` unit, e.g. “line-height: 40pt” or “line-height: 40%”
- Background properties such as `background-color` and `background-image`. They are supported for block type items (e.g. `div`) or inline items (e.g. `span`). For images, the CSS should be formatted as `background-image: url(xx)` and supports local file paths, HTTP links, or base64 embedded content.

> **Attention**
>
> Backgrounds are not supported for curved text and are always rendered above any background shape for the label, and below drop shadows/buffers properties.

- Margin properties, available for block type items only, such as `div`, `p`, `h1`,... They can be specified in either the longhand or shorthand way, in `points` unit only. Negative values can be set for the bottom margin.

```
<div class="myDiv">
  <h2 style="margin-left: 5pt; margin-right: 10pt">QGIS still rocks...</h2>
  <p style="margin: 5pt 0pt -10pt 0pt">Thanks to you!!</p>
</div>
```

CSS properties can be set on HTML tags with the `style` attribute. The HTML tag `span` does not apply any formatting to text by itself and is ideal if you just want to apply CSS styling. A CSS property name and its value are separated by a colon (`:`). Multiple CSS properties are separated by semicolon (`;`), e.g.:

```
<span style="text-decoration:underline;text-align:center;color:blue;word-spacing:20
↪">I will be displayed as blue underlined and centered text with increased space␣
↪between words</span>
```

Below an example of a HTML-based expression and rendering (applies different colors and underline to the same label):

```
format(
  '<span style="color:blue">%1</span> ( <span style="color:red"><u>%2 ft</u></span>
↪ )',
  title( lower( "Name" ) ),
  round($length)
)
```

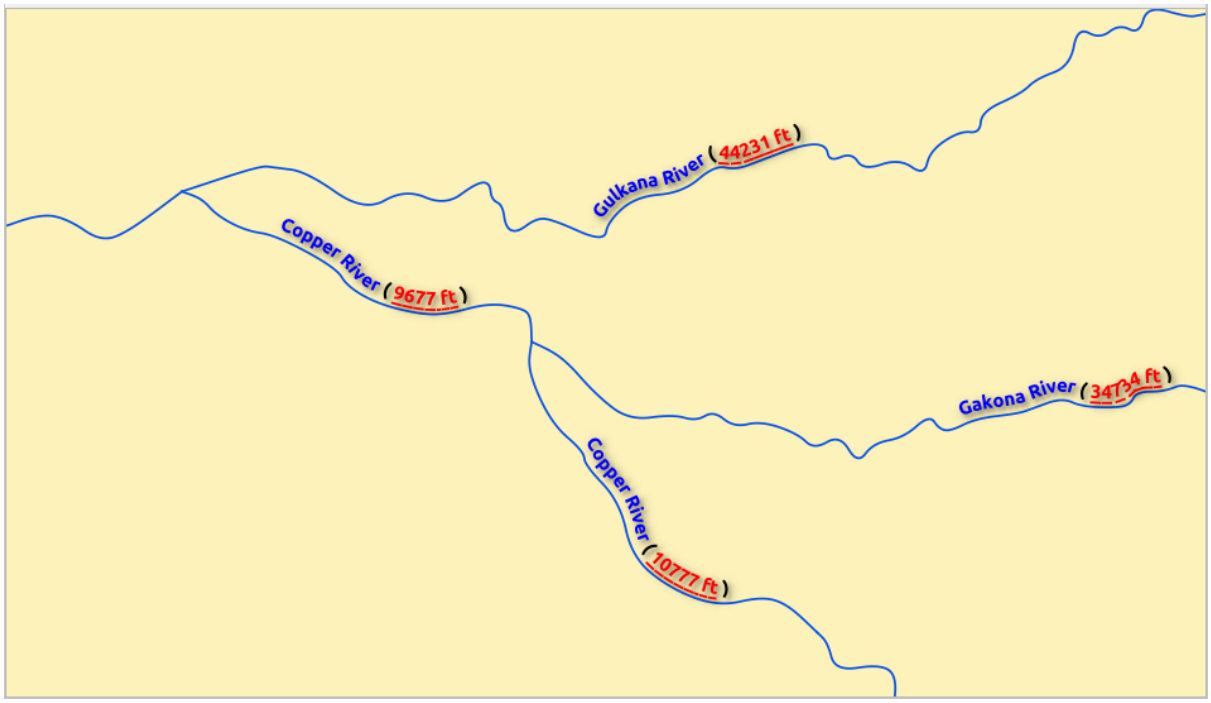


Fig. 10.26: Labeling with HTML formatting enabled

Formatting tab

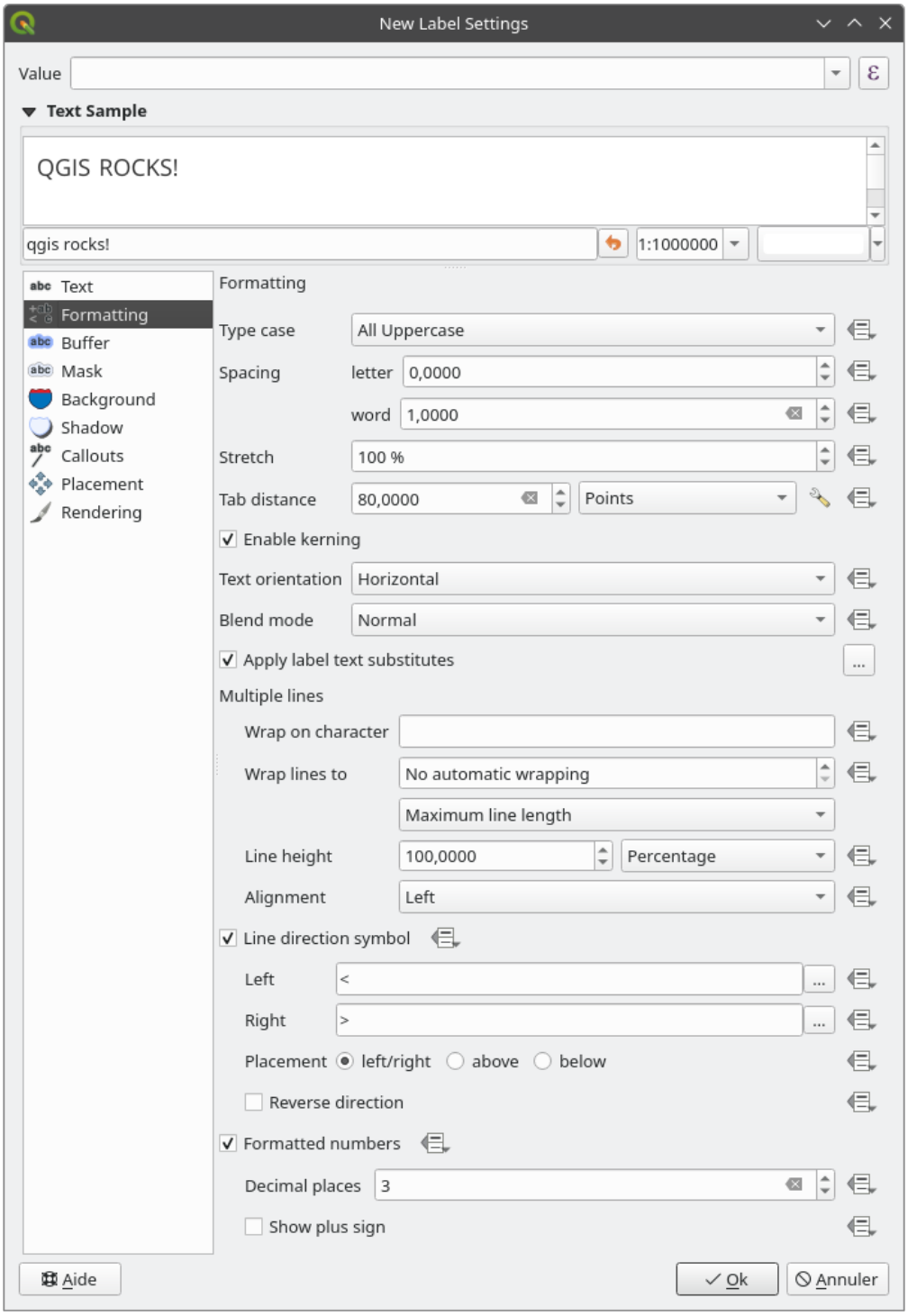


Fig. 10.27: Label settings - Formatting tab

In the *Formatting* tab, you can:

- Use the *Type case* option to change the capitalization style of the text. You have the possibility to render the text as:
  - *No change*

  - *All uppercase*
  - *All lowercase*
  - *Small Caps*: renders lowercase characters as small caps.
  - *All Small Caps*: renders all characters as small caps regardless of their original case.
  - *Title case*: modifies the first letter of each word into capital, and turns the other letters into lower case if the original text is using a single type case. In case of mixed type cases in the text, the other letters are left untouched.
  - *Force first letter to capital*: modifies the first letter of each word into capital and leaves the other letters in the text untouched.
- Under *Spacing*, change the space between words and between individual letters.
- With *Tab distance* you can adjust the replacement spacing for `Tab` characters in the label, in any supported unit. This allows e.g. to properly separate or vertically align elements of a layout legend text.

  Pressing the Configure tab stops button on the right, you can add a list of tab positions, instead of a single distance. This allows e.g. the creation of table-like multiline labels, where texts are split over different "columns" and lined up nicely, based on the `Tab` characters.
- *Stretch* ratio: allows text to be horizontally stretched or condensed by a factor. Handy for tweaking the widths of fonts to fit a bit of extra text into labels.
- *Enable kerning* of the text font
- Set the *Text orientation* which can be *Horizontal* or *Vertical*. It can also be *Rotation-based* when setting a label (e.g., to properly label line features in *parallel* placement mode).
- Use the *Blend mode* option to determine how your labels will mix with the map features below them (more details at *Blending Modes*).
- The *Apply label text substitutes* option allows you to specify a list of texts to substitute to texts in feature labels (e.g., abbreviating street types). Replacement texts are used when displaying labels on the map. Users can also export and import lists of substitutes to make reuse and sharing easier.
- Configure *Multiple lines*:
  - Set a character that will force a line break in the text with the *Wrap on character* option
  - Set an ideal line size for auto-wrapping using the *Wrap lines to* option. The size can represent either the *Maximum line length* or the *Minimum line length*.
  - Decide the *Line Height*: values can be set to be in *Millimeters*, *Points*, *Pixels*, *Percentage*, or *Inches*. When line height is set to percentage it is the percentage of the default text line spacing of that font family. Typically 1.2 to 1.5 times the text size.
  - Format the *Alignment*: typical values available are *Left*, *Right*, *Justify* and *Center*.

    When setting point labels properties, the text alignment can also be *Follow label placement*. In that case, the alignment will depend on the final placement of the label relative to the point. E.g., if the label is placed to the left of the point, then the label will be right aligned, while if it is placed to the right, it will be left aligned.

> **Note**
>
> The *Multiple lines* formatting is not yet supported by curve based *label placement*. The options will then be deactivated.

- For line labels you can include *Line direction symbol* to help determine the line directions, with symbols to use to indicate the *Left* or *Right*. They work particularly well when used with the *curved* or *Parallel* placement options from the *Placement* tab. There are options to set the symbols position, and to *Reverse direction*.

- Use the *Formatted numbers* option to format numeric texts. You can set the number of *Decimal places*. By default, 3 decimal places will be used. Use the *Show plus sign* if you want to show the plus sign for positive numbers.

Buffer tab

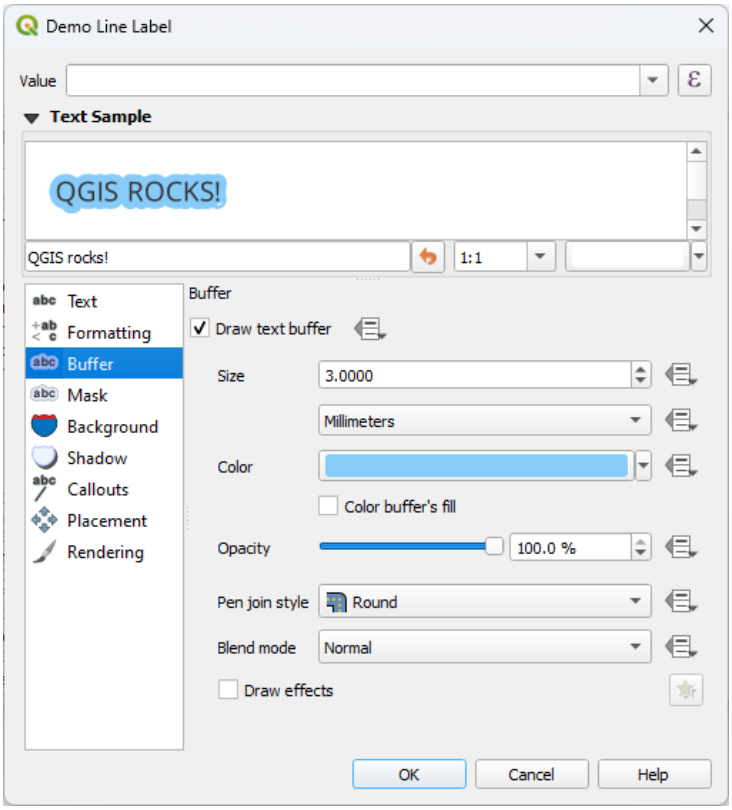


Fig. 10.28: Label settings - Buffer tab

To create a buffer around the label, activate the *Draw text buffer* checkbox in the *Buffer* tab. Then you can:

- Set the buffer's *Size* in any *supported unit*
- Select the buffer's *Color*
- *Color buffer's fill*: The buffer expands from the label's outline, so, if the option is activated, the label's interior is filled. This may be relevant when using partially transparent labels or with non-normal blending modes, which will allow seeing behind the label's text. Unchecking the option (while using totally transparent labels) will allow you to create outlined text labels.
- Define the buffer's *Opacity*
- Apply a *Pen join style*: it can be *Round*, *Miter* or *Bevel*
- Use the *Blend mode* option to determine how your label's buffer will mix with the map components below them (more details at *Blending Modes*).
- Check *Draw effects* to add advanced *paint effects* for improving text readability, eg through outer glows and blurs.

Background tab

The *Background* tab allows you to configure a shape that stays below each label. To add a background, activate the *Draw Background* checkbox and select the *Shape* type. It can be:

- a regular shape such as *Rectangle*, *Square*, *Circle* or *Ellipse* using full properties of a *fill symbol*
- an *SVG* symbol from a file, a URL or embedded in the project or style database (*more details*)
- or a *Marker Symbol* you can create or select from the *symbol library*.

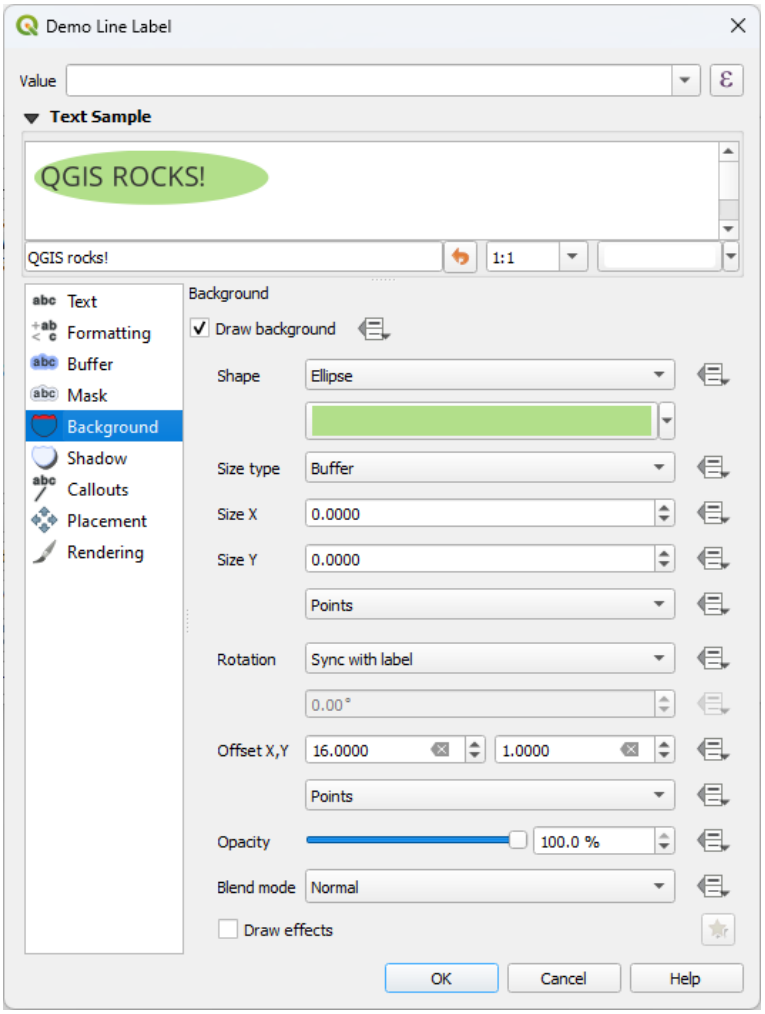


Fig. 10.29: Label settings - Background tab

Depending on the selected shape, you need to configure some of the following properties:

- The *Size type* of the frame, which can be:
  - *Fixed*: using the same size for all the labels, regardless the size of the text
  - or a *Buffer* over the text's bounding box
- The *Size* of the frame in X and Y directions, using any *supported units*
- A *Rotation* of the background, between *Sync with label*, *Offset of label* and *Fixed*. The last two require an angle in degrees.
- An *Offset X,Y* to shift the background item in the X and/or Y directions
- A *Radius X,Y* to round the corners of the background shape (applies to rectangle and square shapes only)

- An *Opacity* of the background
- A *Blend mode* to mix the background with the other items in the rendering (see *Blending Modes*).
- For SVG symbol, you can use its default properties (*Load symbol parameters*) or set a custom *Fill color*, *Stroke color* and *Stroke width*.
- *Draw effects* to add advanced *paint effects* for improving text readability, eg through outer glows and blurs.

Shadow tab

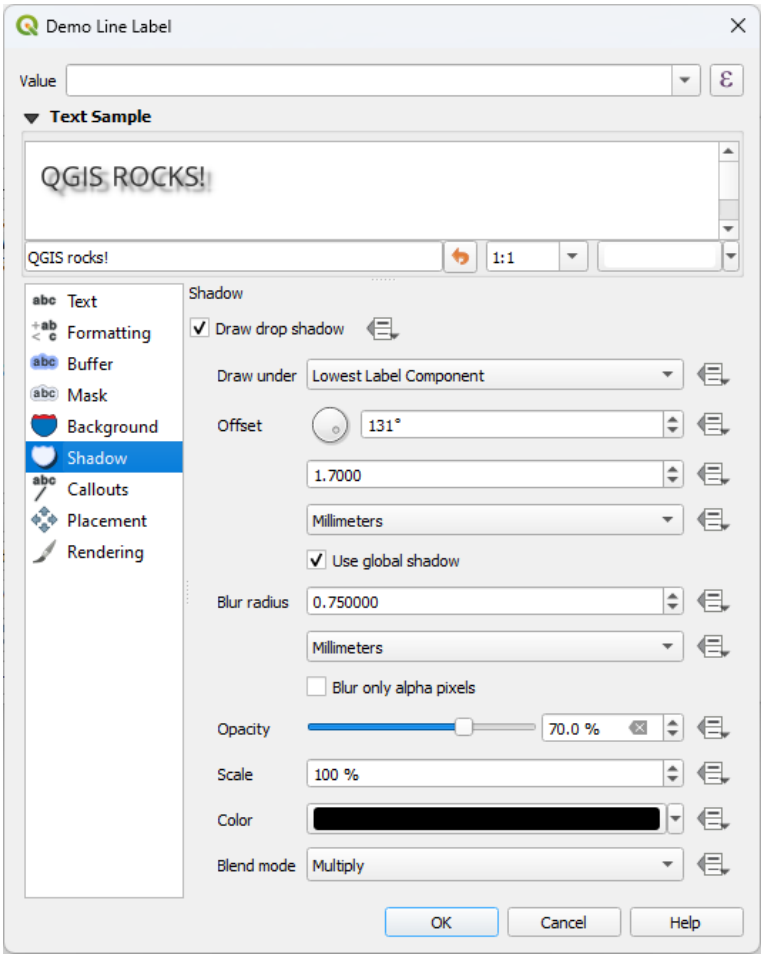


Fig. 10.30: Label settings - Shadow tab

To add a shadow to the text, enable the *Shadow* tab and activate the *Draw drop shadow*. Then you can:

- Indicate the item used to generate the shadow with *Draw under*. It can be the *Lowest label component* or a particular component such as the *Text* itself, the *Buffer* or the *Background*.
- Set the shadow's *Offset* from the item being shadowed, ie:
  - The angle: clockwise, it depends on the underlying item orientation
  - The distance of offset from the item being shadowed
  - The units of the offset

  If you tick the *Use global shadow* checkbox, then the zero point of the angle is always oriented to the north and doesn't depend on the orientation of the label's item.

- Influence the appearance of the shadow with the *Blur radius*. The higher the number, the softer the shadows, in the units of your choice.
- Define the shadow's *Opacity*
- Rescale the shadow's size using the *Scale* factor
- Choose the shadow's *Color*
- Use the *Blend mode* option to determine how your label's shadow will mix with the map components below them (more details at *Blending Modes*).

10.3.2 Configuring interaction with labels

Other than the text formatting settings exposed above, you can also set how labels interact with each others or with the features.

Mask tab

The *Mask* tab allows you to define a mask area around the labels. This feature is very useful when you have overlapping symbols and labels with similar colors, and you want to make the labels visible. A label mask prevents specified features from drawing within the boundary set for the mask. For example, you could set a label mask so that a specified layer does not draw within 2mm of the label, but allow features from another layer to still show. Label masks are similar to label buffers in that they allow control of the legibility of labels that cover other features. The label buffer draws on top of any underlying features, while the label mask selectively stops other layers from drawing.

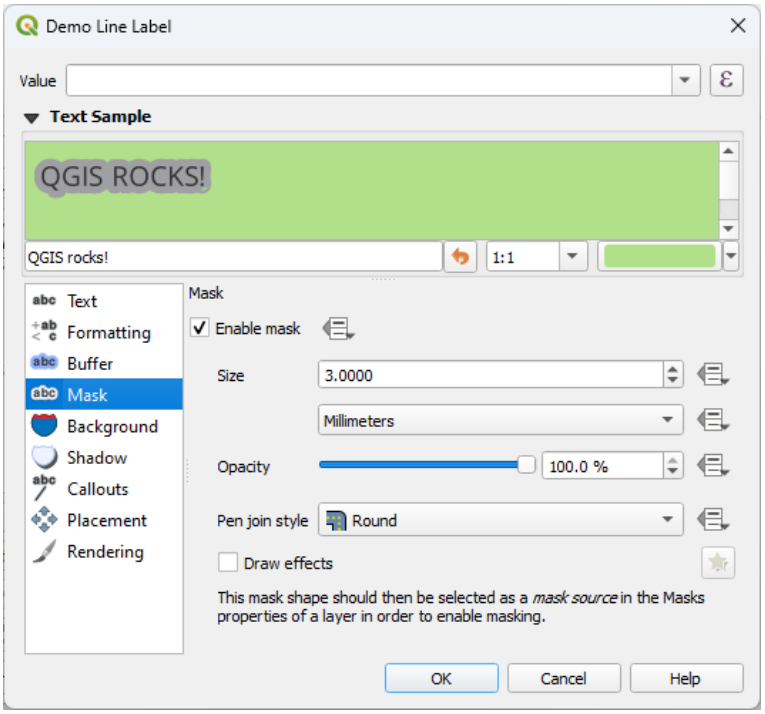


Fig. 10.31: Labels settings - Mask tab (with the text sample showing a green background representing another layer being excluded)

To create masking effects on labels:

1. Activate the *Enable mask* checkbox in the tab.
2. Then you can set:

- the mask's *Size* in the *supported units*
- the *Opacity* of the mask area around the label
- a *Pen Join Style*
- *paint effects* through the *Draw effects* checkbox.

3. Select this mask shape as a mask source in the overlapping layer properties *Mask* tab (see *Masks Properties*).

Callouts tab

A common practice when placing labels on a crowded map is to use **callouts** - labels which are placed outside (or displaced from) their associated feature are identified with a dynamic line connecting the label and the feature. If one of the two endings (either the label or the feature) is moved, the shape of the connector is recomputed.



Fig. 10.32: Labels with various callouts settings

To add a callout to a label, enable the *Callouts* tab and activate the *Draw callouts*. Then you can:

1. Select the *Style* of connector, one of:
   - *Simple lines*: a straight line, the shortest path
   - *Manhattan style*: a 90° broken line
   - *Curved lines*: a curved line
   - *Balloons*: a speech bubble surrounding the label and pointing to the feature. It can have rounded corners.
2. Set the properties of the callout. The following table shows the different properties, with description and compatible connector style.

Table 10.2: Label callout properties

Property	Style of callout	Description
Fill style	Balloons	A *fill symbol* with full display capabilities, including layer effects, data-defined settings, … for drawing the balloon shape.
Corner radius		Corner radius of the speech bubble
Wedge width		Sets how large the bubble speech connection with feature's pointer should be
Margins		Margins around the label's text, in the unit of your choice
Line style	All but balloons	A *line symbol* with full display capabilities, including layer effects, data-defined settings, … for drawing the connector line.
Curvature	Curved lines	The percentage of curvature of the connection line
Orientation		Orientation, starting from the label to the feature. It can be *Clockwise*, *Counter-clockwise*, or *Automatic* (determining an optimal orientation for each label).
Minimum length	All but balloons	Minimum length of the connector line
Draw lines to all feature parts		In case of a multi-part feature, indicates whether a connector line should be drawn from the label to each geometry part.
Label anchor point		Controls where the connector line should join to the label text. Available options: • *Closest point* • *Centroid* • Fixed position at the edge (*Top left*, *Top center*, *Top right*, *Left middle*, *Right middle*, *Bottom left*, *Bottom center* and *Bottom right*).
Offset from label area		Controls the distance from the label anchor point (where the callout line ends). This avoids drawing lines right up against the text.
Offset from feature	All	Controls the distance from the feature (or its anchor point if a polygon) where callout lines end. E.g., this avoids drawing lines right up against the edges of the features.
Feature anchor point		Where the connector line ends on the (polygon) feature. Available options: • *Pole of inaccessibility* • *Point on exterior* • *Point on surface* • *Centroid*
End point marker	Balloons	A *marker symbol* with full display capabilities including layer effects, data-defined, … for rendering a marker symbol below the

Under the *Data defined placement* group, coordinates of the *Origin* (on the label side) and/or *Destination* (on the feature side) points of the callout can be controlled. Callouts can also be controlled manually by using the  Move Label, Diagram or Callout tool in the *Labeling Toolbar*. The start and end points of each callout can be moved this way. The nodes should be highlighted when the mouse pointer is nearby. If needed the `Shift` Key can be held during the movement. This will snap the point in a way that the angle between the two callout points increments by 15 degrees.

Placement tab

Choose the  *Placement* tab for configuring label placement and labeling priority. Note that the placement options differ according to the type of vector or mesh layer, namely point, line or polygon, and are affected by the global *PAL setting*.

Placement for point layers

Point labels placement modes available are:

- *Cartographic*: point labels are generated with a better visual relationship with the point feature, following ideal cartographic placement rules. Labels can be placed:
  - at a set *Distance* in *supported units*, either from the point feature itself or from the bounds of the symbol used to represent the feature (set in *Distance offset from*). The latter option is especially useful when the symbol size isn't fixed, e.g. if it's set by a data defined size or when using different symbols in a *categorized* renderer.
  - within a *Maximum Distance* from the feature, which is an optional setting that allows you to control how far a label can be placed from the feature it's labeling. This works alongside the *Distance* setting to create a range for label placement, adding flexibility to position labels more effectively, especially on busy maps, ensuring they fit neatly around their corresponding features.
  - using the *Prioritize Placement* option, which decides what's more important when placing labels. There are two options:
    - *Prefer closer labels*: By default, labels are kept close to the feature.
    - *Prefer position ordering*: The label will try to stay in a specific position (like top left or top right), even if it's a bit farther away from the feature. The label only moves to other positions if there's no room within the maximum distance at your preferred position.
  - following a *Position priority* which dictates placement candidates for anchoring labels around and (centered) over the point feature, and the order in which the positions are tested. The default order, based on guidelines from Krygier and Wood (2011) and other cartographic textbooks, is as follows:
    1. top right
    2. top left
    3. bottom right
    4. bottom left
    5. middle right
    6. middle left
    7. top, slightly right
    8. bottom, slightly left.

    Using the  Data-defined override button, you can provide a comma separated list of placements in order of priority. This also allows only certain placements to be used, for certain features only, so e.g., for coastal features you can prevent labels being placed over the land.

- *Around Point*: labels are placed in a circle around the feature with an equal radius set in *Distance*. Additionally you can set *Maximum Distance* from the feature, to control how far a label can be placed from the feature it's labeling. The placement priority is clockwise from the "top right". The position can be constrained using the data-defined *Quadrant* option.
- *Offset from Point*: labels are placed at an *Offset X,Y* distance from the point feature, in various units, or preferably over the feature. You can use a data-defined *Quadrant* to constrain the placement and can assign a *Rotation* to the label.

Placement for line layers

Label modes for line layers include:

- *Parallel*: draws the label parallel to a generalised line representing the feature, with preference for placement over straighter portions of the line. You can define:
  - *Allowed positions*: *Above line*, *On line*, *Below line* and *Line orientation dependent position* (placing the label at the left or the right of the line). It's possible to select several options at once. In that case, QGIS will look for the optimal label position.
  - *Distance* between the label and the line
- *Curved*: draws the label following the curvature of the line feature. In addition to the parameters available with the *Parallel* mode, you can set the *Maximum angle between curved characters*, either inside or outside.
- *Horizontal*: draws labels horizontally along the length of the line feature.

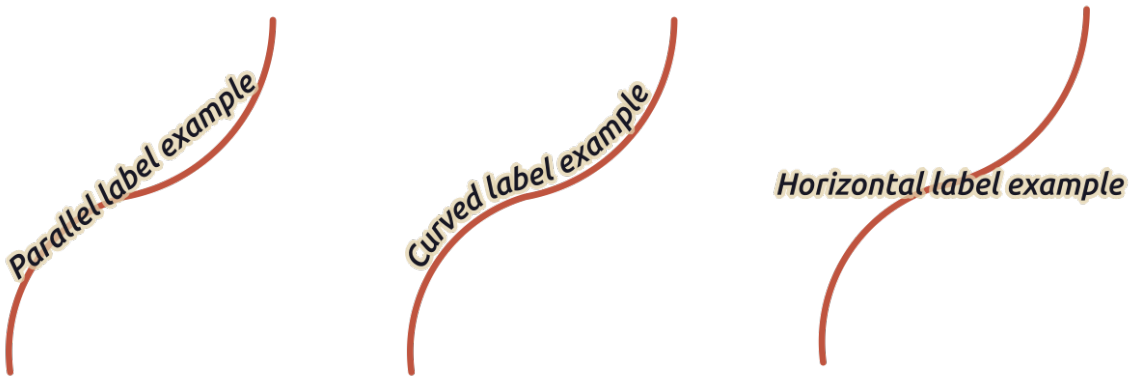


Fig. 10.33: Label placement examples for lines

Next to placement modes, you can set:

- *Repeating Labels Distance* to display multiple times the label over the length of the feature. The distance can be in `Millimeters`, `Points`, `Pixels`, `Meters at scale`, `Map Units` and `Inches`.
- A *Label Overrun Distance* (not available for horizontal mode): specifies the maximal allowable distance a label may run past the end (or start) of line features. Increasing this value can allow for labels to be shown for shorter line features.
- *Label Anchoring*: controls the placement of the labels along the line feature they refer to. Click on *Settings* ... to choose:
  - the position along the line (as a ratio) which labels will be placed close to. It can be data-defined and possible values are:
    -  *Center of Line*
    -  *Start of Line*

    -  *End of Line*
    - or  *Custom…*.
  - *Clipping*: Determines how the label placement on a line is calculated. By default only the visible extent of the line is used but the whole extent can be used to have more consistent results.
  - *Anchor text*: controls which part of the text (start, center or end) will line up with the anchor point. Using *Automatic* anchoring means that:
    - For labels anchored near the start of the line (0-25%), the anchor placement will be the **start** of the label text
    - For labels anchored near the end of the line (75-100%), the anchor placement will be the **end** of the label text
    - For labels anchored near the center of the line (25-75%), the anchor placement will be the **center** of the label text
  - *Placement Behavior*: use *Preferred Placement Hint* to treat the label anchor only as a hint for the label placement. By choosing *Strict*, labels are placed exactly on the label anchor.

Placement for polygon layers

You can choose one of the following modes for placing labels of polygons:

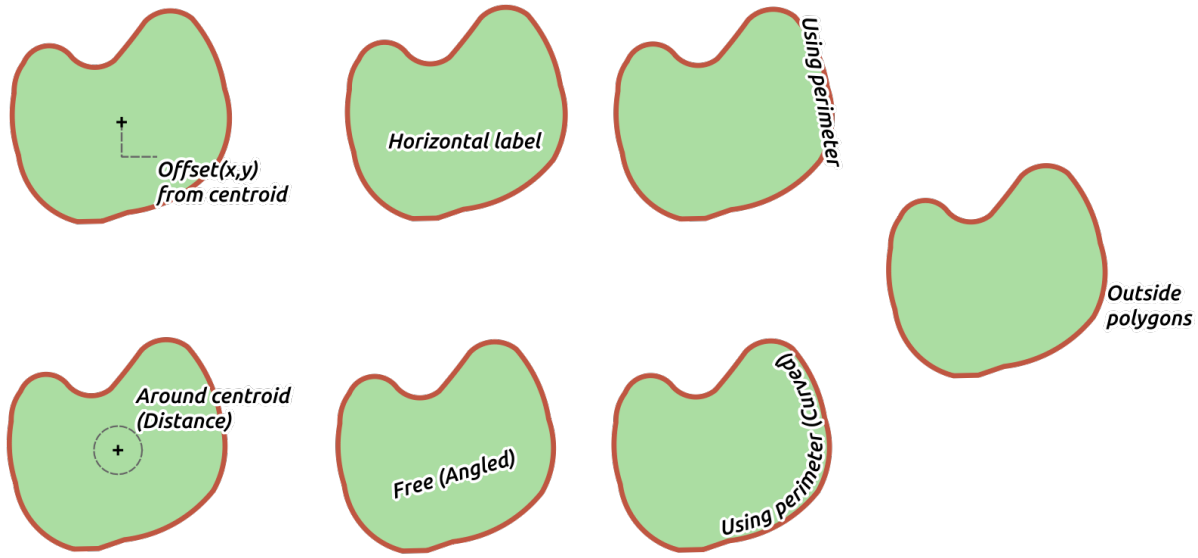


Fig. 10.34: Label placement examples for polygons

- *Offset from Centroid*: labels are placed over the feature centroid or at a fixed *Offset X,Y* distance (in *supported units*) from the centroid. The reference centroid can be determined based on the part of the polygon rendered in the map canvas (*visible polygon*) or the *whole polygon*, no matter if you can see it. You can also:
  - force the centroid point to lay inside their polygon
  - place the label within a specific quadrant
  - assign a rotation
  - *Allow placing labels outside of polygons* when it is not possible to place them inside the polygon. Thanks to data-defined properties, this makes possible to either allow outside labels, prevent outside labels, or force outside labels on a feature-by-feature basis.

- *Around Centroid*: places the label within a preset distance around the centroid, with a preference for the placement directly over the centroid. Again, you can define whether the centroid is the one of the *visible polygon* or the *whole polygon*, and whether to force the centroid point inside the polygon.
- *Horizontal*: places at the best position a horizontal label inside the polygon. The preferred placement is further from the edges of the polygon. It's possible to *Allow placing labels outside of polygons*.
- *Free (Angled)*: places at the best position a rotated label inside the polygon. The rotation respects the polygon's orientation and the preferred placement is further from the edges of the polygon. It's possible to *Allow placing labels outside of polygons*.
- *Using Perimeter*: draws the label parallel to a generalised line representing the polygon boundary, with preference for straighter portions of the perimeter. You can define:
  - *Allowed positions*: *Above line*, *On line*, *Below line* and *Line orientation dependent position* (placing the label at the left or the right of the polygon's boundary). It's possible to select several options at once. In that case, QGIS will look for the optimal label position.
  - *Distance* between the label and the polygon's outline
  - the *Repeating Labels Distance* to display multiple times the label over the length of the perimeter.
- *Using Perimeter (Curved)*: draws the label following the curvature of the polygon's boundary. In addition to the parameters available with the *Using Perimeter* mode, you can set the *Maximum angle between curved characters polygon*, either inside or outside.
- *Outside Polygons*: always places labels outside the polygons, at a set *Distance*

Common placement settings

Some label placement settings are available for all layer geometry types:

Label Spacing

The *Label Spacing* option allows you to set the margin around the label, preventing all other labels from being placed within that area. You can dynamically control *Margin around label* using the Data-defined override button. Choose unit from the unit selector, and there you can also *adjust scaling range*.

> **Attention**
>
> This option requires a build based on GEOS >= 3.10, it may not be available on your QGIS installation depending on the underlying GEOS version in use.

Duplicate Labels

Use *Avoid duplicate labels* to improve label appearance in situations like road networks, where features such as dual carriageways or service roads can result in many identical labels appearing close together. When active, it prevents placing labels with the exact same text (case-sensitive) closer than the minimum distance defined in the *Settings…* option. The setting applies across layers, so duplicate text from any layer will be considered.

> **Attention**
>
> This option requires a build based on GEOS >= 3.10, it may not be available on your QGIS installation depending on the underlying GEOS version in use.

Geometry Generator

The *Geometry Generator* section allows a user to alter the underlying geometry used to place and render the label, by using *expressions*. This can be useful to perform displacement of the geometry dynamically or to convert it to another geometry (type).

In order to use the geometry generator:

1. Check the *Geometry generator* option
2. Enter the expression generating the geometry to rely on
3. If relevant, select the geometry type of the expression output: the label geometry-based settings such as placement or rendering are updated to match the new geometry type capabilities.

Some use cases include:

- Use a geometry which is saved in another field "label_position"
- Use the *generated geometry* from the symbology also for labeling
- Use the @map_scale variable to calculate distances / sizes be zoom level independent.
- Combined with the curved placement mode, creates a circular label around a point feature:

```
exterior_ring(make_circle($geometry, 20))
```

- Add a label at the start and the end of a line feature:

```
collect_geometries( start_point($geometry), end_point($geometry) )
```

- Rely on a smoothed line of a river to get more room for label placement:

```
smooth( $geometry, iterations:=30, offset:=0.25, min_length:=10 )
```

Data Defined

The *Data Defined* group provides direct control on labels placement, on a feature-by-feature basis. It relies on their attributes or an expression to set:

- the *X* and *Y* coordinate
- the text alignment over the custom position set above:
  - *Horizontal*: it can be **Left**, **Center** or **Right**
  - the text *Vertical*: it can be **Bottom**, **Base**, **Half**, **Cap** or **Top**
- the text *Rotation*. Rotation is defined as clockwise angle with 0° pointing in the direction of East for `Horizontal` oriented text and with 0° pointing in North direction for `Vertical` oriented text. Different units can be defined for the labeling rotation (e.g. `degrees`, `minutes of arc`, `turns`). QGIS prioritizes screen display rotation angles by default and always rotates text to be most readable on screen, so "upside-down" rotation angles are overridden while rendering. To change this behavior and force QGIS to render exact data-defined rotation angles there is an option in *Rendering* tab, under *Show upside-down labels*. Check the *Preserve data rotation values* entry if you want to keep the rotation value in the associated field and apply it to the label, whether the label is pinned or not. If unchecked, unpinning the label rotation is reset and its value cleared from the attribute table.

> **Note**
>
> Data-defined rotation with polygon features is currently supported only with the *Around centroid* placement mode.

> **Note**
>
> Expressions can not be used in combination with the labels map tools (ie the *Rotate label* and *Move label* tools) to *data-define* labels placement. The widget will be reset to the corresponding *auxiliary storage field*.

Priority

In the *Priority* section you can define the placement priority rank of each label, ie if there are different diagrams or labels candidates for the same location, the item with the higher priority will be displayed and the others could be left out.

The priority rank is also used to evaluate whether a label could be omitted due to a greater weighted *obstacle feature*.

Obstacles

In some contexts (eg, high density labels, overlapping features…), the labels placement can result in labels being placed over unrelated features.

An obstacle is a feature over which QGIS avoids placing other features' labels or diagrams. This can be controlled from the *Obstacles* section:

1. Activate the *Features act as obstacles* option to decide that features of the layer should act as obstacles for any label and diagram (including items from other features in the same layer).

   Instead of the whole layer, you can select a subset of features to use as obstacles, using the Data-defined override control next to the option.

2. Use the *Settings* button to tweak the obstacle's weighting.

   - For every potential obstacle feature you can assign an *Obstacle weight*: any *label* or *diagram* whose placement priority rank is greater than this value can be placed over. Labels or diagrams with lower rank will be omitted if no other placement is possible.

     This weighting can also be data-defined, so that within the same layer, certain features are more likely to be covered than others.

   - For polygon layers, you can choose the kind of obstacle the feature is:

     – **over the feature's interior**: avoids placing labels over the interior of the polygon (prefers placing labels totally outside or just slightly inside the polygon)

     – or **over the feature's boundary**: avoids placing labels over the boundary of the polygon (prefers placing labels outside or completely inside the polygon). This can be useful for layers where the features cover the whole area (administrative units, categorical coverages, …). In this case, it is impossible to avoid placing labels within these features, and it looks much better when placing them over the boundaries between features is avoided.

Rendering tab

In the *Rendering* tab, you can tune when the labels can be rendered and their interaction with other labels and features.

Label options

Under *Label options*:

- You find the *scale-based* and the *Pixel size-based* visibility settings.
- The *Label z-index* determines the order in which labels are rendered, as well in relation with other feature labels in the layer (using data-defined override expression), as with labels from other layers. Labels with a higher z-index are rendered on top of labels (from any layer) with lower z-index.

  Additionally, the logic has been tweaked so that if two labels have matching z-indexes, then:

  – if they are from the same layer, the smaller label will be drawn above the larger label

  – if they are from different layers, the labels will be drawn in the same order as their layers themselves (ie respecting the order set in the map legend).

  Note

  This setting doesn't make labels to be drawn below the features from other layers, it just controls the order in which labels are drawn on top of all the layers' features.

- *Allow inferior fallback placements*: By default QGIS tries to render labels at their best placement, following your settings. Check this mode to allow features to fallback to worse placement options when there's no other choice, e.g. when a line is too short to fit a curved label text then the label may be placed horizontally just over the feature's center point.
- With data-defined expressions in *Show label* and *Always Show* you can fine tune which labels should be rendered.
- Allow to *Show upside-down labels*: alternatives are **never**, **when rotation defined** or **always**.
  – **never** - default setting, screen readability is prioritized,
  – **when rotation defined** - label rotation should be defined under *Placement* tab, within the *Data Defined* group
  – **always** - upside-down labels are allowed

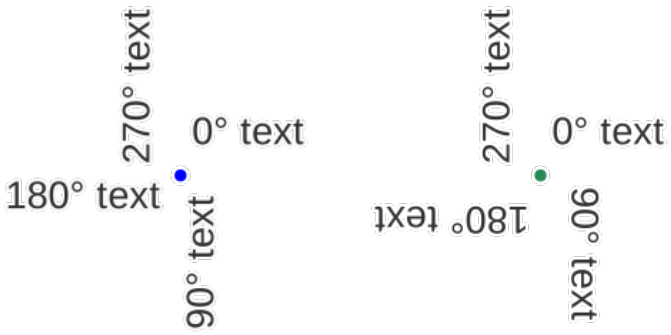


Fig. 10.35: Data defined label rotation with show upside-down labels option set to: "never" (left), "when rotation defined" (right)

- The *Overlapping labels* group allows you to control whether overlapping labels are permitted for features in the layer and how each of them should be handled:
  – *Never overlap*: never ever place overlapping labels for the layer, even if it means some labels will be missing

- *Allow overlaps if required*: if the label can't otherwise be placed, draw an overlapping label. This mode will cause the label to be moved to a less ideal placement if possible, e.g. moving the label further from the center of a line or polygon, IF doing so will avoid overlapping labels. But if there's no other positions possible, then draw the label overlapping.
- *Allow overlaps without penalty*: It doesn't matter at all if the label overlaps other labels or obstacles, that's fine to do and the best placement (e.g most central placement) should always be used even if an alternate further placement is possible which avoids overlaps entirely.

Allowing both overlapping labels and fallback placements options will guarantee that all features in the layer are labeled… not necessarily at their best rendering!

Feature options

Under *Feature options*:

- You can choose to *Label every part of a multi-part features* and *Limit number of features to be labeled to*.
- Both line and polygon layers offer the option to set a minimum size for the features to be labeled, using *Suppress labeling of features smaller than*.
- For polygon features, you can also filter the labels to show according to whether they completely fit within their feature or not.
- For line features, you can choose to *Merge connected lines to avoid duplicate labels*, rendering a quite airy map in conjunction with the *Distance* or *Repeat* options in the *Placement* tab.

10.4 Creating 3D Symbols

The *Style Manager* helps you create and store 3D symbols for every geometry type to render in the *3D map view*.

As of the other items, enable the  *3D Symbols* tab and expand the  button menu to create:

- *3D point symbols*
- *3D line symbols*
- *3D polygon symbols*

10.4.1 Point Layers

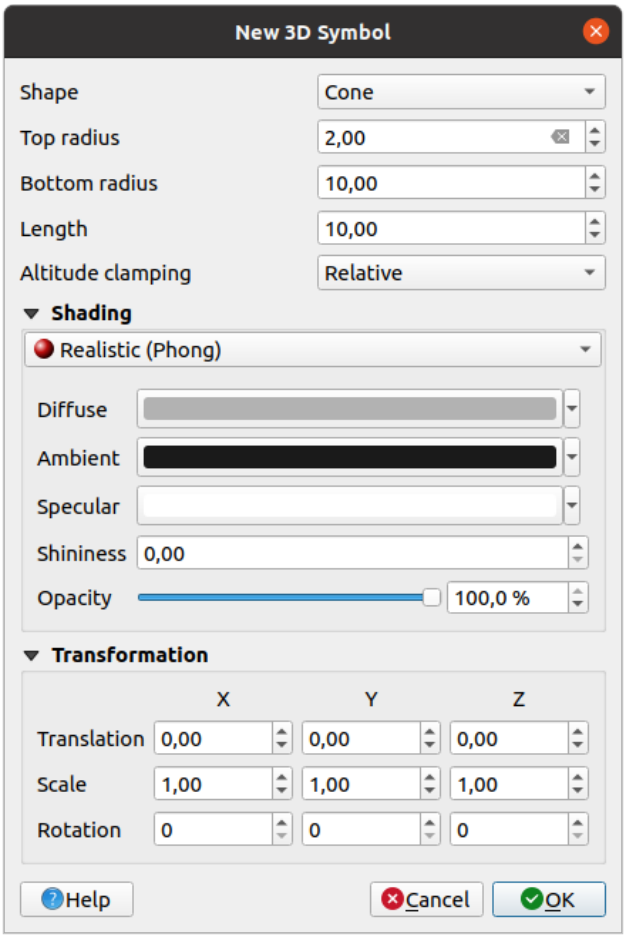


Fig. 10.36: Properties of a 3D point symbol

- You can define different types of 3D *Shape* to use for point symbols. They are mainly defined by their dimensions whose unit refers to the CRS of the project. Available types are:
  - *Sphere* defined by a *Radius*
  - *Cylinder* defined by a *Radius* and *Length*
  - *Cube* defined by a *Size*
  - *Cone* defined by a *Top radius*, a *Bottom radius* and a *Length*
  - *Plane* defined by a *Size*
  - *Torus* defined by a *Radius* and a *Minor radius*
  - *3D Model*, using a 3D model file: supported formats include wavefront `.obj`, `.glTF` and `.fbx`. Models can be a file on disk, a remote URL or *embedded in the project*. Community-created models are shared on the QGIS Hub.
  - *Billboard*, defined by the *Billboard height* and the *Billboard symbol* (usually based on a *marker symbol*). The symbol will have a stable size. Convenient for visualizing 3D point clouds Shapes.
- The *Altitude clamping* can be set to *Absolute*, *Relative* or *Terrain*. The *Absolute* setting can be used when height values of the 3d vectors are provided as absolute measures from 0. *Relative* and *Terrain* add given elevation values to the underlying terrain elevation.
- The *shading* properties can be defined.

- Under the *Transformations* frame, you can apply affine transformation to the symbol:
  - *Translation* to move objects in x, y and z axis.
  - *Scale* to resize the 3D shapes
  - *Rotation* around the x-, y- and z-axis.

10.4.2 Line layers

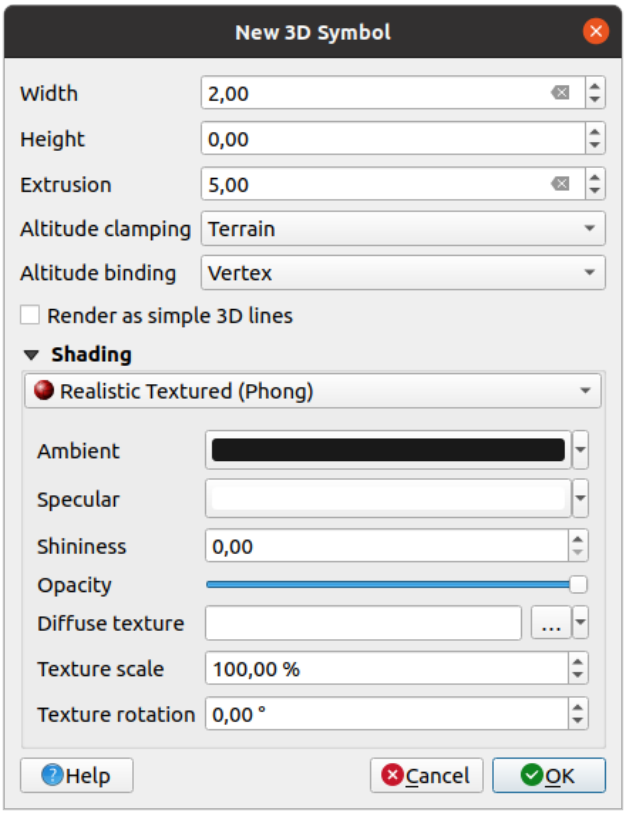


Fig. 10.37: Properties of a 3D line symbol

- Beneath the *Width* and *Height* settings you can define the *Extrusion* of the vector lines. If the lines do not have z-values, you can define the 3d volumes with this setting.
- With the *Altitude clamping* you define the position of the 3D lines relative to the underlying terrain surface, if you have included raster elevation data or other 3D vectors.
- The *Altitude binding* defines how the feature is clamped to the terrain. Either every *Vertex* of the feature will be clamped to the terrain or this will be done by the *Centroid*.
- It is possible to *Render as simple 3D lines*.
- The *shading* properties can be defined.

10.4.3 Polygon Layers

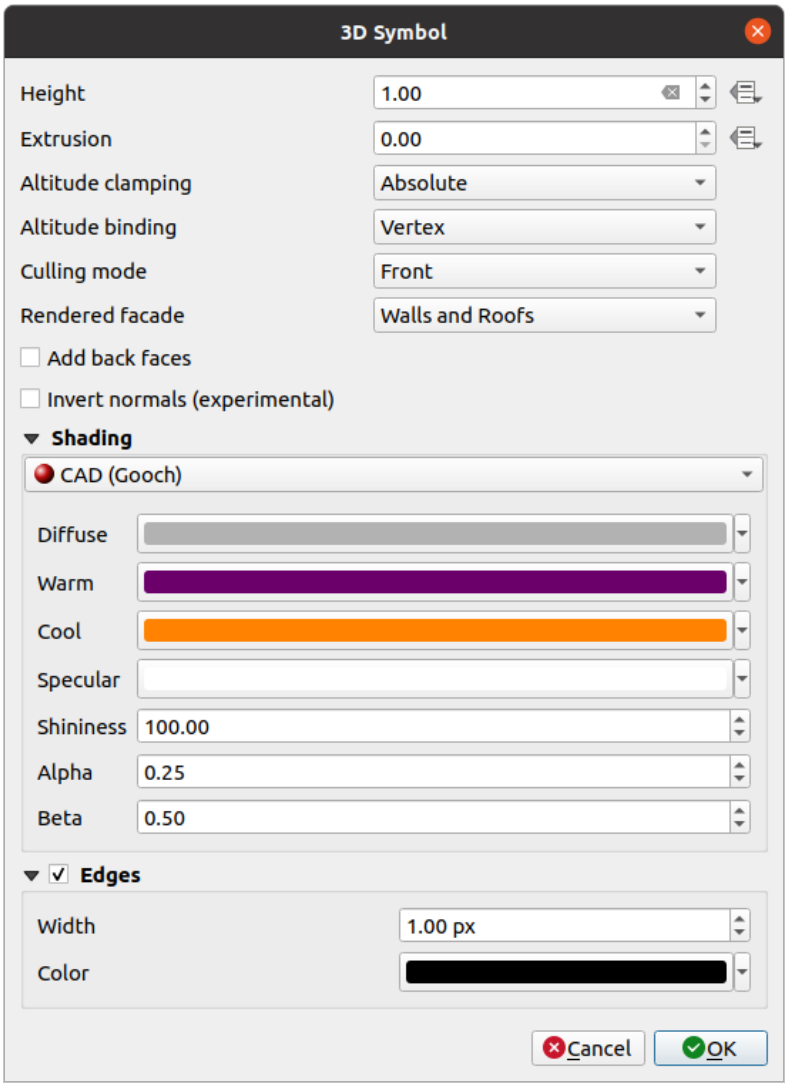


Fig. 10.38: Properties of a 3D polygon symbol

- As for the other ones, *Height* can be defined in CRS units. You can also use the button to overwrite the value with a custom expression, a variable or an entry of the attribute table
- Again, *Extrusion* is possible for missing z-values. Also for the extrusion you can use the button in order to use the values of the vector layer and have different results for each polygon:

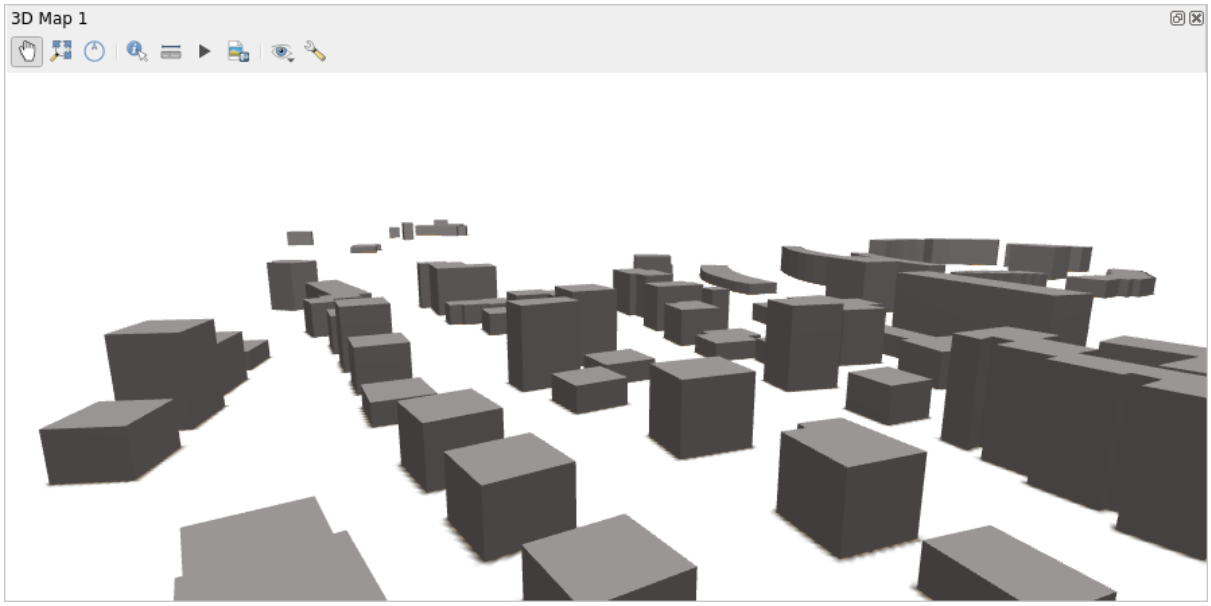


Fig. 10.39: Data Defined Extrusion

- The *Altitude clamping*, *Altitude binding* can be defined as explained above.
- The *Culling mode* to apply to the symbol; it can be:
  - *No Culling*: this can help to avoid seemingly missing surfaces when polygonZ/multipatch data do not have consistent ordering of vertices (e.g. all clock-wise or counter clock-wise)
  - *Front*
  - or *Back*
- The *Rendered facade* determines the faces to display. Possible values are *No facades*, *Walls*, *Roofs*, or *Walls and roofs*
- ☑ *Add back faces*: for each triangle, creates both front and back face with correct normals - at the expense of increased number of vertex data. This option can be used to fix shading issues (e.g., due to data with inconsistent order of vertices).
- ☑ *Invert normals (experimental)*: can be useful for fixing clockwise/counter-clockwise face vertex orders
- The *shading* properties can be defined.
- Display of the ☑ *Edges* of the symbols can be enabled and assigned a *Width* and *Color*.

Hint

Combination for best rendering of 3D data

Culling mode, *Add back faces* and *Invert normals* are all meant to fix the look of 3D data if it does not look right. Typically when loading some data, it is best to first try `culling mode=back` and `add back faces=disabled` - it is the most efficient. If the rendering does not look correct, try `add back faces=enabled` and keep `culling mode=no culling`. Other combinations are more advanced and useful only in some scenarios based on how mixed up is the input dataset.

10.4.4 Shading the texture

Shading helps you reveal 3d details of objects which may otherwise be hidden due to the scene's lighting. Ultimately, it's an easier material to work with as you don't need to worry about setting up appropriate scene lighting in order to visualise features.

Various techniques of shading are used in QGIS and their availability depends on the geometry type of the symbol:

- *Realistic (Phong)*: describes the way a surface reflects light as a combination of the *Diffuse* reflection of rough surfaces with the *Specular* reflection of shiny surfaces (*Shininess*). It also includes an *Ambient* option to account for the small amount of light that is scattered about the entire scene. Use the *Opacity* slider to render semi-transparent objects in 3D. Read more at Phong reflection description.
- *Realistic Textured (Phong)*: same as the *Realistic (Phong)* except that an image is used as *Diffuse Texture*. The image can be a file on disk, a remote URL or *embedded in the project*. The *Texture scale* and *Texture rotation* are required. Use the *Opacity* slider to render semi-transparent objects in 3D.
- *CAD (Gooch)*: this technique allows shading to occur only in mid-tones so that edge lines and highlights remain visually prominent. Along with the *Diffuse*, *Specular*, *Shininess* options, you need to provide a *Warm* color (for surface facing toward the light) and a *Cool* color (for the ones facing away). Also, the relative contributions to the cool and warm colors by the diffuse color are controlled by *Alpha* and *Beta* properties respectively. See also Gooch shading.
- *Metal Roughness*: a physically based rendering material that provides an accurate representation of how light interacts with surfaces. Options are available for setting the material *Base color*, *Metalness* and *Roughness*.
- *Embedded Textures* with 3D models shape

10.4.5 Application example

To go through the settings explained above you can have a look at https://app.merginmaps.com/projects/saber/luxembourg/tree.

CHAPTER

ELEVEN

MANAGING DATA SOURCE

11.1 Opening Data

As part of an Open Source Software ecosystem, QGIS is built upon different libraries that, combined with its own providers, offer capabilities to read and often write a lot of formats:

- Vector data formats include GeoPackage, GML, GeoJSON, GPX, KML, Comma Separated Values, ESRI formats (Shapefile, Geodatabase…), MapInfo and MicroStation file formats, AutoCAD DWG/DXF, GRASS and many more… Read the complete list of supported vector formats.
- Raster data formats include GeoTIFF, JPEG, ASCII Gridded XYZ, MBTiles, R or Idrisi rasters, GDAL Virtual, SRTM, Sentinel Data, ERDAS IMAGINE, ArcInfo Binary Grid, ArcInfo ASCII Grid, and many more… Read the complete list of supported raster formats.
- Database formats include PostgreSQL, SQLite/SpatiaLite, Oracle, MS SQL Server, SAP HANA, MySQL…
- Web map and data services (WM(T)S, WFS, WCS, CSW, XYZ tiles, ArcGIS services, …) are also handled by QGIS providers. See *Working with OGC / ISO protocols* for more information about some of these.
- You can read supported files from archived folders and use QGIS native formats such as QML files (*QML - The QGIS Style File Format*) and virtual and memory layers.

More than 80 vector and 140 raster formats are supported by GDAL and QGIS native providers.

> **Note**
>
> Not all of the listed formats may work in QGIS for various reasons. For example, some require external proprietary libraries, or the GDAL/OGR installation of your OS may not have been built to support the format you want to use. To see the list of available formats, run the command line `ogrinfo --formats` (for vector) and `gdalinfo --formats` (for raster), or check the *Settings* ► *Options* ► *GDAL* menu in QGIS.

In QGIS, depending on the data format, there are different tools to open a dataset, mainly available in the *Layer* ► *Add Layer* ► menu or from the *Manage Layers* toolbar (enabled through *View* ► *Toolbars* menu). However, all these tools point to a unique dialog, the *Data Source Manager* dialog, that you can open with the  Open Data Source Manager button, available on the *Data Source Manager Toolbar*, or by pressing `Ctrl+L`. The *Data Source Manager* dialog (Fig. 11.1) offers a unified interface to open file-based data as well as databases or web services supported by QGIS.

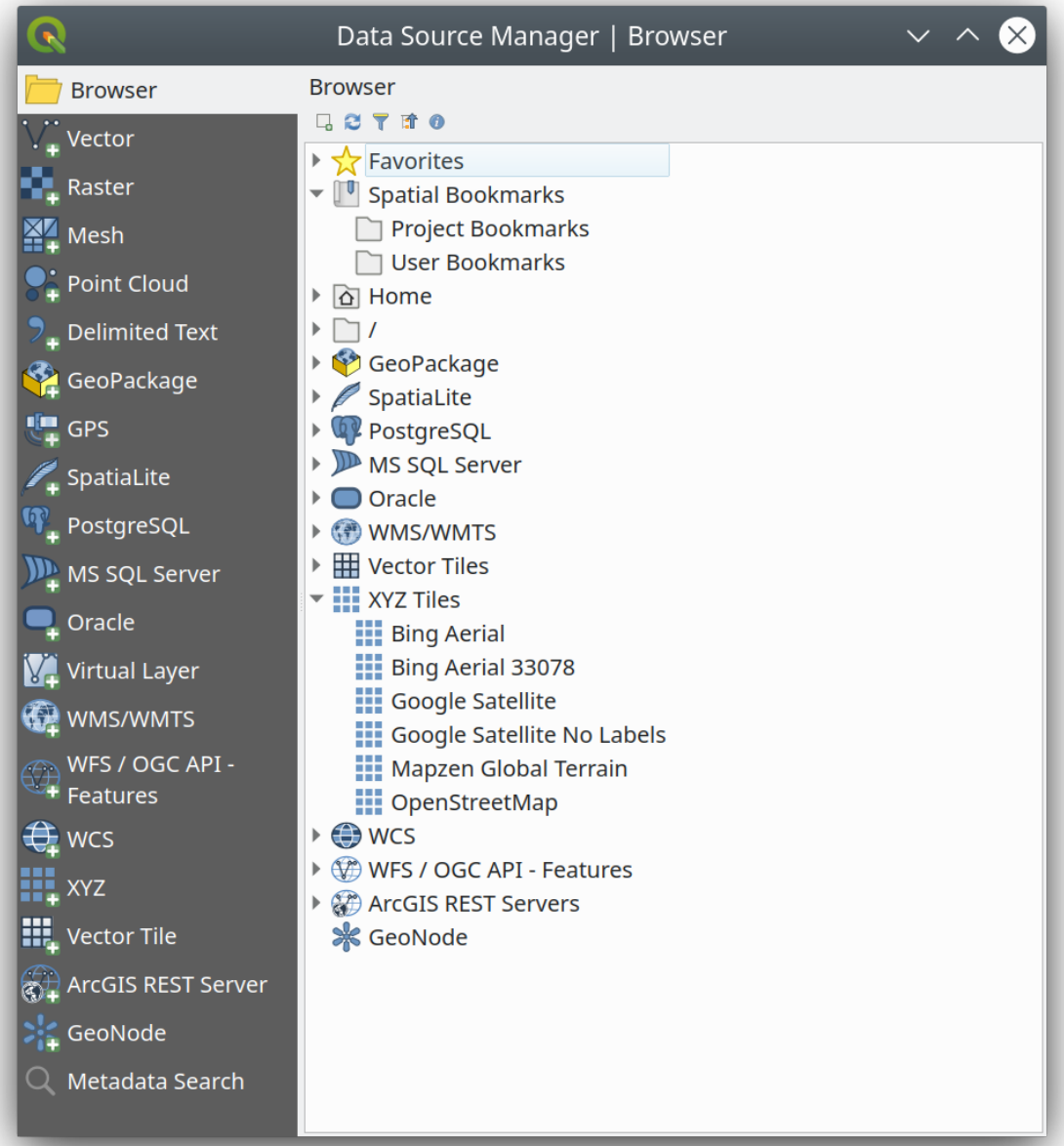


Fig. 11.1: QGIS Data Source Manager dialog

Beside this main entry point, you also have the *DB Manager* plugin that offers advanced capabilities to analyze and manipulate connected databases. More information on DB Manager capabilities can be found in *DB Manager Plugin*.

There are many other tools, native or third-party plugins, that help you open various data formats.

This chapter will describe only the tools provided by default in QGIS for loading data. It will mainly focus on the *Data Source Manager* dialog but more than describing each tab, it will also explore the tools based on the data provider or format specificities.

11.1.1 The Browser Panel

The *Browser* is one of the main ways to quickly and easily add your data to projects. It's available as:

- a *Data Source Manager* tab, enabled pressing the Open Data Source Manager button (`Ctrl+L`);
- as a QGIS panel you can open from the menu *View* ► *Panels* (or *Settings* ► *Panels*) or by pressing `Ctrl+2`.

In both cases, the *Browser* helps you navigate in your file system and manage geodata, regardless the type of layer (raster, vector, table), or the datasource format (plain or compressed files, databases, web services).

Exploring the Interface

At the top of the Browser panel, you find some buttons that help you to:

- Add Selected Layers: you can also add data to the map canvas by selecting **Add selected layer(s)** from the layer's context menu;
- Refresh the browser tree;
- Filter Browser to search for specific data. Enter a search word or wildcard and the browser will filter the tree to only show paths to matching DB tables, filenames or folders – other data or folders won't be displayed. See the Browser Panel(2) example in Fig. 11.2. The comparison can be case-sensitive or not. It can also be set to:
  - *Normal*: show items containing the search text
  - *Wildcard(s)*: fine tune the search using the `?` and/or `*` characters to specify the position of the search text
  - *Regular expression*
- Collapse All the whole tree;
- Enable/disable properties widget: when toggled on, a new widget is added at the bottom of the panel showing, if applicable, metadata for the selected item.

The entries in the *Browser* panel are organised hierarchically, and there are several top level entries:

1. *Favorites* where you can place shortcuts to often used locations
2. *Spatial Bookmarks* where you can store often used map extents (see *Bookmarking extents on the map*)
3. *Project Home*: for a quick access to the folder in which (most of) the data related to your project are stored. The default value is the directory where your project file resides.
4. *Home* directory in the file system and the filesystem root directory.
5. Connected local or network drives
6. Then comes a number of container / database types and service protocols, depending on your platform and underlying libraries:
   - *GeoPackage*
   - *SpatiaLite*
   - *PostgreSQL*
   - *SAP HANA*
   - *MS SQL Server*
   - *Oracle*

-  *WMS/WMTS*
-  *Vector Tiles*
-  *XYZ Tiles*
-  *WCS*
-  *WFS/OGC API-Features*
-  *ArcGIS REST Server*

Interacting with the Browser items

The browser supports drag and drop within the browser, from the browser to the canvas and *Layers* panel, and from the *Layers* panel to layer containers (e.g. GeoPackage) in the browser.

Project file items inside the browser can be expanded, showing the full layer tree (including groups) contained within that project. Project items are treated the same way as any other item in the browser, so they can be dragged and dropped within the browser (for example to copy a layer item to a geopackage file) or added to the current project through drag and drop or double click.

The context menu for an element in the *Browser* panel is opened by right-clicking on it.

For file system directory entries, the context menu offers the following:

- *New* ► to create in the selected entry a:
  - *Directory…*
  - *GeoPackage…*
  - *ShapeFile…*
- *Add as a Favorite*: favorite folders can be renamed (*Rename favorite…*) or removed (*Remove favorite*) any time.
- *Hide from Browser*: hidden folders can be toggled to visible from the *Settings* ► *Options* ► *Data Sources* ► *Hidden browser paths* setting
- *Fast Scan this Directory*
- *Open Directory*
- *Open in Terminal*
- *Properties…*
- *Directory Properties…*

For leaf entries that can act as layers in the project, the context menu will have supporting entries. For example, for non-database, non-service-based vector, raster and mesh data sources:

- *Export Layer* ► *To File…*
- *Add Layer to Project*
- *Layer Properties*
- *Open with Data Source Manager…*
- *Manage* ► *Rename "<name of file>"…* or *Delete "<name of file>"…*
- *Show in Files*
- *File Properties*

In the *Layer properties* entry, you will find (similar to what you will find in the *vector* and *raster* layer properties once the layers have been added to the project):

- *Metadata* for the layer. Metadata groups: *Information from provider* (if possible, *Path* will be a hyperlink to the source), *Identification*, *Extent*, *Access*, *Fields* (for vector layers), *Bands* (for raster layers), *Contacts*, *Links* (for vector layers), *References* (for raster layers), *History*.
- A *Preview* panel
- The attribute table for vector sources (in the *Attributes* panel).

Use *Open with Data Source Manager…* to directly open and configure the data source in the *Data Source Manager* using the URI of your data source. This simplifies the process of adding a layer from the *Browser* by allowing you to set specific opening options for the data source. It is currently available for vector (including the dedicated GeoPackage entry), raster, and SpatiaLite data sources.

To add a layer to the project using the *Browser*:

1. Enable the *Browser* as described above. A browser tree with your file system, databases and web services is displayed. You may need to connect databases and web services before they appear (see dedicated sections).
2. Find the layer in the list.
3. Use the context menu, double-click its name, or drag-and-drop it into the *map canvas*. Your layer is now added to the *Layers panel* and can be viewed on the map canvas.

> **Tip**
>
> **Open a QGIS project directly from the browser**
>
> You can also open a QGIS project directly from the Browser panel by double-clicking its name or by drag-and-drop into the map canvas.

Once a file is loaded, you can zoom around it using the map navigation tools. To change the style of a layer, open the *Layer Properties* dialog by double-clicking on the layer name or by right-clicking on the name in the legend and choosing *Properties* from the context menu. See section *Symbology Properties* for more information on setting symbology for vector layers.

Right-clicking an item in the browser tree helps you to:

- for a file or a table, display its metadata or open it in your project. Tables can even be renamed, deleted or truncated.
- for a folder, bookmark it into your favourites or hide it from the browser tree. Hidden folders can be managed from the *Settings* ► *Options* ► *Data Sources* tab.
- manage your *spatial bookmarks*: bookmarks can be created, exported and imported as `XML` files.
- create a connection to a database or a web service.
- refresh, rename or delete a schema.

You can also import files into databases or copy tables from one schema/database to another with a simple drag-and-drop. There is a second browser panel available to avoid long scrolling while dragging. Just select the file and drag-and-drop from one panel to the other.

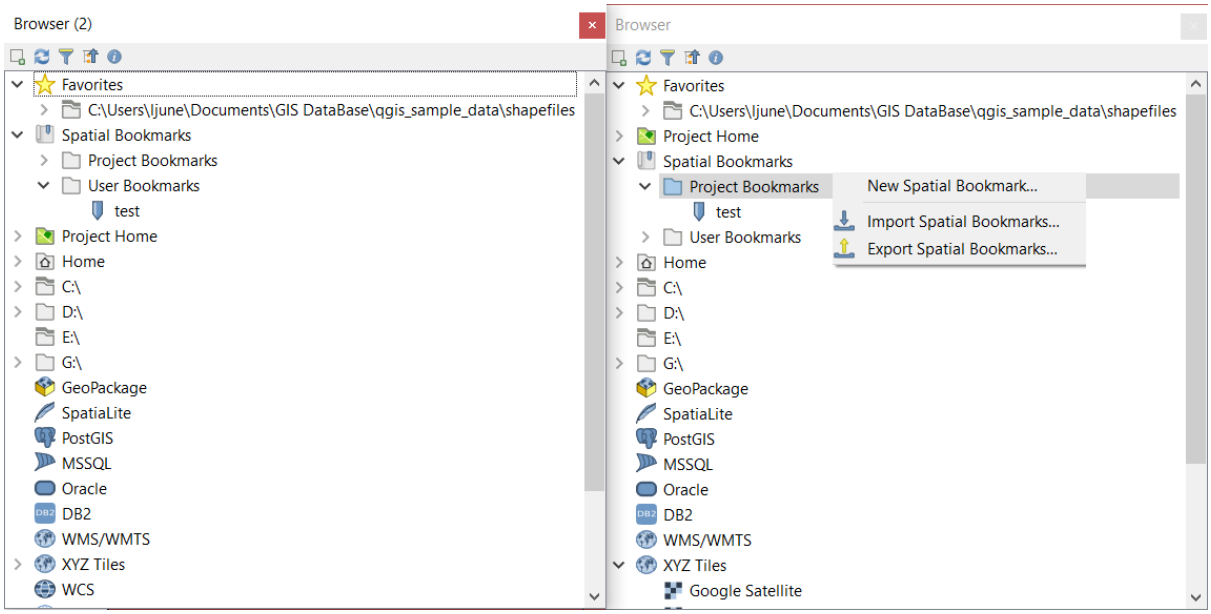


Fig. 11.2: QGIS Browser panels side-by-side

Tip

Add layers to QGIS by simple drag-and-drop from your OS file browser

You can also add file(s) to the project by drag-and-dropping them from your operating system file browser to the *Layers Panel* or the map canvas.

11.1.2 The DB Manager

The *DB Manager* Plugin is another tool for integrating and managing spatial database formats supported by QGIS (PostgreSQL, SpatiaLite, GeoPackage, Oracle Spatial, MS SQL Server, Virtual layers). It can be activated from the *Plugins* ► *Manage and Install Plugins...* menu.

The DB Manager Plugin provides several features:

- connect to databases and display their structure and contents
- preview tables of databases
- add layers to the map canvas, either by double-clicking or drag-and-drop.
- add layers to a database from the QGIS Browser or from another database
- create SQL queries and add their output to the map canvas
- create *virtual layers*

More information on DB Manager capabilities is found in *DB Manager Plugin*.

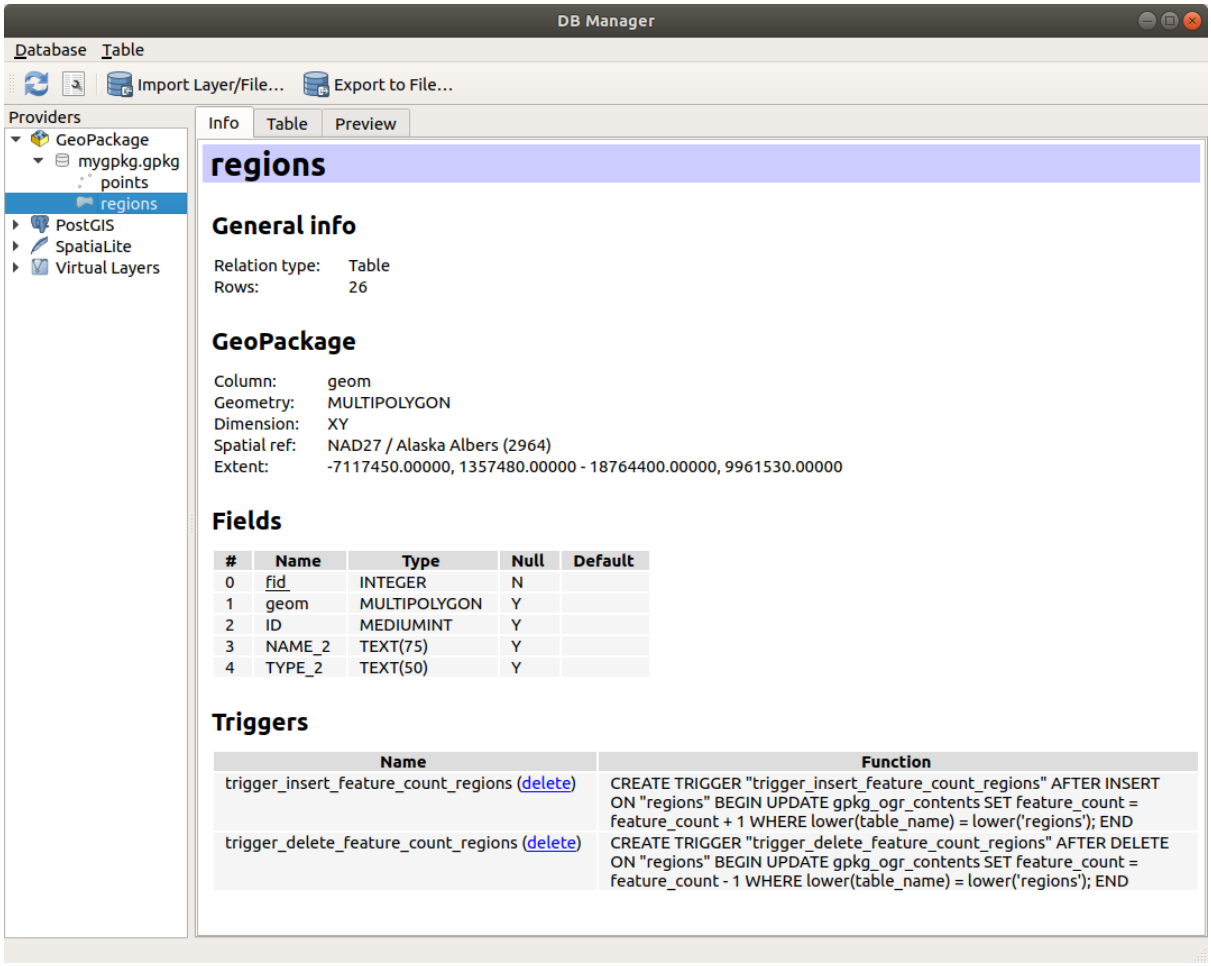


Fig. 11.3: DB Manager dialog

11.1.3 Provider-based loading tools

Beside the Browser Panel and the DB Manager, the main tools provided by QGIS to add layers, you'll also find tools that are specific to data providers.

> **Note**
>
> Some *external plugins* also provide tools to open specific format files in QGIS.

Loading a layer from a file

To load a layer from a file:

1. Open the layer type tab in the *Data Source Manager* dialog, ie click the Open Data Source Manager button (or press `Ctrl+L`) and enable the target tab or:
   - for vector data (like GML, ESRI Shapefile, Mapinfo and DXF layers): press `Ctrl+Shift+V`, select the *Layer* ► *Add Layer* ► *Add Vector Layer* menu option or click on the Add Vector Layer toolbar button.

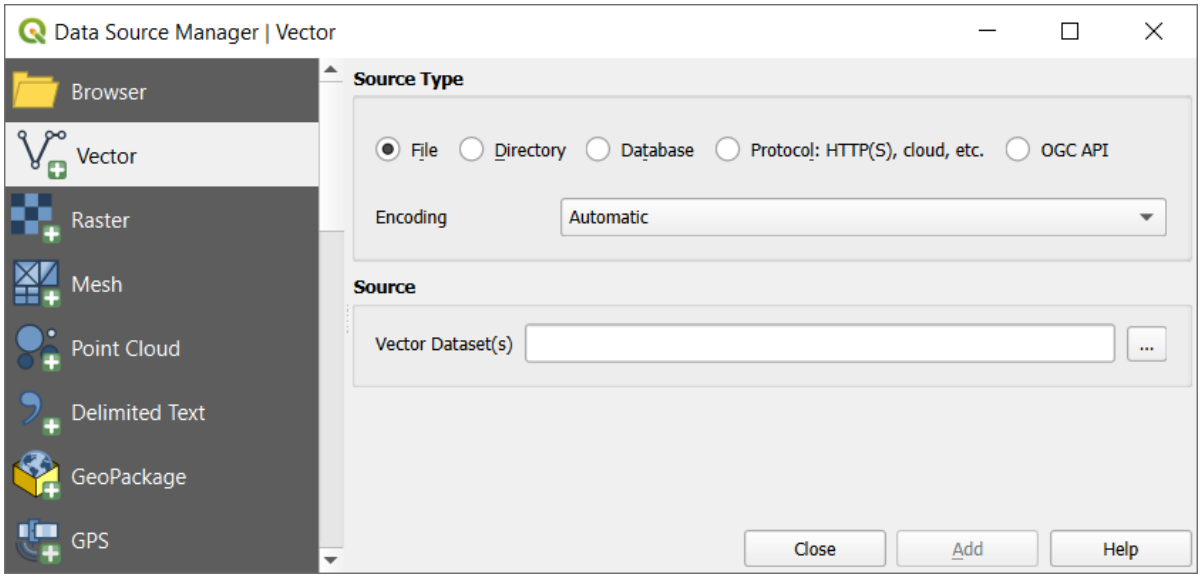


Fig. 11.4: Add Vector Layer Dialog

- for raster data (like GeoTiff, MBTiles, GRIdded Binary and DWG layers): press `Ctrl+Shift+R`, select the *Layer* ► *Add Layer* ► *Add Raster Layer* menu option or click on the Add Raster Layer toolbar button.

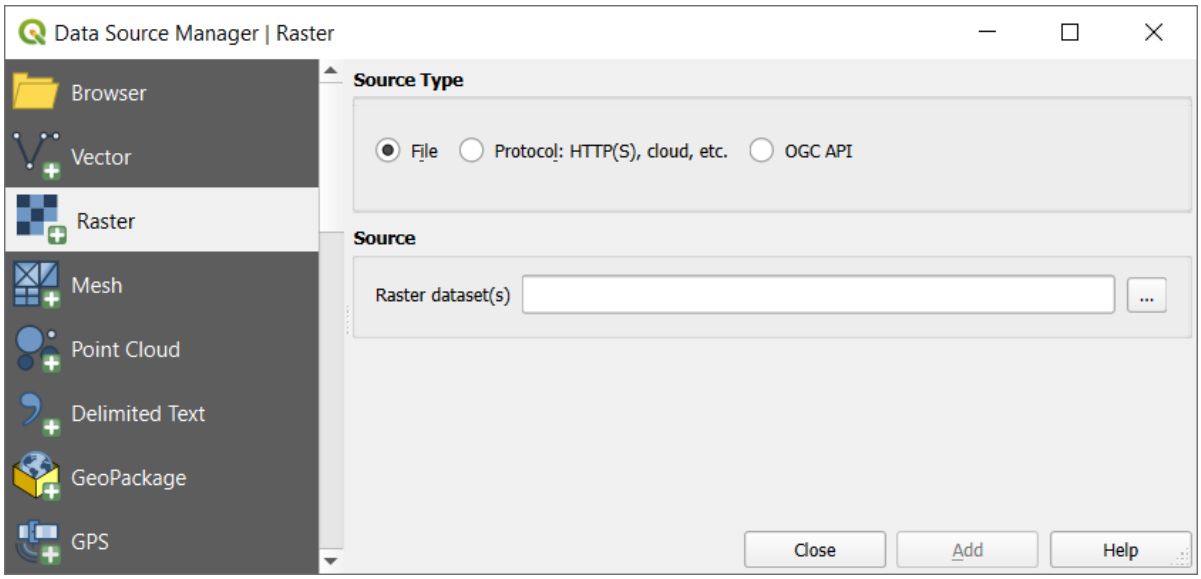


Fig. 11.5: Add Raster Layer Dialog

2. Check *File* source type
3. Click on the … Browse button
4. Navigate the file system and load a supported data source. More than one layer can be loaded at the same time by holding down the `Ctrl` key and clicking on multiple items in the dialog or holding down the `Shift` key to select a range of items by clicking on the first and last items in the range. Only formats that have been well tested appear in the formats filter. Other formats can be loaded by selecting `All files` (the top item in the pull-down menu).
5. Press *Open* to load the selected file into *Data Source Manager* dialog.

Depending on the selected layer type, additional *Options* (encoding, geometry type, table filtering, file locking, data formatting …) are available for configuring. These options are described in detail in the specific GDAL vector or raster driver documentation. At the top of the options, a text with hyperlink will directly lead to the documentation of the appopriate driver for the selected file format.

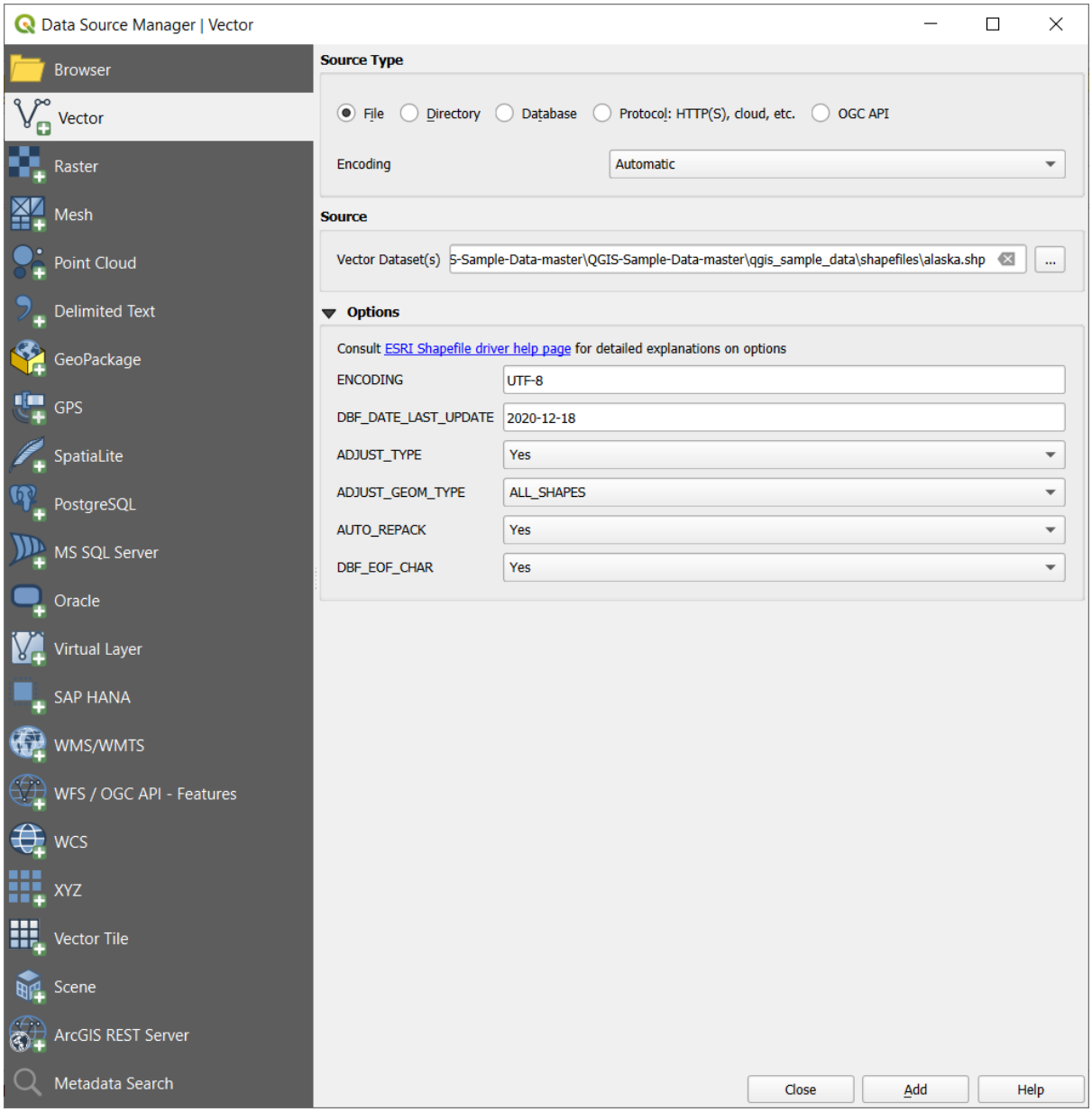


Fig. 11.6: Loading a Shapefile with open options

6. Press *Add* to load the file in QGIS and display them in the map view. When adding vector datasets containing multiple layers, the *Select Items to Add* dialog will appear. In this dialog, you can choose the specific layers from your dataset that you want to add. Also, under *Options* you can choose to:

   - *Add layers to a group*
   - *Show system and internal tables*
   - *Show empty vector layers.*

   Fig. 11.7 shows QGIS after loading the `alaska.shp` file.

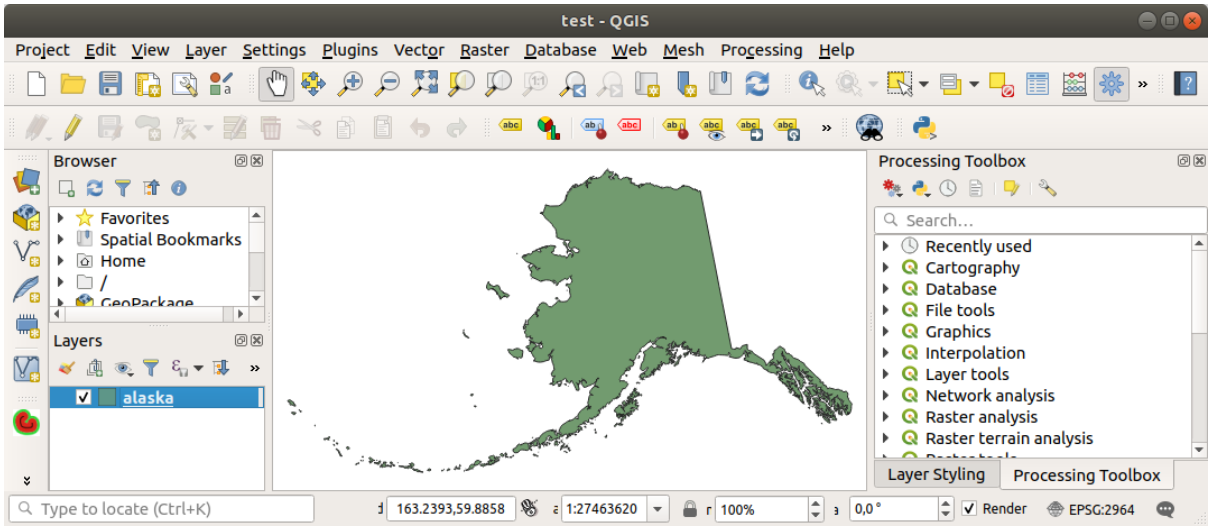


Fig. 11.7: QGIS with Shapefile of Alaska loaded

Note

Because some formats like MapInfo (e.g., `.tab`) or Autocad (`.dxf`) allow mixing different types of geometry in a single file, loading such datasets opens a dialog to select geometries to use in order to have one geometry per layer.

The Add Vector Layer and Add Raster Layer tabs allow loading of layers from source types other than *File*:

- You can load specific vector formats like `ArcInfo Binary Coverage`, `UK. National Transfer Format`, as well as the raw TIGER format of the `US Census Bureau` or `OpenfileGDB`. To do that, you select *Directory* as *Source type*. In this case, a directory can be selected in the dialog after pressing … Browse.

- With the *Database* source type you can select an existing database connection or create one to the selected database type. Some possible database types are `ODBC`, `Esri Personal Geodatabase`, `MS SQL Server` as well as `PostgreSQL` or `MySQL`.

  Pressing the *New* button opens the *Create a New OGR Database Connection* dialog whose parameters are among the ones you can find in *Creating a stored Connection*. Pressing *Open* lets you select from the available tables.

- The *Protocol: HTTP(S), cloud, etc.* source type opens data stored locally or on the network, either publicly accessible, or in private buckets of commercial cloud storage services. Supported protocol types are:

  - `HTTP/HTTPS/FTP`, with a *URI* and, if required, an *authentication*.
  - Cloud storage such as `AWS S3`, `Google Cloud Storage`, `Microsoft Azure Blob`, `Microsoft Azure Data Lake Storage`, `Alibaba OSS Cloud`, and `Open Stack Swift Storage` supports direct control over VSI *Credential Options* when adding OGR vector or GDAL raster layers. You need to fill in the *Bucket or container* and the *Object key* first. After that, you can add the necessary *Credential Options*.

    When adding OGR vector or GDAL raster layers from the cloud based protocols, you can also set additional *Credential options* for that specific driver and bucket. When credential options are found in a layer's URI, they will also be automatically set. This allows different layers to use different credentials.
  - service supporting OGC `WFS 3` (still experimental), using `GeoJSON` or `GEOJSON - Newline Delimited` format or based on `CouchDB` database. A *URI* is required, with optional *authentication*.
  - For all vector source types it is possible to define the *Encoding* or to use the *Automatic* ► setting.

- The *OGC API* source type allows you to access vector and raster data from servers that implement the OGC API standards. To use this option:

  1. Select *OGC API* from the *Data Source Manager* dialog.
  2. Enter the endpoint of the OGC API service you want to connect to. Note that you don't need to prefix the endpoint with "OGCAPI:".
  3. Click *Connect* to establish a connection to the server.

Loading a mesh layer

A mesh is an unstructured grid usually with temporal and other components. The spatial component contains a collection of vertices, edges and faces in 2D or 3D space. More information on mesh layers at *Working with Mesh Data*.

To add a mesh layer to QGIS:

1. Open the *Data Source Manager* dialog, either by selecting it from the *Layer* ► menu or clicking the Open Data Source Manager button.
2. Enable the *Mesh* tab on the left panel
3. Press the … Browse button to select the file. *Various formats* are supported.
4. Select the file and press *Add*. The layer will be added using the native mesh rendering.
5. If the selected file contains many mesh layers, then you'll be prompted with a dialog to choose the sublayers to load. Do your selection and press *OK* and the layers are loaded with the native mesh rendering. It's also possible to load them within a group.

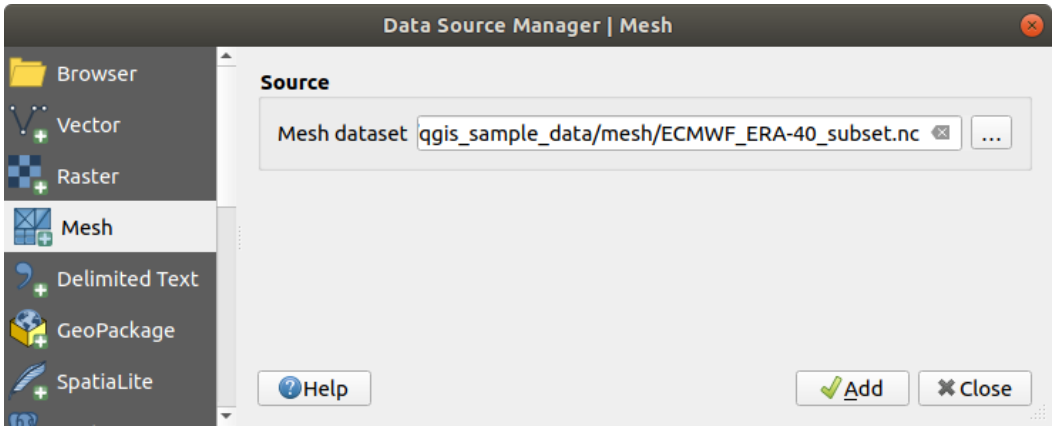


Fig. 11.8: Mesh tab in Data Source Manager

Importing a delimited text file

Delimited text files (e.g. `.txt`, `.csv`, `.dat`, `.wkt`) can be loaded using the tools described above. This way, they will show up as simple tables. Sometimes, delimited text files can contain coordinates / geometries that you could want to visualize. This is what *Add Delimited Text Layer* is designed for.

1. Click the Open Data Source Manager icon to open the *Data Source Manager* dialog
2. Enable the *Delimited Text* tab

3. Select the delimited text file to import (e.g., `qgis_sample_data/csv/elevp.csv`) by clicking on the … Browse button.
4. Configure the settings to meet your dataset and needs, as explained below.

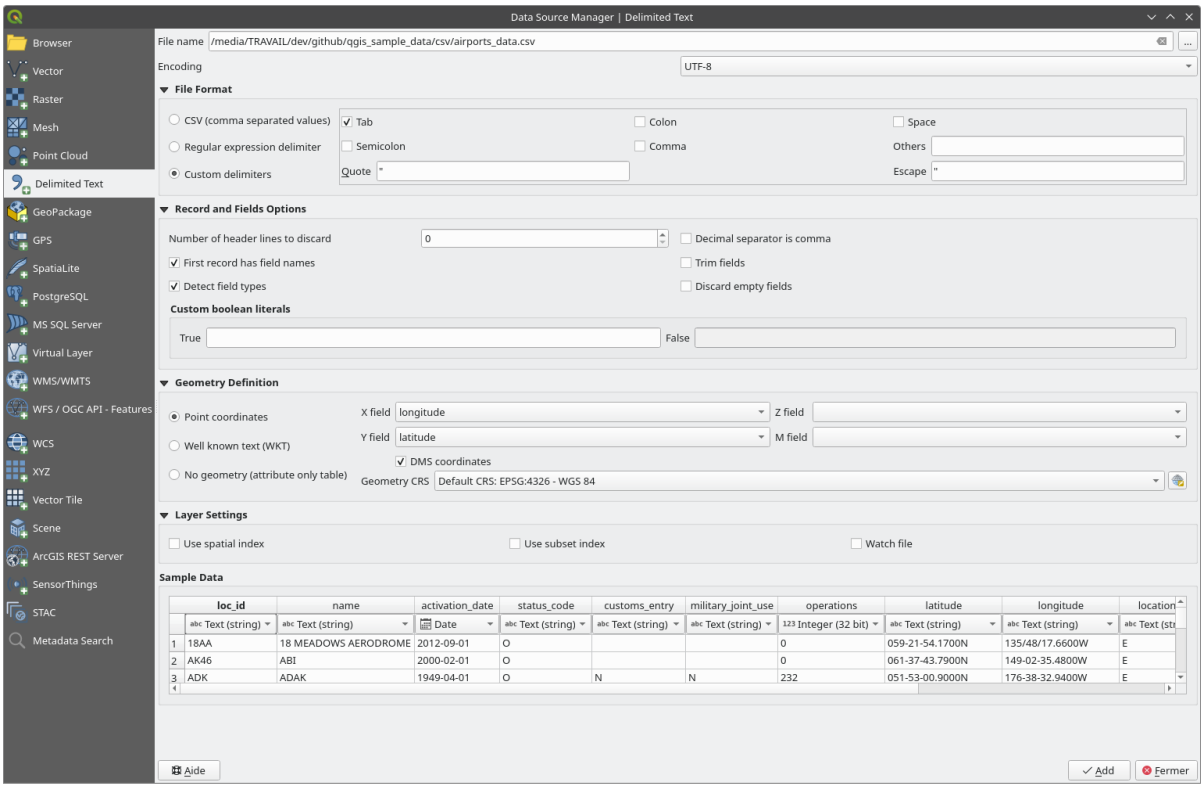


Fig. 11.9: Delimited Text Dialog

File format

Once the file is selected, QGIS attempts to parse the file with the most recently used delimiter, identifying fields and rows. To enable QGIS to correctly parse the file, it is important to select the right delimiter. You can specify a delimiter by choosing between:

- *CSV (comma separated values)* to use the comma character.
- *Regular expression delimiter* and enter text into the *Expression* field. For example, to change the delimiter to tab, use `\t` (this is used in regular expressions for the tab character).
- *Custom delimiters*, choosing among some predefined delimiters like `comma`, `space`, `tab`, `semicolon`, … .

Records and fields

Some other convenient options can be used for data recognition:

- *Number of header lines to discard*: convenient when you want to avoid the first lines in the file in the import, either because those are blank lines or with another formatting.
- *First record has field names*: values in the first line are used as field names, otherwise QGIS uses the field names `field_1`, `field_2`...
- *Detect field types*: automatically recognizes the field type. If unchecked then all attributes are treated as text fields.
- *Decimal separator is comma*: you can force decimal separator to be a comma.
- *Trim fields*: allows you to trim leading and trailing spaces from fields.
- *Discard empty fields*.
- *Custom boolean literals*: allows you to add a custom couple of string that will be detected as boolean values.

Field type detection

QGIS tries to detect the field types automatically (unless *Detect field types* is not checked) by examining the content of an optional sidecar CSVT file (see GeoCSV specification) and by scanning the whole file to make sure that all values can actually be converted without errors, the fall-back field type is text.

The detected field type appears under the field name in sample data preview table and can be manually changed if necessary.

The following field types are supported:

- `Boolean` case-insensitive literal couples that are interpreted as boolean values are `1`/`0`, `true`/`false`, `t`/`f`, `yes`/`no`
- `Whole Number (integer)`
- `Whole Number (integer - 64 bit)`
- `Decimal Number`: double precision floating point number
- `Date`
- `Time`
- `Date and Time`
- `Text`

Geometry definition

Once the file is parsed, set *Geometry definition* to

- *Point coordinates* and provide the *X field*, *Y field*, *Z field* (for 3-dimensional data) and *M field* (for the measurement dimension) if the layer is of point geometry type and contains such fields. If the coordinates are defined as degrees/minutes/seconds, activate the *DMS coordinates* checkbox. Provide the appropriate *Geometry CRS* using the Select CRS widget.
- *Well known text (WKT)* option if the spatial information is represented as WKT: select the *Geometry field* containing the WKT geometry and choose the approriate *Geometry field* or let QGIS auto-detect it. Provide the appropriate *Geometry CRS* using the Select CRS widget.

- If the file contains non-spatial data, activate *No geometry (attribute only table)* and it will be loaded as an ordinary table.

Layer settings

Additionally, you can enable:

- *Use spatial index* to improve the performance of displaying and spatially selecting features.
- *Use subset index* to improve performance of *subset filters* (when defined in the layer properties).
- *Watch file* to watch for changes to the file by other applications while QGIS is running.

At the end, click *Add* to add the layer to the map. In our example, a point layer named `Elevation` is added to the project and behaves like any other map layer in QGIS. This layer is the result of a query on the `.csv` source file (hence, linked to it) and would require *to be saved* in order to get a spatial layer on disk.

Sample Data

As you set the parser properties, the sample data preview updates regarding to the applied settings.

Also in the Sample Data Table it is possible to override the automatically determined column types.

Importing a DXF or DWG file

`DXF` and `DWG` files can be added to QGIS by simple drag-and-drop from the Browser Panel. You will be prompted to select the sublayers you would like to add to the project. Layers are added with random style properties.

> **Note**
>
> For DXF files containing several geometry types (point, line and/or polygon), the name of the layers will be generated as *<filename.dxf> entities <geometry type>*.

To keep the dxf/dwg file structure and its symbology in QGIS, you may want to use the dedicated *Project ► Import/Export ► Import Layers from DWG/DXF…* tool which allows you to:

1. import elements from the drawing file into a GeoPackage database.
2. add imported elements to the project.

In the *DWG/DXF Import* dialog, to import the drawing file contents:

1. Input the location of the *Source drawing*, i.e. the DWG/DXF drawing file to import.
2. Specify the coordinate reference system of the data in the drawing file.
3. Input the location of the *Target package*, i.e. the GeoPackage file that will store the data. If an existing file is provided, then it will be overwritten.
4. Choose how to import `blocks` with the dedicated combobox:
   - *Expand Block Geometries*: imports the blocks in the drawing file as normal elements.
   - *Expand Block Geometries and Add Insert Points*: imports the blocks in the drawing file as normal elements and adds the insertion point as a point layer.
   - *Add Only Insert Points*: adds the blocks insertion point as a point layer.
5. Check *Use curves* to promote the imported layers to a `curved` geometry type.

6. Use the *Import* button to import the drawing into the destination GeoPackage file. The GeoPackage database will be automatically populated with the drawing file content. Depending on the size of the file, this can take some time.

After the `.dwg` or `.dxf` data has been imported into the GeoPackage database, the frame in the lower half of the dialog is populated with the list of layers from the imported file. There you can select which layers to add to the QGIS project:

1. At the top, set a *Group name* to group the drawing files in the project. By default this is set to the filename of the source drawing file.
2. Check layers to show: Each selected layer is added to an ad hoc group which contains vector layers for the point, line, label and area features of the drawing layer. The style of the layers will resemble the look they originally had in *CAD.
3. Choose if the layer should be visible at opening.
4. Checking the *Merge layers* option places all layers in a single group.
5. Press *OK* to open the layers in QGIS.

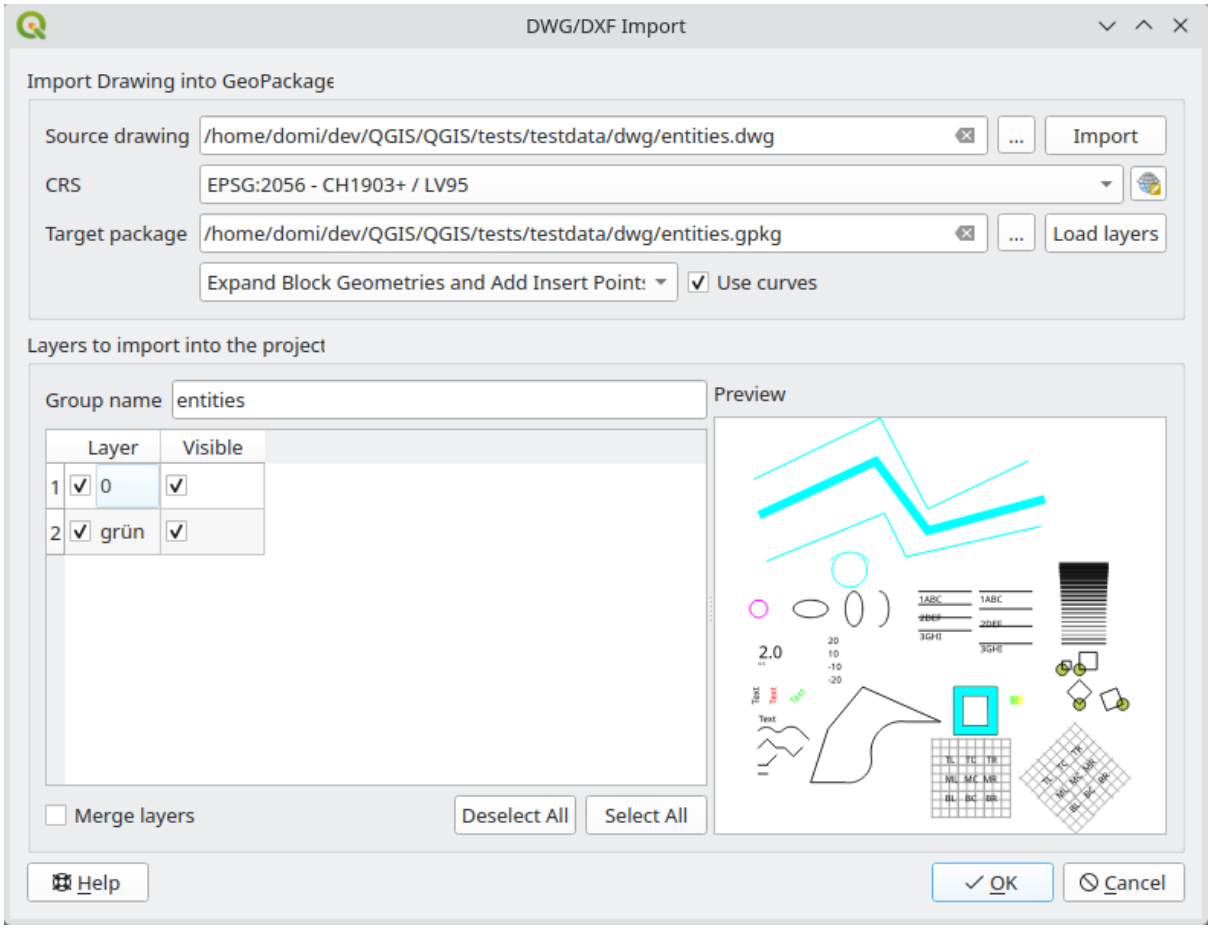


Fig. 11.10: Import dialog for DWG/DXF files

Importing OpenStreetMap Vectors

The OpenStreetMap project is popular because in many countries no free geodata such as digital road maps are available. The objective of the OSM project is to create a free editable map of the world from GPS data, aerial photography and local knowledge. To support this objective, QGIS provides support for OSM data.

Using the *Browser Panel*, you can load an `.osm` file to the map canvas, in which case you'll get a dialog to select sublayers based on the geometry type. The loaded layers will contain all the data of that geometry type in the `.osm` file, and keep the `osm` file data structure.

SpatiaLite Layers

The first time you load data from a SpatiaLite database, begin by:

- clicking on the Add SpatiaLite Layer toolbar button
- selecting the *Add SpatiaLite Layer…* option from the *Layer* ► *Add Layer* menu
- or by typing `Ctrl+Shift+L`

This will bring up a window that will allow you either to connect to a SpatiaLite database already known to QGIS (which you choose from the drop-down menu) or to define a new connection to a new database. To define a new connection, click on *New* and use the file browser to point to your SpatiaLite database, which is a file with a `.sqlite` extension.

QGIS also supports editable views in SpatiaLite.

GPS

There are dozens of different file formats for storing GPS data. The format that QGIS uses is called GPX (GPS eXchange format), which is a standard interchange format that can contain any number of waypoints, routes and tracks in the same file.

Use the … Browse button to select the GPX file, then use the check boxes to select the feature types you want to load from that GPX file. Each feature type will be loaded in a separate layer.

More on GPS data manipulation at *Working with GPS Data*.

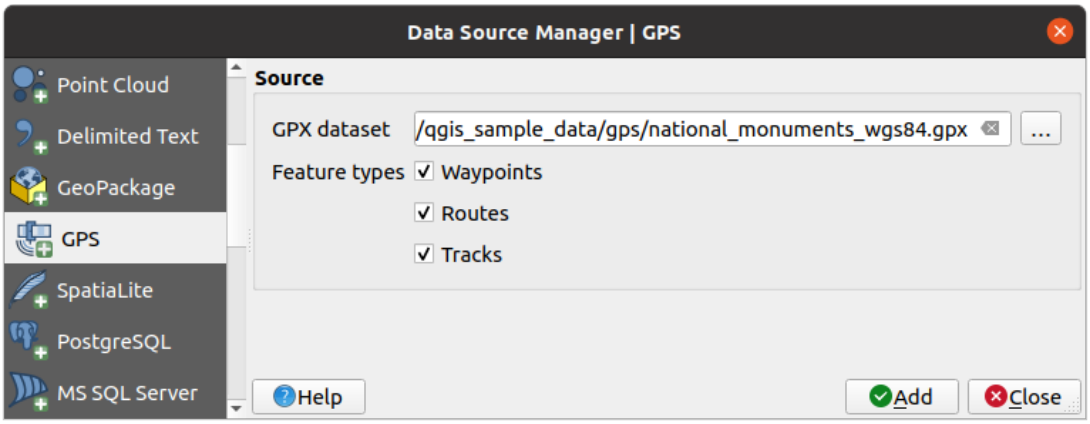


Fig. 11.11: Loading GPS Data dialog

GRASS

Working with GRASS vector data is described in section *GRASS GIS Integration*.

Database related tools

Creating a stored Connection

In order to read and write tables from a database format QGIS supports you have to create a connection to that database. While *QGIS Browser Panel* is the simplest and recommanded way to connect to and use databases, QGIS provides other tools to connect to each of them and load their tables:

- *Add PostgreSQL Layer…* or by typing `Ctrl+Shift+D`
- *Add Oracle Spatial Layer…* or by typing `Ctrl+Shift+O`
- *Add MS SQL Server Layer*
- *Add SAP HANA Spatial Layer…* or by typing `Ctrl+Shift+G`

These tools are accessible either from the *Manage Layers Toolbar* and the *Layer* ► *Add Layer* ► menu. Connecting to SpatiaLite database is described at *SpatiaLite Layers*.

> **Tip**
>
> **Create connection to database from the QGIS Browser Panel**
>
> Selecting the corresponding database format in the Browser tree, right-clicking and choosing connect will provide you with the database connection dialog.

Most of the connection dialogs follow a common structure:

- a section with credentials information to connect to the database
- a section with options to tune which data can be requested in the database

Connecting to PostgreSQL

The first time you use a PostgreSQL data source, you must create a connection to a database that contains the data. Press the appropriate button as exposed above, opening the *PostgreSQL* tab of the *Data Source Manager* dialog. To access the connection manager, click on the *New* button to display the *Create a New PostgreSQL Connection* dialog.

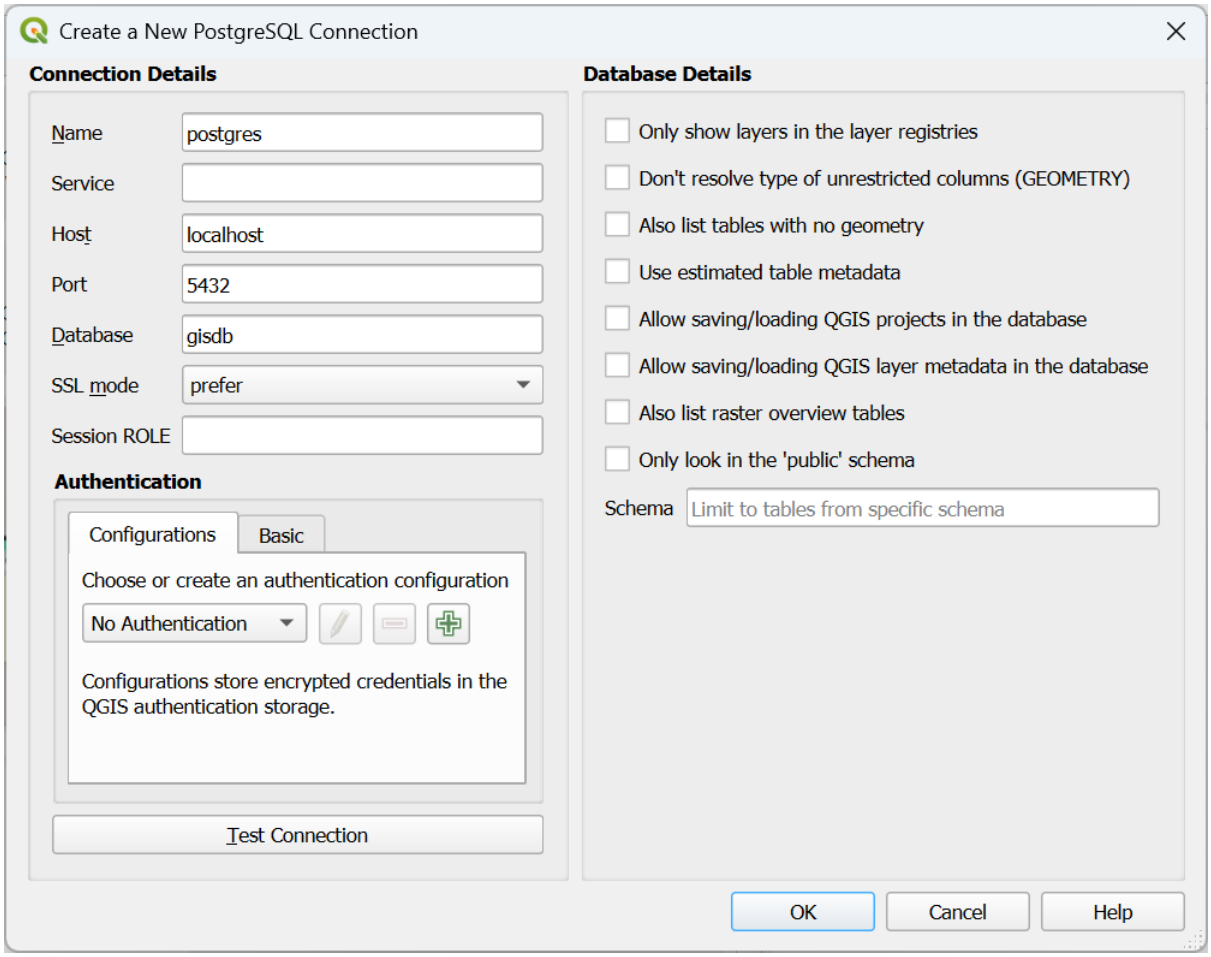


Fig. 11.12: Create a New PostgreSQL Connection Dialog

- *Name*: A name for this connection. It can be the same as *Database*.
- *Service*: Service parameter to be used alternatively to hostname/port (and potentially database). This can be defined in `pg_service.conf`. Check the *PostgreSQL Service connection file* section for more details.
- *Host*: Name of the database host. This must be a resolvable host name such as would be used to open a TCP/IP connection or ping the host. If the database is on the same computer as QGIS, simply enter *localhost* here.
- *Port*: Port number the PostgreSQL database server listens on. The default port for PostgreSQL is `5432`.
- *Database*: Name of the database.
- *SSL mode*: SSL encryption setup. The following options are available:
  - *Prefer* (the default): I don't care about encryption, but I wish to pay the overhead of encryption if the server supports it.
  - *Require*: I want my data to be encrypted, and I accept the overhead. I trust that the network will make sure I always connect to the server I want.
  - *Verify CA*: I want my data encrypted, and I accept the overhead. I want to be sure that I connect to a server that I trust.

  - *Verify Full*: I want my data encrypted, and I accept the overhead. I want to be sure that I connect to a server I trust, and that it's the one I specify.
  - *Allow*: I don't care about security, but I will pay the overhead of encryption if the server insists on it.
  - *Disable*: I don't care about security, and I don't want to pay the overhead of encryption.
- *Session role*: used to set the current user identifier of the current session. This is useful to automatically give the ownership of a new object (table, view, function) to the session_role group and thus share ownership and associated rights with all members of the session_role group. Read more about session role.
- *Authentication*, basic.
  - *User name*: User name used to log in to the database.
  - *Password*: Password used with *Username* to connect to the database.

  You can save any or both of the `User name` and `Password` parameters, in which case they will be used by default each time you need to connect to this database. If not saved, you'll be prompted to supply the credentials to connect to the database in next QGIS sessions. You can cancel the prompt, in which case the connection will fail. Click the *Ignore for session* button to avoid being prompted again during the current QGIS session. It is also possible to *Ignore for 10 seconds*. This will cancel the prompt but you need to wait 10 seconds before you can attempt to connect again. The connection parameters you entered are stored in a temporary internal cache and returned whenever a username/password for the same database is requested, until you end the current QGIS session.

  Warning

  QGIS User Settings and Security

  In the *Authentication* tab, saving **username** and **password** will keep unprotected credentials in the connection configuration. Those **credentials will be visible** if, for instance, you share the project file with someone. Therefore, it is advisable to save your credentials in an *Authentication configuration* instead (*Configurations* tab - See *Authentication System* for more details) or in a service connection file (see *PostgreSQL Service connection file* for example).

- *Authentication*, configurations. Choose an authentication configuration. You can add configurations using the  button. Choices are:
  - Basic authentication
  - PKI PKCS#12 authentication
  - PKI paths authentication
  - PKI stored identity certificate

Optionally, depending on the type of database, you can activate the following checkboxes:

-  *Only show layers in the layer registries*
-  *Don't resolve type of unrestricted columns (GEOMETRY)*
-  *Also list tables with no geometry*: indicates that tables without geometry should also be listed by default.
-  *Use estimated table metadata*: When initializing layers, various queries may be needed to establish the characteristics of the geometries stored in the database table. When this option is checked, these queries examine only a sample of the rows and use the table statistics, rather than the entire table. This can drastically speed up operations on large datasets, but may result in incorrect characterization of layers (e.g. the feature count of filtered layers will not be accurately determined) and may even cause strange behaviour if columns that are supposed to be unique actually are not.
-  *Allow saving/loading QGIS projects in the database* - more details *here*

- ☑ *Allow saving/loading QGIS layer metadata in the database* - more details *here*
- ☑ *Also list raster overview tables*
- ☑ *Only look in the 'public' schema*
- *Schema*: Allows to specify a single schema to limit a connection to. When set, only tables from the matching schema will be included in the browser panel and data source select for the connection. This can be used to limit the database work required to populate tables for a connection pointing to a large database store.

Once all parameters and options are set, you can test the connection by clicking the *Test Connection* button or apply it by clicking the *OK* button.

PostgreSQL Service connection file

The service connection file allows PostgreSQL connection parameters to be associated with a single service name. That service name can then be specified by a client and the associated settings will be used.

It's called `.pg_service.conf` under *nix systems (GNU/Linux, macOS etc.) and `pg_service.conf` on Windows.

The service file can look like this:

```
[water_service]
host=192.168.0.45
port=5433
dbname=gisdb
user=paul
password=paulspass

[wastewater_service]
host=dbserver.com
dbname=water
user=waterpass
```

Note

There are two services in the above example: `water_service` and `wastewater_service`. You can use these to connect from QGIS, pgAdmin, etc. by specifying only the name of the service you want to connect to (without the enclosing brackets). If you want to use the service with `psql`, you can do `psql service=water_service`.

You can find all the PostgreSQL parameters here

Note

If you don't want to save the passwords in the service file you can use the .pg_pass option.

Note

QGIS Server and service

When using a service file and QGIS Server, you must configure the service on the server side as well. You can follow the QGIS Server documentation.

On *nix operating systems (GNU/Linux, macOS etc.) you can save the `.pg_service.conf` file in the user's home directory and PostgreSQL clients will automatically be aware of it. For example, if the logged user is `web`, `.pg_service.conf` should be saved in the `/home/web/` directory in order to directly work (without specifying any other environment variables).

You can specify the location of the service file by creating a `PGSERVICEFILE` environment variable (e.g. run the `export PGSERVICEFILE=/home/web/.pg_service.conf` command under your *nix OS to temporarily set the `PGSERVICEFILE` variable).

You can also make the service file available system-wide (all users) either by placing the `.pg_service.conf` file in `pg_config --sysconfdir` or by adding the `PGSYSCONFDIR` environment variable to specify the directory containing the service file. If service definitions with the same name exist in the user and the system file, the user file takes precedence.

Warning

There are some caveats under Windows:

- The service file should be saved as `pg_service.conf` and not as `.pg_service.conf`.
- The service file should be saved in Unix format in order to work. One way to do it is to open it with Notepad++ and *Edit* ► *EOL Conversion* ► *UNIX Format* ► *File save*.
- You can add environmental variables in various ways; a tested one, known to work reliably, is *Control Panel* ► *System and Security* ► *System* ► *Advanced system settings* ► *Environment Variables* adding `PGSERVICEFILE` with the path - e.g. `C:\Users\John\pg_service.conf`
- After adding an environment variable you may also need to restart the computer.

Connecting to Oracle Spatial

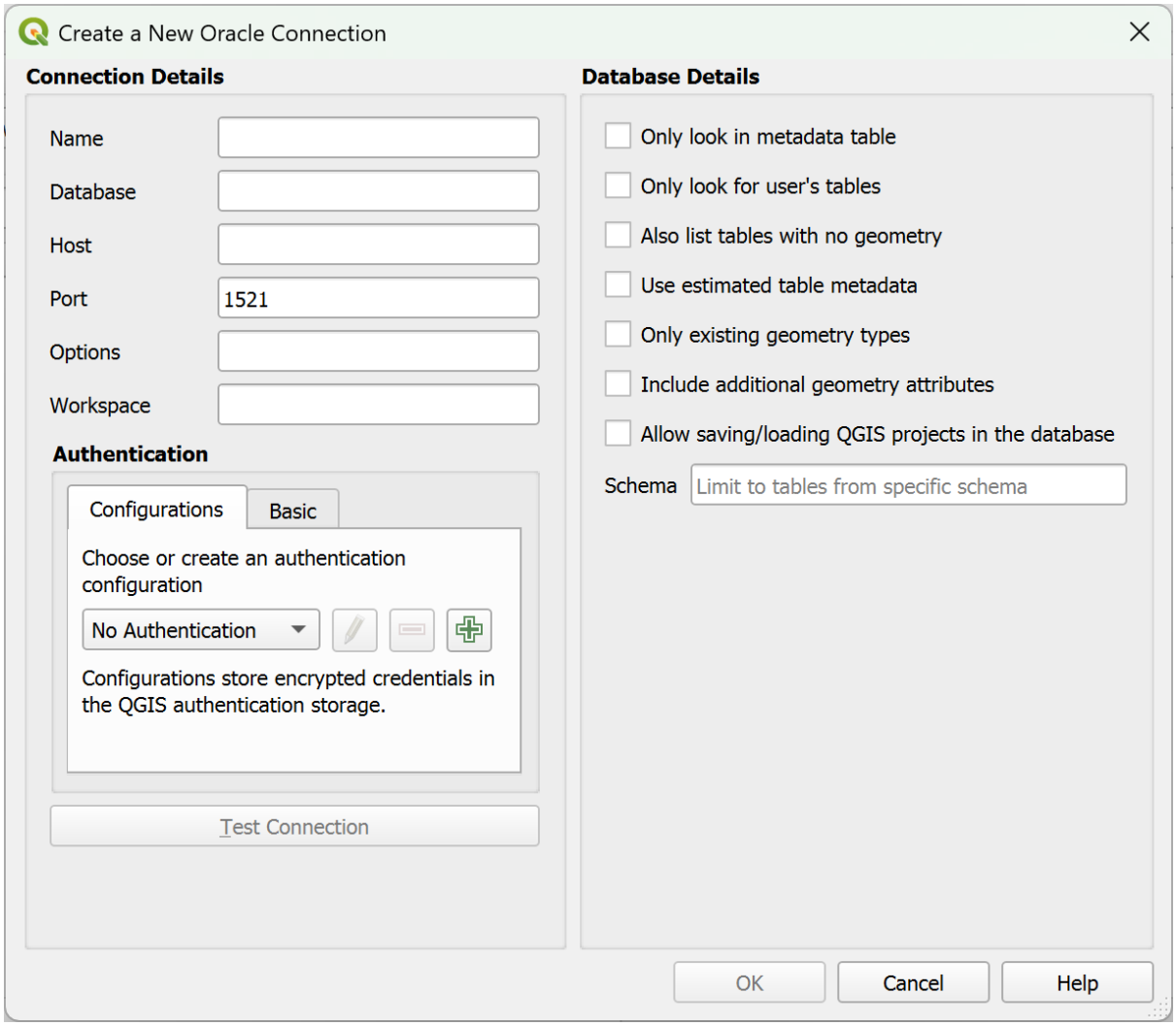


Fig. 11.13: Create a New Oracle Connection Dialog

The spatial features in Oracle Spatial aid users in managing geographic and location data in a native type within an Oracle database. The connection dialog proposes:

- *Name*: A name for this connection. It can be the same as *Database*;
- *Database*: SID or SERVICE_NAME of the Oracle instance;
- *Host*: The name of the database host;
- *Port*: Port number the Oracle database server listens on. The default port is `1521`;
- *Options*: Oracle connection specific options (e.g. OCI_ATTR_PREFETCH_ROWS, OCI_ATTR_PREFETCH_MEMORY). The format of the options string is a semicolon separated list of option names or option=value pairs;
- *Workspace*: Workspace to switch to;
- *Authentication* ► *Basic*.
  - *User name*: User name used to connect to the database.
  - *Password*: Password used to connect to the database.

Optionally, you can activate the following checkboxes:

- *Only look in metadata table*: restricts the displayed tables to those that are in the `all_sdo_geom_metadata` view. This can speed up the initial display of spatial tables.
- *Only look for user's tables*: when searching for spatial tables, restricts the search to tables that are owned by the user.
- *Also list tables with no geometry*: indicates that tables without geometry should also be listed by default.
- *Use estimated table statistics for the layer metadata*: when the layer is set up, various metadata are required for the Oracle table. This includes information such as the table row count, geometry type and spatial extents of the data in the geometry column. If the table contains a large number of rows, determining this metadata can be time-consuming. By activating this option, the following fast table metadata operations are done: Row count is determined from `all_tables.num_rows`. Table extents are always determined with the SDO_TUNE.EXTENTS_OF function, even if a layer filter is applied. Table geometry is determined from the first 100 non-null geometry rows in the table.
- *Only existing geometry types*: only lists the existing geometry types and don't offer to add others.
- *Include additional geometry attributes*.
- *Allow saving/loading QGIS projects in the database* - more details *here*
- *Schema*: Allows to specify a single schema to limit a connection to. When set, only tables from the matching schema will be included in the browser panel and data source select for the connection. This can be used to limit the database work required to populate tables for a connection pointing to a large database store.

> **Tip**
>
> **Oracle Spatial Layers**
>
> Normally, an Oracle Spatial layer is defined by an entry in the **USER_SDO_METADATA** table.
>
> To ensure that selection tools work correctly, it is recommended that your tables have a **primary key**.

Connecting to MS SQL Server

As mentioned in *Creating a stored Connection* QGIS allows you to create MS SQL Server connection through *Data Source Manager*.

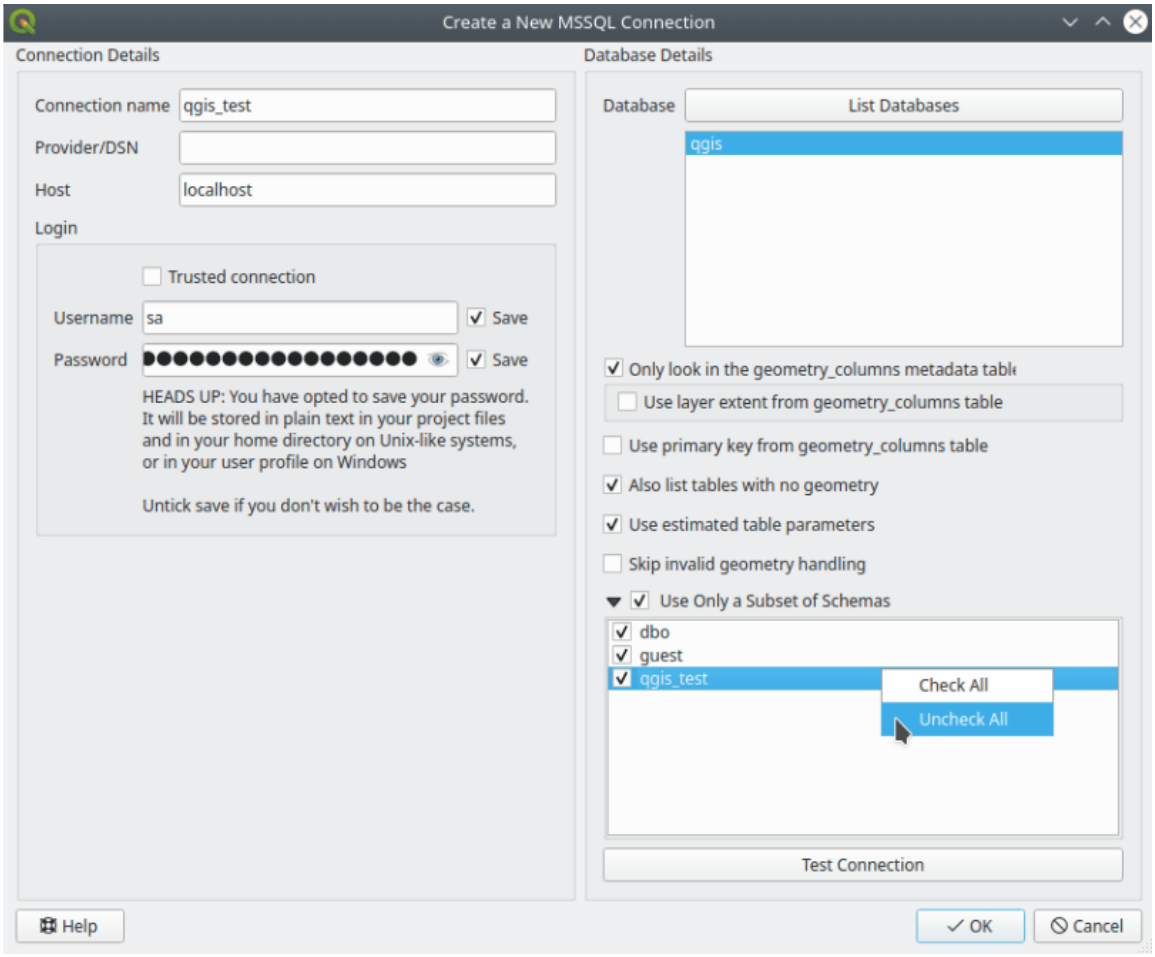


Fig. 11.14: MS SQL Server Connection

To create a new MS SQL Server connection, you need to provide some of the following information in the *Connection Details* dialog:

- *Connection name*
- *Provider/DSN*
- *Host*
- *Login* information. You can choose to *Save* your credentials.

Navigate to the *Database Details* section and click the *List Databases* button to view the available datasets. Select datasets that you want, then press *OK*. Optionally, you can also perform a *Test Connection*. Once you click *OK* the *Create a New MS SQL Server Connection* dialog will close and in the *Data Source Manager* press *Connect*, select a layer and then click *Add*.

Optionally, you can activate the following options:

- *Only look in the geometry_columns metadata table*: restricts the available tables to the ones in the `geometry_columns` metadata table when scanning for tables. This can speed up the table scanning.
- *Use layer extent from geometry_columns table*: this option, dependent on the previous one, allows QGIS to skip extent calculation when loading layers and thus lowering the amount of time needed to load them. It relies on extent manually specified using additional QGIS-specific columns (`qgis_xmin`, `qgis_xmax`, `qgis_ymin`, `qgis_ymax`) in the `geometry_columns` table.
- *Use primary key from geometry_columns table*: allows QGIS to skip primary key calculation for views when loading them, thus lowering the amount of time needed to load them. It relies on names manually filled

in a QGIS-specific `qgis_pkey` column set in the `geometry_columns` table. If more than one column is used for the primary key, they should be filled as comma separated values.

- ☑ *Also list table with no geometry*: tables without a geometry column attached will also be shown in the available table list.
- ☑ *Use estimated table parameters*: only estimated table metadata will be used. This avoids a slow table scan, but may result in incorrect layer properties such as layer extent.
- ☑ *Skip invalid geometry handling*: all handling of records with invalid geometry will be disabled. This speeds up the provider, however, if any invalid geometries are present in a table then the result is unpredictable and may include missing records. Only check this option if you are certain that all geometries present in the database are valid, and any newly added geometries or tables will also be valid.
- ☑ *Use only a Subset of Schemas* will allow you to filter schemas for MS SQL connection. If enabled, only checked schemas will be displayed. You can right-click to *Check* or *Uncheck* any schema in the list.

Connecting to SAP HANA

Note

You require the SAP HANA Client to connect to a SAP HANA database. You can download the SAP HANA Client for your platform at the SAP Development Tools website.

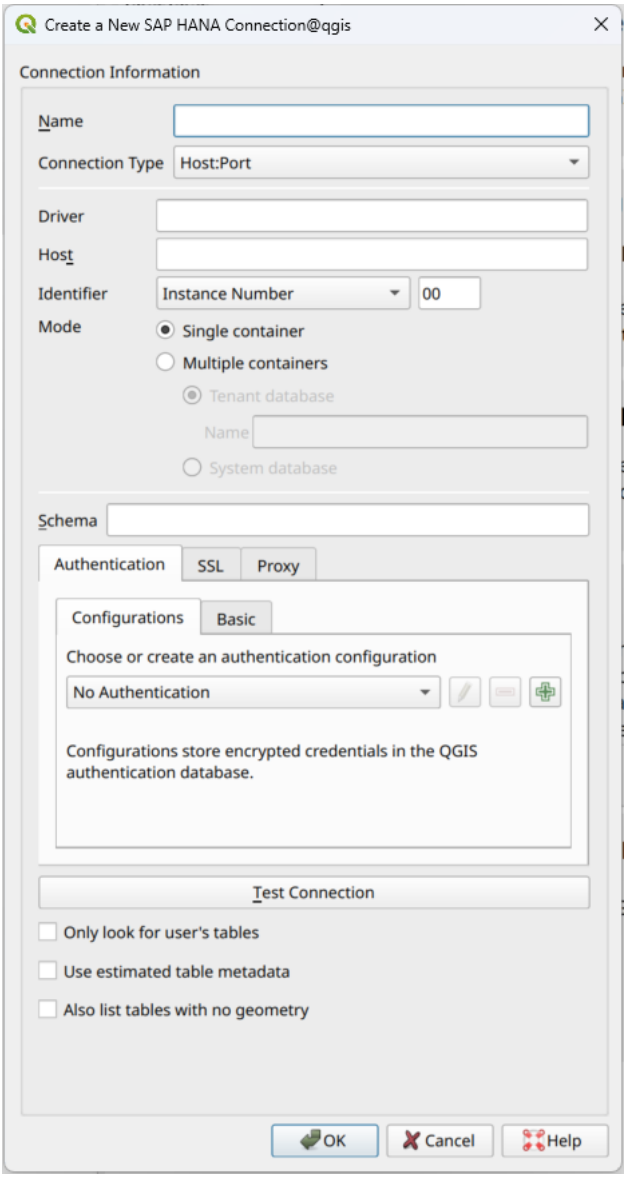


Fig. 11.15: Create a New SAP HANA Connection Dialog

The following parameters can be entered:

- *Name*: A name for this connection.
- *Driver* : The name of the HANA ODBC driver. It is `HDBODBC` if you are using 64-bit QGIS, `HDBODBC32` if you are using 32-bit QGIS. The appropriate driver name is entered automatically.
- *Driver* : Either the name under which the SAP HANA ODBC driver has been registered in `/etc/odbcinst.ini` or the full path to the SAP HANA ODBC driver. The SAP HANA Client installer will install the ODBC driver to `/usr/sap/hdbclient/libodbcHDB.so` by default.
- *Host*: The name of the database host.
- *Identifier*: Identifies the instance to connect to on the host. This can be either *Instance Number* or *Port Number*. Instance numbers consist of two digits, port numbers are in the range from 1 to 65,535.
- *Mode*: Specifies the mode in which the SAP HANA instance runs. This setting is only taken into account if *Identifier* is set to *Instance Number*. If the database hosts multiple containers, you can either connect to a tenant with the name given at *Tenant database* or you can connect to the system database.

- *Schema*: This parameter is optional. If a schema name is given, QGIS will only search for data in that schema. If this field is left blank, QGIS will search for data in all schemas.
- *Authentication* ► *Basic*.
  - *User name*: User name used to connect to the database.
  - *Password*: Password used to connect to the database.
- *SSL Settings*
  - *Enable TLS/SSL encryption*: Enables TLS 1.1 - TLS1.2 encryption. The server will choose the highest available.
  - *Provider*: Specifies the cryptographic library provider used for SSL communication. *sapcrypto* should work on all platforms, *openssl* should work on , *mscrypto* should work on and *commoncrypto* requires CommonCryptoLib to be installed.
  - *Validate SSL certificate*: If checked, the SSL certificate will be validated using the truststore given in *Trust store file with public key*.
  - *Override hostname in certificate*: Specifies the host name used to verify server's identity. The host name specified here verifies the identity of the server instead of the host name with which the connection was established. If you specify `*` as the host name, then the server's host name is not validated. Other wildcards are not permitted.
  - *Keystore file with private key*: Currently ignored. This parameter might allow to authenticate via certificate instead via user and password in future.
  - *Trust store file with public key*: Specifies the path to a trust store file that contains the server's public certificates if using OpenSSL. Typically, the trust store contains the root certificate or the certificate of the certification authority that signed the server's public certificates. If you are using the cryptographic library CommonCryptoLib or msCrypto, then leave this property empty.
- *Only look for user's tables*: If checked, QGIS searches only for tables and views that are owned by the user that connects to the database.
- *Use estimated table metadata*: If checked, estimated table metadata will be used if available. For large tables, this avoids slow table loads and potentially expensive computations, but may result in incorrect layer properties such as layer extent. The fast extent estimation is available starting with QRC1/2024 and SP8 in HANA Cloud and HANA On-Premise respectively.
- *Also list tables with no geometries*: If checked, QGIS searches also for tables and views that do not contain a spatial column.

Tip

Connecting to SAP HANA Cloud

If you'd like to connect to an SAP HANA Cloud instance, you usually must set *Port Number* to `443` and check *Enable TLS/SSL encryption*.

Loading a Database Layer

Once you have one or more connections defined to a database (see section *Creating a stored Connection*), you can load layers from it. Of course, this requires that data are available. See section *Importing Data into PostgreSQL* for a discussion on importing data into a PostgreSQL database.

To load a layer from a database, you can perform the following steps:

1. Open the corresponding tab of the database in the *Data Source Manager* dialog.
2. Choose the connection name from the drop-down list and press *Connect*.
3. The table below will be filled with your data grouped by schema, with a number of metadata information helpful for loading.
4. Select or unselect *Also list tables with no geometry*.
5. Optionally, use some *Search Options* to reduce the list of tables to those matching your search. You can also set this option before you hit the *Connect* button, speeding up the database fetching.
6. Find the layer(s) you wish to add in the list of available layers.
7. Select it by clicking on it. You can select multiple layers by holding down the `Shift` or `Ctrl` key while clicking.

   Layers can be selected only if they have no warning icon at the left-hand side of their first column. This may indicate an issue to detect:

   - features geometry type: in which case you can select the appropriate one in the drop-dowmn list of the *Spatial Type* column
   - layer CRS: you can enter the correct code in the *SRID* column
   - layer's primary key, in order to unequivocally identify each feature: this can be fixed by selecting one or more attributes in the drop-down list at the corresponding *Feature id* column.

   > **Tip**
   >
   > **Use the first column to store "primary keys" for views**
   >
   > Since PostgreSQL views don't support primary keys, a unique attribute or combination of attributes should always be selected. To help users to speed-up workflows, QGIS automatically selects the first attribute in the view. Therefore, users can define their views in a way that a unique column is in the first position of the view's definition. In this way, the view will be loaded with no extra interaction and the warning icon will never appear.

8. If applicable, use the *Set Filter* button (or double-click the layer) to start the *Query Builder* dialog (see section *Query Builder*) and define which features to load from the selected layer. The filter expression appears in the `sql` column. This restriction can be removed or edited in the *Layer Properties* ► *General* ► *Provider Feature Filter* frame.
9. The checkbox in the `Select at id` column that is activated by default gets the feature ids without the attributes and generally speeds up the data loading.
10. Click on the *Add* button to add the layer to the map.

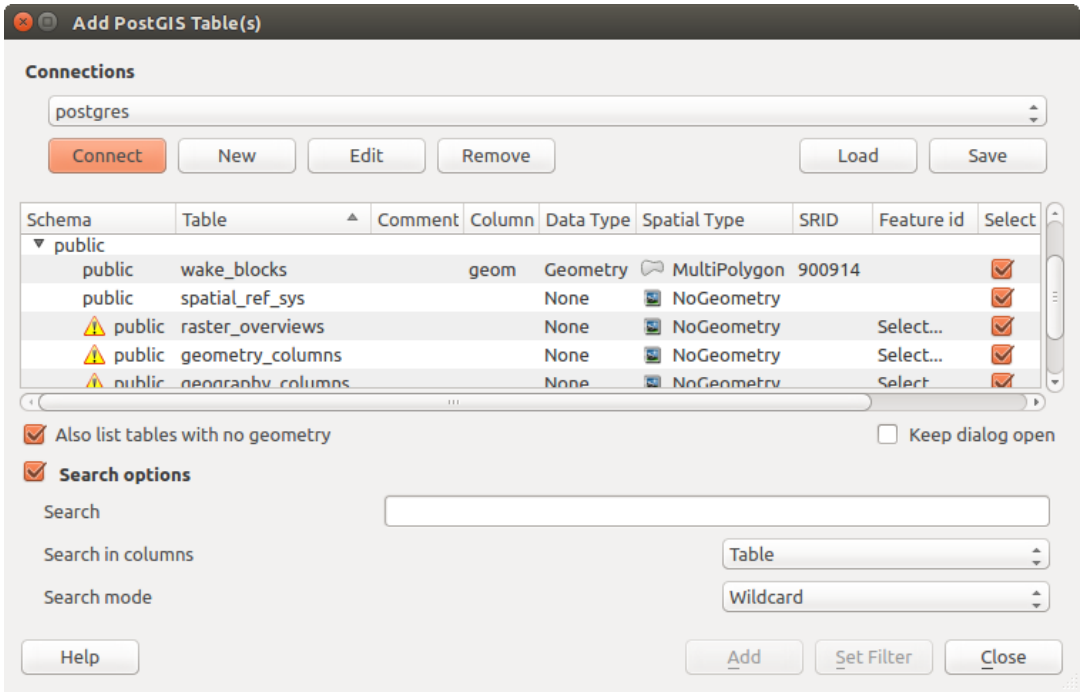


Fig. 11.16: Add PostgreSQL Table(s) Dialog

Tip

Use the Browser Panel to speed up loading of database table(s)

Adding DB tables from the *Data Source Manager* may sometimes be time consuming as QGIS fetches statistics and properties (e.g. geometry type and field, CRS, number of features) for each table beforehand. To avoid this, once *the connection is set*, it is better to use the *Browser Panel* or the *DB Manager* to drag and drop the database tables into the map canvas.

11.1.4 The Layer Metadata Search Panel

By default, QGIS can retrieve layers metadata from the connections or data providers that allow metadata storage (more details on *saving metadata to the database*). The *Metadata search* panel allows to browse the layers by their metadata and add them to the project (either with a double-click or the *Add* button). The list can be filtered:

- by text, watching a set of metadata properties (identifier, title, abstract)
- by spatial extent, using the current *project extent* or the map canvas extent
- by the layer (geometry) type

Note

The sources of metadata are implemented through a layer metadata provider system that can be extended by plugins.

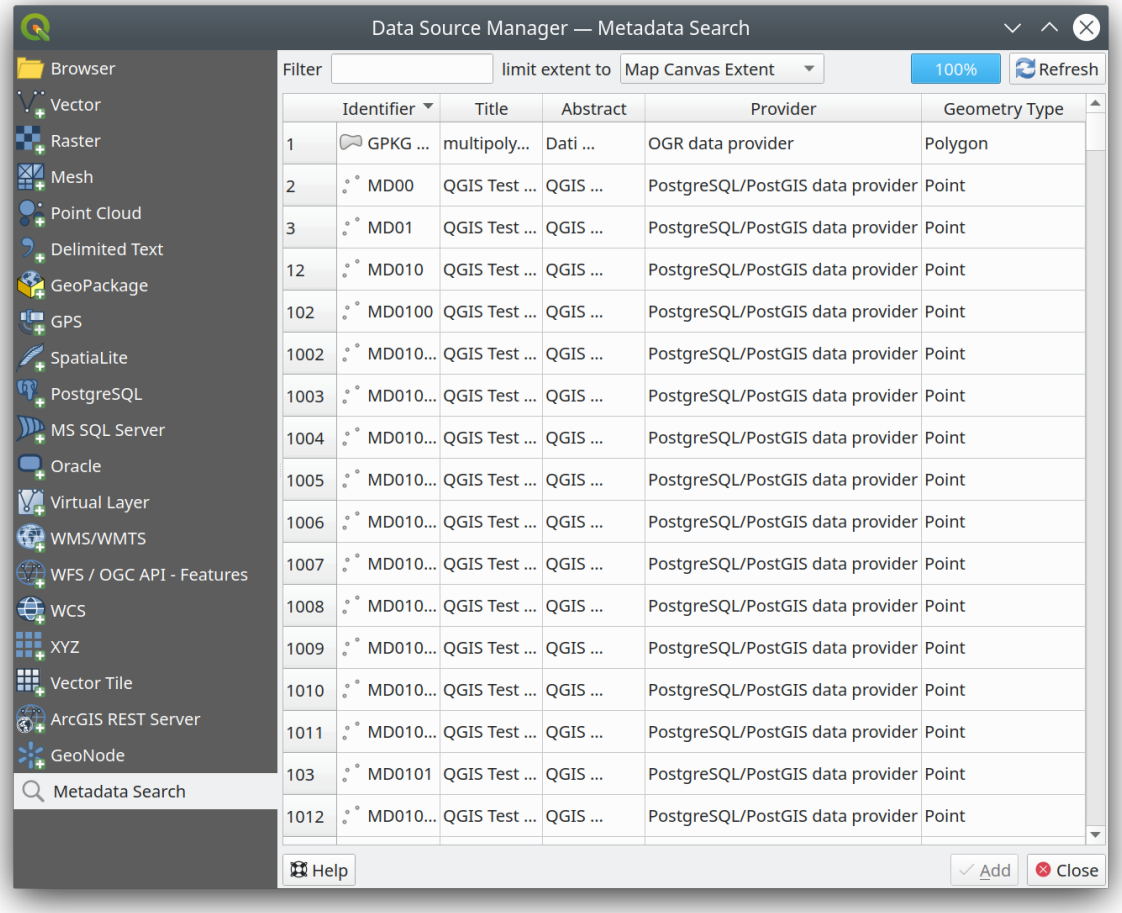


Fig. 11.17: Layer Metadata Search Panel

11.1.5 QGIS Custom formats

QGIS proposes two custom formats:

- Temporary Scratch Layer: a memory layer that is bound to the project (see *Creating a new Temporary Scratch Layer* for more information)
- Virtual Layers: a layer resulting from a query on other layer(s) (see *Creating virtual layers* for more information)

11.1.6 QLR - QGIS Layer Definition File

Layer definitions can be saved as a *Layer Definition File* (QLR - `.qlr`) using *Export* ► *Save As Layer Definition File…* in the layer context menu.

The QLR format makes it possible to share "complete" QGIS layers with other QGIS users. QLR files contain links to the data sources and all the QGIS style information necessary to style the layer.

QLR files are shown in the Browser Panel and can be used to add layers (with their saved styles) to the Layers Panel. You can also drag and drop QLR files from the system file manager into the map canvas.

Available actions for QLR files in the Browser Panel are:

- *Export Layer* ► *To file*
- *Add Layer to Project*

- *Layer Properties…*

11.1.7 Connecting to web services

With QGIS you can get access to different types of OGC web services (WM(T)S, WFS(-T), WCS, CSW, …). Thanks to QGIS Server, you can also publish such services. QGIS-Server-manual contains descriptions of these capabilities.

Using Vector Tiles services

Vector Tile services can be added via the  *Vector Tiles* tab of the *Data Source Manager* dialog or the contextual menu of the *Vector Tiles* entry in the *Browser* panel. Services can be either a *New Generic Connection…* or a *New ArcGIS Vector Tile Service Connection…*.

You set up a service by adding:

- a *Name*
- a *Style URL*: a URL to a MapBox GL JSON style configuration. If provided, then that style will be applied whenever the layers from the connection are added to QGIS. In the case of Arcgis vector tile service connections, the URL overrides the default style configuration specified in the server configuration.

  You can load vector tiles directly from a *Style URL*. The data source is automatically parsed from the style, and URLs with multiple sources are supported. That makes *Source URL* optional.
- the *Source URL*: of the type `http://example.com/{z}/{x}/{y}.pbf` for generic services and `http://example.com/arcgis/rest/services/Layer/VectorTileServer` for ArcGIS based services. The service must provide tiles in `.pbf` format.
- the *Min. Zoom Level* and the *Max. Zoom Level*. Vector Tiles have a pyramid structure. By using these options you have the opportunity to individually generate layers from the tile pyramid. These layers will then be used to render the Vector Tile in QGIS.

  For Mercator projection (used by OpenStreetMap Vector Tiles) Zoom Level 0 represents the whole world at a scale of 1:500.000.000. Zoom Level 14 represents the scale 1:35.000.
- the *authentication* configuration if necessary
- a *Referer*

Fig. 11.18 shows the dialog with the Vector Tiles service configuration.

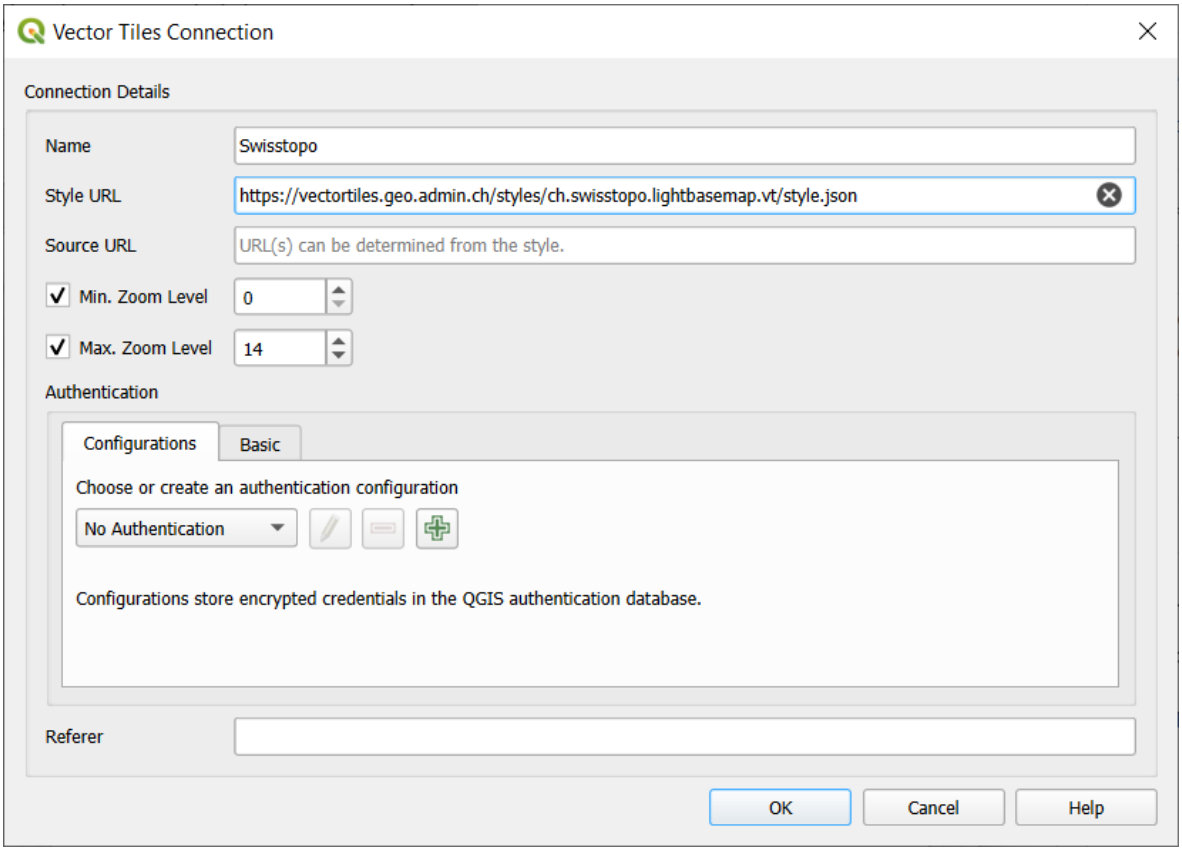


Fig. 11.18: Vector Tiles - Service configuration

Configurations can be saved to `.XML` file (*Save Connections*) through the *Vector Tiles* entry in *Data Source Manager* dialog or its context menu in the *Browser* panel. Likewise, they can be added from a file (*Load Connections*).

Once a connection to a vector tile service is set, it's possible to:

- *Edit* the vector tile connection settings
- *Remove* the connection
- From the *Browser* panel, right-click over the entry and you can also:
  - *Add layer to project*: a double-click also adds the layer
  - View the *Layer Properties...* and get access to metadata and a preview of the data provided by the service. More settings are available when the layer has been loaded into the project.

Using XYZ Tile services

XYZ Tile services can be added via the *XYZ* tab of the *Data Source Manager* dialog or the contextual menu of the *XYZ Tiles* entry in the *Browser* panel. By default, QGIS provides some default and ready-to-use XYZ Tiles services:

- *Mapzen Global Terrain*, allowing an immediate access to global DEM source for the projects. More details and resources at https://registry.opendata.aws/terrain-tiles/
- *OpenStreetMap* to access the world 2D map. Fig. 11.19 shows the dialog with the OpenStreetMap XYZ Tile service configuration.

To add a new service, press *New* (respectively *New Connection* from the Browser panel) and provide:

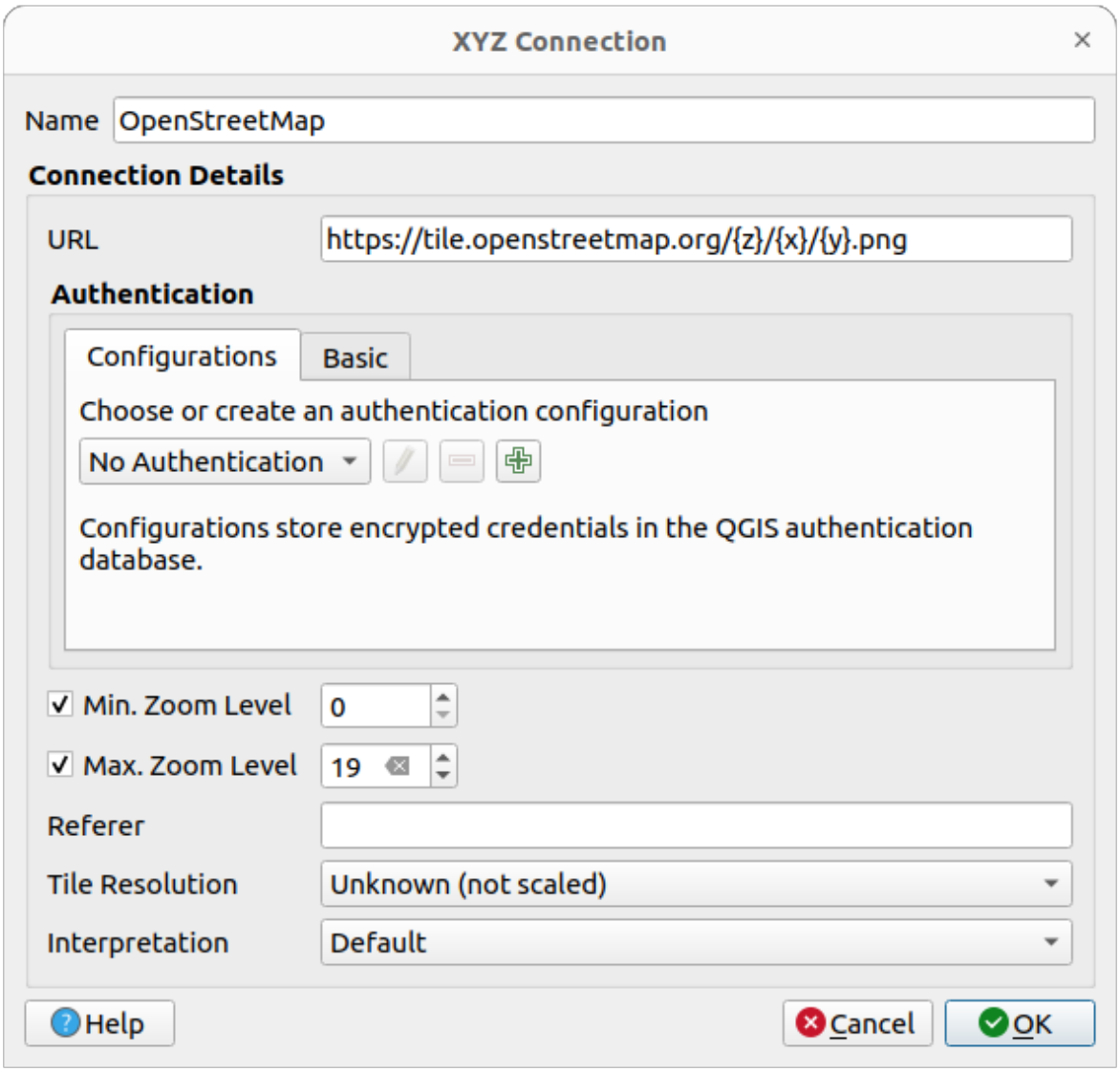


Fig. 11.19: XYZ Tiles - OpenStreetMap configuration

- a *Name*
- the *URL*, you can add `http://example.com/{z}/{x}/{y}.png` or `file:///local_path/{z}/{x}/{y}.png`
- the *authentication* configuration if necessary
- the *Min. Zoom level* and *Max. Zoom level*
- a *Referer*
- the *Tile Resolution*: possible values are *Unknown (not scaled)*, *Standard (256x256 / 96DPI)* and *High (512x512 / 192DPI)*
- *Interpretation*: converts WMTS/XYZ raster datasets to a raster layer of single band float type following a predefined encoding scheme. Supported schemes are *Default* (no conversion is done), *MapTiler Terrain RGB* and *Terrarium Terrain RGB*. The selected converter will translate the RGB source values to float values for each pixel. Once loaded, the layer will be presented as a single band floating point raster layer, ready for styling using QGIS usual *raster renderers*.

Press *OK* to establish the connection. It will then be possible to:

- *Add* the new layer to the project; it is loaded with the name given in the settings.
- *Edit* the XYZ connection settings
- *Remove* the connection
- From the *Browser* panel, right-click over the entry and you can also:
  - *Export layer…* ► *To File*, *saving it as a raster*
  - *Add layer to project*: a double-click also adds the layer
  - View the *Layer Properties…* and get access to metadata and a preview of the data provided by the service. More settings are available when the layer has been loaded into the project.

Configurations can be saved to `.XML` file (*Save Connections*) through the *XYZ* entry in *Data Source Manager* dialog or its contextual menu in the *Browser* panel. Likewise, they can be added from a file (*Load Connections*).

The XML file for OpenStreetMap looks like this:

```
<!DOCTYPE connections>
<qgsXYZTilesConnections version="1.0">
  <xyztiles url="https://tile.openstreetmap.org/{z}/{x}/{y}.png"
   zmin="0" zmax="19" tilePixelRatio="0" password="" name="OpenStreetMap"
   username="" authcfg="" referer=""/>
</qgsXYZTilesConnections>
```

Tip

Loading XYZ tiles without creating a connection

It is also possible to add XYZ tiles to a project without necessarily storing its connection settings in you user profile (e.g. for a dataset you may need once). In the *Data Source Manager* ► *XYZ* tab, edit any properties in the *Connection Details* group. The *Name* field above should turn into `Custom`. Press *Add* to load the layer in the project. It will be named by default `XYZ Layer`.

Examples of XYZ Tile services:

- OpenStreetMap Monochrome: *URL*: `http://tiles.wmflabs.org/bw-mapnik/{z}/{x}/{y}.png`, *Min. Zoom Level*: 0, *Max. Zoom Level*: 19.
- Google Maps: *URL*: `https://mt1.google.com/vt/lyrs=m&x={x}&y={y}&z={z}`, *Min. Zoom Level*: 0, *Max. Zoom Level*: 19.
- Open Weather Map Temperature: *URL*: `http://tile.openweathermap.org/map/temp_new/{z}/{x}/{y}.png?appid={api_key}` *Min. Zoom Level*: 0, *Max. Zoom Level*: 19.

Using ArcGIS REST Servers

ArcGIS REST Servers can be added via the  *ArcGIS REST Server* tab of the *Data Source Manager* dialog or the contextual menu of the *ArcGIS REST Servers* entry in the *Browser* panel. Press *New* (respectively *New Connection*) and provide:

- a *Name*
- the *URL*
- a *Prefix*: This is used to specify the proxy prefix in the URL, which is necessary for some ArcGIS servers that use web proxy prefixes.
- a *Community endpoint URL*
- a *Content endpoint URL*
- the *authentication* configuration if necessary

- a *Referer*

> **Note**
>
> ArcGIS Feature Service connections which have their corresponding Portal endpoint URLS set can be explored by content groups in the browser panel.
>
> If a connection has the Portal endpoints set, then expanding out the connection in the browser will show a "Groups" and "Services" folder, instead of the full list of services usually shown. Expanding out the groups folder will show a list of all content groups that the user is a member of, each of which can be expanded to show the service items belonging to that group.

Configurations can be saved to `.XML` file (*Save Connections*) through the *ArcGIS REST Server* entry in *Data Source Manager* dialog. Likewise, they can be added from a file (*Load Connections*).

Once a connection to an ArcGIS REST Server is set, it's possible to:

- *Edit* the ArcGIS REST Server connection settings
- *Remove* the connection
- *Refresh* the connection
- use a filter for the available layers
- choose from a list of available layers with the option to *Only request features overlapping the current view extent*
- From the *Browser* panel, right-click over the connection entry and you can:
  - *Refresh*
  - *Edit connection…*
  - *Remove connection…*
  - *View Service Info* which will open the default web browser and display the Service Info.
- Right-click over the layer entry and you can also:
  - *View Service Info* which will open the default web browser and display the Service Info.
  - *Export layer…* ► *To File*
  - *Add layer to project*: a double-click also adds the layer
  - View the *Layer Properties…* and get access to metadata and a preview of the data provided by the service. More settings are available when the layer has been loaded into the project.

Using 3D tiled scene services

QGIS supports multiple formats of 3D tiled datasets, grouped together as "tiled scenes". These include Cesium 3D Tiles and Quantized Mesh tiles.

To load a tiled scene dataset into QGIS, use the *Scene* tab in the *Data Source Manager* dialog.

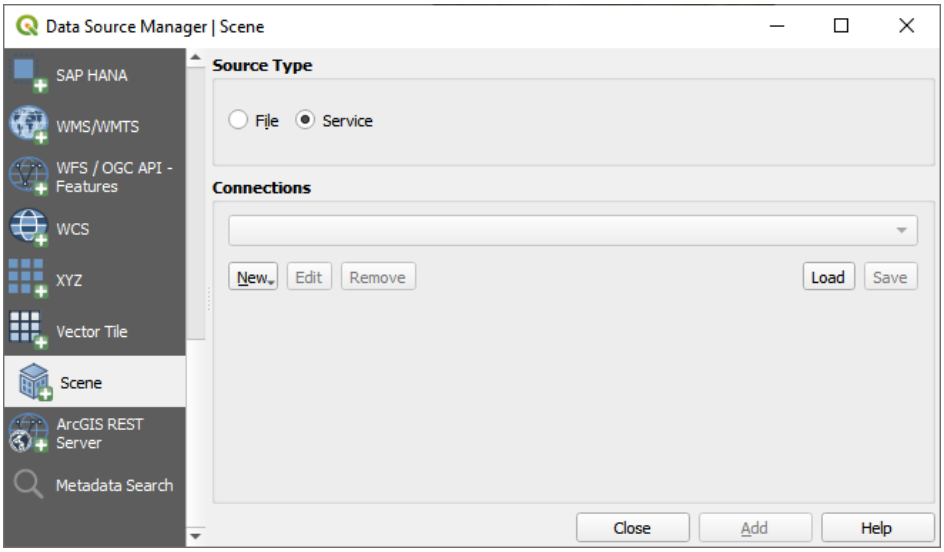


Fig. 11.20: Data Source Manager - Scene

Create a connection by clicking on *New*. You can add a *New Cesium 3D Tiles Connection* or a *New Quantized Mesh Connection*.

Choose a *Name* and set the *URL* to the URL of a layer description JSON file.

The URL may be remote (e.g. `http://example.com/tileset.json`) or local (e.g. `file:///path/to/tiles/tileset.json`).

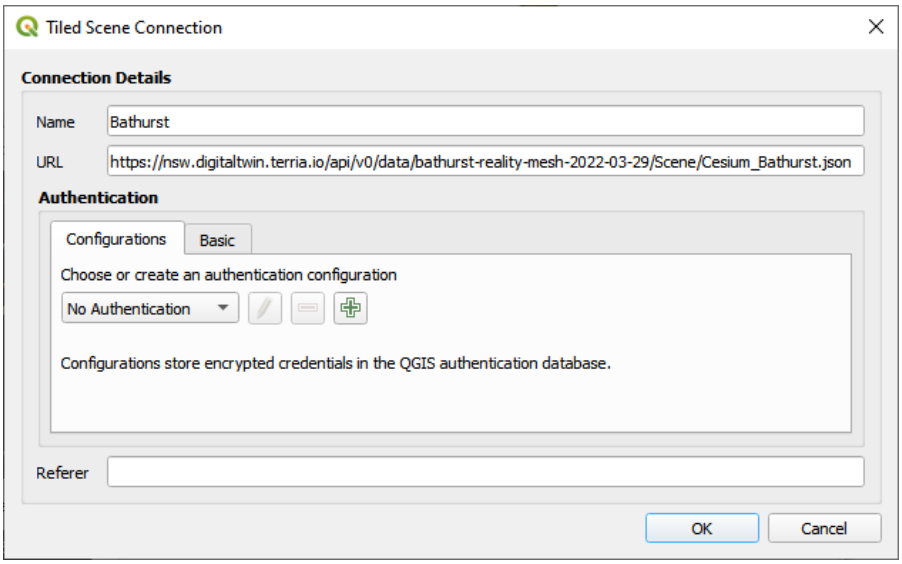


Fig. 11.21: Tiled Scene Connection

You can also add the service from *Browser Panel*.

After creating new connection you are able to *Add* the new layer to your map.

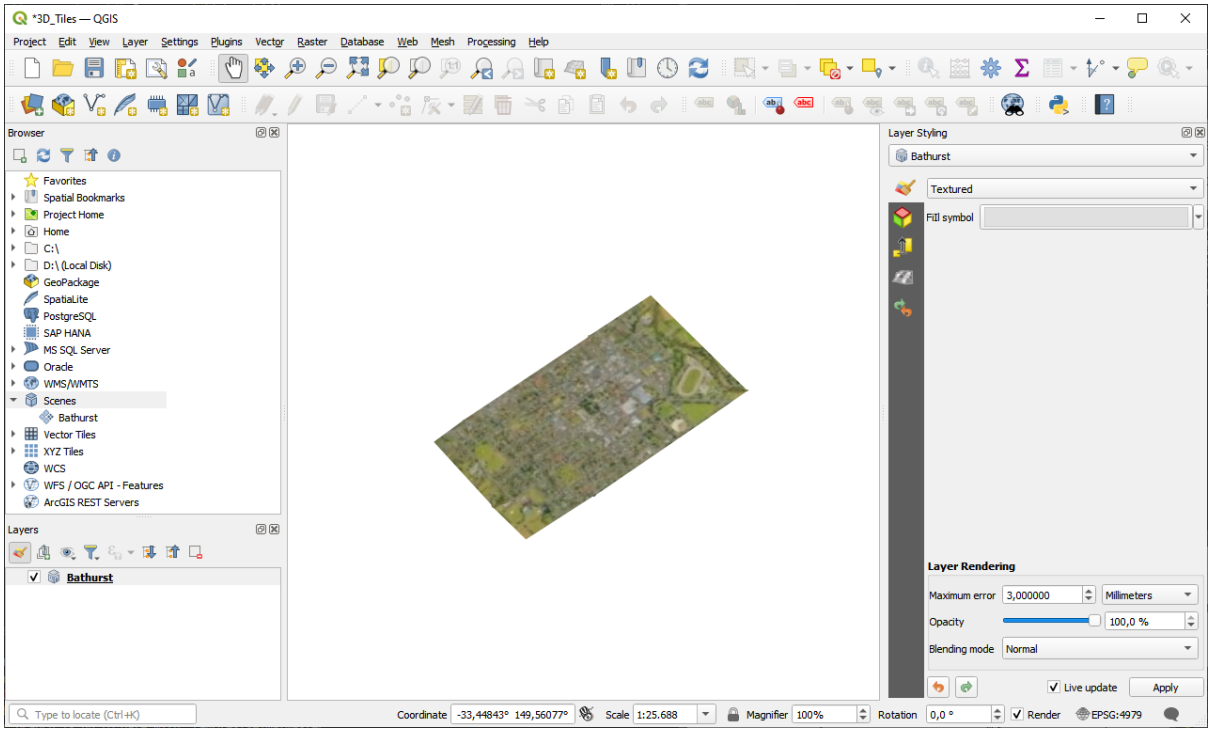


Fig. 11.22: 3D Tiles Layer - Textured

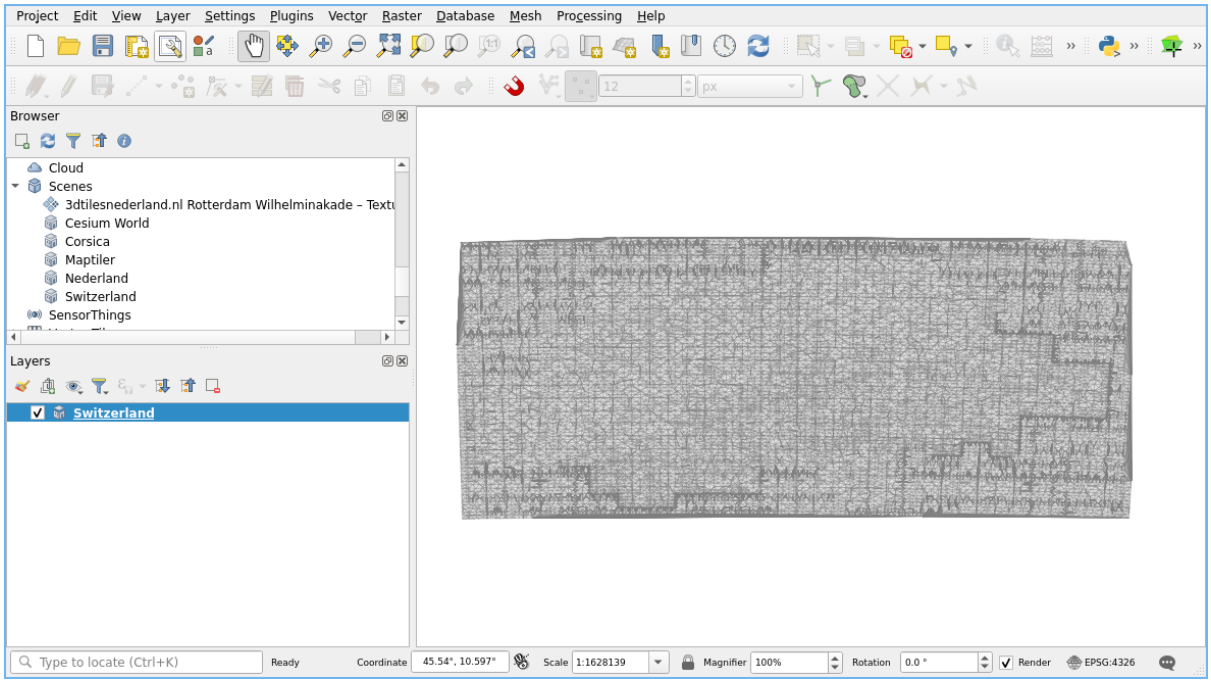


Fig. 11.23: Quantized Mesh layer

Using Cloud Connections

QGIS supports connections to cloud services like Alibaba Cloud OSS, Amazon S3, Google Cloud Storage, Microsoft Azure Blob Storage, Microsoft Azure Data Lake Storage, and OpenStack Swift Object Storage. You can load vector and raster data from these services into QGIS. Set up a new *Cloud* connection in the *Browser* panel by right-clicking on the *Cloud* entry and selecting *New Connection*. You will see a drop-down list of available cloud services. Select the service you want to connect to and fill in the required fields:

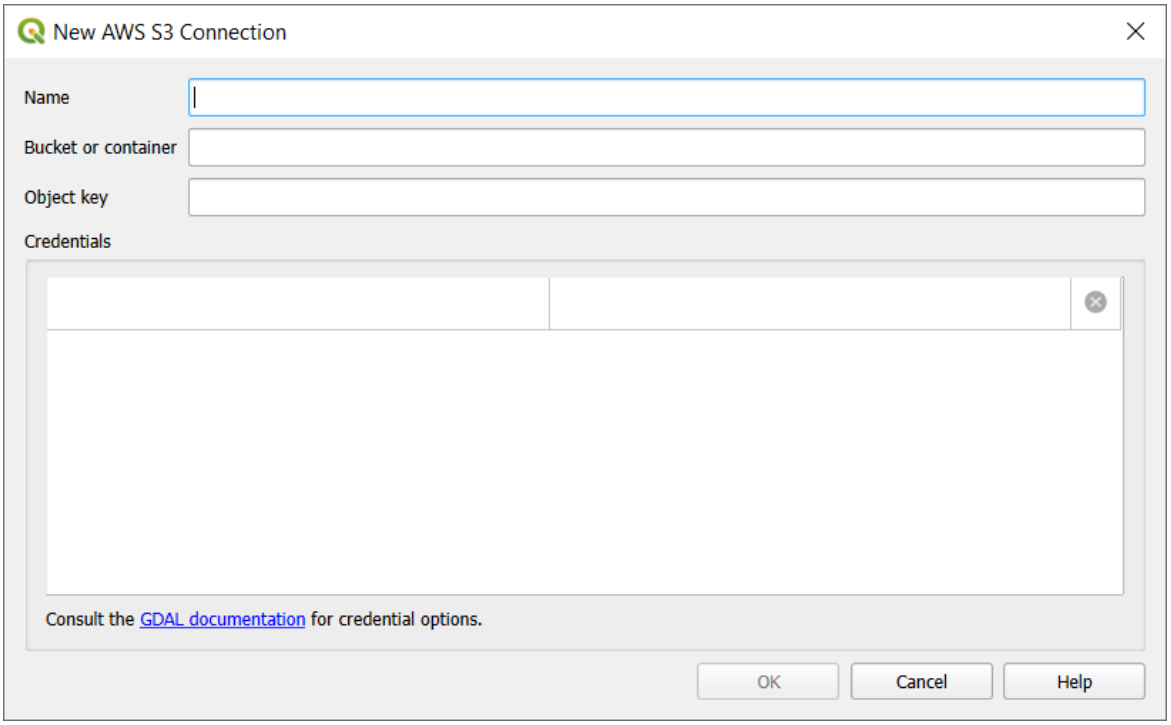


Fig. 11.24: Cloud Connection Dialog

- *Name*: A name for the connection.
- *Bucket or Container*: The name of the bucket or container in the cloud service.
- *Object Key* (optional): The key of the object in the bucket or container.
- *Credentials*: The credentials to access the cloud service.

You can also choose to *Save Connection* to an XML file or *Load Connection* from an XML file.

11.2 Creating Layers

Layers can be created in many ways, including:

- empty layers from scratch
- layers from existing layers
- layers from the clipboard
- layers as a result of an SQL-like query based on one or many layers (*virtual layers*)

QGIS also provides tools to import/export from/to different formats.

11.2.1 Creating new vector layers

QGIS allows you to create new layers in different formats. It provides tools for creating GeoPackage, Shapefile, SpatiaLite, GPX format and Temporary Scratch layers (aka memory layers). Creation of a *new GRASS layer* is supported within the GRASS plugin.

Creating a new GeoPackage layer

To create a new GeoPackage layer, press the *New GeoPackage Layer…* button in the *Layer* ► *Create Layer* ► menu or from the *Data Source Manager* toolbar. You can also create a new GeoPackage layer through the *Browser Panel* by selecting the *Create Database and Layer…*. The *New GeoPackage Layer* dialog will be displayed as shown in Fig. 11.25.

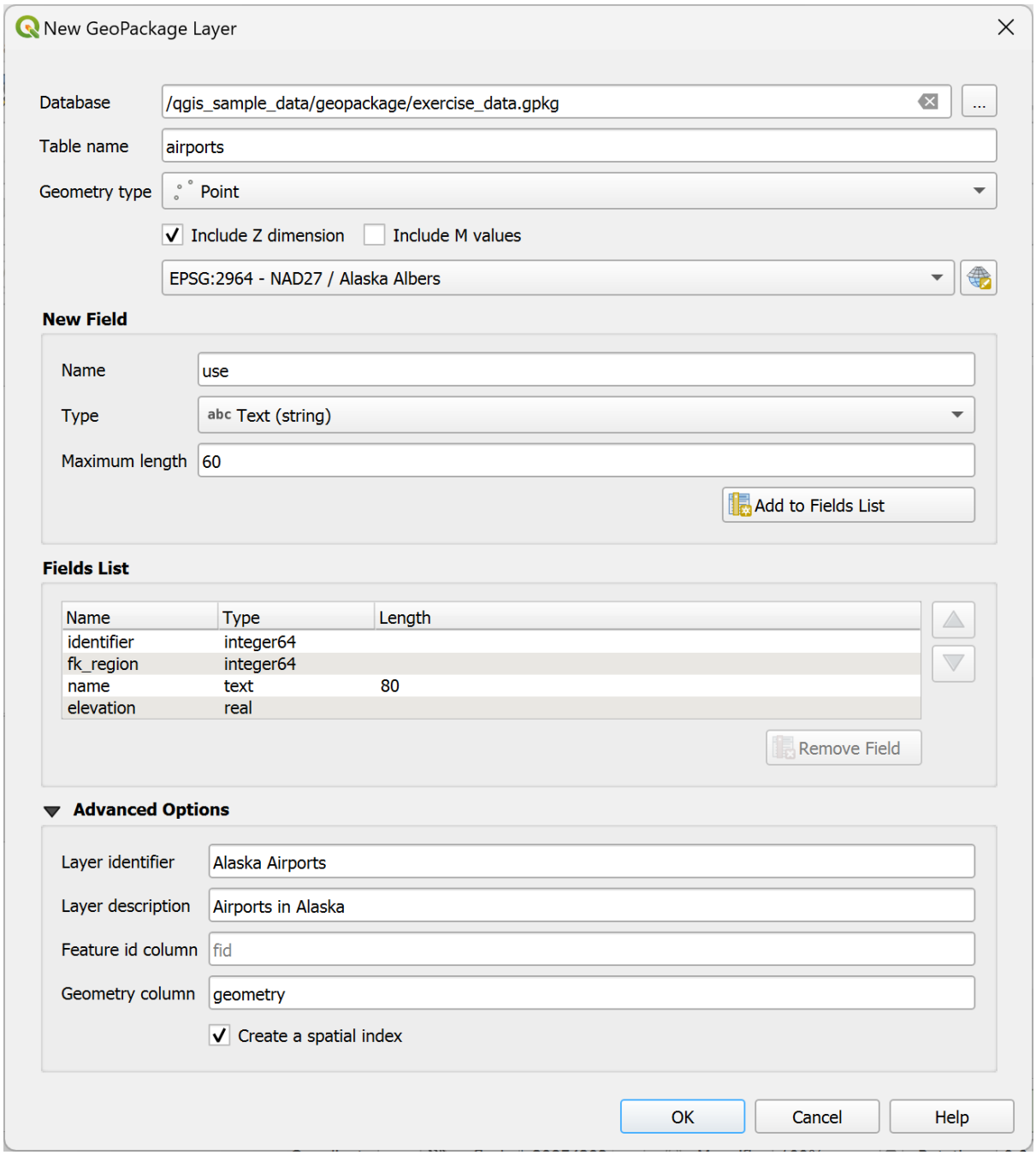


Fig. 11.25: Creating a New GeoPackage layer dialog

1. The first step is to indicate the database file location. This can be done by pressing the … button to the right of the *Database* field and select an existing GeoPackage file or create a new one. QGIS will automatically add the right extension to the name you provide.
2. Give the new layer / table a name (*Table name*)
3. Define the *Geometry type*. If not a geometryless layer, you can specify whether it should *Include Z dimension* and/or *Include M values*.
4. Specify the coordinate reference system using the button

To add fields to the layer you are creating:

1. Enter the *Name* of the field
2. Select the data *Type*. Supported types are *Text data*, *Whole number* (both integer and integer64), *Decimal number*, *Date* and *Date and time*, *Binary (BLOB)* and *Boolean*.
3. Depending on the selected data format, enter the *Maximum length* of values.
4. Click on the *Add to Fields List* button
5. Reproduce the steps above for each field you need to add
6. You can later change the fields order using the Move Up and Move Down buttons
7. Once you are happy with the attributes, click *OK*. QGIS will add the new layer to the legend, and you can edit it as described in section *Digitizing an existing layer*.

By default, when creating a GeoPackage layer, QGIS generates a *Feature id column* called `fid` which acts as the primary key of the layer. The name can be changed. The geometry field, if availabe, is named `geometry`, and you can choose to *Create a spatial index* on it. These options can be found under the *Advanced Options* together with the *Layer identifier* (short human readable name of the layer) and the *Layer description*.

Further management of GeoPackage layers can be done with the *DB Manager*.

Creating a new Shapefile layer

To create a new ESRI Shapefile format layer, press the *New Shapefile Layer…* button in the *Layer* ► *Create Layer* ► menu or from the *Data Source Manager* toolbar. The *New Shapefile Layer* dialog will be displayed as shown in Fig. 11.26.

1. Provide a path and file name using the … button next to *File name*. QGIS will automatically add the right extension to the name you provide.
2. Next, indicate the *File encoding* of the data
3. Choose the *Geometry type* of the layer: No Geometry (resulting in a `.DBF` format file), point, multipoint, line or polygon
4. Specify whether the geometry should have additional dimensions: *None*, *Z (+ M values)* or *M values*
5. Specify the coordinate reference system using the button

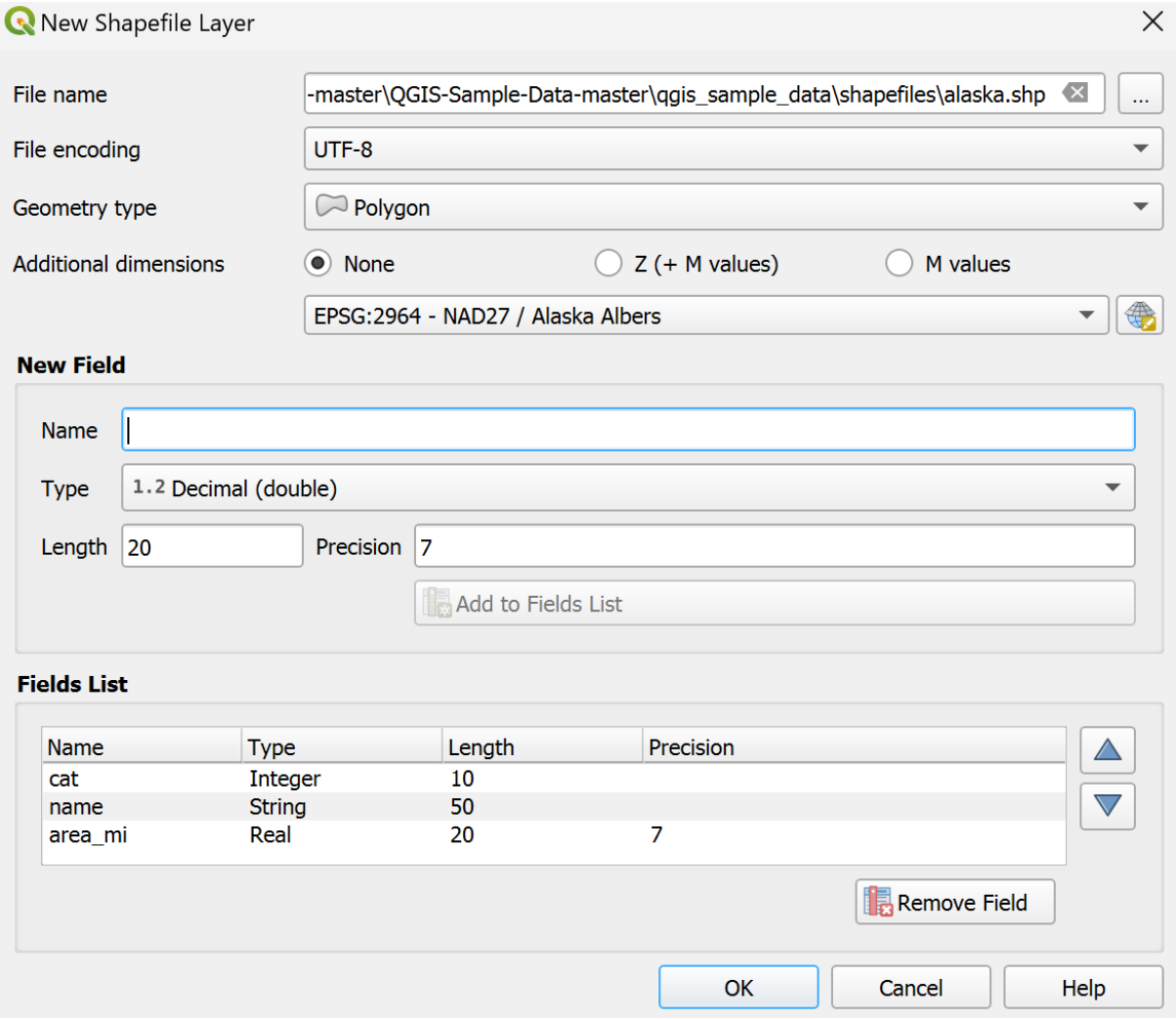


Fig. 11.26: Creating a new Shapefile layer dialog

To add fields to the layer you are creating:

1. Enter the *Name* of the field
2. Select the data *Type*. Only *Decimal number*, *Whole number*, *Text data*, *Date*, and *Boolean* data types attributes are supported.
3. Depending on the selected data format, enter the *Length* and *Precision*.
4. Click on the *Add to Fields List* button
5. Reproduce the steps above for each field you need to add
6. You can later change the fields order using the Move Up and Move Down buttons
7. Once you are happy with the attributes, click *OK*. QGIS will add the new layer to the legend, and you can edit it as described in section *Digitizing an existing layer*.

By default, a first integer `id` column is added but can be removed.

Creating a new SpatiaLite layer

To create a new SpatiaLite layer, press the *New SpatiaLite Layer…* button in the *Layer* ► *Create Layer* ► menu or from the *Data Source Manager* toolbar. The *New SpatiaLite Layer* dialog will be displayed as shown in Fig. 11.27.

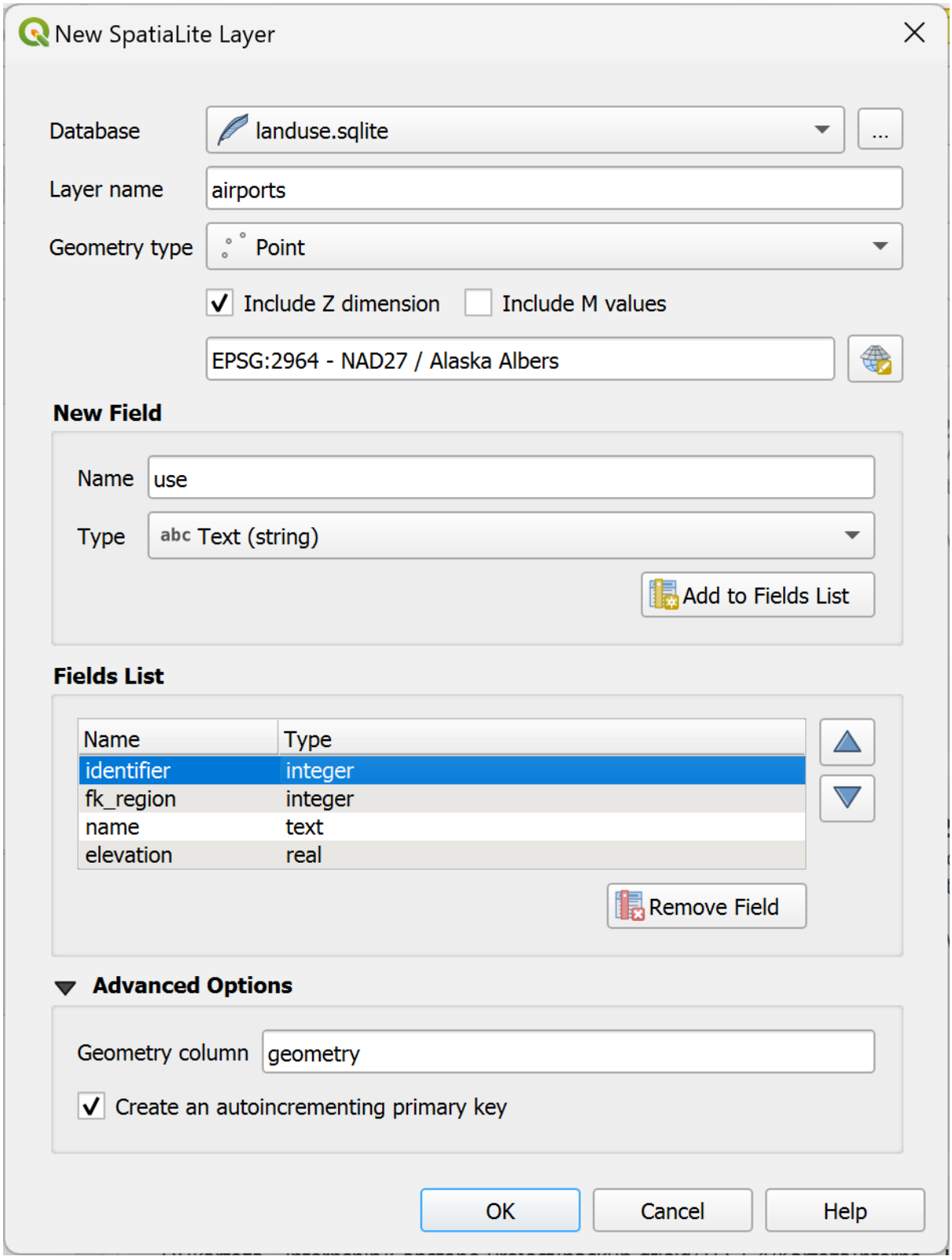


Fig. 11.27: Creating a New SpatiaLite layer dialog

1. The first step is to indicate the database file location. This can be done by pressing the … button to the right of the *Database* field and select an existing SpatiaLite file or create a new one. QGIS will automatically add the right extension to the name you provide.
2. Provide a name (*Layer name*) for the new layer
3. Define the *Geometry type*. If not a geometryless layer, you can specify whether it should *Include Z dimension* and/or *Include M values*.
4. Specify the coordinate reference system using the button.

To add fields to the layer you are creating:

1. Enter the *Name* of the field
2. Select the data *Type*. Supported types are *Text data*, *Whole number*, *Decimal number*, *Date* and *Date time*.
3. Click on the *Add to Fields List* button
4. Reproduce the steps above for each field you need to add
5. You can later change the fields order using the Move Up and Move Down buttons
6. Once you are happy with the attributes, click *OK*. QGIS will add the new layer to the legend, and you can edit it as described in section *Digitizing an existing layer*.

If desired, you can select *Create an autoincrementing primary key* under the *Advanced Options* section. You can also rename the *Geometry column* (`geometry` by default).

Further management of SpatiaLite layers can be done with *DB Manager*.

Creating a new Mesh layer

To create a new Mesh layer, press the *New Mesh Layer…* button in the *Layer* ► *Create Layer* ► menu or from the *Data Source Manager* toolbar. The *New Mesh Layer* dialog will be displayed as shown in Fig. 11.28.

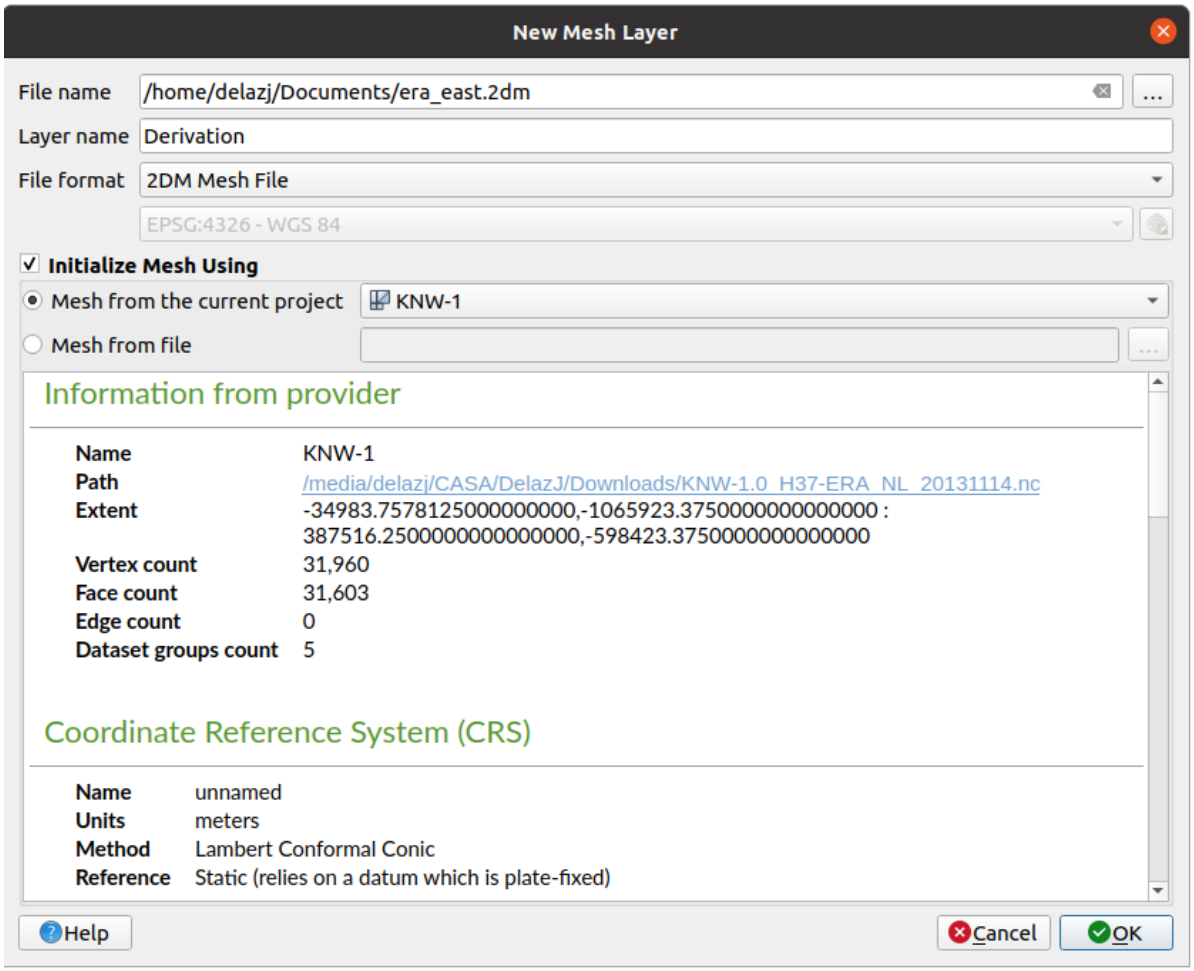


Fig. 11.28: Creating a New Mesh layer dialog

1. The first step is to indicate the mesh file location. This can be done by pressing the … button to the right of the *File name* field and select an existing mesh file or create a new one.
2. Provide a name (*Layer name*), i.e. the name the layer is displayed with in the *Layers* panel
3. Select the *File format*: currently supported mesh file formats are `2DM Mesh File (*.2dm)`, `Selafin File (*.slf)` and `UGRID (*.nc)`.
4. Indicate the *Coordinate Reference System* to assign to the dataset
5. The above steps will generate an empty layer that you can afterwards digitize vertices and add dataset groups to. It's however also possible to initialize the layer with an existing mesh layer, i.e. populate the new layer with vertices or faces from the other. To do so:
   1. Check *Initialize Mesh using*
   2. and select either a *Mesh from the current project* or *Mesh from a file*. Informations on the selected mesh file are displayed for checkup.

   Note that only the frame of the mesh layer is transferred to the new layer; their datasets are not copied.

Creating a new GPX layer

To create a new GPX file:

1. Select *Create Layer* ► *New GPX Layer…* from the *Layer* menu.
2. In the dialog, choose where to save the new file, name it and press *Save*.
3. Three new layers are added to the *Layers Panel*:
   - a point layer to digitize locations (`waypoints`) with fields storing the name, elevation, comment, description, source, url and url name
   - a line layer to digitize sequences of locations that make up a planned route (`routes`) with fields storing the name, symbol, number, comment, description, source, url, url name
   - and a line layer to track the receiver's movement over time (`tracks`) with fields storing the name, symbol, number, comment, description, source, url, url name.
4. You can now edit any of them as described in section *Digitizing an existing layer*.

Creating a new Temporary Scratch Layer

Temporary Scratch Layers are in-memory layers, meaning that they are not saved on disk and will be discarded when QGIS is closed. They can be handy for storing features you temporarily need or as intermediate layers during geoprocessing operations.

To create a new Temporary Scratch layer, choose the *New Temporary Scratch Layer…* entry in the *Layer* ► *Create Layer* ► menu or in the *Data Source Manager* toolbar. The *New Temporary Scratch Layer* dialog will be displayed as shown in Fig. 11.29. Then:

1. Provide the *Layer name*
2. Select the *Geometry type*. Here you can create a:
   - `No geometry` type layer, served as simple table,
   - `Point` or `MultiPoint` layer,
   - `LineString/CompoundCurve` or `MultiLineString/MultiCurve` layer,
   - `Polygon/CurvePolygon` or `MultiPolygon/MultiSurface` layer.
3. For geometric types, specify the dimensions of the dataset: check whether it should *Include Z dimension* and/or *Include M values*
4. Specify the coordinate reference system using the button.
5. Add fields to the layer. Note that unlike many formats, temporary layers can be created without any fields. This step is thus optional.
   1. Enter the *Name* of the field
   2. Select the data *Type*: *Text*, *Whole number*, *Decimal number*, *Boolean*, *Date*, *Time*, *Date & Time* and *Binary (BLOB)* are supported.
   3. Depending on the selected data format, enter the *Length* and *Precision*
   4. Click on the *Add to Fields List* button
   5. Repeat the steps above for each field you need to add
   6. You can later change the fields order using the Move Up and Move Down buttons
6. Once you are happy with the settings, click *OK*. QGIS will add the new layer to the *Layers* panel, and you can edit it as described in section *Digitizing an existing layer*.

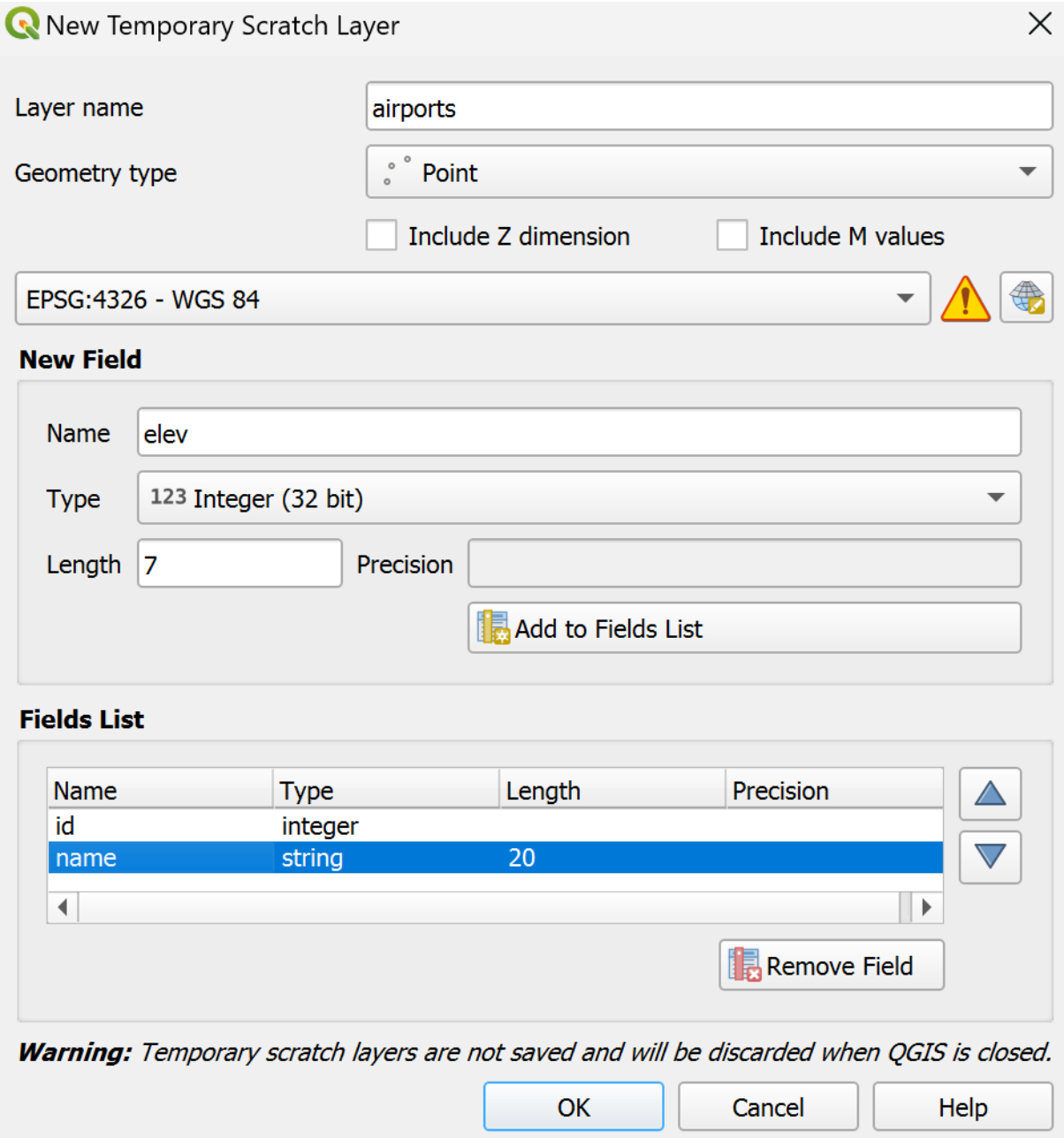


Fig. 11.29: Creating a new Temporary Scratch layer dialog

You can also create prepopulated temporary scratch layers using e.g. the clipboard (see *Creating new layers from the clipboard*) or as a result of a *Processing algorithm*.

Tip

Permanently store a memory layer on disk

To avoid data loss when closing a project with temporary scratch layers, you can save these layers to any vector format supported by QGIS:

- clicking the indicator icon next to the layer;
- selecting the *Make permanent* entry in the layer contextual menu;
- using the *Export* ► entry from the contextual menu or the *Layer* ► *Save As...* menu.

Each of these commands opens the *Save Vector Layer as* dialog described in the *Creating new layers from an existing layer* section and the saved file replaces the temporary one in the *Layers* panel.

11.2.2 Creating new layers from an existing layer

Layers (raster, vector and point cloud) can be saved in a different format and/or reprojected to a different coordinate reference system (CRS) using the *Layer* ► *Save As…* menu or right-clicking on the layer in the *Layers panel* and selecting:

- *Export* ► *Save As…* for raster and point cloud layers
- *Export* ► *Save Features As…* or *Export* ► *Save Selected Features As…* for vector layers.
- Drag and drop the layer from the layer tree to the PostgreSQL entry in the *Browser Panel*. Note that you must have a PostgreSQL connection in the *Browser Panel*.

Common parameters

The *Save Layer as…* dialog shows several parameters to change the behavior when saving the layer. Among the common parameters for raster and vector are:

- *File name*: the location of the file on the disk. It can refer to the output layer or to a container that stores the layer (for example database-like formats such as GeoPackage, SpatiaLite or Open Document Spreadsheets).
- *CRS*: can be changed to reproject the data
- *Extent*: restricts the extent of the input that is to be exported using the *extent_selector* widget
- *Add saved file to map*: to add the new layer to the canvas

However, some parameters are specific to certain formats:

Raster specific parameters

Depending on the format of export, some of these options may not be available:

- *Output mode* (it can be **raw data** or **rendered image**)
- *Format*: exports to any raster format GDAL can write to, such as GeoTiff, GeoPackage, MBTiles, Geospatial PDF, SAGA GIS Binary Grid, Intergraph Raster, ESRI .hdr Labelled…
- *Resolution*
- *Create Options*: use advanced options (file compression, block sizes, colorimetry…) when generating files, either from the *predefined create profiles* related to the output format or by setting each parameter.
- *Pyramids* creation
- *VRT Tiles* in case you opted to  *Create VRT*
- *No data values*

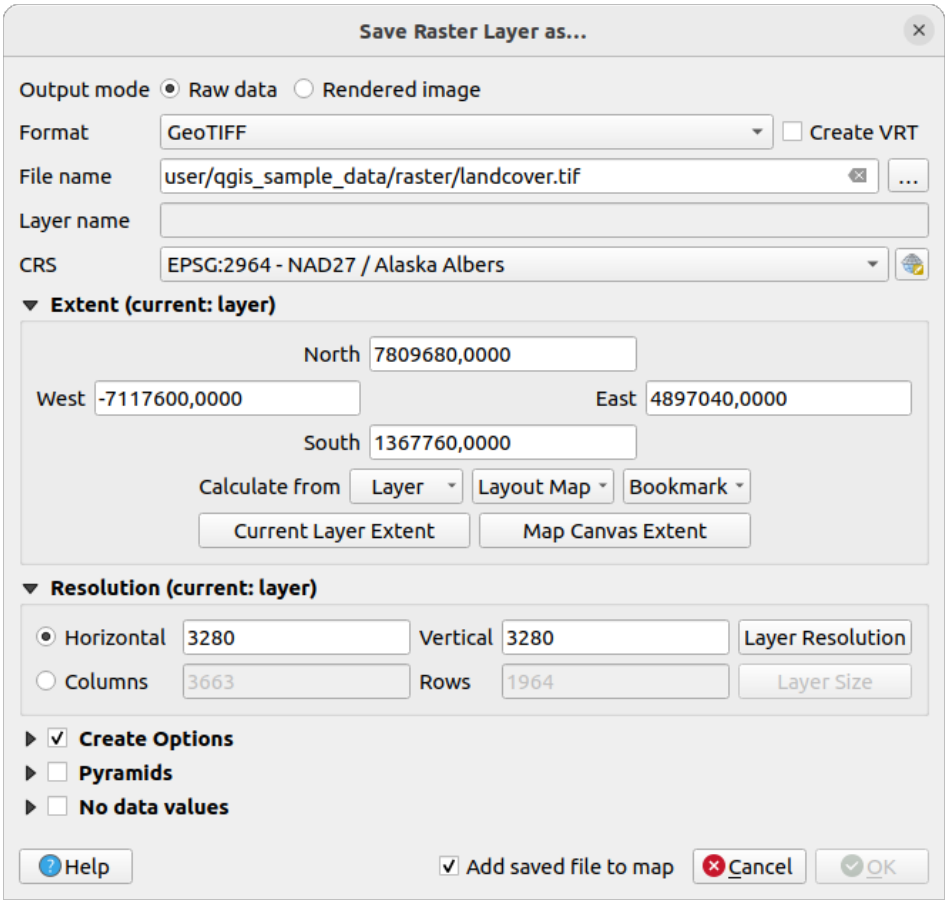


Fig. 11.30: Saving as a new raster layer

Vector specific parameters

Depending on the format of export, some of these options may be available:

- *Format*: exports to any vector format GDAL can write to, such as GeoPackage, GML, ESRI Shapefile, AutoCAD DXF, ESRI FileGDB, Mapinfo TAB or MIF, SpatiaLite, CSV, KML, ODS, …
- *Layer name*: available when the *File name* refers to a container-like format, this entry represents the output layer.
- *Encoding*
- *Save only selected features*
- *Select fields to export and their export options*: provides means to export fields with custom names and *form widget* settings:
  - Check rows under the *Name* column to choose fields to keep in the output layer, or press *Select All* or *Deselect All* buttons
  - Toggle the *Use aliases for exported name* checkbox to populate the *Export name* column with corresponding field aliases or reset to the original field name. Double-clicking a cell will also edit the name.
  - Depending on whether attribute form custom widgets are in use, you can *Replace all selected raw field values by displayed values*. E.g. if a `value map` widget is applied to a field, the output layer will contain the description values instead of the original values. The replacement can also be done on a field by field basis, in the *Replace with displayed values* column.
- *Persist layer metadata*: ensures that any layer *metadata* present in the source layer will be copied and stored:
  - in the newly created layer, if the output is of GeoPackage format

  - as a `.qmd` file along with the output layer, for other formats. Note that file-based formats supporting more than one dataset (e.g. SpatiaLite, DXF,…) may have unintended behavior.

- *Symbology export*: can be used mainly for DXF export and for all file formats who manage OGR feature styles (see note below) as DXF, KML, tab file formats:
  - **No symbology**: default style of the application that reads the data
  - **Feature symbology**: save style with OGR Feature Styles (see note below)
  - **Symbol Layer symbology**: save with OGR Feature Styles (see note below) but export the same geometry multiple times if there are multiple symbology symbol layers used
  - A **Scale** value can be applied to the latest options

Note

OGR Feature Styles are a way to store style directly in the data as a hidden attribute. Only some formats can handle this kind of information. KML, DXF and TAB file formats are such formats. For advanced details, you can read the OGR Feature Styles specification document.

- *Geometry*: you can configure the geometry capabilities of the output layer
  - *geometry type*: keeps the original geometry of the features when set to **Automatic**, otherwise removes or overrides it with any type. You can add an empty geometry column to an attribute table and remove the geometry column of a spatial layer.
  - *Force multi-type*: forces creation of multi-geometry features in the layer.
  - *Include z-dimension* to geometries.

Tip

Overriding layer geometry type makes it possible to do things like save a geometryless table (e.g. `.csv` file) into a shapefile WITH any type of geometry (point, line, polygon), so that geometries can then be manually added to rows with the  Add Part tool.

- *Datasource Options*, *Layer Options* or *Custom Options* which allow you to configure advanced parameters depending on the output format. Some are described in *Exploring Data Formats and Fields* but for full details, see the GDAL driver documentation. Each file format has its own custom parameters, e.g. for the `GeoJSON` format have a look at the GDAL GeoJSON documentation.

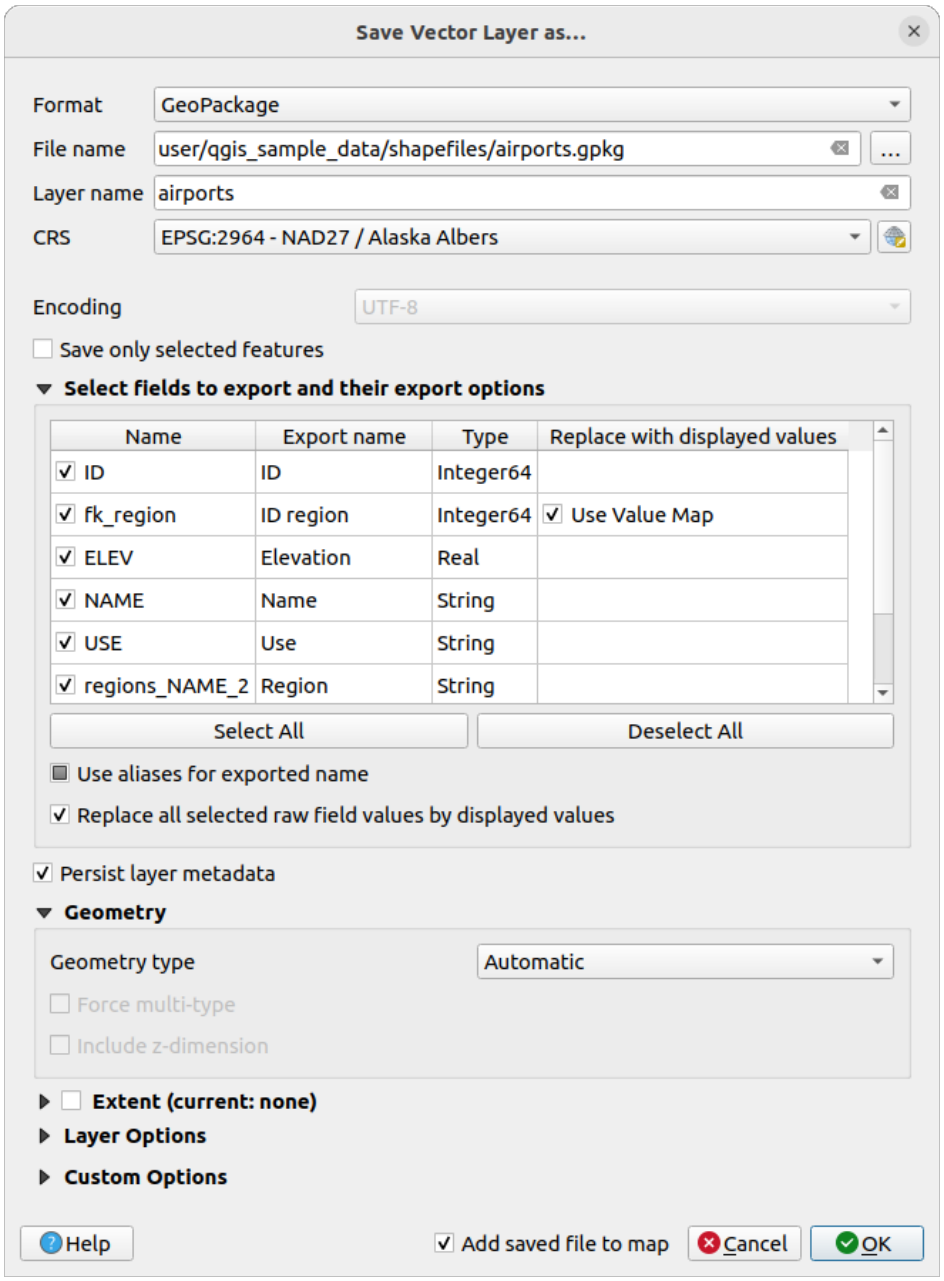


Fig. 11.31: Saving as a new vector layer

When saving a vector layer into an existing file, depending on the capabilities of the output format (Geopackage, SpatiaLite, FileGDB…), the user can decide whether to:

- overwrite the whole file
- overwrite only the target layer (the layer name is configurable)
- append features to the existing target layer
- append features, add new fields if there are any.

For formats like ESRI Shapefile, MapInfo .tab, feature append is also available.

Point Cloud specific parameters

Similar to raster and vector layers, point cloud layers can be saved in a different format and/or reprojected to a different coordinate reference system (CRS). This allows you to export a point cloud layer to vector or point cloud formats. Current supported formats are: Temporary scratch (memory layer), GeoPackage, ESRI Shapefile, DXF and LAS/LAZ point cloud. In addition to the common parameters listed above, exporting point cloud layers includes the following options:

- *Filter by Polygon Layer*: Allows you to filter the point cloud data based on a polygon layer.
- *Elevation Range*: Enables filtering of the point cloud data based on a specified Z range.
- *Limit number of points*: Provides an option to limit the number of points exported from the point cloud layer.

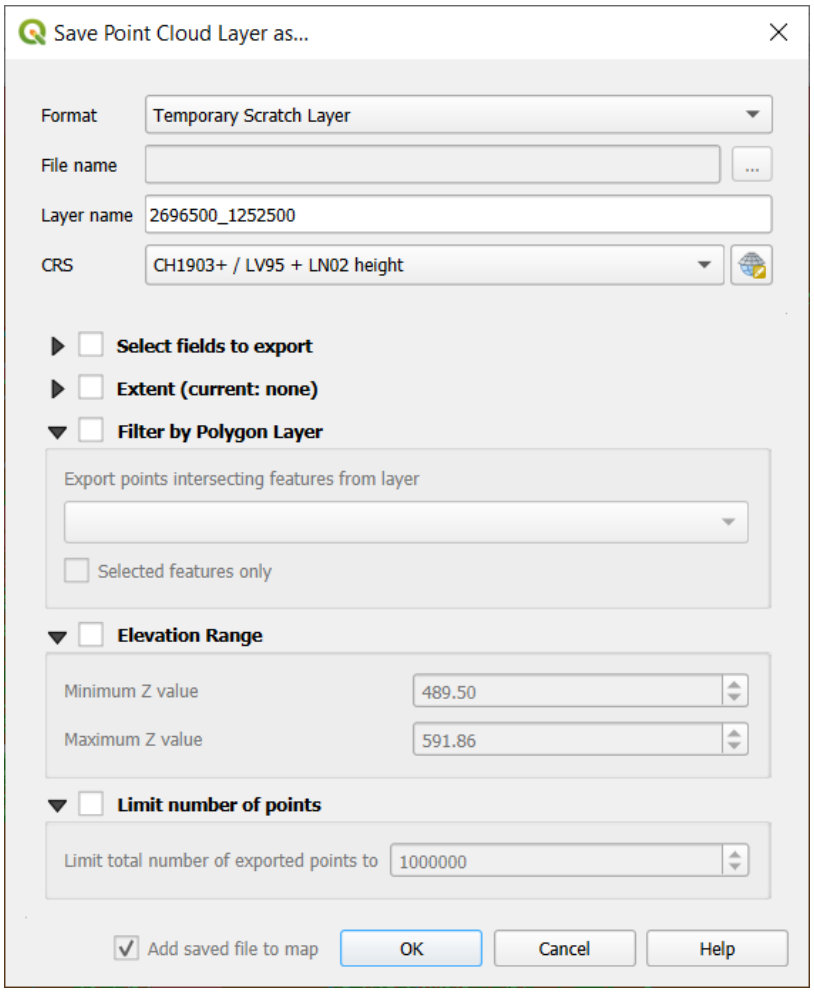


Fig. 11.32: Saving a point cloud layer as a new layer

11.2.3 Creating new DXF files

Besides the *Save As...* dialog which provides options to export a single layer to another format, including `*.DXF`, QGIS provides another tool to export multiple layers as a single DXF layer. It's accessible in the *Project* ► *Import/Export* ► *Export Project to DXF...* menu.

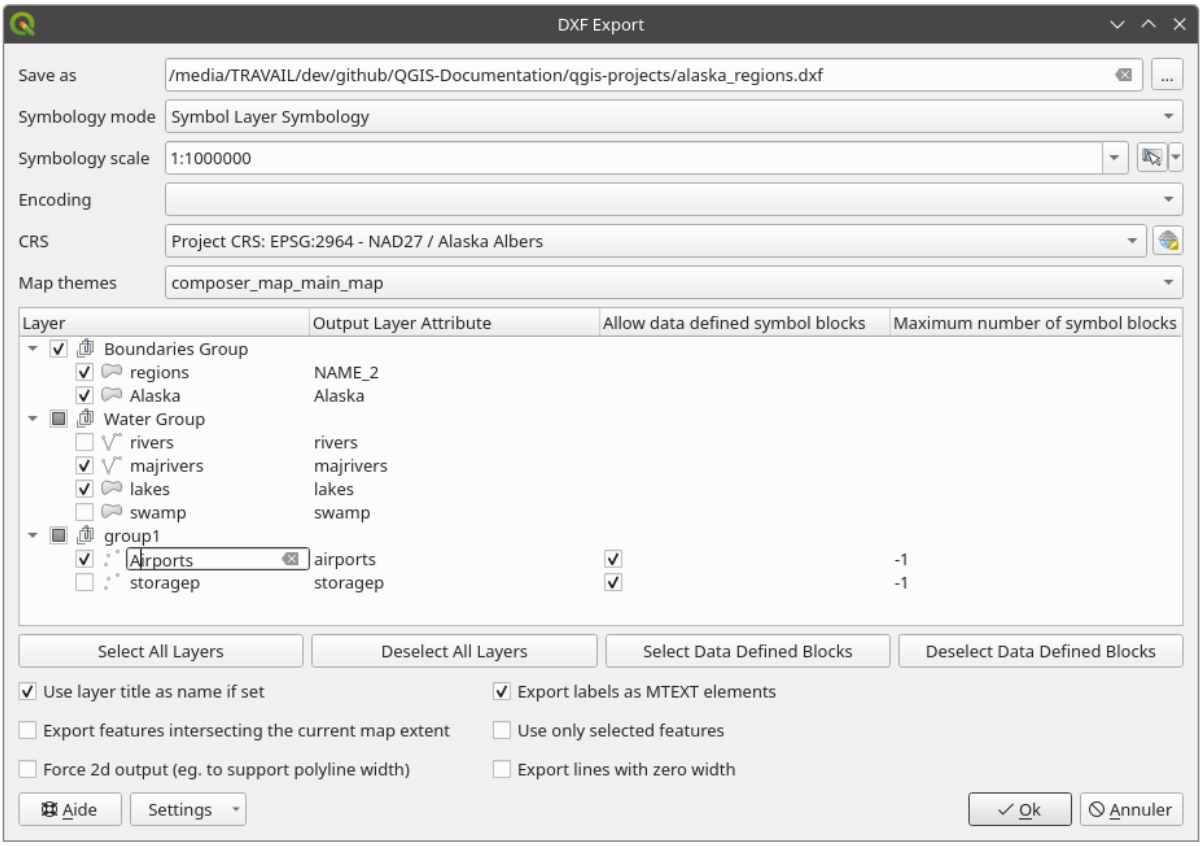


Fig. 11.33: Exporting a project to DXF dialog

In the *DXF Export* dialog:

1. Provide the destination file.
2. Choose the symbology mode and scale (see the *OGR Feature Styles* note), if applicable.
3. Select the data *Encoding*.
4. Select the *CRS* to apply: the selected layers will be reprojected to the given CRS.
5. Select the layers to include in the DXF files either by checking them in the table widget or automatically picking them from an existing *map theme*. The *Select All* and *Deselect All* buttons can help to quickly set the data to export.

   For each layer, you can:

   - Override the output layer name without altering the original project layer. For this, click on the *Layer* name in the dialog and write the output name to use.
   - *Output layer attribute*: Choose whether to export all the features in a single DXF layer or rely on a field whose values are used to split the features into layers in the DXF output. In the latter case, each layer will take its name from the corresponding field value.
   - *Allow data defined symbol blocks*:

- *Maximum number of symbol blocks*: creates symbol blocks up to the specified limit, starting with the ones containing the highest number of references. The other symbols are written as they are. `-1` means no limitation.

Optionally, you can also choose to:

- ☐ *Use the layer title as name if set* instead of the layer name itself: the title is taken from the *metadata* or *server* properties of the layer;
- ☐ *Export features intersecting the current map extent*;
- ☐ *Force 2d output (eg. to support polyline width)*;
- ☑ *Export label as MTEXT elements* or TEXT elements;
- ☐ *Use only selected features*;
- ☐ *Export lines with zero width*: all the lines are exported with minimal width `0` (hairline) if enabled. This helps keep the lines minimal in the file regardless of the zoom level, and can be handy for doing further CAD-editing with the exported dxf, especially if there are many features next to each other on the map.

Note

The precedence for defining the output layer name is as follows:

1. The field value from *Output layer attribute*
2. The overridden name in *Layer* column
3. The *Use the layer title as name if set* option
4. The layer name

Current settings defined in the *DXF Export* dialog may be stored in an XML file for reusing them in other sessions. For this, the *Settings* combo box has two options: *Load Settings from File…* and *Save Settings to File…*.

11.2.4 Creating new layers from the clipboard

Features that are on the clipboard can be pasted into a new layer. To do this, Select some features, copy them to the clipboard, and then paste them into a new layer using *Edit* ► *Paste Features as* ► and choosing:

- *New Vector Layer…*: the *Save vector layer as…* dialog appears (see *Creating new layers from an existing layer* for parameters)
- or *Temporary Scratch Layer…*: you need to provide a name for the layer

A new layer, filled with selected features and their attributes is created (and added to map canvas).

Note

Creating layers from the clipboard is possible with features selected and copied within QGIS as well as features from another application, as long as their geometries are defined using well-known text (WKT).

11.2.5 Creating SQL Query Layers

Beside loading an entire layer in a project or creating new layers from scratch or pasted features, you can also load layers generated on the fly from other layer(s). They are the result of a more or less advanced filter using SQL language, applied to layers regardless of their data provider or their availability in the active project. Depending on the provider, one or more layers can be used to write the query. The generated layer remains dependent on the layer(s) involved in the query and is loaded with the Filter icon next to it.

This feature is accessible:

- From the *Browser* panel, right-click on a supported data (plain layer, database connection, schema or table) and select *Execute SQL…* entry in the contextual menu.
- From the *Layers* panel, select a loaded layer, right-click and select *Execute SQL…* entry in the contextual menu.

This opens a window with a central text box widget where you can write SQL queries.

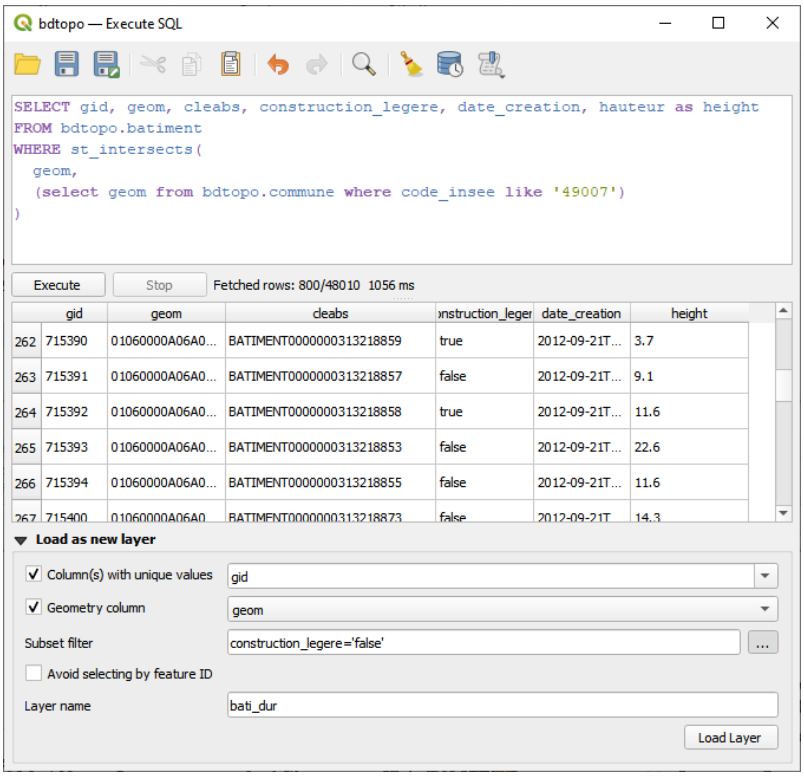


Fig. 11.34: Executing SQL queries in the Execute SQL window

At the top of the dialog, a toolbar provides a set of tools to create, store and manipulate your queries:

- Open Queries…: fills the text editor widget with contents from an existing `.sql` file
- Save Queries… and Save Queries as… help you store the written query to a `.sql` file
- Statements of the query can be adjusted using the Cut, Copy and Paste buttons. Likewise, you can Undo or Redo your changes.
- The Find & Replace enables, at the bottom of the dialog, a widget allowing to look for a particular string in your SQL code. The search can be case sensitive, affect partial or whole word, rely on a regular expression. It is then possible to navigate through the found strings, replacing them one by one or all in a row.
- Use Clear to wipe the text editor.

- The  History button opens a dialog storing previously run queries. More at *Query History*.
- As previously mentioned, queries can be saved as an `.sql` file stored on disk. Using the  Store Current Query button, they can also be stored:
  - In the active *User Profile*, in the associated `QGIS3.ini` file, thus accessible in subsequent projects
  - or as part of the *Current Project*.

  Clicking an entry from the stored queries drop-down menu inserts that query in the expression being written. The stored entry can also be deleted, from the drop-down menu.

Running and loading queries as layer

In the central part of the *Execute SQL* dialog, you build your query using the SQL syntax supported by the underlying provider (e.g., OGR, GeoPackage, PostgreSQL).

By default, if opened from a layer entry, a sample SQL query is provided. Editing tools to select, cut, copy, paste, undo and redo are available as well from the contextual menu.

> **Tip**
>
> **Finding the right SQL syntax for your dataset**
>
> To find the appropriate SQL dialect for your dataset, press the … button next to the *Subset filter* option in the bottom part of the dialog.

When ready, pressing the *Execute* button below the text area will run the query. It is possible to highlight a portion of the SQL to only execute that portion when pressing `Ctrl+R` or clicking the *Execute selection* button. Use the *Stop* button to abort the execution.

A successful query execution will display a table at the bottom of the dialog with returned features. You can select specific cells in the result set. Use the `Ctrl+C` shortcut to copy the selected cells to the clipboard. The copied data is available as a formatted table. This allows you to paste the data into other applications, such as spreadsheet where it will show up as a table.

The returned table can be loaded in QGIS expanding the *Load as new layer* group and configuring parameters (their availability depends on the layer provider):

- *Column(s) with unique values* to indicate primary key of the data,
- *Geometry column*: check the box to load the layer as a spatial one, and indicate the geometry field name
- *Subset filter*: allows to filter the results using a `WHERE` clause. It can be written in the text box, or built using the *Query builder* after you pressed …. Make sure to use fields that are available in the SQL layer.
- *Avoid selecting by feature ID*
- *Layer name* in the project

At any time, in the *Layers panel*, you can adjust the output layer by right-clicking and selecting *Update SQL expression*…. The *Update SQL* dialog opens, prefilled with the applied query that you can edit as you wish. Once ready, press *Update layer* and the layer will be modified in-place.

Query History

The *Query History* dialog is accessible when pressing History button in the *Execute SQL* dialog or from the *Database* ► *Query History…* menu. It displays all the previously run queries, sorted by date and provider type, that you can preview and reuse.

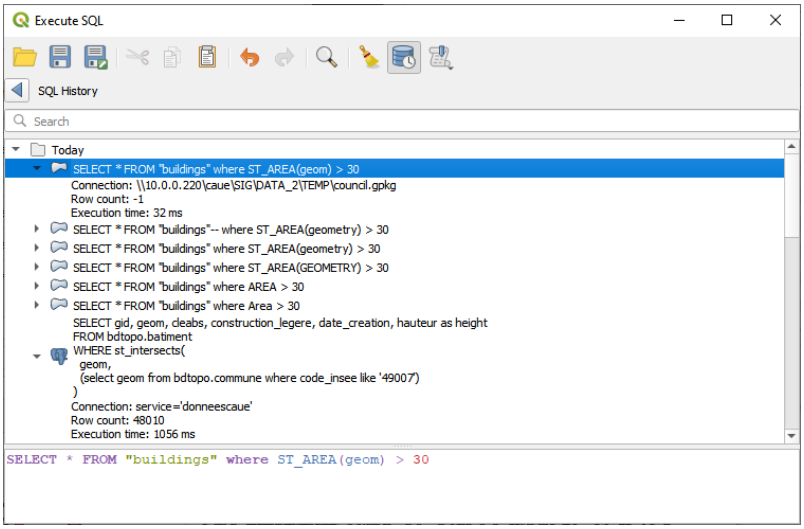


Fig. 11.35: History of the executed SQL queries

Hover over an entry and the full query is displayed over, as a tooltip. Right-click and you can either:

- *Load SQL Command…*, loads the target command into the *Execute SQL* dialog, replacing any existing query. It is the same as double clicking the entry.
- *Copy SQL Command* and paste it wherever you want

When a query is selected, the full query is displayed in the lower part of the dialog. You can interact with the text, copying all or part of it.

11.2.6 Creating virtual layers

A virtual layer is a special kind of vector layer. It allows you to define a layer as the result of an SQL query involving any number of other vector layers that QGIS is able to open. Virtual layers do not carry data by themselves and can be seen as views.

To create a virtual layer, open the virtual layer creation dialog by:

- choosing the *Add/Edit Virtual Layer* entry in the *Layer* ► *Add Layer* ► menu;
- enabling the *Add Virtual Layer* tab in the *Data Source Manager* dialog;
- or using the *DB Manager* dialog tree.

The dialog allows you to specify a *Layer name* and an SQL *Query*. The query can use the name (or id) of loaded vector layers as tables, as well as their field names as columns.

For example, if you have a layer called `airports`, you can create a new virtual layer called `public_airports` with an SQL query like:

```
SELECT *
FROM airports
WHERE USE = "Civilian/Public"
```

The SQL query will be executed, regardless of the underlying provider of the `airports` layer, even if this provider does not directly support SQL queries.

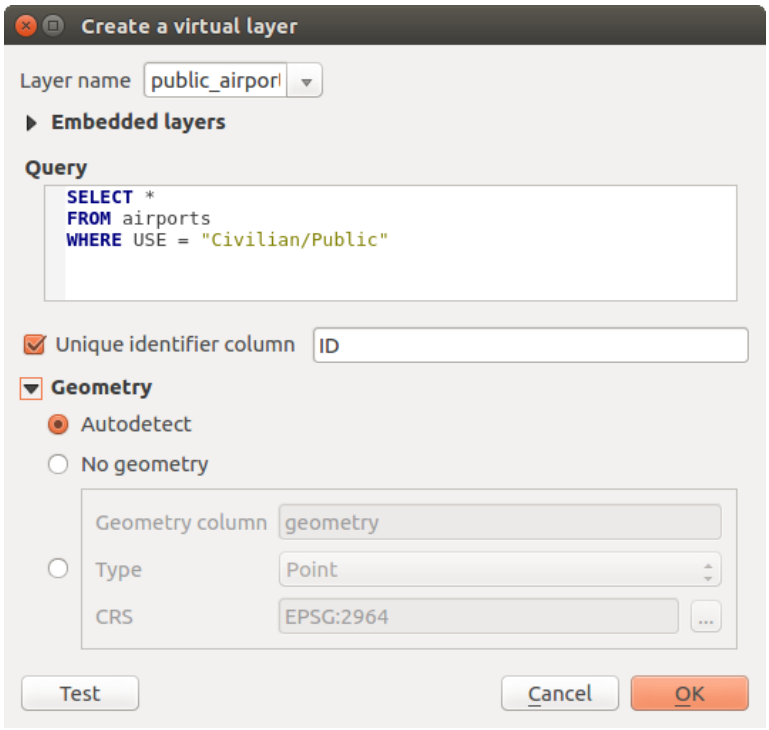


Fig. 11.36: Create virtual layers dialog

Joins and complex queries can also be created, for example, to join airports and country information:

```
SELECT airports.*, country.population
FROM airports
JOIN country
ON airports.country = country.name
```

Note

It's also possible to create virtual layers using the SQL window of *DB Manager Plugin*.

Embedding layers for use in queries

Besides the vector layers available in the map canvas, the user can add layers to the *Embedded layers* list, which can be used in queries without the need to have them showing in the map canvas or Layers panel.

To embed a layer, click *Add* and provide the *Local name*, *Provider*, *Encoding* and the path to the *Source*.

The *Import* button allows adding layers in the map canvas into the Embedded layers list. Those layers can then be removed from the Layers panel without breaking existent queries.

Supported query language

The underlying engine uses SQLite and SpatiaLite to operate.

It means you can use all of the SQL your local installation of SQLite understands.

Functions from SQLite and spatial functions from SpatiaLite can also be used in a virtual layer query. For instance, creating a point layer out of an attribute-only layer can be done with a query similar to:

```
SELECT id, MakePoint(x, y, 4326) as geometry
FROM coordinates
```

Functions of QGIS expressions can also be used in a virtual layer query.

To refer the geometry column of a layer, use the name `geometry`.

Contrary to a pure SQL query, all the fields of a virtual layer query must be named. Don't forget to use the `as` keyword to name your columns if they are the result of a computation or a function call.

Performance issues

With default parameters, the virtual layer engine will try its best to detect the type of the different columns of the query, including the type of the geometry column if one is present.

This is done by introspecting the query when possible or by fetching the first row of the query (LIMIT 1) as a last resort. Fetching the first row of the result just to create the layer may be undesirable for performance reasons.

The creation dialog parameters:

- *Unique identifier column*: specifies a field of the query that represents unique integer values that QGIS can use as row identifiers. By default, an autoincrementing integer value is used. Defining a unique identifier column speeds up the selection of rows by id.
- *No geometry*: forces the virtual layer to ignore any geometry field. The resulting layer is an attribute-only layer.
- Geometry *Column*: specifies the name of the geometry column.
- Geometry *Type*: specifies the type of the geometry.
- Geometry *CRS*: specifies the coordinate reference system of the virtual layer.

Special comments

The virtual layer engine tries to determine the type of each column of the query. If it fails, the first row of the query is fetched to determine column types.

The type of a particular column can be specified directly in the query by using some special comments.

The syntax is the following: `/*:type*/`. It has to be placed just after the name of a column. `type` can be either `int` for integers, `real` for floating point numbers or `text`.

For instance:

```
SELECT id+1 as nid /*:int*/
FROM table
```

The type and coordinate reference system of the geometry column can also be set thanks to special comments with the following syntax `/*:gtype:srid*/` where `gtype` is the geometry type (`point`, `linestring`, `polygon`, `multipoint`, `multilinestring` or `multipolygon`) and `srid` an integer representing the EPSG code of a coordinate reference system.

Use of indexes

When requesting a layer through a virtual layer, the source layer indices will be used in the following ways:

- if an = predicate is used on the primary key column of the layer, the underlying data provider will be asked for a particular id (FilterFid)
- for any other predicates (>, <=, !=, etc.) or on a column without a primary key, a request built from an expression will be used to request the underlying vector data provider. It means indexes may be used on database providers if they exist.

A specific syntax exists to handle spatial predicates in requests and triggers the use of a spatial index: a hidden column named `_search_frame_` exists for each virtual layer. This column can be compared for equality to a bounding box. Example:

```
SELECT *
FROM vtab
WHERE _search_frame_=BuildMbr(-2.10,49.38,-1.3,49.99,4326)
```

Spatial binary predicates like `ST_Intersects` are sped up significantly when used in conjunction with this spatial index syntax.

11.3 Georeferencer

The  Georeferencer is a tool for aligning unreferenced raster or vector layers to known coordinate systems using Ground Control Points (GCPs). It supports exporting transformed rasters (GeoTIFF) and vectors (shapefiles, geopackages, etc.) or writing accompanying world files (for raster only). The basic approach to georeferencing a layer is to locate points on it for which you can accurately determine coordinates.

Features

Table 11.1: Georeferencer Tools

Icon	Purpose	Icon	Purpose
	Open raster		Open vector
	Start georeferencing		Generate GDAL Script
	Load GCP Points		Save GCP Points As
	Transformation settings		
	Add GCP Point		Delete GCP Point
	Move GCP Point		Snapping Type
	Show Advanced Digitizing Dock		Pan
	Zoom In		Zoom Out
	Zoom To Layer		Zoom Last
	Zoom Next		Link Georeferencer to QGIS
	Link QGIS to Georeferencer		Full histogram stretch
	Local histogram stretch		

11.3.1 Georeferencing raster layer

As X and Y coordinates (DMS (dd mm ss.ss), DD (dd.dd) or projected coordinates (mmmm.mm)), which correspond with the selected point on the image, two alternative procedures can be used:

- The raster itself sometimes provides crosses with coordinates "written" on the image. In this case, you can enter the coordinates manually.
- Using already georeferenced layers. This can be either vector or raster data that contain the same objects/features that you have on the image that you want to georeference and with the projection that you want for your image. In this case, you can enter the coordinates by clicking on the reference dataset loaded in the QGIS map canvas.

The usual procedure for georeferencing an image involves selecting multiple points on the raster, specifying their coordinates, and choosing a relevant transformation type. Based on the input parameters and data, the Georeferencer will compute the world file parameters. The more coordinates you provide, the better the result will be.

The first step is to start QGIS and click on *Layer* ► *Georeferencer*, which appears in the QGIS menu bar. The Georeferencer dialog appears as shown in Fig. 11.37.

For this example, we are using a topo sheet of South Dakota from SDGS. It can later be visualized together with the data from the GRASS `spearfish60` location. You can download the topo sheet here: https://grass.osgeo.org/sampledata/spearfish_toposheet.tar.gz.

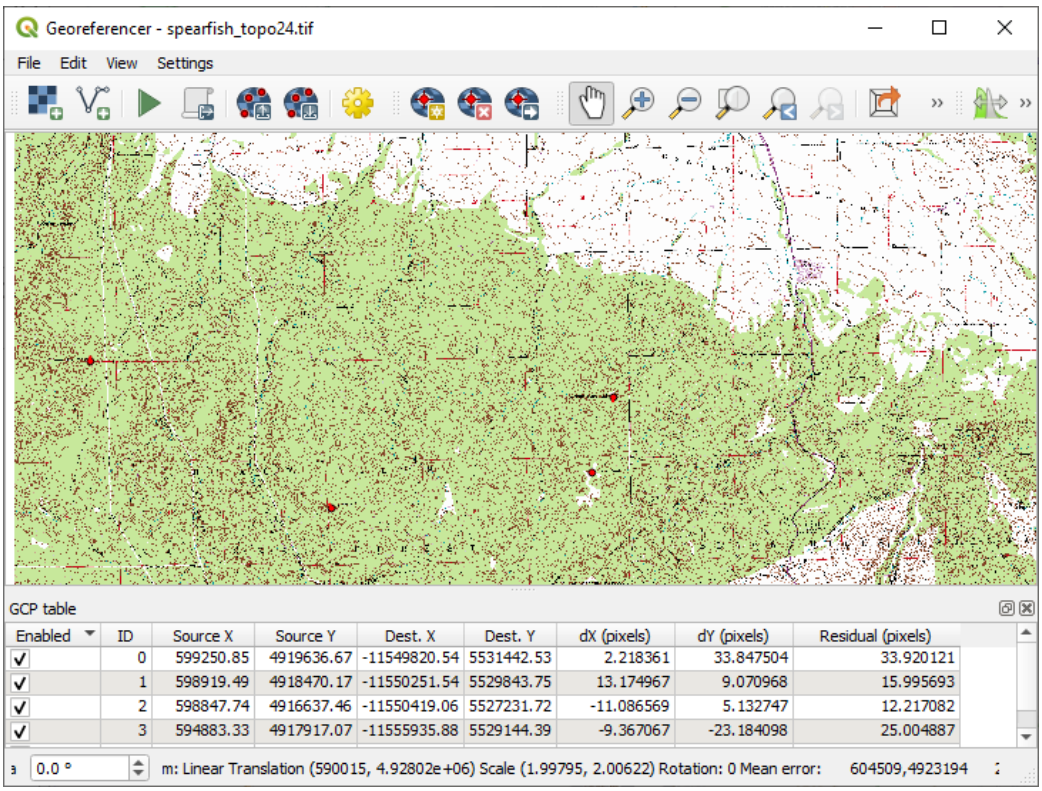


Fig. 11.37: Georeferencer Dialog

Entering ground control points (GCPs)

1. To start georeferencing an unreferenced raster, we must load it using the button. The raster will show up in the main working area of the dialog. Once the raster is loaded, we can start to enter reference points.

2. Using the Add GCP Point button, add points to the main working area and enter their coordinates (see Figure Fig. 11.38). For this procedure you have the following options:

   - Click on a point in the raster image and enter the X and Y coordinates manually, along with the CRS of the point.
   - Click on a point in the raster image and choose the From map canvas button to add the X and Y coordinates with the help of a georeferenced map already loaded in the QGIS map canvas. The CRS will be set automatically.
   - When entering GCPs from the main map canvas, you have the option to hide the georeferencer window while selecting points from the main canvas. If the *Automatically hide georeferencer window* checkbox is ticked, after clicking *From Map Canvas*, the main georeferencer window will be hidden until a point is added on the map canvas. The *Enter Map Coordinates* dialog will remain open. If the box is unchecked, both windows will remain open while selecting a point on the map canvas. This option only takes effect when the georeferencer window is not docked in the main interface.

3. Continue entering points. You should have at least four points, and the more coordinates you can provide, the better the result will be. There are additional tools for zooming and panning the working area in order to locate a relevant set of GCP points.

> **Tip**
>
> To avoid constant switching between Pan, Add GCP point and Move GCP point buttons, you may use the keyboard arrow keys for moving and the mouse wheel for scaling the georeferenced map conveniently.

4. After you provide a few points, you can use the Link QGIS to Georeferencer and/or Link Georeferencer to QGIS buttons that will adjust, respectively, the map extent of the main QGIS window to the present view in Georeferencer and/or vice versa.

5. With the tool, you can move the GCPs in both the canvas and the georeferencing window, if you need to correct them.

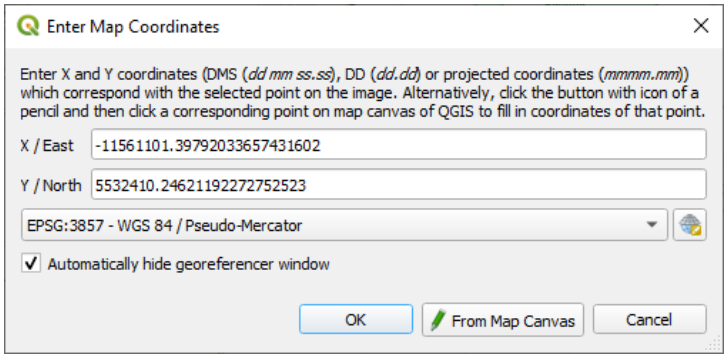


Fig. 11.38: Add points to the raster image

The points that are added to the map will be stored in a separate text file (`[filename].points`) usually together with the raster image. This allows us to reopen the Georeferencer at a later date and add new points or delete existing

ones to optimize the result. The points file contains values of the form: `mapX, mapY, pixelX, pixelY`. You can use the Load GCP points and Save GCP points as buttons to manage the files.

Defining the transformation settings

After you have added your GCPs to the raster image, you need to define the transformation settings for the georeferencing process.

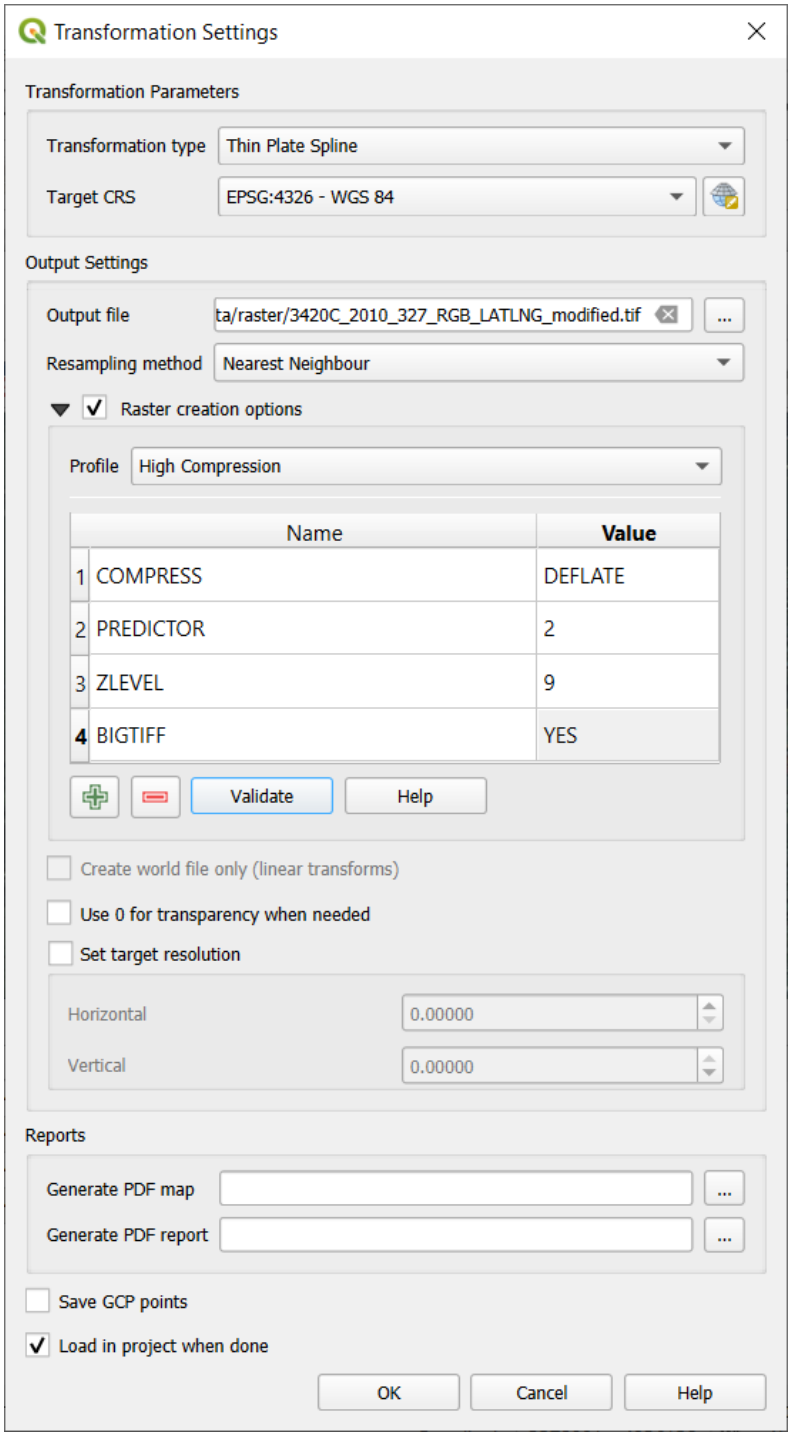


Fig. 11.39: Defining the georeferencer transformation settings

Available Transformation algorithms

A number of transformation algorithms are available, dependent on the type and quality of input data, the nature and amount of geometric distortion that you are willing to introduce to the final result, and the number of ground control points (GCPs).

Currently, the following *Transformation types* are available:

- The **Linear** algorithm is used to create a world file and is different from the other algorithms, as it does not actually transform the raster pixels. It allows positioning (translating) the image and uniform scaling, but no rotation or other transformations. It is the most suitable if your image is a good quality raster map, in a known CRS, but is just missing georeferencing information. At least 2 GCPs are needed.
- The **Helmert** transformation also allows rotation. It is particularly useful if your raster is a good quality local map or orthorectified aerial image, but not aligned with the grid bearing in your CRS. At least 2 GCPs are needed.
- The **Polynomial 1** algorithm allows a more general affine transformation, in particular also a uniform shear. Straight lines remain straight (i.e., collinear points stay collinear) and parallel lines remain parallel. This is particularly useful for georeferencing data cartograms, which may have been plotted (or data collected) with different ground pixel sizes in different directions. At least 3 GCP's are required.
- The **Polynomial** algorithms 2-3 use more general 2nd or 3rd degree polynomials instead of just affine transformation. This allows them to account for curvature or other systematic warping of the image, for instance photographed maps with curving edges. At least 6 (respectively 10) GCP's are required. Angles and local scale are not preserved or treated uniformly across the image. In particular, straight lines may become curved, and there may be significant distortion introduced at the edges or far from any GCPs arising from extrapolating the data-fitted polynomials too far.
- The **Projective** algorithm generalizes Polynomial 1 in a different way, allowing transformations representing a central projection between 2 non-parallel planes, the image and the map canvas. Straight lines stay straight, but parallelism is not preserved and scale across the image varies consistently with the change in perspective. This transformation type is most useful for georeferencing angled photographs (rather than flat scans) of good quality maps, or oblique aerial images. A minimum of 4 GCPs is required.
- Finally, the **Thin Plate Spline** (TPS) algorithm "rubber sheets" the raster using multiple local polynomials to match the GCPs specified, with overall surface curvature minimized. Areas away from GCPs will be moved around in the output to accommodate the GCP matching, but will otherwise be minimally locally deformed. TPS is most useful for georeferencing damaged, deformed, or otherwise slightly inaccurate maps, or poorly orthorectified aerials. It is also useful for approximately georeferencing and implicitly reprojecting maps with unknown projection type or parameters, but where a regular grid or dense set of ad-hoc GCPs can be matched with a reference map layer. It technically requires a minimum of 10 GCPs, but usually more to be successful.

In all of the algorithms except TPS, if more than the minimum GCPs are specified, parameters will be fitted so that the overall residual error is minimized. This is helpful to minimize the impact of registration errors, i.e. slight imprecisions in pointer clicks or typed coordinates, or other small local image deformations. Absent other GCPs to compensate, such errors or deformations could translate into significant distortions, especially near the edges of the georeferenced image. However, if more than the minimum GCPs are specified, they will match only approximately in the output. In contrast, TPS will precisely match all specified GCPs, but may introduce significant deformations between nearby GCPs with registration errors.

Define the Resampling method

The type of resampling you choose will likely depend on your input data and the ultimate objective of the exercise. If you don't want to change statistics of the raster (other than as implied by nonuniform geometric scaling if using other than the Linear, Helmert, or Polynomial 1 transformations), you might want to choose 'Nearest neighbour'. In contrast, 'cubic resampling', for instance, will usually generate a visually smoother result.

It is possible to choose between five different resampling methods:

1. Nearest neighbour
2. Bilinear (2x2 kernel)
3. Cubic (4x4 kernel)
4. Cubic B-Spline (4x4 kernel)
5. Lanczos (6x6 kernel)

Define the Raster creation options

When exporting a raster, *Raster creation options* allows you to define additional options that control how the output file is structured and compressed. See more at *Raster driver options*.

> **Tip**
>
> Select an empty entry if you want to create your own custom combination of parameters.

Define the transformation settings

There are several options that need to be defined for the georeferenced output raster.

- The *Create world file* checkbox is only available if you decide to use the linear transformation type, because this means that the raster image actually won't be transformed. In this case, the *Output raster* field is not activated, because only a new world file will be created.
- For all other transformation types, you have to define an *Output raster*. As default, a new file ([filename]_modified) will be created in the same folder together with the original raster image.
- As a next step, you have to define the *Target CRS* (Coordinate Reference System) for the georeferenced raster (see *Working with Projections*).
- If you like, you can **generate a pdf map** and also **a pdf report**. The report includes information about the used transformation parameters, an image of the residuals and a list with all GCPs and their RMS errors.
- Furthermore, you can activate the *Set Target Resolution* checkbox and define the pixel resolution of the output raster. Default horizontal and vertical resolution is 1.
- The *Use 0 for transparency when needed* can be activated, if pixels with the value 0 shall be visualized transparent. In our example toposheet, all white areas would be transparent.
- The *Save GCP Points* will store GCP Points in a file next to the output raster.
- Finally, *Load in project when done* loads the output raster automatically into the QGIS map canvas when the transformation is done.

Running the transformation

After all GCPs have been collected and all transformation settings are defined, just press the Start georeferencing button to create the new georeferenced raster.

11.3.2 Georeferencing vector layer

Georeferencing vector layers works similarly to raster georeferencing, but instead of matching image pixels, you match vector geometries (points, lines, or polygons) to known spatial references.

The standard procedure starts the same as for raster georeferencing: open QGIS and add a layer to the map canvas to use as a reference. This can be a georeferenced raster or vector layer, or a WMS layer.

Open the Georeferencer dialog from *Layer* ► *Georeferencer*.

Start georeferencing by following these steps (in this example, we use an unreferenced `alaska.shp`):

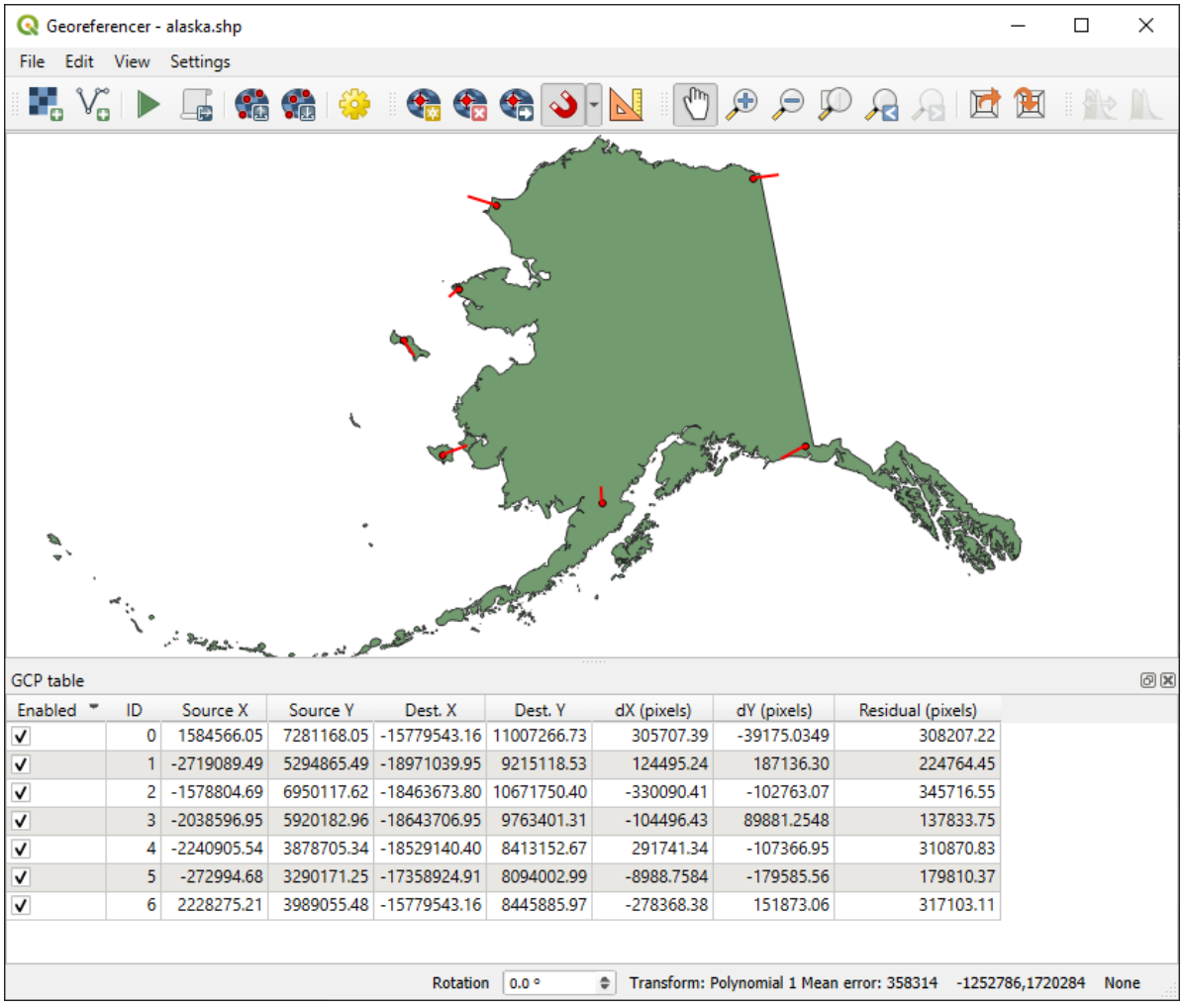


Fig. 11.40: Vector Georeferencer Dialog

1. Load the unreferenced vector layer using the button. The vector layer will appear in the main working area of the dialog.

2. Activate snapping by clicking the button and selecting the desired snapping type(s). This enables snapping to the reference layer when placing GCPs.

You can also use the  Advanced Digitizing dock to ensure high-precision point selection. For more information, refer to the *Advanced Digitizing* section.

3. Use the  Add GCP Point button to add a point to the working area. Enter its coordinates manually and set the CRS, or click the  From map canvas button to pick the coordinates from a georeferenced layer in the main QGIS map canvas. In that case, the CRS will be set automatically.
4. Define the transformation settings:
   - Select the *Transformation type*. The transformation algorithms are the same as those for raster georeferencing. See *Defining the transformation settings* for more details.
   - Define the *Target CRS* (Coordinate Reference System) for the georeferenced vector (see *Working with Projections*).
   - Set the output file format and path (e.g., GeoPackage, Shapefile). By default, a new file with suffix `_modified` will be created in the same folder as the original vector file.
   - Optionally, enable **Generate PDF map** and **Generate PDF report**. The report includes transformation parameters, GCP residuals, and a summary of RMS errors.
   - Enable  *Save GCP Points* to store GCPs in a file alongside the output vector layer.
   - Enable  *Load in project when done* to add the result directly to the map canvas.
5. Click  Start georeferencing to run the transformation and generate the georeferenced vector layer.

11.3.3 Show and adapt layer properties

Clicking on the *Source properties* option in the *Settings* menu opens the *Raster Layer properties* or *Vector Layer properties* depending on the type of layer you are georeferencing.

11.3.4 Configure the georeferencer

You can customize the behavior of the georeferencer in *Settings* ► *Configure Georeferencer* (or use keyboard shortcut `Ctrl+P`).

- Under *Point Tip* you can use the checkboxes to toggle displaying GCP IDs and X/Y coordinates in both the Georeferencer window and the main map canvas.
- *Residual Units* controls whether residual units are given in pixels or map units
- *PDF Report* allows you to set margin size in mm for the report export
- *PDF Map* allows you to choose a paper size for the map export
- Finally, you can activate to  *Show Georeferencer window docked*. This will dock the Georeferencer window in the main QGIS window rather than showing it as a separate window that can be minimized.

11.4 Exploring Data Formats and Fields

11.4.1 Raster data

GIS raster data are matrices of discrete cells that represent features / phenomena on, above or below the earth's surface. Each cell in the raster grid has the same size, and cells are usually rectangular (in QGIS they will always be rectangular). Typical raster datasets include remote sensing data, such as aerial photography, or satellite imagery and modelled data, such as elevation or temperature.

Unlike vector data, raster data typically do not have an associated database record for each cell. They are geocoded by pixel resolution and the X/Y coordinate of a corner pixel of the raster layer. This allows QGIS to position the data correctly on the map canvas.

The GeoPackage format is convenient for storing raster data when working with QGIS. The popular and powerful GeoTiff format is a good alternative.

QGIS makes use of georeference information inside the raster layer (e.g., GeoTiff) or an associated *world file* to properly display the data.

11.4.2 Vector Data

Many of the features and tools available in QGIS work the same, regardless of the vector data source. However, because of the differences in format specifications (GeoPackage, ESRI Shapefile, MapInfo and MicroStation file formats, AutoCAD DXF, PostgreSQL, SpatiaLite, Oracle Spatial, MS SQL Server, SAP HANA Spatial databases and many more), QGIS may handle some of their properties differently. Support is provided by the GDAL vector drivers. This section describes how to work with some of these specifics.

> **Note**
>
> QGIS supports (multi)point, (multi)line, (multi)polygon, CircularString, CompoundCurve, CurvePolygon, MultiCurve, MultiSurface feature types, all optionally with Z and/or M values.
>
> You should also note that some drivers don't support some of these feature types, like CircularString, CompoundCurve, CurvePolygon, MultiCurve, MultiSurface feature type. QGIS will convert them.

GeoPackage

The GeoPackage (GPKG) format is platform-independent, and is implemented as a SQLite database container, and can be used to store both vector and raster data. The format was defined by the Open Geospatial Consortium (OGC), and was published in 2014.

GeoPackage can be used to store the following in a SQLite database:

- **vector** features
- **tile matrix sets of imagery** and **raster** maps
- attributes (non-spatial data)
- extensions

Since QGIS version 3.8, GeoPackage can also store QGIS projects. GeoPackage layers can have JSON fields.

GeoPackage is the default format for vector data in QGIS.

ESRI Shapefile format

The ESRI Shapefile format is still one of the most used vector file formats, even if it has some limitations compared to for instance GeoPackage and SpatiaLite.

An ESRI Shapefile format dataset consists of several files. The following three are required:

1. `.shp` file containing the feature geometries
2. `.dbf` file containing the attributes in dBase format
3. `.shx` index file

An ESRI Shapefile format dataset can also include a file with a `.prj` suffix, which contains projection information. While it is very useful to have a projection file, it is not mandatory. A Shapefile format dataset can contain additional files. For further details, see the the ESRI technical specification.

GDAL has read-write support for compressed ESRI Shapefile format (`shz` and `shp.zip`).

Improving Performance for ESRI Shapefile format datasets

To improve the drawing performance for an ESRI Shapefile format dataset, you can create a spatial index. A spatial index will improve the speed of both zooming and panning. Spatial indexes used by QGIS have a `.qix` extension.

Use these steps to create the index:

1. Load an ESRI Shapefile format dataset (see *The Browser Panel*)
2. Open the *Layer Properties* dialog by double-clicking on the layer name in the legend or by right-clicking and choosing *Properties...* from the context menu
3. In the *Source* tab, click the *Create Spatial Index* button

Problem loading a .prj file

If you load an ESRI Shapefile format dataset with a `.prj` file and QGIS is not able to read the coordinate reference system from that file, you will need to define the proper projection manually in the *Layer Properties* ► *Source* tab of the layer by clicking the Select CRS button. This is due to the fact that `.prj` files often do not provide the complete projection parameters as used in QGIS and listed in the *CRS* dialog.

For the same reason, if you create a new ESRI Shapefile format dataset with QGIS, two different projection files are created: a `.prj` file with limited projection parameters, compatible with ESRI software, and a `.qpj` file, providing all the parameters of the CRS. Whenever QGIS finds a `.qpj` file, it will be used instead of the `.prj`.

Delimited Text Files

Delimited text files are very common and widely used because of their simplicity and readability – data can be viewed and edited in a plain text editor. A delimited text file is tabular data with columns separated by a defined character and rows separated by line breaks. The first row usually contains the column names. A common type of delimited text file is a CSV (Comma Separated Values), with columns separated by commas. Delimited text files can also contain positional information (see *Storing geometry information in delimited text files*).

QGIS allows you to load a delimited text file as a layer or an ordinary table (see *The Browser Panel* or *Importing a delimited text file*). First check that the file meets the following requirements:

1. The file must have a delimited header row of field names. This must be the first line of the data (ideally the first row in the text file).
2. If geometry should be enabled, the file must contain field(s) that define the geometry. These field(s) can have any name.
3. The X and Y coordinates fields (if geometry is defined by coordinates) must be specified as numbers. The coordinate system is not important.
4. If you have a CSV file with non-string columns, you can have an accompanying CSVT file (see section *Using CSVT file to control field formatting*).

The elevation point data file `elevp.csv` in the QGIS sample dataset (see section *Downloading sample data*) is an example of a valid text file:

```
X;Y;ELEV
-300120;7689960;13
-654360;7562040;52
1640;7512840;3
[...]
```

Some things to note about the text file:

1. The example text file uses `;` (semicolon) as delimiter (any character can be used to delimit the fields).
2. The first row is the header row. It contains the fields `X`, `Y` and `ELEV`.
3. No quotes (`"`) are used to delimit text fields
4. The X coordinates are contained in the `X` field
5. The Y coordinates are contained in the `Y` field

Storing geometry information in delimited text files

Delimited text files can contain geometry information in two main forms:

- As coordinates in separate columns (eg. `Xcol`, `Ycol`...), for point geometry data;
- As well-known text (WKT) representation of geometry in a single column, for any geometry type.

Features with curved geometries (CircularString, CurvePolygon and CompoundCurve) are supported. Here are some examples of geometry types in a delimited text file with geometries coded as WKT:

```
Label;WKT_geom
LineString;LINESTRING(10.0 20.0, 11.0 21.0, 13.0 25.5)
CircularString;CIRCULARSTRING(268 415,227 505,227 406)
CurvePolygon;CURVEPOLYGON(CIRCULARSTRING(1 3, 3 5, 4 7, 7 3, 1 3))
CompoundCurve;COMPOUNDCURVE((5 3, 5 13), CIRCULARSTRING(5 13, 7 15,
  9 13), (9 13, 9 3), CIRCULARSTRING(9 3, 7 1, 5 3))
```

Delimited text files also support Z and M coordinates in geometries:

```
LINESTRINGZ(10.0 20.0 30.0, 11.0 21.0 31.0, 11.0 22.0 30.0)
```

Using CSVT file to control field formatting

When loading CSV files, the GDAL driver assumes all fields are strings (i.e. text) unless it is told otherwise. You can create a CSVT file to tell GDAL (and QGIS) the data type of the different columns:

Type	Name	Example
Whole number	Integer	4
Boolean	Integer(Boolean)	true
Decimal number	Real	3.456
Date	Date (YYYY-MM-DD)	2016-07-28
Time	Time (HH:MM:SS+nn)	18:33:12+00
Date & Time	DateTime (YYYY-MM-DD HH:MM:SS+nn)	2016-07-28 18:33:12+00
CoordX	CoordX	8.8249
CoordY	CoordY	47.2274
Point(X)	Point(X)	8.8249
Point(Y)	Point(Y)	47.2274
WKT	WKT	POINT(15 20)

The CSVT file is a **ONE line** plain text file with the data types in quotes and separated by commas, e.g.:

```
"Integer","Real","String"
```

You can even specify width and precision of each column, e.g.:

```
"Integer(6)","Real(5.5)","String(22)"
```

This file is saved in the same folder as the `.csv` file, with the same name, but `.csvt` as the extension.

You can find more information at GDAL CSV Driver.

Tip

Detect Field Types

Instead of using a CSVT file to tell the data types, QGIS provides the possibility to automatically detect the field types and to change the assumed field types.

PostgreSQL Database

In order to process geographical information in a PostgreSQL database, the PostGIS extension must first be installed. PostGIS extends the capabilities of the PostgreSQL relational database by adding support for storing, indexing, and querying geospatial data.

To enable PostGIS in your database you have to activate the extension in your database (open database and run `CREATE EXTENSION postgis;`). PostGIS enabled databases are also often named "PostGIS layer". Using PostGIS, vector functions such as select and identify work more accurately than they do with GDAL layers in QGIS.

Tip

PostGIS Layers

Normally, a PostGIS layer is identified by an entry in the geometry_columns table. QGIS can load layers that do not have an entry in the geometry_columns table. This includes both tables and views. Refer to your PostgreSQL manual for information on creating views.

This section contains some details on how QGIS accesses PostgreSQL layers. Most of the time, QGIS should simply provide you with a list of database tables that can be loaded, and it will load them on request. However, if you have trouble loading a PostgreSQL table into QGIS, the information below may help you understand QGIS messages and give you directions for modifying the PostgreSQL table or view definition to allow QGIS to load it.

Note

A PostgreSQL database can also store QGIS projects.

Primary key

QGIS requires that PostgreSQL layers contain a column that can be used as a unique key for the layer. For tables, this usually means that the table needs a primary key, or a column with a unique constraint on it. In QGIS, this column needs to be of type int4 (an integer of size 4 bytes). Alternatively, the ctid column can be used as primary key. If a table lacks these items, the oid column will be used instead. Performance will be improved if the column is indexed (note that primary keys are automatically indexed in PostgreSQL).

QGIS offers a checkbox **Select at id** that is activated by default. This option gets the ids without the attributes, which is faster in most cases.

View

If the PostgreSQL layer is a view, the same requirement exists, but views do not always have primary keys or columns with unique constraints on them. You have to define a primary key field (has to be integer) in the QGIS dialog before you can load the view. If a suitable column does not exist in the view, QGIS will not load the layer. If this occurs, the solution is to alter the view so that it does include a suitable column (a type of integer and either a primary key or with a unique constraint, preferably indexed).

As for table, a checkbox **Select at id** is activated by default (see above for the meaning of the checkbox). It can make sense to disable this option when you use expensive views.

> **Note**
>
> **PostgreSQL foreign table**
>
> PostgreSQL foreign tables are not explicitely supported by the PostgreSQL provider and will be handled like a view.

QGIS layer_style table and database backup

If you want to make a backup of your PostgreSQL database using the `pg_dump` and `pg_restore` commands, and the default layer styles as saved by QGIS fail to restore afterwards, you need to set the XML option to `DOCUMENT` before the restore command:

1. Make a PLAIN backup of the `layer_style` table
2. Open the file within a text editor
3. Change the line `SET xmloption = content;` into `SET XML OPTION DOCUMENT;`
4. Save the file
5. Use psql to restore the table in the new database

Filter database side

QGIS allows to filter features already on server side. Check *Settings* ► *Options* ► *Data Sources* ►  *Execute expressions on server-side if possible* to do so. Only supported expressions will be sent to the database. Expressions using unsupported operators or functions will gracefully fallback to local evaluation.

Support of PostgreSQL data types

Data types supported by the PostgreSQL provider include: integer, float, boolean, binary object, varchar, geometry, timestamp, array, hstore and json.

Importing Data into PostgreSQL

Data can be imported into PostgreSQL using several tools, including the Browser, DB Manager plugin and the command line tools shp2pgsql and ogr2ogr.

DB Manager

QGIS comes with a core plugin named  DB Manager. It can be used to load data, and it includes support for schemas. See section *DB Manager Plugin* for more information.

shp2pgsql

PostGIS includes a utility called **shp2pgsql**, that can be used to import Shapefile format datasets into a PostGIS-enabled database. For example, to import a Shapefile format dataset named `lakes.shp` into a PostgreSQL database named `gis_data`, use the following command:

```
shp2pgsql -s 2964 lakes.shp lakes_new | psql gis_data
```

This creates a new layer named `lakes_new` in the `gis_data` database. The new layer will have a spatial reference identifier (SRID) of 2964. See section *Working with Projections* for more information about spatial reference systems and projections.

> **Tip**
>
> **Exporting datasets from PostgreSQL**
>
> There is also a tool for exporting postgreSQL datasets wth geographic data to Shapefile format: **pgsql2shp**. It is shipped within your PostGIS distribution.

ogr2ogr

In addition to **shp2pgsql** and **DB Manager**, there is another tool for feeding geographical data in PostgreSQL: **ogr2ogr**. It is part of your GDAL installation.

To import a Shapefile format dataset into PostgreSQL, do the following:

```
ogr2ogr -f "PostgreSQL" PG:"dbname=postgis host=myhost.de user=postgres
password=topsecret" alaska.shp
```

This will import the Shapefile format dataset `alaska.shp` into the database *postgis* using the user *postgres* with the password *topsecret* on the host server *myhost.de*.

Note that GDAL must be built with PostgreSQL. You can verify this by typing (in ):

```
ogrinfo --formats | grep -i post
```

If you prefer to use the PostgreSQL's **COPY** command instead of the default **INSERT INTO** method, you can export the following environment variable (at least available on  and ):

```
export PG_USE_COPY=YES
```

ogr2ogr does not create spatial indexes like **shp2pgsl** does. You need to create them manually, using the normal SQL command **CREATE INDEX** afterwards, as an extra step (as described in the next section *Improving Performance*).

Improving Performance

Retrieving features from a PostgreSQL database can be time-consuming, especially over a network. You can improve the drawing performance of PostgreSQL layers by ensuring that a spatial index exists on each layer in the database. PostGIS supports creation of a GiST (Generalized Search Tree) index to speed up spatial searching (GiST index information is taken from the PostGIS documentation available at https://postgis.net).

Tip

You can use the DBManager to create an index for your layer. You should first select the layer and click on *Table* ► *Edit table*, go to *Indexes* tab and click on *Add Spatial Index*.

The syntax for creating a GiST index is:

```
CREATE INDEX [indexname] ON [tablename]
  USING GIST ( [geometryfield] GIST_GEOMETRY_OPS );
```

Note that for large tables, creating the index can take a long time. Once the index is created, you should perform a `VACUUM ANALYZE`. See the PostGIS documentation (POSTGIS-PROJECT in *Literature and Web References*) for more information.

The following example creates a GiST index:

```
gsherman@madison:~/current$ psql gis_data
Welcome to psql 8.3.0, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit

gis_data=# CREATE INDEX sidx_alaska_lakes ON alaska_lakes
gis_data-# USING GIST (geom GIST_GEOMETRY_OPS);
CREATE INDEX
gis_data=# VACUUM ANALYZE alaska_lakes;
VACUUM
gis_data=# \q
gsherman@madison:~/current$
```

SpatiaLite Layers

If you want to save a vector layer using the SpatiaLite format, you can do this by following instructions at *Creating new layers from an existing layer*. You select `SpatiaLite` as *Format* and enter both *File name* and *Layer name*.

Also, you can select `SQLite` as format and then add `SPATIALITE=YES` in the *Custom Options* ► *Data source* field. This tells GDAL to create a SpatiaLite database. See also https://gdal.org/en/latest/drivers/vector/sqlite.html.

QGIS also supports editable views in SpatiaLite. For SpatiaLite data management, you can also use the core plugin *DB Manager*.

If you want to create a new SpatiaLite layer, please refer to section *Creating a new SpatiaLite layer*.

GeoJSON specific parameters

When *exporting layers* to GeoJSON, there are some specific *Layer Options* available. These options come from GDAL which is responsible for the writing of the file:

- *COORDINATE_PRECISION* the maximum number of digits after the decimal separator to write in coordinates. Defaults to 15 (note: for Lat Lon coordinates 6 is considered enough). Truncation will occur to remove trailing zeros.
- *RFC7946* by default GeoJSON 2008 will be used. If set to YES, the updated RFC 7946 standard will be used. Default is NO (thus GeoJSON 2008). See https://gdal.org/en/latest/drivers/vector/geojson.html#rfc-7946-write-support for the main differences, in short: only EPSG:4326 is allowed, other crs's will be transformed, polygons will be written such as to follow the right-hand rule for orientation, values of a "bbox" array are [west, south, east, north], not [minx, miny, maxx, maxy]. Some extension member names are forbidden in FeatureCollection, Feature and Geometry objects, the default coordinate precision is 7 decimal digits
- *WRITE_BBOX* set to YES to include the bounding box of the geometries at the feature and feature collection level

Besides GeoJSON there is also an option to export to "GeoJSON - Newline Delimited" (see https://gdal.org/en/latest/drivers/vector/geojsonseq.html). Instead of a FeatureCollection with Features, you can stream one type (probably only Features) sequentially separated with newlines.

GeoJSON - Newline Delimited has some specific Layer options availabe too:

- *COORDINATE_PRECISION* see above (same as for GeoJSON)
- *RS* whether to start records with the RS=0x1E character. The difference is how the features are separated: only by a newline (LF) character (Newline Delimited JSON, geojsonl) or by also prepending a record-separator (RS) character (giving GeoJSON Text Sequences, geojsons). Default to NO. Files are given the `.json` extension if extension is not provided.

SAP HANA Spatial Layers

This section contains some details on how QGIS accesses SAP HANA layers. Most of the time, QGIS should simply provide you with a list of database tables and views that can be loaded, and it will load them on request. However, if you have trouble loading an SAP HANA table or view into QGIS, the information below may help you understand the root cause and assist in resolving the issue.

Feature Identification

If you'd like to use all of QGIS' feature editing capabilities, QGIS must be able to unambiguously identify each feature in a layer. Internally, QGIS uses a 64-bit signed integer to identify features, whereas the negative range is reserved for special purposes.

Therefore, the SAP HANA provider requires a unique key that can be mapped to a positive 64-bit integer to fully support QGIS' feature editing capabilities. If it is not possible to create such a mapping, you might still view the features, but editing might not work.

Adding tables

When adding a table as a layer, the SAP HANA provider uses the table's primary key to map it to a unique feature id. Therefore, to have full feature editing support, you need to have a primary key to your table definition.

The SAP HANA provider supports multi-column primary keys, but if you'd like to get the best performance, your primary key should be a single column of type `INTEGER`.

Adding views

When adding a view as a layer, the SAP HANA provider cannot automatically identify columns that unambiguously identify a feature. Furthermore, some views are read-only and cannot be edited.

To have full feature editing support, the view must be updatable (check column `IS_READ_ONLY` in system view `SYS.VIEWS` for the view in question) and you must manually provide QGIS with one or more columns that identify a feature. The columns can be given by using *Layer* ► *Add Layer* ► *Add SAP HANA Spatial Layer* and then selecting the columns in the *Feature id* column. For best performance, the *Feature id* value should be a single `INTEGER` column.

11.4.3 Layers crossing 180° longitude

Many GIS packages don't wrap layers with a geographic reference system (lat/lon) crossing the 180 degrees longitude line. As result, if we open such a layer in QGIS, we could see two widely separated locations, that should appear near each other. In Fig. 11.41, the tiny point on the far left of the map canvas (Chatham Islands) should be within the grid, to the right of the New Zealand main islands.



Fig. 11.41: Map in lat/lon crossing the 180° longitude line

Solving in PostgreSQL

A work-around is to transform the longitude values using PostgreSQL and the ST_ShiftLongitude function. This function reads every point/vertex in every component of every feature in a geometry, and shifts its longitude coordinate from -180..0° to 180..360° and vice versa if between these ranges. This function is symmetrical so the result is a 0..360° representation of a -180..180° data and a -180..180° representation of a 0..360° data.

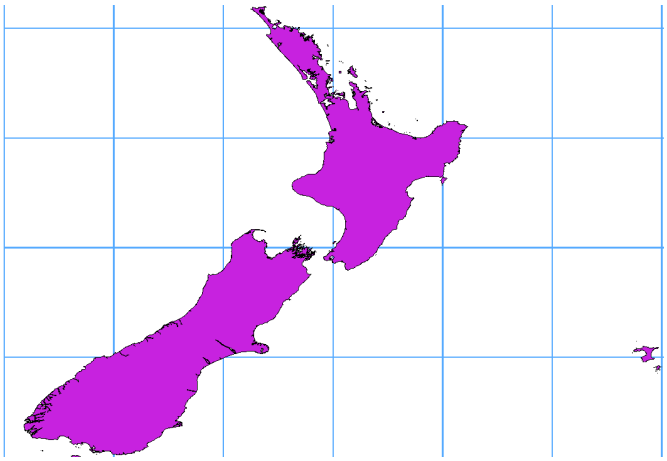


Fig. 11.42: Crossing 180° longitude applying the **ST_ShiftLongitude** function

1. Import data into PostgreSQL (*Importing Data into PostgreSQL*) using, for example, the DB Manager plugin.
2. Use the PostgreSQL command line interface to issue the following command:

```
-- In this example, "TABLE" is the actual name of your PostgreSQL table
update TABLE set geom=ST_ShiftLongitude(geom);
```

3. If everything went well, you should receive a confirmation about the number of features that were updated. Then you'll be able to load the map and see the difference (*Figure_vector_crossing_map*).

CHAPTER

TWELVE

WORKING WITH VECTOR DATA

12.1 The Vector Properties Dialog

The *Layer Properties* dialog for a vector layer provides general settings to manage appearance of layer features in the map (symbology, labeling, diagrams), interaction with the mouse (actions, map tips, form design). It also provides information about the layer.

To access the *Layer Properties* dialog:

- In the *Layers* panel, double-click the layer or right-click and select *Properties…* from the pop-up menu;
- Go to *Layer* ► *Layer Properties…* menu when the layer is selected.

The vector *Layer Properties* dialog provides the following sections:

Information	*Source*	*Symbology*[1]
Labels[1]	*Mask*[1]	*3D View*[1]
Diagrams	*Fields*	*Attributes Form*
Joins	*Auxiliary Storage*	*Actions*
Display	*Rendering*	*Temporal*
Variables	*Elevation*	*Metadata*
Dependencies	*Legend*	*QGIS Server*
Digitizing	*External plugins*[2] tabs	

[1] Also available in the *Layer styling panel*

[2] *External plugins* you install can optionally add tabs to this dialog. Those are not presented in this document. Refer to their documentation.

> **Tip**
>
> **Share full or partial properties of the layer styles**
>
> The *Style* menu at the bottom of the dialog allows you to import or export these or part of these properties from/to several destination (file, clipboard, database). See *Managing Custom Styles*.

> **Note**
>
> Because properties (symbology, label, actions, default values, forms…) of embedded layers (see *Embedding layers from external projects*) are pulled from the original project file and to avoid changes that may break this behavior, the layer properties dialog is made unavailable for these layers.

12.1.1 Information Properties

The *Information* tab is read-only and represents an interesting place to quickly grab summarized information and metadata on the current layer. Provided information are:

- general such as name in the project, source path, list of auxiliary files, last save time and size, the used provider
- custom properties, used to store in the active project additional information about the layer. Default custom properties may include *layer notes*, *legend widgets*, *layer variables*, *form properties*… More custom properties can be created and managed using PyQGIS, specifically through the setCustomProperty() method.
- based on the provider of the layer: format of storage, geometry type, data source encoding, extent, feature count…
- the Coordinate Reference System: name, units, method, accuracy, reference (i.e. whether it's static or dynamic)
- picked from the *filled metadata*: access, extents, links, contacts, history…
- and related to its geometry (spatial extent, CRS…) or its attributes (number of fields, characteristics of each…).

12.1.2 Source Properties

Use this tab to define general settings for the vector layer.

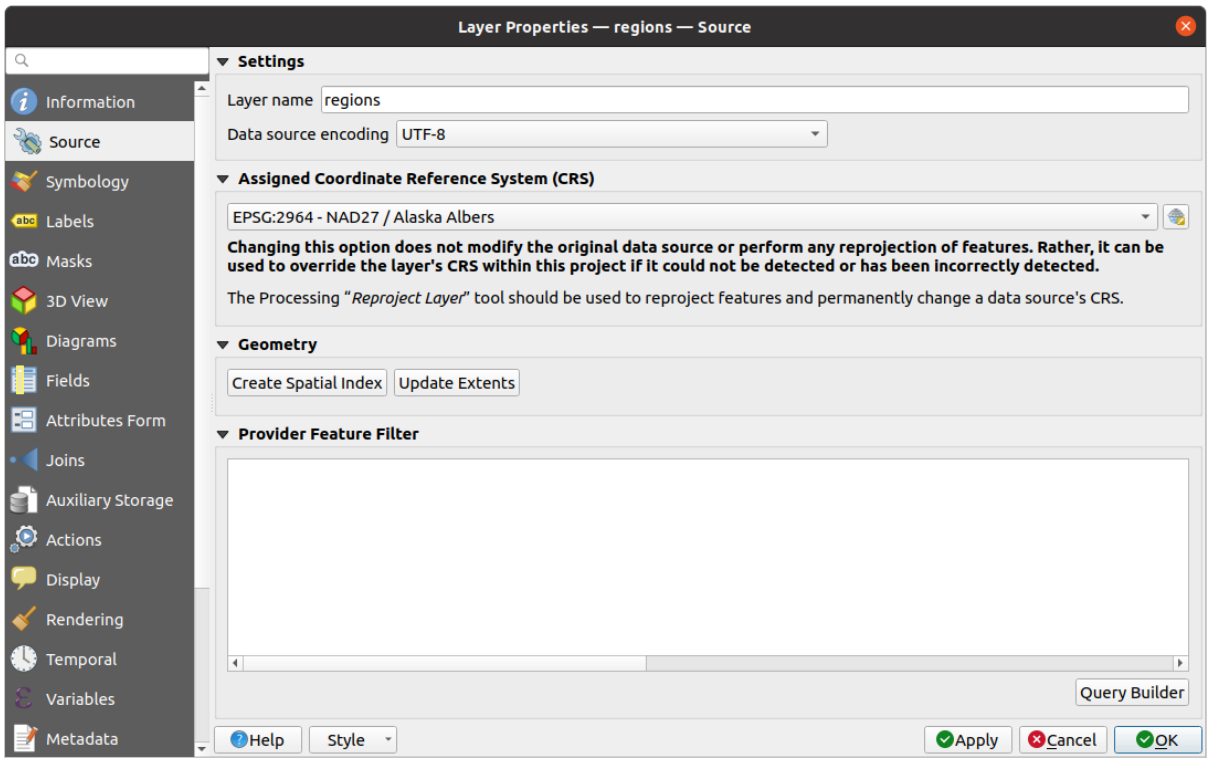


Fig. 12.1: Source tab in vector Layer Properties dialog

Settings

- Set a *Layer name* different from the layer filename that will be used to identify the layer in the project (in the *Layers Panel*, with expressions, in print layout legend, …)
- Depending on the data format, select the *Data source encoding* if not correctly detected by QGIS.

Coordinate Reference System and Geometry

- Displays the layer's *Assigned Coordinate Reference System (CRS)*. You can change the layer's CRS, selecting a recently used one in the drop-down list or clicking on  Select CRS button (see *Coordinate Reference System Selector*). Use this process only if the CRS applied to the layer is a wrong one or if none was applied. If you wish to reproject your data into another CRS, rather use layer reprojection algorithms from Processing or *Save it into another layer*.
- *Create spatial index* (only for OGR-supported formats).
- *Update extents* information for a layer.

Query Builder

The Query Builder dialog is accessible through the *Query Builder* button at the bottom of the *Source* tab in the Layer Properties dialog, under the *Provider feature filter* group.

The Query Builder provides an interface that allows you to define a subset of the features in the layer using a SQL-like WHERE clause and to display the result in the main window. As long as the query is active, only the features corresponding to its result are available in the project.

You can use one or more layer attributes to define the filter in the `Query Builder`. The use of more than one attribute is shown in Fig. 12.2. In the example, the filter combines the attributes

- `toa` (`DateTime` field: `cast("toa" as character) > '2017-05-17'` and `cast("toa" as character) < '2019-12-24T18:00:00'`),
- `name` (`String` field: `"name" > 'S'`) and
- `FID` (`Integer` field: `FID > 10`)

using the AND, OR and NOT operators and parenthesis. This syntax (including the DateTime format for the `toa` field) works for GeoPackage datasets.

The filter is made at the data provider (OGR, PostgreSQL, MS SQL Server…) level. So the syntax depends on the data provider (DateTime is for instance not supported for the ESRI Shapefile format). The complete expression:

```
cast("toa" as character) > '2017-05-17' AND
cast("toa" as character) < '2019-12-24T18:00:00' AND
NOT ("name" > 'S' OR FID > 10)
```

You can also open the *Query Builder* dialog using the *Filter…* option from the *Layer* menu or the layer contextual menu. The *Fields*, *Values* and *Operators* sections in the dialog help you to construct the SQL-like query exposed in the *Provider specific filter expression* box.

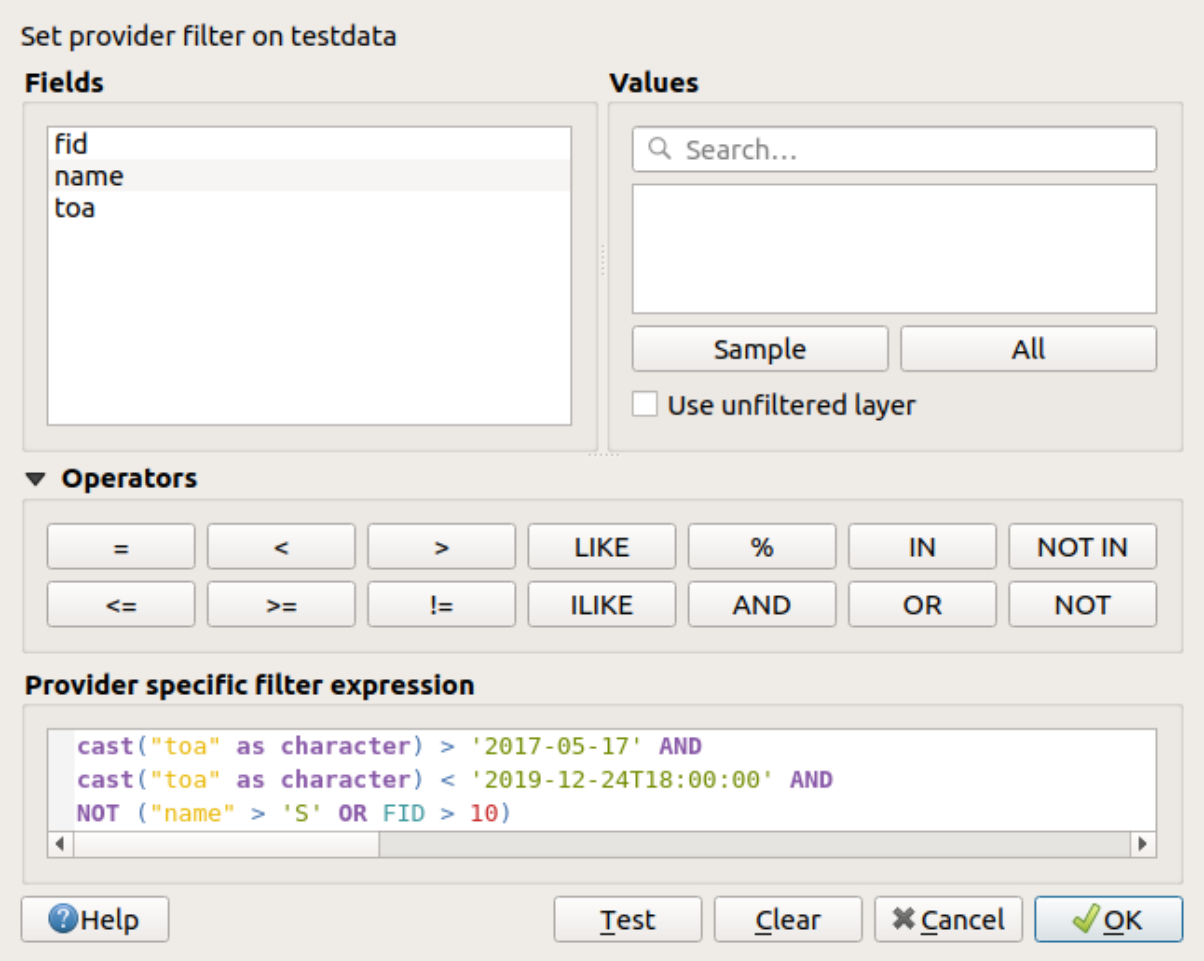


Fig. 12.2: Query Builder

The **Fields** list contains all the fields of the layer. To add an attribute column to the SQL WHERE clause field, double-click its name or just type it into the SQL box.

The **Values** frame lists the values of the currently selected field. To list all unique values of a field, click the *All* button. To instead list the first 25 unique values of the column, click the *Sample* button. To add a value to the SQL WHERE clause field, double click its name in the Values list. You can use the search box at the top of the Values frame to easily browse and find attribute values in the list.

The **Operators** section contains all usable operators. To add an operator to the SQL WHERE clause field, click the appropriate button. Relational operators (= , > , ...), string comparison operator (`LIKE`), and logical operators (`AND`, `OR`, ...) are available.

The *Test* button helps you check your query and displays a message box with the number of features satisfying the current query. Use the *Clear* button to wipe the SQL query and revert the layer to its original state (ie, fully load all the features). It is possible to *Save...* the query as a `.QQF` file, or *Load...* the query from a file into the dialog.

When a filter is applied, QGIS treats the resulting subset acts as if it were the entire layer. For example if you applied the filter above for 'Borough' (`"TYPE_2" = 'Borough'`), you can not display, query, save or edit `Anchorage`, because that is a 'Municipality' and therefore not part of the subset.

Tip

Filtered layers are indicated in the Layers Panel

In the *Layers* panel, filtered layer is listed with a Filter icon next to it indicating the query used when the mouse

hovers over the button. Double-click the icon opens the *Query Builder* dialog for edit. This can also be achieved through the *Layer* ► *Filter…* menu.

12.1.3 Symbology Properties

The  *Symbology* tab provides you with a comprehensive tool for rendering and symbolizing your vector data. You can use tools that are common to all vector data, as well as special symbolizing tools that were designed for the different kinds of vector data. However all types share the following dialog structure: in the upper part, you have a widget that helps you prepare the classification and the symbol to use for features and at the bottom the *Layer rendering* widget.

Tip

Switch quickly between different layer representations

Using the *Styles* ► *Add* menu at the bottom of the *Layer Properties* dialog, you can save as many styles as needed. A style is the combination of all properties of a layer (such as symbology, labeling, diagram, fields form, actions…) as you want. Then, simply switch between styles from the context menu of the layer in *Layers Panel* to automatically get different representations of your data.

Tip

Export vector symbology

You have the option to export vector symbology from QGIS into Google *.kml, *.dxf and MapInfo *.tab files. Just open the right mouse menu of the layer and click on *Save As…* to specify the name of the output file and its format. In the dialog, use the *Symbology export* menu to save the symbology either as *Feature symbology* ► or as *Symbol layer symbology* ►. If you have used symbol layers, it is recommended to use the second setting.

Features rendering

The renderer is responsible for drawing a feature together with the correct symbol. Regardless layer geometry type, there are four common types of renderers: single symbol, categorized, graduated and rule-based. For point layers, there are point displacement, point cluster and heatmap renderers available while polygon layers can also be rendered with the merged features, inverted polygons and 2.5 D renderers.

There is no continuous color renderer, because it is in fact only a special case of the graduated renderer. The categorized and graduated renderers can be created by specifying a symbol and a color ramp - they will set the colors for symbols appropriately. For each data type (points, lines and polygons), vector symbol layer types are available. Depending on the chosen renderer, the dialog provides different additional sections.

Note

If you change the renderer type when setting the style of a vector layer the settings you made for the symbol will be maintained. Be aware that this procedure only works for one change. If you repeat changing the renderer type the settings for the symbol will get lost.

Single Symbol Renderer

The *Single Symbol* renderer is used to render all features of the layer using a single user-defined symbol. See *The Symbol Selector* for further information about symbol representation.

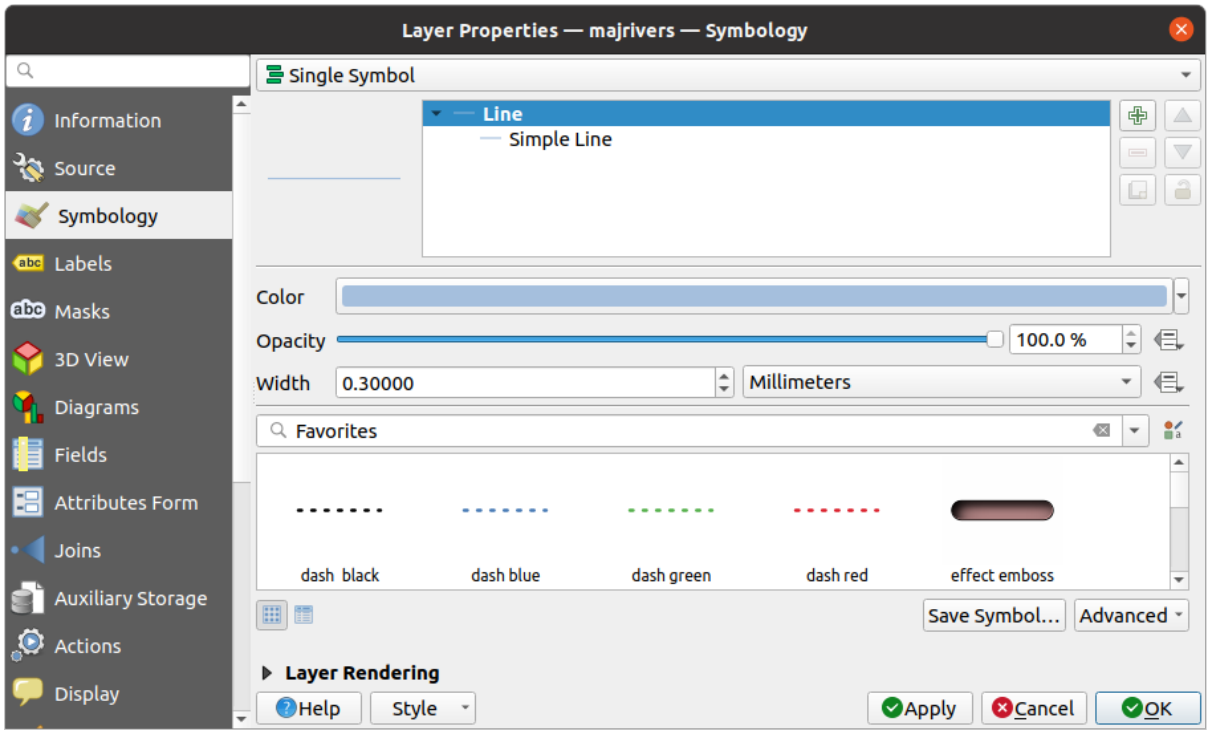


Fig. 12.3: Single symbol line properties

No Symbols Renderer

The *No Symbols* renderer is a special use case of the Single Symbol renderer as it applies the same rendering to all features. Using this renderer, no symbol will be drawn for features, but labeling, diagrams and other non-symbol parts will still be shown.

Selections can still be made on the layer in the canvas and selected features will be rendered with a default symbol. Features being edited will also be shown.

This is intended as a handy shortcut for layers which you only want to show labels or diagrams for, and avoids the need to render symbols with totally transparent fill/border to achieve this.

Categorized Renderer

The *Categorized* renderer is used to render the features of a layer, using a user-defined symbol whose aspect reflects the discrete values of a field or an expression.

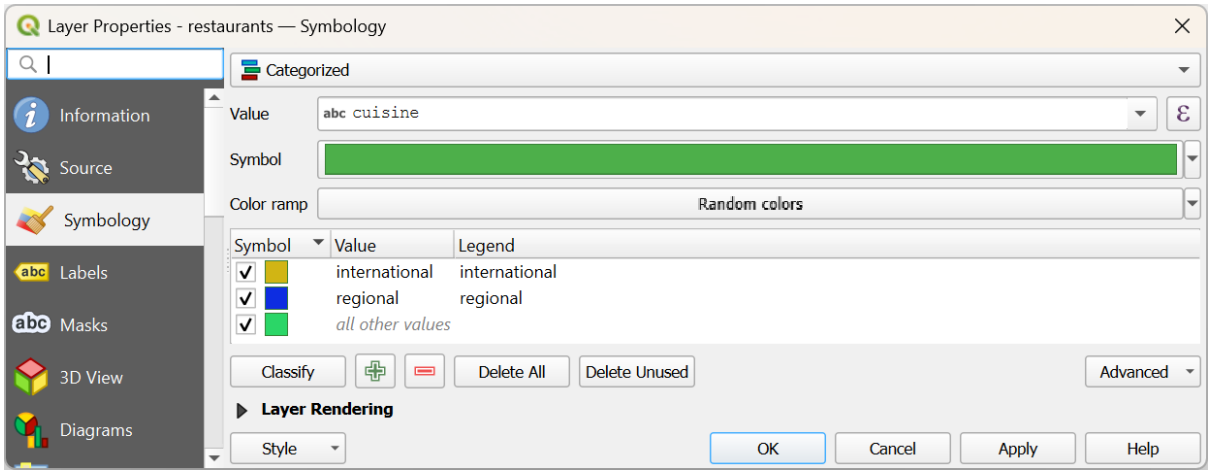


Fig. 12.4: Categorized Symbolizing options

To use categorized symbology for a layer:

1. Select the *Value* of classification: it can be an existing field or an *expression* you can type in the box or build using the associated ε button. Using expressions for categorizing avoids the need to create a field for symbology purposes only (eg, if your classification criteria are derived from one or more attributes).

   The expression used to classify features can be of any type; eg, it can:

   - be a comparison. In this case, QGIS returns values `1` (**True**) and `0` (**False**). Some examples:

     ```
     myfield >= 100
     $id = @atlas_featureid
     myfield % 2 = 0
     within( $geometry, @atlas_geometry )
     ```

   - combine different fields:

     ```
     concat( field_1, ' ', field_2 )
     ```

   - be a calculation on fields:

     ```
     myfield % 2
     year( myfield )
     field_1 + field_2
     substr( field_1, -3 )
     ```

   - be used to transform linear values to discrete classes, e.g.:

     ```
     CASE WHEN x > 1000 THEN 'Big' ELSE 'Small' END
     ```

   - combine several discrete values into a single category, e.g.:

     ```
     CASE
     WHEN building IN ('residence', 'mobile home') THEN 'residential'
     WHEN building IN ('commercial', 'industrial') THEN 'Commercial and␣
     ↪Industrial'
     END
     ```

> **Tip**
>
> While you can use any kind of expression to categorize features, for some complex expressions it might be simpler to use *rule-based rendering*.

> **Note**
>
> Keep in mind that some values may use widgets that do not display the actual value stored in the field. For example, a checkbox widget may store `1` and `0` for checked and unchecked states, while displaying `True` and `False` labels. In this case, to categorize features based on the checkbox state, you need to use the stored values (`1` and `0`) in the expression.

2. Configure the *Symbol*, which will be used as base symbol for all the classes;
3. Indicate the *Color ramp*, i.e. the range of colors from which the color applied to each symbol is selected.

   Besides the common options of the *color ramp widget*, you can apply a *Random Color Ramp* to the categories. You can click the *Shuffle Random Colors* entry to regenerate a new set of random colors if you are not satisfied.
4. Then click on the *Classify* button to create classes from the distinct values of the provided field or expression.
5. *Apply* the changes if the *live update* is not in use and each feature on the map canvas will be rendered with the symbol of its class.

   By default, QGIS appends an *all other values* class to the list. While empty at the beginning, this class is used as a default class for any feature not falling into the other classes (eg, when you create features with new values for the classification field / expression).

Further tweaks can be done to the default classification:

- You can Add new categories, Remove selected categories, *Delete All* of them or *Delete Unused* categories.
- A class can be disabled by unchecking the checkbox to the left of the class name; the corresponding features are hidden on the map.
- Drag-and-drop the rows to reorder the classes
- To change the symbol, the value or the legend of a class, double click the item.

Right-clicking over selected item(s) shows a contextual menu to:

- *Copy Symbol* and *Paste Symbol*, a convenient way to apply the item's representation to others
- *Change Color…* of the selected symbol(s)
- *Change Opacity…* of the selected symbol(s)
- *Change Output Unit…* of the selected symbol(s)
- *Change Width…* of the selected line symbol(s)
- *Change Size…* of the selected point symbol(s)
- *Change Angle…* of the selected point symbol(s)
- *Merge Categories*: Groups multiple selected categories into a single one. This allows simpler styling of layers with a large number of categories, where it may be possible to group numerous distinct categories into a smaller and more manageable set of categories which apply to multiple values.

> **Tip**
>
> Since the symbol kept for the merged categories is the one of the topmost selected category in the list, you may want to move the category whose symbol you wish to reuse to the top before merging.

- *Unmerge Categories* that were previously merged

The created classes also appear in a tree hierarchy in the *Layers* panel. Double-click an entry in the map legend to edit the assigned symbol. Right-click and you will get some *more options*.

The *Advanced* menu gives access to options to speed classification or fine-tune the symbols rendering:

- *Match to saved symbols*: Using the *symbols library*, assigns to each category a symbol whose name represents the classification value of the category
- *Match to symbols from file…*: Provided a file with symbols, assigns to each category a symbol whose name represents the classification value of the category
- *Symbol levels…* to define the order of symbols rendering.

Graduated Renderer

The  *Graduated* renderer is used to render all the features from a layer, using an user-defined symbol whose color or size reflects the assignment of a selected feature's attribute to a class.

Like the Categorized Renderer, the Graduated Renderer allows you to define rotation and size scale from specified columns.

Also, analogous to the Categorized Renderer, it allows you to select:

- The *Value* of classification: it can be an existing field or an *expression* you can type in the box or build using the associated  button. Using expressions for graduating avoids the need to create a field for symbology purposes only (eg, if your classification criteria are derived from one or more attributes).
- The symbol (using the Symbol selector dialog)
- The legend format and the precision
- The method to use to change the symbol: color or size
- The colors (using the color Ramp list) if the color method is selected
- The size (using the size domain and its unit)

Then you can use the Histogram tab which shows an interactive histogram of the values from the assigned field or expression. Class breaks can be moved or added using the histogram widget.

> **Note**
>
> You can use Statistical Summary panel to get more information on your vector layer. See *Statistical Summary Panel*.

Back to the Classes tab, you can specify the number of classes and also the mode for classifying features within the classes (using the Mode list). The available modes are:

- Equal Count (Quantile): each class will have the same number of elements (the idea of a boxplot).
- Equal Interval: each class will have the same size (e.g. with the values from 1 to 16 and four classes, each class will have a size of four).
- Fixed Interval: each class will have a fixed range of values (e.g. with the values from 1 to 16 and an interval size of 4, the classes will be 1-4, 5-8, 9-12 and 13-16).
- Logarithmic scale: suitable for data with a wide range of values. Narrow classes for low values and wide classes for large values (e.g. for decimal numbers with range [0..100] and two classes, the first class will be from 0 to 10 and the second class from 10 to 100).
- Natural Breaks (Jenks): the variance within each class is minimized while the variance between classes is maximized.
- Pretty Breaks: computes a sequence of about n+1 equally spaced nice values which cover the range of the values in x. The values are chosen so that they are 1, 2 or 5 times a power of 10. (based on pretty from the R statistical environment https://www.rdocumentation.org/packages/base/topics/pretty).
- Standard Deviation: classes are built depending on the standard deviation of the values.

The listbox in the center part of the *Symbology* tab lists the classes together with their ranges, labels and symbols that will be rendered.

Click on **Classify** button to create classes using the chosen mode. Each classes can be disabled unchecking the checkbox at the left of the class name.

To change symbol, value and/or label of the class, just double click on the item you want to change.

Right-clicking over selected item(s) shows a contextual menu to:

- *Copy Symbol* and *Paste Symbol*, a convenient way to apply the item's representation to others
- *Change Color…* of the selected symbol(s)
- *Change Opacity…* of the selected symbol(s)
- *Change Output Unit…* of the selected symbol(s)
- *Change Width…* of the selected line symbol(s)
- *Change Size…* of the selected point symbol(s)
- *Change Angle…* of the selected point symbol(s)

The example in Fig. 12.5 shows the graduated rendering dialog for the major_rivers layer of the QGIS sample dataset.

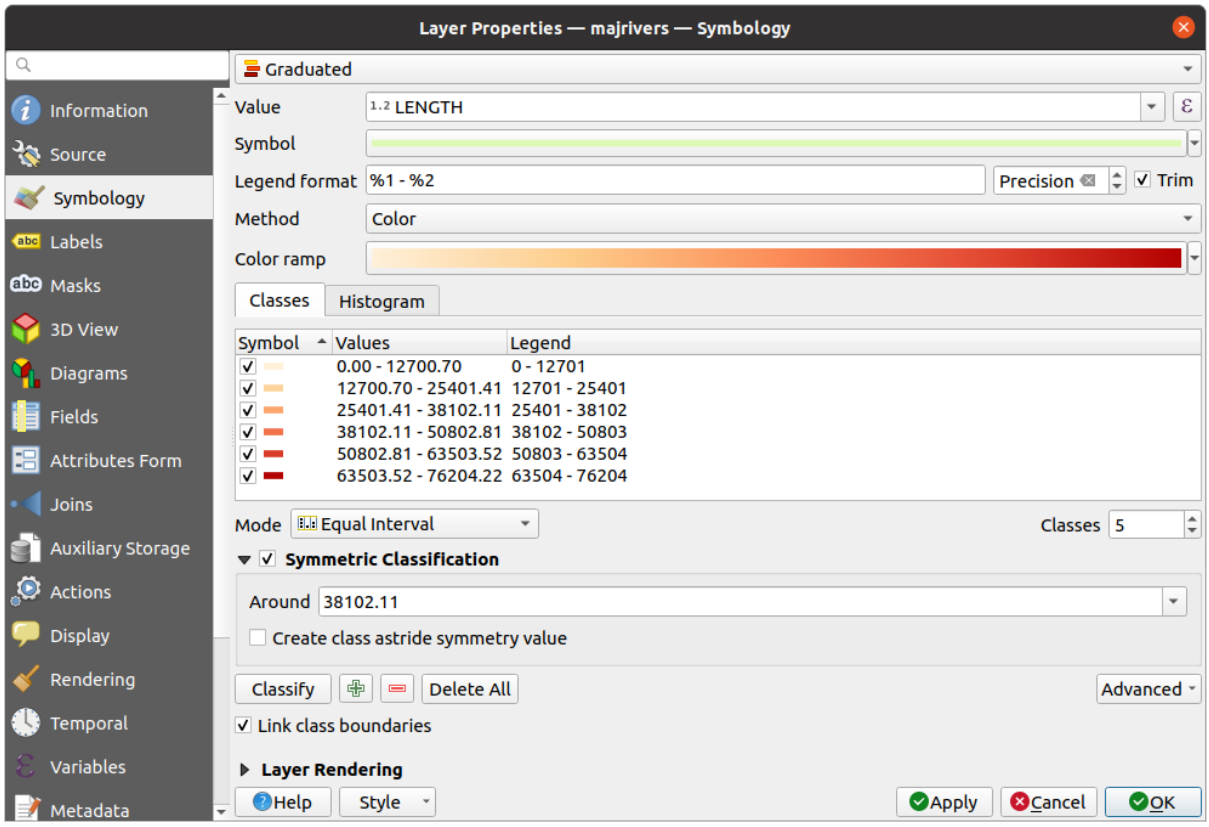


Fig. 12.5: Graduated Symbolizing options

The created classes also appear in a tree hierarchy in the *Layers* panel. Double-click an entry in the map legend to edit the assigned symbol. Right-click and you will get some *more options*.

Proportional Symbol and Multivariate Analysis

Proportional Symbol and Multivariate Analysis are not rendering types available from the Symbology rendering drop-down list. However with the *data-defined override* options applied over any of the previous rendering options, QGIS allows you to display your point and line data with such representation.

Creating proportional symbol

To apply a proportional rendering:

1. First apply to the layer the *single symbol renderer*.
2. Then set the symbol to apply to the features.
3. Select the item at the upper level of the symbol tree, and use the  Data-defined override *button* next to the *Size* (for point layer) or *Width* (for line layer) option.
4. Select a field or enter an expression, and for each feature, QGIS will apply the output value to the property and proportionally resize the symbol in the map canvas.

   If need be, use the *Size assistant…* option of the  menu to apply some transformation (exponential, flannery…) to the symbol size rescaling (see *Using the data-defined assistant interface* for more details).

You can choose to display the proportional symbols in the *Layers panel* and the *print layout legend item*: unfold the *Advanced* drop-down list at the bottom of the main dialog of the *Symbology* tab and select **Data-defined size legend…** to configure the legend items (see *Data-defined size legend* for details).

Creating multivariate analysis

A multivariate analysis rendering helps you evaluate the relationship between two or more variables e.g., one can be represented by a color ramp while the other is represented by a size.

The simplest way to create multivariate analysis in QGIS is to:

1. First apply a categorized or graduated rendering on a layer, using the same type of symbol for all the classes.
2. Then, apply a proportional symbology on the classes:
   1. Click on the *Change* button above the classification frame: you get the *The Symbol Selector* dialog.
   2. Rescale the size or width of the symbol layer using the  *data defined override* widget as seen above.

Like the proportional symbol, the scaled symbology can be added to the layer tree, on top of the categorized or graduated classes symbols using the *data defined size legend* feature. And both representation are also available in the print layout legend item.

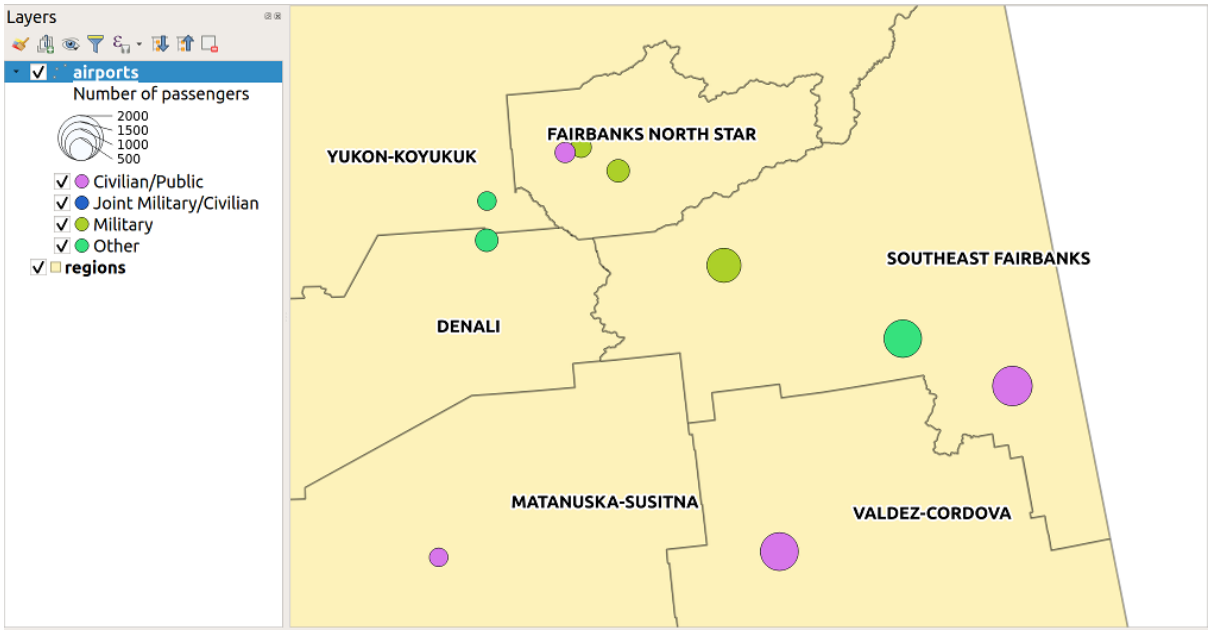


Fig. 12.6: Multivariate example with scaled size legend

Rule-based Renderer

Rules are QGIS *expressions* used to discriminate features according to their attributes or properties in order to apply specific rendering settings to them. Rules can be nested, and features belong to a class if they belong to all the upper nesting level(s).

The *Rule-based* renderer is thus designed to render all the features from a layer, using symbols whose aspect reflects the assignment of a selected feature to a fine-grained class.

To create a rule:

1. Activate an existing row by double-clicking it (by default, QGIS adds a symbol without a rule when the rendering mode is enabled) or click the Edit rule or Add rule button.
2. In the *Edit Rule* dialog that opens, you can define a label to help you identify each rule. This is the label that will be displayed in the *Layers Panel* and also in the print composer legend.
3. Manually enter an expression in the text box next to the *Filter* option or press the button next to it to open the expression string builder dialog.
4. Use the provided functions and the layer attributes to build an *expression* to filter the features you'd like to retrieve. Press the *Test* button to check the result of the query.
5. You can enter a longer label to complete the rule description.
6. You can use the *Scale Range* option to set scales at which the rule should be applied.
7. Finally, configure the *symbol to use* for these features.
8. And press *OK*.

A new row summarizing the rule is added to the Layer Properties dialog. You can create as many rules as necessary following the steps above or copy pasting an existing rule. Drag-and-drop the rules on top of each other to nest them and refine the upper rule features in subclasses.

The rule-based renderer can be combined with categorized or graduated renderers. Selecting a rule, you can organize its features in subclasses using the *Refine selected rules* drop-down menu. Refined classes appear like sub-items of the

rule, in a tree hierarchy and like their parent, you can set the symbology and the rule of each class. Automated rule refinement can be based on:

- **scales**: given a list of scales, this option creates a set of classes to which the different user-defined scale ranges apply. Each new scale-based class can have its own symbology and expression of definition. This can e.g. be a convenient way to display the same features with various symbols at different scales, or display only a set of features depending on the scale (e.g. local airports at large scale vs international airports at small scale).
- **categories**: applies a *categorized renderer* to the features falling in the selected rule.
- or **ranges**: applies a *graduated renderer* to the features falling in the selected rule.

Refined classes appear like sub-items of the rule, in a tree hierarchy and like above, you can set symbology of each class. Symbols of the nested rules are stacked on top of each other so be careful in choosing them. It is also possible to uncheck *Symbols* in the *Edit rule* dialog to avoid rendering a particular symbol in the stack.

In the *Edit rule* dialog, you can avoid writing all the rules and make use of the *Else* option to catch all the features that do not match any of the other rules, at the same level. This can also be achieved by writing `Else` in the *Rule* column of the *Layer Properties* ► *Symbology* ► *Rule-based* dialog.

Right-clicking over selected item(s) shows a contextual menu to:

- *Copy* and *Paste*, a convenient way to create new item(s) based on existing item(s)
- *Copy Symbol* and *Paste Symbol*, a convenient way to apply the item's representation to others
- *Change Color…* of the selected symbol(s)
- *Change Opacity…* of the selected symbol(s)
- *Change Output Unit…* of the selected symbol(s)
- *Change Width…* of the selected line symbol(s)
- *Change Size…* of the selected point symbol(s)
- *Change Angle…* of the selected point symbol(s)
- *Refine Current Rule*: open a submenu that allows to refine the current rule with **scales**, **categories** or **Ranges**. Same as selecting the *corresponding menu* at the bottom of the dialog.

Unchecking a row in the rule-based renderer dialog hides in the map canvas the features of the specific rule and the nested ones.

The created rules also appear in a tree hierarchy in the map legend. Double-click an entry in the map legend to edit the assigned symbol. Right-click and you will get some *more options*.

The example in Fig. 12.7 shows the rule-based rendering dialog for the rivers layer of the QGIS sample dataset.

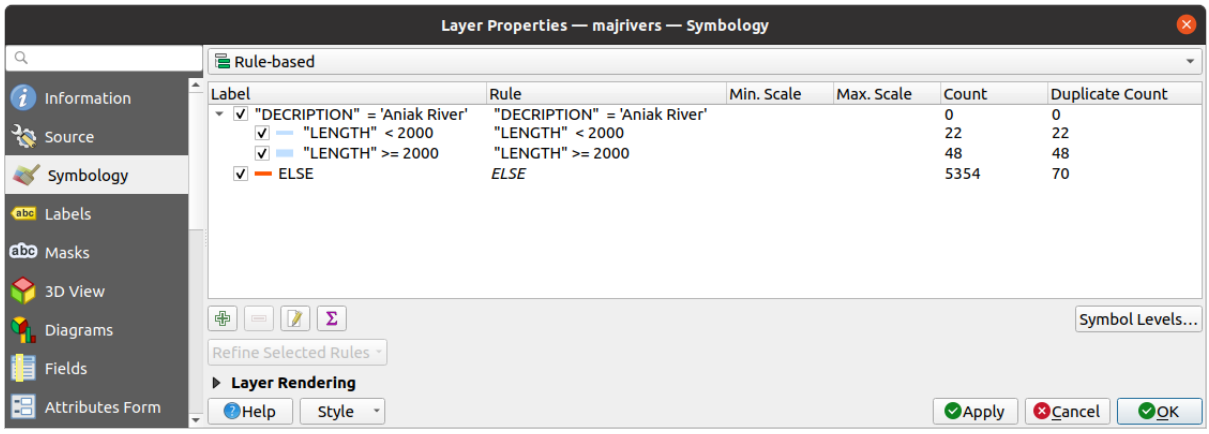


Fig. 12.7: Rule-based Symbolizing options

Point displacement Renderer

The  *Point Displacement* renderer takes the point features falling in a given distance tolerance from each other and places their symbols around their barycenter, following different placement methods. This can be a convenient way to visualize all the features of a point layer, even if they have the same location (e.g. amenities in a building).

To configure a point displacement renderer, you have to:

1. Set the *Center symbol*: how the virtual point at the center will look like
2. Select the *Renderer* type: how you want to classify features in the layer (single, categorized, rule-based…)
3. Press the *Renderer Settings…* button to configure features' symbology according to the selected renderer
4. Indicate the *Distance* tolerance in which close features are considered overlapping and then displaced over the same virtual point. Supports common symbol units.
5. Configure the *Placement methods*:
   - **Ring**: places all the features on a circle whose radius depends on the number of features to display.
   - **Concentric rings**: uses a set of concentric circles to show the features.
   - **Grid**: generates a regular grid with a point symbol at each intersection.
6. Displaced symbols are placed on the *Displacement lines*. While the minimal spacing of the displacement lines depends on the point symbols renderer, you can still customize some of their settings such as the *Stroke width*, *Stroke color* and *Size adjustment* (e.g., to add more spacing between the rendered points).
7. Use the *Labels* group options to perform points labeling: the labels are placed near the displaced symbol, and not at the feature real position.
   1. Select the *Label attribute*: a field of the layer to use for labeling
   2. Indicate the *Label font* properties and size
   3. Pick a *Label color*
   4. Set a *Label distance factor*: for each point feature, offsets the label from the symbol center proportionally to the symbol's diagonal size.
   5. Turn on  *Use scale dependent labeling* if you want to display labels only on scales larger than a given *Minimum map scale*.

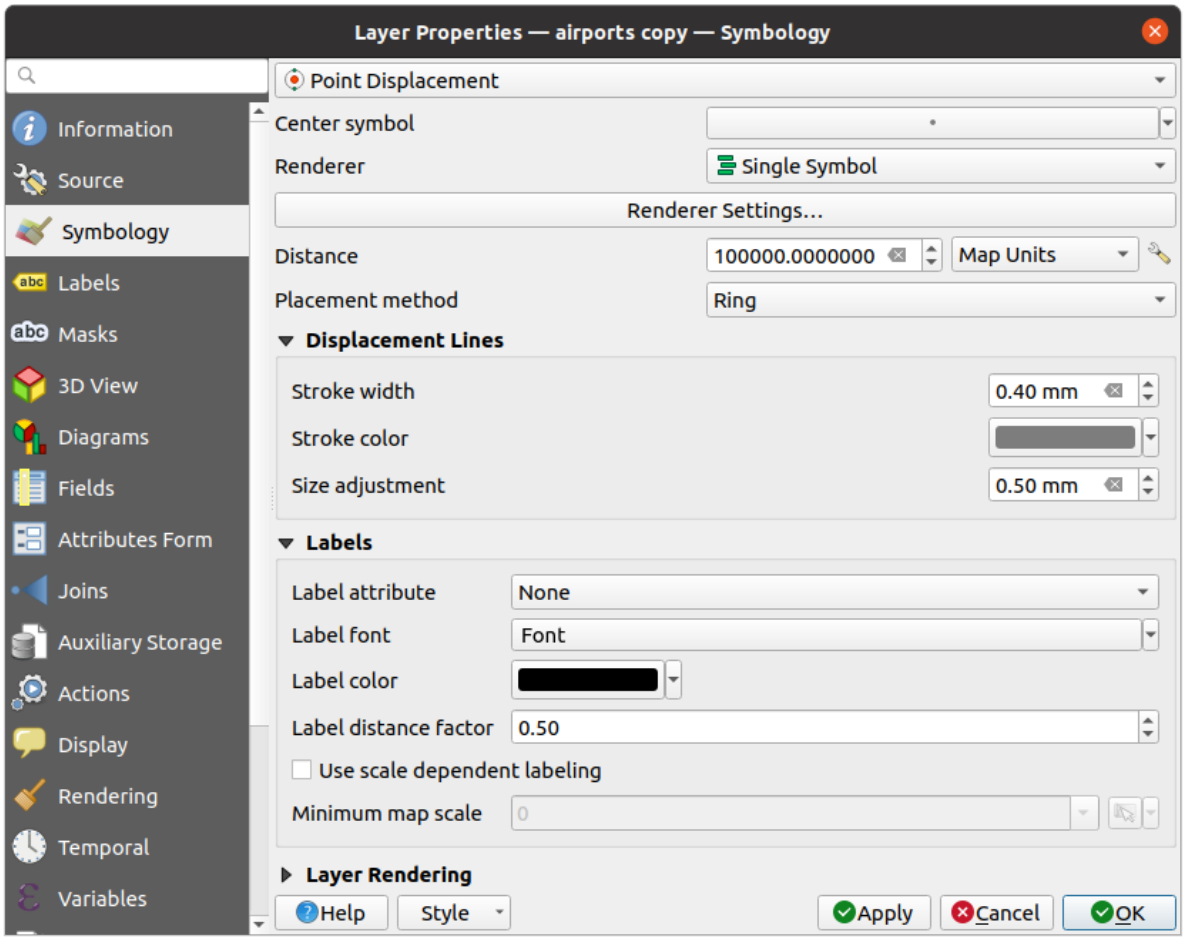


Fig. 12.8: Point displacement dialog

Note

Point Displacement renderer does not alter feature geometry, meaning that points are not moved from their position. They are still located at their initial place. Changes are only visual, for rendering purpose. Use instead the Processing *Points displacement* algorithm if you want to create displaced features.

Point Cluster Renderer

Unlike the *Point Displacement* renderer which blows up nearest or overlaid point features placement, the *Point Cluster* renderer groups nearby points into a single rendered marker symbol. Points that fall within a specified distance from each others are merged into a single symbol. Points aggregation is made based on the closest group being formed, rather than just assigning them the first group within the search distance.

From the main dialog, you can:

1. Set the symbol to represent the point cluster in the *Cluster symbol*; the default rendering displays the number of aggregated features thanks to the `@cluster_size` *variable* on Font marker symbol layer.
2. Select the *Renderer* type, i.e. how you want to classify features in the layer (single, categorized, rule-based…)
3. Press the *Renderer Settings…* button to configure features' symbology as usual. Note that this symbology is only visible on features that are not clustered, the *Cluster symbol* being applied otherwise. Also, when all the point

features in a cluster belong to the same rendering class, and thus would be applied the same color, that color represents the `@cluster_color` variable of the cluster.

4. Indicate the maximal *Distance* to consider for clustering features. Supports common symbol units.

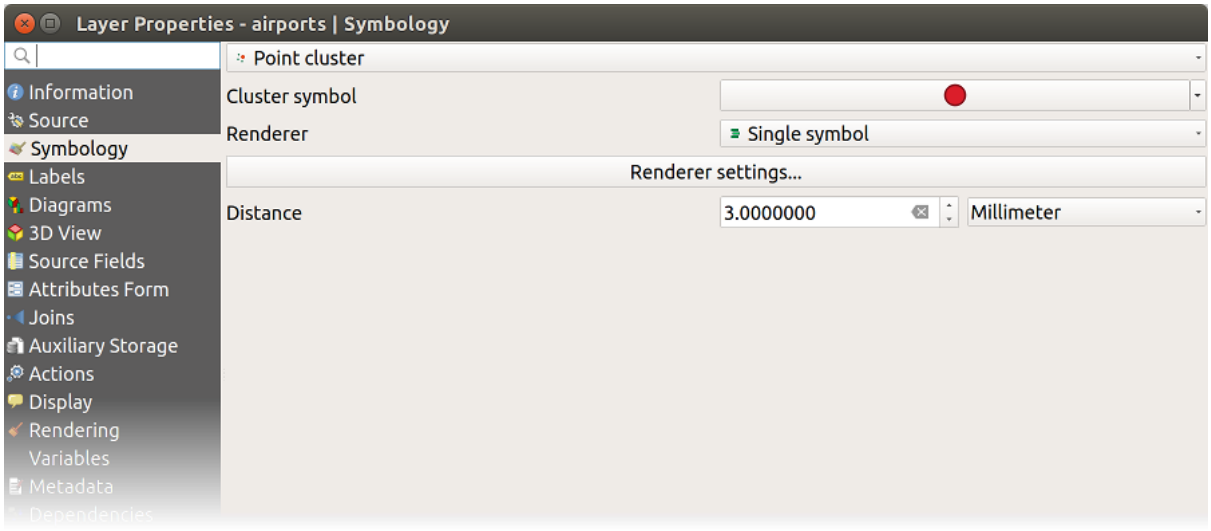


Fig. 12.9: Point Cluster dialog

Note

Point Cluster renderer does not alter feature geometry, meaning that points are not moved from their position. They are still located at their initial place. Changes are only visual, for rendering purpose. Use instead the Processing *K-means clustering* or *DBSCAN clustering* algorithm if you want to create cluster-based features.

Merged Features Renderer

The *Merged Features* renderer allows area and line features to be "dissolved" into a single object prior to rendering to ensure that complex symbols or overlapping features are represented by a uniform and contiguous cartographic symbol.

Inverted Polygon Renderer

The *Inverted Polygon* renderer allows user to define a symbol to fill in outside of the layer's polygons. As above you can select subrenderers, namely Single symbol, Graduated, Categorized, Rule-Based or 2.5D renderer.

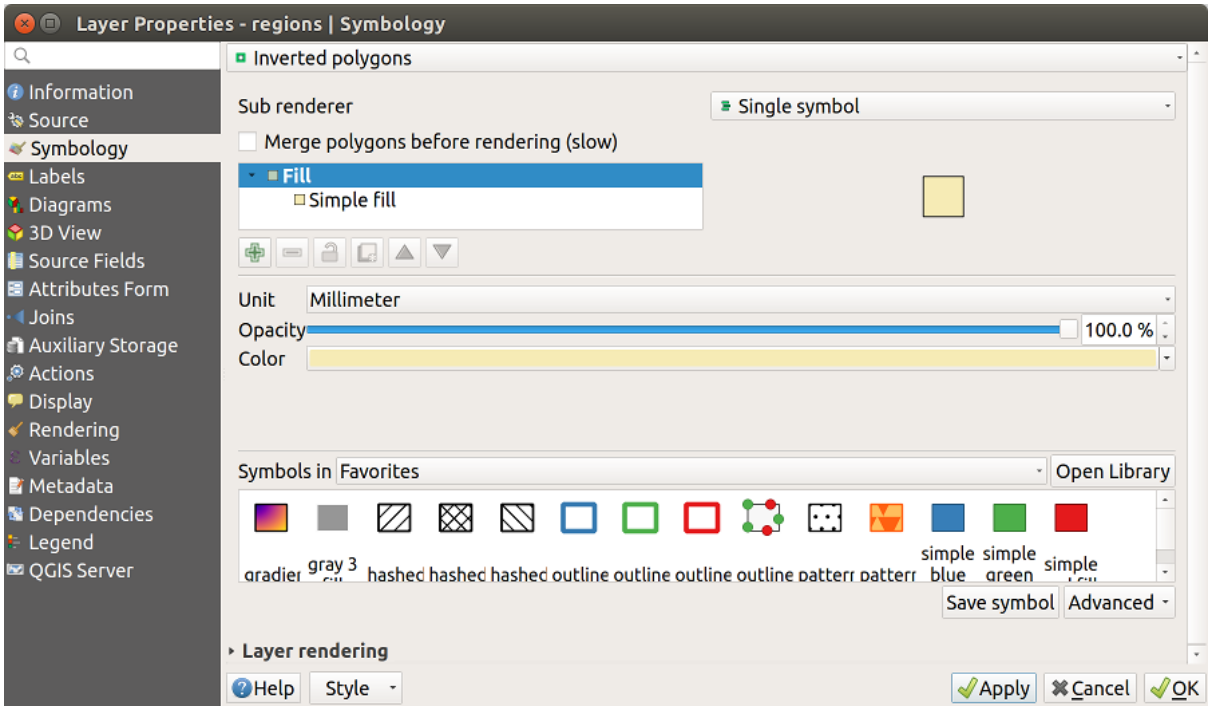


Fig. 12.10: Inverted Polygon dialog

Heatmap Renderer

With the *Heatmap* renderer you can create live dynamic heatmaps for (multi)point layers. You can specify the heatmap *Radius* in millimeters, points, pixels, map units or inches, choose and edit a *Color ramp* for the heatmap style and use a slider for selecting a trade-off between render speed and quality. You can also define a *Maximum value* limit and *Weight points by* using a field or an expression.

Use Data defined override to dynamically control *Radius* and *Maximum value* based on the attributes of your data. For example, the radius of a heatmap point could be determined by its population attribute, or the maximum value could be based on a temporal range.

When adding or removing a feature the heatmap renderer updates the heatmap style automatically. The *Color ramp* will be shown as a legend bar and in the *Legend settings* you can set the *Labels* for the *Maximum* and *Minimum* values. You can also change the orientation and direction of the legend in the *Layout*.

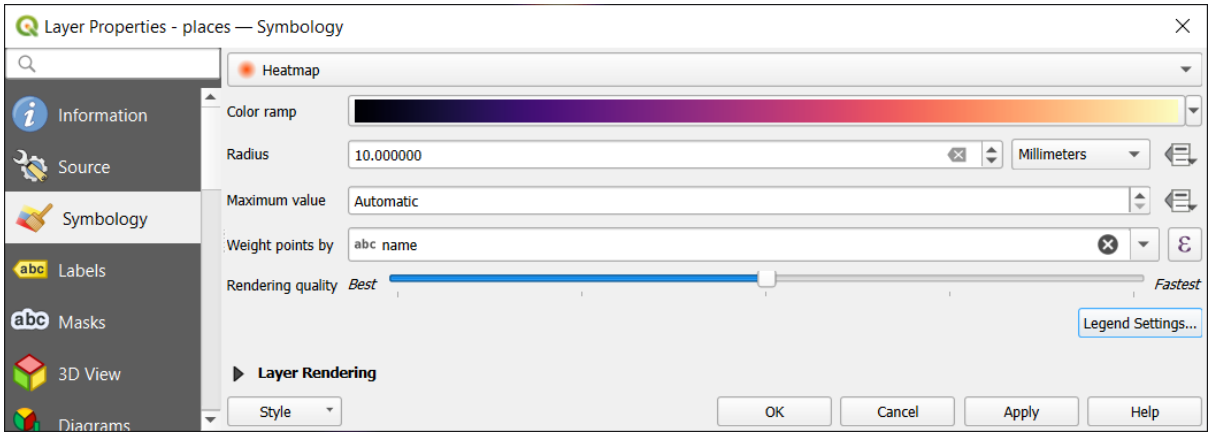


Fig. 12.11: Heatmap dialog

2.5D Renderer

Using the *2.5D* renderer it's possible to create a 2.5D effect on your layer's features. You start by choosing a *Height* value (in map units). For that you can use a fixed value, one of your layer's fields, or an expression. You also need to choose an *Angle* (in degrees) to recreate the viewer position (0° means west, growing in counter clock wise). Use advanced configuration options to set the *Roof Color* and *Wall Color*. If you would like to simulate solar radiation on the features walls, make sure to check the *Shade walls based on aspect* option. You can also simulate a shadow by setting a *Color* and *Size* (in map units).

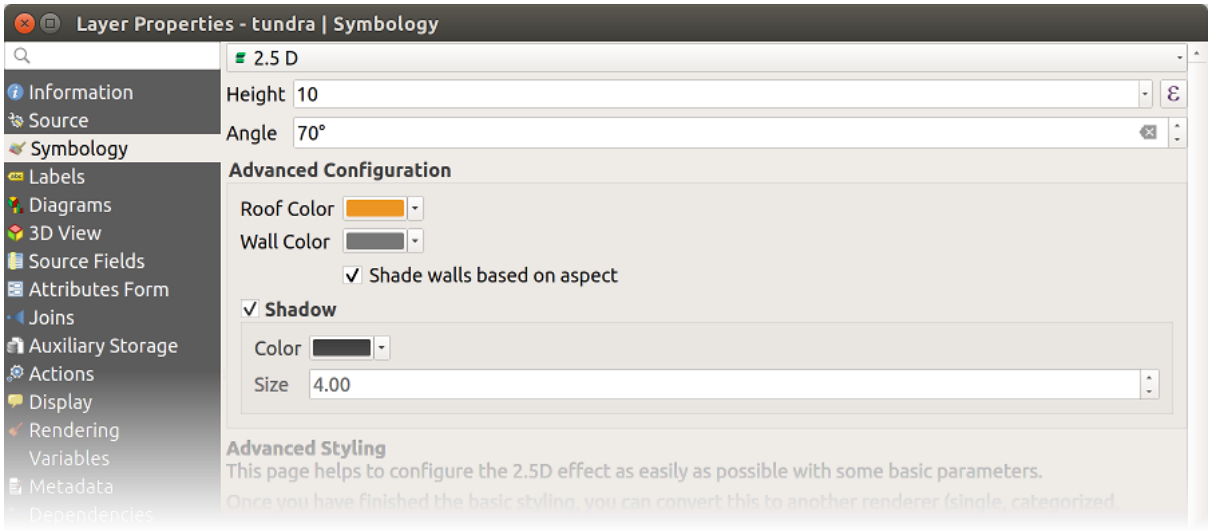


Fig. 12.12: 2.5D dialog

> **Tip**
>
> **Using 2.5D effect with other renderers**
>
> Once you have finished setting the basic style on the 2.5D renderer, you can convert this to another renderer (single, categorized, graduated). The 2.5D effects will be kept and all other renderer specific options will be available for you to fine tune them (this way you can have for example categorized symbols with a nice 2.5D representation or add some extra styling to your 2.5D symbols). To make sure that the shadow and the "building"

itself do not interfere with other nearby features, you may need to enable Symbols Levels (*Advanced* ► *Symbol levels…*). The 2.5D height and angle values are saved in the layer's variables, so you can edit it afterwards in the variables tab of the layer's properties dialog.

Embedded Renderer

The *Embedded Symbols* renderer allows to display the 'native' symbology of a provided datasource. This is mostly the case with `KML` and `TAB` datasets that have predefined symbology.

Layer rendering

From the Symbology tab, you can also set some options that invariably act on all features of the layer:

- *Opacity*  : You can make the underlying layer in the map canvas visible with this tool. Use the slider to adapt the visibility of your vector layer to your needs. You can also make a precise definition of the percentage of visibility in the menu beside the slider.
- *Blending mode* at the *Layer* and *Feature* levels: You can achieve special rendering effects with these tools that you may previously only know from graphics programs. The pixels of your overlaying and underlaying layers are mixed through the settings described in *Blending Modes*.
- Apply *paint effects* on all the layer features with the *Draw Effects* button.
- *Control feature rendering order* allows you, using features attributes, to define the z-order in which they shall be rendered. Activate the checkbox and click on the  button beside. You then get the *Define Order* dialog in which you:

  1. Choose a field or build an expression to apply to the layer features.
  2. Set in which order the fetched features should be sorted, i.e. if you choose **Ascending** order, the features with lower value are rendered under those with higher value.
  3. Define when features returning NULL value should be rendered: **first** (bottom) or **last** (top).
  4. Repeat the above steps as many times as rules you wish to use.

  The first rule is applied to all the features in the layer, z-ordering them according to their returned value. Then, within each group of features with the same value (including those with NULL value) and thus the same z-level, the next rule is applied to sort them. And so on…

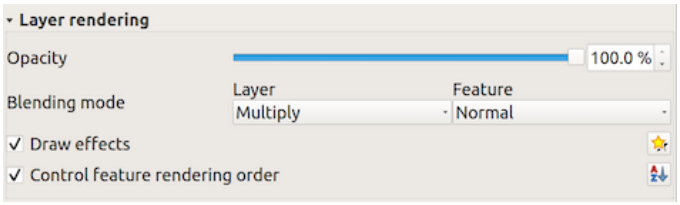


Fig. 12.13: Layer rendering options

Other Settings

Symbol levels

For renderers that allow stacked symbol layers (only heatmap doesn't) there is an option to control the rendering order of each symbol's levels.

For most of the renderers, you can access the Symbols levels option by clicking the *Advanced* button below the saved symbols list and choosing *Symbol levels*. For the *Rule-based Renderer* the option is directly available through *Symbols Levels…* button, while for *Point displacement Renderer* renderer the same button is inside the *Rendering settings* dialog.

To activate symbols levels, select the *Enable symbol levels*. Each row will show up a small sample of the combined symbol, its label and the individual symbols layer divided into columns with a number next to it. The numbers represent the rendering order level in which the symbol layer will be drawn. Lower values levels are drawn first, staying at the bottom, while higher values are drawn last, on top of the others.

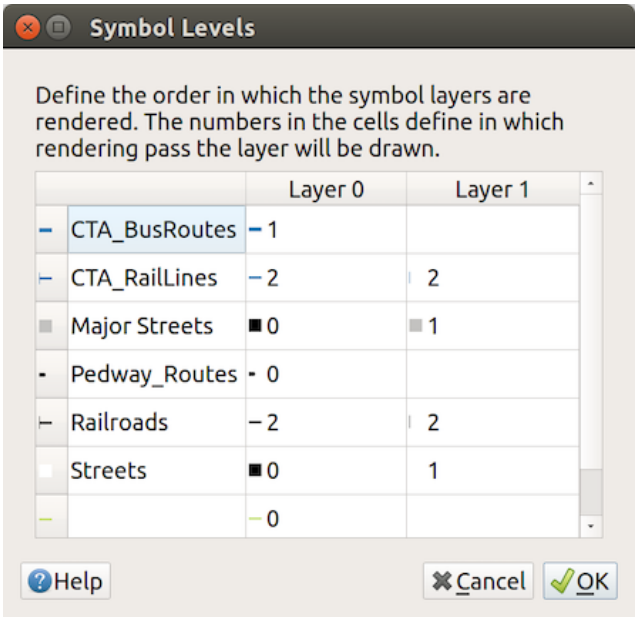


Fig. 12.14: Symbol levels dialog

Note

If symbols levels are deactivated, the complete symbols will be drawn according to their respective features order. Overlapping symbols will simply obfuscate to other below. Besides, similar symbols won't "merge" with each other.

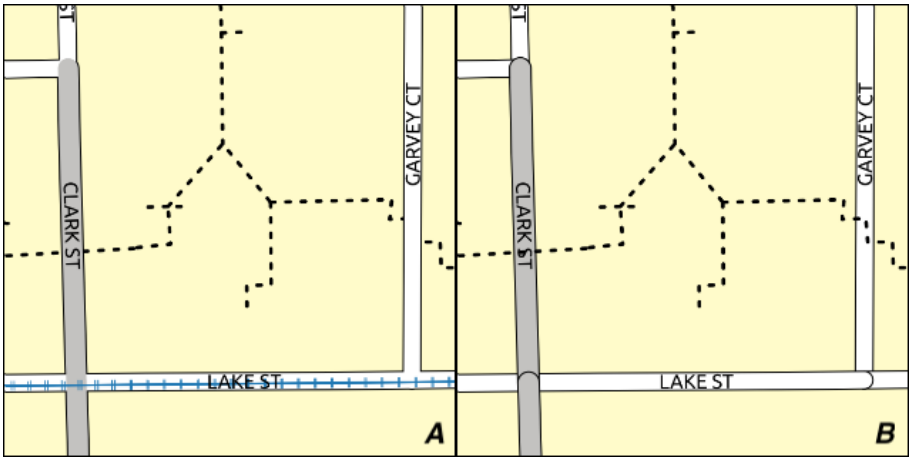


Fig. 12.15: Symbol levels activated (A) and deactivated (B) difference

Data-defined size legend

When a layer is rendered with the *proportional symbol or the multivariate rendering* or when a *scaled size diagram* is applied to the layer, you can allow the display of the scaled symbols in both the *Layers panel* and the *print layout legend*.

To enable the *Data-defined Size Legend* dialog to render symbology, select the eponym option in the *Advanced* button below the saved symbols list. For diagrams, the option is available under the *Legend* tab. The dialog provides the following options to:

- select the type of legend: *Legend not enabled*, *Separated legend items* and *Collapsed legend*. For the latter option, you can select whether the legend items are aligned at the **Bottom** or at the **Center**;
- preview the *symbol to use* for legend representation;
- insert the title in the legend;
- resize the classes to use: by default, QGIS provides you with a legend of five classes (based on natural pretty breaks) but you can apply your own classification using the *Manual size classes* option. Use the and buttons to set your custom classes values and labels.
- For collapsed legend, it's possible to:
  - *Align symbols* in the center or the bottom
  - configure the horizontal leader *Line symbol* from the symbol to the corresponding legend text.

A preview of the legend is displayed in the right panel of the dialog and updated as you set the parameters.

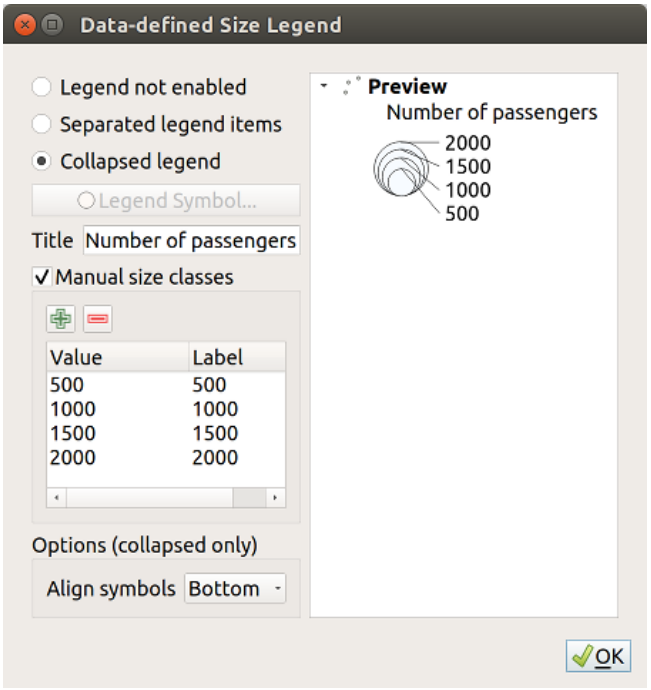


Fig. 12.16: Setting size scaled legend

Note

Currently, data-defined size legend for layer symbology can only be applied to point layer using single, categorized or graduated symbology.

Animation settings

To allow any symbol to become an *animated symbol*, you can utilize *Animation settings* panel. In this panel, you can enable animation for the symbol and set a specific frame rate for the symbol's redrawing.

1. Start by going to the top symbol level and select *Advanced* menu in the bottom right of the dialog
2. Find *Animation settings* option
3. Check *Is Animated* to enable animation for the symbol
4. Configure the *Frame rate*, i.e. how fast the animation would be played
5. You can now use `@symbol_frame` variable in any sub-symbol data defined property in order to animate that property.

For example, setting the symbol's rotation to data defined expression `@symbol_frame % 360` will cause the symbol to rotate over time, with rotation speed dictated by the symbol's frame rate:

Extent buffer

You may set an extent buffer for a *symbol*. This means that a buffer is applied to the current map extent so that if a feature is outside of the actual map extent but inside the buffered extent it will still be rendered. This is useful for example with symbols which use the geometry generator where you would like to still see the generated geometries even if the actual feature is outside of the map extent.

To edit the extent buffer you can utilize the *Extent buffer* panel.

1. Start by going to the top symbol level and select *Advanced* menu in the bottom right of the dialog
2. Find *Extent buffer* option
3. In the new panel you can set the buffer distance

The buffer distance units can be changed. You can also control the distance value by using the data defined override widget. For example you can change the value based on the current map scale `if(@map_scale > 50000, 5000, 0)`:

Draw effects

In order to improve layer rendering and avoid (or at least reduce) the resort to other software for final rendering of maps, QGIS provides another powerful functionality: the *Draw Effects* options, which adds paint effects for customizing the visualization of vector layers.

The option is available in the *Layer Properties* ► *Symbology* dialog, under the *Layer rendering* group (applying to the whole layer) or in *symbol layer properties* (applying to corresponding features). You can combine both usage.

Paint effects can be activated by checking the *Draw effects* option and clicking the Customize effects button. That will open the *Effect Properties* Dialog (see Fig. 12.17). The following effect types, with custom options are available:

- **Source**: Draws the feature's original style according to the configuration of the layer's properties. The *Opacity* of its style can be adjusted as well as the *Blend mode* and *Draw mode*. These are common properties for all types of effects.

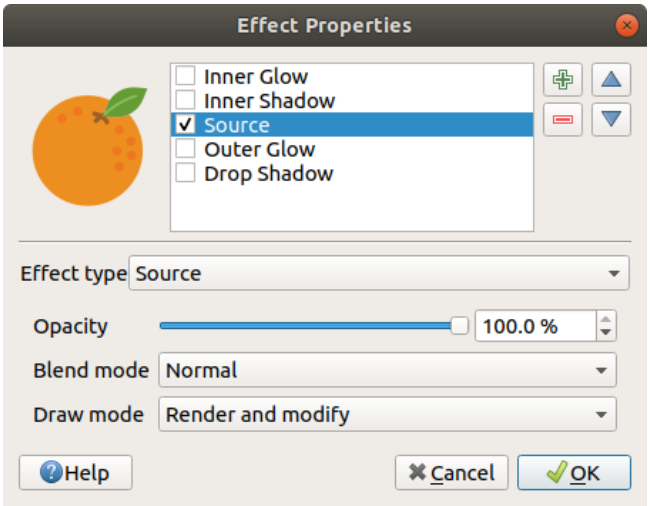


Fig. 12.17: Draw Effects: Source dialog

- **Blur**: Adds a blur effect on the vector layer. The custom options that you can change are the *Blur type* (*Stack blur (fast)* or *Gaussian blur (quality)*) and the *Blur strength*.

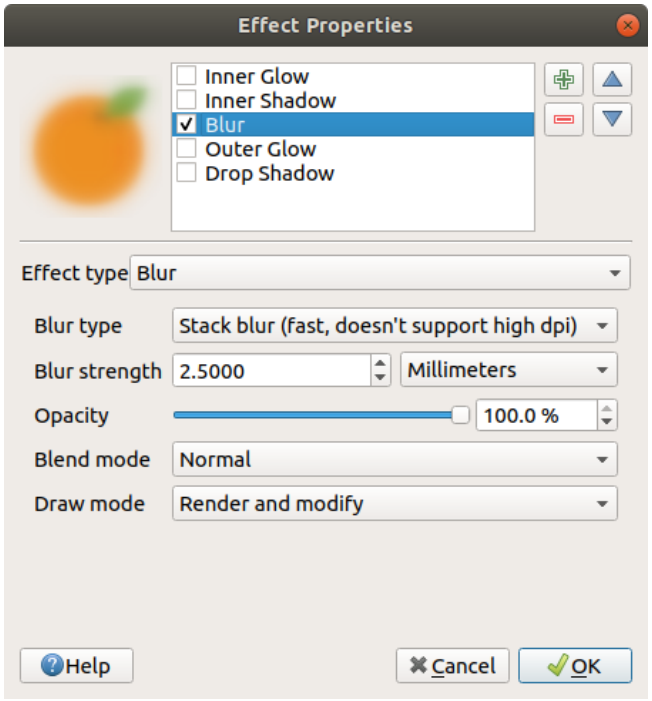


Fig. 12.18: Draw Effects: Blur dialog

- **Colorise**: This effect can be used to make a version of the style using one single hue. The base will always be a grayscale version of the symbol and you can:
  - Use the  *Grayscale* to select how to create it: options are 'By lightness', 'By luminosity', 'By average' and 'Off'.
  - If  *Colorise* is selected, it will be possible to mix another color and choose how strong it should be.
  - Control the *Brightness*, *Contrast* and *Saturation* levels of the resulting symbol.

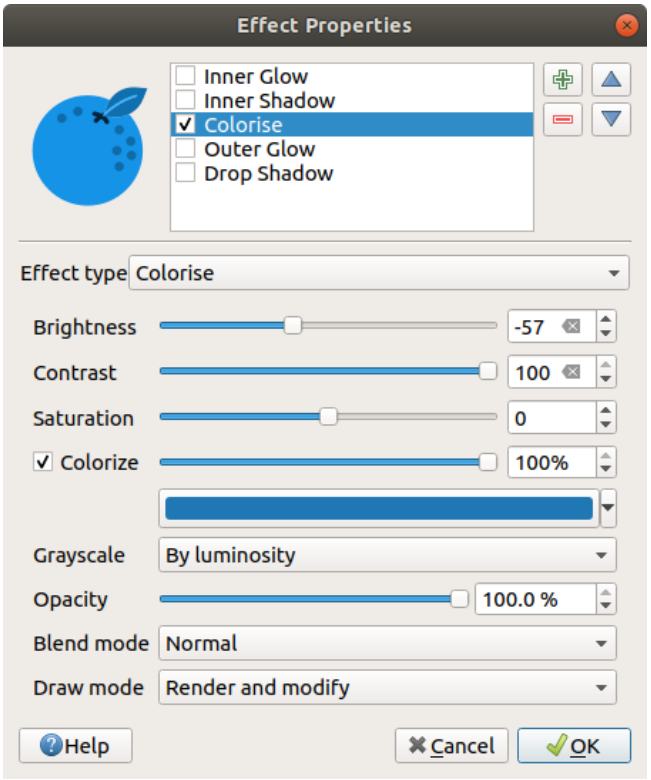


Fig. 12.19: Draw Effects: Colorize dialog

- **Drop Shadow**: Using this effect adds a shadow on the feature, which looks like adding an extra dimension. This effect can be customized by changing the *Offset* angle and distance, determining where the shadow shifts towards to and the proximity to the source object. *Drop Shadow* also has the option to change the *Blur radius* and the *Color* of the effect.

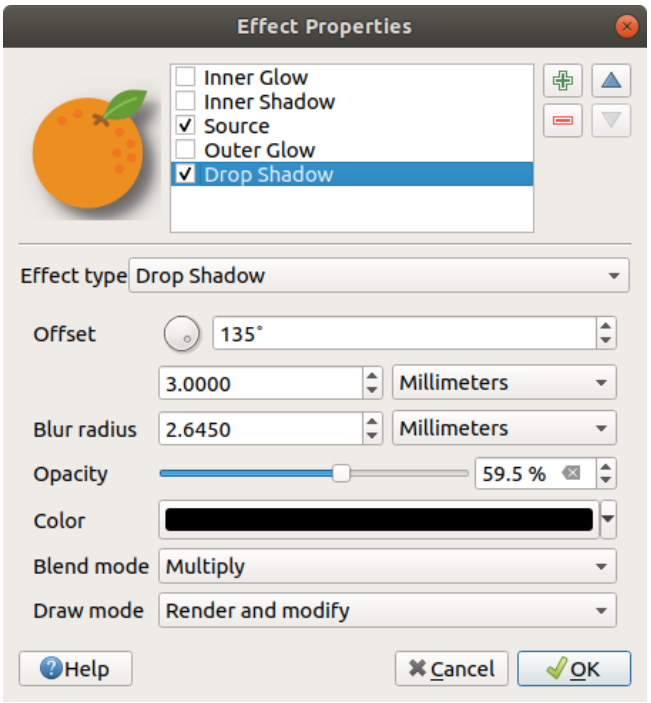


Fig. 12.20: Draw Effects: Drop Shadow dialog

- **Inner Shadow**: This effect is similar to the *Drop Shadow* effect, but it adds the shadow effect on the inside of the edges of the feature. The available options for customization are the same as the *Drop Shadow* effect.

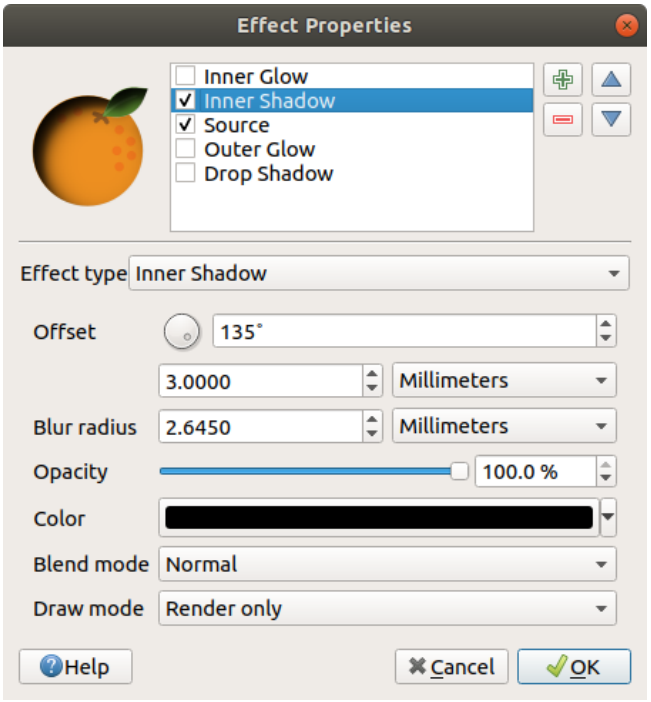


Fig. 12.21: Draw Effects: Inner Shadow dialog

- **Inner Glow**: Adds a glow effect inside the feature. This effect can be customized by adjusting the *Spread* (width) of the glow, or the *Blur radius*. The latter specifies the proximity from the edge of the feature where you want any blurring to happen. Additionally, there are options to customize the color of the glow using a *Single color* or a *Color ramp*.

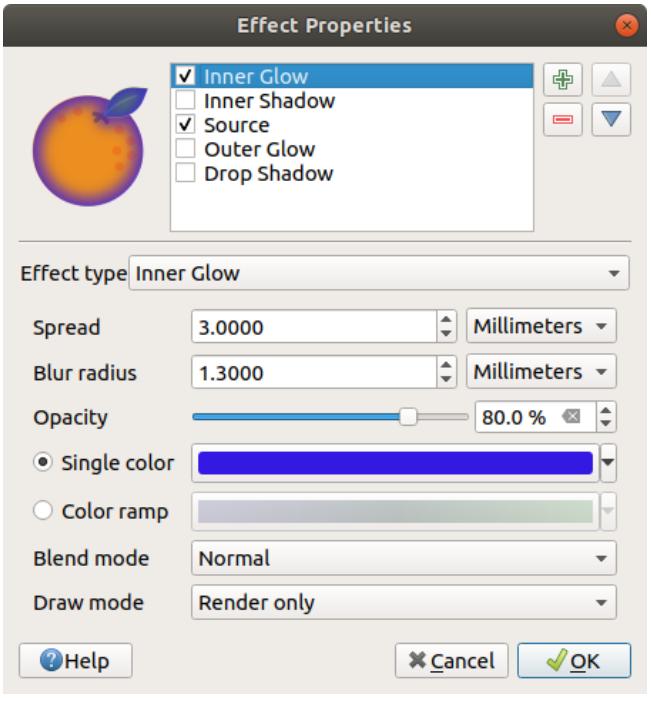


Fig. 12.22: Draw Effects: Inner Glow dialog

- **Outer Glow**: This effect is similar to the *Inner Glow* effect, but it adds the glow effect on the outside of the edges of the feature. The available options for customization are the same as the *Inner Glow* effect.

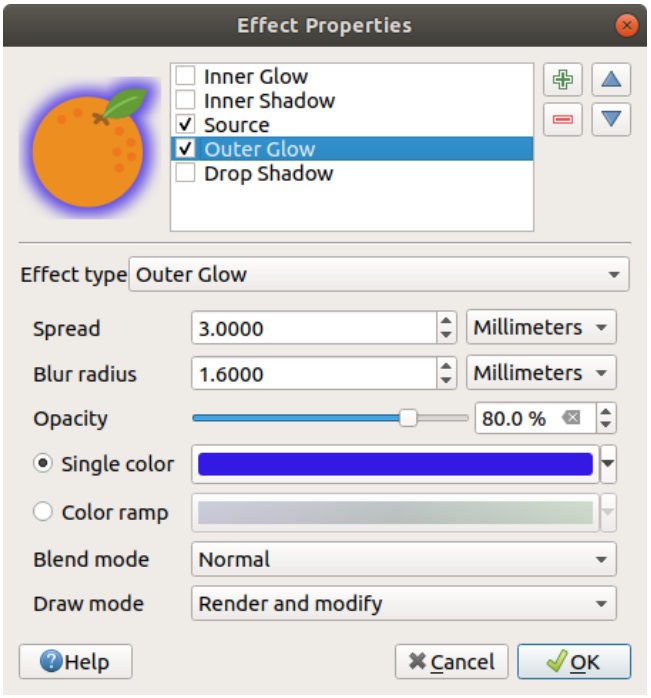


Fig. 12.23: Draw Effects: Outer Glow dialog

- **Transform**: Adds the possibility of transforming the shape of the symbol. The first options available for customization are the *Reflect horizontal* and *Reflect vertical*, which actually create a reflection on the horizontal and/or vertical axes. The other options are:
  - *Shear X,Y*: Slants the feature along the X and/or Y axis.
  - *Scale X,Y*: Enlarges or minimizes the feature along the X and/or Y axis by the given percentage.
  - *Rotation*: Turns the feature around its center point.
  - and *Translate X,Y* changes the position of the item based on a distance given on the X and/or Y axis.

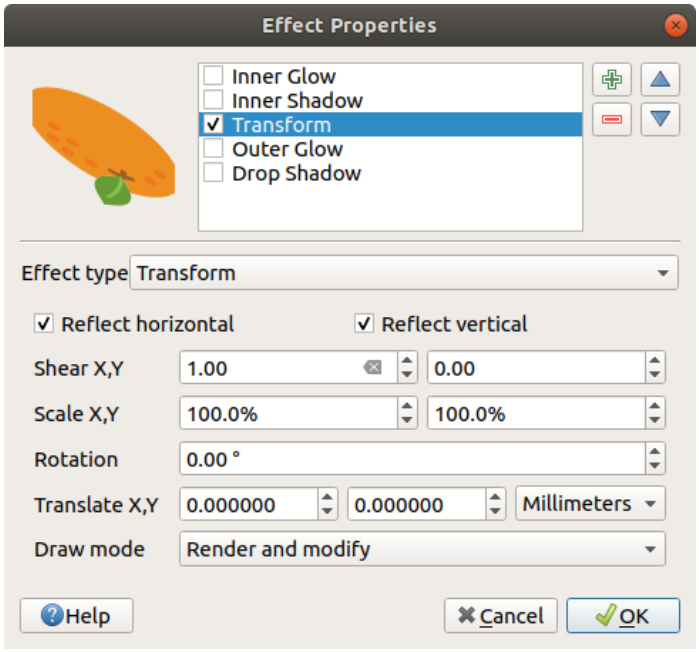


Fig. 12.24: Draw Effects: Transform dialog

One or more effect types can be used at the same time. You (de)activate an effect using its checkbox in the effects list. You can change the selected effect type by using the *Effect type* option. You can reorder the effects using Move up and Move down buttons, and also add/remove effects using the Add new effect and Remove effect buttons.

There are some common options available for all draw effect types. *Opacity* and *Blend mode* options work similar to the ones described in *Layer rendering* and can be used in all draw effects except for the transform one.

There is also a *Draw mode* option available for every effect, and you can choose whether to render and/or modify the symbol, following some rules:

- Effects render from top to bottom.
- *Render only* mode means that the effect will be visible.
- *Modifier only* mode means that the effect will not be visible but the changes that it applies will be passed to the next effect (the one immediately below).
- The *Render and Modify* mode will make the effect visible and pass any changes to the next effect. If the effect is at the top of the effects list or if the immediately above effect is not in modify mode, then it will use the original source symbol from the layers properties (similar to source).

12.1.4 Labels Properties

The *Labels* properties provides you with all the needed and appropriate capabilities to configure smart labeling on vector layers. This dialog can also be accessed from the *Layer Styling* panel, or using the Layer Labeling Options button of the **Labels toolbar**.

At the top of the dialog, you have:

- a combobox for selecting the appropriate labeling method for the active layer
- the Configure project labeling rules button: helps you control interactions between labels and features across the layers in the project. More details at *Configuring project labeling rules*.
- the Automated placement settings (applies to all layers) button: configure general properties on label placement and conflicts resolution. More details at *Setting the automated placement engine*.

The first step is to choose the labeling method from the drop-down list. Available methods are:

- *No labels*: the default value, showing no labels from the layer
- *Single labels*: Show labels on the map using a single attribute or an expression
- *Rule-based labeling*
- and *Blocking*: allows to set a layer as just an obstacle for other layer's labels without rendering any labels of its own.

The next steps assume you select the *Single labels* option, opening the following dialog.

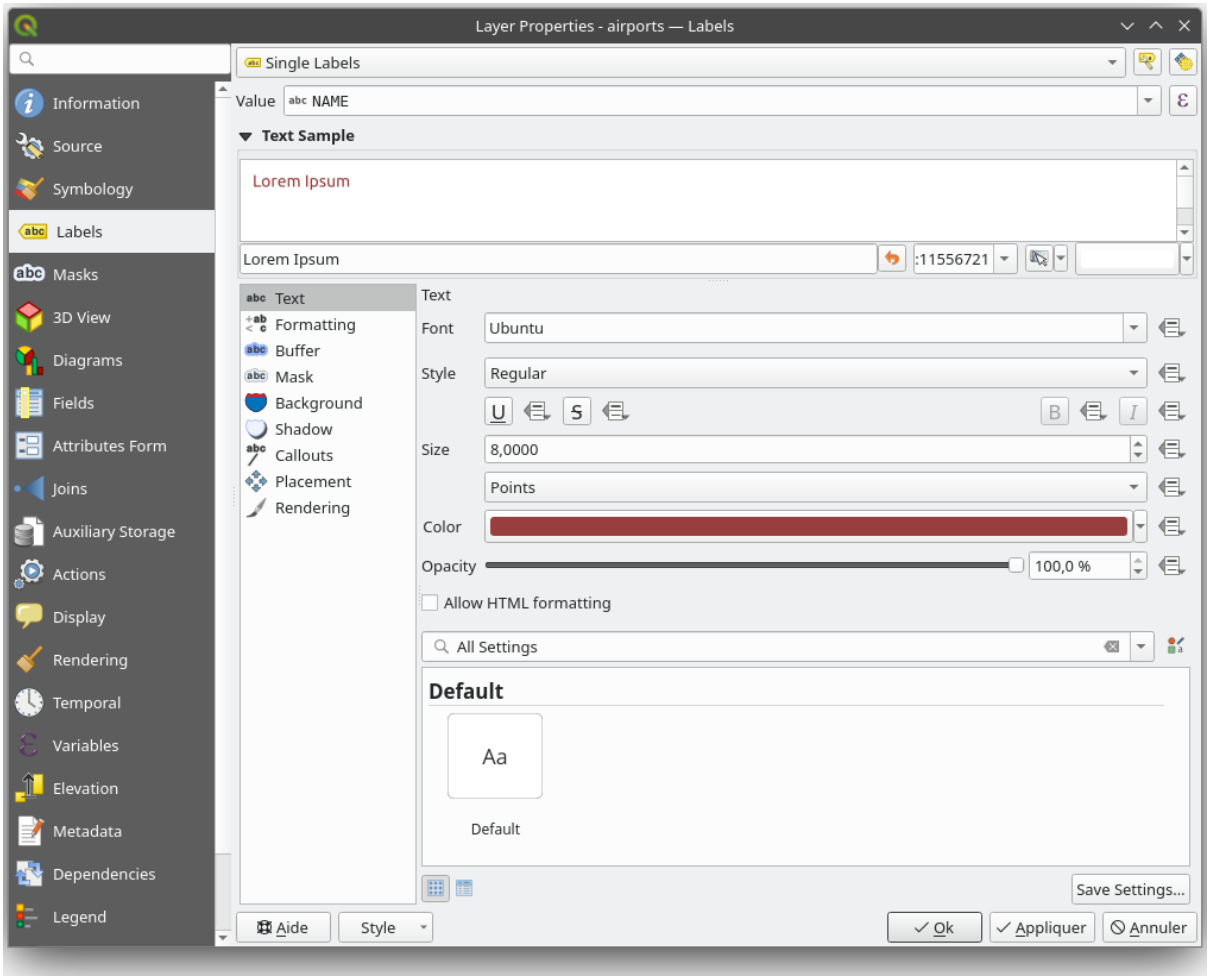


Fig. 12.25: Layer labeling settings - Single labels

At the top of the dialog, a *Value* drop-down list is enabled. You can select an attribute column to use for labeling. By default, the *display field* is used. Click ε if you want to define labels based on expressions - See *Define labels based on expressions*.

> **Note**
>
> Labels with their formatting can be displayed as entries in the legends, if enabled in the *Legend* tab.

Below are displayed options to customize the labels, under various tabs:

- abc *Text*

- *Formatting*
- *Buffer*
- *Mask*
- *Background*
- *Shadow*
- *Callouts*
- *Placement*
- *Rendering*

Description of how to set each property is exposed at *Setting a label*.

Configuring project labeling rules

Pressing the Configure project labeling rules button next to the labeling method drop-down selector, you can create rules that controls how labels from a layer can interact with labels or features from another layer.

1. Press the Add rule button and in the drop-down menu, select one of the rule types:
   - *Prevent labels overlapping features*: prevents labels being placed overlapping features from a different layer.
   - *Pull labels towards features*: prevents labels being placed too far from features from a different layer. The maximum distance can be set in the unit of your choice.
   - *Push labels away from features*: prevents labels being placed too close to features from a different layer. The minimum distance can be set in the unit of your choice, as well as the *rule's priority* (The highest-priority rules are more important to respect in the event of a label placement conflict).
   - *Push labels away from other labels*: prevents labels being placed too close to labels from a different layer.

   Attention

   Because the last three require a build based on GEOS >= 3.10, they may not be available on your QGIS installation depending on the underlying GEOS version in use.

2. Fill the properties at your will; you can provide a more meaningful name to the rule.
3. Press *OK*.
4. Add as many rules as necessary.
5. If necessary, press Edit rule to modify the selected rule or Remove rule to delete it from the project.

The set rules are available from any layer *Labels* properties tab, pressing the Configure project labeling rules button. You can temporary enable or disable any of them, using the checkbox next to the name. Hover over a rule to preview its details.

Setting the automated placement engine

You can use the automated placement settings to configure a project-level automated behavior of the labels. In the top right corner of the *Labels* tab, click the Automated placement settings (applies to all layers) button, opening a dialog with the following options:

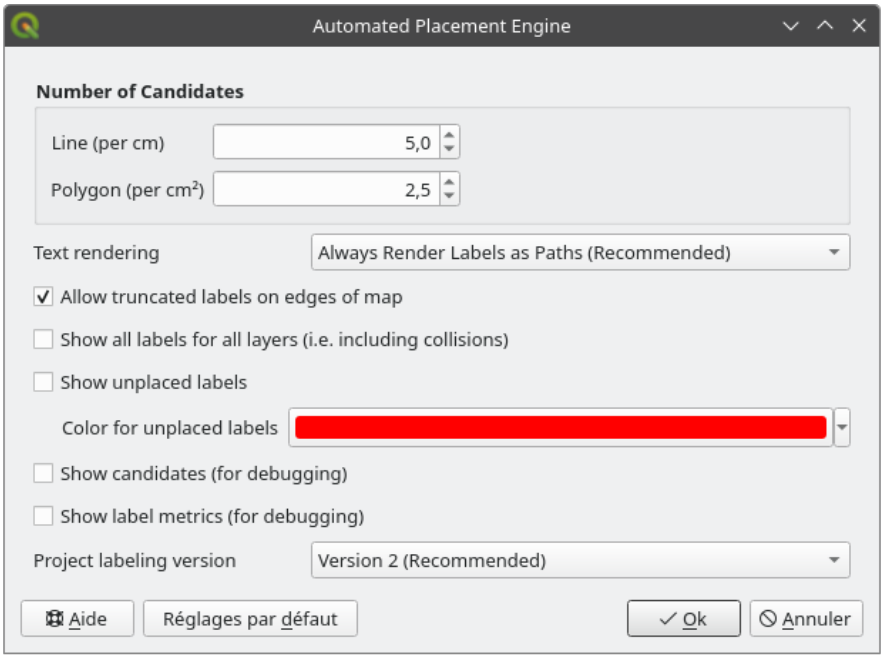


Fig. 12.26: The labels automated placement engine

- *Number of candidates*: calculates and assigns to line and polygon features the number of possible labels placement based on their size. The longer or wider a feature is, the more candidates it has, and its labels can be better placed with less risk of collision.
- *Text rendering*: sets the default value for label rendering widgets when *exporting a map canvas* or *a layout* to PDF or SVG. If *Always render labels as text* is selected then labels can be edited in external applications (e.g. Inkscape) as normal text. BUT the side effect is that the rendering quality is decreased, and there are issues with rendering when certain text settings like buffers are in place. That's why *Always render labels as paths (recommended)* which exports labels as outlines but guarantees complete compatibility with the full range of formatting options available, is recommended. With *Prefer rendering labels as text* option, labels are rendered as text objects, unless doing so results in rendering artifacts or poor quality rendering (depending on text format settings).

> **Note**
>
> When rendering labels as text to a vector based device (e.g. PDF or SVG), care must be taken to ensure that the required fonts are available to users when opening the created files, or default fallback fonts will be used to display the output instead. (Although PDF exports MAY automatically embed some fonts when possible, depending on the user's platform).

- *Allow truncated labels on edges of map*: controls whether labels which fall partially outside of the map extent should be rendered. If checked, these labels will be shown (when there's no way to place them fully within the visible area). If unchecked then partially visible labels will be skipped. Note that this setting has no effects on labels' display in the *layout map item*.
- *Show all labels for all layers (i.e. including colliding objects)*. Note that this option can be also set per layer (see *Rendering tab*)

- *Show unplaced labels*: allows to determine whether any important labels are missing from the maps (e.g. due to overlaps or other constraints). They are displayed using a customizable color.
- *Show candidates (for debugging)*: controls whether boxes should be drawn on the map showing all the candidates generated for label placement. Like the label says, it's useful only for debugging and testing the effect different labeling settings have. This could be handy for a better manual placement with tools from the *label toolbar*.
- *Show label metrics (for debugging)*: displays the text bounds of the label in red and baselines in blue
- *Project labeling version*: QGIS supports two different versions of label automatic placement:
  - *Version 1*: the old system (used by QGIS versions 3.10 and earlier, and when opening projects created in these versions in QGIS 3.12 or later). Version 1 treats label and obstacle priorities as "rough guides" only, and it's possible that a low-priority label will be placed over a high-priority obstacle in this version. Accordingly, it can be difficult to obtain the desired labeling results when using this version and it is thus recommended only for compatibility with older projects.
  - *Version 2 (recommended)*: this is the default system in new projects created in QGIS 3.12 or later. In version 2, the logic dictating when labels are allowed to overlap *obstacles* has been reworked. The newer logic forbids any labels from overlapping any obstacles with a greater obstacle weight compared to the label's priority. As a result, this version results in much more predictable and easier to understand labeling results.

Rule-based labeling

With rule-based labeling multiple label configurations can be defined and applied selectively on the base of expression filters and scale range, as in *Rule-based rendering*.

To create a rule:

1. Select the **Rule-based labeling** option in the main drop-down list from the *Labels* tab
2. Click the Add rule button at the bottom of the dialog.
3. Fill the new dialog with:
   - *Description*: a text used to identify the rule in the *Labels* tab and as a *label legend entry* in the print layout legend
   - *Filter*: an expression to select the features to apply the label settings to
   - If there are rules already set, the *Else* option can be used to select all the features not matching any filter of the rules in the same group.
4. You can set a *scale range* in which the label rule should be applied.
5. The options available under the *Labels* group box are the usual *label settings*. Configure them and press *OK*.

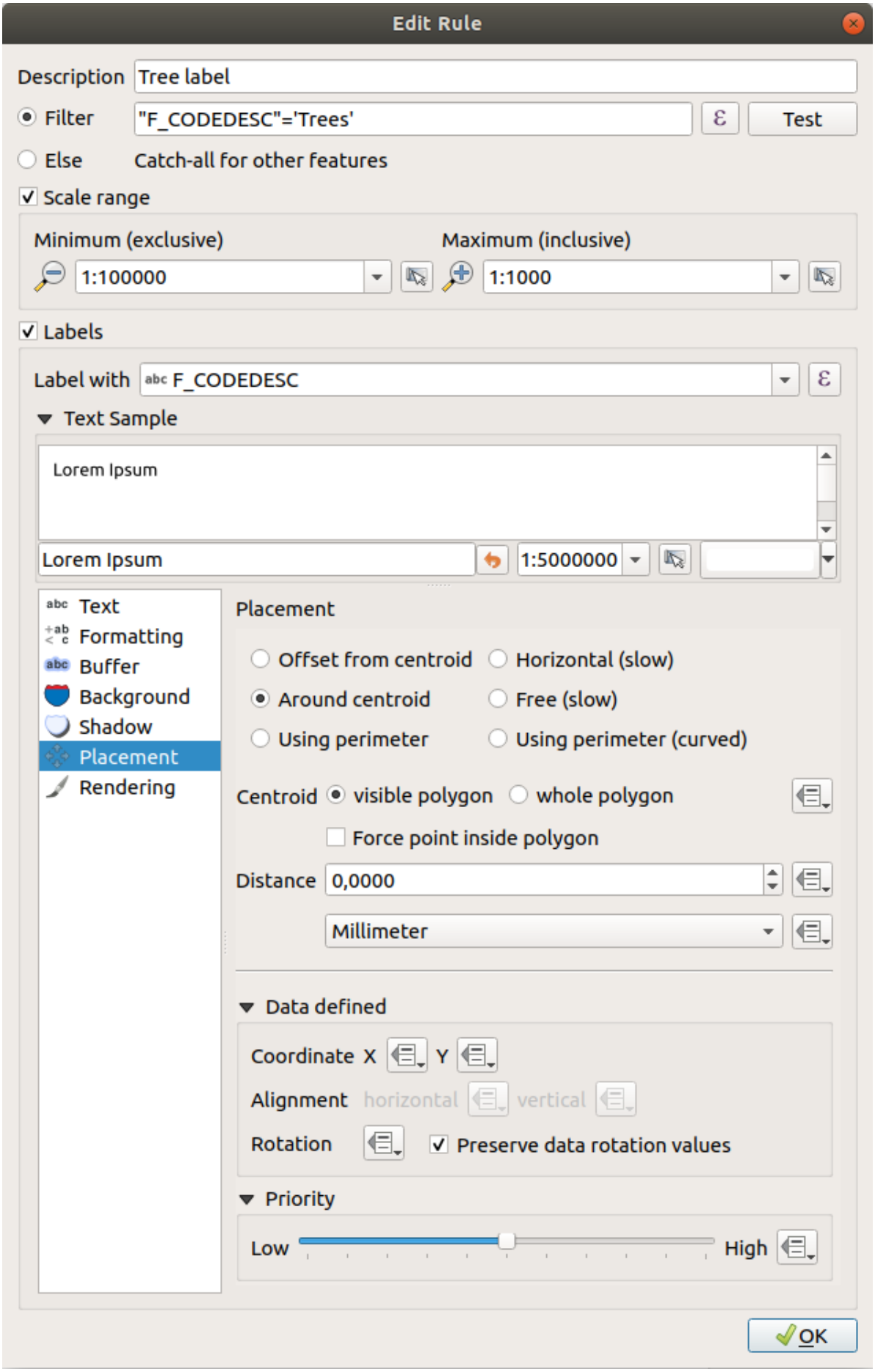


Fig. 12.27: Rule settings

A summary of existing rules is shown in the main dialog (see Fig. 12.28). You can add multiple rules, reorder or imbricate them with a drag-and-drop. You can as well remove them with the button or edit them with button or a double-click.

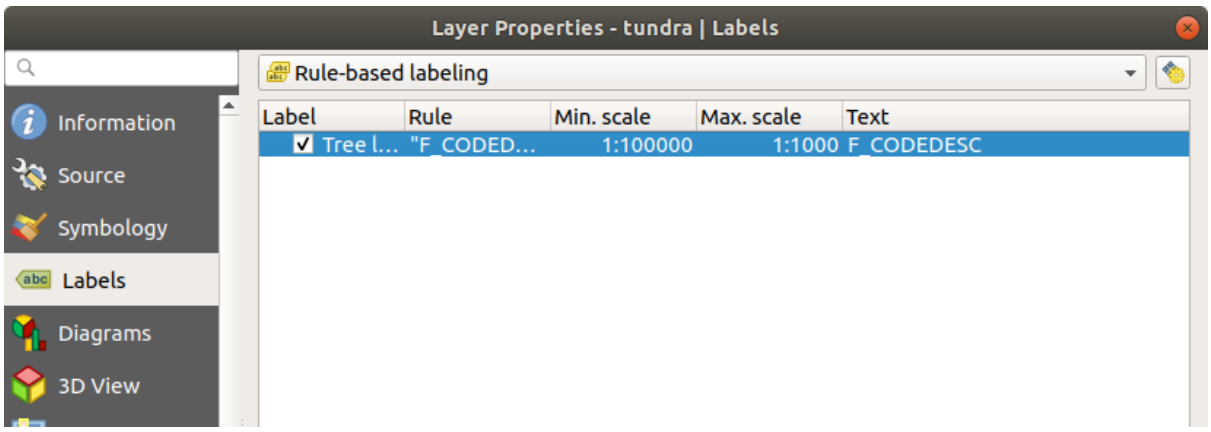


Fig. 12.28: Rule based labeling panel

Define labels based on expressions

Whether you choose single or rule-based labeling type, QGIS allows using expressions to label features.

Assuming you are using the *Single labels* method, click the button near the *Value* drop-down list in the *Labels* tab of the properties dialog.

In Fig. 12.29, you see a sample expression to label the alaska trees layer with tree type and area, based on the field 'VEGDESC', some descriptive text, and the function `$area` in combination with `format_number()` to make it look nicer.

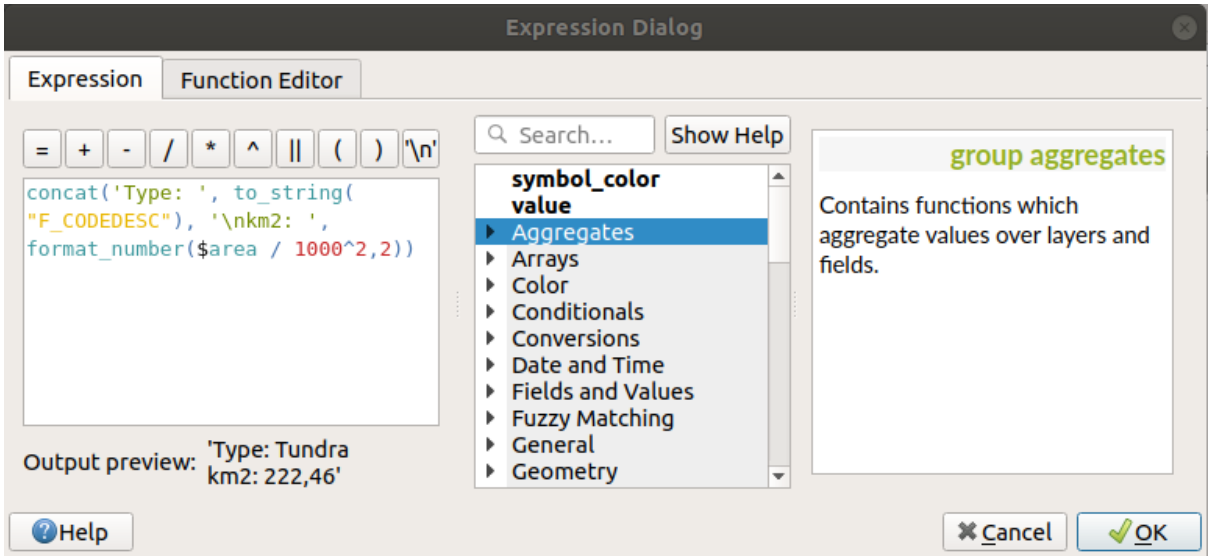


Fig. 12.29: Using expressions for labeling

Expression based labeling is easy to work with. All you have to take care of is that:

- You may need to combine all elements (strings, fields, and functions) with a string concatenation function such as `concat`, `+` or `||`. Be aware that in some situations (when null or numeric value are involved) not all of these tools will fit your need.

- Strings are written in 'single quotes'.
- Fields are written in "double quotes" or without any quote.

Let's have a look at some examples:

1. Label based on two fields 'name' and 'place' with a comma as separator:

```
"name" || ', ' || "place"
```

   Returns:

```
John Smith, Paris
```

2. Label based on two fields 'name' and 'place' with other texts:

```
'My name is ' + "name" + 'and I live in ' + "place"
'My name is ' || "name" || 'and I live in ' || "place"
concat('My name is ', name, ' and I live in ', "place")
```

   Returns:

```
My name is John Smith and I live in Paris
```

3. Label based on two fields 'name' and 'place' with other texts combining different concatenation functions:

```
concat('My name is ', name, ' and I live in ' || place)
```

   Returns:

```
My name is John Smith and I live in Paris
```

   Or, if the field 'place' is NULL, returns:

```
My name is John Smith
```

4. Multi-line label based on two fields 'name' and 'place' with a descriptive text:

```
concat('My name is ', "name", '\n' , 'I live in ' , "place")
```

   Returns:

```
My name is John Smith
I live in Paris
```

5. Label based on a field and the $area function to show the place's name and its rounded area size in a converted unit:

```
'The area of ' || "place" || ' has a size of '
|| round($area/10000) || ' ha'
```

   Returns:

```
The area of Paris has a size of 10500 ha
```

6. Create a CASE ELSE condition. If the population value in field *population* is <= 50000 it is a town, otherwise it is a city:

```
concat('This place is a ',
CASE WHEN "population" <= 50000 THEN 'town' ELSE 'city' END)
```

   Returns:

```
This place is a town
```

7. Display name for the cities and no label for the other features (for the "city" context, see example above):

```
CASE WHEN "population" > 50000 THEN "NAME" END
```

Returns:

```
Paris
```

As you can see in the expression builder, you have hundreds of functions available to create simple and very complex expressions to label your data in QGIS. See *Expressions* chapter for more information and examples on expressions.

Using data-defined override for labeling

With the Data defined override function, the settings for the labeling are overridden by entries in the attribute table or expressions based on them. This feature can be used to set values for most of the labeling options described above.

For example, using the Alaska QGIS sample dataset, let's label the `airports` layer with their name, based on their militarian `USE`, i.e. whether the airport is accessible to :

- military people, then display it in gray color, size 8;
- others, then show in blue color, size 10.

To do this, after you enabled the labeling on the `NAME` field of the layer (see *Setting a label*):

1. Activate the *Text* tab.
2. Click on the icon next to the *Size* property.
3. Select *Edit...* and type:

```
CASE
  WHEN "USE" like '%Military%' THEN 8 -- because compatible values are
↪'Military'
                                      -- and 'Joint Military/Civilian'
  ELSE 10
END
```

4. Press *OK* to validate. The dialog closes and the button becomes meaning that an rule is being run.
5. Then click the button next to the color property, type the expression below and validate:

```
CASE
  WHEN "USE" like '%Military%' THEN '150, 150, 150'
  ELSE '0, 0, 255'
END
```

Likewise, you can customize any other property of the label, the way you want. See more details on the Data-define override widget's description and manipulation in *Data defined override setup* section.

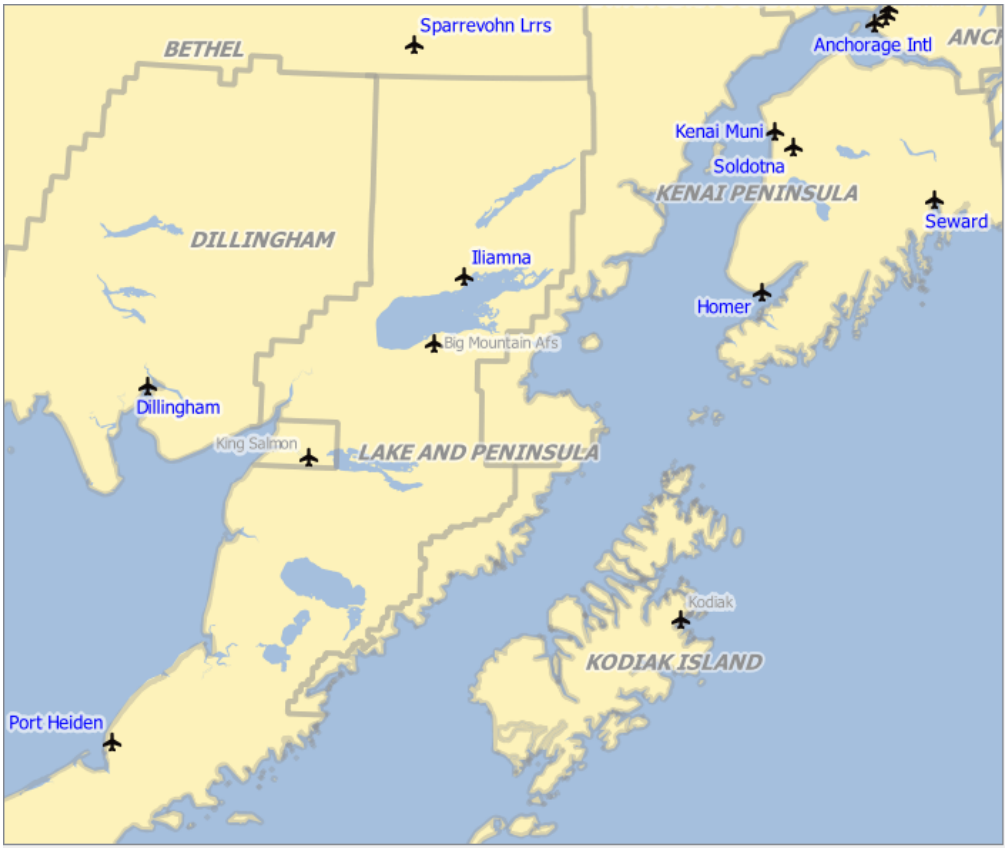


Fig. 12.30: Airports labels are formatted based on their attributes

Tip

Use the data-defined override to label every part of multi-part features

There is an option to set the labeling for multi-part features independently from your label properties. Choose the *Rendering*, `Feature options`, go to the Data-define override button next to the checkbox *Label every part of multipart-features* and define the labels as described in *Data defined override setup*.

The Label Toolbar

The *Label Toolbar* provides some tools to manipulate *label* (including their *callout*) or *diagram* properties:



Fig. 12.31: The Label toolbar

- Highlight Pinned Labels, Diagrams and Callouts. If the vector layer of the item is editable, then the highlighting is green, otherwise it's blue.
- Toggle Display of Unplaced Labels: Allows to determine whether any important labels are missing from the maps (e.g. due to overlaps or other constraints). They are displayed with a customizable color (see *Setting the automated placement engine*).

- Pin/Unpin Labels and Diagrams. By clicking or draging an area, you pin overlaid items. If you click or drag an area holding `Shift`, the items are unpinned. Finally, you can also click or drag an area holding `Ctrl` to toggle their pin status.
- Show/Hide Labels and Diagrams. If you click on the items, or click and drag an area holding `Shift`, they are hidden. When an item is hidden, you just have to click on the feature to restore its visibility. If you drag an area, all the items in the area will be restored.
- Move a Label, Diagram or Callout: click to select the item and click to move it to the desired place. The new coordinates are stored in *auxiliary fields*. Selecting the item with this tool and hitting the `Delete` key will delete the stored position value.
- Rotate a Label. Click to select the label and click again to apply the desired rotation. Likewise, the new angle is stored in an auxiliary field. Selecting a label with this tool and hitting the `Delete` key will delete the rotation value of this label.
- Change Label Properties. It opens a dialog to change the clicked label properties; it can be the label itself, its coordinates, angle, font, size, multiline alignment … as long as this property has been mapped to a field. Here you can set the option to *Label every part of a feature*.

Warning

Label tools overwrite current field values

Using the *Label toolbar* to customize the labeling actually writes the new value of the property in the mapped field. Hence, be careful to not inadvertently replace data you may need later!

Note

The *Auxiliary Storage Properties* mechanism may be used to customize labeling (position, and so on) without modifying the underlying data source.

Customize the labels from the map canvas

Combined with the *Label Toolbar*, the data defined override setting helps you manipulate labels in the map canvas (move, edit, rotate). We now describe an example using the data-defined override function for the Move Label, Diagram or Callout function (see Fig. 12.32).

1. Import `lakes.shp` from the QGIS sample dataset.
2. Double-click the layer to open the Layer Properties. Click on *Labels* and *Placement*. Select *Offset from centroid*.
3. Look for the *Data defined* entries. Click the icon to define the field type for the *Coordinate*. Choose `xlabel` for X and `ylabel` for Y. The icons are now highlighted in yellow.

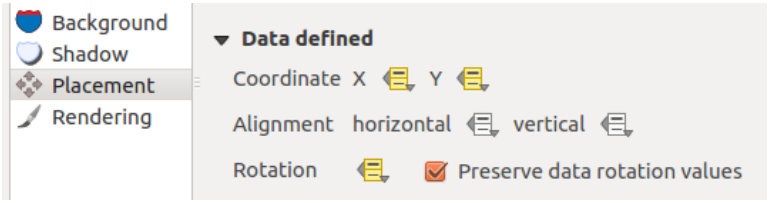


Fig. 12.32: Labeling of vector polygon layers with data-defined override

4. Zoom into a lake.
5. Set editable the layer using the Toggle Editing button.
6. Go to the Label toolbar and click the icon. Now you can shift the label manually to another position (see Fig. 12.33). The new position of the label is saved in the `xlabel` and `ylabel` columns of the attribute table.
7. It's also possible to add a line connecting each lake to its moved label using:
   - the label's *callout property*
   - or the *geometry generator symbol layer* with the expression below:

```
make_line( centroid( $geometry ), make_point( "xlabel", "ylabel" ) )
```

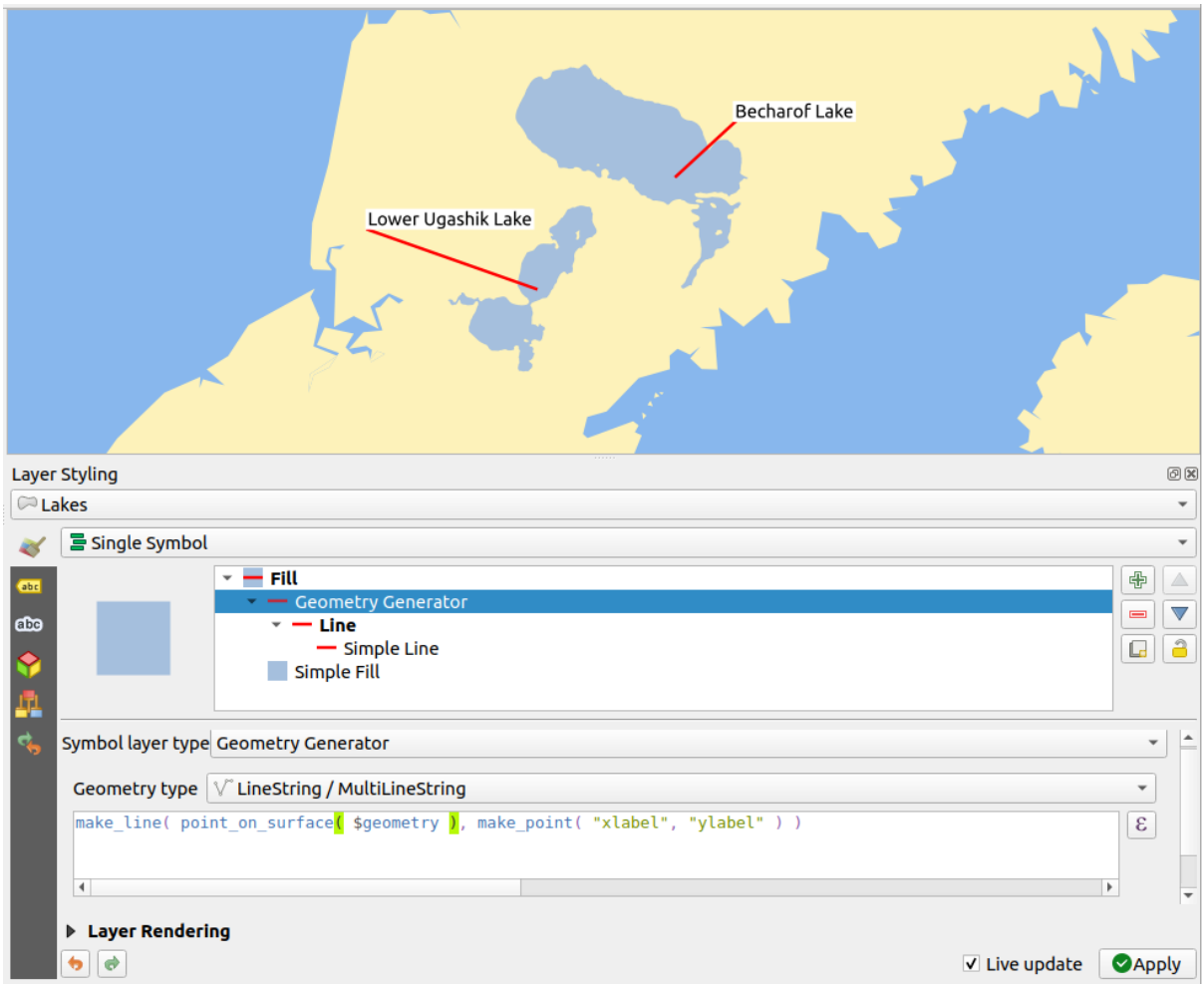


Fig. 12.33: Moved labels

Note

The *Auxiliary Storage Properties* mechanism may be used with data-defined properties without having an editable data source.

12.1.5 Diagrams Properties

The *Diagrams* tab allows you to add a graphic overlay to a vector layer (see Fig. 12.34). This dialog can also be accessed from the *Layer Styling* panel, or using the Layer Diagram Options button of the **Labels toolbar**.

The current core implementation of diagrams provides support for:

- *No diagrams*: the default value with no diagram displayed over the features;
- *Pie chart*, a circular statistical graphic divided into slices to illustrate numerical proportion. The arc length of each slice is proportional to the quantity it represents;
- *Text diagram*, a horizontally divided circle showing statistic values inside;
- *Histogram*, bars of varying colors for each attribute aligned next to each other;
- *Stacked bars*, stacks bars of varying colors for each attribute on top of each other vertically or horizontally;
- *Stacked diagram*, stacks diagrams of equal or varying types, next to each other, vertically or horizontally. More details at *Stacked Diagrams*.

In the top right corner of the *Diagrams* tab, the Automated placement settings (applies to all layers) button provides means to control diagram *labels placement* on the map canvas.

> **Tip**
>
> **Switch quickly between types of diagrams**
>
> Given that the settings are almost common to the different types of diagram, when designing your diagram, you can easily change the diagram type and check which one is more appropriate to your data without any loss.

For each type of diagram, the properties are divided into several tabs:

- *Attributes*
- *Rendering*
- *Size*
- *Placement*
- *Options*
- *Legend*

Attributes

Attributes defines which variables to display in the diagram. Use add item button to select the desired fields into the 'Assigned Attributes' panel. Generated attributes with *Expressions* can also be used.

You can move up and down any row with click and drag, sorting how attributes are displayed. You can also change the label in the 'Legend' column or the attribute color by double-clicking the item.

This label is the default text displayed in the legend of the print layout or of the layer tree.

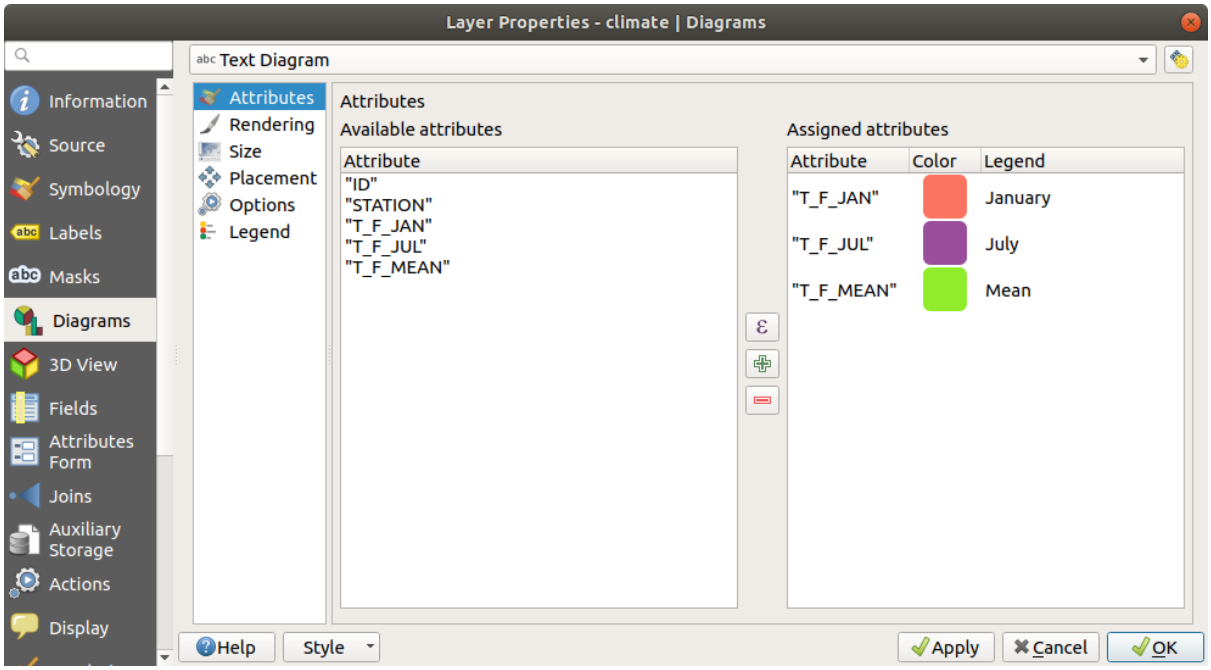


Fig. 12.34: Diagram properties - Attributes tab

Rendering

Rendering defines how the diagram looks like. It provides general settings that do not interfere with the statistic values such as:

- the graphic's opacity, its outline width and color;
- depending on the type of diagram:
  - for histogram and stacked bars, the width of the bar and the spacing between the bars. You may want to set the spacing to `0` for stacked bars. Moreover, the *Axis line symbol* can be made visible on the map canvas and customized using *line symbol properties*.
  - for text diagram, the circle background color and the *font* used for texts;
  - for pie charts, the *Start angle* of the first slice and their *Direction* (clockwise or not).
- the use of *paint effects* on the graphics.

In this tab, you can also manage and fine tune the diagram visibility with different options:

- *Diagram z-index*: controls how diagrams are drawn on top of each other and on top of labels. A diagram with a high index is drawn over other diagrams and labels;
- *Show all diagrams*: shows all the diagrams even if they overlap each other;
- *Show diagram*: allows only specific diagrams to be rendered;
- *Always Show*: selects specific diagrams to always render, even when they overlap other diagrams or map labels;
- setting the *Scale dependent visibility*;

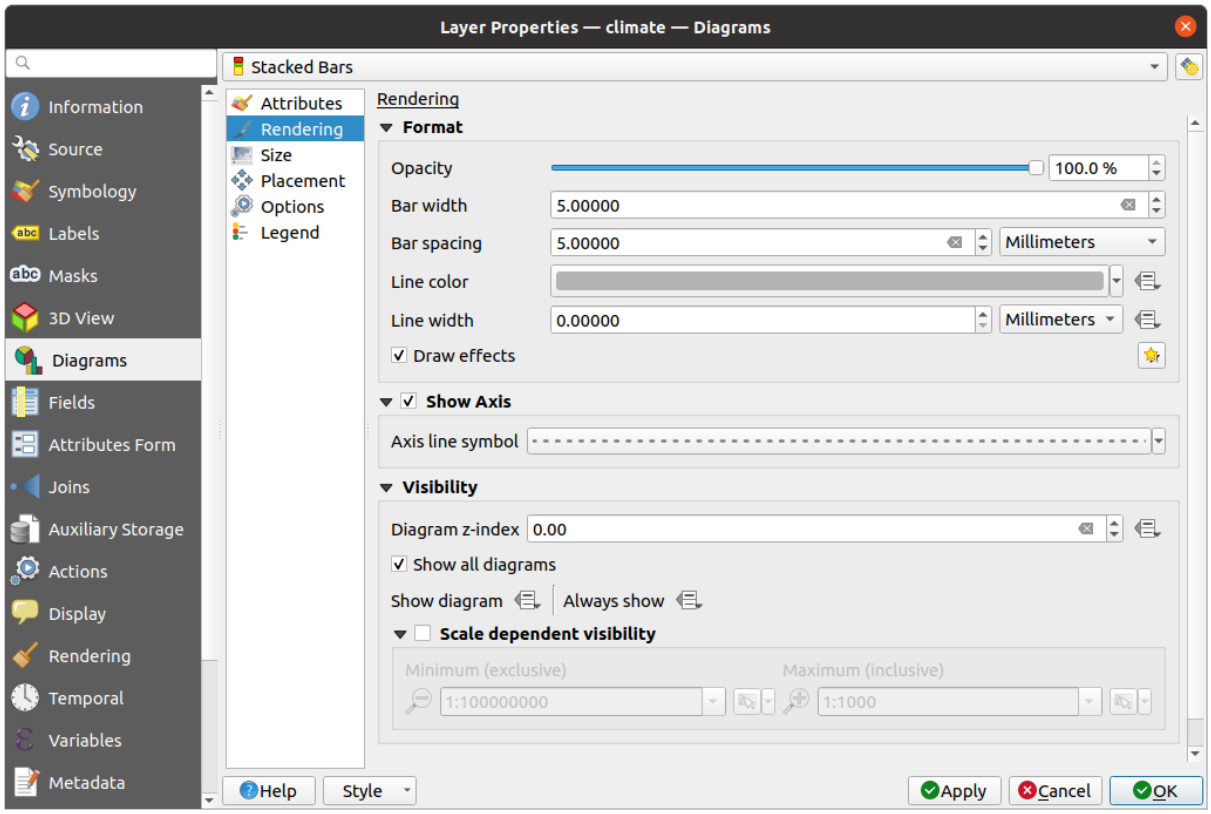


Fig. 12.35: Diagram properties - Rendering tab

Size

Size is the main tab to set how the selected statistics are represented. The diagram size *units* can be 'Millimeters', 'Points', 'Pixels', 'Map Units' or 'Inches'. You can use:

- *Fixed size*, a unique size to represent the graphic of all the features (not available for histograms)
- or *Scaled size*, based on an expression using layer attributes:

  1. In *Attribute*, select a field or build an expression
  2. Press *Find* to return the *Maximum value* of the attribute or enter a custom value in the widget.
  3. For histogram and stacked bars, enter a *Bar length* value, used to represent the *Maximum value* of the attributes. For each feature, the bar length will then be scaled linearly to keep this matching.
  4. For pie chart and text diagram, enter a *Size* value, used to represent the *Maximum value* of the attributes. For each feature, the circle area or diameter will then be scaled linearly to keep this matching (from `0`). A *Minimum size* can however be set for small diagrams.

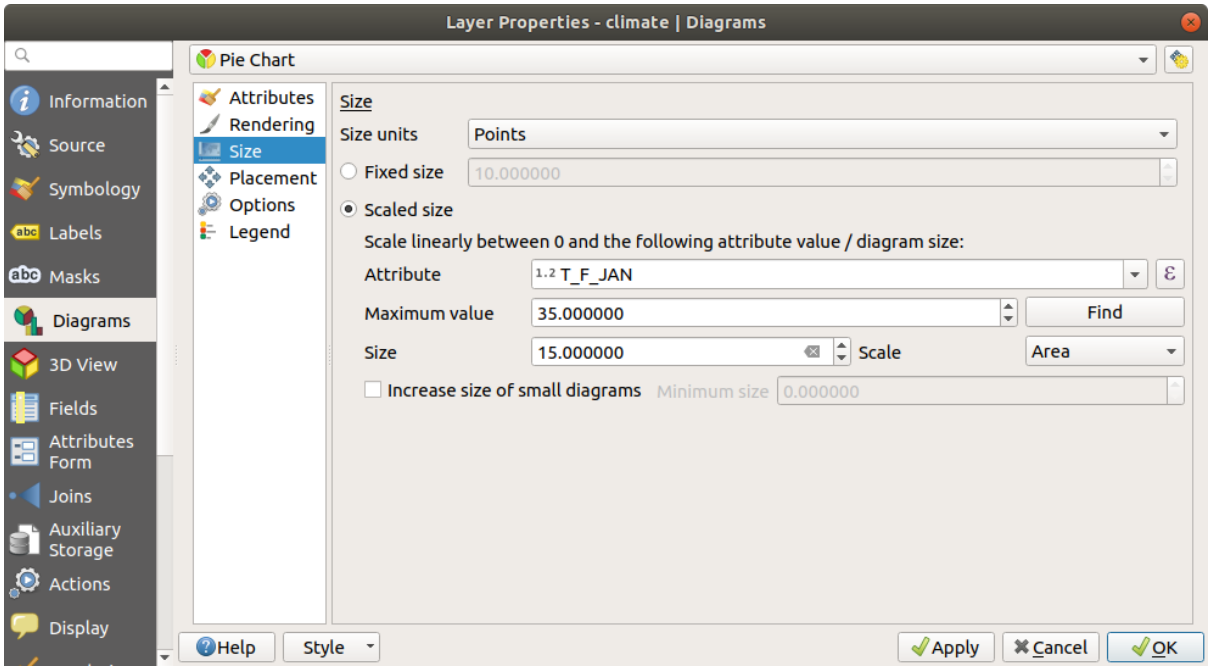


Fig. 12.36: Diagram properties - Size tab

Placement

Placement defines the diagram position. Depending on the layer geometry type, it offers different options for the placement (more details at *Placement*):

- *Around point* or *Over point* for point geometry. The former variable requires a radius to follow.
- *Around line* or *Over line* for line geometry. Like point feature, the first variable requires a distance to respect and you can specify the diagram placement relative to the feature ('above', 'on' and/or 'below' the line) It's possible to select several options at once. In that case, QGIS will look for the optimal position of the diagram. Remember that you can also use the line orientation for the position of the diagram.
- *Around centroid* (at a set *Distance*), *Over centroid*, *Using perimeter* and *Inside polygon* are the options for polygon features.

The *Coordinate* group provides direct control on diagram placement, on a feature-by-feature basis, using their attributes or an expression to set the *X* and *Y* coordinate. The information can also be filled using the *Move labels and diagrams* tool.

In the *Priority* section, you can define the placement priority rank of each diagram, i.e. if there are different diagrams or labels candidates for the same location, the item with the higher priority will be displayed and the others could be left out.

Discourage diagrams and labels from covering features defines features to use as *obstacles*, i.e. QGIS will try to not place diagrams nor labels over these features. The priority rank is then used to evaluate whether a diagram could be omitted due to a greater weighted obstacle feature.

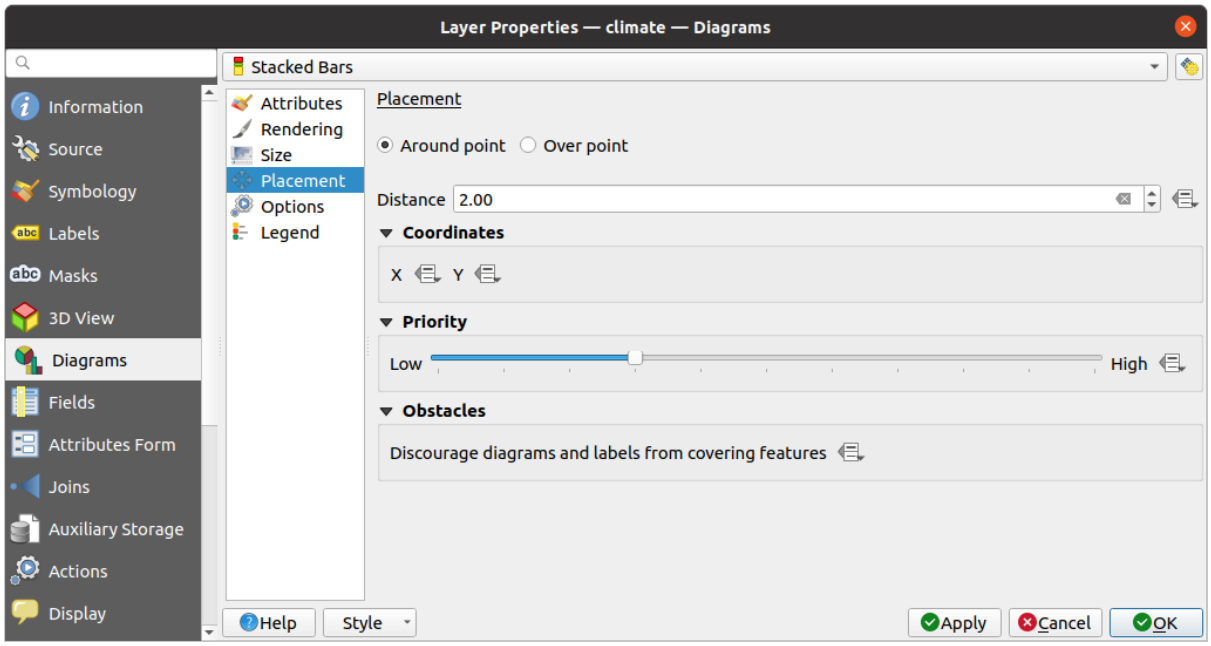


Fig. 12.37: Vector properties dialog with diagram properties, Placement tab

Options

The *Options* tab has settings for histograms and stacked bars. You can choose whether the *Bar orientation* should be *Up*, *Down*, *Right* or *Left*, for horizontal and vertical diagrams.

Legend

From the *Legend* tab, you can choose to display items of the diagram in the *Layers panel*, and in the *print layout legend*, next to the layer symbology:

- check *Show legend entries for diagram attributes* to display in the legends the `Color` and `Legend` properties, as previously assigned in the *Attributes* tab;
- and, when a *scaled size* is being used for the diagrams, push the *Legend Entries for Diagram Size…* button to configure the diagram symbol aspect in the legends. This opens the *Data-defined Size Legend* dialog whose options are described in *Data-defined size legend*.

When set, the diagram legend items (attributes with color and diagram size) are also displayed in the print layout legend, next to the layer symbology.

Stacked Diagrams

Stacked diagrams allow users to create complex diagrams like population pyramids, where two subdiagrams, namely histograms, are located side by side and displayed horizontally.

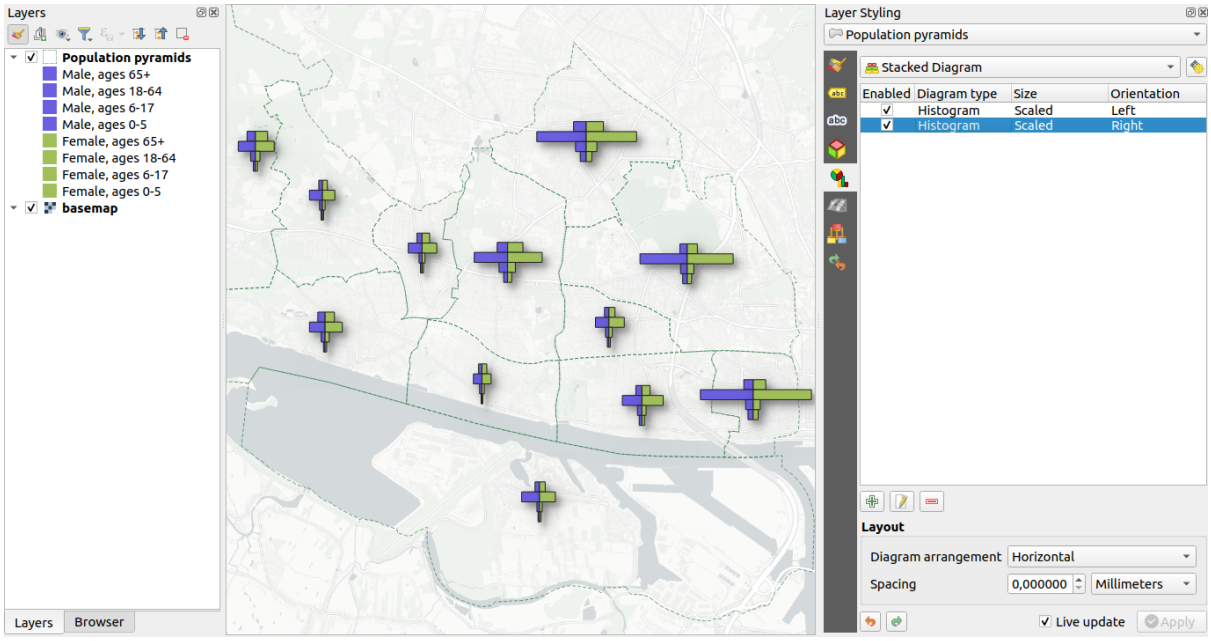


Fig. 12.38: Population pyramids built for each layer feature

Multi-temporal diagrams can also be constructed as stacked diagrams. The number of subdiagrams, as well as the spacing between them can be configured.

Moreover, subdiagrams can have different types (e.g., a pie chart alongside a histogram) and have their own independent settings like *Attributes*, *Rendering*, *Size*, *Options* and *Legend*.

Placement settings in a stacked diagram, as well as some visibility settings (located in the *Rendering* tab), are determined by the placement and visibility settings of the first subdiagram in the stack.

Finally, subdiagram ordering is given by the item ordering in the Stacked Diagram's list. The first subdiagram appears to the left in a horizontal stacked diagram, or in the upper part of a vertical one.

12.1.6 Masks Properties

The *Masks* tab helps you configure the current layer symbols overlay with other symbol layers or labels, from any layer. This is meant to improve the readability of symbols and labels whose colors are close and can be hard to decipher when overlapping; it adds a custom and transparent mask around the items to "hide" parts of the symbol layers of the current layer.

To apply masks on the active layer, you first need to enable in the project either *mask symbol layers* or *mask labels*. Then, from the *Masks* tab, check:

- the *Masked symbol layers*: lists in a tree structure all the symbol layers of the current layer. There you can select the symbol layer item you would like to transparently "cut out" when they overlap the selected mask sources
- the *Mask sources* tab: list all the mask labels and mask symbol layers defined in the project. Select the items that would generate the mask over the selected masked symbol layers

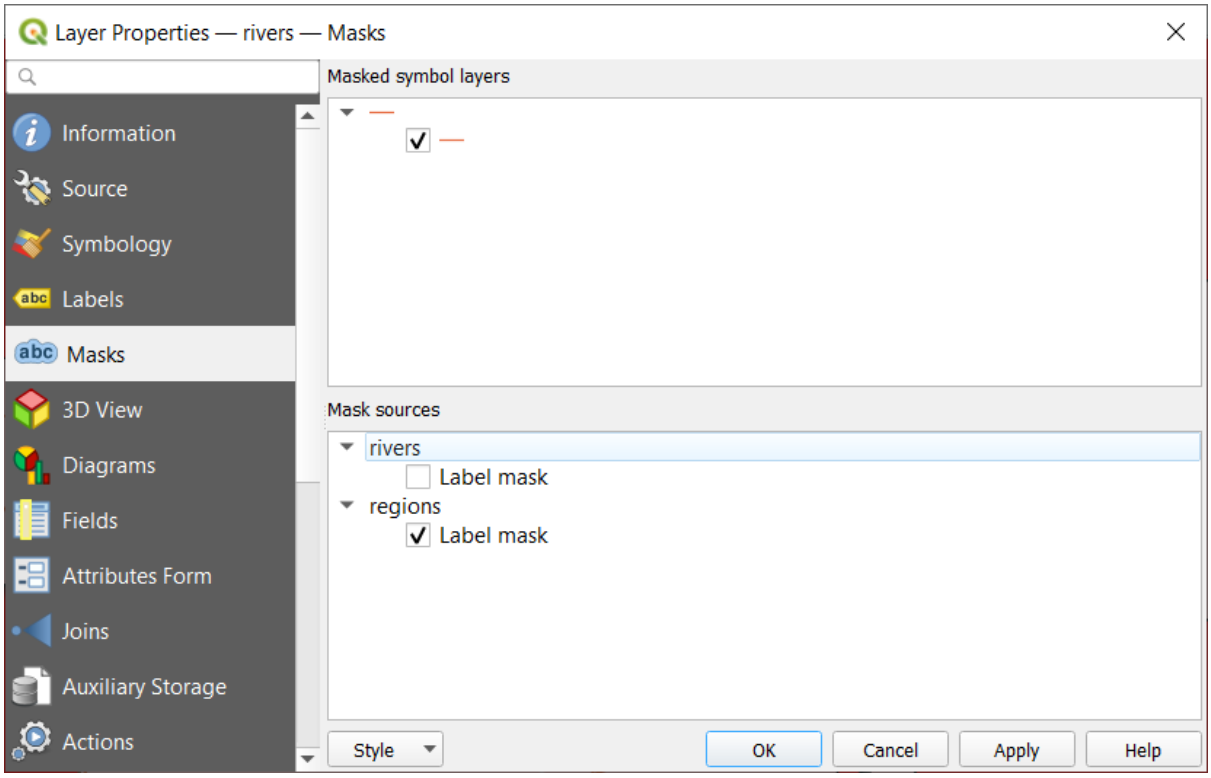


Fig. 12.39: Layer properties - Masks tab

12.1.7 3D View Properties

The *3D View* tab provides settings for vector layers that should be depicted in the *3D Map view* tool.

To display a layer in 3D, select from the combobox at the top of the tab, either:

- *Single symbol*: features are rendered using a common 3D symbol whose properties can be *data-defined* or not. Read details on *setting a 3D symbol* for each layer geometry type.
- *Rule-based*: multiple symbol configurations can be defined and applied selectively based on expression filters and scale range. More details on how-to at *Rule-based rendering*.

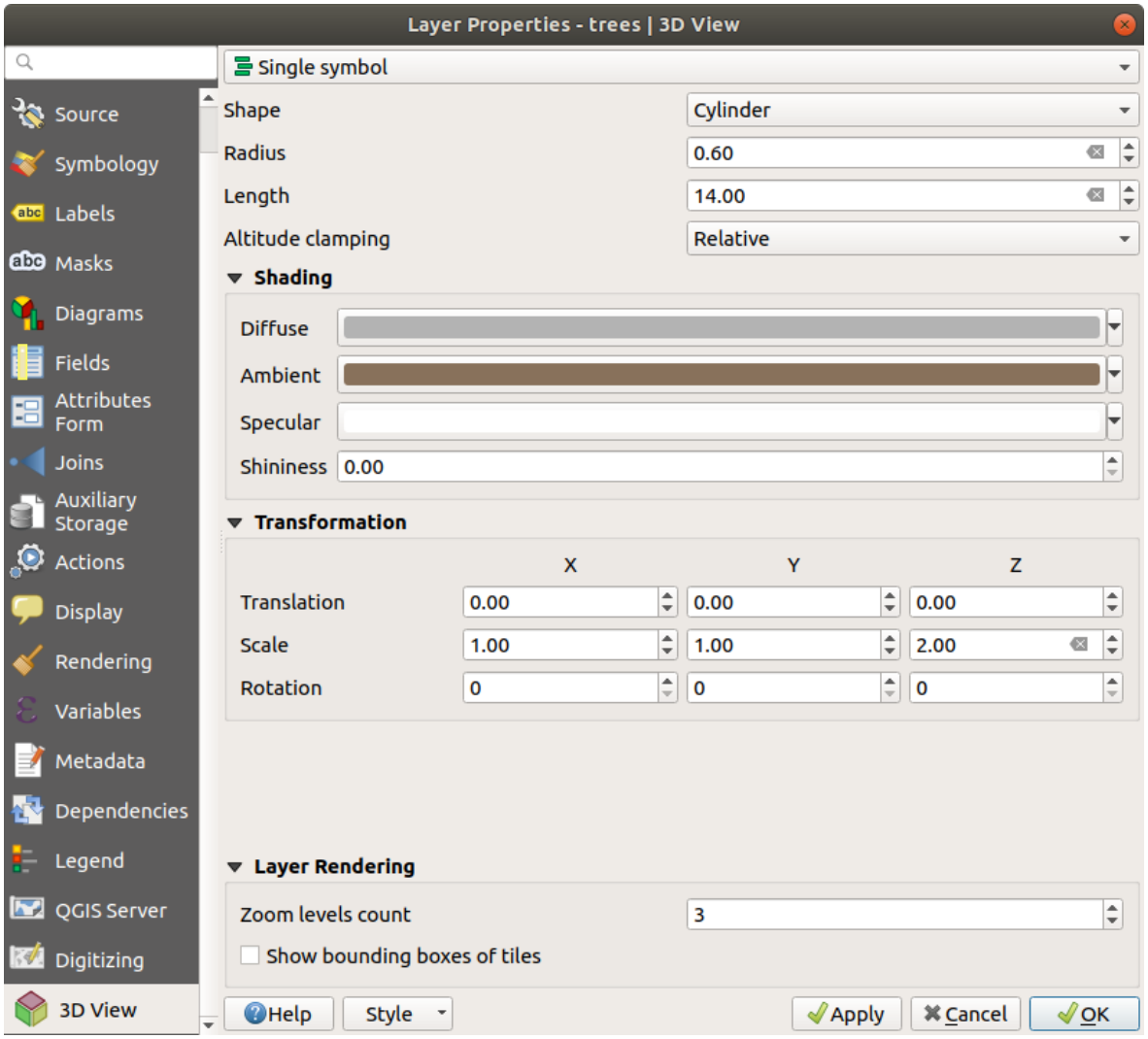


Fig. 12.40: 3D properties of a point layer

Attention

Prefer the *Elevation* tab for symbol elevation and terrain settings

Features' elevation and altitude related properties (*Altitude clamping*, *Altitude binding*, *Extrusion* or *Height*) in the *3D View* tab inherit their default values from the layer's *Elevation properties* and should preferably be set from within the *Elevation* tab.

For better performance, data from vector layers are loaded in the background, using multithreading, and rendered in tiles whose size can be controlled from the *Layer rendering* section of the tab:

- *Zoom levels count*: determines how deep the quadtree will be. For example, one zoom level means there will be a single tile for the whole layer. Three zoom levels means there will be 16 tiles at the leaf level (every extra zoom level multiplies that by 4). The default is `3` and the maximum is `8`.
- *Show bounding boxes of tiles*: especially useful if there are issues with tiles not showing up when they should.

12.1.8 Fields Properties

The *Fields* tab provides information on fields related to the layer and helps you organize them.

The layer can be made *editable* using the Toggle editing mode. At this moment, you can modify its structure using the New field and Delete field buttons.

When creating New field, the *Comment* option is available only for data sources that allow editing comments (See *Database entries* for more details). You can also set aliases within *Add Field* dialog, for supported OGR formats (GeoPackage and ESRI File Geodatabase).

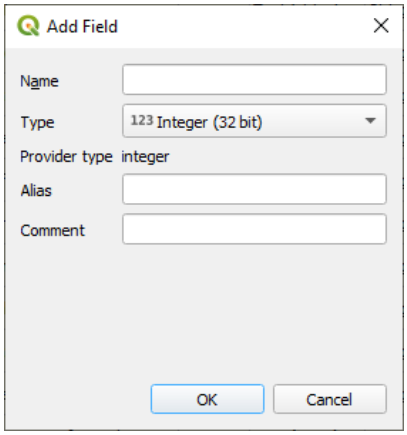


Fig. 12.41: Add Field Dialog

You can also rename fields by double-clicking its name. This is only supported for data providers like PostgreSQL, Oracle, Memory layer and some GDAL layers depending on the GDAL version.

If set in the underlying data source or in the *forms properties*, the field's alias is also displayed. An alias is a human readable field name you can use in the feature form or the attribute table. Aliases are saved in the project file.

Other than the fields contained in the dataset, *virtual fields* and *Auxiliary Storage* included, the *Fields* tab also lists fields from any *joined layers*. Depending on the origin of the field, a different background color is applied to it.

For each listed field, the dialog also lists read-only characteristics such as its *Type*, *Type name*, *Length* and *Precision*`.

Depending on the data provider, you can associate a comment with a field, for example at its creation. This information is retrieved and shown in the *Comment* column and is later displayed when hovering over the field label in a feature form.

Under the *Configuration* column, you can set how the field should behave in certain circumstances:

- `Not searchable`: check this option if you do not want this field to be queried by the *search locator bar*
- `Do not expose via WMS`: check this option if you do not want to display this field if the layer is served as WMS from QGIS server
- `Do not expose via WFS`: check this option if you do not want to display this field if the layer is served as WFS from QGIS server

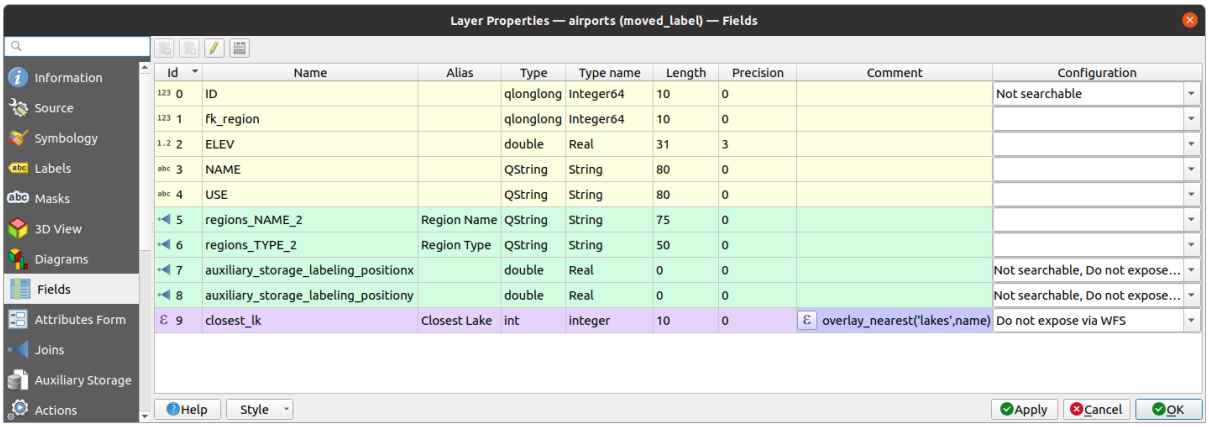


Fig. 12.42: Fields properties tab

12.1.9 Attributes Form Properties

The *Attributes Form* tab helps you set up the form to display when creating new features, editing or querying existing one. This affects them in both *table and form views*. You can define:

- the look and the behavior (label, visibility, widget, constraints…) of each field in the current layer, the *joined layers* and the *relational ones*;
- the placement of enabled *actions*;
- the form's structure (custom or autogenerated);
- extra HTML, QML, text widgets to enhance the form;
- extra logic in Python to handle interaction with the form or field widgets.

At the top right of the dialog, you can set whether the form is opened by default when creating new features. This can be configured per layer or globally with the *Suppress attribute form pop-up after feature creation* option in the *Settings* ► *Options* ► *Digitizing* menu.

Customizing a form for your data

By default, when you click on a feature with the Identify Features tool or switch the attribute table to the *form view* mode, QGIS displays a basic form with predefined widgets (generally spinboxes and textboxes — each field is represented on a dedicated row by its label next to the widget). If *relations* are set on the layer, fields from the referencing layers are shown in an embedded frame at the bottom of the form, following the same basic structure.

This rendering is the result of the default `Autogenerate` value of the *Attribute editor layout* setting in the *Layer properties* ► *Attributes Form* tab. This property holds three different values:

- `Autogenerate`: keeps the basic structure of "one row - one field" for the form but allows to customize each corresponding widget.
- `Drag-and-drop designer`: other than widget customization, the form structure can be made more complex eg, with widgets embedded in groups and tabs.
- `Provide ui file`: allows to use a Qt designer file, hence a potentially more complex and fully featured template, as feature form.

Just below the *Attribute editor layout* setting, a search box allows you to filter fields, relations, containers and other available widgets.

For items that support aliases, like fields or relations, the filter targets both names and aliases.

The Show field aliases instead of names button allows you to toggle field aliases and field names.

When showing aliases, fields without an alias set will have their names displayed in light gray to help identify them easily.

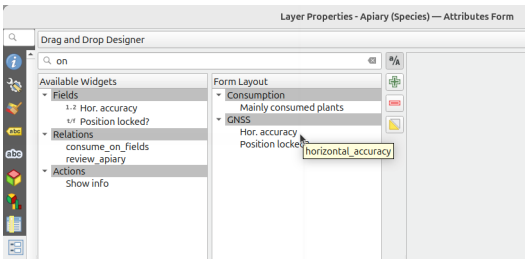


Fig. 12.43: Search fields using aliases and names in Attributes Form page

The autogenerated form

When the `Autogenerate` option is on, the *Available widgets* panel displays lists of fields (from the current layer, the join and relation layers) and the enabled actions that would be shown in the form. Select a field and you can configure its appearance and behavior in the right panel:

- adding *custom label and automated checks* to the field;
- setting a *particular widget* to use.

In this editor mode, only the properties can be viewed for the actions. Moreover, they will be displayed through an *Actions* drop-down menu in the feature form, and following the *Show in attribute table* property in other contexts.

The drag and drop designer

The drag and drop designer allows you to create a form with several containers (tabs or groups) to present the attribute fields or other widgets that are not directly linked to a particular field (like the HTML/QML widgets or the *actions* defined for the layer), as shown for example in Fig. 12.44.

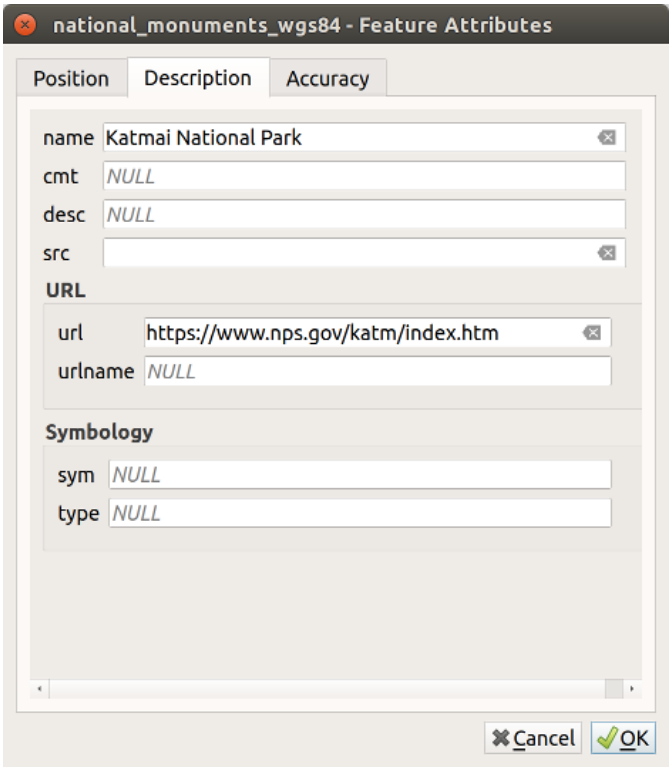


Fig. 12.44: Resulting built-in form with tabs and named groups

1. Choose `Drag and drop designer` from the *Select attribute layout editor* combobox. This enables the *Form Layout* panel next to the *Available widgets* panel, filled with existing fields. The selected field displays its *properties* in a third panel.

2. Select fields you do not want to use in your *Form Layout* panel and hit the button to remove them. You can also toggle the selection with the Invert selection button.

3. Drag and drop fields from the first panel to the *Form Layout* one to re-add them. The same field can be added multiple times.

4. Drag and drop fields within the *Form Layout* panel to reorder their position.

5. Add containers to associate fields that belong to the same category and better structure the form.

    1. The first step is to use the Add a new tab or group to the form layout icon. Fields and other groups will be displayed in it.

    2. Then set the properties of the container, i.e.:

        - the *Label*: the title that will be used for the container
        - the *Container Type*: it can be a *Tab*, *Group box in container* (a collapsible group box inside a tab or another group) or a *Row* (a container type that allows you to arrange your widgets in a horizontal row, automatically determining the number of columns based on the number of widgets),
        - the *Within*: this optional feature allows you to select an existing container in which the new container (*Group box in container* or *Row*) will be embedded.
        - and the *Number of columns* the embedded fields should be distributed over

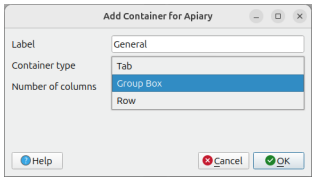


Fig. 12.45: Dialog to create containers with the **Attribute editor layout**

These, and other properties can later be updated by selecting the item and, from the third panel:

- hide or show the container's label
- rename the container
- set the number of columns
- enter an expression to control the container's visibility. The expression will be re-evaluated every time values in the form change, and the tab or group box shown/hidden accordingly
- *Show as Group Box*: converts a tab into a top-level group box and vice versa
- in case of a group box, set whether at the form opening, it should show as *Collapsed* for every features, or only for those matching an expression (*Control collapsed by expression*).
- configure the *Style* look of the container, with custom *Background color*, label color and font properties

3. You can create and embed as many containers as you want by pressing again the Add a new tab or group to the form layout icon.

6. The next step is to assign the relevant fields to each container, by simple drag and drop. Group boxes and tabs can also be moved the same way.
7. *Customize the widget* of the fields in use
8. In case the layer is involved in a *one or many to many relation*, drag-and-drop the relation name from the *Available Widgets* panel to the *Form Layout* panel. The associated layer attribute form will be embedded at the chosen place in the current layer's form. As for the other items, select the relation label to configure some properties:
   - hide or show the relation label
   - show the link button
   - show the unlink button
9. In case the layer has one or more *actions* enabled for `Layer` or `Feature` scope, the actions will be listed under *Actions* and you can drag and drop them as with the other fields. The associated action will be embedded at the chosen place in the current layer's form.
10. Further customize the form by adding one or more widgets from *Other Widgets* (see *Other Widgets*)
11. Apply the layer's properties dialog
12. Open a feature attribute form (eg, using the Identify features tool) and it should display the new form.

Other Widgets

The drag and drop designer offers a number of widgets that are not connected to a particular field. They can be used to enhance the appearance of the form or to display dynamically calculated values.

- *HTML Widget*: embeds an HTML page, the HTML source may contain the result of dynamically calculated expressions.

  HTML widgets can be used for example to display images stored as BLOB in a field (let's call it `photo`):

  1. In the `Drag-and-drop designer` mode, add a *HTML Widget* to your *Form Layout*.
  2. Double-click on the *HTML Widget* to configure it.
  3. Change the default *Title* or hide it.
  4. Press the  button and enter the following QGIS expression:

     ```
     '<img src= "' || to_base64("photo") || '">'
     ```

     Ensure that you replace *photo* with your own BLOB field name. The above expression creates a string with HTML image tag in which the BLOB file is encoded.
  5. Apply the dialog and then press the  button.
  6. QGIS automatically applies HTML formatting and functions to evaluate your expression, resulting in following code:

     ```
     <script>document.write(expression.evaluate("'<img src=\"data:image/png;
     ↪base64,' || to_base64(\"photo\") || '\">'"));</script>
     ```

     A preview of your image is displayed on the right.

- *QML Widget*: embeds a Qt QML document, displaying graphical elements in the attribute form. Beside the custom *Title* of the added widget and whether it should be shown or not, you can select from predefined *QML code* elements:
  - *Free text...*: allows you to write from scratch or paste an existing code
  - *Rectangle*: provides a minimal code for displaying a rectangle
  - *Bar chart*: provides a minimal code for displaying a bar chart
  - *Pie chart*: provides a minimal code for displaying a pie chart

  You can extend the code with QML syntax, use layer fields or QGIS expressions that are dynamically calculated.

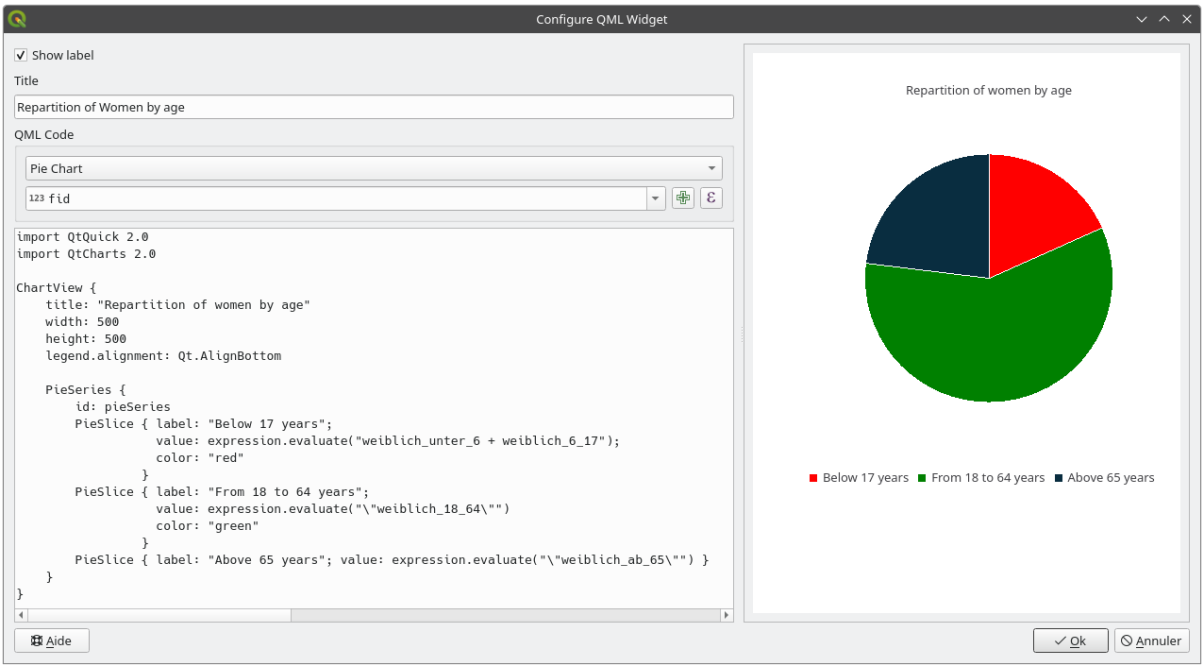


Fig. 12.46: Setting a QML graph to display in attribute form

- *Text Widget*: displays a text widget which supports basic HTML markup and may contain the result of dynamically calculated expressions.
- *Spacer Widget*: inserts an empty transparent rectangle, increasing the vertical distance between two widgets.

Tip

Display Dynamic Content

The widgets mentioned above (except the *Spacer Widget*) support expressions that can be used to display content that dynamically changes whenever another field in the form changes. This can be achieved using the `current_value('field_name')` function in the expression to examine the value of another field.

Using custom ui-file

The `Provide ui-file` option allows you to use complex dialogs made with Qt-Designer. Using a UI-file allows a great deal of freedom in creating a dialog. Note that, in order to link the graphical objects (textbox, combobox…) to the layer's fields, you need to give them the same name.

Use the *Edit UI* to define the path to the file to use.

UI-files can also be hosted on a remote server. In this case, you provide the URL of the form instead of the file path in *Edit UI*.

You'll find some example in the Creating a new form lesson of the QGIS-training-manual-index-reference. For more advanced information, see https://woostuff.wordpress.com/2011/09/05/qgis-tips-custom-feature-forms-with-python-logic/.

Enhance your form with custom functions

QGIS forms can have a Python function that is called when the dialog is opened. Use this function to add extra logic to your dialogs. The form code can be specified in three different ways:

- `load from the environment`: use a function, for example in `startup.py` or from an installed plugin
- `load from an external file`: a file chooser will let you select a Python file from your filesystem or enter a URL for a remote file.
- `provide code in this dialog`: a Python editor will appear where you can directly type the function to use.

In all cases you must enter the name of the function that will be called (`open` in the example below).

An example is (in module MyForms.py):

```
def open(dialog,layer,feature):
    geom = feature.geometry()
    control = dialog.findChild(QWidget,"My line edit")
```

Reference in Python Init Function like so: `open`

Configure the field behavior

The main part of the *Attributes Form* tab helps you set the type of widget used to fill or display values of the field, in the attribute table or the feature form: you can define how user interacts with each field and the values or range of values that are allowed to be added to each.

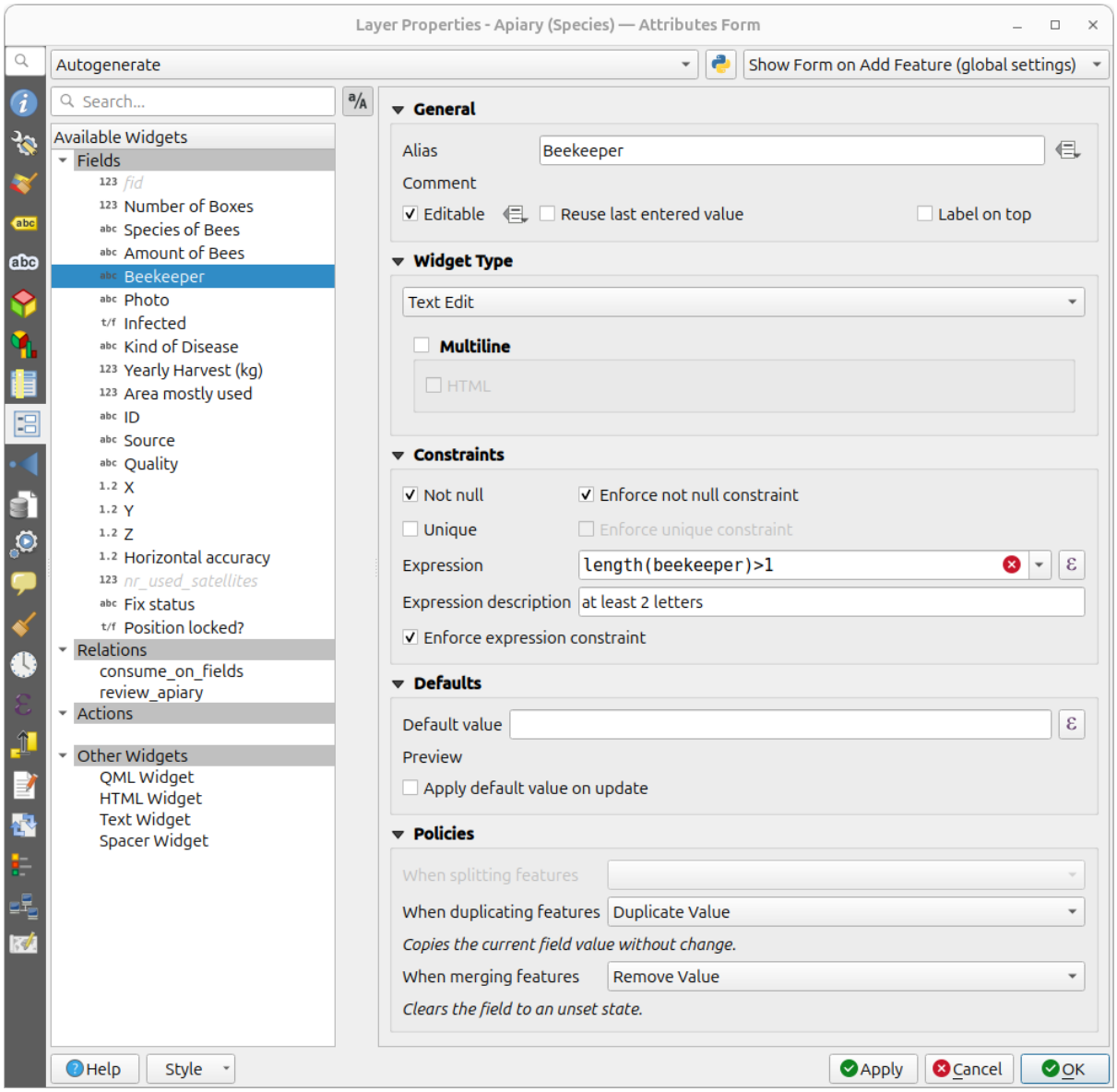


Fig. 12.47: Dialog to select an edit widget for an attribute column

Tip

Copy and paste widget configuration

The field configuration can be copied to the clipboard and then pasted to another field via right click on a field in the *Available widgets* panel.

This comes handy to save time and to make sure that common settings are correctly applied to several related fields, e.g., when configuring latitude and longitude field widgets.

Copy and paste of widget configuration can be done between fields of the same layer, between fields from different layers in the QGIS project, or between fields from layers in different QGIS instances.

Note that neither aliases nor comments are copied, since those are specific to each particular field.

Common settings

Regardless the type of widget applied to the field, there are some common properties you can set to control whether and how a field can be edited.

Widget display

Only available for the *Drag and drop* designer mode, this group helps you configure the look of the widget assigned to the field:

- *Show label*: indicates whether the field name should be displayed in the form
- *Override label color*: applies specific color to the field's label
- *Override label font*: applies specific font properties (bold, italic, underline, strikethrough, or font family) to the field's label
- *Size*: allows to control how widgets will relatively resize when resizing an attribute form.
  - *Horizontal stretch*: sets a higher horizontal value for widgets that need more horizontal space.
  - *Vertical stretch*: determines how widgets resize vertically when the form is resized.

General options

- *Alias*: a human readable name to use for fields. The alias will be displayed in the feature form, the attribute table, or in the *Identify results* panel. It can also be used as field name replacement in the *expression builder*, easing expressions understanding and reviews. Aliases are saved in project file.
- *Comment*: displays the field's comment as shown in the *Fields* tab, in a read-only state. This information is shown as tooltip when hovering over the field label in a feature form.
- ☑ *Editable*: uncheck this option to set the field read-only (not manually modifiable) even when the layer is in edit mode. Note that checking this setting doesn't override any edit limitation from the provider. This option can be controlled by a data-defined property thanks to the Data-defined override button.
- ☑ *Reuse last entered value*: remembers the last value entered in this field and uses it as default for the next feature being edited in the layer.
- ☑ *Label on top*: places the field name above or beside the widget in the feature form.

Constraints

You can constrain the value to insert in the field. This constraint can be:

- ☑ *Not null*: requires the user to provide a value;
- ☑ *Unique*: guarantee the inserted value to be unique throughout the field;
- based on a custom *expression*: e.g. `not regexp_match(col0,'[^A-Za-z]')` will ensure that the value of the field *col0* has only alphabet letters. A short description can be added to help you remember the constraint.

Whenever a value is added or edited in a field, it's submitted to the existing constraints and:

- if it meets all the requirements, a green check is shown beside the field in the form;
- if it does not meet all the requirements, then the field is colored in yellow or orange and a corresponding cross is displayed next to the widget. You can hover over the cross to remind which constraints are applied to the field and fix the value:

  - A yellow cross appears when the unmet constraint is an unenforced one (`soft constraint`) and it does not prevent you to save the changes with the "wrong" values;
  - An orange cross can not be ignored and does not allow you to save your modifications until they meet the constraints. It appears when the *Enforce constraint* option is checked (`hard constraint`).

Default values

- *Default value*: for new features, automatically populates by default the field with a predefined value or an *expression-based one*. For example, you can:
  - use `$x`, `$length`, `$area` to automatically populate a field with the feature's X coordinate, length, area or any geometric information at its creation;
  - increment a field by 1 for each new feature using `maximum("field")+1`;
  - save the feature creation datetime using `now()`;
  - use *variables* in expressions, making it easier to e.g. insert the operator name (`@user_full_name`), the project file path (`@project_path`), ...

  A preview of the resulting default value is displayed at the bottom of the widget.

  Note

  The `Default value` option is not aware of the values in any other field of the feature being created so it won't be possible to use an expression combining any of those values i.e using an expression like `concat(field1, field2)` may not work.

- *Apply default value on update*: whenever the feature attribute or geometry is changed, the default value is recalculated. This could be handy to save values like last user that modifies data, last time it was changed...

Policies

Field policies determine how values are assigned to fields during various editing operations:

Split and Duplicate Policies

These policies apply when *Splitting features* or *Duplicating features*:

- *Duplicate Values*: Keeps the existing value of the field for the new features.
- *Use Default Value*: Resets the field by recalculating its *default value*. If no default value clause exists, the existing value is kept for the new features.
- *Remove Value*: Clears the field to an unset state.
- *Use Ratio Geometries*: Recalculates the field value for all split portions by multiplying the existing value by ratio of the split parts lengths or areas.

Merge Policies

These policies determine initial values when *Merging features*:

- *Remove Value*: Clears the field to an unset state (data provider may populate default value).
- *Use Default Value*: Uses the default field value set in QGIS. (expressions use a dummy feature with the merged geometry; references to other fields won't work).
- *Use Sum* (numeric fields only): Sums values from all merged features.
- *Use Minimum Value* (numeric fields only): Uses smallest value from the selected features.
- *Use Maximum Value* (numeric fields only): Uses largest value from the selected features.

- *Average Weighted by Geometry* (numeric fields only): New values are computed as the weighted average of the source values.
- *Use Largest Feature*: Uses value from feature with the largest geometry.
- *Set to NULL*: Sets attribute to NULL.

Edit widgets

Based on the field type, QGIS automatically determines and assigns a default widget type to it. You can then replace the widget with any other compatible with the field type. The available widgets are:

- **Binary (BLOB)**: Available only for binary fields, it displays by default a label with the size of the embedded data, if not empty. A drop-down button next to the label allows to:
  - *Embed file*, replacing or filling the field
  - *Clear contents*, removing any data in the field
  - *Save contents to file*, exporting the data as a file on disk

  It is also possible to preview the embedded binary file in the field, if combined in a drag-and-drop form with e.g. a *QML or HTML widget*.
- **Checkbox**: Displays a checkbox whose state defines the value to insert. Internally, the widget stores a value (e.g. 1 for checked, 0 for unchecked) while representing it visually as a toggled box. You can configure the stored values for the checked and unchecked states in the representation. It is also possible to display the checkbox state as a plain text using "True"/"False" labels. These settings influence how the values are shown in classification renderers.
- **Classification**: Only available when a *categorized symbology* is applied to the layer, displays a combo box with the values of the classes.
- **Color**: Displays a *color widget* allowing to select a color; the color value is stored as a html notation in the attribute table.
- **Date/Time**: Displays a line field which can open a calendar widget to enter a date, a time or both. Column type must be text. You can select a custom format, pop-up a calendar, etc.
- **Enumeration**: Opens a combo box with predefined values fetched from the database. This is currently only supported by the PostgreSQL provider, for fields of `enum` type.
- **Attachment**: Uses a "Open file" dialog to store file path in a relative or absolute mode. It can be used to display the document path as a hyperlink or render the document within a dedicated widget in the form. Supported document types are image, web page, audio and video, and the supported file formats depend on the Operating System. You can also configure an *external storage system* to fetch/store resources.

  > **Tip**
  >
  > **Relative Path in Attachment widget**
  >
  > If the path which is selected with the file browser is located in the same directory as the `.qgs` project file or below, paths are converted to relative paths. This increases portability of a `.qgs` project with multimedia information attached.

- **Geometry**: Compatible with the *Geometry* field type. For example, PostgreSQL and Memory layers can contain multiple geometry fields. QGIS uses the first geometry column as the layer geometry. Applying the Geometry widget to additional geometry fields allows lossless use of the data in the attribute form:
  - Copy in the formats WKT (Well-known text representation, e.g. `POINT (30 10)`) or GeoJSON
  - Paste
  - Clear

- **Hidden**: A hidden attribute column is invisible. The user is not able to see its contents.
- **Key/Value**: Displays a two-columns table to store sets of key/value pairs within a single field. This is currently supported by the PostgreSQL provider, for fields of `hstore` type.
- **JSON View**: Displays JSON data in a syntax highlighted text edit or in tree view. This widget is currently read only. Several options are available to change how the data is displayed. 'Default view' specify if the widget should appear in Text or Tree mode. 'Format JSON' has three options which are related to the tree view only:
  - Indented: Display data in a human readable form with newlines and four space characters for indentation.
  - Compact: Display data in a one-line size optimized string without newlines or spaces.
  - Disabled: Display data as it comes from the provider.
- **List**: Displays a single column table to add different values within a single field. This is currently supported by the PostgreSQL provider, for fields of `array` type.
- **Range**: Allows you to set numeric values from a specific range. The edit widget can be either a slider or a spin box.

  > **Note**
  >
  > Some layers, such as GeoPackage or ESRI File Geodatabase, with predefined **range Field Domains** will be automatically recognized by QGIS and assigned a **Range** widget for the relevant fields. The widget will be prefilled with the minimum and maximum values specified in the domain.

- **Relation Reference**: This is the default widget assigned to the referencing field (i.e., the foreign key in the child layer) when a *relation* is set. It provides direct access to the parent feature's form which in turn embeds the list and form of its children. The number of entries in the widget can be limited for efficiency, and if limit is not set, all entries will be loaded. The field defined in the display expression is used as representable value in classification renderers.
- **Text Edit** (default): This opens a text edit field that allows simple text or multiple lines to be used. If you choose multiple lines you can also choose html content.
- **Unique Values**: You can select one of the values already used in the attribute table. If 'Editable' is activated, a line edit is shown with autocompletion support, otherwise a combo box is used.
- **Uuid Generator**: Generates a read-only UUID (Universally Unique Identifiers) field, if empty.
- **Value Map**: A combo box with predefined items. The value is stored in the attribute, the description is shown in the combo box. You can define values manually or load them from a layer or a CSV file. The description is used as the representable value, meaning it will be shown in classification renderers.

  > **Note**
  >
  > Some layers, such as GeoPackage or ESRI File Geodatabase, with predefined **coded Field Domains** will be automatically recognized by QGIS and assigned a **Value Map** widget for the relevant fields.

- **Value Relation**: Offers values from a related table in a combobox. You can select layer, key column and value column. The key column is the stored value in the attribute table, while the value column is the label shown in the combobox. Value column is also the representable value that will be shown in the classification renderers. You may also choose to order the list of displayed values by the *Key* column (the default sort order), the *Value* column, or a specific *Field* of the layer. Enable the  *Descending order* if you wish the values to be sorted in reverse order.

  Several options are available to change the standard behavior: allow NULL value, allow multiple selections and use of auto-completer. The forms will display either a drop-down list or a line edit field when completer checkbox is enabled.

If a layer that is stored in PostgreSQL, GeoPackage or SpatiaLite is configured to use a value relation widget, but the required layer is not already loaded into the project, QGIS will automatically search for the layer in the same database/connection.

12.1.10 Joins Properties

The  *Joins* tab allows you to associate features of the current layer to features from another loaded vector layer (or table). The join is based on an attribute that is shared by the layers, in a one-to-one relationship. For more details on joins, please read *Joining features between two layers*.

12.1.11 Auxiliary Storage Properties

The regular way to customize styling and labeling is to use data-defined properties as described in *Data defined override setup*. However, it may not be possible if the underlying data is read only. Moreover, configuring these data-defined properties may be very time consuming or not desirable! For example, if you want to fully use map tools coming with *The Label Toolbar*, then you need to add and configure more than 20 fields in your original data source (X and Y positions, rotation angle, font style, color and so on).

The Auxiliary Storage mechanism provides the solution to these limitations and awkward configurations. Auxiliary fields are a roundabout way to automatically manage and store these data-defined properties (labels, diagram, symbology…) in a SQLite database thanks to editable joins. This allows you to store properties for layers that aren't editable.

A tab is available in vector layer properties dialog to manage auxiliary storage:

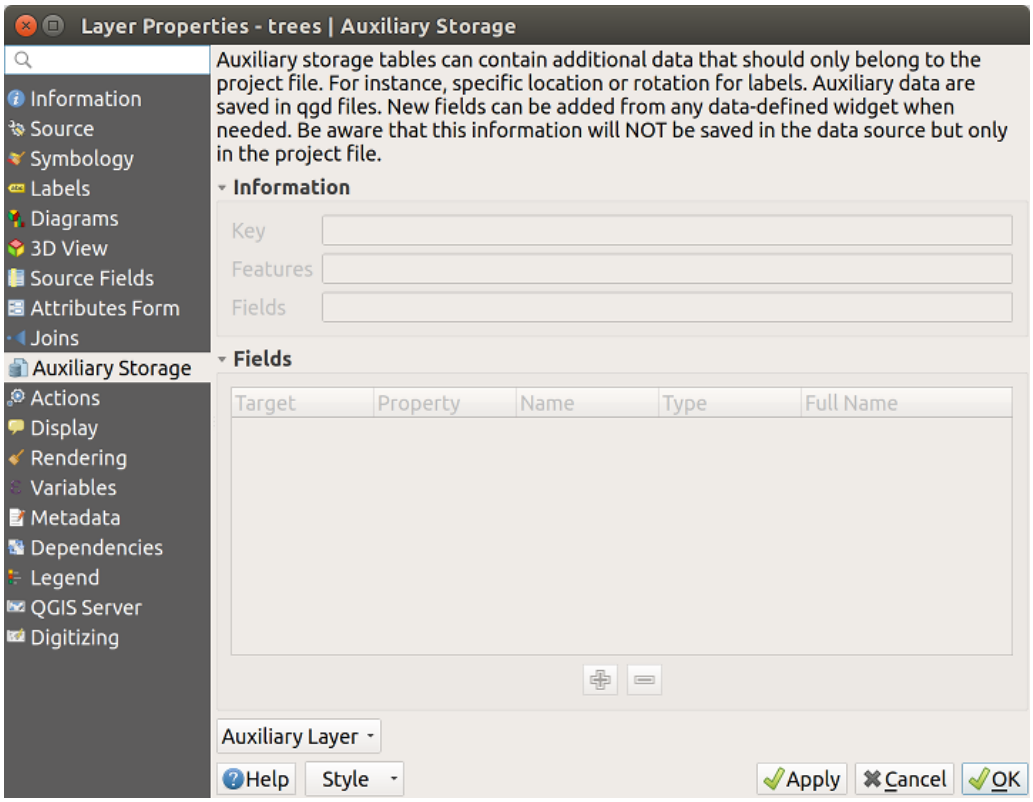


Fig. 12.48: Auxiliary Storage tab

Labeling

Considering that the data source may be customized thanks to data-defined properties without being editable, labeling map tools described in *The Label Toolbar* are always available as soon as labeling is activated.

Actually, the auxiliary storage system needs an auxiliary layer to store these properties in a SQLite database (see *Auxiliary storage database*). Its creation process is run the first time you click on the map while a labeling map tool is currently activated. Then, a window is displayed, allowing you to select the primary key to use for joining (to ensure that features are uniquely identified):

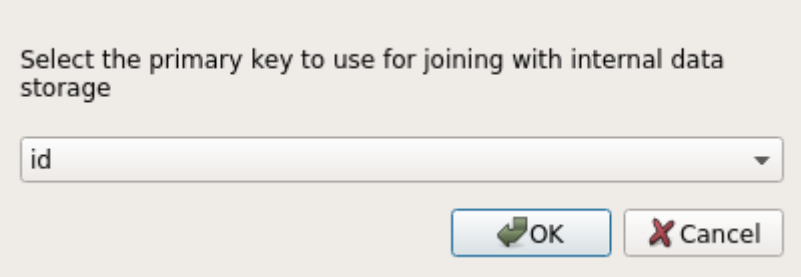


Fig. 12.49: Auxiliary Layer creation dialog

As soon as an auxiliary layer is configured for the current data source, you can retrieve its information in the tab:

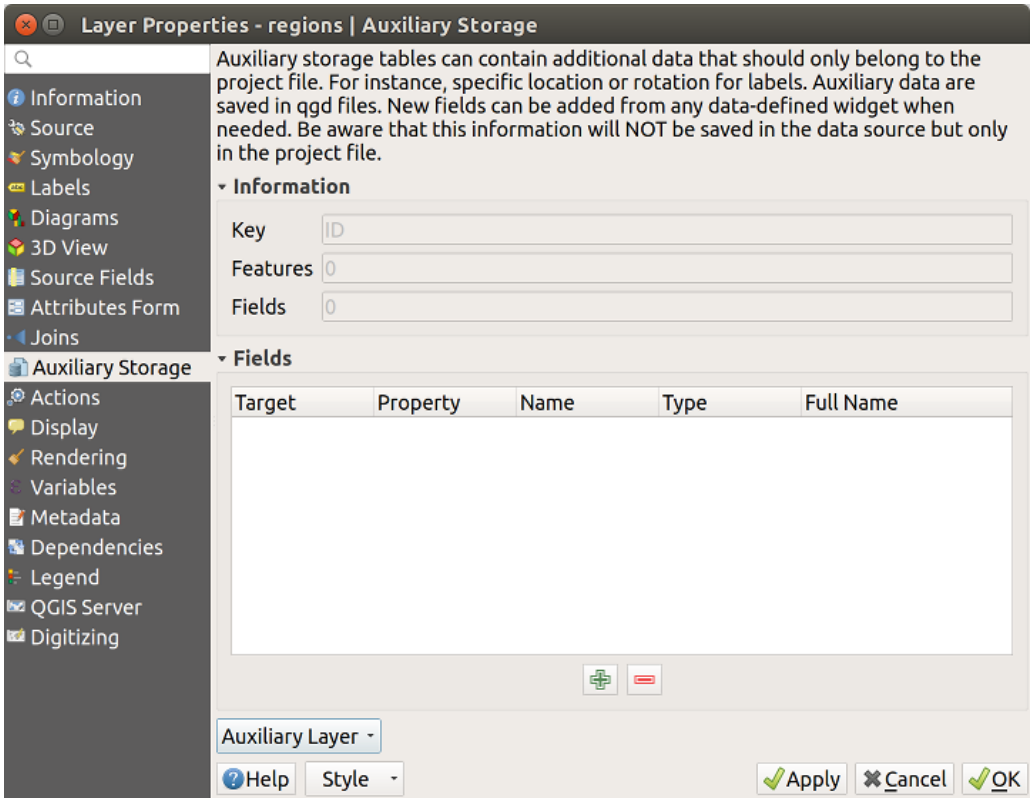


Fig. 12.50: Auxiliary Layer key

The auxiliary layer now has these characteristics:

- the primary key is `ID`,
- there are `0` features using an auxiliary field,
- there are `0` auxiliary fields.

Now that the auxiliary layer is created, you can edit the layer labels. Click on a label while the Change Label map tool is activated, then you can update styling properties like sizes, colors, and so on. The corresponding data-defined properties are created and can be retrieved:

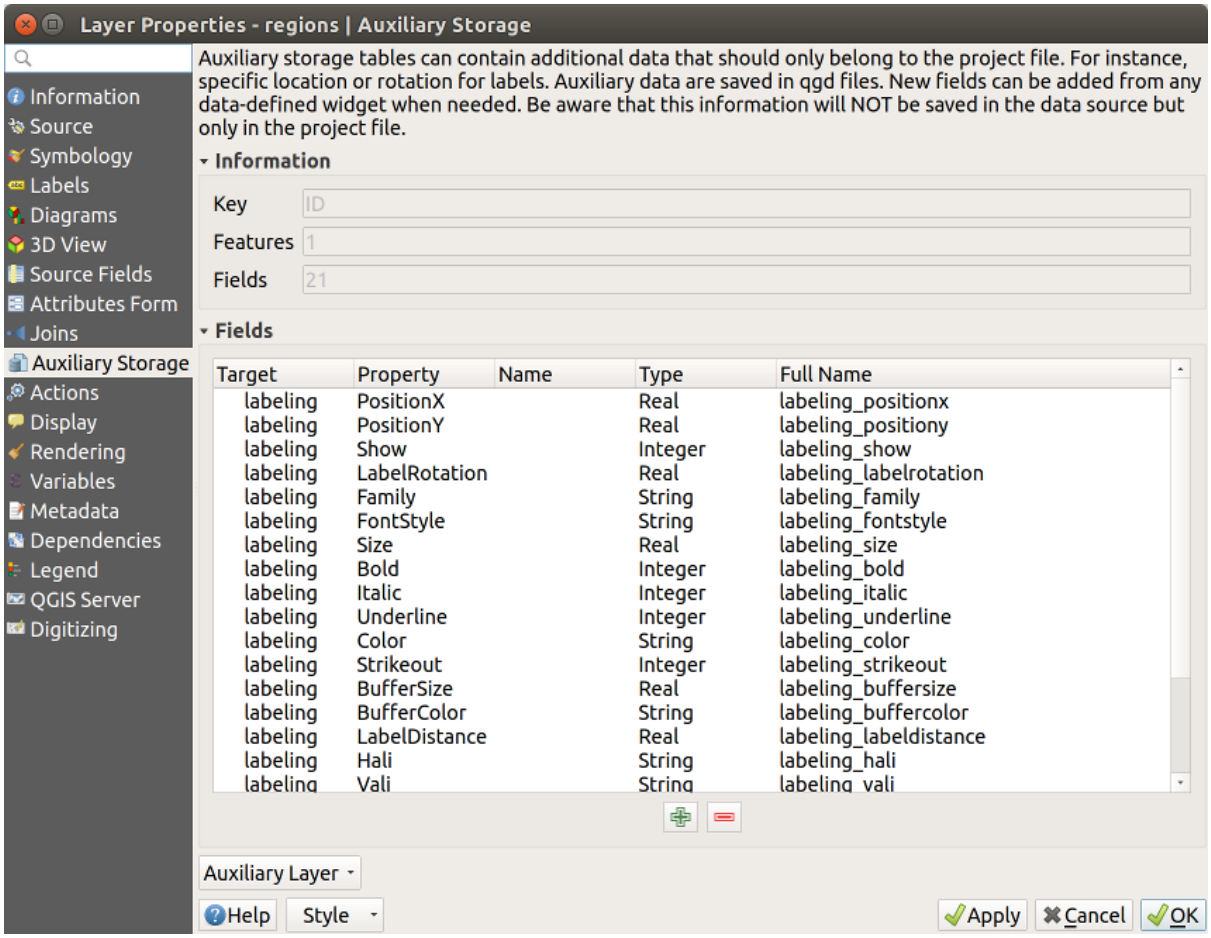


Fig. 12.51: Auxiliary Fields

As you can see in the figure above, `21` fields are automatically created and configured for labeling. For example, the `FontStyle` auxiliary field type is a `String` and is named `labeling_fontstyle` in the underlying SQLite database. There is also `1` feature which is currently using these auxiliary fields.

Notice that the icon is displayed in the *Labels* properties tab indicating that the data-defined override options are set correctly:

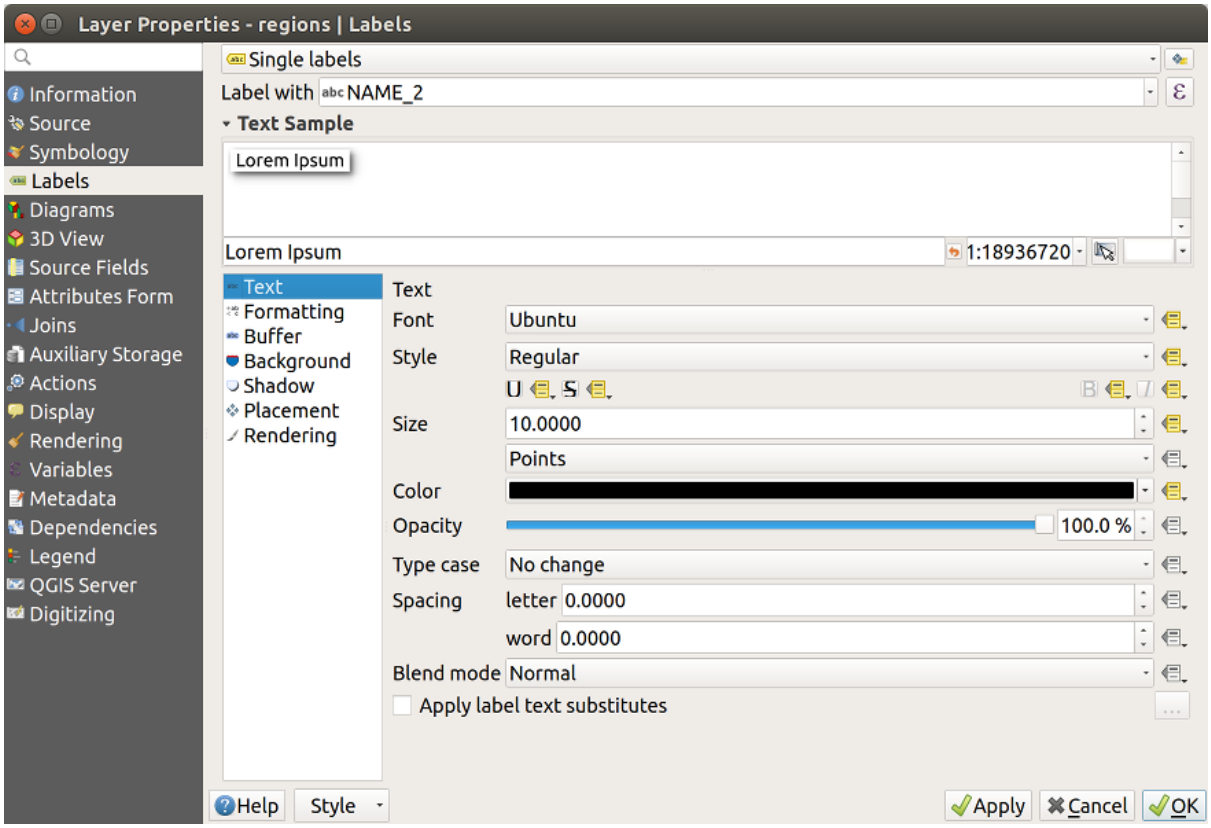


Fig. 12.52: Data-defined properties automatically created

Otherwise, there's another way to create an auxiliary field for a specific property thanks to the Data-defined override button. By clicking on *Store data in the project*, an auxiliary field is automatically created for the *Opacity* field. If you click on this button and the auxiliary layer is not created yet, a window (Fig. 12.49) is first displayed to select the primary key to use for joining.

Symbology

Like the method described above for customizing labels, auxiliary fields can also be used to stylize symbols and diagrams. To do this, click on Data-defined override and select *Store data in the project* for a specific property. For example, the *Fill color* field:

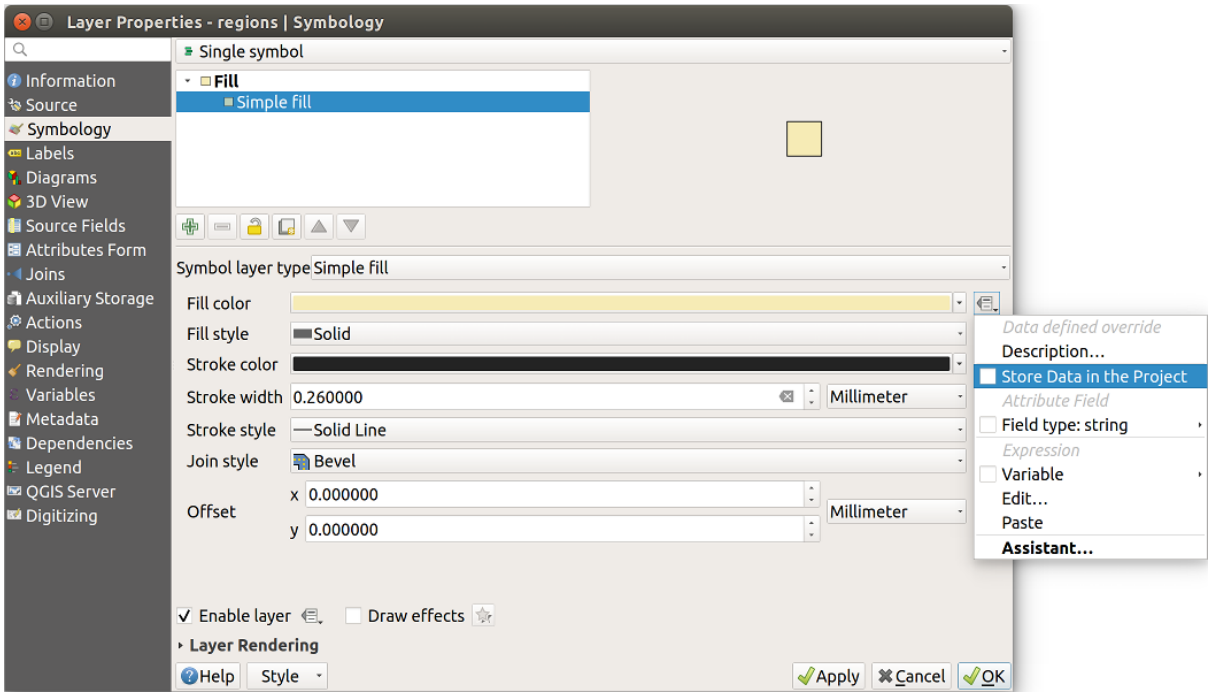


Fig. 12.53: Data-defined property menu for symbol

There are different attributes for each symbol (e.g. fill style, fill color, stroke color, etc…), so each auxiliary field representing an attribute requires a unique name to avoid conflicts. After selecting *Store data in the project*, a window opens and displays the *Type* of the field and prompts you to enter a unique name for the auxiliary field. For example, when creating a *Fill color* auxiliary field the following window opens:

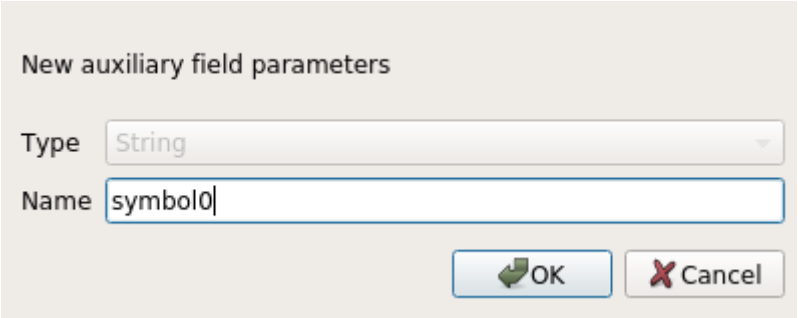


Fig. 12.54: Name of the auxiliary field for a symbol

Once created, the auxiliary field can be retrieved in the auxiliary storage tab:

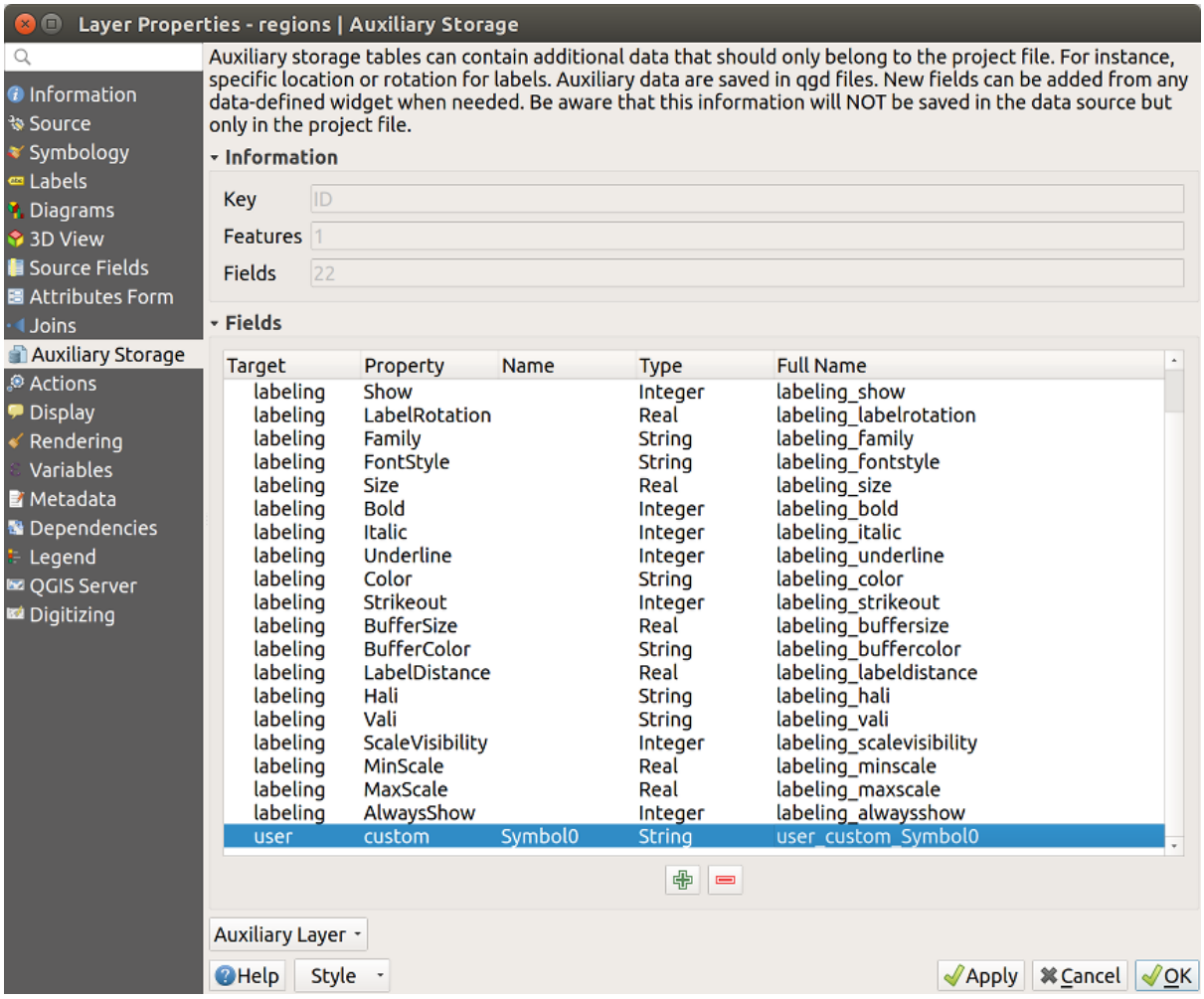


Fig. 12.55: Auxiliary field symbol

Attribute table and widgets

Auxiliary fields can be edited using the *attribute table*. However, not all auxiliary fields are initially visible in the attribute table.

Auxiliary fields representing attributes of a layer's symbology, labeling, appearance, or diagrams will appear automatically in the attribute table. The exception are attributes that can be modified using the *Label Toolbar* which are hidden by default. Auxiliary fields representing a `Color` have a widget **Color** set by default, otherwise auxiliary fields default to the **Text Edit** widget.

Auxiliary fields that represent attributes that can be modified using the *Label toolbar* are **Hidden** in the attribute table by default. To make a field visible, open the *Attribute Form properties tab* and change the value of an auxiliary field *Widget Type* from **Hidden** to another relevant value. For example, change the **auxiliary_storage_labeling_size** to **Text Edit** or change **auxiliary_storage_labeling_color** to the **Color** widget. Those fields will now be visible in the attribute table.

Auxiliary fields in the attribute table will appear like the following image:

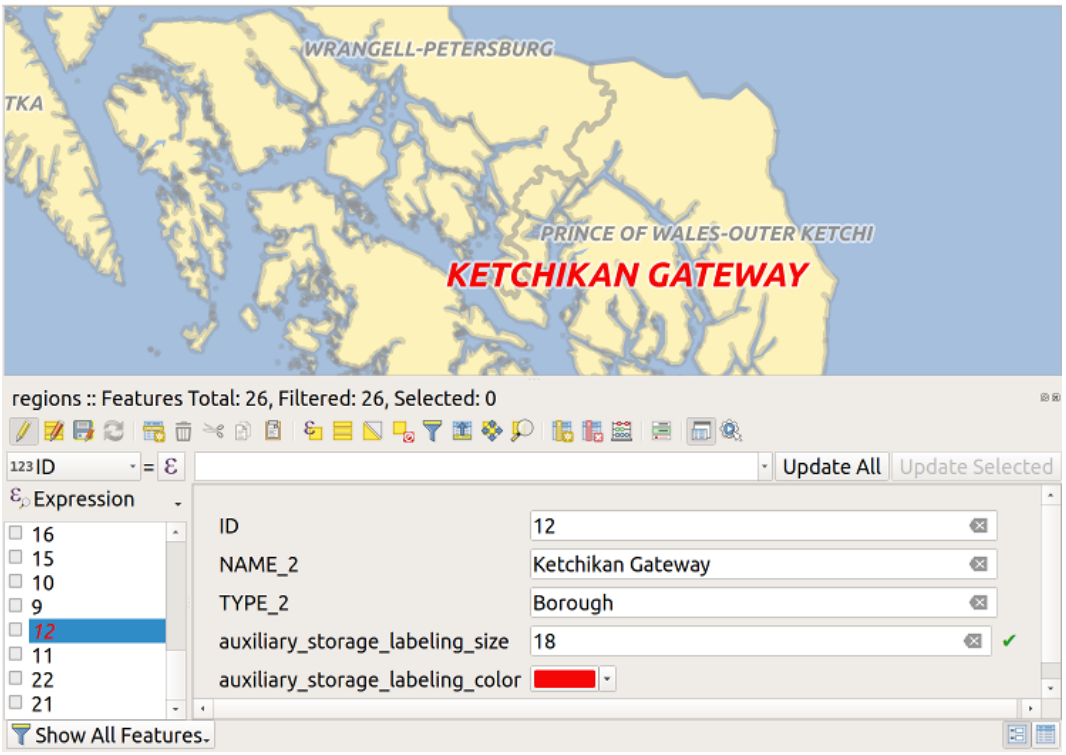


Fig. 12.56: Form with auxiliary fields

Management

The *Auxiliary Layer* menu allows you to manage the auxiliary fields:

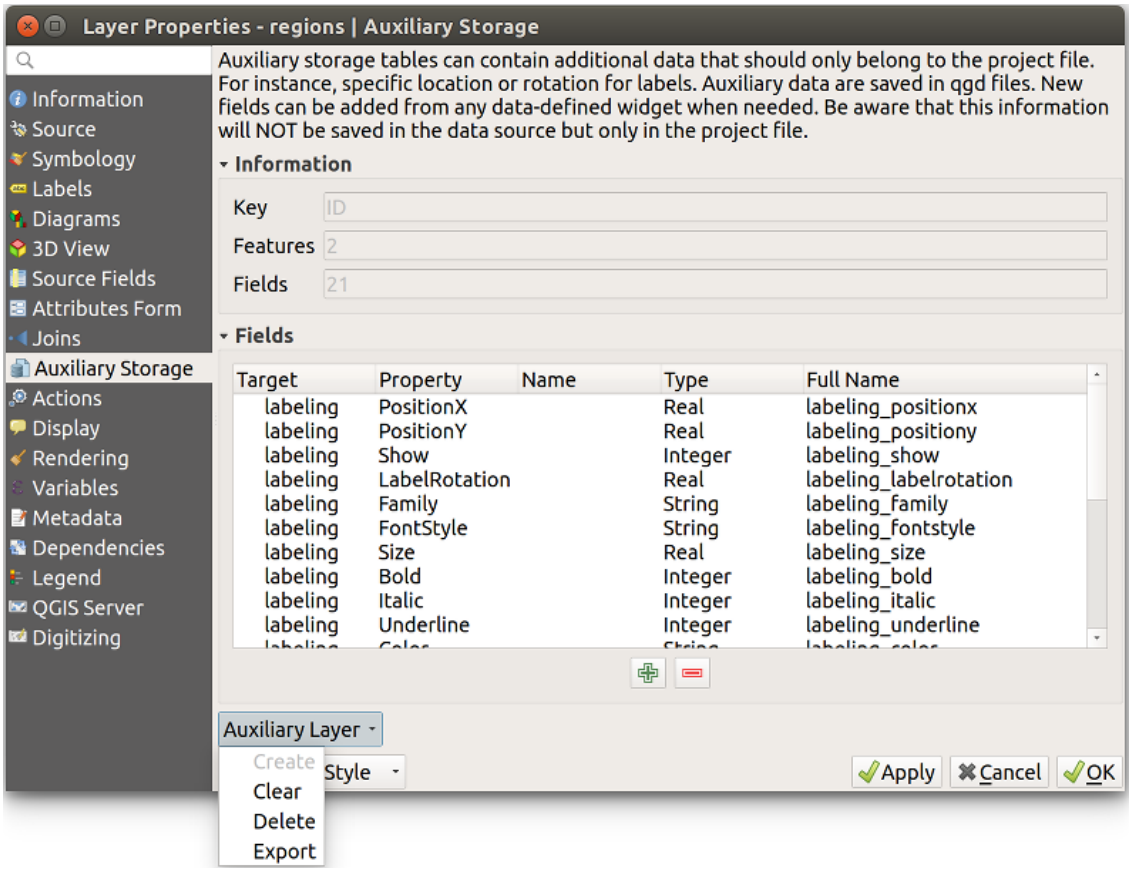


Fig. 12.57: Auxiliary layer management

The first item *Create* is disabled in this case because the auxiliary layer is already created. But in case of a fresh work, you can use this action to create an auxiliary layer. As explained in *Labeling*, a primary key will be needed then.

The *Clear* action allows to keep all auxiliary fields, but remove their contents. This way, the number of features using these fields will fall to `0`.

The *Delete* action completely removes the auxiliary layer. In other words, the corresponding table is deleted from the underlying SQLite database and properties customization are lost.

Finally, the *Export* action allows to save the auxiliary layer as a *new vector layer*. Note that geometries are not stored in auxiliary storage. However, in this case, geometries are exported from the original data source too.

Auxiliary storage database

When you save your project with the `.qgs` format, the SQLite database used for auxiliary storage is saved at the same place but with the extension `.qgd`.

For convenience, an archive may be used instead thanks to the `.qgz` format. In this case, `.qgd` and `.qgs` files are both embedded in the archive.

12.1.12 Actions Properties

The  *Actions* tab provides the ability to perform an action based on the attributes of a feature. This can be used to perform any number of actions, for example, running a program with arguments built from the attributes of a feature or passing parameters to a web reporting tool.

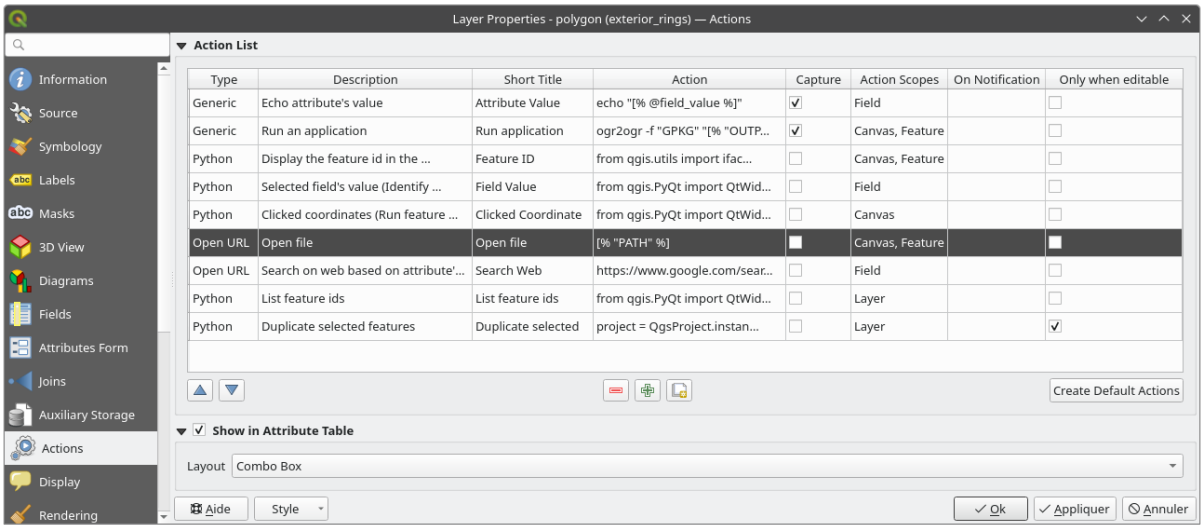


Fig. 12.58: Overview action dialog with some sample actions

Actions are useful when you frequently want to run an external application or view a web page based on one or more values in your vector layer. They are of different types and can be used like this:

- *Generic*, *macOS*, *Windows* and *Unix* actions start an external process.
- *Python* actions execute a Python expression.
- *Generic* and *Python* actions are visible everywhere.
- *macOS*, *Windows* and *Unix* actions are visible only on the respective platform (i.e., you can define three "Edit" actions to open an editor and the users can only see and execute the one "Edit" action for their platform to run the editor).
- *Open URL*: Uses a HTTP GET request to open a provided URL.
- *Submit URL (urlencoded or JSON)*: Same as the *Open URL* action but using a HTTP POST request. Data are posted to a URL, using "application/x-www-form-urlencoded" or "application/json" if the body is a valid JSON.

  An example of action call could be:

  ```
  http://localhost:8000?/[% url_encode(map('file', 'index.php')) %]
  ```

- *Submit URL (multipart)*: Same as the *Open URL* action but using a HTTP POST request. Data are posted to a URL, using "multipart/form-data".

There are several examples included in the dialog. You can load them by clicking on *Create Default Actions*. To edit any of the examples, double-click its row. One example is performing a search based on an attribute value. This concept is used in the following discussion.

The *Show in Attribute Table* allows you to display in the attribute table dialog the checked feature-scoped actions, either as *Combo Box* or as *Separate Buttons* (see *Configuring the columns*).

Defining Actions

To define an attribute action, open the vector *Layer Properties* dialog and click on the *Actions* tab. In the *Actions* tab, click the Add a new action to open the *Edit Action* dialog.

Select the action *Type* and provide a descriptive name for the action. The action itself must contain the name of the application that will be executed when the action is invoked. You can add one or more attribute field values as arguments to the application. When the action is invoked, any set of characters that start with a `%` followed by the name of a field will be replaced by the value of that field. The special characters `%%` will be replaced by the value of the field that was selected from the identify results or attribute table (see *Using Actions*). Double quote marks can be used to group text into a single argument to the program, script or command. Double quotes will be ignored if preceded by a backslash.

Actions can invoke a single process, with arguments, so Boolean operators (such as `&`, `&&`, `;`, `|`) will not work. In UNIX-like operating systems multiple commands can be executed via `bash -c`.

The *Action Scopes* allows you to define where the action should be available. You have following choices:

1. *Field*: action is available when right click in the cell within the attribute table, in the feature form and in the default action button of the main toolbar.
2. *Feature*: action is available when right click in the cell within the attribute table.
3. *Canvas*: action is available in the main action button in the toolbar.
4. *Form*: action is available only in a feature form designed using the *drag-and-drop* mode.
5. *Layer*: action is available in the action button in the attribute table toolbar. Be aware that this type of action involves the entire layer and not the single features.

If you have field names that are substrings of other field names (e.g., `col1` and `col10`), you should indicate that by surrounding the field name (and the % character) with square brackets (e.g., `[%col10]`). This will prevent the `%col10` field name from being mistaken for the `%col1` field name with a `0` on the end. The brackets will be removed by QGIS when it substitutes in the value of the field. If you want the substituted field to be surrounded by square brackets, use a second set like this: `[[%col10]]`.

Using the *Identify Features* tool, you can open the *Identify Results* dialog. It includes a *(Derived)* item that contains information relevant to the layer type. The values in this item can be accessed in a similar way to the other fields by proceeding the derived field name with `(Derived).`. For example, a point layer has an `X` and `Y` field, and the values of these fields can be used in the action with `%(Derived).X` and `%(Derived).Y`. The derived attributes are only available from the *Identify Results* dialog box, not the *Attribute Table* dialog box.

Two example actions are shown below:

- `konqueror https://www.google.com/search?q=%nam`
- `konqueror https://www.google.com/search?q=%%`

In the first example, the web browser konqueror is invoked and passed a URL to open. The URL performs a Google search on the value of the `nam` field from our vector layer. Note that the application or script called by the action must be in the path, or you must provide the full path. To be certain, we could rewrite the first example as: `/opt/kde3/bin/konqueror https://www.google.com/search?q=%nam`. This will ensure that the konqueror application will be executed when the action is invoked.

The second example uses the %% notation, which does not rely on a particular field for its value. When the action is invoked, the %% will be replaced by the value of the selected field in the identify results or attribute table.

Duplicating Actions

QGIS allows you to duplicate existing actions. To duplicate an attribute action, open the vector *Layer Properties* dialog and click on the *Actions* tab. In the *Actions* tab, click the  Duplicate an action to open the *Duplicate Action* dialog. You must have selected at least one existing action in order to create a duplicate.

In the dialogue that appears, make any changes that are necessary. See *Defining Actions* for further information. Once finished, press *OK* to create a duplicate of the action with any changes that you made. If you did not edit the description, or if you changed it to be identical to the description of any other existing action, "_1" will be added to the end of it.

Using Actions

QGIS offers many ways to execute actions you enabled on a layer. Depending on their settings, they can be available:

- in the drop-down menu of  Run Feature Action button from the *Attributes toolbar* or *Attribute table* dialog;
- when right-clicking a feature with the  Identify Features tool (see *Identifying Features* for more information);
- from the *Identify Results* panel, under the *Actions* section;
- as items of an *Actions* column in the *Attribute Table* dialog.

If you are invoking an action that uses the `%%` notation, right-click on the field value in the *Identify Results* dialog or the *Attribute Table* dialog that you wish to pass to the application or script.

Here is another example that pulls data out of a vector layer and inserts it into a file using bash and the `echo` command (so it will only work on  or perhaps ). The layer in question has fields for a species name `taxon_name`, latitude `lat` and longitude `long`. We would like to be able to make a spatial selection of localities and export these field values to a text file for the selected record (shown in yellow in the QGIS map area). Here is the action to achieve this:

```
bash -c "echo \"%taxon_name %lat %long\" >> /tmp/species_localities.txt"
```

After selecting a few localities and running the action on each one, opening the output file will show something like this:

```
Acacia mearnsii -34.0800000000 150.0800000000
Acacia mearnsii -34.9000000000 150.1200000000
Acacia mearnsii -35.2200000000 149.9300000000
Acacia mearnsii -32.2700000000 150.4100000000
```

As an exercise, we can create an action that does a Google search on the `lakes` layer. First, we need to determine the URL required to perform a search on a keyword. This is easily done by just going to Google and doing a simple search, then grabbing the URL from the address bar in your browser. From this little effort, we see that the format is https://www.google.com/search?q=QGIS, where `QGIS` is the search term. Armed with this information, we can proceed:

1. Make sure the `lakes` layer is loaded.
2. Open the *Layer Properties* dialog by double-clicking on the layer in the legend, or right-click and choose *Properties* from the pop-up menu.
3. Click on the *Actions* tab.
4. Click  Add a new action.
5. Choose the *Open URL* action type,
6. Enter a name for the action, for example `Google Search`.
7. Additionally you can add a *Short Name* or even an *Icon*.

8. Choose the *Action Scopes*. See *Defining Actions* for further information. Leave the default settings for this example.
9. For the action, add the URL used for doing a Google search, up to but not including the search term: `https://www.google.com/search?q=`
10. The text in the *Action* field should now look like this:

```
https://www.google.com/search?q=
```

11. Click on the drop-down box containing the field names for the `lakes` layer. It's located just to the left of the *Insert* button.
12. From the drop-down box, select *NAMES* and click *Insert*.
13. Your action text now looks like this:

```
https://www.google.com/search?q=[%NAMES%]
```

14. To finalize and add the action, click the *OK* button.

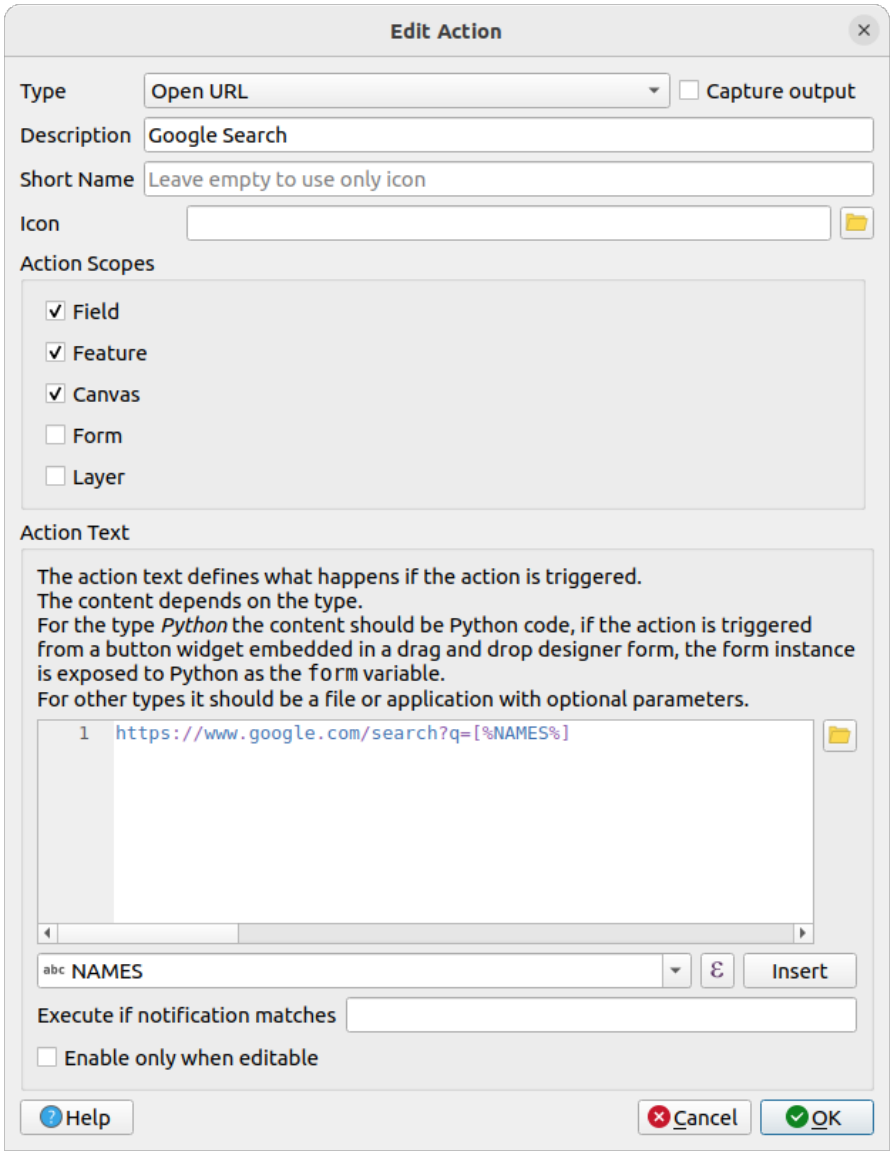


Fig. 12.59: Edit action dialog configured with the example

This completes the action, and it is ready to use.

Close the *Layer Properties* dialog and zoom in to an area of interest. Make sure the `lakes` layer is active and identify a lake. In the result box you'll now see that our action is visible:

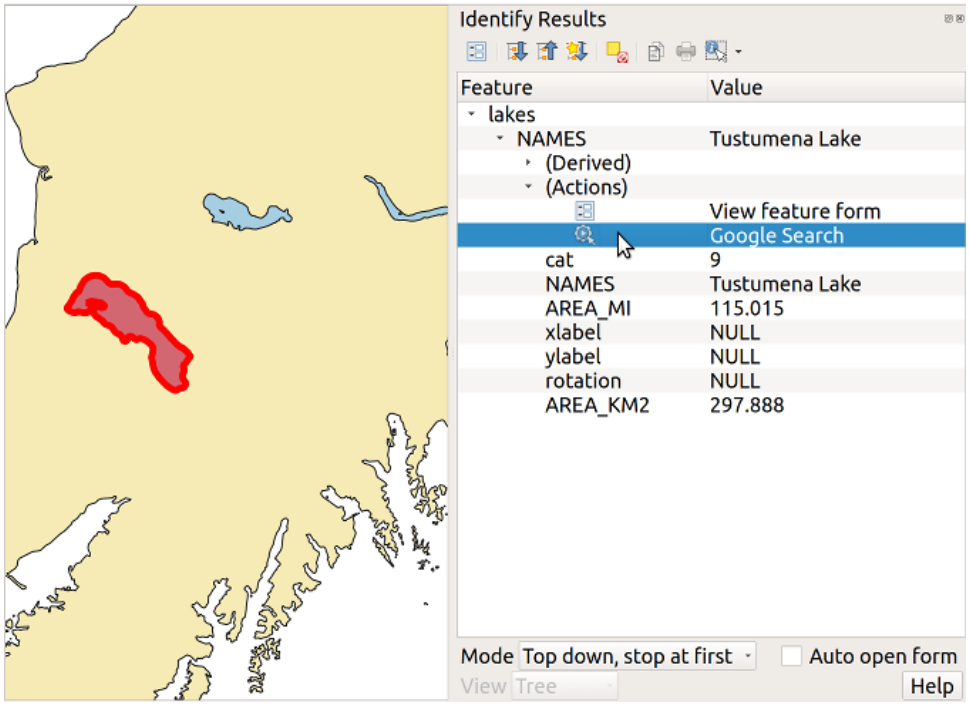


Fig. 12.60: Select feature and choose action

When we click on the action, it brings up our default browser and navigates to the URL https://www.google.com/search?q=Tustumena. It is also possible to add further attribute fields to the action. Therefore, you can add a + to the end of the action text, select another field and click on *Insert Field*. In this example, there is just no other field available that would make sense to search for.

You can define multiple actions for a layer, and each will show up in the *Identify Results* dialog.

You can also invoke actions from the attribute table by selecting a row and right-clicking, then choosing the action from the pop-up menu.

There are all kinds of uses for actions. For example, if you have a point layer containing locations of images or photos along with a file name, you could create an action to launch a viewer to display the image. You could also use actions to launch web-based reports for an attribute field or combination of fields, specifying them in the same way we did in our Google search example.

We can also make more complex examples, for instance, using **Python** actions.

Usually, when we create an action to open a file with an external application, we can use absolute paths, or eventually relative paths. In the second case, the path is relative to the location of the external program executable file. But what about if we need to use relative paths, relative to the selected layer (a file-based one, like Shapefile or SpatiaLite)? The following code will do the trick:

```
command = "firefox"
imagerelpath = "images_test/test_image.jpg"
layer = qgis.utils.iface.activeLayer()
import os.path
layerpath = layer.source() if layer.providerType() == 'ogr'
  else (qgis.core.QgsDataSourceURI(layer.source()).database()
    if layer.providerType() == 'spatialite' else None)
path = os.path.dirname(str(layerpath))
image = os.path.join(path,imagerelpath)
```

(continues on next page)

(continued from previous page)

```
import subprocess
subprocess.Popen( [command, image ] )
```

We just have to remember that the action is one of type *Python* and the *command* and *imagerelpath* variables must be changed to fit our needs.

But what about if the relative path needs to be relative to the (saved) project file? The code of the Python action would be:

```
command = "firefox"
imagerelpath = "images_test/test_image.jpg"
projectpath = qgis.core.QgsProject.instance().fileName()
import os.path
path = os.path.dirname(str(projectpath)) if projectpath != '' else None
image = os.path.join(path, imagerelpath)
import subprocess
subprocess.Popen( [command, image ] )
```

Another Python action example is the one that allows us to add new layers to the project. For instance, the following examples will add to the project respectively a vector and a raster. The names of the files to be added to the project and the names to be given to the layers are data driven (*filename* and *layername* are column names of the table of attributes of the vector where the action was created):

```
qgis.utils.iface.addVectorLayer('/yourpath/[% "filename" %].shp',
  '[% "layername" %]', 'ogr')
```

To add a raster (a TIF image in this example), it becomes:

```
qgis.utils.iface.addRasterLayer('/yourpath/[% "filename" %].tif',
  '[% "layername" %]')
```

12.1.13 Display Properties

The *Display* tab helps you configure fields to use for feature identification:

- The *Display name*: based on a field or an *expression*. By default, this is set to the first field in the layer if no field with `<name>` component exists. This is used as:
  - the label shown on top of the feature information in the *Identify tool* results
  - the field used in the *locator bar* when looking for features in all layers
  - the feature identifier in the attribute table *form view*
  - the feature identifier when the map or layout is exported to a layered output format such as Geospatial PDF
  - the map tip information, i.e. the message displayed in the map canvas when hovering over a feature of the active layer with the Show Map Tips icon pressed. Applicable when *Enable Map Tips* is active and no *HTML Map Tip* is set.
- *Enable Map Tips* controls whether to display map tips for the layer
- The *HTML Map Tip* provides a complex and full HTML text editor for map tips, mixing QGIS expressions and html styles and tags (multiline, fonts, images, hyperlink, tables, …). You can check the result of your code sample in the *Preview* frame (also convenient for previewing the *Display name* output). Additionally, you can select and edit existing expressions using the *Insert/Edit Expression* button.

Note

Understanding the *Insert/Edit Expression* button behavior

If you select some text within an expression (between "[%" and "%]"), or if no text is selected but the cursor is inside an expression, the whole expression will be automatically selected for editing. If the cursor or a selected text is outside an expression, the dialog opens with the selection.

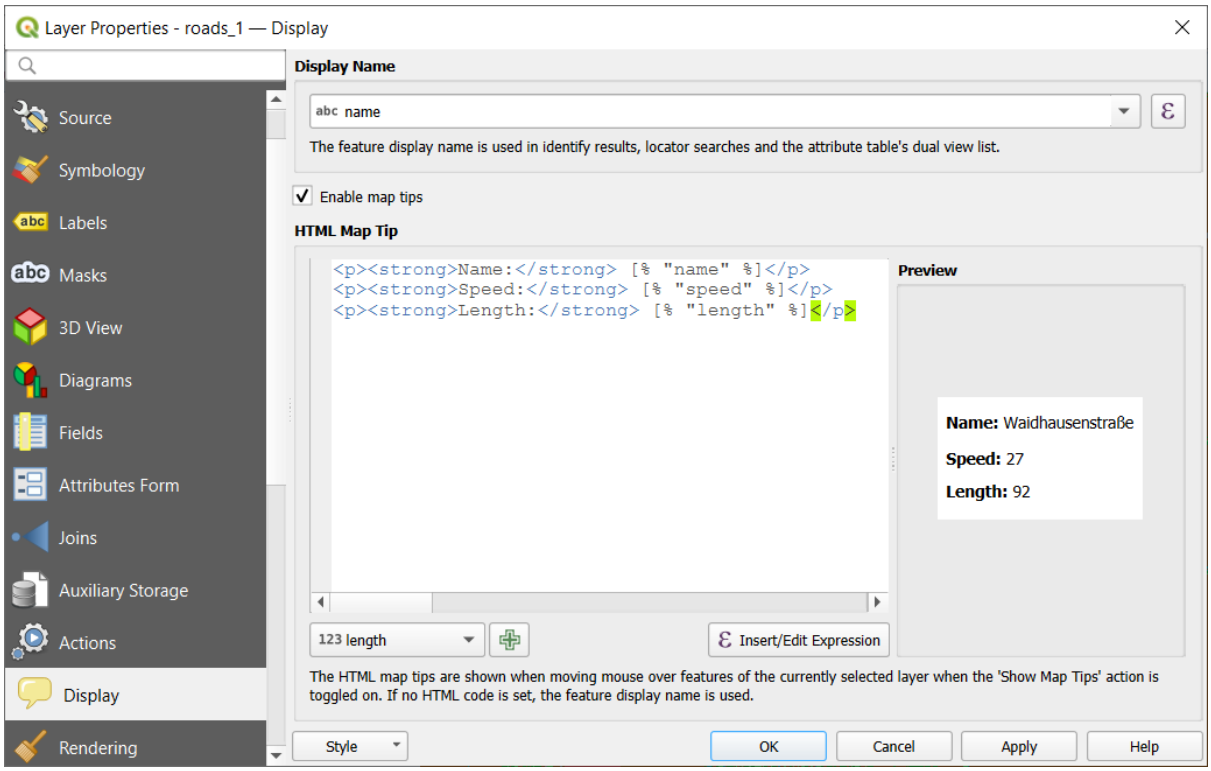


Fig. 12.61: HTML code for map tip

To display map tips:

1. Select the menu option *View* ► *Show Map Tips* or click on the Show Map Tips icon of the *Attributes Toolbar*.
2. Make sure that the layer you target is active and has the *Enable Map Tips* property checked.
3. Move over a feature, and the corresponding information will be displayed over.

Map tip is a cross-layer feature meaning that once activated, it stays on and applies to any map tip enabled layer in the project until it's toggled off.



Fig. 12.62: Map tip made with HTML code

12.1.14 Rendering Properties

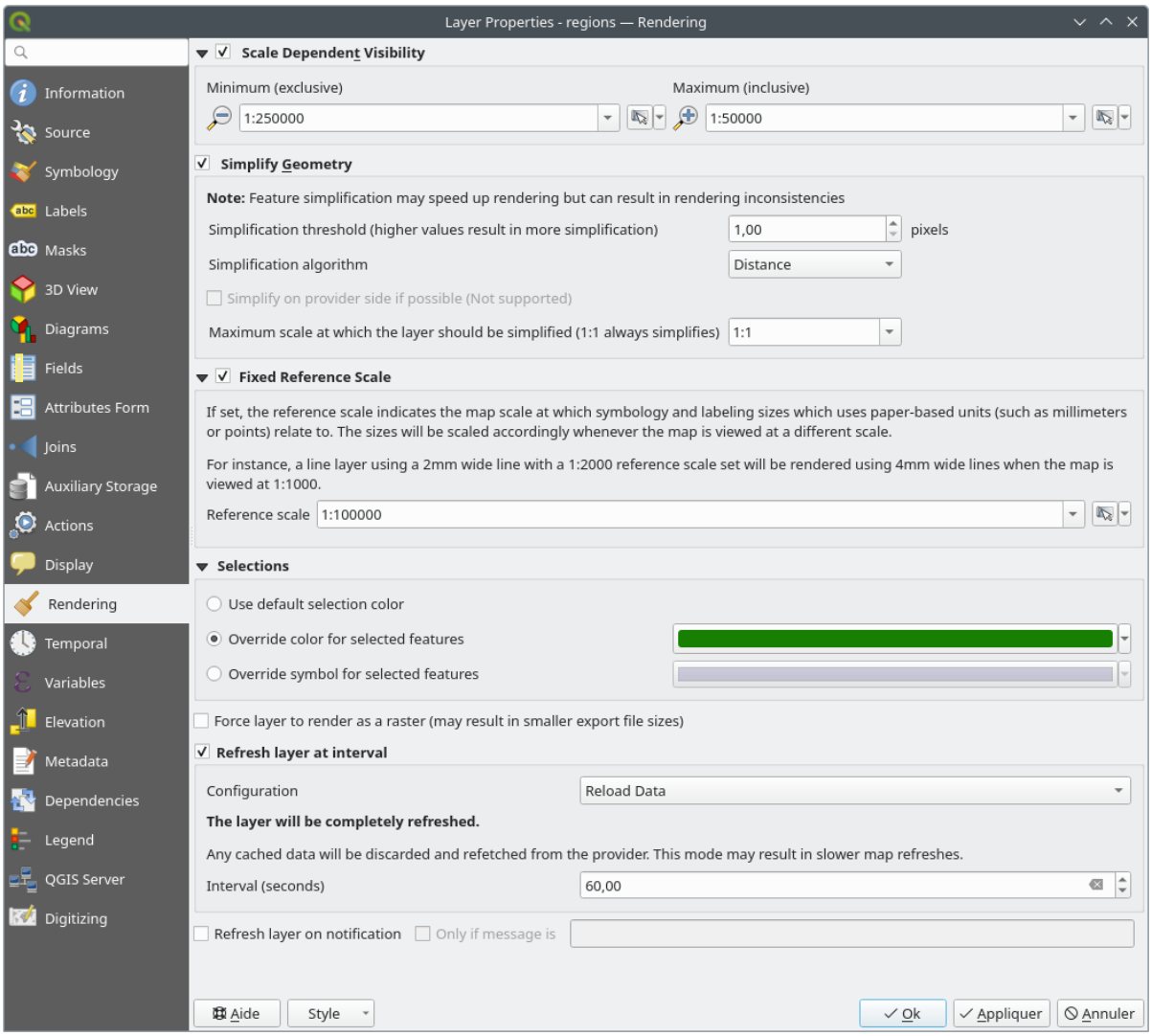


Fig. 12.63: Layer Rendering Properties dialog

The *Rendering* tab offers following properties:

- Under *Scale dependent visibility*, you can set the *Maximum (inclusive)* and *Minimum (exclusive)* scales, defining a range of scales in which features will be visible. Out of this range, they are hidden. The Set to current canvas scale button helps you use the current map canvas scale as boundary of the range visibility. See *Visibility Scale Selector* for more information.

  > **Note**
  >
  > You can also activate scale dependent visibility on a layer from within the *Layers* panel: right-click on the layer and in the contextual menu, select *Set Layer Scale Visibility*.

- QGIS offers support for on-the-fly feature generalisation. This can improve rendering times when drawing many complex features at small scales. This feature can be enabled or disabled in the layer settings using the *Simplify geometry* option. There is also a global setting that enables generalisation by default for newly added layers (see *global simplification* for more information).

> **Note**
>
> Feature generalisation may introduce artefacts into your rendered output in some cases. These may include slivers between polygons and inaccurate rendering when using offset-based symbol layers.

- The *Fixed reference scale* indicates the map scale at which symbology and labeling sizes which uses paper-based units (such as millimeters or points) relate to. The sizes will be scaled accordingly whenever the map is viewed at a different scale.

  For instance, a line layer using a 2mm wide line with a 1:2000 *Reference scale* set will be rendered using 4mm wide lines when the map is viewed at 1:1000.

- The *Selections* group allows you to control whether a specific color or symbol should be used in place of the defaults (*Project properties* ► *General* ► *Selection color*) for a particular layer. This is useful to improve the visibility of selected features with certain symbology:

  - *Use default selection color*
  - *Override color for selected features*: e.g. if the layer uses a yellow color by default and the standard yellow selection is not visible.
  - *Override symbol for selected features*: e.g. if a line layer uses a thin symbol, and coloring the lines might not make them visible enough, overriding the symbol with a thicker line can help. Also, if the layer uses raster symbology or gradient fills/lines/shapeburst with color ramp symbology, then the default selection color is not applied at all; being able to set a specific simpler symbol to use for selected features in the layer can help.

- Rendering extremely detailed layers (e.g. polygon layers with a huge number of nodes), can cause layout exports in PDF/SVG format to be huge as all nodes are included in the exported file. This can also make the resultant file very slow to work with/open in other programs.

  Checking *Force layer to render as raster* forces these layers to be rasterised so that the exported files won't have to include all the nodes contained in these layers and the rendering is therefore sped up.

  You can also do this by forcing the layout to export as a raster, but that is an all-or-nothing solution, given that the rasterisation is applied to all layers. Alternatively, you can rely on geometry simplification in *layout export settings*.

- *Refresh layer at interval*: controls whether and how regular a layer can be refreshed. Available *Configuration* options are:

  - *Reload data*: the layer will be completely refreshed. Any cached data will be discarded and refetched from the provider. This mode may result in slower map refreshes.
  - *Redraw layer only*: this mode is useful for animation or when the layer's style will be updated at regular intervals. Canvas updates are deferred in order to avoid refreshing multiple times if more than one layer has an auto update interval set.

  It is also possible to set the *Interval (seconds)* between consecutive refreshments.

- Depending on the data provider (e.g. PostgreSQL), notifications can be sent to QGIS when changes are applied to the data source, out of QGIS. Use the *Refresh layer on notification* option to trigger an update. You can also limit the layer refresh to a specific message set in the *Only if message is* text box.

12.1.15 Temporal Properties

The *Temporal* tab provides options to control the rendering of the layer over time. Such dynamic rendering requires the *temporal navigation* to be enabled over the map canvas.

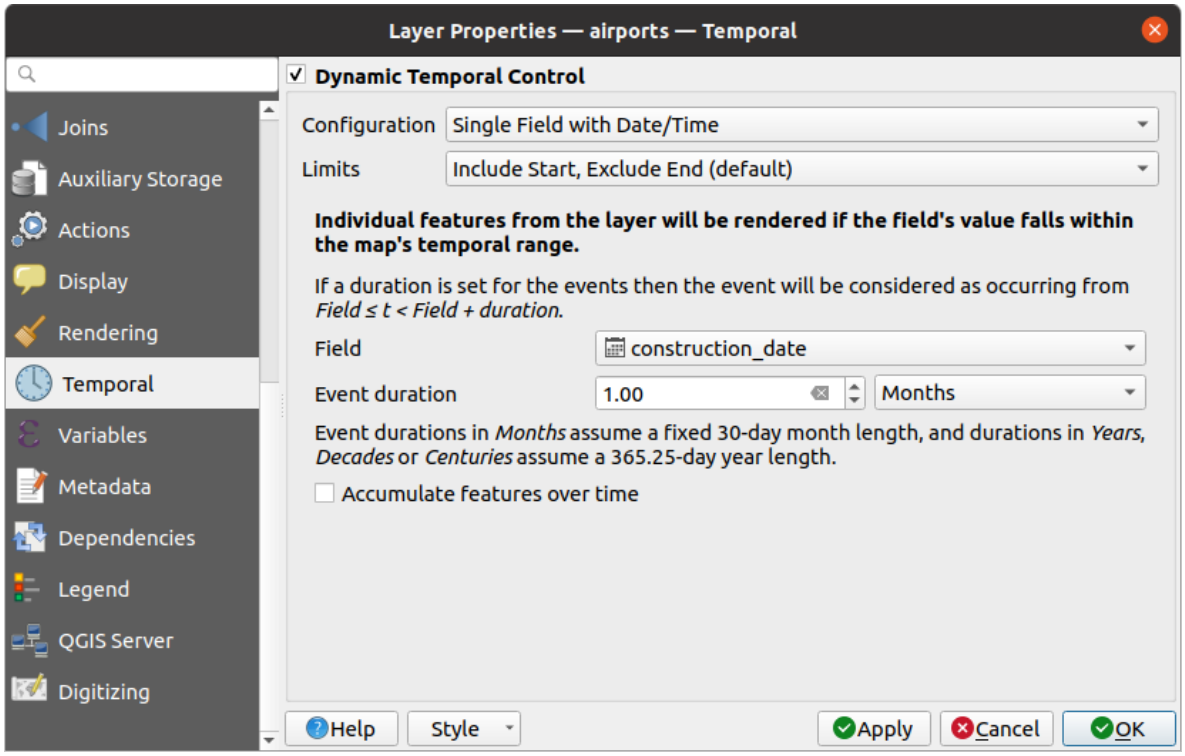


Fig. 12.64: Vector layer temporal properties dialog

Check the *Dynamic Temporal Control* option to configure the vector layer temporal rendering. Depending on the structure of your dataset, you may want to use one of the provided *Configuration* options:

- *Fixed time range*: all the features are rendered if the map canvas temporal frame overlaps the given *Start date* and *End date* range.
- *Single field with date/time*: features are rendered if their *Field*'s value falls within the map canvas temporal frame. An *Event duration* can be set. With checking the *Accumulate features over time* option, all features which occur before or within the map's temporal range will continue to be rendered. The event duration is thus ignored.
- *Separate fields for start and end date/time*: features are rendered if the range specified by their *Start field* and *End field* values overlaps the map canvas temporal.
- *Separate fields for start and event duration*: features are rendered if the range defined by their *Start field* and *Event duration field* values overlaps the map canvas temporal.
- *Start and end date/time from expressions*: features are rendered if the time range specified by the fields *Start expression* and *End expression* overlaps the map canvas temporal.
- *Redraw layer only*: the layer is redrawn at each new animation frame but no time-based filtering is applied to the features. It's useful when the layer uses time-based expression values for renderer settings (e.g. data-defined symbology).

It is also possible to set the *Limits* of the features time range as:

- *Include start, exclude end*
- *Include start, include end*

12.1.16 Variables Properties

The *Variables* tab lists all the variables available at the layer's level (which includes all global and project's variables).

It also allows the user to manage layer-level variables. Click the button to add a new custom layer-level variable. Likewise, select a custom layer-level variable from the list and click the button to remove it.

More information on variables usage in the General Tools *Storing values in Variables* section.

12.1.17 Elevation Properties

The *Elevation* tab provides options to control the layer elevation properties within a *3D map view* and its appearance in the *profile tool charts*. Specifically, you can set:

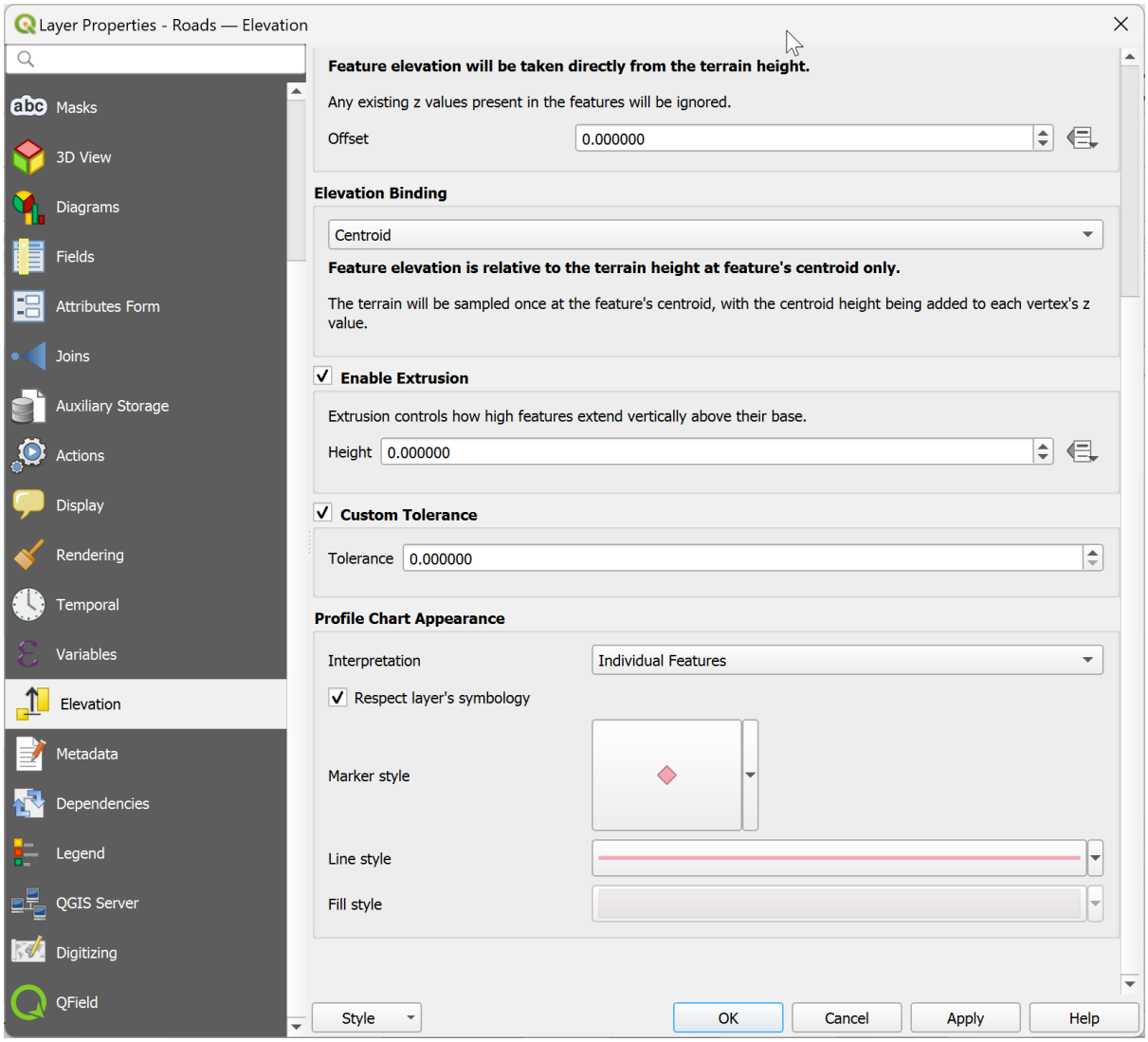


Fig. 12.65: Vector layer elevation properties dialog

- *Vertical Reference System*: If the CRS of your vector layer is a compound one (including a Z dimension), then the vertical CRS used for the layer will be the vertical component of the layer CRS. In this case, you cannot

manually set a different vertical CRS. If your layer CRS is horizontal (2D), then you can select a specific vertical CRS by clicking on the  Select CRS. Vertical reference systems are supported for vector layers by:

  - *Elevation profiles*
  - *Identify Tool Results*
  - *3D map views*

- *Elevation Clamping*: defines how and whether the features altitude should be:
  - *Clamped to terrain*: takes elevation directly from the terrain height and ignores any existing Z values in the features. A data-defined *Offset* value from the terrain can also be filled.
  - *Relative to terrain*: any existing Z values in the features are added to the terrain height. A *Scale* factor followed by a data-defined *Offset* can be used to adjust the elevation. This option is not available for 2D geometry layers.
  - *Absolute*: ignores the terrain height and directly takes Z values from the features for the elevation. A *Scale* factor followed by a data-defined *Offset* can be used to adjust the elevation. For 2D geometry layers (with no Z values), a data-defined *Base height* can instead be set.
-  *Enable extrusion*: you can set a *Height* to control how high features vertically extend above their base. This is convenient to indicate that a 2D geometry layers, e.g. a polygon building footprints layer, actually represents 3D objects.
-  *Custom tolerance*: allows you to override the global sampling tolerance defined in the elevation profile settings, specifically for this vector layer. When enabled, you can set *Tolerance* that determines how the layer is sampled along the profile line. This is useful when you need finer or coarser sampling than the global default, for example, to improve performance or accuracy for complex geometries.
- *Elevation Binding*: only relevant when combining an *Elevation clamping* relying on the terrain with a line or polygon layer, this option controls how feature elevation is set relative to the terrain height. The terrain can be sampled:
  - at the feature's *Centroid*, with the centroid height being added to each vertex's z value
  - at every individual *Vertex* before being added to the vertex's z value
- *Profile Chart Appearance*: controls how features are rendered when drawing a profile chart. Two main *Interpretation* modes are available:
  - as *Individual features*: samples discrete positions where the cross section profile line intersects the vector features. That intersection can be represented as point, line or surface depending on the layer type and whether an extrusion is applied.

    With checking  *Respect layer symbology*, features will be rendered on the profile chart with their corresponding *layer styling* (allowing e.g. categorized classes to be visible on the profile chart). If the profile symbol type does not match the layer's renderer symbol types, only the symbol color from the renderer is applied to the profile symbol.

    Depending on the layer settings, profile symbols can be represented with a custom style, using:

    * *Marker style*: for non-extruded point and line features, and for non-extruded polygon features touched by the profile line
    * *Line style*: for extruded point and line features, and for non-extruded polygon features intersected by the profile line
    * *Fill style*: for extruded polygon features
  - as *Continuous Surface (e.g. contours)*: the elevation chart will be rendered as a surface instead of separate features by joining the sampled elevation results into a continuous line. This can enhance visualisation and is designed for vector layers which represent a continuous elevation surface, e.g. contour lines or surveyed elevation points. The profile *Style* can be set as:
    * a *Line* with a specific *Line style*

    * an elevation surface rendered using a fill symbol either above (*Fill above*) or below (*Fill below*) the elevation curve line. The surface symbology is represented using:
        - a *Fill style*
        - and a *Limit*: the maximum (respectively minimum) altitude determining how high the fill surface will be

  Moreover, you can check *Show markers at sampled points* to make them visible over the interpretation line and assign them a *Marker style*.

12.1.18 Metadata Properties

The *Metadata* tab provides you with options to create and edit a metadata report on your layer. See *Metadata* for more information.

12.1.19 Dependencies Properties

The *Dependencies* tab allows to declare data dependencies between layers. A data dependency occurs when a data modification in a layer, not by direct user manipulation, may modify data of other layers. This is the case for instance when geometry of a layer is updated by a database trigger or custom PyQGIS scripting after modification of another layer's geometry.

In the *Dependencies* tab, you can select any layers which may externally alter the data in the current layer. Correctly specifying dependent layers allows QGIS to invalidate caches for this layer when the dependent layers are altered.

12.1.20 Legend Properties

The *Legend* properties tab provides you with advanced settings for the *Layers panel* and/or the *print layout legend*. These options include:

- Depending on the symbology applied to the layer, you may end up with several entries in the legend, not necessarily readable/useful to display. The *Legend placeholder image* helps you *select an image* for replacement, displayed both in the *Layers* panel and the print layout legend.
- *Show label legend*: Displays overviews of the different label settings as entries in the legends. The *label style* is previewed along with the description.
- *Text on symbols*: In some cases it can be useful to add extra information to the symbols in the legend. With this frame, you can affect to any of the symbols used in the layer symbology a text that is displayed over the symbol, in both *Layers* panel and print layout legend. This mapping is done by typing each text next to the symbol in the table widget or filling the table using the *Set Labels from Expression* button. Text appearance is handled through the font and color selector widgets of the *Text Format* button.

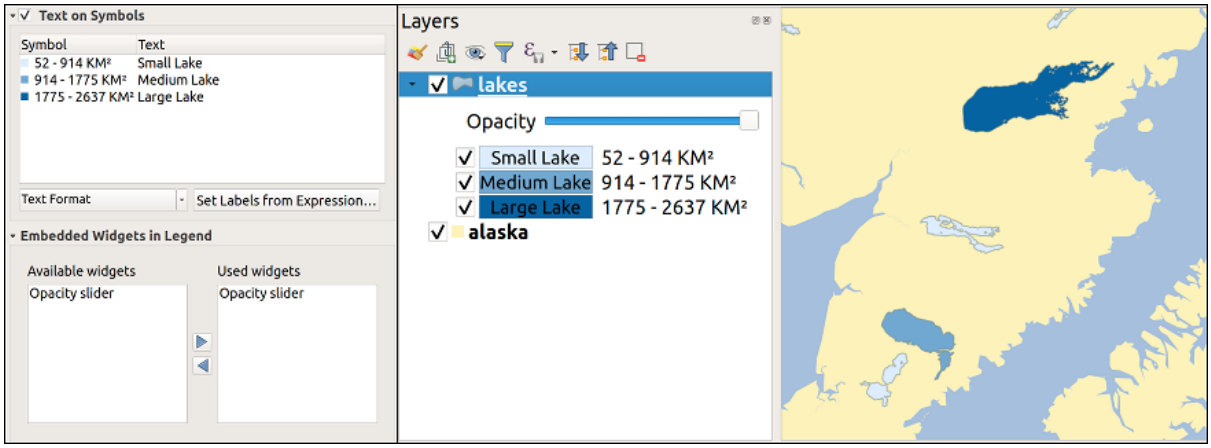


Fig. 12.66: Setting text on symbols (left) and its rendering in the *Layers* panel (right)

- a list of widgets you can embed within the layer tree in the Layers panel. The idea is to have a way to quickly access some actions that are often used with the layer (setup transparency, filtering, selection, style or other stuff…).

  By default, QGIS provides transparency widget but this can be extended by plugins registering their own widgets and assign custom actions to layers they manage.

12.1.21 QGIS Server Properties

The *QGIS Server* tab consists of *Description*, *Attribution*, *Metadata URL*, and *Legend URL* sections.

From the *Description* section, you can change the *Short name* used to reference the layer in requests (to learn more about short names, read services_basics_short_name). You can also add or edit a *Title*, an alternative *WFS Title* and *Abstract* for the layer, or define a *Keyword list* here. These keyword lists can be used in a metadata catalog. If you want to use a title from an XML metadata file, you have to fill in a link in the *Data URL* field.

Use *Attribution* to get attribute data from an XML metadata catalog.

In *Metadata URL*, you can add the general paths to the XML metadata catalog. This information will be saved in the QGIS project file for subsequent sessions and will be used for QGIS Server.

In the *Legend URL* section, you can provide the url of a legend image in the url field. You can use the Format drop-down option to apply the appropriate format of the image. Currently png, jpg and jpeg image formats are supported.

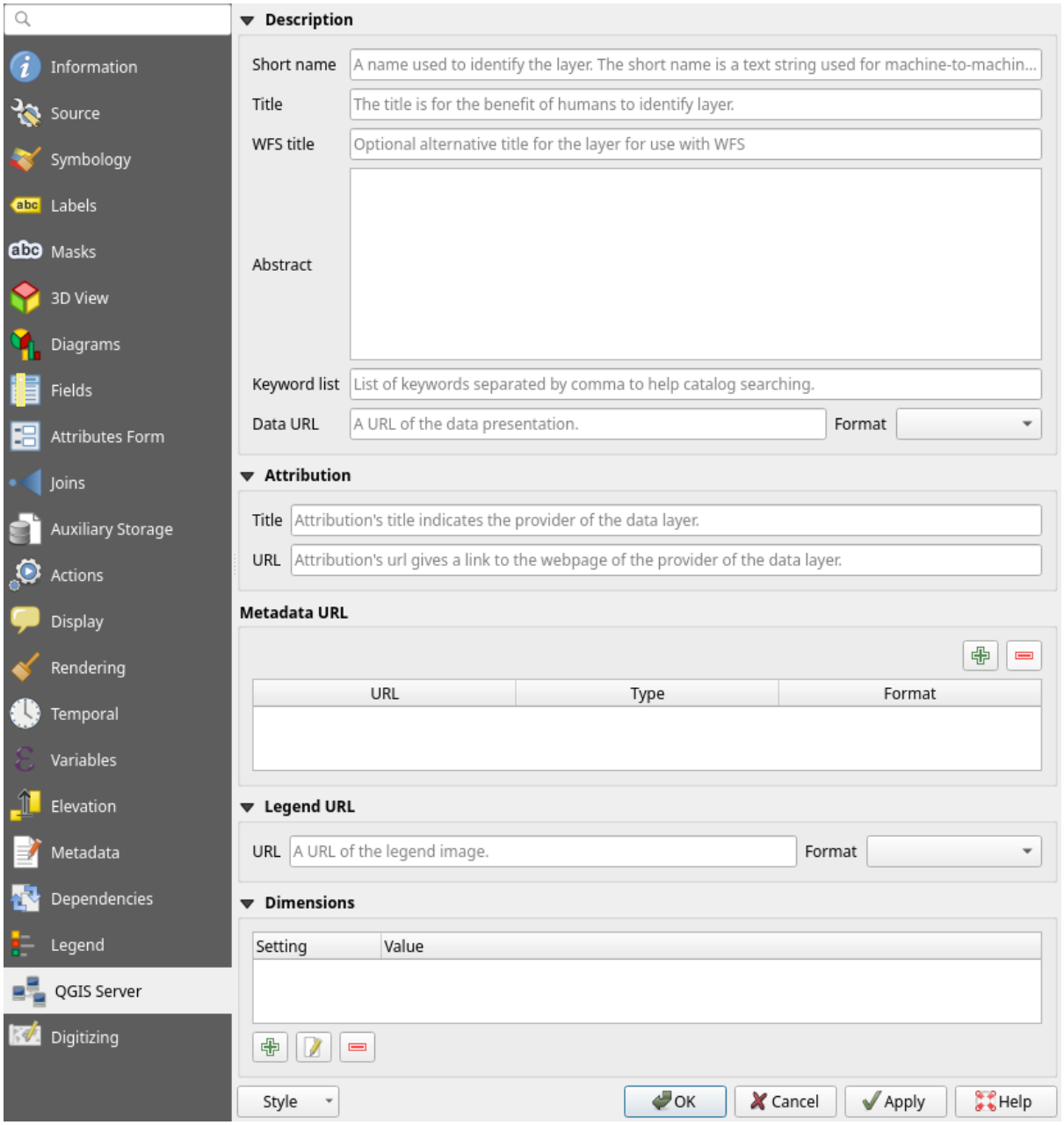


Fig. 12.67: QGIS Server tab in vector layers properties dialog

To learn more about QGIS Server, read the QGIS-Server-manual.

12.1.22 Digitizing Properties

The *Digitizing* tab gives access to options that help to ensure the quality of digitized geometries.

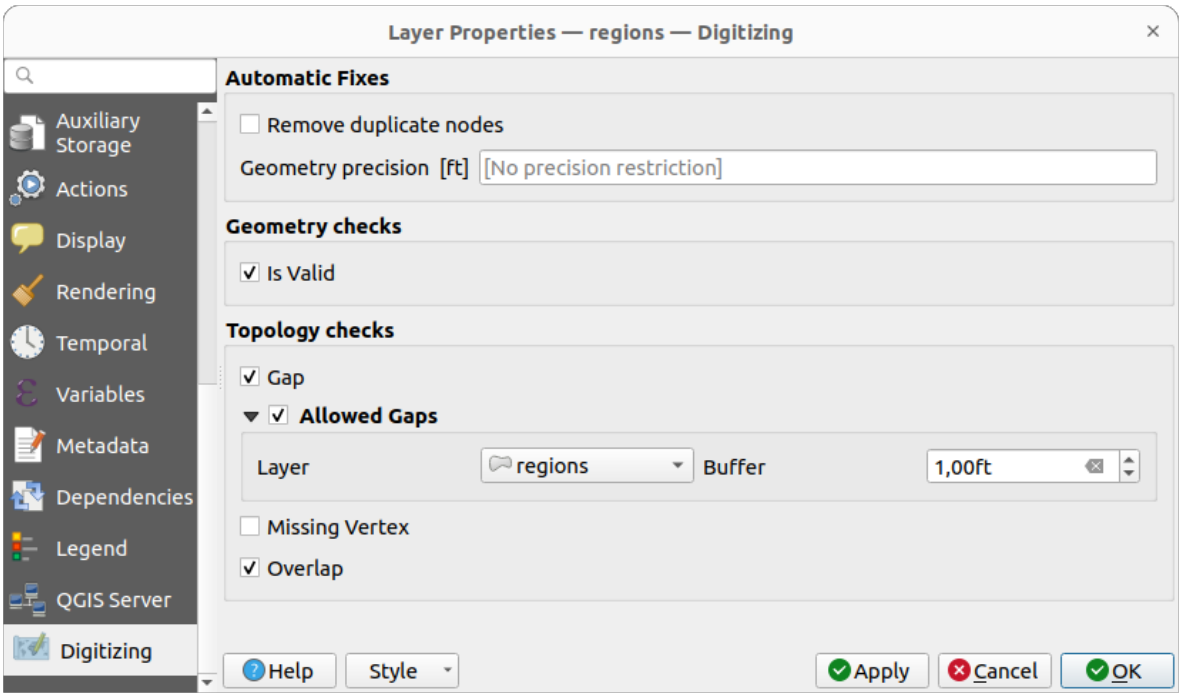


Fig. 12.68: The QGIS Digitizing tab in the vector layers properties dialog

Automatic Fixes

Options in the *Automatic Fixes* section will directly affect the vertices of any geometry which is added or modified. If the *Remove duplicate nodes* option is checked, any two subsequent vertices with exactly the same coordinates will be removed. If the *Geometry precision* is set, all vertices will be rounded to the closest multiple of the configured geometry precision. The rounding will happen in the layer coordinate reference system. Z and M values are not rounded. With many map tools, a grid is shown on the canvas while digitizing.

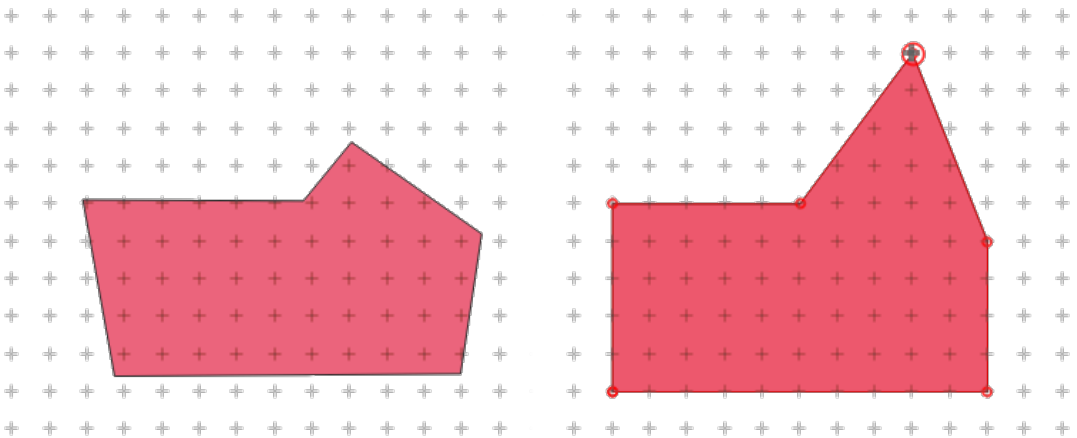


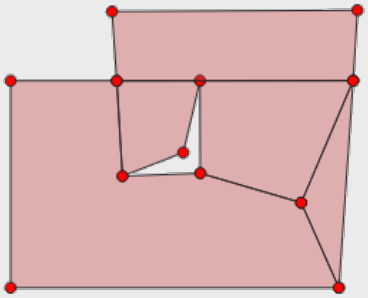
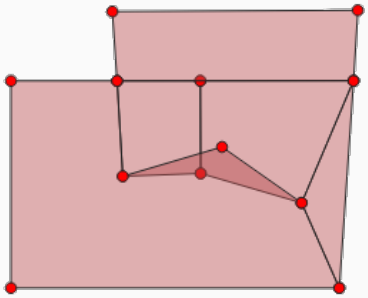
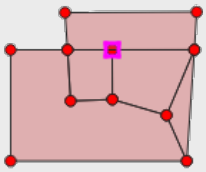
Fig. 12.69: Moving the top vertex snaps all the vertices to the grid

Geometry Checks

In the *Geometry checks* section, additional validations on a per geometry basis can be activated. Immediately after any geometry modification, failures in these checks are reported to the user in the *Geometry validation* panel. As long as a check is failing, it is not possible to save the layer. The *Is valid* check will run basic validity checks like self intersection on geometries.

Topology Checks

In the *Topology checks* section, additional topology validation checks can be activated. Topology checks will be executed when the user saves the layer. Check errors will be reported in the *Geometry validation* panel. As long as validation errors are present, the layer can not be saved. Topology checks are executed in the area of the bounding box of the modified features. Since other features may be present in the same area, topological errors concerning these features are reported as well as errors introduced in the current edit session.

Topology check option	Illustration
The *Gap* check will check for gaps between neighbouring polygons.	
The *Overlap* check will check for overlaps between neighbouring polygons.	
The *Missing vertex* check will check for shared boundaries of neighbouring polygons where one border misses a vertex which is present on the other one.	

Gap check exceptions

Sometimes it is desirable to keep gaps inside an area in a polygon layer that otherwise is fully covered by polygons. For example, a land use layer may have acceptable holes for lakes. It is possible to define areas that are ignored in the gap check. Since gaps inside these areas are allowed, we will refer to them as *Allowed Gaps* areas.

In the options for the gap checks under *Allowed Gaps*, an *Allowed Gaps layer* can be configured.

Whenever the gap check is executed, gaps which are covered by one or more polygons in the *Allowed Gaps Layer* are not reported as topology errors.

It is also possible to configure an additional *Buffer*. This buffer is applied to each polygon on the *Allowed Gaps Layer*. This makes it possible to make the tests less susceptible to small changes in the outlines at the borders of gaps.

When *Allowed Gaps* are enabled, an additional button (*Add Allowed Gap*) for detected gap errors is available in the geometry validation dock, where gaps are reported during digitizing. If the *Add Allowed Gap* button is pushed, a new polygon with the geometry of the detected gap is inserted into the *Allowed Gaps Layer*. This makes it possible to quickly flag gaps as allowed.

Geometry validation panel

The *Geometry Validation* panel is triggered when any of the abovementioned digitizing checks finds an error. The dialog provides you with the list of errors and their description, and you can to browse the list using the keyboard arrows or dedicated arrows.

You'll need to address all the issues before you can save edits to the layer. To do so:

1. Select an error, and it's possible to:
   -  *Zoom to Feature(s)*
   -  *Zoom to problem*
2. Pick the usual *digitizing tools* to fix the issue.

12.2 Working with the Attribute Table

The attribute table displays information on features of a selected layer. Each row in the table represents a feature (with or without geometry), and each column contains a particular piece of information about the feature. Features in the table can be searched, selected, moved or even edited.

12.2.1 Foreword: Spatial and non-spatial tables

QGIS allows you to load spatial and non-spatial layers. This currently includes tables supported by GDAL and delimited text, as well as the PostgreSQL, MS SQL Server, SpatiaLite and Oracle providers. All loaded layers are listed in the *Layers* panel. Whether a layer is spatially enabled or not determines whether you can interact with it on the map.

Non-spatial tables can be browsed and edited using the attribute table view. Furthermore, they can be used for field lookups. For example, you can use columns of a non-spatial table to define attribute values, or a range of values that are allowed, to be added to a specific vector layer during digitizing. Have a closer look at the edit widget in section *Attributes Form Properties* to find out more.

12.2.2 Introducing the attribute table interface

To open the attribute table for a vector layer, activate the layer by clicking on it in the *Layers Panel*. Then, from the main *Layer* menu, choose *Open Attribute Table*. It is also possible to right-click on the layer and choose *Open Attribute Table* from the drop-down menu, or to click on the *Open Attribute Table* button in the Attributes toolbar. If you prefer shortcuts, `F6` will open the attribute table. `Shift+F6` will open the attribute table filtered to selected features and `Ctrl+F6` will open the attribute table filtered to visible features.

This will open a new window that displays the feature attributes for the layer (*figure_attributes_table*). According to the setting in *Settings* ► *Options* ► *Data sources* menu, the attribute table will open in a docked window or a regular window. The total number of features in the layer and the number of currently selected/filtered features are shown in the attribute table title, as well as if the layer is spatially limited.

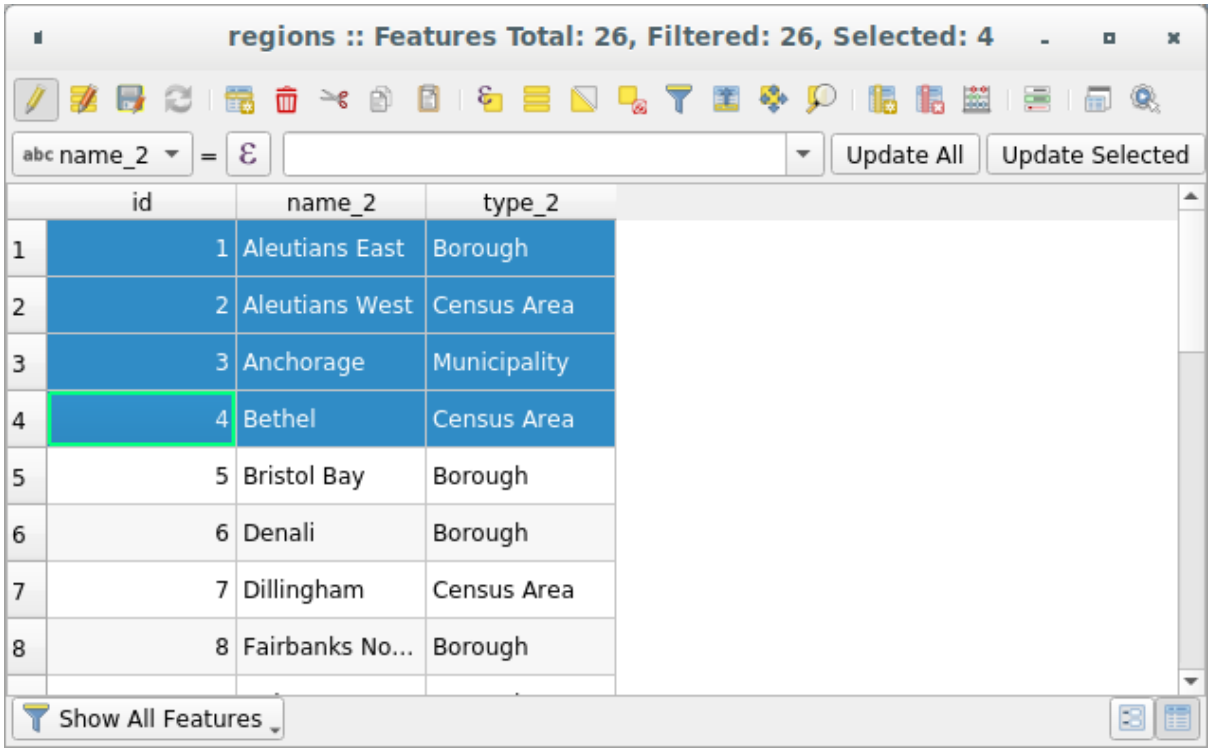


Fig. 12.70: Attribute Table for regions layer

The buttons at the top of the attribute table window provide the following functionality:

Table 12.1: Available Tools

Icon	Label	Purpose	Default Shortcut
	Toggle editing mode	Enable editing functionalities	`Ctrl+E`
	Toggle multi edit mode	Update multiple fields of many features	
	Save Edits	Save current modifications	
	Reload the table		
	Add feature	Add new geometryless feature	
	Delete selected features	Remove selected features from the layer	
	Cut selected features to clipboard		`Ctrl+X`

continues on next page

Table 12.1 – continued from previous page

Icon	Label	Purpose	Default Shortcut
	Copy selected features to clipboard		`Ctrl+C`
	Paste features from clipboard	Insert new features from copied ones	`Ctrl+V`
	Select features using an Expression		
	Select All	Select all features in the layer	`Ctrl+A`
	Invert selection	Invert the current selection in the layer	`Ctrl+R`
	Deselect all	Deselect all features in the current layer	`Ctrl+Shift+A`
	Filter/Select features using form		`Ctrl+F`
	Move selected to top	Move selected rows to the top of the table	
	Pan map to the selected rows		`Ctrl+P`
	Zoom map to the selected rows		`Ctrl+J`
	New field	Add a new field to the data source	`Ctrl+W`
	Delete field	Remove a field from the data source	
	Organize columns	Show/hide fields from the attribute table	
	Open field calculator	Update field for many features in a row	`Ctrl+I`
	Conditional formatting	Enable table formatting	
	Dock attribute table	Allows to dock/undock the attribute table	
	Actions	Lists the actions related to the layer	

Note

Depending on the format of the data and the GDAL library built with your QGIS version, some tools may not be available.

Below these buttons is the Quick Field Calculation bar (enabled only in *edit mode*), which allows to quickly apply calculations to all or part of the features in the layer. This bar uses the same *expressions* as the  Field Calculator (see *Editing attribute values*).

Table view vs Form view

QGIS provides two view modes to easily manipulate data in the attribute table:

- The  Table view, displays values of multiple features in a tabular mode, each row representing a feature and each column a field. A right-click on the column header allows you to *configure the table display* while a right-click on a cell provides *interaction with the feature*.

  The attribute table supports `Shift+Mouse Wheel` scrolling in table view mode to switch between vertical and horizontal scrolling movements. This can also be achieved replacing the mouse with the trackpad on macOS.

- The  Form view shows *feature identifiers* in a first panel and displays only the attributes of the clicked identifier in the second one. There is a pull-down menu at the top of the first panel where the "identifier" can be specified using an attribute (*Column preview*) or an *Expression*. The pull-down also includes the last 10 expressions for re-use. Form view uses the layer fields configuration (see *Attributes Form Properties*).

You can browse through the feature identifiers with the arrows on the bottom of the first panel. The features attributes update in the second panel as you go. It's also possible to identify or move to the active feature in the map canvas with pushing down any of the button at the bottom:

- Highlight current feature if visible in the map canvas
- Automatically pan to current feature
- Zoom to current feature

You can switch from one mode to the other by clicking the corresponding icon at the bottom right of the dialog.

You can also specify the *Default view* mode at the opening of the attribute table in *Settings* ► *Options* ► *Data Sources* menu. It can be 'Remember last view', 'Table view' or 'Form view'.

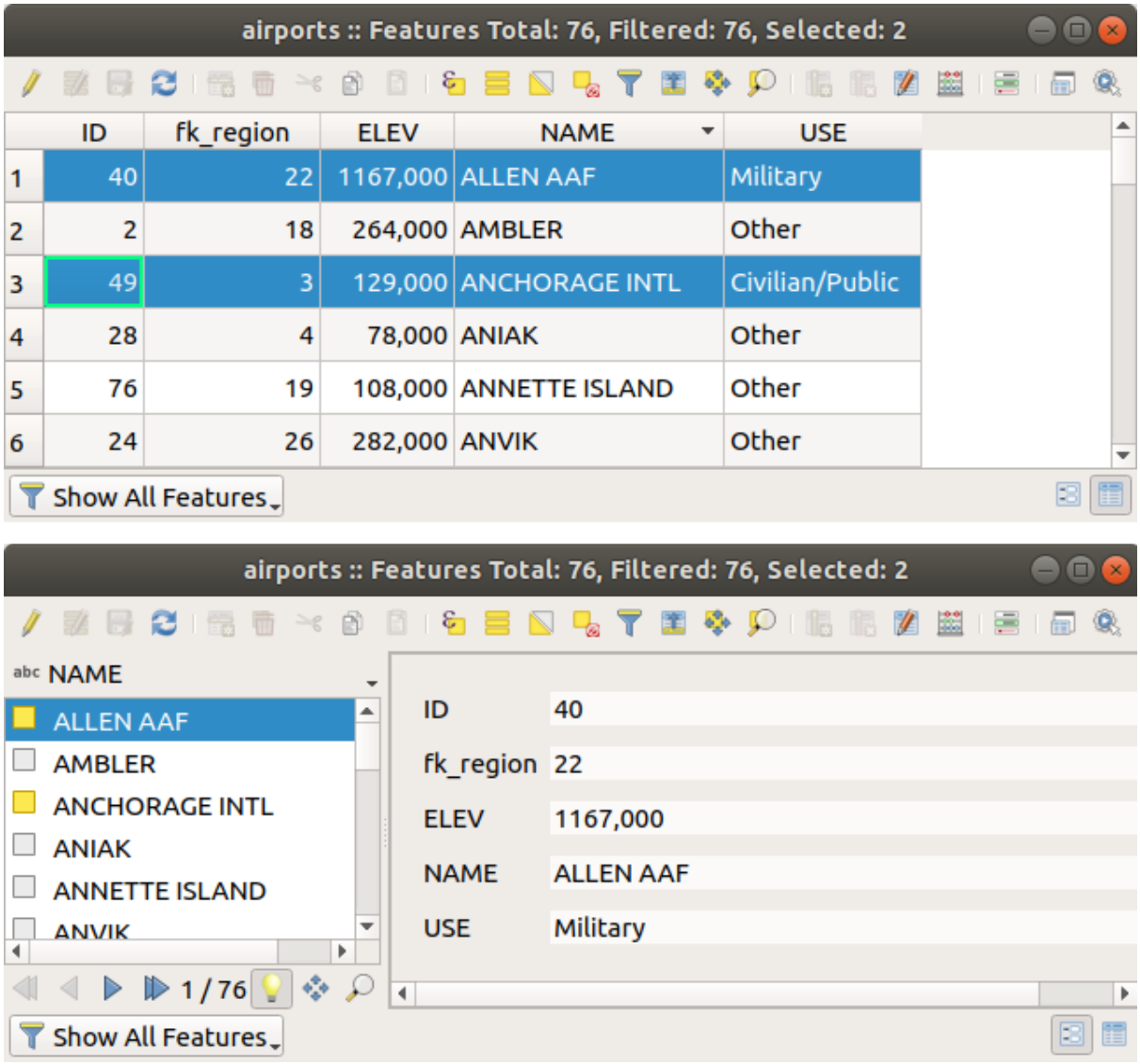


Fig. 12.71: Attribute table in table view (top) vs form view (bottom)

Configuring the columns

Right-click in a column header when in table view to have access to tools that help you control:

- the *column(s) size*
- the *column(s) visibility and order*
- the *sort order of the data*

Resizing columns widths

Columns width can be set through a right-click on the column header and select either:

- *Set width…* to enter the desired value. By default, the current value is displayed in the widget
- *Set all column widths…* to the same value
- *Autosize* to resize at the best fit the column.
- *Autosize all columns*

A column size can also be changed by dragging the boundary on the right of its heading. The new size of the column is maintained for the layer, and restored at the next opening of the attribute table.

In the *Data Source Settings*, you can choose to  *Autosize all columns by default when opening attribute table*, which will make "Autosize All Columns" the default view every time attribute tables are opened in QGIS.

Hiding and organizing columns and enabling actions

By right-clicking in a column header, you can choose to *Hide column* from the attribute table (in "table view" mode). For more advanced controls, press the  Organize columns… button from the dialog toolbar or choose *Organize columns…* in a column header contextual menu.

In the new dialog, you can:

- check/uncheck columns you want to show or hide: a hidden column will disappear from every instance of the attribute table dialog until it is actively restored. It is also possible to:
  - choose *Show All* to display all the fields (columns) and actions in the table
  - choose *Hide All* to hide all the fields (columns) and actions in the table
  - use the *Toggle selection* to invert visibility of the current selection of columns. You can use *keyboard combination* for selecting multiple columns.
- drag-and-drop items to reorder the columns in the attribute table. Note that this change is for the table rendering and does not alter the fields order in the layer datasource
- add a new virtual *Actions* column that displays in each row a drop-down box or a button list of enabled actions. See *Actions Properties* for more information about actions.

Sorting rows

The rows can be sorted by any column, by clicking on the column header. A small arrow indicates the sort order (downward pointing means descending values from the top row down, upward pointing means ascending values from the top row down). You can also choose to sort the rows with the *Sort...* option of the column header context menu and write an expression. E.g. to sort the rows using multiple columns you can write `concat(col0, col1)`.

In form view, features identifier can be sorted using the  *Sort by preview expression* option.

Note that sorting the rows only affects the table rendering and does not alter the features order in the layer datasource.

> **Tip**
>
> **Sorting based on columns of different types**
>
> Trying to sort an attribute table based on columns of string and numeric types may lead to unexpected result because of the `concat("USE", "ID")` expression returning string values (ie, `'Borough105' < 'Borough6'`). You can workaround this by using eg `concat("USE", lpad("ID", 3, 0))` which returns `'Borough105' > 'Borough006'`.

Formatting of table cells using conditions

Conditional formatting settings can be used to highlight in the attribute table features you may want to put a particular focus on, using custom conditions on feature's:

- geometry (e.g., identifying multi-parts features, small area ones or in a defined map extent…);
- or field value (e.g., comparing values to a threshold, identifying empty cells, duplicates, …).

You can enable the conditional formatting panel clicking on  Conditional formatting button at the top right of the attributes window in table view (not triggered in form view).

The new panel allows user to add new rules to format rendering of  *Field* or  *Full row*. Adding new rule opens a form to define:

- the name of the rule;
- a condition using any of the *expression builder* functions;
- the formatting: it can be choosen from a list of predefined formats or created based on properties like:
  - background and text colors;
  - use of icon;
  - bold, italic, underline, or strikeout;
  - font.

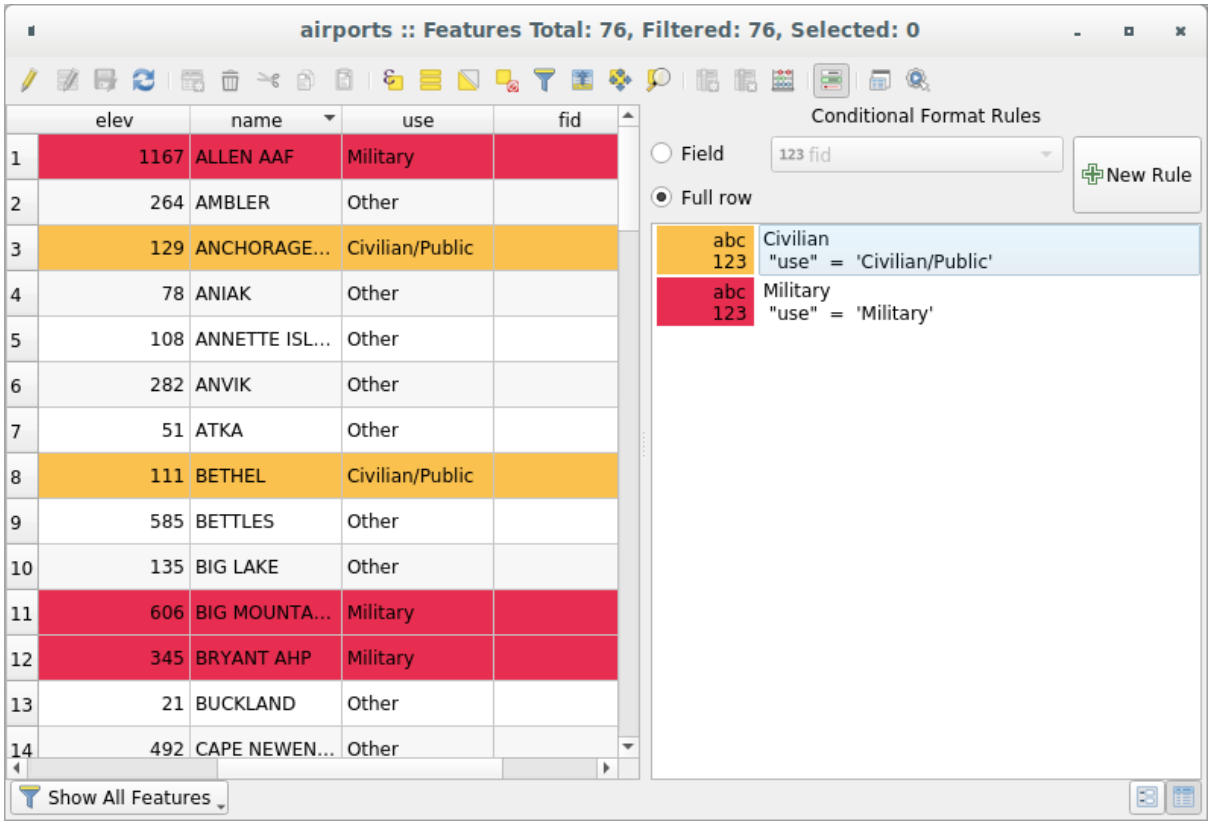


Fig. 12.72: Conditional Formatting of an attribute table

12.2.3 Interacting with features in an attribute table

Selecting features

In table view, each row in the attribute table displays the attributes of a unique feature in the layer. Selecting a row selects the feature and likewise, selecting a feature in the map canvas (in case of geometry enabled layer) selects the row in the attribute table. If the set of features selected in the map canvas (or attribute table) is changed, then the selection is also updated in the attribute table (or map canvas) accordingly.

Rows can be selected by clicking on the row number on the left side of the row. **Multiple rows** can be marked by holding the `Ctrl` key. A **continuous selection** can be made by holding the `Shift` key and clicking on several row headers on the left side of the rows. All rows between the current cursor position and the clicked row are selected. Moving the cursor position in the attribute table, by clicking a cell in the table, does not change the row selection. Changing the selection in the main canvas does not move the cursor position in the attribute table.

In form view of the attribute table, features are by default identified in the left panel by the value of their displayed field (see *Display Properties*). This identifier can be replaced using the drop-down list at the top of the panel, either by selecting an existing field or using a custom expression. You can also choose to sort the list of features from the drop-down menu.

Click a value in the left panel to display the feature's attributes in the right one. To select a feature, you need to click inside the square symbol at the left of the identifier. By default, the symbol turns into yellow. Like in the table view, you can perform multiple feature selection using the keyboard combinations previously exposed.

Beyond selecting features with the mouse, you can perform automatic selection based on feature's attribute using tools available in the attribute table toolbar, such as (see section *Automatic selection* and subsequent for more information and use case):

- *Select By Expression…*

-  *Select Features By Value…*
-  *Deselect All Features from the Layer*
-  *Select All Features*
-  *Invert Feature Selection*.

It is also possible to *select features using forms*.

Filtering features

Once you have selected features in the attribute table, you may want to display only these records in the table. This can be easily done using the *Show Selected Features* item from the drop-down list at the bottom left of the attribute table dialog. This list offers the following filters:

-  *Show All Features*
-  *Show Selected Features* - same as using *Open Attribute Table (Selected Features)* from the *Layer* menu or the *Attributes Toolbar* or pressing `Shift+F6`
-  *Show Features visible on map* - same as using *Open Attribute Table (Visible Features)* from the *Layer* menu or the *Attributes Toolbar* or pressing `Ctrl+F6`
-  *Show Features with Failing Constraints* - features will be filtered to only show the ones which have failing *constraints*. Depending on whether the unmet constraint is hard or soft, failing field values are displayed in respectively dark or light orange cells.
-  *Show Edited and New Features* - same as using *Open Attribute Table (Edited and New Features)* from the *Layer* menu or the *Attributes Toolbar*
- *Field Filter* - allows the user to filter based on value of a field: choose a column from a list, type or select a value and press `Enter` to filter. Then, only the features matching `num_field = value` or `string_field ilike '%value%'` expression are shown in the attribute table. You can check *Case sensitive* to be less permissive with strings.
-  *Advanced filter (Expression)* - Opens the expression builder dialog. Within it, you can create *complex expressions* to match table rows. For example, you can filter the table using more than one field. When applied, the filter expression will show up at the bottom of the form.
-  *Stored filter expressions* ►: a shortcut to *saved expressions* frequently used for filtering your attribute table.

It is also possible to *filter features using forms*.

> **Note**
>
> Filtering records out of the attribute table does not filter features out of the layer; they are simply momentaneously hidden from the table and can be accessed from the map canvas or by removing the filter. For filters that do hide features from the layer, use the *Query Builder*.

> **Tip**
>
> **Update datasource filtering with** `Show Features Visible on Map`

When for performance reasons, features shown in attribute table are spatially limited to the canvas extent at its opening (see *Data Source Options* for a how-to), selecting *Show Features Visible on Map* on a new canvas extent updates the spatial restriction.

Storing filter expressions

Expressions you use for attribute table filtering can be saved for further calls. When using *Field Filter* or *Advanced Filter (expression)* entries, the expression used is displayed in a text widget in the bottom of the attribute table dialog. Press the Save expression with text as name next to the box to save the expression in the project. Pressing the drop-down menu next to the button allows to save the expression with a custom name (*Save expression as...*). Once a saved expression is displayed, the button is triggered and its drop-down menu allows you to *Edit the expression* and name if any, or *Delete stored expression*.

Saved filter expressions are saved in the project and available through the *Stored filter expressions* menu of the attribute table. They are different from the *user expressions*, shared by all projects of the active user profile.

Filtering and selecting features using forms

Clicking the Filter/Select features using form or pressing `Ctrl+F` will make the attribute table dialog switch to form view and replace each widget with its search variant.

From this point onwards, this tool functionality is similar to the one described in *Select Features By Value*, where you can find descriptions of all operators and selecting modes.

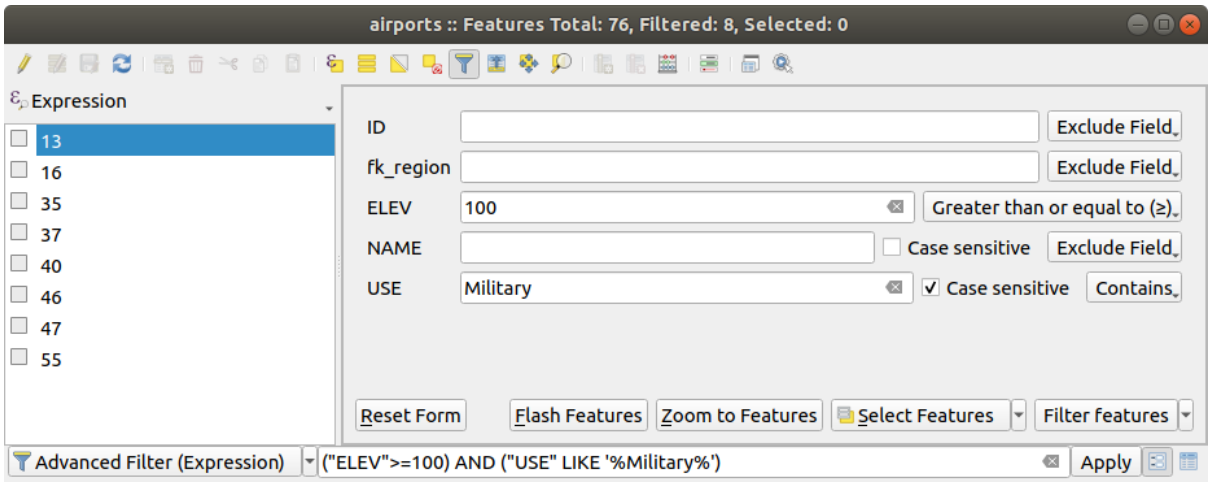


Fig. 12.73: Attribute table filtered by the filter form

When selecting / filtering features from the attribute table, there is a *Filter features* button that allows defining and refining filters. Its use triggers the *Advanced filter (Expression)* option and displays the corresponding filter expression in an editable text widget at the bottom of the form.

If there are already filtered features, you can refine the filter using the drop-down list next to the *Filter features* button. The options are:

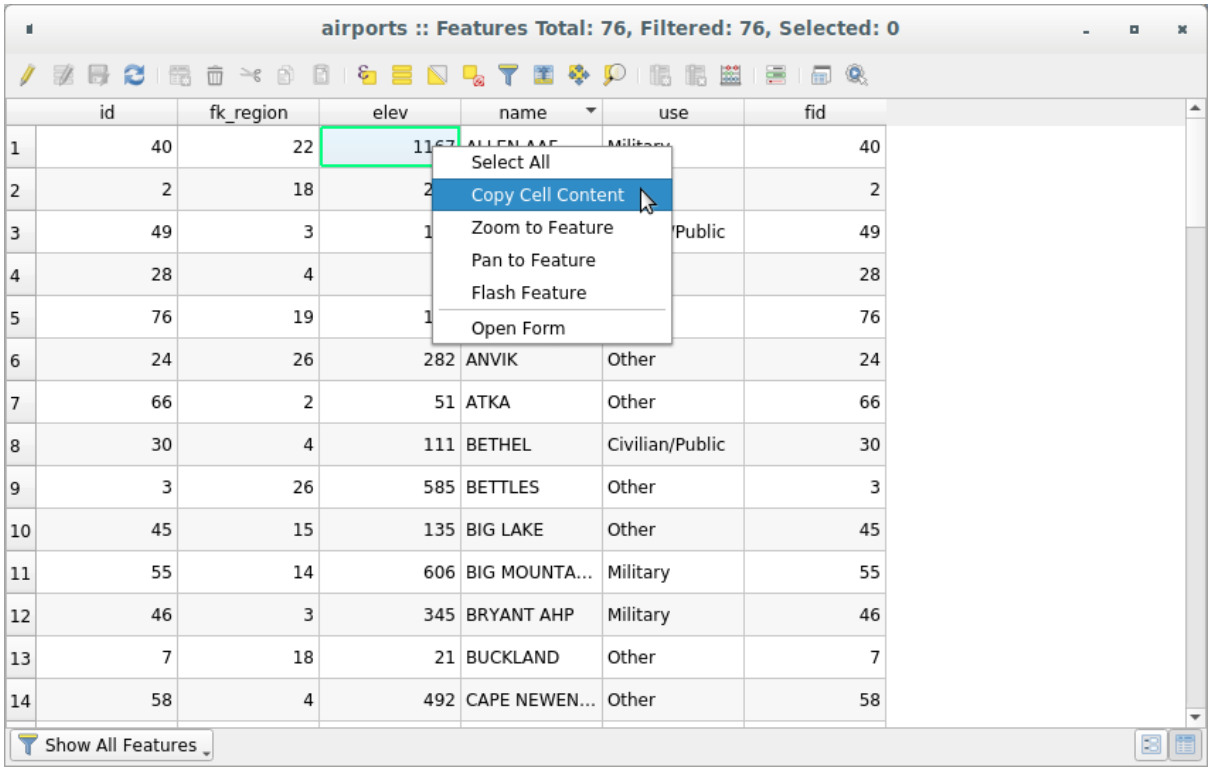
- *Filter within ("AND")*
- *Extend filter ("OR")*

To clear the filter, either select the *Show all features* option from the bottom left pull-down menu, or clear the expression and click *Apply* or press `Enter`.

More actions on features

Users have several possibilities to manipulate feature in an attribute table. Right-click in a cell and you can:

- *Select all* (`Ctrl+A`) the features;
- Copy the content of a cell in the clipboard with *Copy cell content*;
- *Zoom to feature* without having to select it beforehand;
- *Pan to feature* without having to select it beforehand;
- *Flash feature*, to highlight it in the map canvas;
- *Open form*: it toggles attribute table into form view with a focus on the clicked feature.
- Display a *list of actions*, previously enabled in the *Layer properties* ► *Actions* tab.



	id	fk_region	elev	name	use	fid
1	40	22	1167	ALLEN AAF	Military	40
2	2	18	2			2
3	49	3	1		Public	49
4	28	4				28
5	76	19	1			76
6	24	26	282	ANVIK	Other	24
7	66	2	51	ATKA	Other	66
8	30	4	111	BETHEL	Civilian/Public	30
9	3	26	585	BETTLES	Other	3
10	45	15	135	BIG LAKE	Other	45
11	55	14	606	BIG MOUNTA...	Military	55
12	46	3	345	BRYANT AHP	Military	46
13	7	18	21	BUCKLAND	Other	7
14	58	4	492	CAPE NEWEN...	Other	58

Fig. 12.74: Copy cell content button

If you want to use attribute data in external programs (such as Excel, LibreOffice, or a custom web application), select one or more row(s) and use the Copy selected rows to clipboard button or press `Ctrl+C`. Moreover, in *Settings* ► *Options* ► *Data Sources* menu you can define the format to paste to with the *Copy features as* option. More details at *Data Sources settings*.

12.2.4 Editing attribute values

In order to modify data in an attribute table, you should first toggle the layer into edit. Press the  Toggle Editing button. Depending on the layer geometry type and the clipboard state, a few more tools are enabled in the attribute table top toolbar.

Editing attribute values can then be done by:

- typing the new value directly in the cell, whether the attribute table is in table or form view. Changes are hence done cell by cell, feature by feature;
- using the *field calculator*: update in a row a field that may already exist or to be created but for multiple features. It can be used to create virtual fields;
- using the quick field *calculation bar*: same as above but for only existing field;
- or using the *multi edit* mode: update in a row multiple fields for multiple features.

Putting the layer into edit mode will also allow you to  Paste features from clipboard (`Ctrl+V`)  Cut selected rows to clipboard (`Ctrl+X`) or  Delete selected features. More details at *Editing*.

Using the Field Calculator

The  Field Calculator button in the attribute table allows you to perform calculations on the basis of existing attribute values or defined functions, for instance, to calculate length or area of geometry features. The results can be used to update an existing field, or written to a new field (that can be a *virtual* one).

The field calculator is available on any layer that supports edit. When you click on the field calculator icon the dialog opens (see Fig. 12.75). If the layer is not in edit mode, a warning is displayed and using the field calculator will cause the layer to be put in edit mode before the calculation is made.

Based on the *Expression Builder* dialog, the field calculator dialog offers a complete interface to define an expression and apply it to an existing or a newly created field. To use the field calculator dialog, you must select whether you want to:

1. apply calculation on the whole layer or on selected features only
2. create a new field for the calculation or update an existing one.

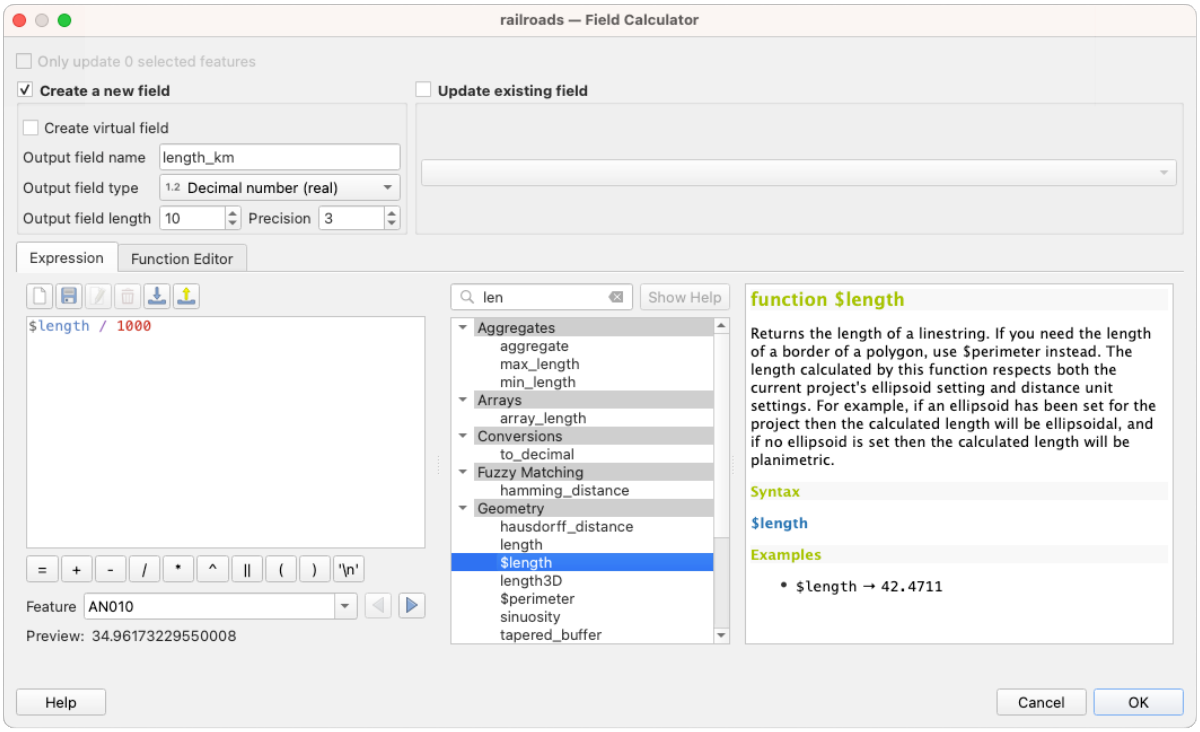


Fig. 12.75: Field Calculator

If you choose to add a new field, you need to enter a field name, a field type (integer, real, date or string) and if needed, the total field length and the field precision. For example, if you choose a field length of 10 and a field precision of 3, it means you have 7 digits before the dot, and 3 digits for the decimal part.

A short example illustrates how field calculator works when using the *Expression* tab. We want to calculate the length in km of the `railroads` layer from the QGIS sample dataset:

1. Load the shapefile `railroads.shp` in QGIS and press Open Attribute Table.
2. Click on Toggle editing mode and open the Field Calculator dialog.
3. Select the *Create a new field* checkbox to save the calculations into a new field.
4. Set *Output field name* to `length_km`
5. Select `Decimal number (real)` as *Output field type*
6. Set the *Output field length* to `10` and the *Precision* to `3`
7. Double click on `$length` in the *Geometry* group to add the length of the geometry into the Field calculator expression box (you will begin to see a preview of the output, up to 60 characters, below the expression box updating in real-time as the expression is assembled).
8. Complete the expression by typing `/ 1000` in the Field calculator expression box and click *OK*.
9. You can now find a new *length_km* field in the attribute table.

Creating a Virtual Field

A virtual field is a field based on an expression calculated on the fly, meaning that its value is automatically updated as soon as an underlying parameter changes. The expression applies to all the features in the layer and is set once; you no longer need to recalculate the field each time underlying values change. For example, you may want to use a virtual field if you need area to be evaluated as you digitize features or to automatically calculate a duration between dates that may change (e.g., using `now()` function).

Creating a virtual field is done through the *Field calculator* dialog and follows the *same procedure* as for regular fields. Simply remember to check the *Create virtual field* option and use a field type compatible with the data your expression would generate.

Editing a virtual field is done through the *Fields* tab of the layer properties dialog (see *Fields Properties*). The expression defining the field is exposed in the *Comment* column, and pressing the ε button next to it opens an expression editor window for update.

Note

Use of Virtual Fields

- A field can be set virtual only at its creation.
- Virtual fields are not permanent in the layer attributes, meaning that they're only saved and available in the project file they've been created.

Using the Quick Field Calculation Bar

While Field calculator is always available, the quick field calculation bar on top of the attribute table is only visible if the layer is in edit mode. Thanks to the expression engine, it offers a quicker access to edit an already existing field:

1. Select the field to update in the drop-down list.
2. Fill the textbox with a value, an expression you directly write or build using the ε expression button.
3. Click on *Update All*, *Update Selected* or *Update Filtered* button according to your need.

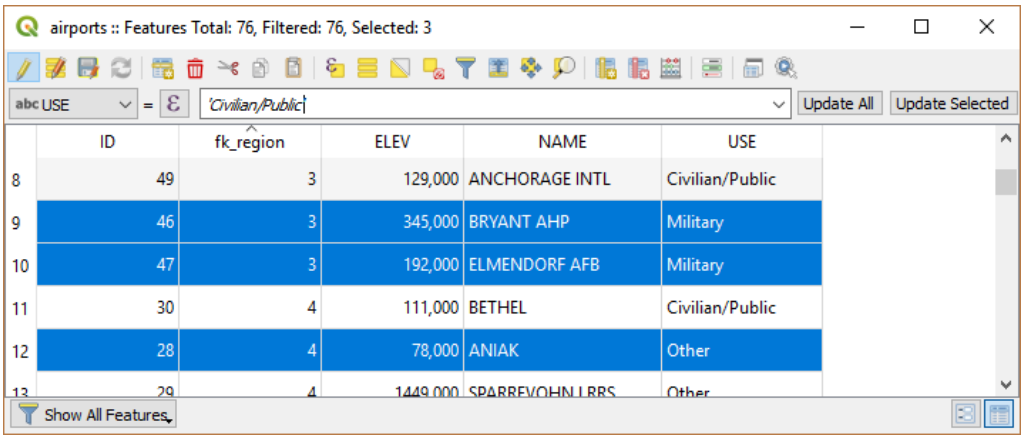


Fig. 12.76: Quick Field Calculation Bar

Editing multiple fields

Unlike the previous tools, multi edit mode allows multiple attributes of different features to be edited simultaneously. When the layer is toggled to edit, multi edit capabilities are accessible:

- using the  Toggle multi edit mode button from the toolbar inside the attribute table dialog;
- or selecting *Edit* ►  *Modify attributes of selected features* menu.

> **Note**
>
> Unlike the tool from the attribute table, hitting the *Edit* ► *Modify Attributes of Selected Features* option provides you with a modal dialog to fill attributes changes. Hence, features selection is required before execution.

In order to edit multiple fields in a row:

1. Select the features you want to edit.
2. From the attribute table toolbar, click the  button. This will toggle the dialog to its form view. Feature selection could also be made at this step.
3. At the right side of the attribute table, fields (and values) of selected features are shown. New widgets appear next to each field allowing for display of the current multi edit state:
   -  The field contains different values for selected features. It's shown empty and each feature will keep its original value. You can reset the value of the field from the drop-down list of the widget.
   -  All selected features have the same value for this field and the value displayed in the form will be kept.
   -  The field has been edited and the entered value will be applied to all the selected features. A message appears at the top of the dialog, inviting you to either apply or reset your modification.

   Clicking any of these widgets allows you to either set the current value for the field or reset to original value, meaning that you can roll back changes on a field-by-field basis.

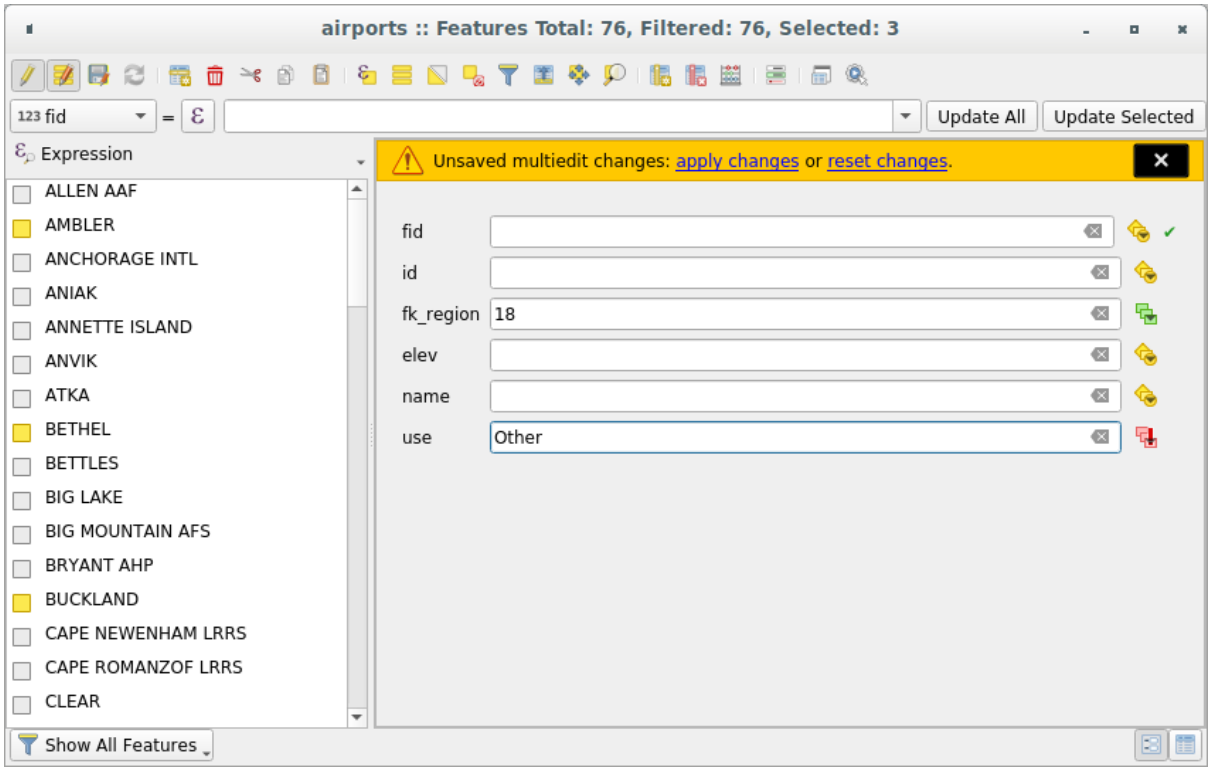


Fig. 12.77: Editing fields of multiple features

4. Make the changes to the fields you want.
5. Click on **Apply changes** in the upper message text or any other feature in the left panel.

Changes will apply to **all selected features**. If no feature is selected, the whole table is updated with your changes. Modifications are made as a single edit command. So pressing Undo will rollback the attribute changes for all selected features at once.

> **Note**
>
> Multi edit mode is only available for auto generated and drag and drop forms (see *Customizing a form for your data*); it is not supported by custom ui forms.

12.2.5 Exploring features attributes through the Identify Tool

The *Identify features* tool can be used to display all attributes of a feature in the map canvas. It is a quick way to view and verify all data without having to search for it in the attribute table.

To use the *Identify features* tool for vector layers, follow these steps:

1. Select the vector layer in the Layers panel.
2. Click on the *Identify features* tool in the toolbar or press `Ctrl+Shift+I`.
3. Click on a feature in the map view.

The *Identify results* panel will display different features information depending on the layer type. There are two columns in the panel, on the left side you can see *Feature* and on the right side *Value*. Under the *Feature* column, panel will display following information:

- **Derived** section - those are the information calculated or derived from other information in the layer. For example, the area of a polygon or the length of a line. General information that can be found in this section:
  - Depending on the geometry type, cartesian measurements of length, perimeter, or area in the layer's CRS units. For 3D line vectors, the cartesian line length is available.
  - Depending on the geometry type and if an ellipsoid is set in the *Project Properties* dialog (*General* ► *Measurements*), ellipsoidal values of length, perimeter, or area using the specified units.
  - The count of geometry parts in the feature and the number of the part clicked.
  - The count of vertices in the feature.

  Coordinate information that can be found in this section:
  - X and Y coordinate values of the clicked point.
  - The number of the closest vertex to the clicked point.
  - X and Y coordinate values of the closest vertex.
  - If you click on a curved segment, the radius of that section is also displayed.
- **Data attributes**: This is the list of attribute fields and values for the feature that has been clicked.
- information about the related child feature if you defined a *relation*:
  - the name of the relation
  - the entry in reference field, e.g. the name of the related child feature
  - **Actions**: lists actions defined in the layer's properties dialog (see *Actions Properties*) and the default action is `View feature form`.
  - **Data attributes**: This is the list of attributes fields and values of the related child feature.

12.2.6 Storing and fetching an external resource

A field may target a resource stored on an external storage system. Attribute forms can be configured so they act as a client to an external storage system in order to store and fetch those resources, on users demand, directly from the forms.

Configuring an external storage

In order to setup an external storage, you have to first configure it from the vector *attribute form properties* and select the *Attachment* widget.



Fig. 12.78: Editing a WebDAV external storage for a given field

From the *Attachment* widget, you have to first select the *Storage type*:

- *Select Existing File*: The target URL already exists. When you select a resource, no store operation is achieved, the attribute is simply updated with the URL.
- *Simple Copy*: Stores a copy of the resource on a file disk destination (which could be a local or network shared file system) and the attribute is updated with the path to the copy.
- *WebDAV Storage*: The resource is pushed to a HTTP server supporting the WebDAV protocol and the attribute is updated with its URL. Nextcloud, Pydio or other file hosting software support this protocol.
- *AWS S3*: The resource is pushed to a server supporting AWS Simple Storage Service protocol and the attribute is updated with its URL. Amazon Web Service and MinIO hosting software support this protocol.

Then, you have to set up the *Store URL* parameter, which provides the URL to be used when a new resource needs to be stored. It's possible to set up an expression using the *data defined override widget* in order to have specific values according to feature attributes.

The variable **@selected_file_path** could be used in that expression and represent the absolute file path of the user selected file (using the file selector or drag'n drop).

> **Note**
>
> Using the **WebDAV** or **AWS S3** external storage, if the URL ends with a "/", it is considered as a folder and the selected file name will be appended to get the final URL.

If the external storage system needs to, it's possible to configure an *authentication*.

> **Note**
>
> To use the **AWS S3** external storage, you must use an **AWS S3** authentication type.

Using an external storage

Once configured, you can select a local file using the button … when editing a feature's attribute. Depending on the configured *storage type*, the file will be stored on the external storage system (except if *Select existing file* has been selected) and the field will be updated with the new resource URL.

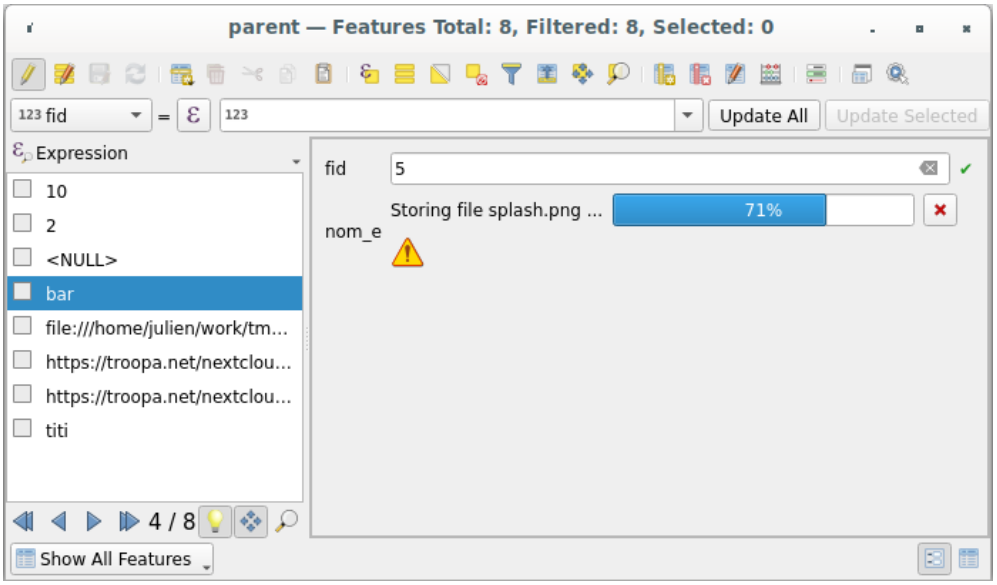


Fig. 12.79: Storing a file to a WebDAV external storage

> **Note**
>
> User can also achieve the same result if he drags and drops a file on the whole attachment widget.

Use the Cancel button to abort the storing process. It's possible to configure a viewer using the *Integrated document viewer* so the resource will be automatically fetched from the external storage system and displayed directly below the URL. The above icon indicates that the resource cannot be fetched from the external storage system. In that case, more details might appear in the *Log Messages Panel*.

12.3 Editing

QGIS has various capabilities for editing OGR, SpatiaLite, PostgreSQL, MS SQL Server and Oracle Spatial vector layers and tables. They can be of 2D or 3D geometry type.

> **Note**
>
> The procedure for editing GRASS layers is different - see section *Digitizing and editing a GRASS vector layer* for details.

> **Attention**
>
> **Concurrent Edits**

QGIS does not track if somebody else is editing the same feature at the same time as you are. The last person to save the edits wins.

Tip

Validating Edits

Continuous validation can be activated on a layer basis in the *Layer Properties* ► *Digitizing* tab. More at *Digitizing Properties*.

12.3.1 Setting the snapping tolerance and search radius

Under the *Settings* ► *Options...* ► *Digitizing* menu, QGIS provides a number of parameters to configure default behaviour of editing tools. More information at *Digitizing settings*.

For optimal and accurate editing of vector layer geometries, we need to set an appropriate value of snapping tolerance and search radius for features vertices. The *Snapping* group provides related options, namely handling of the snapping tolerance and the search radius.

- Snapping tolerance: When you add a new vertex or move an existing one, the snapping tolerance is the distance QGIS uses to search for the closest vertex or segment you are trying to connect to. If you are not within the snapping tolerance, QGIS will leave the vertex where you release the mouse button, instead of snapping it to an existing vertex or segment.

  The tolerance setting affects all tools that work with snapping and applies by default to new layers and projets. It can however be overridden at layer level (see *Snapping and Digitizing Options*).

- Search radius: *Search radius for vertex edits* is the distance QGIS uses to `search` for the vertex to select when you click on the map. If you are not within the search radius, QGIS will not find and select any vertex for editing.

Snap tolerance and search radius are set in `map units` or `pixels`. You may need to experiment to get them right. If you specify a too big tolerance, QGIS may snap to the wrong vertex, especially if you are dealing with a large number of vertices in close proximity. The smaller the search radius, the more difficult it will be to hit what you want to move.

12.3.2 Snapping and Digitizing Options

Global *snapping and digitizing settings* (snapping mode, tolerance value, and units…) can be overridden in the project from the *Project* ► *Snapping Options...* menu. In the *Snapping and Digitizing Options*, you can also configure some other properties (snapping layers, scale limit, topology…) The *Snapping Toolbar* gives access to most of these features.

By default, snapping is disabled in a project until you press the  Enable snapping button or press `S`. The snapping mode, tolerance value, and units can also be configured in this toolbar.

Snapping properties

There are three options to select the layer(s) to snap to:

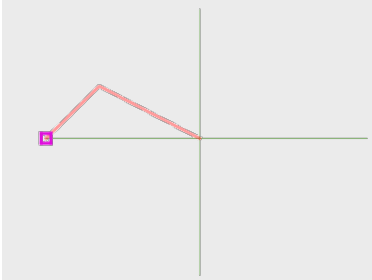
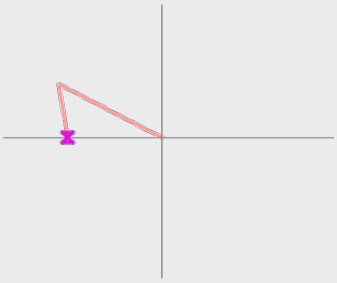
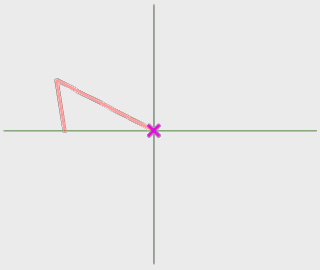
- *All layers*: quick setting for all visible layers in the project so that the pointer snaps to all vertices and/or segments. In most cases, it is sufficient to use this snapping mode, but beware when using it for projects with many vector layers, as it may affect performance.
- *Current layer*: only the active layer is used, a convenient way to ensure topological consistency within the layer being edited.
- *Advanced Configuration*: allows you to enable and adjust snapping mode, tolerance and units, overlaps and scales of snapping on a layer basis (see Fig. 12.80). If you need to edit a layer and snap its vertices to another, make sure that the target layer is checked and increase the snapping tolerance to a higher value. Snapping will not occur to a layer that is not checked in the snapping options dialog.

When moving or creating vertex, you can opt for the following snapping modes:

- *Vertex*
- *Segment*: snaps along a line or a polygon perimeter. If topological editing is enabled, then a new vertex is added at the snapping location.
- *Area*: guarantees that the snap point lies anywhere on a polygon's area, not necessarily on its boundary
- *Centroid*: snaps to the centroid of the geometry of a feature. In case of a multipart geometry, the target point may be distinct from the existing parts.
- *Middle of Segments* on line or polygon feature
- *Line Endpoints*: snaps to the first or last vertex of every part of a line or polygon feature.

QGIS will show different *snap* icons depending on the kind of *snap*:

Table 12.2: Snapping icons

		
Snapping to a vertex: box icon	Snapping to a segment: hourglass icon	Snapping to an intersection: cross icon

Note that it is possible to change the color of these icons in the *Digitizing* part of the global settings.

The tolerance values can be set either in the project's `map units` or in `pixels`. The advantage of choosing `pixels` is that it keeps the snapping constant at different map scales. 10 to 12 pixels is normally a good value, but it depends on the DPI of your screen. Using map units allows the tolerance to be related to real ground distances. For example, if you have a minimum distance between elements, this option can be useful to ensure that you don't add vertices too close to each other.

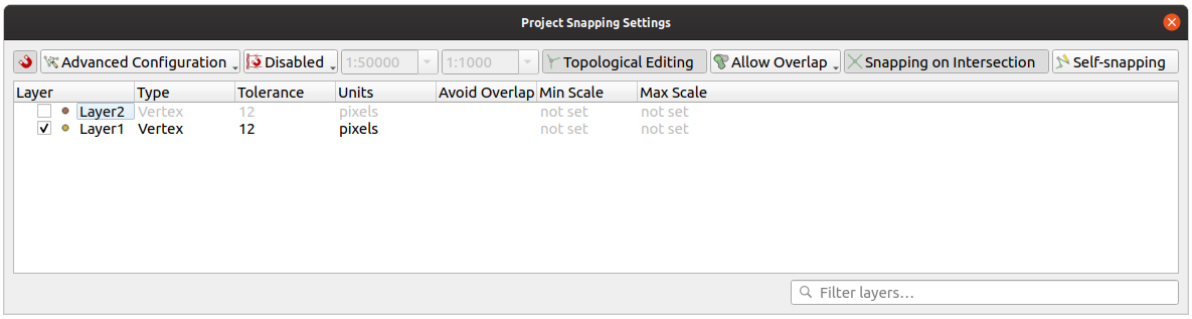


Fig. 12.80: Snapping options (Advanced Configuration mode)

Note

By default, only visible features (the features whose style is displayed, except for layers where the symbology is "No symbols") can be snapped. You can enable the snapping on invisible features by checking *Enable snapping on invisible features* under the *Settings* ► *Options* ► *Digitizing* tab.

Tip

Enable snapping by default

You can set snapping to be enabled by default on all new projects in the *Settings* ► *Options* ► *Digitizing* tab. You can also set the default snapping mode, tolerance value, and units, which will populate the *Snapping Options* dialog.

Enable snapping on intersections

Another available option is to use *snapping on intersection*, which allows you to snap to geometry intersections of snapping enabled layers, even if there are no vertices at the intersections.

Limit snapping to a scale range

In some cases snapping can become very slow. This is often caused by the amount of features in some layers that require a heavy index to compute and maintain. Some parameters exist to enable snapping only when the map view is inside a relevant scale range. This allows to only do the costly index computation related to snapping at a scale where drawing is relevant.

Scale limit to snapping is configured in *Project* ► *Snapping Options...*. Limiting snapping to scale is only available in *Advanced Configuration* mode.

To limit snapping to a scale range you have three modes available:

- *Disabled*: Snapping is enabled whatever the current map scale is. This is the default mode.
- *Global*: Snapping is limited and only enabled when the current scale of the map is between a global minimum and a global maximum value. When selecting this mode two widgets become available to configure the range of scales in which snapping is enabled.
- *Per layer*: The snapping scale range limit is defined for each layer. When selecting this mode two columns become available to configure the minimum and maximum scales for each layer.

Please note that the minimum and maximum scales follow the QGIS convention: minimum scale is the most "zoomed out" scale while maximum scale is the most "zoomed in". A minimum or maximum scale that is set to "0" or "not set" is considered not limiting.

Self-snapping

The  Self-snapping option allows you to snap to the geometry that is being edited. Combined with the *advanced digitizing panel*, this provides a handy way to digitize new edges relative to the previous edges or vertices. Self-snapping can cause invalid geometries, use with caution.

Snapping on custom grid

A snapping distance can also be customized on a layer basis in the *Digitizing* tab of the layer properties dialog. With setting the *Geometry precision* distance, you enable a dotted grid visible when the map canvas is at a coherent scale for display. Snapping can then be performed on the dots of the grid: an added or modified geometry will have all of its vertices snapped automatically to the closest node of the grid. More information at *Digitizing Properties*.

12.3.3 Topological editing

In addition to these snapping options, the *Snapping options...* dialog (*Project* ► *Snapping options*) and the *Snapping* toolbar allow you to enable / disable some other topological functionalities.

Enable topological editing

The  Topological editing button helps when editing and maintaining features with common boundaries. With this option enabled, QGIS 'detects' shared boundaries. When you move common vertices/segments, QGIS will also move them in the geometries of the neighboring features.

Topological editing works with features from different layers, as long as the layers are visible and in editing mode.

In layer with Z or M values, topological editing will interpolate the Z or M value of the vertex based on the value of the edge used for the connection.

Overlapping control

Overlapping prevents you from drawing new features that overlap existing ones in the selected layer, speeding up digitizing of adjacent polygons. It can be controlled by the overlap tool. Three modes are available:

1.  *Allow Overlap* (default)
2.  *Avoid Overlap on Active Layer*: prevents any overlap with other features from the layer being edited. Digitize the new geometries so that they overlap their neighbours and QGIS will cut the overlapping part(s) of the new geometries and snap them to the boundary of the existing features. The advantage is that you don't have to digitize the common vertices on boundary.
3.  *Follow Advanced Configuration*: allows the overlapping setting to be set on a layer basis in the *Advanced configuration* view mode.

> **Note**
>
> If the new geometry is totally covered by existing ones, it gets cleared, and QGIS will show an error message.

Warning

Use cautiously the *Avoid overlap* option

Since this option will cut new overlapping geometries of any polygon layer, you can get unexpected geometries if you forget to uncheck it when no longer needed.

Automatic Tracing

Usually, when using capturing map tools (add feature, add part, add ring, reshape and split), you need to click each vertex of the feature. With the automatic tracing mode, you can speed up the digitization process as you no longer need to manually place all the vertices during digitization:

1. Enable the  Tracing tool (in the *Snapping* toolbar) by pushing the icon or pressing `T` key.
2. *Snap to* a vertex or segment of a feature you want to trace along.
3. Move the mouse over another vertex or segment you'd like to snap and, instead of the usual straight line, the digitizing rubber band represents a path from the last point you snapped to the current position. The tool also works with curved geometries.

   QGIS actually uses the underlying features topology to build the shortest path between the two points. Tracing requires snapping to be activated in traceable layers to build the path. You should also snap to an existing vertex or segment while digitizing and ensure that the two nodes are topologically connectable through existing features edges, otherwise QGIS is unable to connect them and thus traces a single straight line.
4. Click and QGIS places the intermediate vertices following the displayed path.

Unfold the  Enable Tracing icon and set the *Offset* option to digitize a path parallel to the features instead of tracing along them. A positive value shifts the new drawing to the left side of the tracing direction and a negative value does the opposite.

Note

Adjust map scale or snapping settings for an optimal tracing

If there are too many features in map display, tracing is disabled to avoid potentially long tracing structure preparation and large memory overhead. After zooming in or disabling some layers the tracing is enabled again.

Note

Does not add topological points

This tool does not add points to existing polygon geometries even if *Topological editing* is enabled. If geometry precision is activated on the edited layer, the resulting geometry might not exactly follow an existing geometry.

Tip

Quickly enable or disable automatic tracing by pressing the `T` key

By pressing the `T` key, tracing can be enabled/disabled anytime (even while digitizing a feature), so it is possible to digitize parts of the feature with tracing enabled and other parts with tracing disabled. Tools behave as usual when tracing is disabled.

> **Tip**
>
> **Convert tracing to curved geometries**
>
> By using *Settings* ► *Options* ► *Digitizing* ► *Tracing* you can create curved geometries while digitizing. See *digitizing options*.

12.3.4 Digitizing an existing layer

By default, QGIS loads layers read-only. This is a safeguard to avoid accidentally editing a layer if there is a slip of the mouse. However, you can choose to edit any layer as long as the data provider supports it (see *Exploring Data Formats and Fields*), and the underlying data source is writable (i.e., its files are not read-only).

> **Tip**
>
> **Restrict edit permission on layers within a project**
>
> From the *Project* ► *Properties...* ► *Data Sources* ► *Layers Capabilities* table, you can choose to set any layer read-only regardless the provider permission. This can be a handy way, in a multi-users environment to avoid unauthorized users to mistakenly edit layers (e.g., Shapefile), hence potentially corrupt data. Note that this setting only applies inside the current project.

In general, tools for editing vector layers are divided into a digitizing and an advanced digitizing toolbar, described in section *Advanced digitizing*. You can select and unselect both under *View* ► *Toolbars* ►.

Using the basic digitizing tools, you can perform the following functions:

Table 12.3: Vector layer basic editing toolbar

Tool	Purpose	Tool	Purpose
	Access to save, rollback or cancel changes in all or selected layers simultaneously		Turn on or off edit status of selected layer(s) based on the active layer status
	Save edits to the active layer		
	Digitize using straight segments		Digitize using curve lines
	Enable freehand digitizing		Digitize polygon of regular shape
	Add new record		Add Feature: Capture Point
	Add Feature: Capture Line		Add Feature: Capture Polygon
	Vertex Tool (All Layers)		Vertex Tool (Current Layer)
Show Vertex Editor	Set whether the vertex editor panel should auto-open		Modify the attributes of all selected features simultaneously
	Delete Selected features from the active layer		Cut Features from the active layer
	Copy selected Features from the active layer		Paste Features into the active layer
	Undo changes in the active layer		Redo changes in active layer

Note that while using any of the digitizing tools, you can still *zoom or pan* in the map canvas without losing the focus on the tool.

All editing sessions start by choosing the Toggle editing option found in the context menu of a given layer, from the attribute table dialog, the digitizing toolbar or the *Layer* menu.

Once the layer is in edit mode, additional tool buttons on the editing toolbar will become available and markers will appear at the vertices of all features unless *Show markers only for selected features* option under *Settings* ► *Options…* ► *Digitizing* menu is checked.

> **Tip**
>
> **Save Regularly**
>
> Remember to  Save Layer Edits regularly. This will also check that your data source can accept all the changes.

Geometry editing techniques

When a geometry drawing tool (mainly the ones that add, split, reshape features) is enabled for a line or polygon based layer, you can select the technique for adding new vertices:

- The  Digitize with Segment: draws straight segment whose start and end points are defined by left clicks.
- The  Digitize with Curve: draws curve line based on three consecutive nodes defined by left clicks (start, point along the arc, end). If the geometry type does not support curves, then consecutive smaller segments are used to approximate the curvature.
- The  Stream Digitizing: draws lines in freehand mode, i.e. nodes are added following cursor movement in the map canvas and a *Streaming Tolerance*. The streaming tolerance defines the spacing between consecutive vertices. Currently, the only supported unit is pixels (`px`). Only the starting left click and the ending right click are necessary in this mode.
- The  Digitize Shape: triggers tools on the *Shape Digitizing Toolbar* to draw a polygon of a regular shape.

The selected technique remains while switching among the digitizing tools. You can combine any of the first three methods while drawing the same geometry.

Adding Features

Depending on the layer type, you can use the  Add Record,  Add Point Feature,  Add Line Feature or  Add Polygon Feature icons on the toolbar to add new features into the current layer.

To add a geometryless feature, click on the  Add Record button and you can enter attributes in the feature form that opens.

To create features with the spatially enabled tools, you first digitize the geometry then enter its attributes. To digitize the geometry:

1. (Optional as it is the default) Select the  Digitize With Segment geometry drawing method
2. Left-click on the map area to create the first point of your new feature. For point features, this should be enough and trigger, if required, the feature form to fill in their attributes.
3. For line or polygon geometries, keep on left-clicking for each additional point you wish to capture. You can rely on the *snapping to features* options, the *snap-to-grid* or the *advanced digitizing* panel to accurately position each vertex.

   Along with drawing straight segments between nodes you click one by one, lines and polygons can be:

   - *traced automatically*, accelerating the digitization. This will create consecutive straight lines between the vertices you place, following existing features.
   - free-hand digitized, pressing `R` or activating  Stream Digitizing.

- drawn as curve, pressing `Ctrl+Shift+G` or activating Digitize with Curve.

> **Note**
>
> While digitizing line or polygon geometries, you can switch back and forth between the geometry drawing methods, allowing you to create features mixing straight segments, free-hand ones and curved parts.

4. Press `Delete` or `Backspace` key to revert the last node(s) you may wrongly add.
5. When you have finished adding points, right-click anywhere on the map area to confirm you have finished entering the geometry of that feature.

> **Tip**
>
> **Customize the digitizing rubber band**
>
> While capturing polygon, the by-default red rubber band can hide underlying features or places you'd like to capture a point. This can be fixed by setting a lower opacity (or alpha channel) to the rubber band's *Fill Color* in *Settings* ► *Options* ► *Digitizing* menu. You can also avoid the use of the rubber band by checking *Don't update rubber band during node editing*.

6. For line feature pressing `Shift` + right-click will close the line automatically.
7. The attribute window will appear, allowing you to enter the information for the new feature. Fig. 12.81 shows setting attributes for a fictitious new river. However, in the *Digitizing* menu under the *Settings* ► *Options* menu, you can also:
   - *Suppress attributes pop-up windows after each created feature* to avoid the form opening;
   - or *Reuse last entered attribute values* to have fields automatically filled at the opening of the form and just have to type changing values.

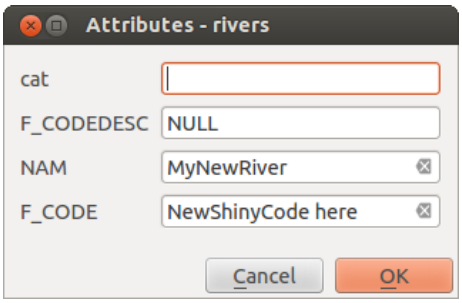


Fig. 12.81: Enter Attribute Values Dialog after digitizing a new vector feature

Vertex tool

QGIS provides two tools to interact with vector features vertices:

- Vertex Tool (Current Layer): only overlaid features in the active layer (in the *Layers* panel) are affected
- Vertex Tool (All Layers): any overlaid features in all editable layers are affected. This allows you to edit features without switching the active layer or edit multiple layers at once (e.g., country and their regions boundaries)

For any editable vector layer, the vertex tools provide manipulation capabilities of feature vertices similar to CAD programs. It is possible to select multiple vertices at once and to move, add or delete them altogether. The vertex tools

also support the topological editing feature. They are selection persistent, so when some operation is done, selection stays active for this feature and tool.

> **Tip**
>
> **Drawing a series of new vertices**
>
> The vertex tool does not support *automatic tracing* and is optimized for editing individual vertices and moving or deleting an arbitrary selection of multiple vertices. Try using the Reshape Features tool when you need to replace or insert a series of new vertices, especially if you want to trace other features.

It is important to set the property *Settings* ► *Options* ► *Digitizing* ► *Search Radius:* 1,00 to a number greater than zero. Otherwise, QGIS will not be able to tell which vertex is being edited and will display a warning.

> **Tip**
>
> **Vertex Markers**
>
> QGIS supports different kinds of vertex markers: 'Semi-transparent circle', 'Cross' and 'None'. To change the marker style, choose *Options* from the *Settings* menu, click on the *Digitizing* tab and select the appropriate entry.

Basic operations

Given a layer in edit mode, start by activating the vertex tool. Red circles will appear when hovering vertices.

- **Selecting vertices**: You can select vertices by:
  - Clicking on them one at a time holding `Shift` key pressed
  - Click-and-dragging a rectangle surrounding the target vertices
  - Drawing a polygon surrounding the target vertices: Hold `Alt` and click using the vertex tool to start digitizing a polygon. Each subsequent click adds a new vertex to the rubberband polygon. `Backspace` or `Delete` removes last added rubberband vertex. `Esc` cancels the polygon selection mode, as also does backspacing/deleting all of the rubberband's vertices. Right click finalizes the polygon digitizing and selects all vertices within the rubberband polygon.

  When a vertex is selected, its color changes to blue. To add more vertices to the current selection, hold down the `Shift` key while proceeding as above. To remove vertices from the selection, hold down `Ctrl`.

  > **Tip**
  >
  > **Feature selection bounds vertex tool**
  >
  > Vertices can be selected accross different features (or layers). If you are looking for vertices of a specific feature in a crowded place, first select that feature. Then draw the rectangle or polygon selector with the vertex tool around the vertices: only the selected feature's vertices are selected.
  >
  > This is also the case if you display the feature in the *vertex editor* panel.

- **Batch vertex selection mode**: The batch selection mode can be activated by pressing `Shift+R`. Select a first node with one single click, and then hover **without clicking** another vertex. This will dynamically select all the nodes in between using the shortest path (for polygons).

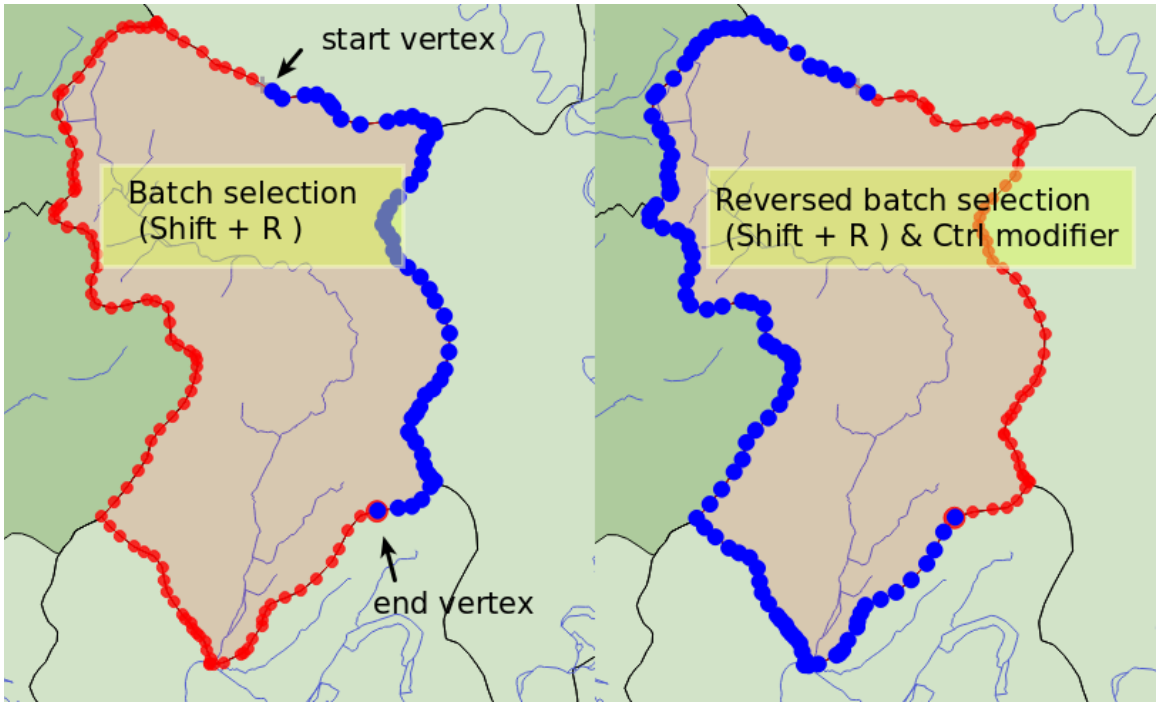


Fig. 12.82: Batch vertex selection using `Shift+R`

Press `Ctrl` will invert the selection, selecting the longest path along the feature boundary. Ending your node selection with a second click, or pressing `Esc` will escape the batch mode.

- **Adding vertices**: To add a vertex to a line or polygon geometry, hold `Shift` and double-click the place on the segment.

  When hovering a segment, a virtual new node appears on the center. Click on it, move the cursor to a new location and click again to add a new vertex. For lines, a virtual node is also proposed at both extremities: click on it, do subsequent clicks and finish with a right-click; this allows to easily extend an existing line.

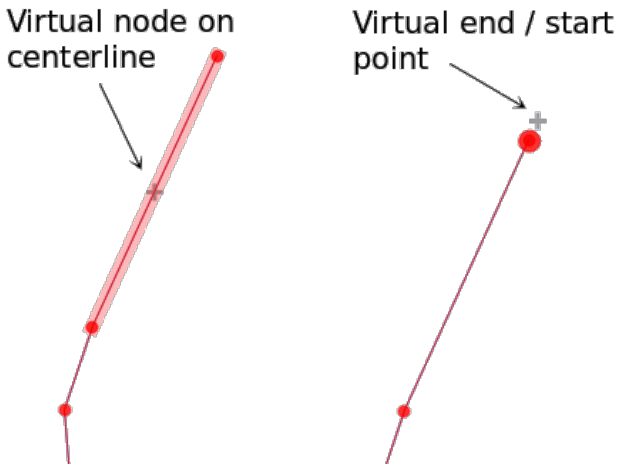


Fig. 12.83: Virtual nodes for adding vertices

- **Deleting vertices**: Select the vertices and click the `Delete` key. Deleting all the vertices of a feature generates, if compatible with the datasource, a geometryless feature. Note that this doesn't delete the complete feature, just the geometry part. To delete a complete feature use the Delete Selected tool.
- **Moving vertices**: Select all the vertices you want to move, click on a selected vertex or edge, and click on

the desired new location. You can use the *snapping to feature capabilities* and the *Advanced Digitizing Panel* constraints for distance, angles, exact X and Y location before the second click. All the selected vertices will be translated.

However, if the *snap-to-grid* option is enabled, selected vertices are snapped to the closest grid intersection to their translated position. Unselected vertices are also moved to their closest grid intersection. There is no simple translation.

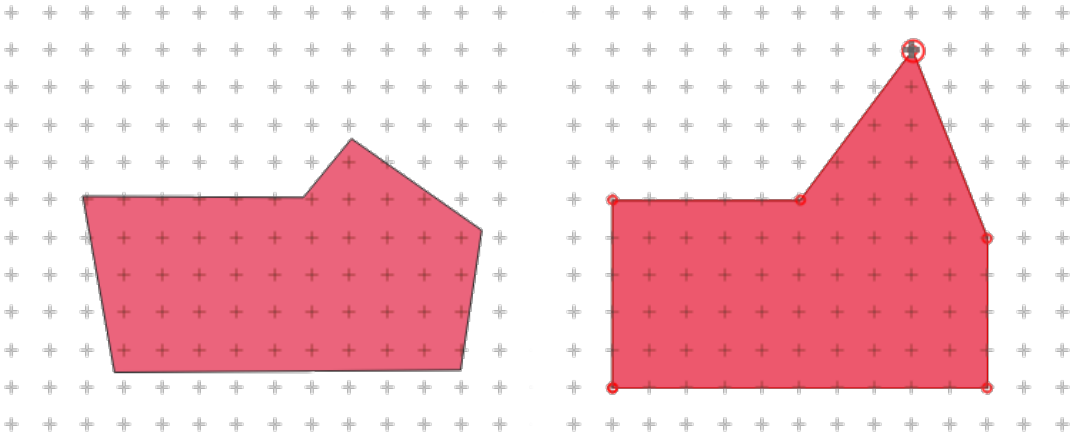


Fig. 12.84: Moving the top vertex snaps all the vertices to the grid

- **Converting adjacent segments to/from curve**: Select the center vertex of the segment you want to convert, hit the ○ letter key. If the vertex was in a curve, the curve is converted into straight lines. If the vertex was between two straight lines, they are converted into a curve. A first or a last vertex of a line can't be converted to a center vertex curve. The layer must be compatible with curve geometry type.

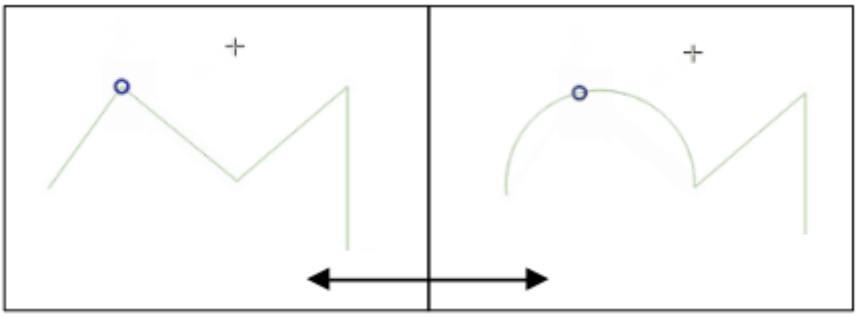


Fig. 12.85: Switch from curve to straight lines with ○ letter

Each change made with the vertex tool is stored as a separate entry in the *Undo* dialog. Remember that all operations support topological editing when this is turned on. On-the-fly projection is also supported.

The Vertex Editor Panel

With enabling a vertex tool, you also open the *Vertex Editor* panel. Right-clicking over a feature fills the panel with the list of all the vertices of the feature with their *x*, *y* (*z*, *m* if applicable) coordinates and *r* (for the radius, in case of circular geometry). The feature is also made exclusive for editing, meaning that the edit of any other features is disabled:

- Selecting a row in the table does select the corresponding vertex in the map canvas, and vice versa.
- Clicking or dragging over the map canvas will only select or move vertices and segments of that feature

- Change a coordinate in the table and the vertex position is updated. This is a convenient way to edit Z coordinate or M value on vertices.
- You can also select multiple rows and delete them altogether.
- New vertices can only be added to the bound feature

If you do not want the *Vertex Editor* panel to immediately show each time you interact with vertex tools (and potentially hide other panels or disturb panels placement), uncheck the *Auto-open table* entry in the ☰ Options menu at the top of the panel. You can then also close the panel. To reopen the panel, you would need to right-click over a panel or toolbar and select it in the list or tick the *Show vertex editor* entry in the *Digitizing toolbar*.

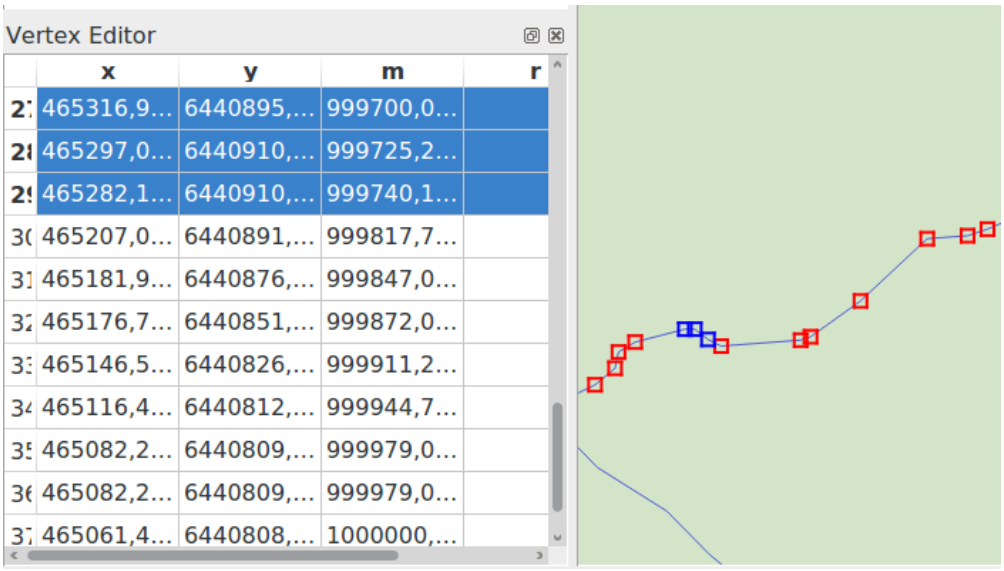


Fig. 12.86: Vertex editor panel showing selected nodes

Rules of Z coordinate or M value assignment

Digitizing 3D vector features or features with M value is not that different from (X,Y) 2D layers'. Tools and options described in this chapter are still available and help you place the vertex or point in a planar environment. Then you may need to handle the Z coordinate (or M value) assignment:

- By default, QGIS will assign to new vertices the *Default Z value* (respectively *Default M value*) set in the *Settings* ► *Options* ► *Digitizing* tab. If the *Advanced Digitizing Panel* is in use, then the value is taken from its *z* (respectively *m*) widget.
- When snapping to a vertex, the new or moved vertex takes the snapped one's Z or M value.
- When snapping to a segment while the topological editing is on, then the new vertex Z or M value is interpolated along the segment.
- If the *z* (respectively *m*) widget of the *Advanced Digitizing Panel* is locked, then its value is applied to the vertex, taking precedence over any snapped vertex or segment Z or M value.

To edit Z or M values of an existing feature, you can use the *Vertex editor panel*. To create features with custom Z or M values you may want to rely on the *Advanced Digitizing Panel*.

Cutting, Copying and Pasting Features

Selected features can be cut, copied and pasted between layers in the same QGIS project, as long as destination layers are set to  Toggle editing beforehand.

> **Tip**
>
> **Transform polygon into line and vice-versa using copy/paste**
>
> Copy a line feature and paste it in a polygon layer: QGIS pastes in the target layer a polygon whose boundary corresponds to the closed geometry of the line feature. This is a quick way to generate different geometries of the same data.

Features can also be pasted to external applications as text. That is, the features are represented in CSV format, with the geometry data appearing in the OGC Well-Known Text (WKT) format. WKT and GeoJSON features from outside QGIS can also be pasted to a layer within QGIS.

When would the copy and paste function come in handy? Well, it turns out that you can edit more than one layer at a time and copy/paste features between layers. Why would we want to do this? Say we need to do some work on a new layer but only need one or two lakes, not the 5,000 on our `big_lakes` layer. We can create a new layer and use copy/paste to plop the needed lakes into it.

As an example, we will copy some lakes to a new layer:

1. Load the layer you want to copy from (source layer)
2. Load or create the layer you want to copy to (target layer)
3. Start editing for target layer
4. Make the source layer active by clicking on it in the legend
5. Use the  Select Features by area or single click tool to select the feature(s) on the source layer
6. Click on the  Copy Features tool
7. Make the destination layer active by clicking on it in the legend
8. Click on the  Paste Features tool
9. Stop editing and save the changes

What happens if the source and target layers have different schemas (field names and types are not the same)? QGIS populates what matches and ignores the rest. If you don't care about the attributes being copied to the target layer, it doesn't matter how you design the fields and data types. If you want to make sure everything - the feature and its attributes - gets copied, make sure the schemas match.

> **Note**
>
> **Congruency of Pasted Features**
>
> If your source and destination layers use the same projection, then the pasted features will have geometry identical to the source layer. However, if the destination layer is a different projection, then QGIS cannot guarantee the geometry is identical. This is simply because there are small rounding-off errors involved when converting between projections.

> **Tip**
>
> **Copy string attribute into another**

If you have created a new column in your attribute table with type 'string' and want to paste values from another attribute column that has a greater length the length of the column size will be extended to the same amount. This is because the GDAL Shapefile driver knows to auto-extend string and integer fields to dynamically accommodate for the length of the data to be inserted.

Deleting Selected Features

If we want to delete an entire feature (attribute and geometry), we can do that by first selecting the geometry using the regular Select Features by area or single click tool. Selection can also be done from the attribute table. Once you have the selection set, press `Delete` or `Backspace` key or use the Delete Selected tool to delete the features. Multiple selected features can be deleted at once.

The Cut Features tool on the digitizing toolbar can also be used to delete features. This effectively deletes the feature but also places it on a "spatial clipboard". So, we cut the feature to delete. We could then use the Paste Features tool to put it back, giving us a one-level undo capability. Cut, copy, and paste work on the currently selected features, meaning we can operate on more than one at a time.

Undo and Redo

The Undo and Redo tools allows you to undo or redo vector editing operations. There is also a dockable widget, which shows all operations in the undo/redo history (see Fig. 12.87). This widget is not displayed by default; it can be displayed by right-clicking on the toolbar and activating the *Undo/Redo Panel* checkbox. The Undo/Redo capability is however active, even if the widget is not displayed.

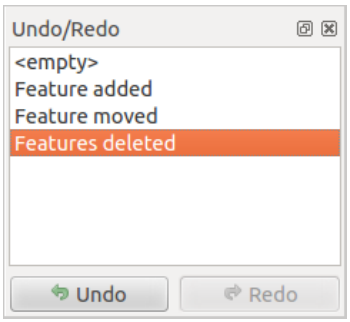


Fig. 12.87: Redo and Undo digitizing steps

When Undo is hit or `Ctrl+Z` (or `Cmd+Z`) pressed, the state of all features and attributes are reverted to the state before the reverted operation happened. Changes other than normal vector editing operations (for example, changes done by a plugin) may or may not be reverted, depending on how the changes were performed.

To use the undo/redo history widget, simply click to select an operation in the history list. All features will be reverted to the state they were in after the selected operation.

Saving Edited Layers

When a layer is in editing mode, any changes remain in the memory of QGIS. Therefore, they are not committed/saved immediately to the data source or disk. If you want to save edits to the current layer but want to continue editing without leaving the editing mode, you can click the  Save Layer Edits button. When you turn editing mode off with  Toggle editing (or quit QGIS for that matter), you are also asked if you want to save your changes or discard them.

If the changes cannot be saved (e.g., disk full, or the attributes have values that are out of range), the QGIS in-memory state is preserved. This allows you to adjust your edits and try again.

> **Tip**
>
> **Data Integrity**
>
> It is always a good idea to back up your data source before you start editing. While the authors of QGIS have made every effort to preserve the integrity of your data, we offer no warranty in this regard.

Saving multiple layers at once

This feature allows the digitization of multiple layers. Choose  *Save for Selected Layers* to save all changes you made in multiple layers. You also have the opportunity to  *Rollback for Selected Layers*, so that the digitization may be withdrawn for all selected layers. If you want to stop editing the selected layers,  *Cancel for Selected Layer(s)* is an easy way.

The same functions are available for editing all layers of the project.

> **Tip**
>
> **Use transaction group to edit, save or rollback multiple layers changes at once**
>
> When working with layers from the same PostGreSQL database, activate the *Automatically create transaction groups where possible* option in *Project* ► *Properties...* ► *Data Sources* to sync their behavior (enter or exit the edit mode, save or rollback changes at the same time).

12.3.5 Advanced digitizing

Table 12.4: Vector layer advanced editing toolbar

Icon	Purpose	Icon	Purpose
	Enable Advanced Digitizing Tools		
	Move Feature(s)		Copy and Move Feature(s)
	Rotate Feature(s)		Simplify Feature
	Scale Feature		
	Add Ring		Add Part
	Fill Ring		Swap direction
	Delete Ring		Delete Part
	Offset Curve		Reshape Features
	Split Parts		Split Features
	Merge Attributes of Selected Features		Merge Selected Features
	Rotate Point Symbols		Offset Point Symbols
	Trim or Extend Feature		

Move Feature(s)

The  Move Feature(s) tool allows you to move existing features:

1. Select the feature(s) to move.
2. Click on the map canvas to indicate the origin point of the displacement; you can rely on snapping capabilities to select an accurate point.

   You can also take advantages of the *advanced digitizing constraints* to accurately set the origin point coordinates. In that case:

   1. First click on the  button to enable the panel.
   2. Type `x` and enter the corresponding value for the origin point you'd like to use. Then press the  button next to the option to lock the value.
   3. Do the same for the `y` coordinate.
   4. Click on the map canvas and your origin point is placed at the indicated coordinates.
3. Move over the map canvas to indicate the destination point of the displacement, still using snapping mode or, as above, use the advanced digitizing panel which would provide complementary `distance` and `angle` placement constraints to place the end point of the translation.
4. Click on the map canvas: the whole features are moved to new location.

Likewise, you can create a translated copy of the feature(s) using the  Copy and Move Feature(s) tool.

Note

If no feature is selected when you first click on the map canvas with any of the *Move Feature(s)* or *Copy and Move Feature(s)* tools, then only the feature under the mouse is affected by the action. So, if you want to move several features, they should be selected first.

Rotate Feature(s)

Use the  Rotate Feature(s) tool to rotate one or multiple features in the map canvas:

1. Press the  Rotate Feature(s) icon
2. Then click on the feature to rotate. The feature's centroid is referenced as rotation center, a preview of the rotated feature is displayed and a widget opens showing the current *Rotation* angle.
3. Click on the map canvas when you are satisfied with the new placement or manually enter the rotation angle in the text box. You can also use the *Snap to* ° box to constrain the rotation values.
4. If you want to rotate several features at once, they shall be selected first, and the rotation is by default around the centroid of their combined geometries.

You can also use an anchor point different from the default feature centroid: press the `Ctrl` button, click on the map canvas and that point will be used as the new rotation center.

If you hold `Shift` before clicking on the map, the rotation will be done in 45 degree steps, which can be modified afterwards in the user input widget.

To abort feature rotation, press the `ESC` button or click on the  Rotate Feature(s) icon.

Scale Feature

The  Scale Feature tool is similar to the Rotate feature. Though instead of performing a rotation of selected features, it rescales their geometry. The change is performed in relation to the anchor point and the scale ratio can be manually specified in the widget that appears in the upper corner of the canvas.

Simplify Feature

The  Simplify Feature tool allows you to interactively reshape a line or polygon geometry by reducing or densifying the number of vertices, as long as the geometry remains valid:

1. Select the  Simplify Feature tool.
2. Click on the feature or drag a rectangle over the features.
3. A dialog pops up allowing you to define the *Method* to apply, ie whether you would like to:
   - *simplify the geometry*, meaning less vertices than the original. Available methods are `Simplify by distance`, `Simplify by snapping to grid` or `simplify by area (Visvalingam)`. You'd then need to indicate the value of *Tolerance* in `Layer units`, `Pixels` or `map units` to use for simplification. The higher the tolerance is the more vertices can be deleted.
   - or *densify the geometries* with new vertices thanks to the `Smooth` option: for each existing vertex, two vertices are placed on each of the segments originated from it, at an *Offset* distance representing the percentage of the segment length. You can also set the number of *Iterations* the placement would be processed: the more iterations, the more vertices and smoother is the feature.

   Settings that you used will be saved when leaving a project or an edit session. So you can go back to the same parameters the next time you simplify a feature.

4. A summary of the modifications that would apply is shown at the bottom of the dialog, listing number of features and number of vertices (before and after the operation and the ratio the change represents). Also, in the map canvas, the expected geometry is displayed over the existing one, using the rubberband color.
5. When the expected geometry fits your needs, click *OK* to apply the modification. Otherwise, to abort the operation, you can either press *Cancel* or right-click in the map canvas.

> **Note**
>
> Unlike the feature simplification option in *Settings* ► *Options* ► *Rendering* menu which simplifies the geometry just for rendering, the  Simplify Feature tool permanently modifies feature's geometry in data source.

Add Part

You can  Add Part to a selected feature generating a multipoint, multiline or multipolygon feature. The new part must be digitized outside the existing one which should be selected beforehand.

The  Add Part can also be used to add a geometry to a geometryless feature. First, select the feature in the attribute table and digitize the new geometry with the  Add Part tool.

> **Note**
>
> **Order of vertices in polygon parts**
>
> Unlike the OGC standards, QGIS doesn't constrain vertices of the exterior boundary of a polygon feature to be ordered counterclockwise. Thus, you can find both directions in a layer. However, every parts of the same multipolygon feature will have their outer vertices ordered following the same direction.
>
> You can however use the *Force right-hand-rule* algorithm to constrain features of a layer to have vertices of their outer boundaries ordered in the clockwise direction.

Delete Part

The  Delete Part tool allows you to delete parts from multifeatures (e.g., to delete polygons from a multi-polygon feature). This tool works with all multi-part geometries: point, line and polygon. Furthermore, it can be used to totally remove the geometric component of a feature. To delete a part, simply click within the target part.

Add Ring

You can create ring polygons using the  Add Ring icon in the toolbar. This means that inside an existing area, it is possible to digitize further polygons that will occur as a 'hole', so only the area between the boundaries of the outer and inner polygons remains as a ring polygon.

To add a ring:

1. Select the feature(s) to modify
2. Activate the  Add Ring tool
3. Draw a polygon within the selected geometries, using the aforementioned *techniques*. A hole appears in the selected geometries.

4. If no geometry is selected when the ring is drawn, then a hole is added to each of the polygons the ring is drawn over.

> **Note**
>
> **Order of vertices in polygon rings**
>
> Unlike the OGC standards, QGIS doesn't constrain vertices of the exterior boundary of a polygon feature to be ordered counterclockwise. Thus, you can find both directions in a layer. However, every rings of the same (multi)polygon feature will have their vertices ordered in the opposite direction to the outer boundary's.
>
> You can however use the *Force right-hand-rule* algorithm to constrain features of a layer to have vertices of their outer boundaries ordered in the clockwise direction, and vertices of their interior rings ordered in the counter-clockwise direction.

Fill Ring

The  Fill Ring tool helps you create polygon feature that totally falls within another one without any overlapping area; that is the new feature covers a hole within the existing one. To create such a feature:

1. Select the  Fill Ring tool.
2. Draw a new polygon over the existing feature: QGIS adds a ring to its geometry (like if you used the  Add Ring tool) and creates a new feature whose geometry matches the ring (like if you *traced* over the interior boundaries with the  Add polygon feature tool).
3. Or alternatively, if the ring already exists on the feature, place the mouse over the ring and left-click while pressing `Shift`: a new feature filling the hole is drawn at that place.

   The *Feature Attributes* form of the new feature opens, pre-filled with values of the "parent" feature and/or *fields constraints*.

Delete Ring

The  Delete Ring tool allows you to delete rings within an existing polygon, by clicking inside the hole. This tool only works with polygon and multi-polygon features. It doesn't change anything when it is used on the outer ring of the polygon.

Reshape Features

You can reshape line and polygon features using the  Reshape Features tool on the toolbar. For lines, it replaces the line part from the first to the last intersection with the original line.

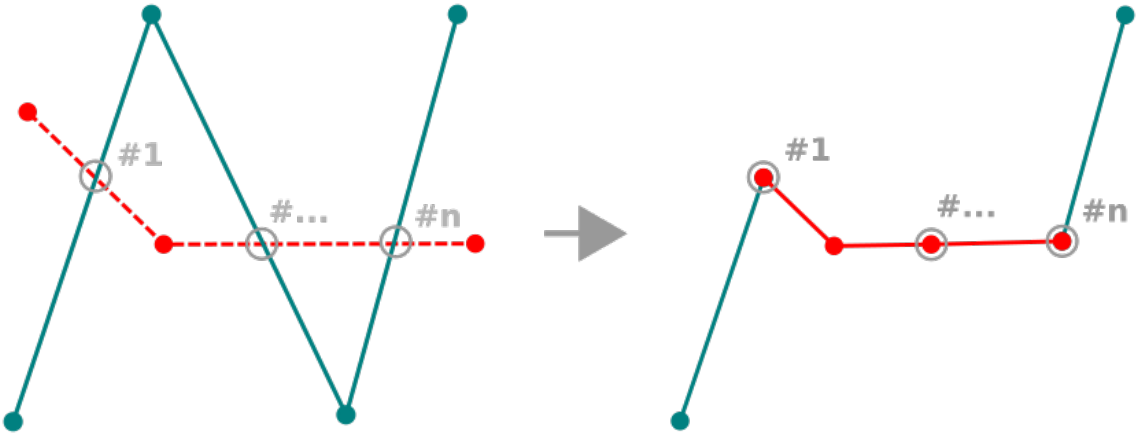


Fig. 12.88: Reshape line

Tip

Extend linestring geometries with reshape tool

Use the Reshape Features tool to extend existing linestring geometries: snap to the first or last vertex of the line and draw a new one. Validate and the feature's geometry becomes the combination of the two lines.

For polygons, it will reshape the polygon's boundary. For it to work, the reshape tool's line must cross the polygon's boundary at least twice. To draw the line, click on the map canvas to add vertexes. To finish it, just right-click. Like with the lines, only the segment between the first and the last intersections is considered. The reshape line's segments that are inside the polygon will result in cropping it, where the ones outside the polygon will extend it.

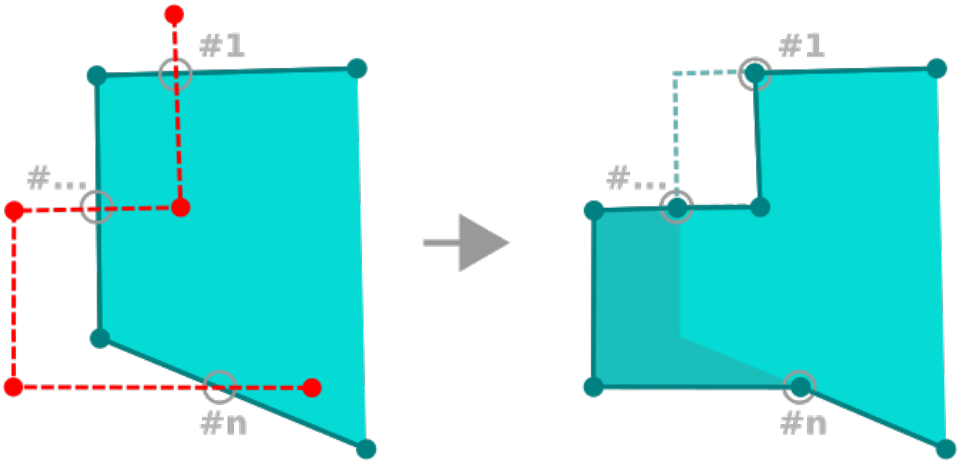


Fig. 12.89: Reshape polygon

With polygons, reshaping can sometimes lead to unintended results. It is mainly useful to replace smaller parts of a polygon, not for major overhauls, and the reshape line is not allowed to cross several polygon rings, as this would generate an invalid polygon.

> **Note**
>
> The reshape tool may alter the starting position of a polygon ring or a closed line. So, the point that is represented 'twice' will not be the same any more. This may not be a problem for most applications, but it is something to consider.

Offset Curves

The  Offset Curve tool creates parallel shifts of line layers. The tool can be applied to the edited layer (the geometries are modified) or also to background layers (in which case it creates copies of the lines / rings and adds them to the edited layer). It is thus ideally suited for the creation of distance line layers. The *User Input* dialog pops-up, showing the displacement distance and other settings.

To create a shift of a line or polygon layer, you must first go into editing mode and activate the  Offset Curve tool. Then click on a feature to shift it. Move the mouse and click where wanted or enter the desired distance in the user input widget. Holding `Ctrl` during the 2nd click will make an offset copy. Your changes may then be saved with the  Save Layer Edits tool.

For geometries on background layers make sure that snapping is on and hold `Ctrl` to select the geometry from the background. Also hold `Ctrl` when doing the second click. Geometries will be converted to the target layer geometry type.

QGIS options dialog (Digitizing tab then **Curve offset tools** section) or the  icon in the *User Input* dialog allows you to configure *some parameters* like **Join style**, **Quadrant segments**, **Miter limit** and **End cap style**.

Reverse Line

Changing the direction of a line geometry can be useful for cartographical purposes or when preparing for network analysis.

To change a line direction:

1. Activate the reverse line tool by clicking  Reverse line.
2. Click on the line. The direction of the line is reversed.

Split Features

Use the  Split Features tool to split a feature into two or more new and independent features, ie. each geometry corresponding to a new row in the attribute table.

To split line or polygon features:

1. Select the  Split Features tool.
2. Draw a line across the feature(s) you want to split. If a selection is active, only selected features are split. The original feature is then assigned the biggest geometry resulting from the splitting, and new features are created for the remaining parts. Fields of the features are filled/updated according to the datasource provider rules or their *splitting policy*.
3. You can then as usual modify any of the attributes of any resulting feature.

> **Tip**
>
> **Split a polyline into new features in one-click**
>
> Using the  Split Features tool, snap and click on an existing vertex of a polyline feature to split that feature into two new features.

Split parts

In QGIS it is possible to split the parts of a multi part feature so that the number of parts is increased. Just draw a line across the part you want to split using the  Split Parts icon.

> **Tip**
>
> **Split a polyline into new parts in one-click**
>
> Using the  Split Parts tool, snap and click on an existing vertex of a polyline feature to split the feature into two new polylines belonging to the same feature.

Merge selected features

The  Merge Selected Features tool allows you to create a new feature by merging existing ones: their geometries are merged to generate a new one. If features don't have common boundaries, a multipolygon/multipolyline/multipoint feature is created.

1. First, select the features you'd like to combine.
2. Then press the  Merge Selected Features button.
3. In the new dialog, the *Merge* line at the bottom of the table shows the attributes of the resulting feature. If any *Merge policy* have been defined, they are automatically applied to populate this row. However, you can override these values manually. Modify the values in the following ways:
   - manually replacing the value in the corresponding cell;
   - selecting a row in the table and pressing *Take attributes from selected feature* to use the values of this initial feature;
   - pressing the *Take attributes from the largest geometry* to use the attributes from the longest line feature, the largest polygon, or the multipoints with the most parts;
   - pressing *Skip all fields* to use empty attributes;
   - expanding the drop down menu at the top of the table, select any of the above options to apply to the corresponding field only. There, you can also choose to aggregate the initial features attributes (Minimum, Maximum, Median, Sum, Count, Concatenation… depending on the type of the field. see *Statistical Summary Panel* for the full list of functions).

   > **Note**
   >
   > If the layer has default values or clauses present on fields, these are used as the initial value for the merged feature.

4. Press *OK* to apply the modifications. A single (multi)feature is created in the layer, replacing the previously selected ones.

Merge attributes of selected features

The Merge Attributes of Selected Features tool allows you to apply same attributes to features without merging their boundaries. The dialog is the same as the `Merge Selected Features` tool's except that unlike that tool, selected objects are kept with their geometry while some of their attributes are made identical.

Rotate Point Symbols

The Rotate Point Symbols allows you to individually change the rotation of point symbols in the map canvas.

1. First, you need to indicate the field to store the rotation value in. This is made by assigning a field to the symbol *data-defined* rotation property:
   1. In the *Layer Properties* ► *Symbology* dialog, browse to the symbol editor dialog.
   2. Click the *Data-defined override* widget near the *Rotation* option of the top *Marker* level (preferably) of the symbol layers.
   3. Choose a field in the *Field Type* combobox. Values of this field are hence used to rotate each feature's symbol accordingly.

      You can also check the Store data in project entry to generate an *auxiliary data storage* field to control the rotation value.

> **Note**
>
> **Make sure that the same field is assigned to all the symbol layers**
>
> Setting the data-defined rotation field at the topmost level of the symbol tree automatically propagates it to all the symbol layers, a prerequisite to perform graphical symbol rotation with the *Rotate Point Symbols* tool. Indeed, if a symbol layer does not have the same field attached to its rotation property, the tool will not work.

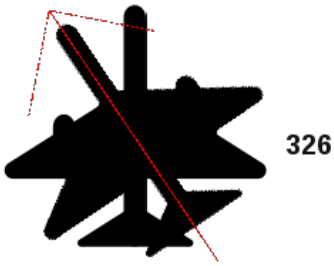


Fig. 12.90: Rotating a point symbol

2. Then click on a point symbol in the map canvas with the Rotate Point Symbols tool
3. Move the mouse around. A red arrow with the rotation value will be visualized (see Fig. 12.90). If you hold the `Ctrl` key while moving, the rotation will be done in 15 degree steps.
4. When you get the expected angle value, click again. The symbol is rendered with this new rotation and the associated field is updated accordingly.

   You can right-click to abort symbol rotation.

Offset Point Symbols

The  Offset Point Symbols allows you to interactively change the rendered position of point symbols in the map canvas. This tool behaves like the  Rotate Point Symbols tool except that it requires you to connect a field to the data-defined *Offset (X,Y)* property of each layer of the symbol. The field will then be populated with the offset coordinates for the features whose symbol is moved in the map canvas.

1. Associate a field to the data-defined widget of the *Offset (X,Y)* property of the symbol. If the symbol is made with many layers, you may want to assign the field to each of them
2. Select the  Offset Point Symbols tool
3. Click a point symbol
4. Move to a new location
5. Click again. The symbol is moved to the new place. Offset values from the original position are stored in the linked field.

   You can right-click to abort symbol offset.

> **Note**
>
> The  Offset Point Symbols tool doesn't move the point feature itself; you should use the  Vertex Tool (Current Layer) or  Move Feature tool for this purpose.

Trim/Extend Feature

The  Trim/Extend tool allows you to shorten or lengthen segments of a (multi)line or (multi)polygon geometry to converge with a selected segment (the cutting line). This results in a modified geometry with a vertex snapped to a reference segment or in its prolongation. Depending on how the selected geometries are placed in relation to each other, the tool will either:

- **Trim**: removes parts of the line segment or polygon boundary, beyond the cutting line
- **Extend**: extends polygon boundaries or line segments so that they can snap to the cutting line.

In order to trim or extend existing geometries:

1. Select the  Trim/Extend tool
2. Click the reference segment, i.e., the segment with respect to which you want to extend or trim another segment. It doesn't need to be on the active layer. A red dotted line appears, visually extending the reference segment across the map canvas according to the active layer's CRS.
3. Hover over the target segment, i.e., the one you want to trim or extend. It does not need to be the last segment of the geometry, but has to be on the active layer. QGIS displays a preview of the feature's geometry with the segment limited to its intersection with the highlighted red dotted extending line.
   1. If an extension of the segment is required, click anywhere on the segment.
   2. In the case of a trim, click the part that should remain.
   3. In either case, the feature's geometry is updated accordingly. When both segments are in 3D, the tool performs an interpolation on the limit segment to get the Z value.
4. If you need to use the same reference segment for trimming or extending many features:
   1. Press `Shift` while selecting the reference segment.
   2. Click consecutively on the segments to modify and each will be trimmed or extended accordingly.

Note

Snapping is automatically enabled when this tool is activated. Your original snapping settings will be restored once the tool is deactivated.

Attention

Pay attention to the modified geometry while using the Trim/Extend tool. Depending on the inputs, it can create invalid geometries, potentially resulting in failure at layer saving.

12.3.6 Shape digitizing

The *Shape Digitizing* toolbar offers a set of tools to draw lines or polygons features of regular shape. It is synchronized with the Digitize Shape *geometry drawing method* you can select on the *Digitizing Toolbar*. To use it:

1. Display the toolbar: *View* ► *Toolbars* ► *Shape Digitizing*
2. Select a tool that creates or modifies the shape of a geometry, e.g. Add line feature, Add polygon feature, Add part, Add ring, Reshape Features, ...
3. The Digitize with segment button on the *Digitizing Toolbar* is enabled. The first time, you may need to switch it to the Digitize Shape in order to enable tools on the *Shape Digitizing* toolbar.
4. Pick a shape digitizing tool and draw.

Circular string by radius

The Circular string by radius button allows to add line or polygon features with a circular geometry, given two nodes on the curve and a radius:

1. Left click twice to place the two points on the geometry.
2. A *Radius* widget in the top right corner of the map canvas displays current radius (corresponding to distance between the points). Edit that field to the value you want.
3. An overview of the arcs matching these constraints is displayed while moving around the cursor. Right-click to validate when the expected arc is shown.
4. Add a new point to start shaping another arc.

Note

Curved geometries are stored as such only in compatible data provider

Although QGIS allows to digitize curved geometries within any editable data format, you need to be using a data provider (e.g. PostgreSQL, Oracle Spatial, memory layer, GML or WFS) that supports curves to have features stored as curved, otherwise QGIS segmentizes the circular arcs.

Draw Circles

There is a set of tools for drawing circles. The tools are described below.

Circles are converted into circular strings. Therefore, as explained in *Circular string by radius*, if allowed by the data provider, it will be saved as a curved geometry, if not, QGIS will segmentize the circular arcs.

-  Circle from 2 points: The two points define the diameter and the orientation of the circle. (Left-click, right-click)
-  Circle from 3 points: Draws a circle from three known points on the circle. (Left-click, left-click, right-click)
-  Circle by a center point and another point: Draws a circle with a given center and a point on the circle (Left-click, right-click). When used with the *The Advanced Digitizing panel* this tool can become a "Add circle from center and radius" tool by setting and locking the distance value after first click.
-  Circle from 3 tangents: Draws a circle that is tangential to three segments. **Note that you must activate snapping to segments** (See *Setting the snapping tolerance and search radius*). Click on a segment to add a tangent. If two tangents are parallel, the coordinates of the click on the first parallel tangent are used to determine the positioning of the circle. If three tangents are parallel, an error message appears and the input is cleared. (Left-click, left-click, right-click)
-  Circle from 2 tangents and a point: Similar to circle from 3 tangents, except that you have to select two tangents, enter a radius and select the desired center.

Draw Ellipses

There is a set of tools for drawing ellipses. The tools are described below.

Ellipses cannot be converted as circular strings, so they will always be segmented.

-  Ellipse from center and two points: Draws an ellipse with a given center, major axis and minor axis. (Left-click, left-click, right-click)
-  Ellipse from center and a point: Draws an ellipse into a bounding box with the center and a corner. (Left-click, right-click)
-  Ellipse from extent: Draws an ellipse into a bounding box with two opposite corners. (Left-click, right-click)
-  Ellipse from foci: Draws an ellipse by 2 points for foci and a point on the ellipse. (Left-click, left-click, right-click)

Draw Rectangles

There is a set of tools for drawing rectangles. The tools are described below.

-  Rectangle from center and a point: Draws a rectangle from the center and a corner. (Left-click, right-click)
-  Rectangle from extent: Draws a rectangle from two opposite corners. (Left-click, right-click)
-  Rectangle from 3 points (distance): Draws an oriented rectangle from three points. The first and second points determine the length and angle of the first edge. The third point determines the length of the other edge. One can use *The Advanced Digitizing panel* to set the length of the edges. (Left-click, left-click, right-click)
-  Rectangle from 3 points (projected): Same as the preceding tool, but the length of the second edge is computed from the projection of the third point on the first edge. (Left-click, left-click, right-click)

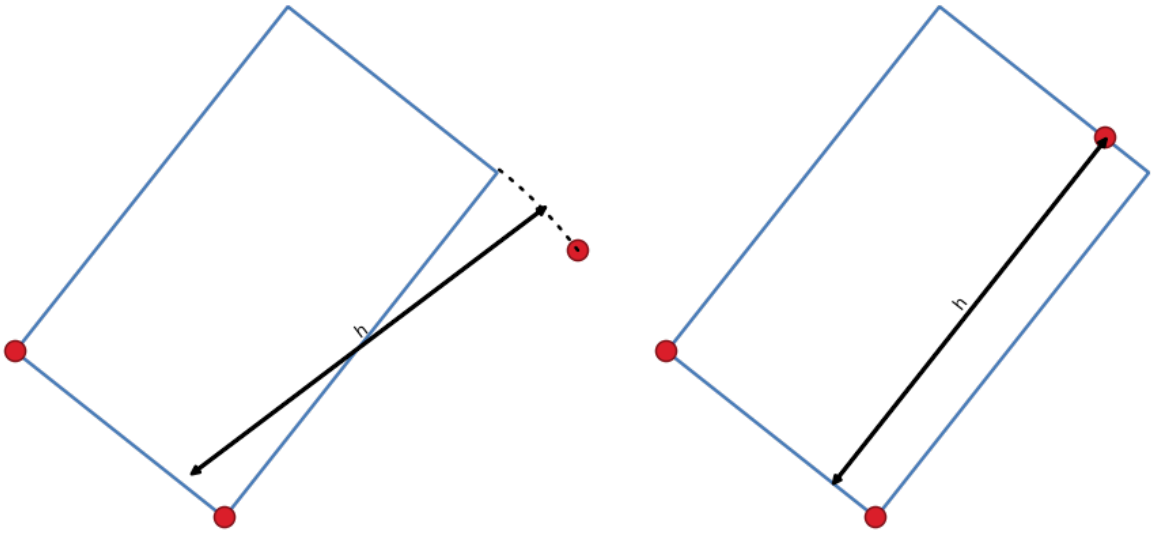


Fig. 12.91: Draw rectangle from 3 points using distance (right) and projected (left)

Draw Regular Polygons

There is a set of tools for drawing regular polygons. The tools are described below. Left-click to place the first point. A dialog appears, where you can set the number of polygon edges. Right-click to finish the regular polygon.

- Regular polygon from two points: Draws a regular polygon where the two points determine the length and angle of the first edge.
- Regular polygon from center and a point: Draws a regular polygon from the provided center point. The second point determines the angle and distance to the middle of an edge.
- Regular polygon from center and a corner: Same as the preceding tool, but the second point determines the angle and distante to a vertex.

12.3.7 The Advanced Digitizing panel

When capturing, reshaping, splitting new or existing geometries you also have the possibility to use the Advanced Digitizing panel. You can digitize lines exactly parallel or perpendicular to a particular angle or lock lines to specific angles. Furthermore, you can make a precise definition of your new geometry by entering X and Y coordinates as well as Z for 3D features, or M values.

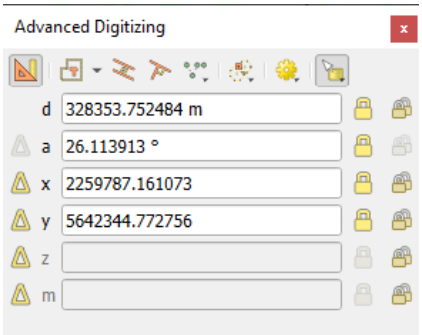


Fig. 12.92: The Advanced Digitizing panel

The *Advanced Digitizing* panel can be opened either with a right-click on the toolbar, from *View* ► *Panels* ► menu or pressing `Ctrl+4`. Once the panel is visible, click the  Enable advanced digitizing tools button to activate the set of tools.

> **Note**
>
> The tools are not enabled if the map view is in geographic coordinates.

The aim of the Advanced Digitizing tool is to lock coordinates, lengths, and angles when moving the mouse during the digitalizing in the map canvas.

You can also create constraints with relative or absolute reference. Relative reference means that the next vertex constraints' values will be relative to the previous vertex or segment.

The toolbar

At the top of the *Digitizing panel*, you find the following buttons:

-  Enable advanced digitizing tools
-  Construction mode: allows to capture the clicks' positions to reuse as reference points to lock distance, angle, X, Y, Z or M relative values. More details at *Construction mode*.
-  Parallel to draw a line parallel to an existing one (more at *Parallel and perpendicular lines*)
-  Perpendicular to draw a line perpendicular to an existing one (more at *Parallel and perpendicular lines*)
-  Construction Tools provides a couple of options that constrain the vertices placement based on extrapolated coordinates of existing elements:
  -  *Line Extension*: hover over a segment and you get a purple dotted line extending the segment across the map canvas. You can snap the vertex anywhere on this virtual line.
  -  *X/Y Point*: hover over a vertex and you get a purple dotted line along its X or Y coordinate, across the map canvas. You can snap the vertex anywhere on this virtual line. It is even possible to hover over two different vertices, generating virtual coordinate lines for both, and snap to their intersection.
-  2-circle Point Intersection: allows you to digitize a point or vertex at the intersection of two circles. (more at *2-circle point intersection*).
-  Snap to common angles: when moving the cursor, displays a virtual line that you can snap to to add the next vertex. The snapping line is defined by the last added vertex and an (absolute or relative to previous segment) angle from a preset list (following steps of 0.1°, 0.5°, 1°, 5°, 10°, 15°, 18°, 22.5°, 30°, 45° or 90°). Choose *Do not snap to common angles* to disable this feature.

  Snapping to features can be used along with snapping to common angles for accurate digitizing. For a fine-grained control on how the target element to snap to is retained, you can indicate whether to prioritize snapping to features over common angles, and vice-versa under the *Snapping priority* entry. You can switch from one method to the other during the digitizing operation, and this avoids disabling any of the snapping options in the meantime. Press `N` (or `Shift+N`) during a digitizing operation to cycle through the angles list.
-  Floater settings: if the *Show floater* item is checked, a contextual menu with digitizing information follows the cursor during digitizing. The values can be accessed using the *panel's shortcuts*, edited and  Locked after validation (pressing `Enter`). The type of information to display can be selected in the bottom part of the menu:
  - *Show distance*
  - *Show angle*

- *Show XY coordinates*
- *Show Z value*
- *Show M value*
- *Show bearing/azimuth*
- *Show common snapping angle*

Below the toolbar, you will find a number of text boxes whose value reflects by default the position or movement of the cursor in the map canvas. Editing these values helps you constrain the position of the items you edit:

- *d* for the distance from a reference position, usually the last edited vertex
- *a* for the angle (absolute or relative) from a reference position, usually the last edited segment
- *x* for the X coordinate of the pointer
- *y* for the Y coordinate of the pointer
- *z* for the default Z value or the Z coordinate of the vertex or segment under the pointer
- *m* for the default M value or the M value of the vertex or segment under the pointer

Keyboard shortcuts

To speed up the use of Advanced Digitizing Panel, there are a couple of keyboard shortcuts available:

Table 12.5: Keyboard shortcuts of the Advanced Digitizing Panel tools

Key	Simple	`Ctrl+` or `Alt+`	`Shift+`
`D`	Set distance	Lock distance	
`A`	Set angle	Lock angle	Toggle relative angle to last segment
`X`	Set X coordinate	Lock X coordinate	Toggle relative X to last vertex
`Y`	Set Y coordinate	Lock Y coordinate	Toggle relative Y to last vertex
`Z`	Set Z coordinate	Lock Z coordinate	Toggle relative Z to last vertex
`M`	Set M value	Lock M value	Toggle relative M to last vertex
`C`	Toggle construction mode		
`P`	Toggle perpendicular and parallel modes		

Note

Z coordinate and M value options are available only if compatible with the layer geometry dimension.

Absolute reference digitizing

When drawing a new geometry from scratch, it is very useful to have the possibility to start digitizing vertexes at given coordinates.

For example, to add a new feature to a polygonal layer, click the button. You can enter the exact coordinates where you want to start editing the feature, i.e.:

1. Click the *x* text box (or use the `X` keyboard shortcut).
2. Type the X coordinate value you want and press `Enter` or click the button to their right to lock the mouse to the X axis on the map canvas.
3. Click the *y* text box (or use the `Y` keyboard shortcut).

4. Type the Y coordinate value you want and press `Enter` or click the button to their right to lock the mouse to the Y axis on the map canvas.
5. If the layer has Z coordinate or M values, the corresponding *z* or *m* widget is enabled and displays its default value, as set in *Settings* ► *Options* ► *Digitizing* tab.
    1. Click the *z* or *m* text box (or use respectively the `Z` or `M` keyboard shortcut).
    2. Type the coordinate value you want and press `Enter` or click the button to their right to lock the value in the widget.

> **Note**
>
> Read *Rules of Z coordinate or M value assignment* for details on how Z coordinate and M values are automatically determined from existing features.

6. Two blue dotted lines and a green cross identify the exact coordinates you entered. Click on the map canvas to add a vertex at the green cross position.

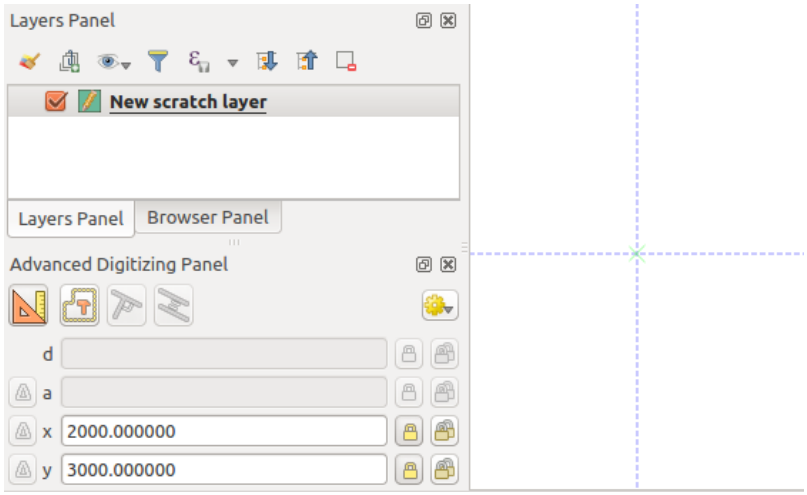


Fig. 12.93: Start drawing at given coordinates

7. You can proceed as above, adding a new set of coordinates for the next vertex, or switch to another *mode of digitizing* (e.g. segment, curve or stream).
8. If you want to draw a segment of a given length:
    1. Click the *d (distance)* text box (keyboard shortcut `D`)
    2. Type the distance value (in map units)
    3. Press `Enter` or click the button on the right to lock the mouse in the map canvas to the length of the segment. In the map canvas, the latest vertex is surrounded by a circle whose radius is the value entered in the distance text box. A cross on the circle shows the position of the next vertex if you click.

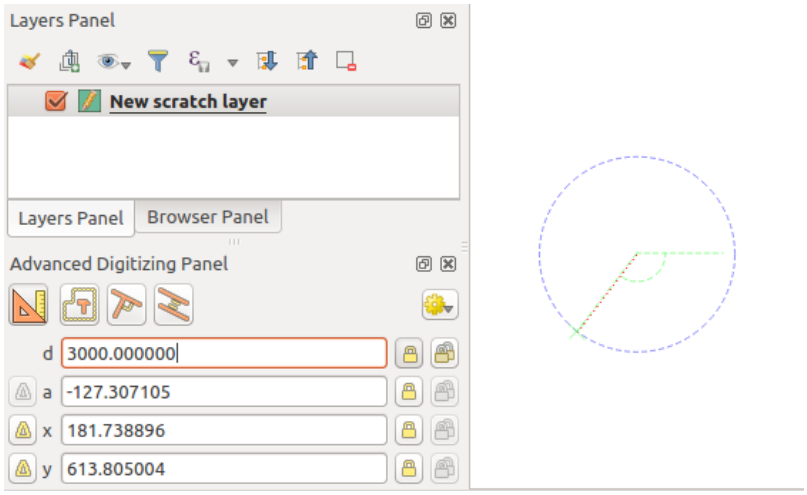


Fig. 12.94: Fixed length segment

9. You can also constrain the vertex position, setting the angle of the segment. As described before:
    1. Click the *a (angle)* text box (keyboard shortcut `A`)
    2. Type the angle value (in degrees)
    3. Press `Enter` or click the  button on the right to lock it. A line going through the latest vertex and rotated based on the set angle appears in the map canvas and a cross on it shows the next vertex position if you click.

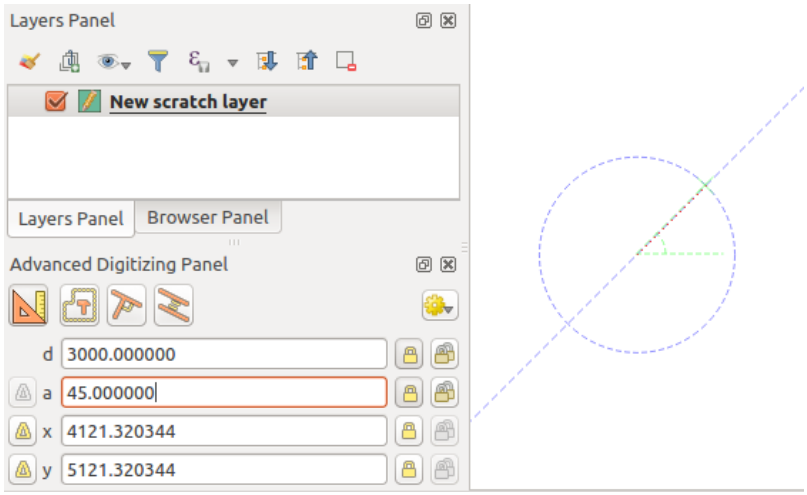


Fig. 12.95: Fixed angle segment

Hint

Pressing `Ctrl+<key>` or `Alt+<key>` automatically locks the target property and puts its value into edit. Modify, press `Enter` and you are done. Combined with the  Toggle floater, this can be a real time saver, with keyboard digitizing.

Relative reference digitizing

Instead of using absolute values of angles or coordinates, you can also use values relative to the last digitized vertex or segment.

For angles, you can click the  button on the left of the *a* text box (or press `Shift+A`) to toggle relative angles to the previous segment. With that option on, angles are measured between the last segment and the mouse pointer.

For coordinates, click the  buttons to the left of the *x*, *y*, *z* or *m* text boxes (or press `Shift+<key>`) to toggle relative coordinates to the previous vertex. With these options on, coordinates measurement will consider the last vertex to be the origin of the set coordinates.

Continuous lock

Both in absolute or relative reference digitizing, angle, distance, X, Y, Z and M constraints can be locked continuously by clicking the  *Continuous lock* buttons. Using continuous lock allows you to digitize several points or vertexes using the same constraints.

Construction mode

You can enable and disable *construction mode* by clicking on the  Construction mode icon or with the `C` keyboard shortcut. While in construction mode, clicking the map canvas won't add new vertexes, but will capture the clicks' positions so that you can use them as reference points to then lock distance, angle or X, Y, Z, M relative values.

In the  Construction mode you will find a drop-down menu where you can choose to:

-  *Record Construction Guides*: All construction steps are rendered as dashed lines. Each step that you perform is visually represented, allowing you to trace the construction process. The guides are displayed as long as *Advanced Digitizing* remains active.
-  *Show Construction Guides*: Allows you to make construction guides visible or hidden on the canvas. When enabled, all active guides are displayed, offering better spatial orientation and precision during the digitizing process.
-  *Snap to Visible Construction Guides*: The guides are snap-able, allowing you to start new construction steps from any point along the existing guides.
- Choose *Clear Construction Guides* to remove all the guides from the canvas.

As an example, the construction mode can be used to draw some point at an exact distance from an existing point.

With an existing point in the map canvas and the snapping mode correctly activated, you can easily draw other points at given distances and angles from it. In addition to the  button, you have to activate also the *construction mode* by clicking the  Construction mode icon or with the `C` keyboard shortcut.

Click next to the point from which you want to calculate the distance and click on the *d* box (`D` shortcut) type the desired distance and press `Enter` to lock the mouse position in the map canvas:

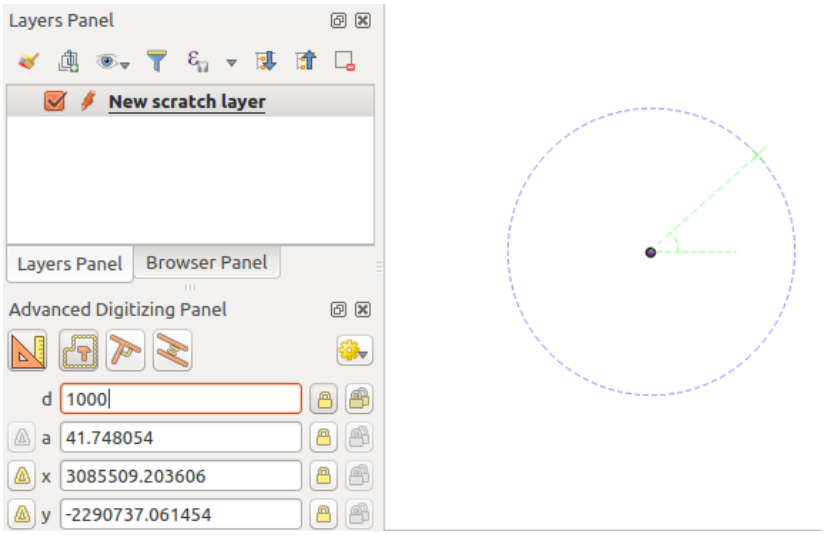


Fig. 12.96: Distance from point

Before adding the new point, press `C` to exit the construction mode. Now, you can click on the map canvas, and the point will be placed at the distance entered.

You can also use the angle constraint to, for example, create another point at the same distance of the original one, but at a particular angle from the newly added point. Click the Construction mode icon or with the `C` keyboard shortcut to enter construction mode. Click the recently added point, and then the other one to set a direction segment. Then, click on the *d* text box (`D` shortcut) type the desired distance and press `Enter`. Click the *a* text box (`A` shortcut) type the angle you want and press `Enter`. The mouse position will be locked both in distance and angle.

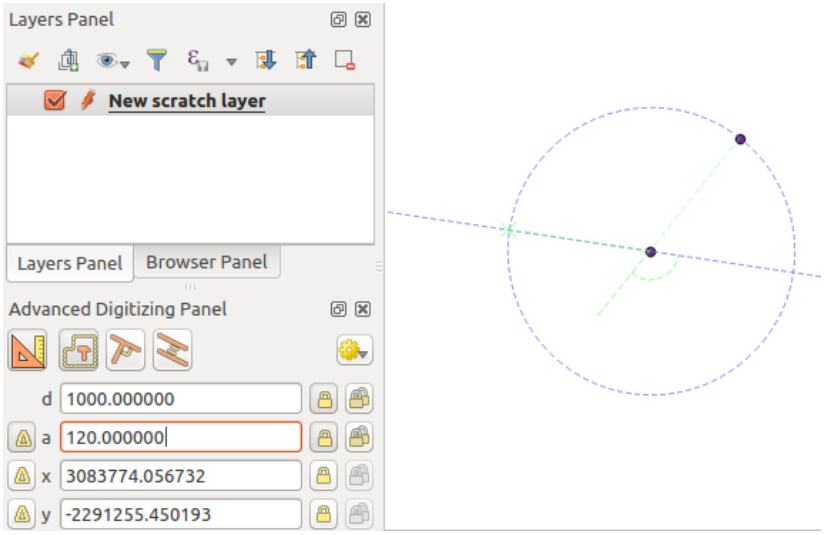


Fig. 12.97: Distance and angle from points

Before adding the new point, press `C` to exit the construction mode. Now, you can click on the map canvas, and the point will be placed at the distance and angle entered. Repeating the process, several points can be added.

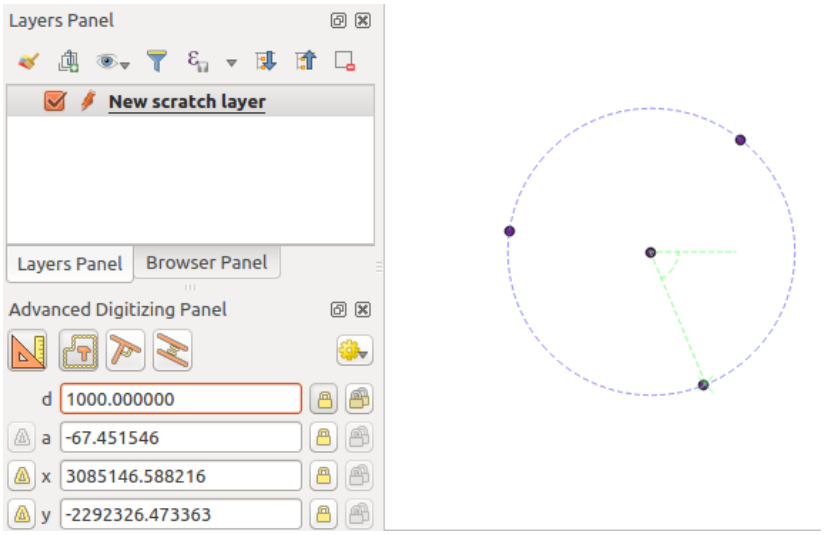


Fig. 12.98: Points at given distance and angle

Parallel and perpendicular lines

All the tools described above can be combined with the Perpendicular and Parallel tools. These two tools allow drawing segments perfectly perpendicular or parallel to another segment. The target segment can be on another layer, another feature within the layer or the feature being digitized (requires *self-snapping option*).

To draw a *perpendicular* segment:

1. First add one of the segment vertices.
2. Click the Perpendicular icon (keyboard shortcut `P`) to activate it.
3. Click on the segment that you want to be perpendicular to.
4. A virtual dotted line perpendicular to the segment through the previous vertex appears. The angle property is locked, constraining the next vertex on that line and, a cross indicates the projected position of the cursor on the line. Click to place the new vertex.

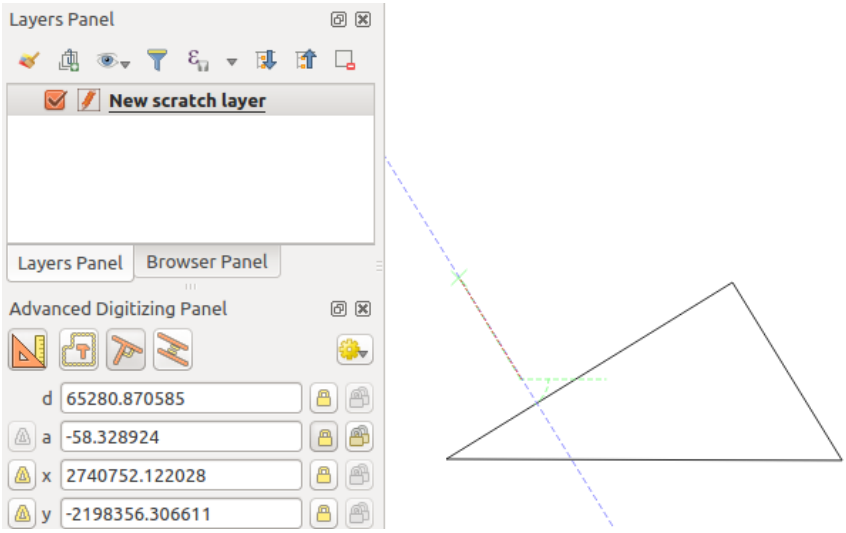


Fig. 12.99: Perpendicular digitizing

To draw a *parallel* segment, the steps are the same except that you need to click on the  Parallel icon (keyboard shortcut `P` twice).

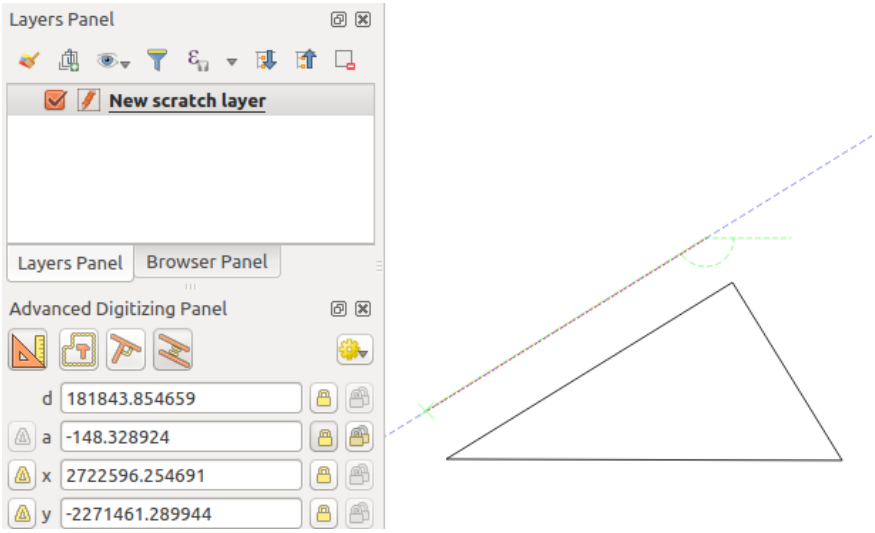


Fig. 12.100: Parallel digitizing

These two tools just find the right angle of the perpendicular and parallel angle and lock this parameter during your editing. Unlock the angle parameter to cancel their use in the middle of the process.

2-circle point intersection

To add a vertex at the intersection of two circles, follow these steps:

1. Click the  2-circle Point Intersection icon.
2. A dialog will open where you can define the parameters for *Circle #1* and *Circle #2*.
3. Click on the map canvas, and the tool will automatically calculate the *X* and *Y* coordinates for the centers of both circles.
4. Enter the distance *d* for each circle.
5. The tool will calculate and display the two intersection points of the circles.
6. Click one of the intersection points to add the new vertex.

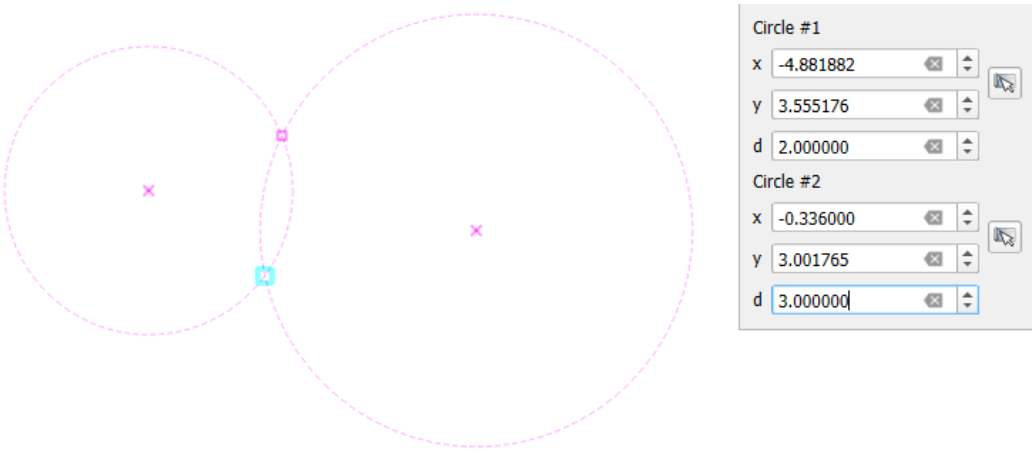


Fig. 12.101: 2-circle Point Intersection

12.3.8 The Processing in-place layer modifier

The *Processing menu* provides access to a large set of tools to analyze and create new features based on the properties of the input features or their relations with other features (within the same layer or not). While the common behavior is to create new layers as outputs, some algorithms also allow modifications to the input layer. This is a handy way to automate multiple features modification using advanced and complex operations.

To edit features in-place:

1. Select the layer to edit in the *Layers* panel.
2. Select the concerned features. You can skip this step, in which case the modification will apply to the whole layer.
3. Press the Edit Features In-Place button at the top of the *Processing toolbox*. The list of algorithms is filtered, showing only those compatible with in-place modifications, i.e.:
   - They work at the feature source and not at the layer level.
   - They do not change the layer structure, e.g. adding or removing fields.
   - They do not change the geometry type, e.g. from line to point layer.

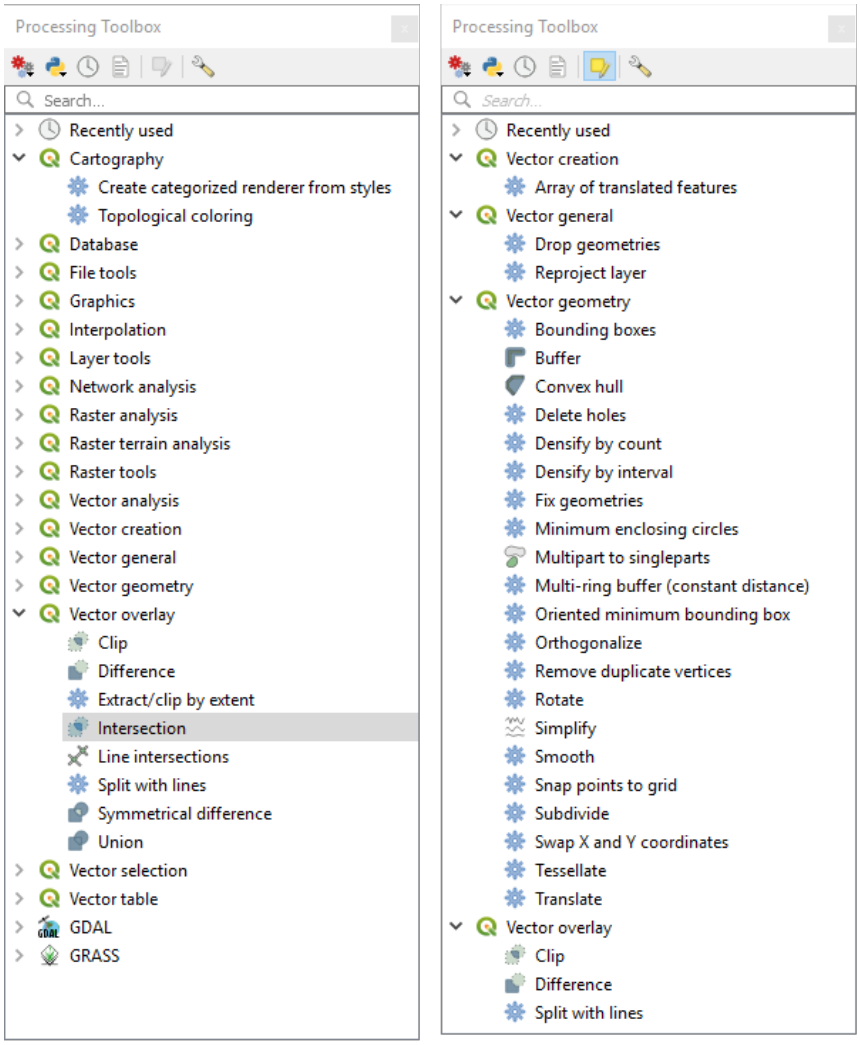


Fig. 12.102: Processing algorithms: all (left) vs polygon in-place editors (right)

4. Find the algorithm you'd like to run and double-click it.

> **Note**
>
> If the algorithm does not need any additional user-set parameters (excluding the usual input and output layer parameters), then the algorithm is run immediately without any dialog popup.

   1. If parameters other than the usual input or output layers are needed, the algorithm dialog pops up. Fill in the required information.
   2. Click *Modify Selected Features* or *Modify All Features* depending on whether there's an active selection.

   Changes are applied to the layer and placed in the edit buffer: the layer is indeed toggled to editing mode with unsaved modification as indicated by the icon next to the layer name.

5. As usual, press Save layer edits to commit the changes in the layer. You can also press Undo to rollback the whole modification.

12.4 Connecting and Editing Data Across Layers

Ability to connect data from different layers is one of the duties of a GIS software. Such a connection can be based on the spatial relationship between the features, or on their shared attributes. QGIS provides tools to handle any of these associations, such as:

- Processing algorithms that can create a new layer as a result of the connection, namely *Join attributes by location*, *Join attributes by nearest*, *Join attributes by field value*, ...
- SQL queries to create a new layer from the *DB Manager* or as a *virtual layer*
- *Joins properties* or *relations settings* that temporarily extend attributes of features in a given layer, with those of features in another layer based on some matching attribute(s).

  Joins and relations are technical concepts borrowed from databases to get the most out of data stored in tables by combining their contents. The idea is that features (rows) of different layers (tables) can be associated to each other. The number of rows which are matching each other can be of any value (zero, one, many).

12.4.1 Joining features between two layers

Joins in QGIS allow you to associate features of the current layer to features from another loaded vector layer. Whether they are spatially enabled and the type of geometry do not matter. The join is based on an attribute that is shared by the layers, in a one-to-one relationship.

To create a join on a layer (identified below as `target layer`):

1. Go to the layer *Properties* ►  *Joins* tab
2. Click the  Add new join button. The *Add vector join* dialog appears.
3. Select the *Join layer* you want to connect with the target vector layer
4. Specify the *Join field* (from the `join layer`) and the *Target field* (from the `target layer`). These are the fields that are used to find matching feature in both layers hence they should have values in common.
5. Press *OK* and a summary of selected parameters is added to the *Join* panel.

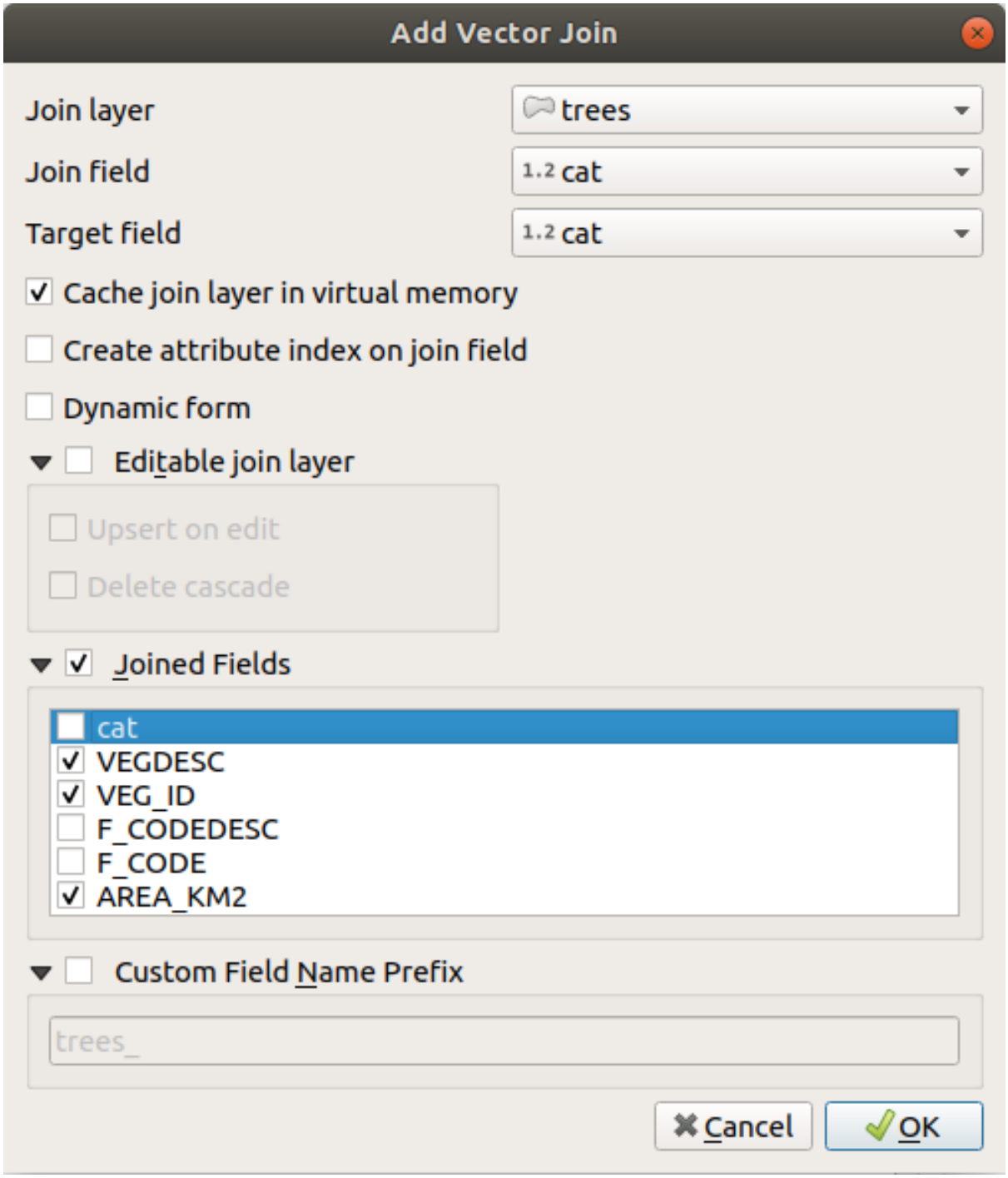


Fig. 12.103: Join an attribute table to an existing vector layer

The steps above will create a join, where **ALL** the attributes of the **first matching feature** in the join layer is added to the target layer's feature. The following logic is used to pair features during a join process:

- All the features in the target layer are returned, regardless they have a match
- If the target field contains duplicate values, these features are assigned the same feature from the join layer.
- If the join field contains duplicate matching values, only the first fetched feature is picked.

Note

Joins in QGIS are based on a single field matching so most of the times, you would want to make sure that values in the matchable fields are unique.

QGIS provides some more options to tweak the join:

- *Cache join layer in virtual memory*: allows you to cache values in memory (without geometries) from the joined layer in order to speed up lookups.
- *Create attribute index on the join field* to speed up lookups
- *Dynamic form*: helps to synchronize join fields on the fly, according to the *Target field*. This way, constraints for join fields are also correctly updated. Note that it's deactivated by default because it may be very time consuming if you have a lot of features or a myriad of joins.
- If the target layer is editable, then some icons will be displayed in the attribute table next to fields, in order to inform about their status:
  - : the join layer is not configured to be editable. If you want to be able to edit join features from the target attribute table, then you have to check the option *Editable join layer*.
  - : the join layer is well configured to be editable, but its current status is read only.
  - : the join layer is editable, but synchronization mechanisms are not activated. If you want to automatically add a feature in the join layer when a feature is created in the target layer, then you have to check the option *Upsert on edit*. Symmetrically, the option *Delete cascade* may be activated if you want to automatically delete join features.
- *Joined fields*: instead of adding all the fields from the joined layer, you can specify a subset.
- *Custom field name prefix* for joined fields, in order to avoid name collision

12.4.2 Setting relations between multiple layers

Unlike joins that define a one-to-one link between features across two layers, relations help you build interconnections between multiple features across two or more layers. As such, relations are project level settings and are set in *Project* ► *Properties* ► *Relations* tab. From there, you can:

- *Add relation* whose type can be:
  - *one to many relation*
  - *many to many relation*
  - *polymorphic relation* that you can add or edit with the dedicated tools in the action drop-down menu.

> **Note**
>
> There is no simple way yet to edit a non-polymorphic relation once it has been created. Only the name can be edited with a double-click. For any other parameters of such a relation you will have to remove and recreate it.

- *Discover relations*: QGIS is able to discover existing relations from supported database formats (PostgreSQL, GeoPackage, ESRI File Geodatabase, …). This can be a good way to ease the relations definition.
- *Remove relation*

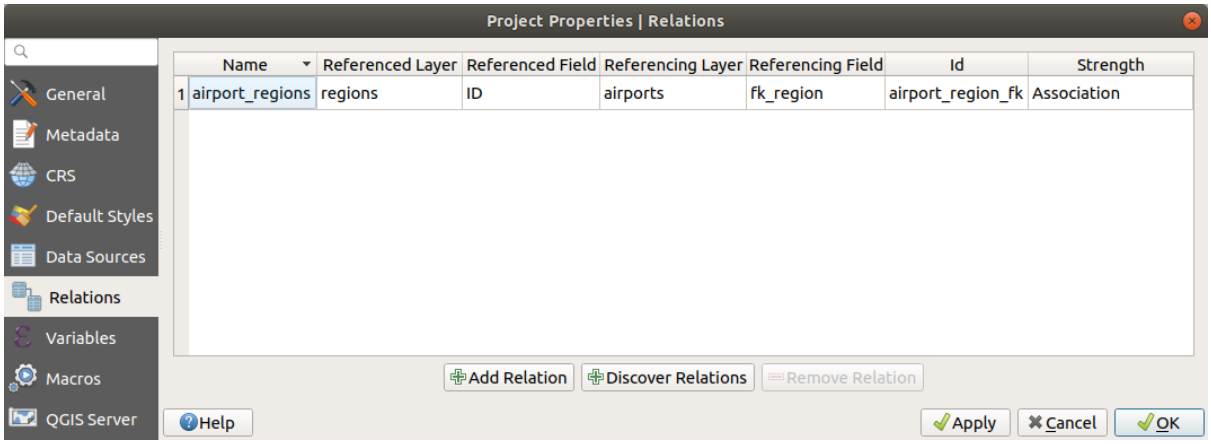


Fig. 12.104: Relations tab

One to many (1-N) relations

As an example you have a layer with all regions of Alaska (polygon) which provides some attributes about its name and region type and a unique id (which acts as primary key).

Then you get another point layer or table with information about airports that are located in the regions and you also want to keep track of these. If you want to add them to the regions layer, you need to create a one to many relation using foreign keys, because there are several airports in most regions.

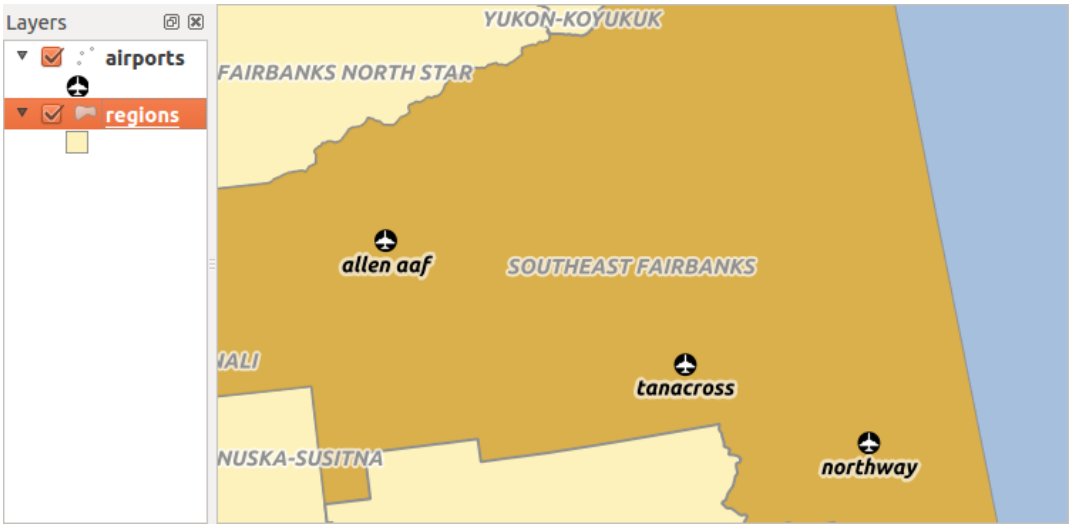


Fig. 12.105: Alaska region with airports

Layers and keys

QGIS makes no difference between a table and a vector layer. Basically, a vector layer is a table with a geometry. So you can add your table as a vector layer. To demonstrate the 1-n relation, you can load the `regions` and `airports` layers in the sample dataset. In practice, each airport belongs to exactly one region while each region can have any number of airports (a typical one to many relation).

In addition to the attributes describing the airports, the aiports layer has another field `fk_region` which acts as a foreign key (if you have a database, you will probably want to define a constraint on it). This `fk_region` field will always contain an id of a region. It can be seen like a pointer to the region it belongs to.

All you have to do is to tell QGIS the relation between the layers so that you can design a custom edit form for editing and QGIS takes care of the setup. It works with different providers (so you can also use it with shape and csv files).

Defining 1-N relations

The first thing we are going to do is to let QGIS know about the relations between the layers. This is done in *Project ► Properties…*. Open the *Relations* tab and click on *Add Relation*.

- **Name** is going to be used as a title. It should be a human readable string describing what the relation is used for. We will just call say **airport_relation** in this case.
- **Referenced Layer (Parent)** also considered as parent layer, is the one with the primary key, pointed to, so here it is the `regions` layer. You need to define the primary key of the referenced layer, so it is `ID`.
- **Referencing Layer (Child)** also considered as child layer, is the one with the foreign key field on it. In our case, this is the `airports` layer. For this layer you need to add a referencing field which points to the other layer, so this is `fk_region`.

> **Note**
>
> Sometimes, you need more than a single field to uniquely identify features in a layer. Creating a relation with such a layer requires a **composite key**, i.e. more than a single pair of matching fields. Use the Add new field pair as part of a composite foreign key button to add as many pairs as necessary.

- **Id** will be used for internal purposes and has to be unique. You may need it to build *custom forms*. If you leave it empty, one will be generated for you but you can assign one yourself to get one that is easier to handle
- **Relationship strength** sets the strength of the relation between the parent and the child layer. The default *Association* type means that the parent layer is *simply* linked to the child one while the *Composition* type allows you to duplicate also the child features when duplicating the parent ones and on deleting a feature the children are deleted as well, resulting in cascade over all levels (means children of children of… are deleted as well).

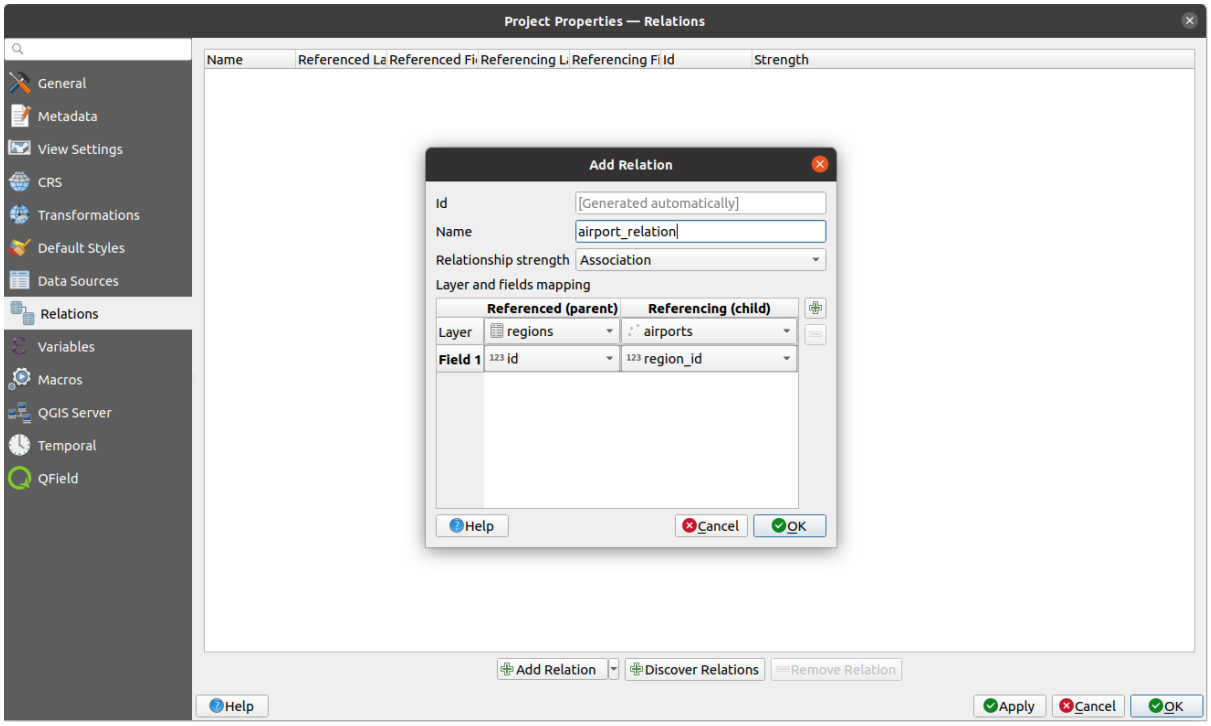


Fig. 12.106: Adding a relation between regions and airports layers

From the *Relations* tab, you can also press the *Discover Relation* button to fetch the relations available from the providers of the loaded layers. This is possible for layers stored in data providers like PostgreSQL or SpatiaLite.

Forms for 1-N relations

Now that QGIS knows about the relation, it will be used to improve the forms it generates. As we did not change the default form method (autogenerated), it will just add a new widget in our form. So let's select the layer region in the legend and use the identify tool. Depending on your settings, the form might open directly or you will have to choose to open it in the identification dialog under actions.

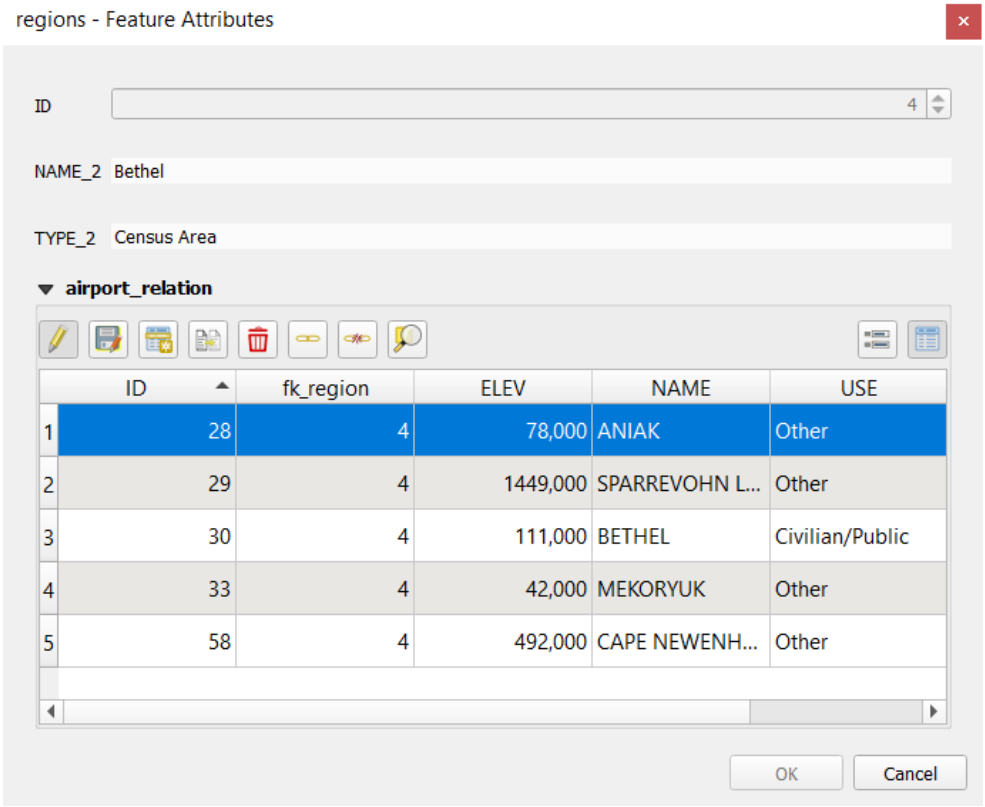


Fig. 12.107: Identification dialog regions with relation to airports

As you can see, the airports assigned to this particular region are all shown in a table. And there are also some buttons available. Let's review them shortly:

- The button is for toggling the edit mode. Be aware that it toggles the edit mode of the airport layer, although we are in the feature form of a feature from the region layer. But the table is representing features of the airport layer.
- The button is for saving all the edits in the child layer (airport).
- The button lets you digitize the airport geometry in the map canvas and assigns the new feature to the current region by default. Note that the icon will change according to the geometry type.
- The button adds a new record to the airport layer attribute table and assigns the new feature to the current region by default. The geometry can be drawn later with the *Add part* digitizing tool.
- The button allows you to copy and paste one or more child features within the child layer. They can later be assigned to a different parent feature or have their attributes modified.
- The button deletes the selected airport(s) permanently.
- The symbol opens a new dialog where you can select any existing airport which will then be assigned to the current region. This may be handy if you created the airport on the wrong region by accident.

- The symbol unlinks the selected airport(s) from the current region, leaving them unassigned (the foreign key is set to NULL) effectively.
- With the  button you can zoom the map to the selected child features.
- The two buttons  and  to the right switch between the *table view and form view* of the related child features.

If you use the *Drag and Drop Designer* for the regions feature, you can select which tools are available. You can even decide whether to open a new form when a new feature is added using *Force hide form on add feature* option. Be aware that this option implies that not null attributes must take a valid default value to work correctly.

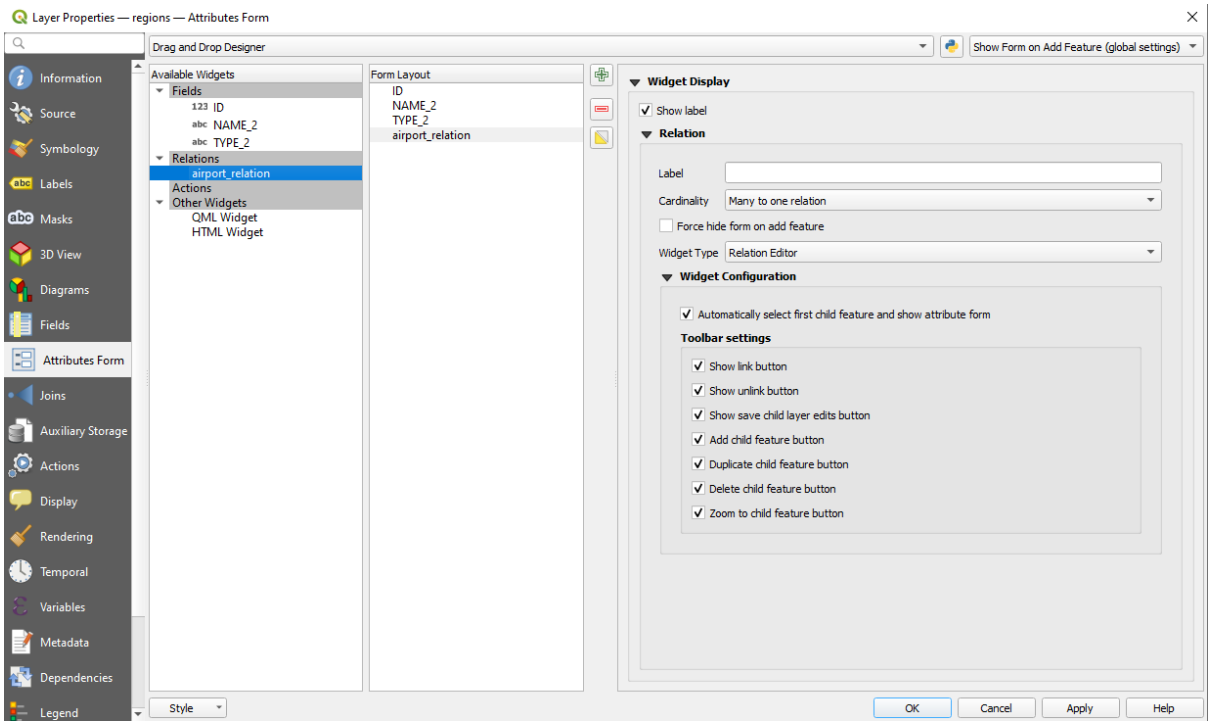


Fig. 12.108: Drag and Drop Designer for configure regions-airports relation tools

In the above example the referencing layer has geometries (so it isn't just an alphanumeric table) so the above steps will create an entry in the layer attribute table that has no corresponding geometric feature. To add the geometry:

1. Choose  *Open Attribute Table* for the referencing layer.
2. Select the record that has been added previously within the feature form of the referenced layer.
3. Use the  Add Part digitizing tool to attach a geometry to the selected attributes table record.

If you work on the airport table, the widget Relation Reference is automatically set up for the `fk_region` field (the one used to create the relation), see *Relation Reference widget*.

In the airport form you will see the  button at the right side of the `fk_region` field: if you click on the button the form of the region layer will be opened. This widget allows you to easily and quickly open the forms of the linked parent features.

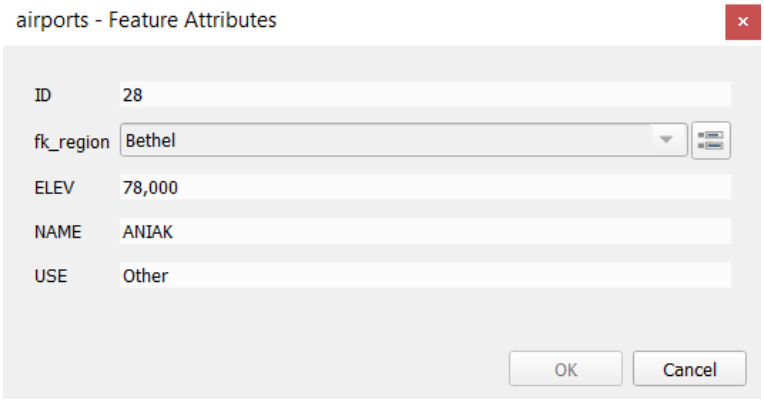
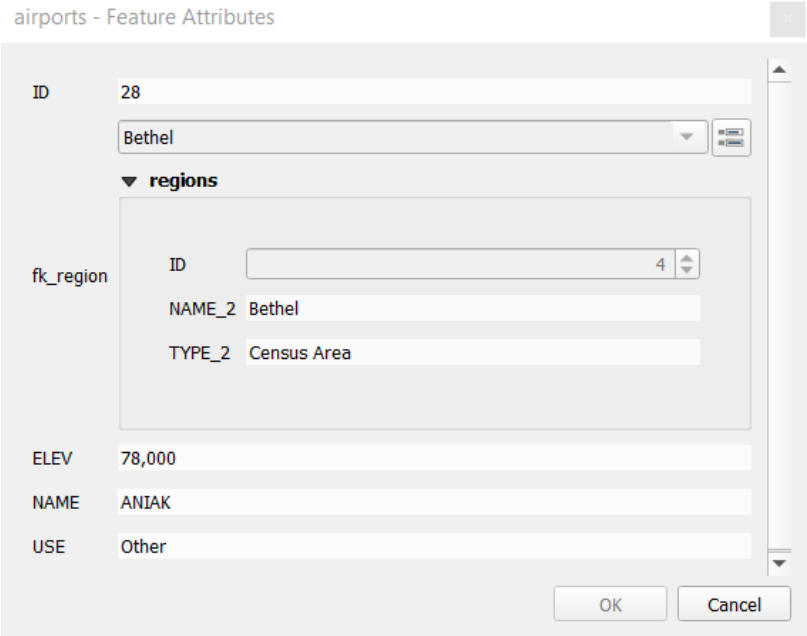


Fig. 12.109: Identification dialog airport with relation to regions

The Relation Reference widget has also an option to embed the form of the parent layer within the child one. It is available in the *Properties* ► *Attributes Form* menu of the airport layer: select the `fk_region` field and check the `Show embedded form` option.

If you look at the feature dialog now, you will see that the form of the region is embedded inside the airports form and will even have a combobox, which allows you to assign the current airport to another region.



Moreover if you toggle the editing mode of the airport layer, the `fk_region` field has also an autocompleter function: while typing you will see all the values of the `id` field of the region layer. Here it is possible to digitize a polygon for the region layer using the button if you chose the option `Allow adding new features` in the *Properties* ► *Attributes Form* menu of the airport layer.

The child layer can also be used in the *Select Features By Value* tool in order to select features of the parent layer based on attributes of their children.

In Fig. 12.110, all the regions where the mean altitude of the airports is greater than 500 meters above sea level are selected.

You will find that many different aggregation functions are available in the form.

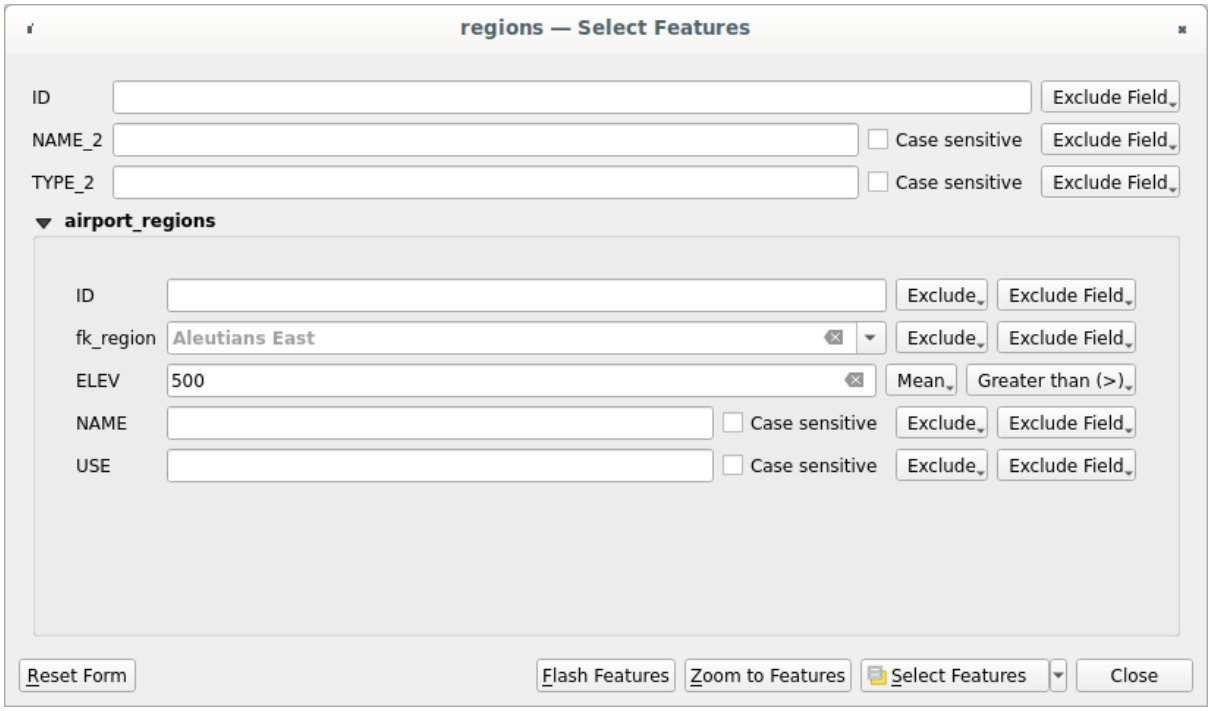


Fig. 12.110: Select parent features with child values

Many-to-many (N-M) relations

N-M relations are many-to-many relations between two tables. For instance, the `airports` and `airlines` layers: an airport receives several airline companies and an airline company flies to several airports.

This SQL code creates the three tables we need for an N-M relationship in a PostgreSQL schema named *locations*. You can run the code using the *Database* ► *DB Manager*… for PostgreSQL or external tools such as pgAdmin. The airports table stores the `airports` layer and the airline table stores the `airlines` layer. In both tables few fields are used for clarity. The *tricky* part is the `airports_airlines` table. We need it to list all airlines for all airports (or vice versa). This kind of table is known as a *pivot table*. The *constraints* in this table force that an airport can be associated with an airline only if both already exist in their layers.

```
CREATE SCHEMA locations;

CREATE TABLE locations.airports
(
   id serial NOT NULL,
   geom geometry(Point, 4326) NOT NULL,
   airport_name text NOT NULL,
   CONSTRAINT airports_pkey PRIMARY KEY (id)
);

CREATE INDEX airports_geom_idx ON locations.airports USING gist (geom);

CREATE TABLE locations.airlines
(
   id serial NOT NULL,
   geom geometry(Point, 4326) NOT NULL,
   airline_name text NOT NULL,
   CONSTRAINT airlines_pkey PRIMARY KEY (id)
);

CREATE INDEX airlines_geom_idx ON locations.airlines USING gist (geom);
```

(continues on next page)

(continued from previous page)

```
CREATE TABLE locations.airports_airlines
(
   id serial NOT NULL,
   airport_fk integer NOT NULL,
   airline_fk integer NOT NULL,
   CONSTRAINT airports_airlines_pkey PRIMARY KEY (id),
   CONSTRAINT airports_airlines_airport_fk_fkey FOREIGN KEY (airport_fk)
      REFERENCES locations.airports (id)
      ON DELETE CASCADE
      ON UPDATE CASCADE
      DEFERRABLE INITIALLY DEFERRED,
   CONSTRAINT airports_airlines_airline_fk_fkey FOREIGN KEY (airline_fk)
      REFERENCES locations.airlines (id)
      ON DELETE CASCADE
      ON UPDATE CASCADE
      DEFERRABLE INITIALLY DEFERRED
 );
```

Instead of PostgreSQL you can also use GeoPackage. In this case, the three tables can be created manually using the *Database* ► *DB Manager*.... In GeoPackage there are no schemas so the *locations* prefix is not needed.

Foreign key constraints in `airports_airlines` table can´t be created using *Table* ► *Create Table*... or *Table* ► *Edit Table*... so they should be created using *Database* ► *SQL Window*.... GeoPackage doesn't support *ADD CONSTRAINT* statements so the `airports_airlines` table should be created in two steps:

1. Set up the table only with the `id` field using *Table* ► *Create Table*...
2. Using *Database* ► *SQL Window*..., type and execute this SQL code:

```
ALTER TABLE airports_airlines
   ADD COLUMN airport_fk INTEGER
   REFERENCES airports (id)
   ON DELETE CASCADE
   ON UPDATE CASCADE
   DEFERRABLE INITIALLY DEFERRED;

ALTER TABLE airports_airlines
   ADD COLUMN airline_fk INTEGER
   REFERENCES airlines (id)
   ON DELETE CASCADE
   ON UPDATE CASCADE
   DEFERRABLE INITIALLY DEFERRED;
```

Then in QGIS, you should set up two *one-to-many relations* as explained above:

- a relation between `airlines` table and the pivot table;
- and a second one between `airports` table and the pivot table.

An easier way to do it (only for PostgreSQL) is using the *Discover Relations* in *Project* ► *Properties* ► *Relations*. QGIS will automatically read all relations in your database and you only have to select the two you need. Remember to load the three tables in the QGIS project first.

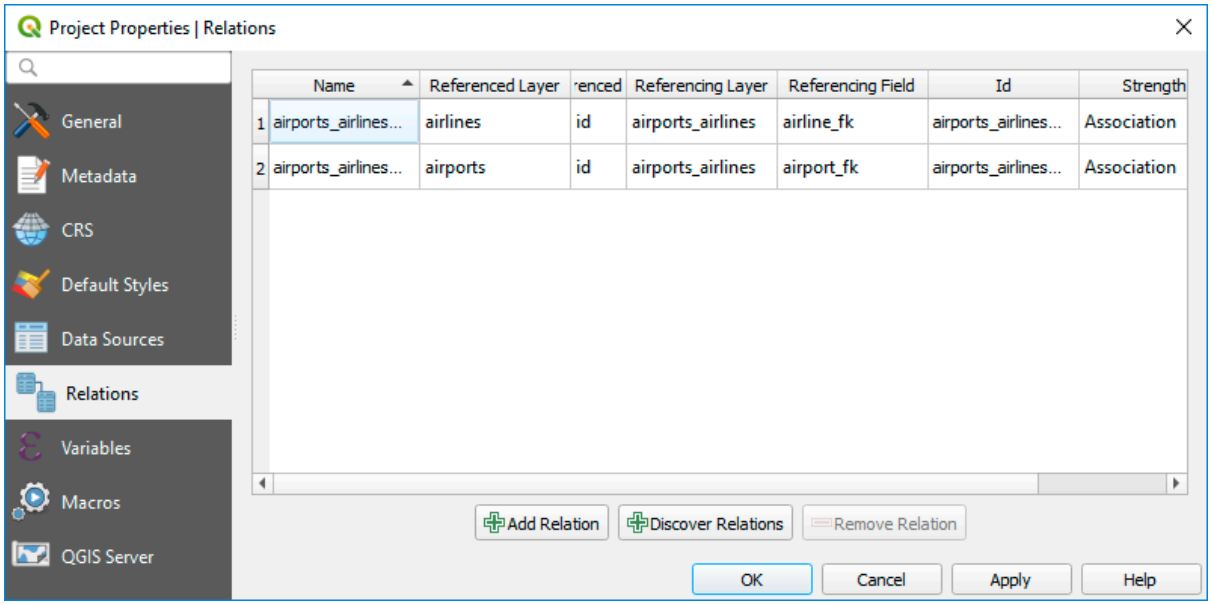


Fig. 12.111: Relations and autodiscover

In case you want to remove an `airport` or an `airline`, QGIS won't remove the associated record(s) in `airports_airlines` table. This task will be made by the database if we specify the right *constraints* in the pivot table creation as in the current example.

Note

Combining N-M relation with automatic transaction group

You should enable the transaction mode in *Project Properties* ► *Data Sources* ► when working on such context. QGIS should be able to add or update row(s) in all tables (airlines, airports and the pivot tables).

Finally we have to select the right cardinality in the *Layer Properties* ► *Attributes Form* for the `airports` and `airlines` layers. For the first one we should choose the **airlines (id)** option and for the second one the **airports (id)** option.

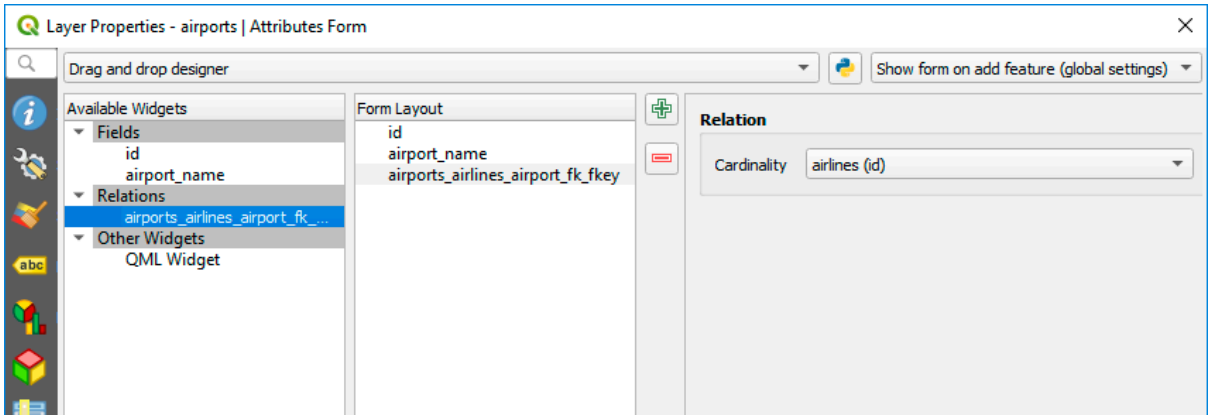


Fig. 12.112: Set relationship cardinality

Now you can associate an airport with an airline (or an airline with an airport) using *Add child feature* or *Link existing child feature* in the subforms. A record will automatically be inserted in the `airports_airlines` table.

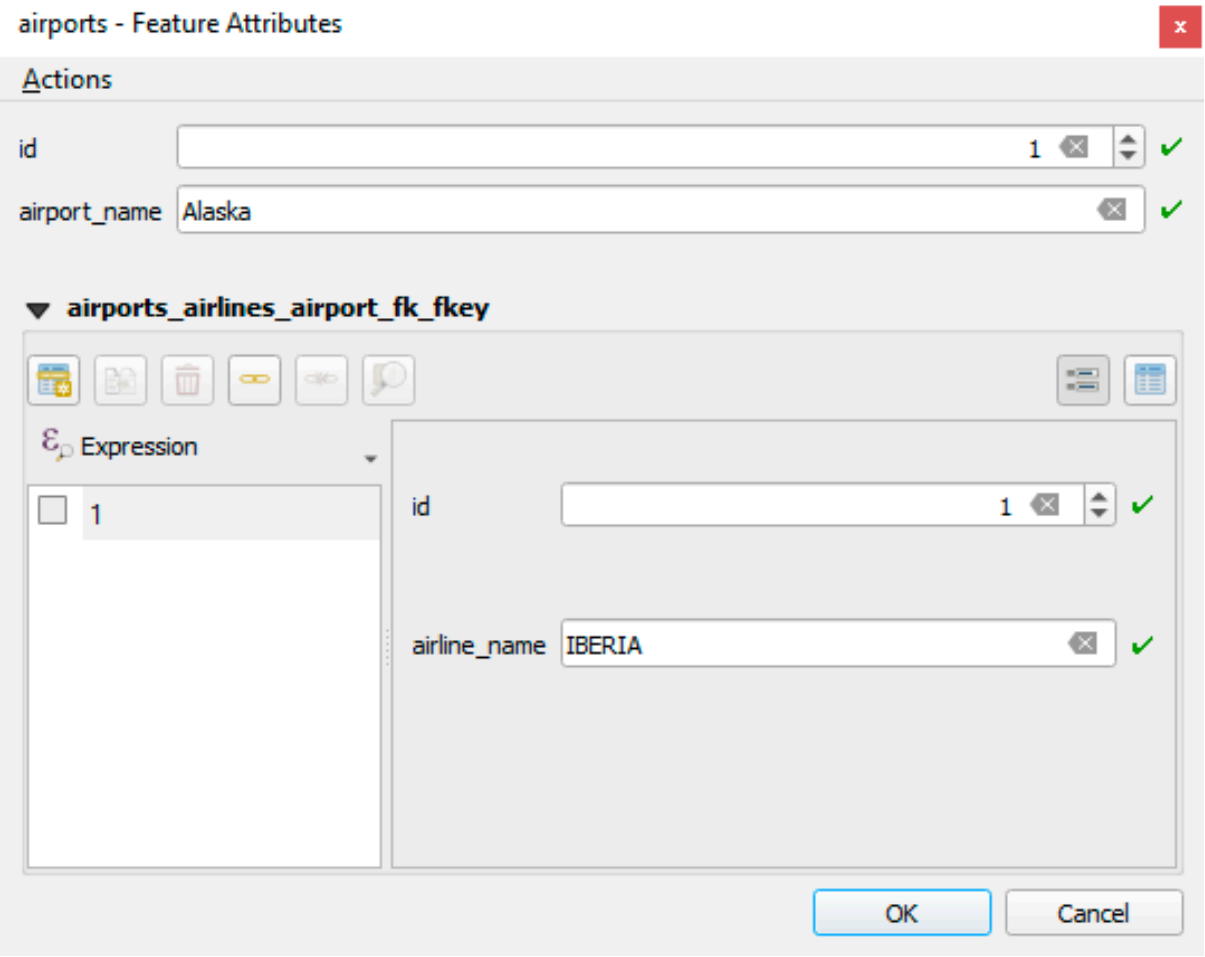


Fig. 12.113: N-M relationship between airports and airlines

Note

Using **Many to one relation** cardinality

Sometimes hiding the pivot table in an N-M relationship is not desirable. Mainly because there are attributes in the relationship that can only have values when a relationship is established. If your tables have a geometry field, it could be interesting to activate the *On map identification* option (*Layer Properties* ► *Attributes Form* ► *Available widgets* ► *Fields*) for the foreign key fields in the pivot table.

Note

Pivot table primary key

Avoid using multiple fields in the primary key in a pivot table. QGIS assumes a single primary key so a constraint like `constraint airports_airlines_pkey primary key (airport_fk, airline_fk)` will not work.

Polymorphic relations

The purpose

Polymorphic relations are special case of 1-N relations, where a single referencing (document) layer contains the features for multiple referenced layers. This differs from normal relations which require different referencing layer for each referenced layer. A single referencing (document) layer is achieved by adding an adiditonal `layer_field` column in the referencing (document) layer that stores information to identify the referenced layer. In its most simple form, the referencing (document) layer will just insert the layer name of the referenced layer into this field.

To be more precise, a polymorphic relation is a set of normal relations having the same referencing layer but having the referenced layer dynamically defined. The polymorphic setting of the layer is solved by using an expression which has to match some properties of the referenced layer like the table name, layer id, layer name.

Imagine we are going to the park and want to take pictures of different species of `plants` and `animals` we see there. Each plant or animal has multiple pictures associated with it, so if we use the normal 1:N relations to store pictures, we would need two separate tables, `animal_images` and `plant_images`. This might not be a problem for 2 tables, but imagine if we want to take separate pictures for mushrooms, birds etc.

Polymorphic relations solve this problem as all the referencing features are stored in the same table `documents`. For each feature the referenced layer is stored in the `referenced_layer` field and the referenced feature id in the `referenced_id` field.

Defining polymorphic relations

First, let QGIS know about the polymorphic relations between the layers. This is done in *Project* ► *Properties…*.

Open the *Relations* tab and click on the little down arrow next to the  *Add Relation* button, so you can select the *Add Polymorphic Relation* option from the newly appeared dropdown.

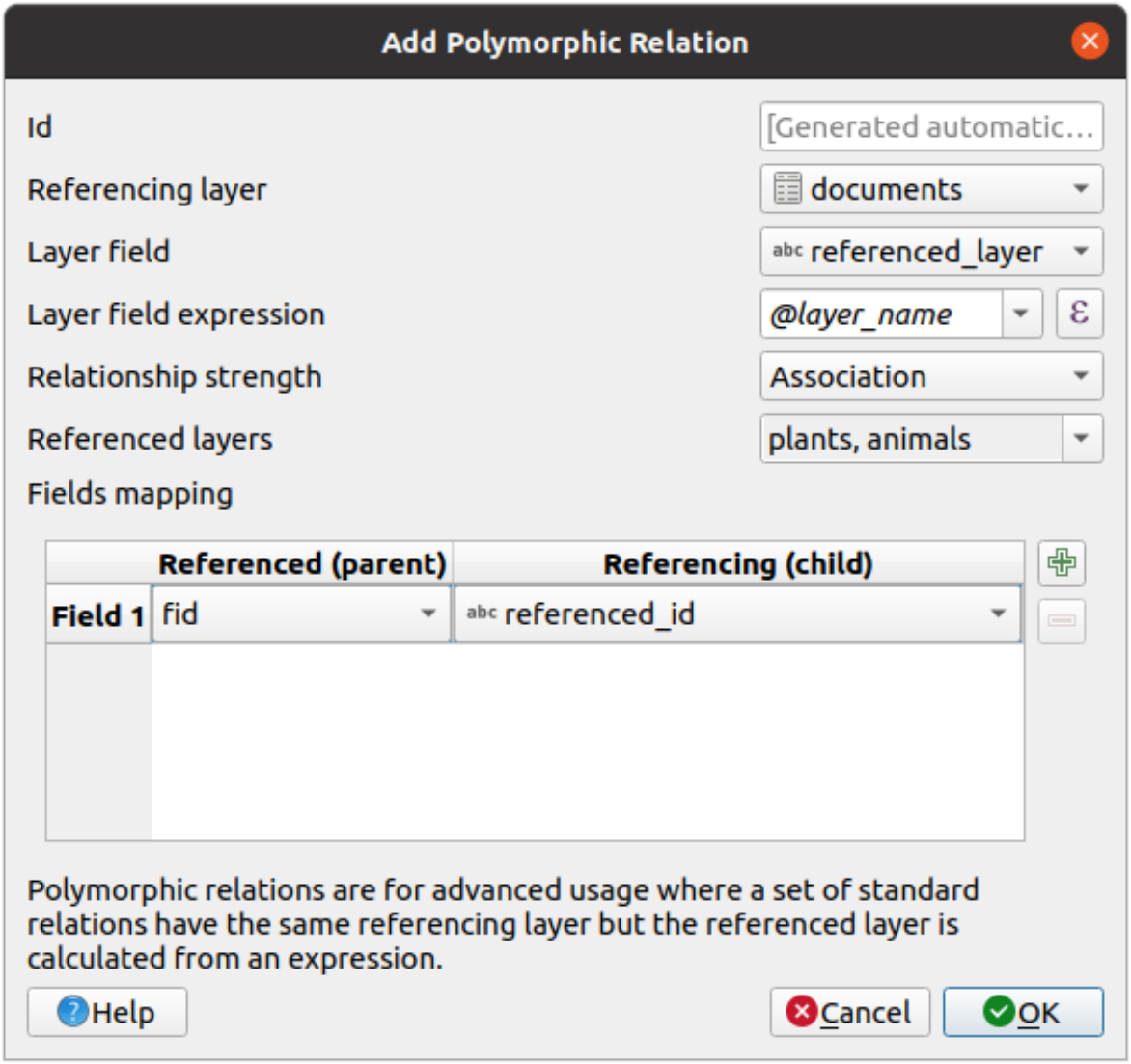


Fig. 12.114: Adding a polymorphic relation using `documents` layer as referencing and `animals` and `plants` as referenced layers.

- **Id** will be used for internal purposes and has to be unique. You may need it to build *custom forms*. If you leave it empty, one will be generated for you but you can assign one yourself to get one that is easier to handle
- **Referencing Layer (Child)** also considered as child layer, is the one with the foreign key field on it. In our case, this is the `documents` layer. For this layer you need to add a referencing field which points to the other layer, so this is `referenced_id`.

> **Note**
>
> Sometimes, you need more than a single field to uniquely identify features in a layer. Creating a relation with such a layer requires a **composite key**, i.e. more than a single pair of matching fields. Use the Add new field pair as part of a composite foreign key button to add as many pairs as necessary.

- **Layer Field** is the field in the referencing table that stores the result of the evaluated layer expression which is the referencing table that this feature belongs to. In our example, this would be the `referenced_layer` field.
- **Layer expression** evaluates to a unique identifier of the layer. This can be the layer name `@layer_name`,

the layer id `@layer_id`, the layer's table name `decode_uri(@layer, 'table')` or anything that can uniquely identifies a layer.

- **Relationship strength** sets the strength of the generated relations between the parent and the child layer. The default *Association* type means that the parent layer is *simply* linked to the child one while the *Composition* type allows you to duplicate also the child features when duplicating the parent ones and on deleting a feature the children are deleted as well, resulting in cascade over all levels (means children of children of… are deleted as well).
- **Referenced Layers** also considered as parent layers, are those with the primary key, pointed to, so here they would be `plants` and `animals` layers. You need to define the primary key of the referenced layers from the dropdown, so it is `fid`. Note that the definition of a valid primary key requires all the referenced layers to have a field with that name. If there is no such field you cannot save a polymorphic relation.

Once added, the polymorphic relation can be edited via the *Edit Polymorphic Relation* menu entry.

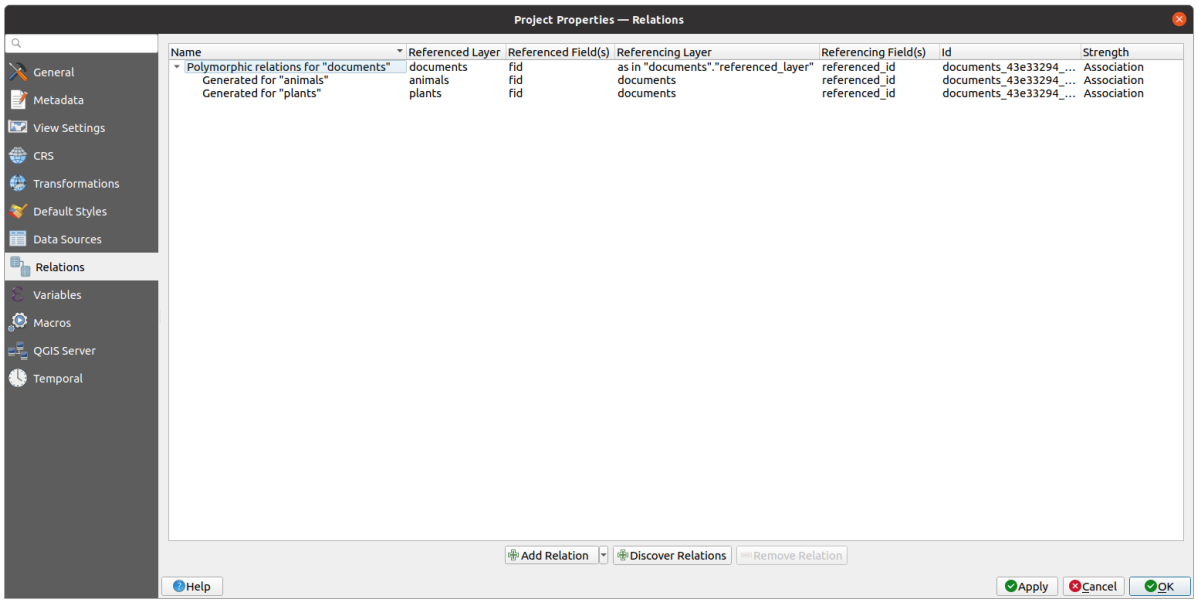


Fig. 12.115: Preview of the newly created polymorphic relation and its child relations for animals and plants.

The example above uses the following database schema:

```
CREATE SCHEMA park;

CREATE TABLE park.animals
(
   fid serial NOT NULL,
   geom geometry(Point, 4326) NOT NULL,
   animal_species text NOT NULL,
   CONSTRAINT animals_pkey PRIMARY KEY (fid)
);

CREATE INDEX animals_geom_idx ON park.animals USING gist (geom);

CREATE TABLE park.plants
(
   fid serial NOT NULL,
   geom geometry(Point, 4326) NOT NULL,
   plant_species text NOT NULL,
   CONSTRAINT plants_pkey PRIMARY KEY (fid)
);
```

(continues on next page)

(continued from previous page)

```
CREATE INDEX plants_geom_idx ON park.plants USING gist (geom);

CREATE TABLE park.documents
(
   fid serial NOT NULL,
   referenced_layer text NOT NULL,
   referenced_id integer NOT NULL,
   image_filename text NOT NULL,
   CONSTRAINT documents_pkey PRIMARY KEY (fid)
);
```

CHAPTER

THIRTEEN

WORKING WITH RASTER DATA

13.1 Raster Properties Dialog

Raster data is made up of pixels (or cells), and each pixel has a value. It is commonly used to store various types of data, including:

- Imagery, such as satellite images, digital aerial photographs, scanned maps
- Elevation data, such as digital elevation models (DEMs), digital terrain models (DTMs)
- Other types of data, such as land cover, soil types, rainfall and many others.

Raster data can be stored in several supported formats, including GeoTIFF, ERDAS Imagine, ArcInfo ASCII GRID, PostgreSQL Raster and others. See more at *Opening Data*.

To view and set the properties for a raster layer, double click on the layer name in the map legend, or right click on the layer name and choose *Properties* from the context menu. This will open the *Raster Layer Properties* dialog.

There are several tabs in the dialog:

Information	*Source*	*Symbology*[1]
Transparency[1]	*Labels*[1]	*Histogram*[1]
Rendering	*Temporal*	*Pyramids*
Elevation	*Metadata*	*Legend*
Display	*QGIS Server*	*External plugins*[2] tabs

[1] Also available in the *Layer styling panel*

[2] *External plugins* you install can optionally add tabs to this dialog. Those are not presented in this document. Refer to their documentation.

Tip

Live update rendering

The *Layer Styling Panel* provides you with some of the common features of the Layer properties dialog and is a good modeless widget that you can use to speed up the configuration of the layer styles and view your changes on the map canvas.

> **Note**
>
> Because properties (symbology, label, actions, default values, forms…) of embedded layers (see *Embedding layers from external projects*) are pulled from the original project file, and to avoid changes that may break this behavior, the layer properties dialog is made unavailable for these layers.

13.1.1 Information Properties

The *Information* tab is read-only and represents an interesting place to quickly grab summarized information and metadata for the current layer. Provided information are:

- general such as name in the project, source path, list of auxiliary files, last save time and size, the used provider
- custom properties, used to store in the active project additional information about the layer. Default custom properties include *Identify/format*, which influences how the results from using the *Identify features* tool over a raster layer are formatted. More properties can be created and managed using PyQGIS, specifically through the setCustomProperty() method.
- based on the provider of the layer: extent, width and height, data type, GDAL driver, bands statistics
- the Coordinate Reference System: name, units, method, accuracy, reference (i.e. whether it's static or dynamic)
- read from layer properties: data type, extent, width/height, compression, pixel size, statistics on bands, number of columns, rows and no-data values of the raster…
- picked from the *filled metadata*: access, extents, links, contacts, history…

13.1.2 Source Properties

The *Source* tab displays basic information about the selected raster, including:

- the *Layer name* to display in the *Layers Panel*;
- the *Coordinate Reference System*: Displays the layer's *Coordinate Reference System (CRS)*. You can change the layer's CRS, by selecting a recently used one in the drop-down list or clicking on the Select CRS button (see *Coordinate Reference System Selector*). Use this process only if the layer CRS is a wrong or not specified. If you wish to reproject your data, use a reprojection algorithm from Processing or *Save it as new dataset*.

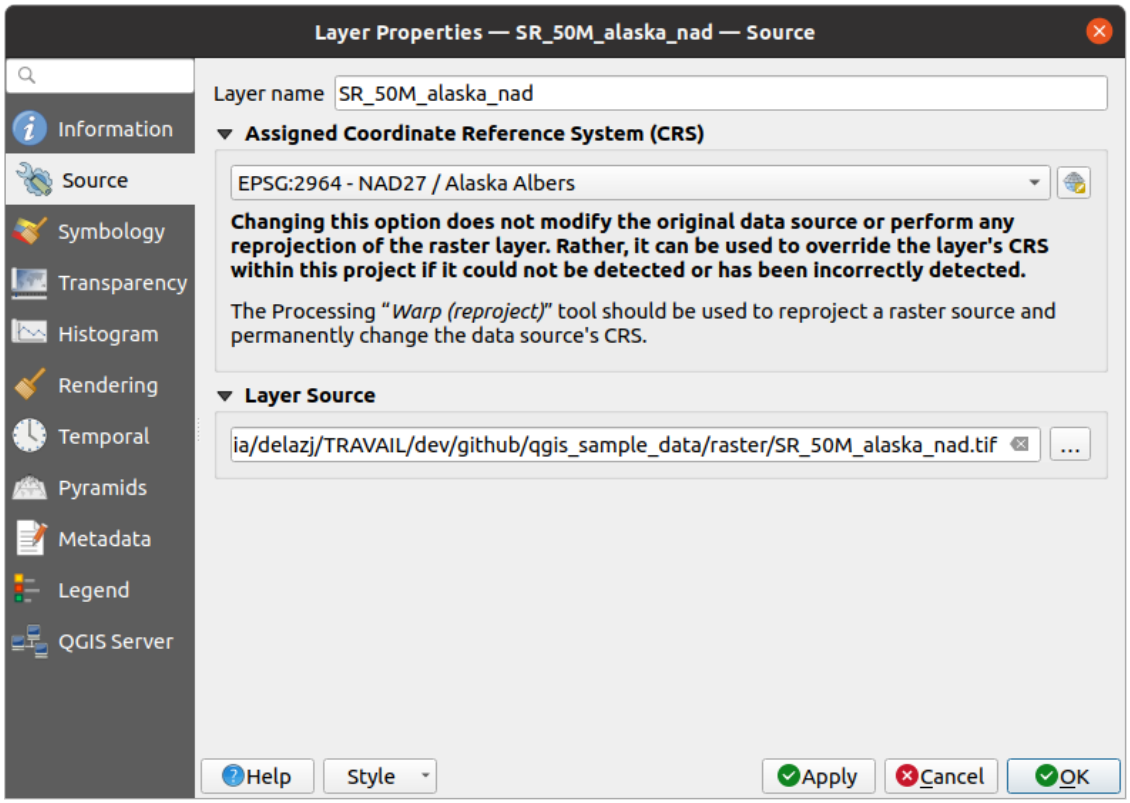


Fig. 13.1: Raster Layer Properties - Source Dialog

13.1.3 Symbology Properties

The raster layer symbology tab is made of three different sections:

- The *Band rendering* where you can control the renderer type to use
- The *Layer rendering* to apply effects on rendered data
- The *Resampling* methods to optimize rendering on map

Band rendering

QGIS offers many different *Render types*. The choice of renderer depends on the data type and the information you'd like to highlight.

1. *Multiband color* - if the file comes with several bands (e.g. a satellite image with several bands).
2. *Paletted/Unique values* - for single band files that come with an indexed palette (e.g. a digital topographic map) or for general use of palettes for rendering raster layers.
3. *Singleband gray* - (one band of) the image will be rendered as gray. QGIS will choose this renderer if the file is neither multiband nor paletted (e.g. a shaded relief map).
4. *Singleband pseudocolor* - this renderer can be used for files with a continuous palette or color map (e.g. an elevation map).
5. *Single color* - the raster layer will be rendered with a single color.
6. *Hillshade* - Creates hillshade from a band.
7. *Contours* - Generates contours on the fly for a source raster band.

Multiband color

With the multiband color renderer, three selected bands from the image will be used as the red, green or blue component of the color image. QGIS automatically fetches *Min* and *Max* values for each band of the raster and scales the coloring accordingly. You can control the value ranges in the *Min/Max Value Settings* section.

A *Contrast enhancement* method can be applied to the values: 'No enhancement', 'Stretch to MinMax', 'Stretch and clip to MinMax' and 'Clip to min max'.

Note

Contrast enhancement

When adding GRASS rasters, the option *Contrast enhancement* will always be set automatically to *stretch to min max*, even if this is set to another value in the QGIS general options.

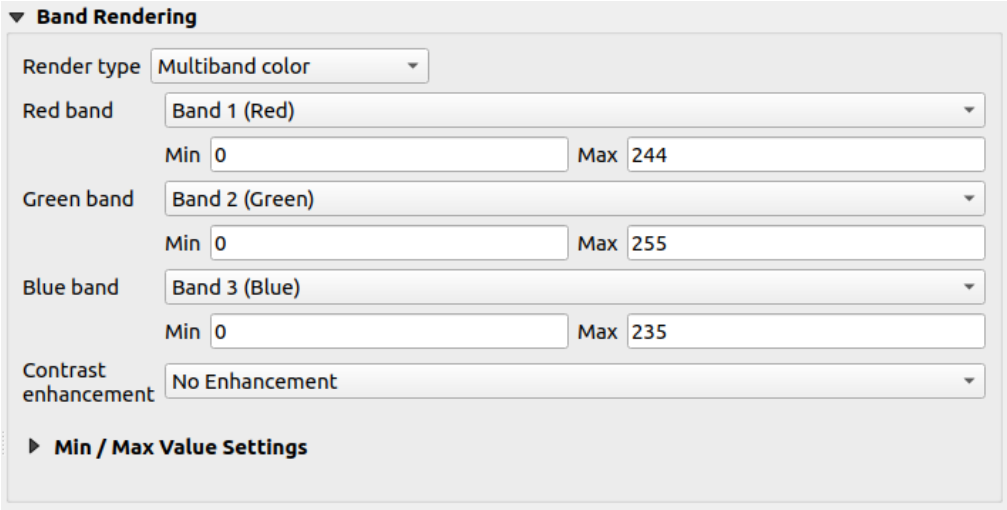


Fig. 13.2: Raster Symbology - Multiband color rendering

Tip

Viewing a Single Band of a Multiband Raster

If you want to view a single band of a multiband image (for example, Red), you might think you would set the Green and Blue bands to *Not Set*. But the preferred way of doing this is to set the image type to *Singleband gray*, and then select Red as the *Gray band* to use.

Paletted/Unique values

This is the standard render option for singleband files that include a color table, where a certain color is assigned to each pixel value. In that case, the palette is rendered automatically.

It can be used for all kinds of raster bands, assigning a color to each unique raster value.

If you want to change a color, just double-click on the color and the *Select color* dialog appears.

It is also possible to assign labels to the colors. The label will then appear in the legend of the raster layer.

Right-clicking over selected rows in the color table shows a contextual menu to:

- *Change Color…* for the selection

- *Change Opacity…* for the selection
- *Change Label…* for the selection

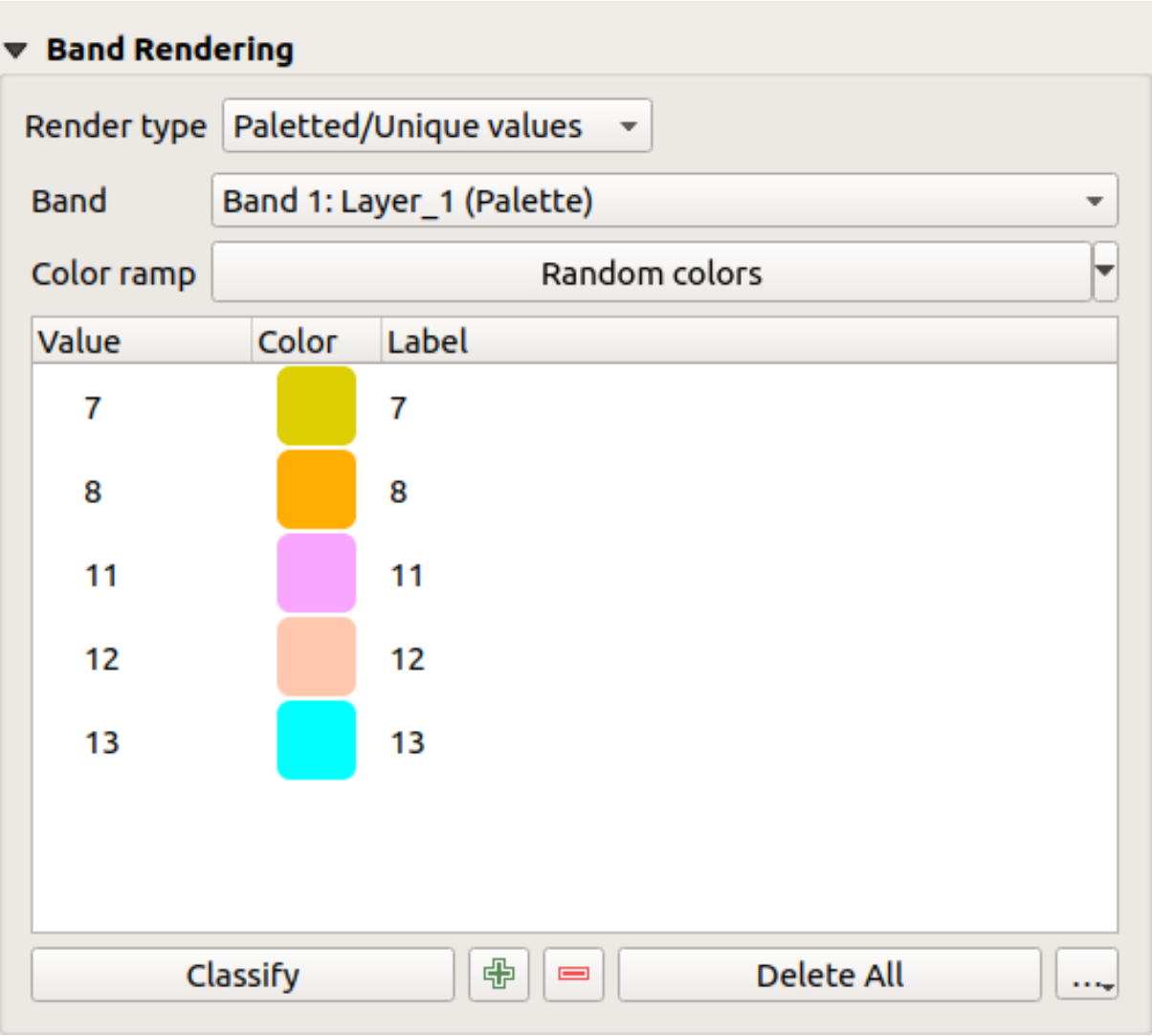


Fig. 13.3: Raster Symbology - Paletted unique value rendering

The pulldown menu, that opens when clicking the … (Advanced options) button below the color map to the right, offers color map loading (*Load Color Map from File…*) and exporting (*Export Color Map to File…*), and loading of classes (*Load Classes from Layer*).

Singleband gray

This renderer allows you to render a layer using only one band with a *Color gradient*: 'Black to white' or 'White to black'. You can change the range of values to color (*Min* and *Max*) in the *Min/Max Value Settings*.

A *Contrast enhancement* method can be applied to the values: 'No enhancement', 'Stretch to MinMax', 'Stretch and clip to MinMax' and 'Clip to min max'.

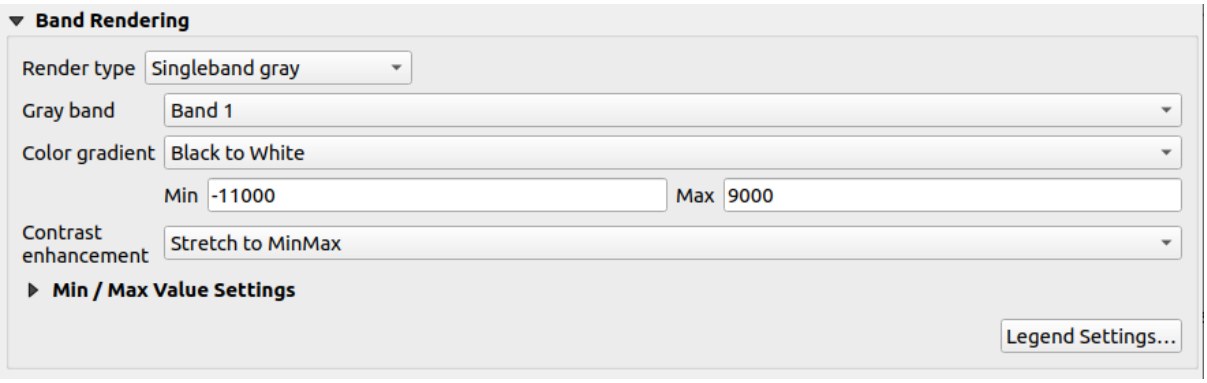


Fig. 13.4: Raster Symbology - Singleband gray rendering

Pixels are assigned a color based on the selected color gradient and the layer's legend (in the *Layers* panel and the layout *legend item*) is displayed using a continuous color ramp. Press *Legend settings...* if you wish to tweak the settings. More details at *Customize raster legend*.

Singleband pseudocolor

This is a render option for single-band files that include a continuous palette. You can also create color maps for a band of a multiband raster.

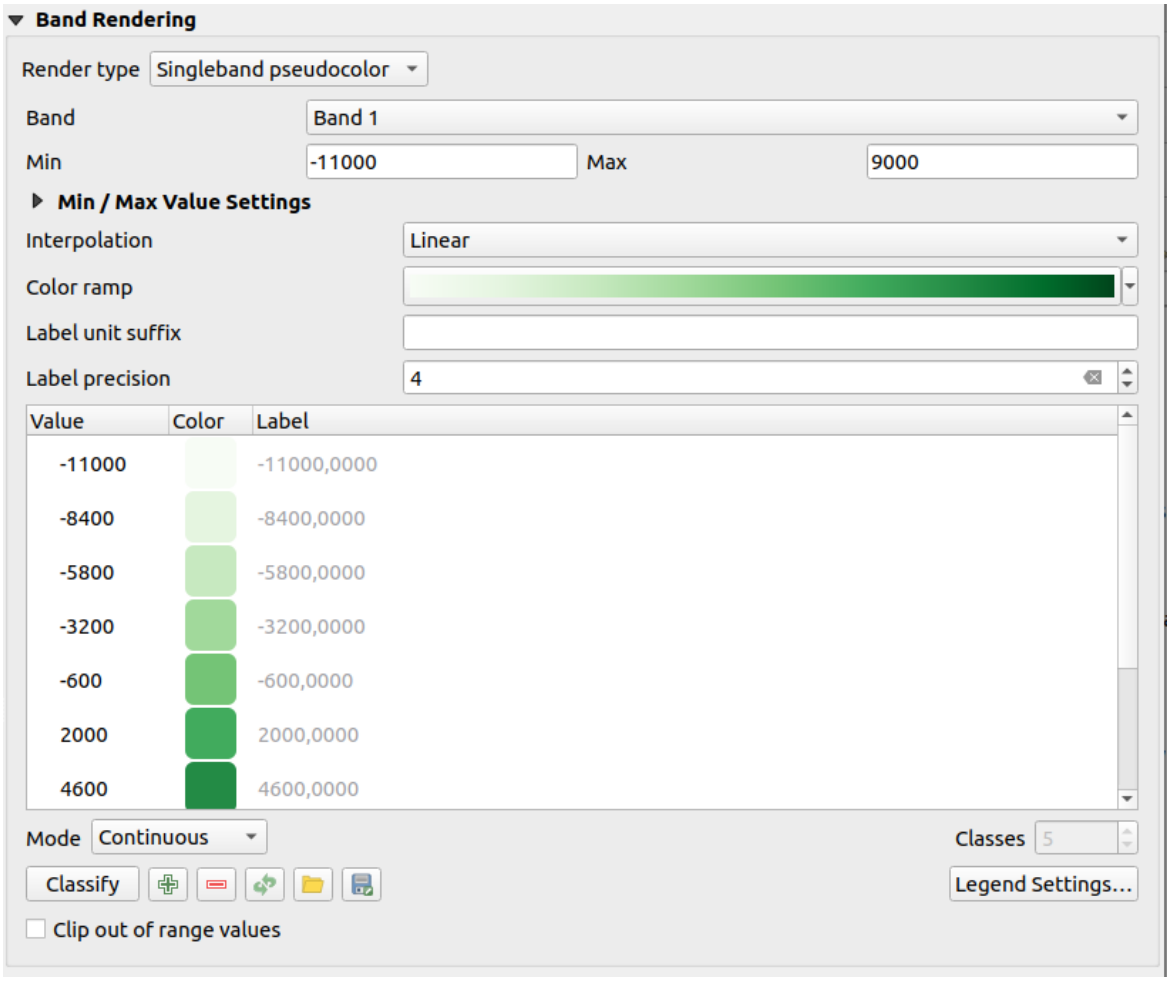


Fig. 13.5: Raster Symbology - Singleband pseudocolor rendering

Using a *Band* of the layer and a *values range*, you can now interpolate and assign representation color to pixels within classes. More at *Color ramp shader classification*.

Pixels are assigned a color based on the selected color ramp and the layer's legend (in the *Layers* panel and the layout *legend item*) is displayed using a continuous color ramp. Press *Legend settings…* if you wish to tweak the settings or instead use a legend with separated classes (and colors). More details at *Customize raster legend*.

Single color

This renderer allows you to render a raster layer using *Single color*. This type of renderer is useful when you want to display a raster layer uniformly, without any variation in color based on pixel values.

The single color renderer can be used with both single-band and multiband raster layers. When used with multiband rasters, you can select which band to apply the single color to, effectively displaying that specific band uniformly across the entire layer.

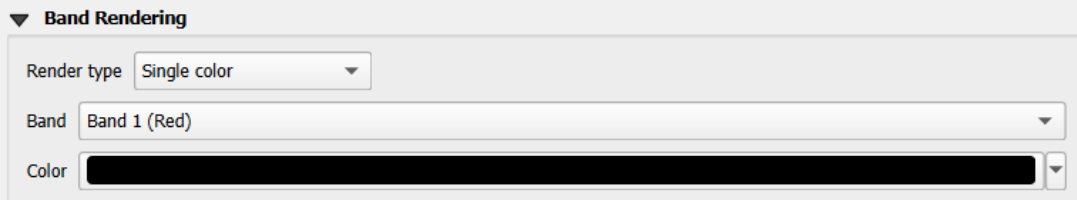


Fig. 13.6: Raster Symbology - Single color rendering

Hillshade

Render a band of the raster layer using hillshading.

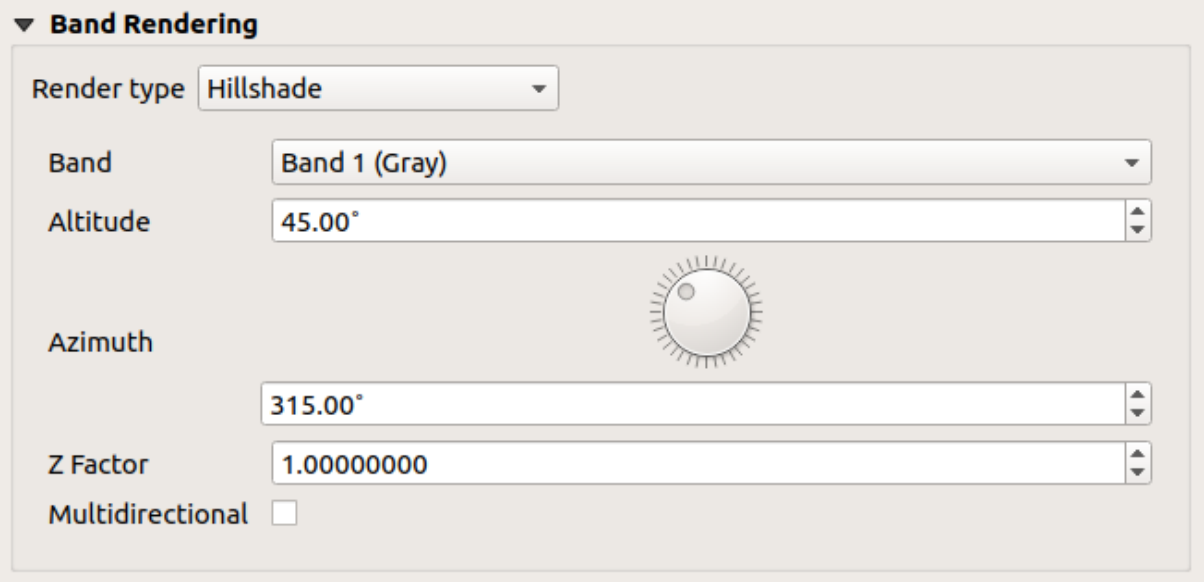


Fig. 13.7: Raster Symbology - Hillshade rendering

Options:

- *Band*: The raster band to use.
- *Altitude*: The elevation angle of the light source (default is `45°`).

- *Azimuth*: The azimuth of the light source (default is `315°`).
- *Z Factor*: Scaling factor for the values of the raster band (default is `1`).
- *Multidirectional*: Specify if multidirectional hillshading is to be used (default is `off`).

Contours

This renderer draws contour lines that are calculated on the fly from the source raster band.

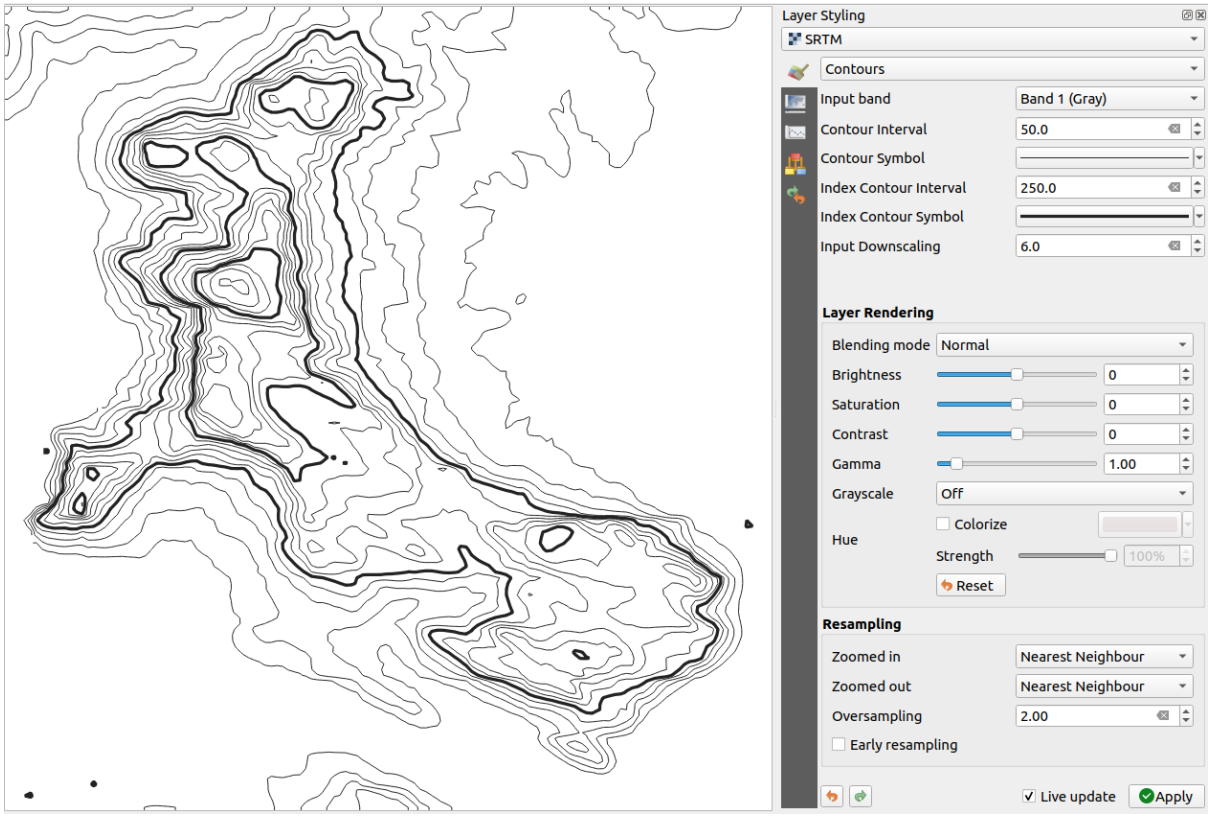


Fig. 13.8: Raster Symbology - Contours rendering

Options:

- *Input band*: the raster band to use.
- *Contour interval*: the distance between two consecutive contour lines
- *Contour symbol*: the *symbol* to apply to the common contour lines.
- *Index contour interval*: the distance between two consecutive **index contours**, that is the lines shown in a distinctive manner for ease of identification, being commonly printed more heavily than other contour lines and generally labeled with a value along its course.
- *Index contour symbol*: the symbol to apply to the index contour lines
- *Input downscaling*: Indicates by how much the renderer will scale down the request to the data provider (default is `4.0`).

  For example, if you generate contour lines on input raster block with the same size as the output raster block, the generated lines would contain too much detail. This detail can be reduced by the "downscale" factor, requesting lower resolution of the source raster. For a raster block 1000x500 with downscale 10, the renderer will request raster 100x50 from provider. Higher downscale makes contour lines more simplified (at the expense of losing some detail).

Setting the min and max values

By default, QGIS reports the *Min* and *Max* values of the band(s) of the raster. A few very low and/or high values can have a negative impact on the rendering of the raster. The *Min/Max Value Settings* frame helps you control the rendering.

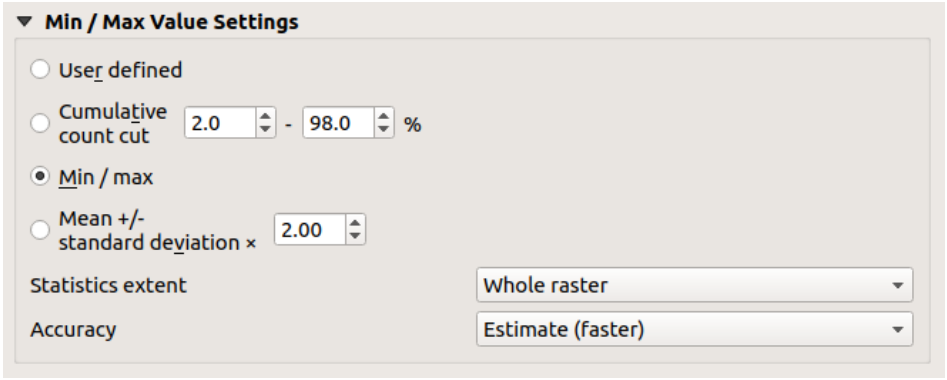


Fig. 13.9: Raster Symbology - Min and Max Value Settings

Available options are:

- *User defined*: The default *Min* and *Max* values of the band(s) can be overridden
- *Cumulative count cut*: Removes outliers. The standard range of values is `2%` to `98%`, but it can be adapted manually.
- *Min / max*: Uses the whole range of values in the image band.
- *Mean +/- standard deviation x*: Creates a color table that only considers values within the standard deviation or within multiple standard deviations. This is useful when you have one or two cells with abnormally high values in a raster layer that impact the rendering of the raster negatively.

Calculations of the min and max values of the bands are made based on the:

- *Statistics extent*: it can be *Whole raster*, *Current canvas* or *Updated canvas*. *Updated canvas* means that min/max values used for the rendering will change with the canvas extent (dynamic stretching).
- *Accuracy*, which can be either *Estimate (faster)* or *Actual (slower)*.

> **Note**
>
> For some settings, you may need to press the *Apply* button of the layer properties dialog in order to display the actual min and max values in the widgets.

Color ramp shader classification

This method can be used to classify and represent scalar dataset (raster or mesh contour) based on their values. Given a *color ramp* and a number of classes, it generates intermediate color map entries for class limits. Each color is mapped with a value interpolated from a range of values and according to a classification mode. The scalar dataset elements are then assigned their color based on their class.

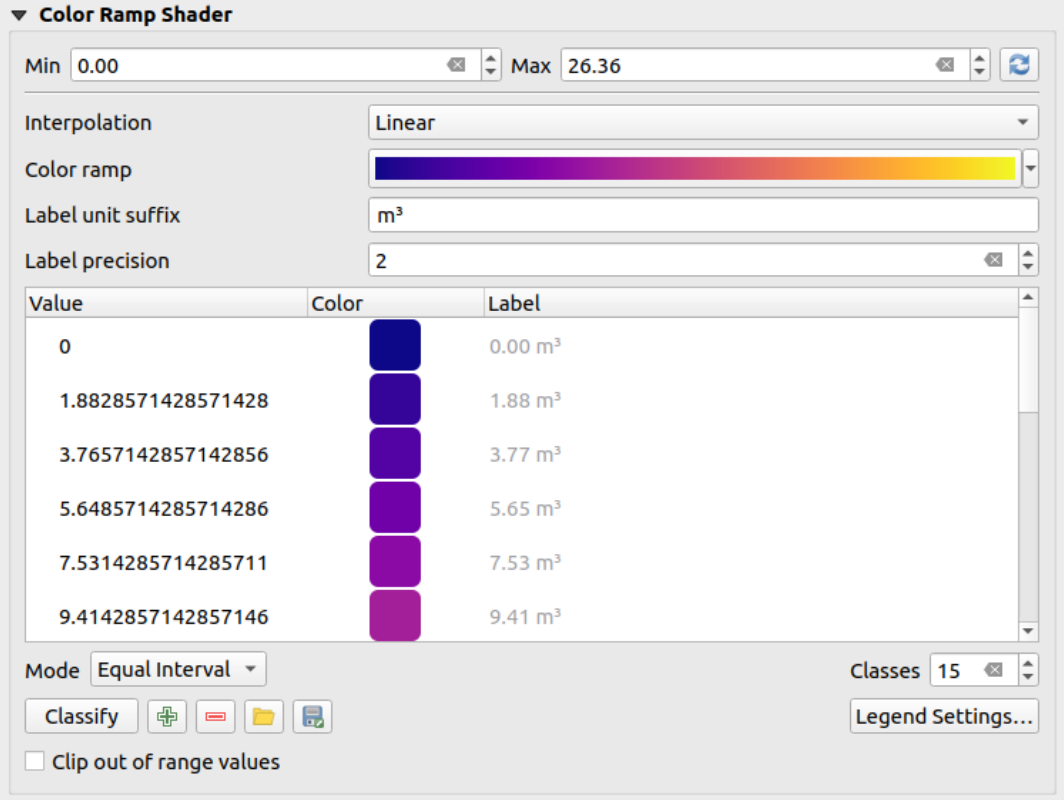


Fig. 13.10: Classifying a dataset with a color ramp shader

1. A *Min* and *Max* values must be defined and used to interpolate classes bounds. By default QGIS detects them from the dataset but they can be modified.
2. The *Interpolation* entry defines how scalar elements are assigned their color :
   - *Discrete* (a <= symbol appears in the header of the *Value* column): The color is taken from the closest color map entry with equal or higher value
   - *Linear*: The color is linearly interpolated from the color map entries above and below the pixel value, meaning that to each dataset value corresponds a unique color
   - *Exact* (a = symbol appears in the header of the *Value* column): Only pixels with value equal to a color map entry are applied a color; others are not rendered.
3. The *Color ramp* widget helps you select the color ramp to assign to the dataset. As usual with *this widget*, you can create a new one and edit or save the currently selected one. The name of the color ramp will be saved in the configuration.
4. The *Label unit suffix* adds a label after the value in the legend, and the *Label precision* controls the number of decimals to display.
5. The classification *Mode* helps you define how values are distributed across the classes:
   - *Equal interval*: Provided the *Number of classes*, limits values are defined so that the classes all have the same magnitude.
   - *Continuous*: Classes number and color are fetched from the color ramp stops; limits values are set following stops distribution in the color ramp.
   - *Quantile*: Provided the *Number of classes*, limits values are defined so that the classes have the same number of elements. Not available with *mesh layers*.
6. You can then *Classify* or tweak the classes:
   - The button Add values manually adds a value to the table.

- The button Remove selected row deletes selected values from the table.
- Double clicking in the *Value* column lets you modify the class value.
- Double clicking in the *Color* column opens the dialog *Change color*, where you can select a color to apply for that value.
- Double clicking in the *Label* column to modify the label of the class, but this value won't be displayed when you use the identify feature tool.
- Right-clicking over selected rows in the color table shows a contextual menu to *Change Color…* and *Change Opacity…* for the selection.

You can use the buttons Load color map from file or Export color map to file to load an existing color table or to save the color table for later use.

7. With linear *Interpolation*, you can also configure:
   - *Clip out of range values*: By default, the linear method assigns the first class (respectively the last class) color to values in the dataset that are lower than the set *Min* (respectively greater than the set *Max*) value. Check this setting if you do not want to render those values.
   - *Legend settings*, for display in the *Layers* panel and the layout *legend item*. More details at *Customize raster legend*.

Customize raster legend

When applying a color ramp to a raster or a mesh layer, you may want to display a legend showing the classification. By default, QGIS displays a continuous color ramp with min and max values in the *Layers* panel and the layout *legend item*. This can be customized using the *Legend settings* button in the classification widget.

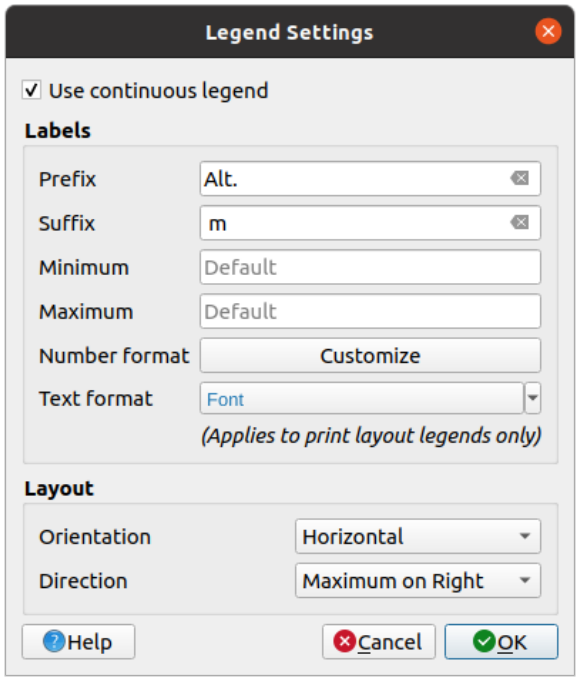


Fig. 13.11: Modifying a raster legend

In this dialog, you can set whether to *Use continuous legend*: if unchecked, the legend displays separated colors corresponding to the different classes applied. This option is not available for raster *singleband gray* symbology.

Checking the *Use continuous legend* allows you to configure both the labels and layout properties of the legend.

Labels

- Add a *Prefix* and a *Suffix* to the labels
- Modify the *Minimum* and a *Maximum* values to show in the legend
- *Customize* the *Number format*
- *Customize* the *Text format* to use in the print layout legend.

Layout

- Control the *Orientation* of the legend color ramp; it can be **Vertical** or **Horizontal**
- Control the *Direction* of the values depending on the orientation:
  - If vertical, you can display the **Maximum on top** or the **Minimum on top**
  - If horizontal, you can display the **Maximum on right** or the **Minimum on right**

Layer rendering

Over the symbology type applied to the layer band(s), you can achieve special rendering effects for the whole raster file(s):

- Use one of the blending modes (see *Blending Modes*)
- Set custom *Brightness*, *Saturation*, *Gamma* and *Contrast* to colors.
- With the *Invert colors*, the layer is rendered with opposite colors. Handy, for example, to switch out-of-the box OpenStreetMap tiles to dark mode.
- Turn the layer to *Grayscale* option either 'By lightness', 'By luminosity' or 'By average'.
- *Colorize* and adjust the *Strength* of *Hue* in the color table

Press *Reset* to remove any custom changes to the layer rendering.

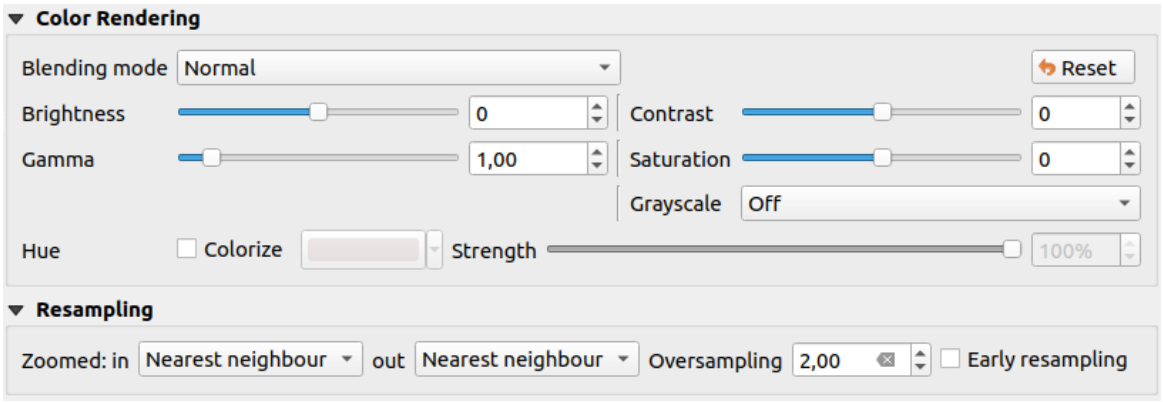


Fig. 13.12: Raster Symbology - Layer rendering and Resampling settings

Resampling

The *Resampling* option has effect when you zoom in and out of an image. Resampling modes can optimize the appearance of the map. They calculate a new gray value matrix through a geometric transformation.

When applying the 'Nearest neighbour' method, the map can get a pixelated structure when zooming in. This appearance can be improved by using the 'Bilinear (2x2 kernel)' or 'Cubic (4x4 kernel)' method, which cause sharp edges to be blurred. The effect is a smoother image. This method can be applied to for instance digital topographic raster maps.

Early resampling: allows to calculate the raster rendering at the provider level where the resolution of the source is known, and ensures a better zoom in rendering with QGIS custom styling. Really convenient for tile rasters loaded using an *interpretation method*.

13.1.4 Transparency Properties

QGIS provides capabilities to set the *Transparency* level of a raster layer.

Use the *Global opacity* slider to set to what extent the underlying layers (if any) should be visible through the current raster layer. This is very useful if you overlay raster layers (e.g., a shaded relief map overlayed by a classified raster map). This will make the look of the map more three dimensional. The opacity of the raster can be data-defined, and vary e.g. depending on the visibility of another layer, by temporal variables, on different pages of an atlas, …

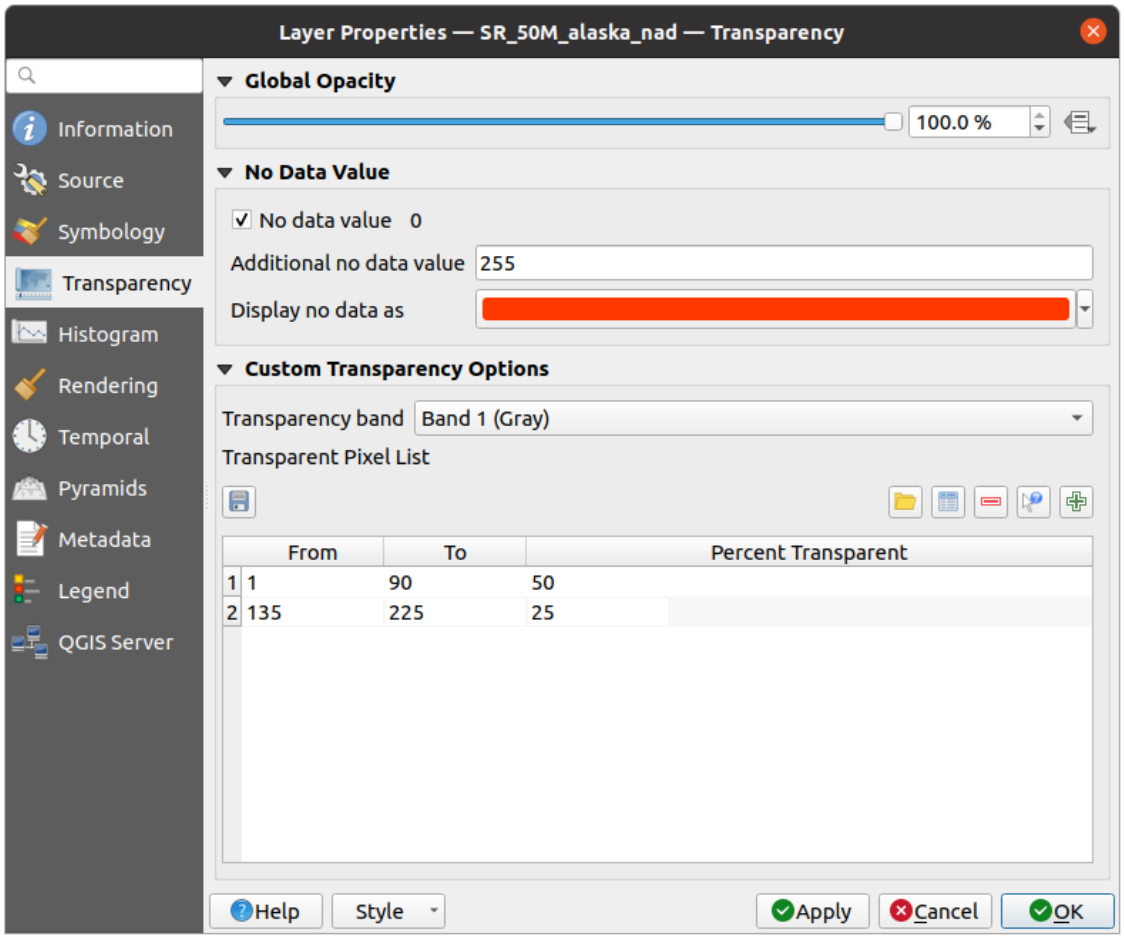


Fig. 13.13: Raster Transparency

With *No data value* QGIS reports the original source no data value (if defined) which you can consider as is in the rendering. Additionally, you can enter a raster value that should be treated as an *Additional no data value*. The

Display no data as color selector allows you to apply a custom color to no data pixels, instead of the default transparent rendering.

An even more flexible way to customize the transparency is available in the *Custom transparency options* section:

- Use *Transparency band* to apply transparency for an entire band.
- Provide a list of pixels to make transparent with corresponding levels of transparency:

  1. Click the  Add values manually button. A new row will appear in the pixel list.
  2. For single-band based symbology (e.g. DEMs), enter the **From** and **To** values and adjust the **Percent Transparent** to apply.
  3. For multiband based symbology (e.g. RGB images) enter the **Red**, **Green** and **Blue** values of the pixel and adjust the **Percent Transparent** to apply. QGIS supports **Tolerance** for pixel values, when defining transparency. This means that pixels with color close to the specified RGB values can also be made transparent. Note that this feature applies only to multiband rasters.
  4. Alternatively, you can fetch the pixel values directly from the raster using the  Add values from display button. Then enter the transparency value.
  5. Repeat the steps to adjust more values with custom transparency.
  6. Press the *Apply* button and have a look at the map.

  As you can see, it is quite easy to set custom transparency, but it can be quite a lot of work. Therefore, you can use the button  Export to file to save your transparency list to a file. The button  Import from file loads your transparency settings and applies them to the current raster layer.

13.1.5 Labels Properties

The  *Labels* properties provides you with all the needed and appropriate capabilities to configure smart labeling on raster layers. This dialog can also be accessed from the *Layer Styling* panel.

At the top of the dialog, you have:

- a combobox for selecting the appropriate labeling method for the active layer
- the  Configure project labeling rules button: helps you control interactions between labels and features across the layers in the project. More details at *Configuring project labeling rules*.
- the  Automated placement settings (applies to all layers) button: configure general properties on label placement and conflicts resolution. More details at *Setting the automated placement engine*.

The first step is to choose the labeling method from the drop-down list. Available methods are:

-  *No labels*: the default value, showing no labels from the layer
-  *Label with pixel values*: Show labels on the map using a band.

Using the  *Label with pixel values* option, the following dialog opens.

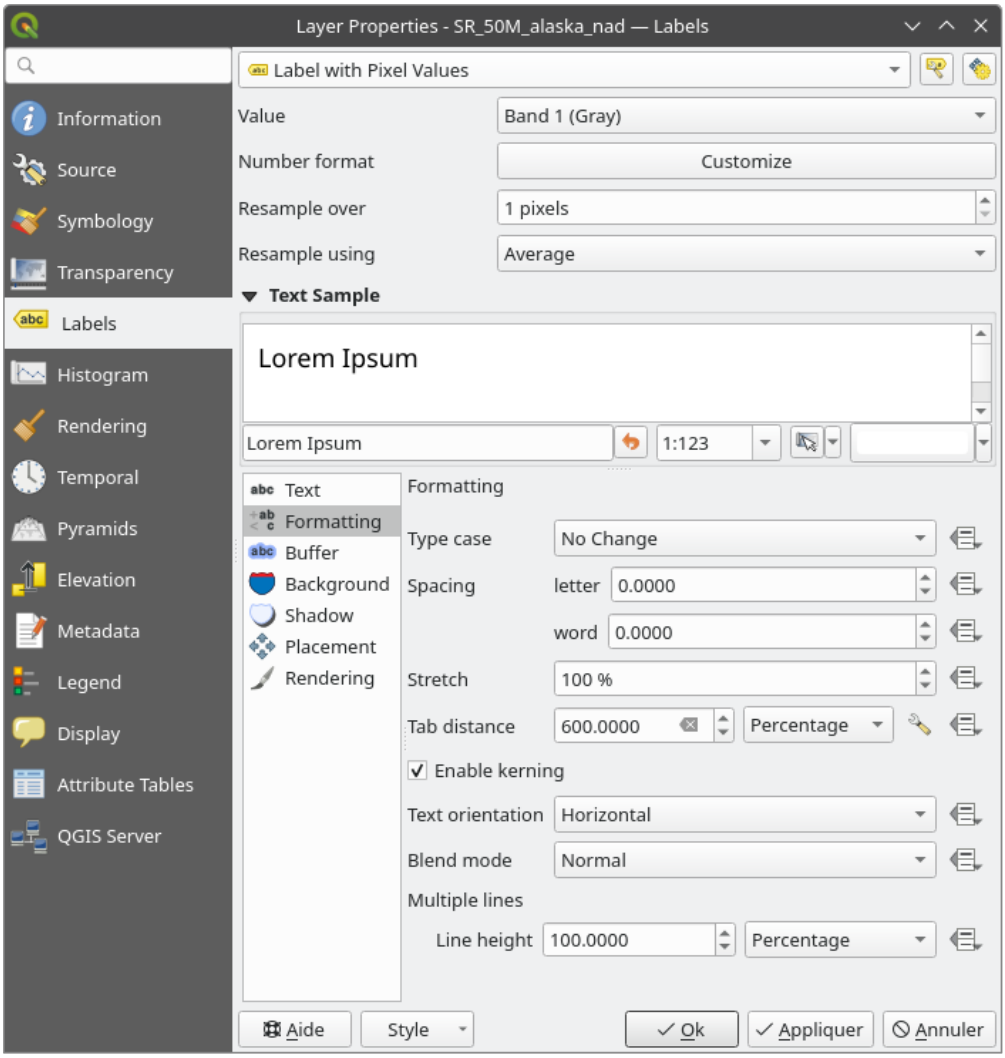


Fig. 13.14: Raster layer labeling settings

At the top of the dialog:

- A *Value* drop-down list allows you to select the band to take the values from
- Press *Customize* to configure a *proper number formatting*
- By default, the displayed value represents individual pixel band value. With *Resample over*, you can compute the value from the neighbouring pixels (setting `2` means `2*2=4` pixels) using a statistical method set in the *Resample using* widget.

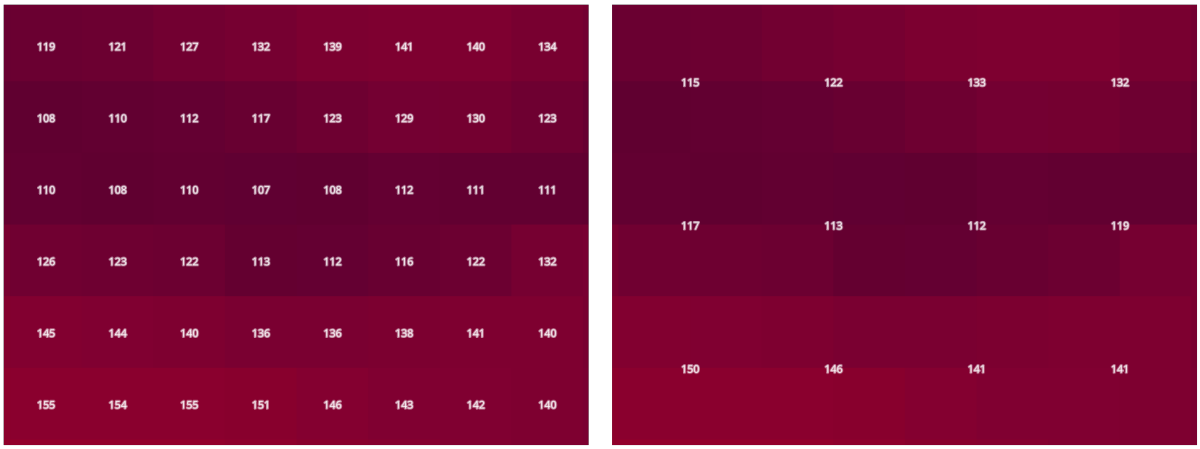


Fig. 13.15: Pixels labeled using various resampling options

Below are displayed options to customize the labels, under various tabs:

- abc *Text*
- *Formatting*
- abc *Buffer*
- *Background*
- *Shadow*
- *Placement*
- *Rendering*

Description of how to set each property is exposed at *Setting a label*.

13.1.6 Histogram Properties

The *Histogram* tab allows you to view the distribution of the values in your raster. The histogram is generated when you press the *Compute Histogram* button. All existing bands will be displayed together. You can save the histogram as an image with the button.

At the bottom of the histogram, you can select a raster band in the drop-down menu and *Set min/max style for* it. The *Prefs/Actions* drop-down menu gives you advanced options to customize the histogram:

- With the *Visibility* option, you can display histograms for individual bands. You will need to select the option *Show selected band*.
- The *Min/max options* allow you to 'Always show min/max markers', to 'Zoom to min/max' and to 'Update style to min/max'.
- The *Actions* option allows you to 'Reset' or 'Recompute histogram' after you have changed the min or max values of the band(s).

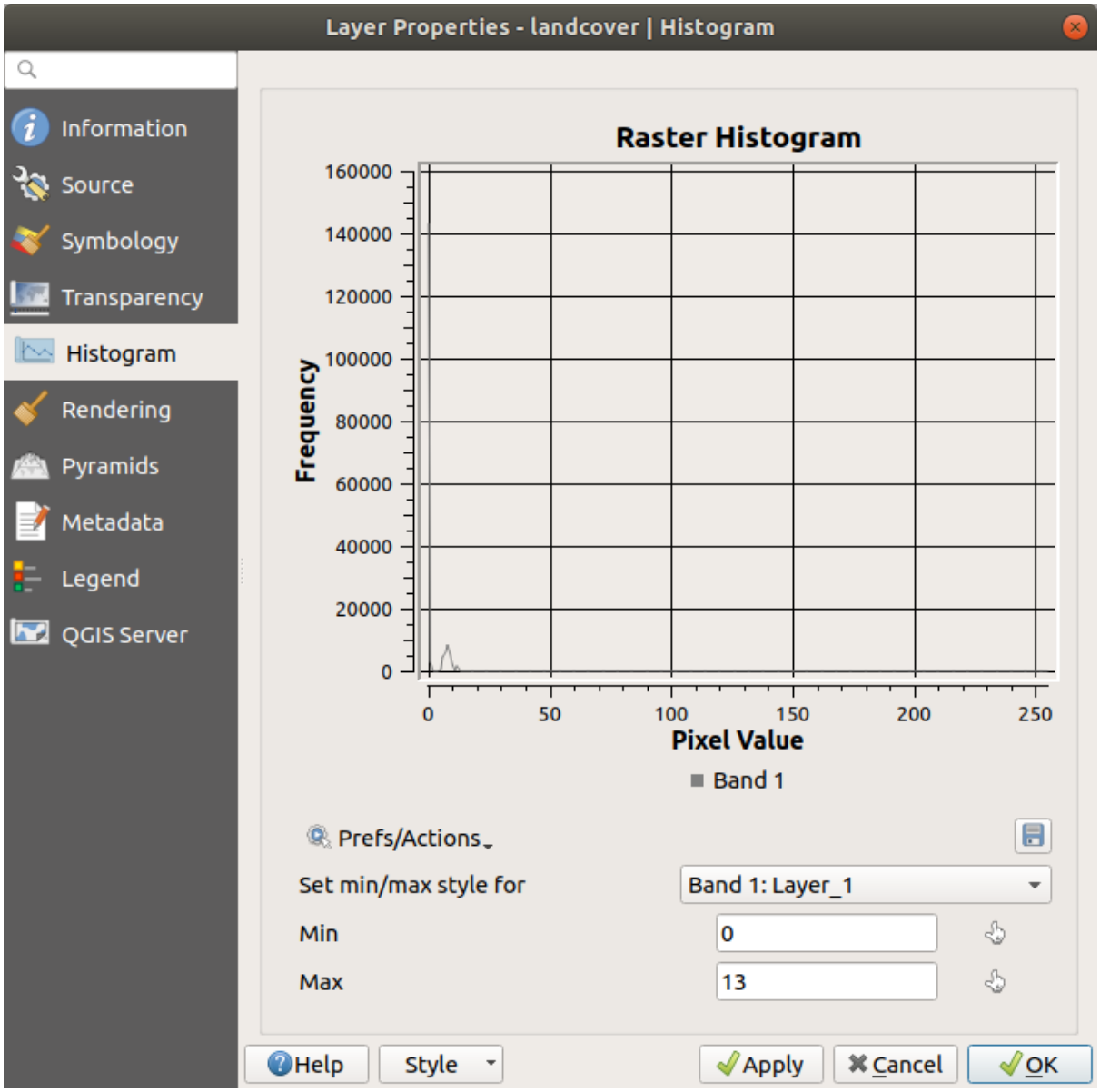


Fig. 13.16: Raster Histogram

13.1.7 Rendering Properties

In the *Rendering* tab, it's possible to:

- set *Scale dependent visibility* for the layer: You can set the *Maximum (inclusive)* and *Minimum (exclusive)* scales, defining a range of scales in which the layer will be visible. It will be hidden outside this range. The Set to current canvas scale button helps you use the current map canvas scale as a boundary. See *Visibility Scale Selector* for more information.

> **Note**
>
> You can also activate scale dependent visibility on a layer from within the *Layers* panel: right-click on the layer and in the contextual menu, select *Set Layer Scale Visibility*.

- *Refresh layer at interval*: controls whether and how regular a layer can be refreshed. Available *Configuration* options are:
  - *Reload data*: the layer will be completely refreshed. Any cached data will be discarded and refetched from the provider. This mode may result in slower map refreshes.
  - *Redraw layer only*: this mode is useful for animation or when the layer's style will be updated at regular intervals. Canvas updates are deferred in order to avoid refreshing multiple times if more than one layer has an auto update interval set.

  It is also possible to set the *Interval (seconds)* between consecutive refreshments.

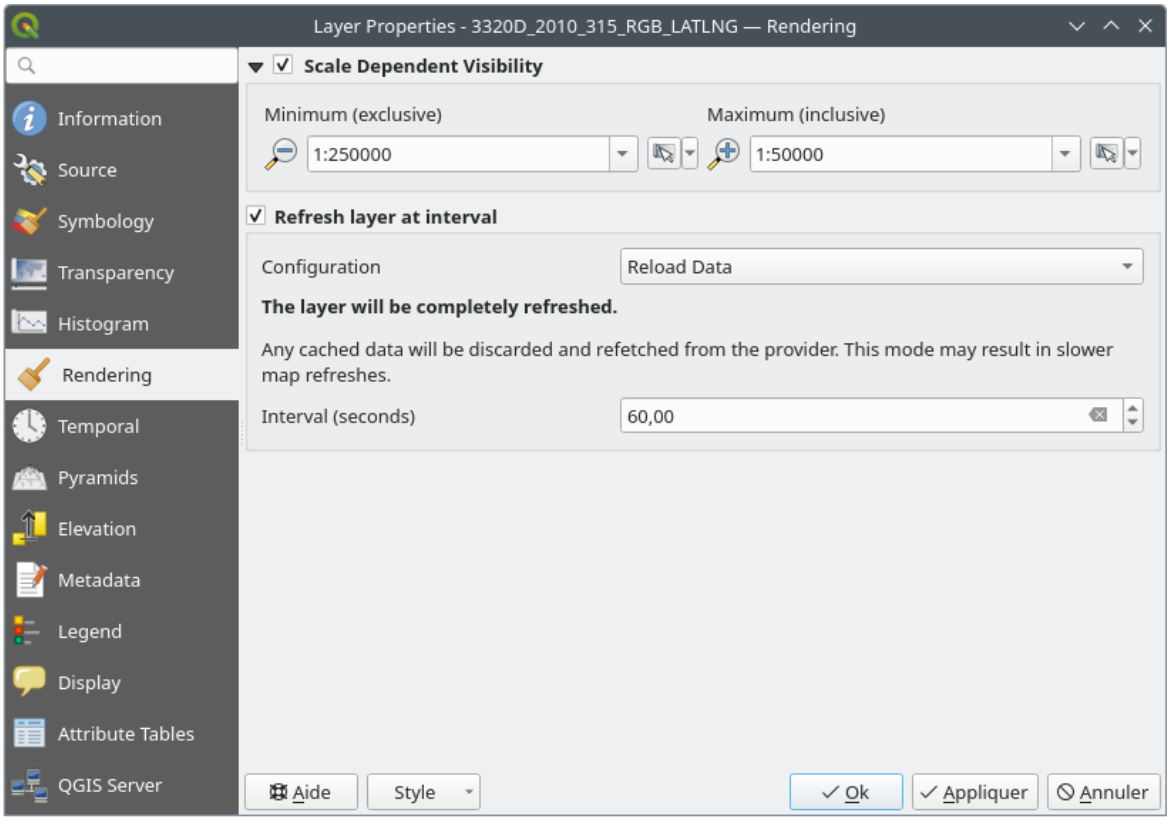


Fig. 13.17: Raster Rendering Properties

13.1.8 Temporal Properties

The *Temporal* tab provides options to control the rendering of the layer over time. Such dynamic rendering requires the *temporal navigation* to be enabled over the map canvas.

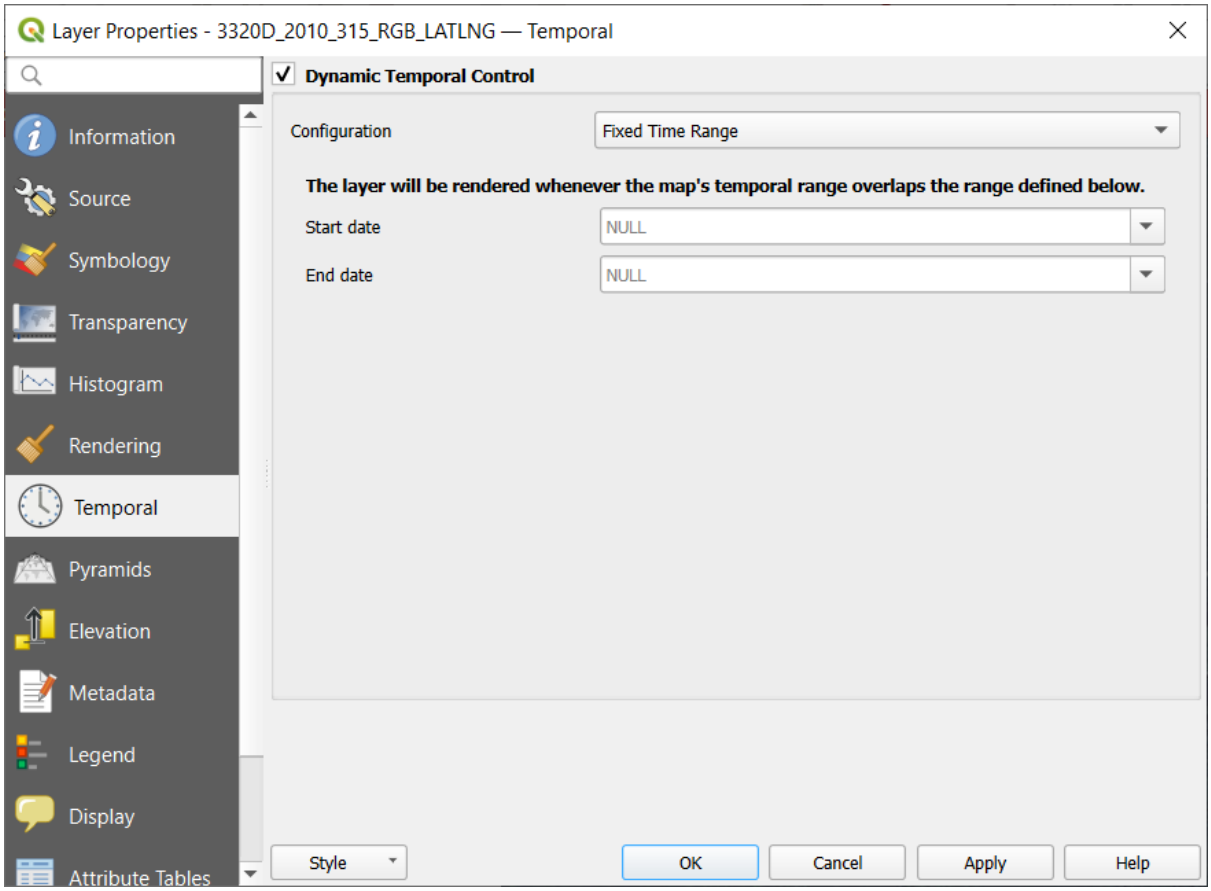


Fig. 13.18: Raster Temporal Properties

Check the *Dynamic Temporal Control* option and set whether the layer redraw should be:

- *Automatic*: the rendering is controlled by the underlying data provider if it suppports temporal data handling. E.g. this can be used with WMS-T layers or PostgreSQL rasters.
- *Fixed Date/Time*: only show the raster layer at a single, specific date/time. This avoids having to enter the same value for both the start and end of the temporal range.
- *Fixed time range*: only show the raster layer if the animation time is within a *Start date* and *End date* range.
- *Fixed Time Range Per Band*: only shows a band when the current animation time is between its *Begin* and *End* date range. This option allows you to either manually set these time ranges for each band or use the ε button to automatically generate datetime values, enabling detailed temporal analysis and visualization. This mode is particularly useful for working with raster layers where each band corresponds to a specific time period, such as NetCDF files.
- *Represents Temporal Values*: interprets each pixel in the raster layer as a datetime value. When this temporal mode is active, pixels that do not fall within the temporal range specified in the render context will be hidden, ensuring that only temporally relevant data is displayed. This mode is effective for:
  - Analyzing land use changes, like observing deforestation patterns.
  - Studying flooding by comparing water coverage across different times.
  - Evaluating movement costs in terrain analysis, for example, using GRASS GIS's r.walk tool to calculate travel costs across a landscape.
- *Redraw layer only*: the layer is redrawn at each new animation frame. It's useful when the layer uses time-based expression values for renderer settings (e.g. data-defined renderer opacity, to fade in/out a raster layer).

13.1.9 Pyramids Properties

High resolution raster layers can slow navigation in QGIS. By creating lower resolution copies of the data (pyramids), performance can be considerably improved, as QGIS selects the most suitable resolution to use depending on the zoom level.

You must have write access in the directory where the original data is stored to build pyramids.

From the *Resolutions* list, select resolutions at which you want to create pyramid levels by clicking on them.

If you choose **Internal (if possible)** from the *Overview format* drop-down menu, QGIS tries to build pyramids internally.

> **Note**
>
> Please note that building pyramids may alter the original data file, and once created they cannot be removed. If you wish to preserve a 'non-pyramided' version of your raster, make a backup copy prior to pyramid building.

If you choose **External** and **External (Erdas Imagine)** the pyramids will be created in a file next to the original raster with the same name and a `.ovr` extension.

Several *Resampling methods* can be used for pyramid calculation:

- Nearest Neighbour
- Average
- Gauss
- Cubic
- Cubic Spline
- Laczos
- Mode
- None

Finally, click *Build Pyramids* to start the process.

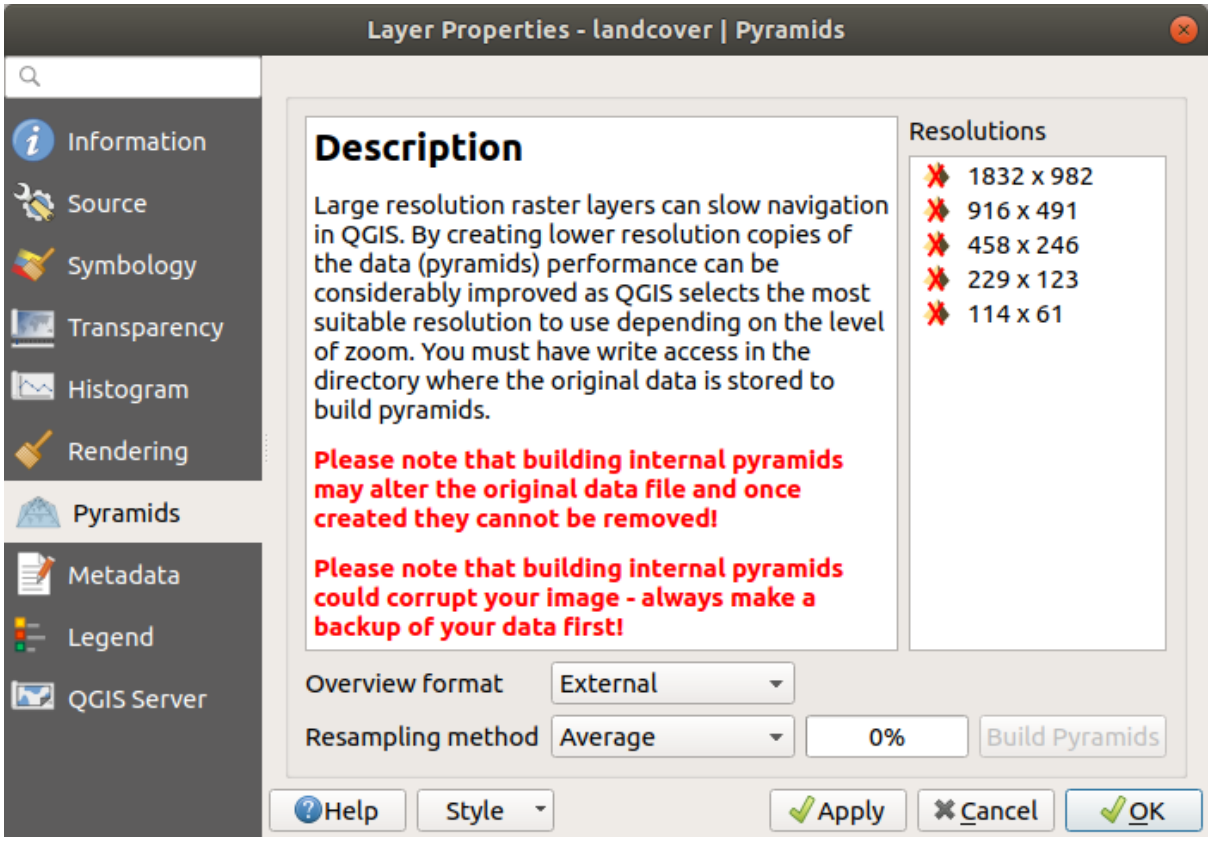


Fig. 13.19: Raster Pyramids

13.1.10 Elevation Properties

The *Elevation* tab provides options to control the layer elevation properties within a *3D map view* and its appearance in the *profile tool charts*. Specifically, you can choose to *Disable* this configuration if the layer does not contain elevation data or you can set:

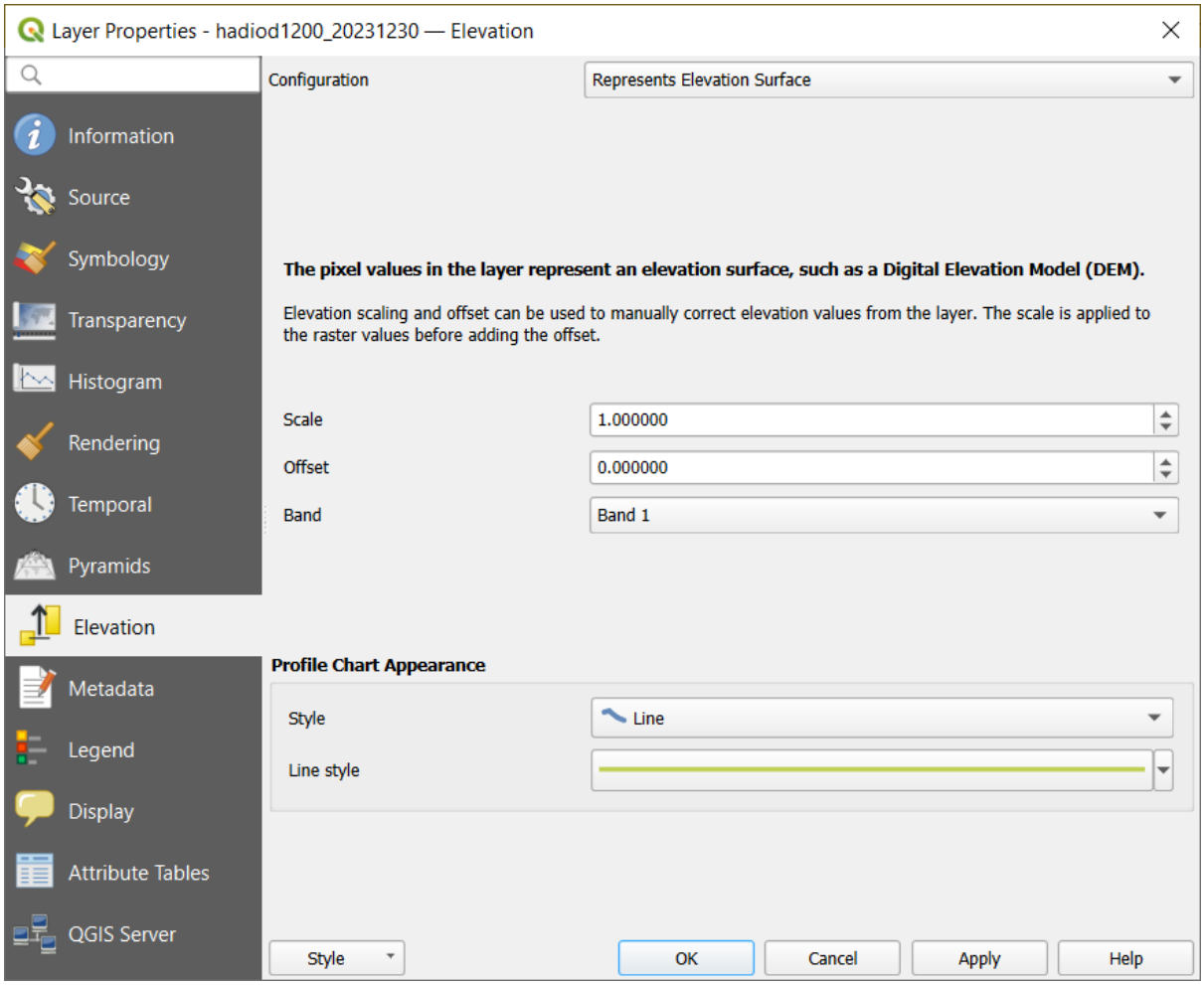


Fig. 13.20: Raster Elevation Properties

- *Represents Elevation Surface*: whether the raster layer represents a height surface (e.g DEM) and the pixel values should be interpreted as elevations. Choose this option if you want to display a raster in an *elevation profile view*. You will also need to fill in the *Band* to pick values from and can apply a *Scale* factor and an *Offset*.
- *Fixed Elevation Range*: The raster layer (or selected raster band) is associated with a fixed elevation range. This mode can be used when a layer has a single fixed elevation or a range (slice) of elevation values. If a range is specified, pixels will be extruded over this range. You can set the *Lower* and *Upper* elevation range values for the layer, and specify whether the lower or upper *Limits* are inclusive or exclusive.
- *Fixed Elevation Range Per Band*: Each band in the raster can have a fixed elevation range associated with it. This is designed for data sources that expose elevation-related data in bands, such as NetCDF files. For example, a raster with temperature data at different ocean depths. When rendering, the uppermost matching band will be selected and used for the layer's data. This feature is exposed as a user-editable table for raster bands with lower and upper values. Users can either populate the lower and upper values manually or use an ε *Expression* to auto-fill all band values based on expression. The expression-based fill allows you to design expressions that extract useful information from band names. For example, extracting the depth value from a band name like "Band 001: depth=-5500 (meters)".
- *Dynamic Elevation Range Per Band*: This mode calculates elevation ranges for raster bands dynamically using QGIS expressions. It's ideal for datasets where elevation values follow a consistent pattern across bands (like equally spaced vertical layers), eliminating the need to manually assign fixed elevations. Instead of entering individual values for each band, you define expressions for the *Lower* and *Upper* elevation bounds. These expressions can use variables like `@band`, `@band_name`, or `@band_description` automatically computing the elevation range based on each band's properties.

- *Profile Chart Appearance*: controls the rendering of the raster elevation data in the profile chart. The profile *Style* can be set as:
  - a *Line* with a specific *Line style*
  - an elevation surface rendered using a fill symbol either above (*Fill above*) or below (*Fill below*) the elevation curve line. The surface symbology is represented using:
    - a *Fill style*
    - and a *Limit*: the maximum (respectively minimum) altitude determining how high the fill surface will be

13.1.11 Metadata Properties

The *Metadata* tab provides you with options to create and edit a metadata report on your layer. See *Metadata* for more information.

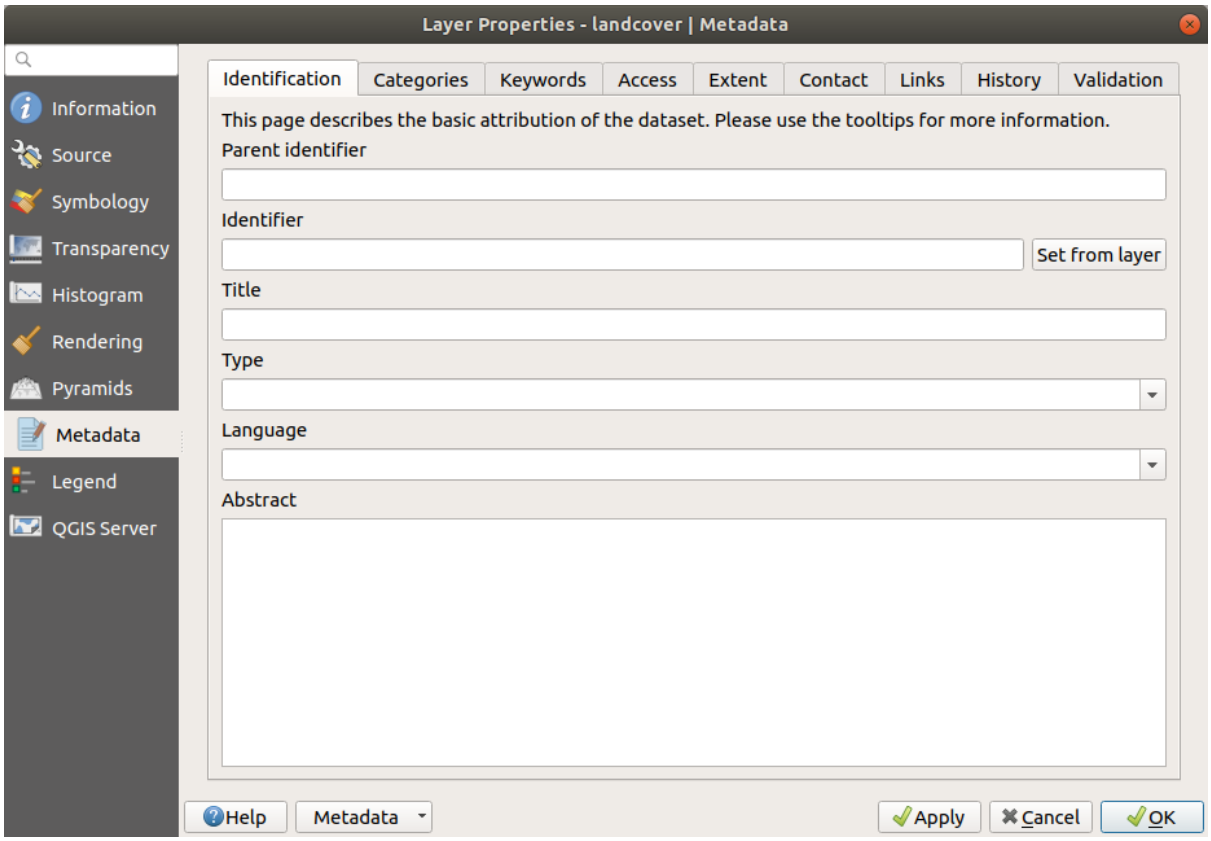


Fig. 13.21: Raster Metadata

13.1.12 Legend Properties

The *Legend* tab provides you with advanced settings for the *Layers panel* and/or the *print layout legend*. These options include:

- Depending on the symbology applied to the layer, you may end up with several entries in the legend, not necessarily readable/useful to display. The *Legend placeholder image* helps you *select an image* for replacement, displayed both in the *Layers* panel and the print layout legend.
- The *Embedded widgets in Legend* provides you with a list of widgets you can embed within the layer tree in the Layers panel. The idea is to have a way to quickly access some actions that are often used with the layer (setup transparency, filtering, selection, style or other stuff…).

  By default, QGIS provides a transparency widget but this can be extended by plugins that register their own widgets and assign custom actions to layers they manage.

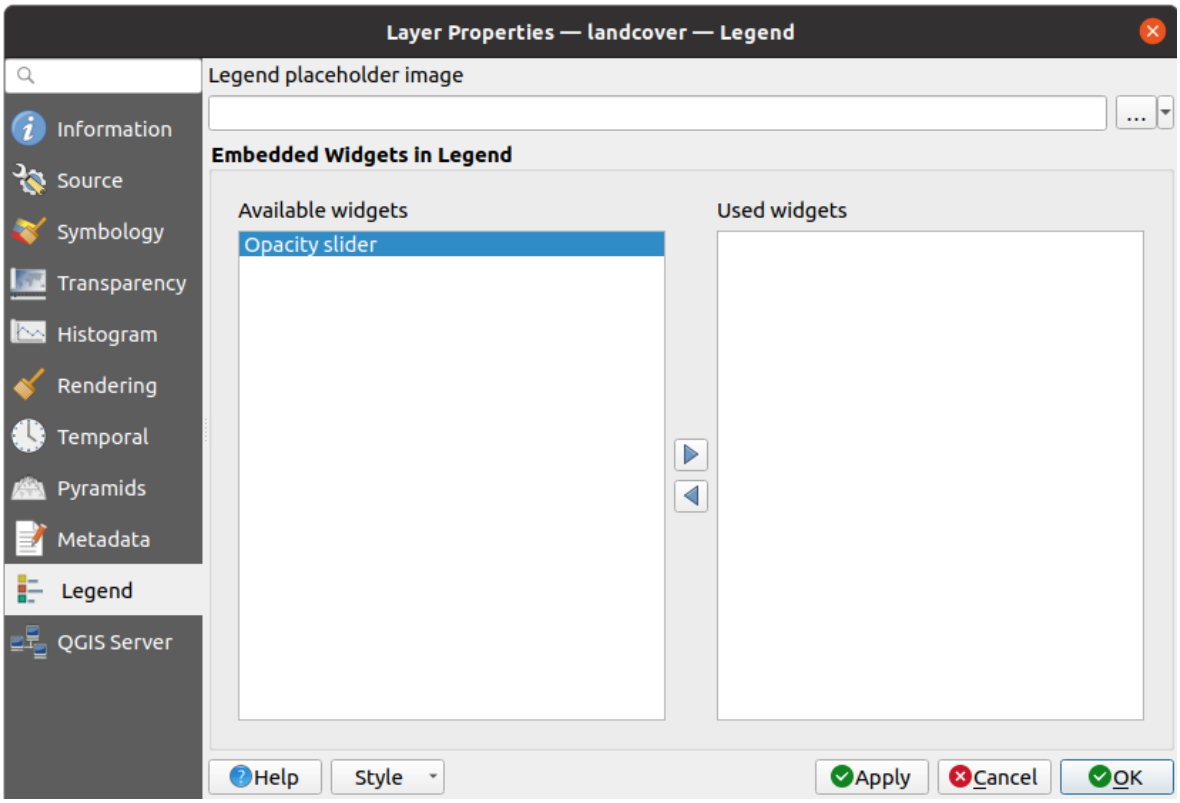


Fig. 13.22: Raster Legend

13.1.13 Display Properties

The *Display* tab helps you configure HTML map tips to use for pixels identification:

- *Enable Map Tips* controls whether to display map tips for the layer
- The *HTML Map Tip* provides a complex and full HTML text editor for map tips, mixing QGIS expressions and html styles and tags (multiline, fonts, images, hyperlink, tables, …). You can check the result of your code sample in the *Preview* frame. You can also select and edit existing expressions using the *Insert/Edit Expression* button.

  You might look for expressions located in *Rasters* group or the `@layer_cursor_point` variable in the *Expressions* dialog.

> **Note**
>
> **Understanding the *Insert/Edit Expression* button behavior**
>
> If you select some text within an expression (between "[%" and "%]"), or if no text is selected but the cursor is inside an expression, the whole expression will be automatically selected for editing. If the cursor or a selected text is outside an expression, the dialog opens with the selection.

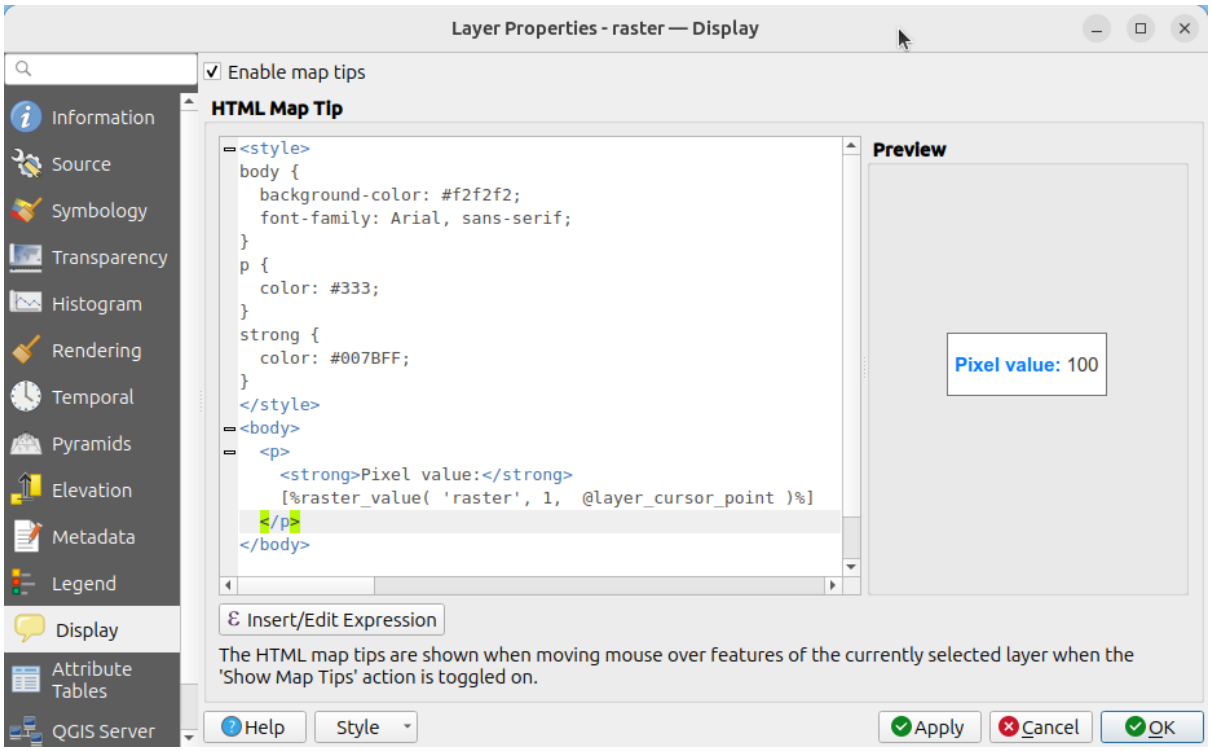


Fig. 13.23: Map tips with raster layer

To display map tips:

1. Select the menu option *View* ► *Show Map Tips* or click on the Show Map Tips icon of the *Attributes Toolbar*.
2. Make sure that the layer you target is active and has the *Enable Map Tips* property checked.
3. Move over a pixel, and the corresponding information will be displayed over.

Map tip is a cross-layer feature meaning that once activated, it stays on and applies to any map tip enabled layer in the project until it is toggled off.

13.1.14 QGIS Server Properties

The *QGIS Server* tab helps you configure settings of the data when published by QGIS Server. The configuration concerns:

- *Description*: provides information to describe the data, such as *Short name*, *Title*, *Summary*, a *List of Keywords*, and a *Data URL* whose *Type* can be in `text/html`, `text/plain` or `application/pdf`.
- *Attribution*: a *Title* and *URL* to identify who provides the data
- *Metadata URL*: a list of *URL* for the metadata that can be of `FGDC` or `TC211` *Type*, and in `text/plain` or `text/xml` *Format*

- *Legend URL*: a *URL* for the legend, in either `image/png` or `image/jpeg` *Format*

> **Note**
>
> When the raster layer you want to publish is already provided by a web service, further *properties* are available for setting.

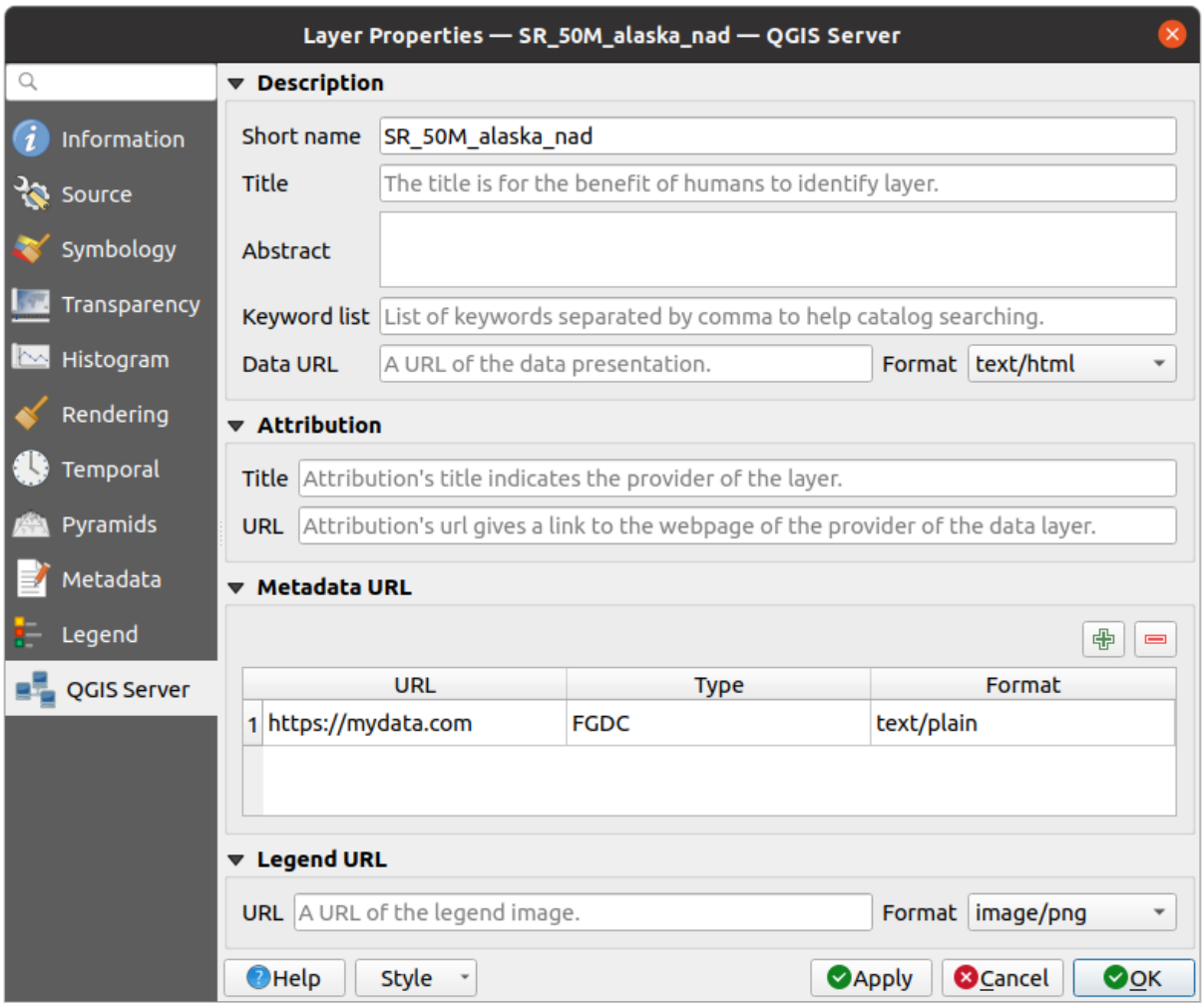


Fig. 13.24: QGIS Server in Raster Properties

13.1.15 Identify raster cells

The  *identify features* tool allows you to get information about specific points in a raster layer.

To use the  *Identify features* tool:

1. Select the raster layer in the Layers panel.
2. Click on the *Identify features* tool in the toolbar or press `Ctrl+Shift+I`.
3. Click on the point in the raster layer that you want to identify.

The Identify Results panel will open in its default `Tree` view and display information about the clicked point. Formatting of the results vary depending on the provider of the layer. For example:

- For a local raster layer: below the name of the layer, you have on the left the band(s) of the clicked pixel, and on the right their respective value.
- For a remote layer such as WMS, a *Format* menu allows you to select whether the information should be displayed as *HTML*, *Feature* or *Text*.

These values can also be rendered (from the *View* menu located at the bottom of the panel) in:

- a `Table` view - organizes the information about the identified features and their values in a table.
- a `Graph` view - organizes the information about the identified features and their values in a graph.

Under the pixel attributes, you will find the *Derived* information, such as:

- `X` and `Y` coordinate values of the point clicked
- Column and row of the point clicked (pixel) when compatible

13.2 Raster Analysis

13.2.1 Raster Calculator

The *Raster Calculator* in the *Raster* menu allows you to perform calculations on the basis of existing raster pixel values (see Fig. 13.25). The results are written to a new raster layer in a GDAL-supported format.

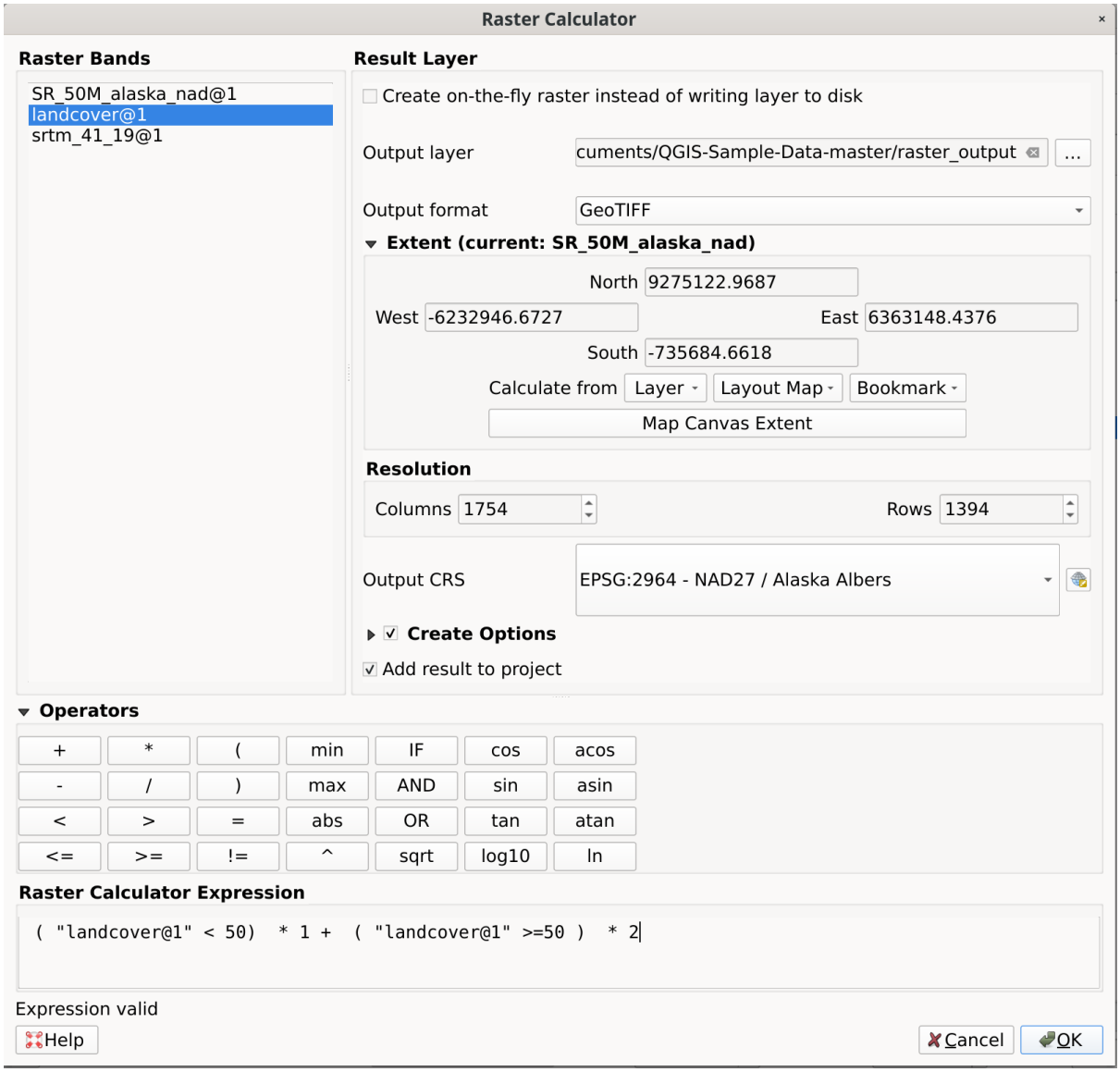


Fig. 13.25: Raster Calculator

The *Raster bands* list contains all loaded raster layers that can be used. To add a raster to the raster calculator expression field, double click its name in the Fields list. You can then use the operators to construct calculation expressions, or you can just type them into the box.

In the *Result layer* section, you will need to define an output layer. You can:

- *Create on-the-fly raster instead of writing layer to disk*:
  - If unchecked, the output is stored on the disk as a new plain file. An *Output layer* path and an *Output format* are required.
  - If checked, a virtual raster layer, i.e. a raster layer defined by its URI and whose pixels are calculated on-the-fly, is created. It's not a new file on disk; the virtual layer is still connected to the rasters used in the calculation meaning that deleting or moving these rasters would break it. A *Layer name* can be provided, otherwise the calculation expression is used as such. Removing the virtual layer from the project deletes it, and it can be made persistent in file using the layer *Export* ► *Save as…* contextual menu.
- Define the *Spatial extent* of the calculation using the *extent selector* widget.
- Set the *Resolution* of the layer using columns and rows number. If the input layer has a different resolution, the values will be resampled with the nearest neighbor algorithm.

- Enable the *Create Options* to define additional raster creation options. These options let you control how the output file is structured and compressed, including format, compression method, and other driver-specific settings. See more at *Raster driver options*.
- With the *Add result to project* checkbox, the result layer will automatically be added to the legend area and can be visualized. Checked by default for virtual rasters.

The *Operators* section contains all available operators. To add an operator to the raster calculator expression box, click the appropriate button. Mathematical calculations (+, −, *, …) and trigonometric functions (`sin`, `cos`, `tan`, …) are available. Conditional expressions (=, !=, <, >=, …) return either 0 for false or 1 for true, and therefore can be used with other operators and functions.

See also

Raster calculator and *Raster calculator (virtual)* algorithms

Raster calculator expression

The dialog

The *Raster calculator expression* dialog provides means to write expressions for pixels calculations between a set of raster layers.

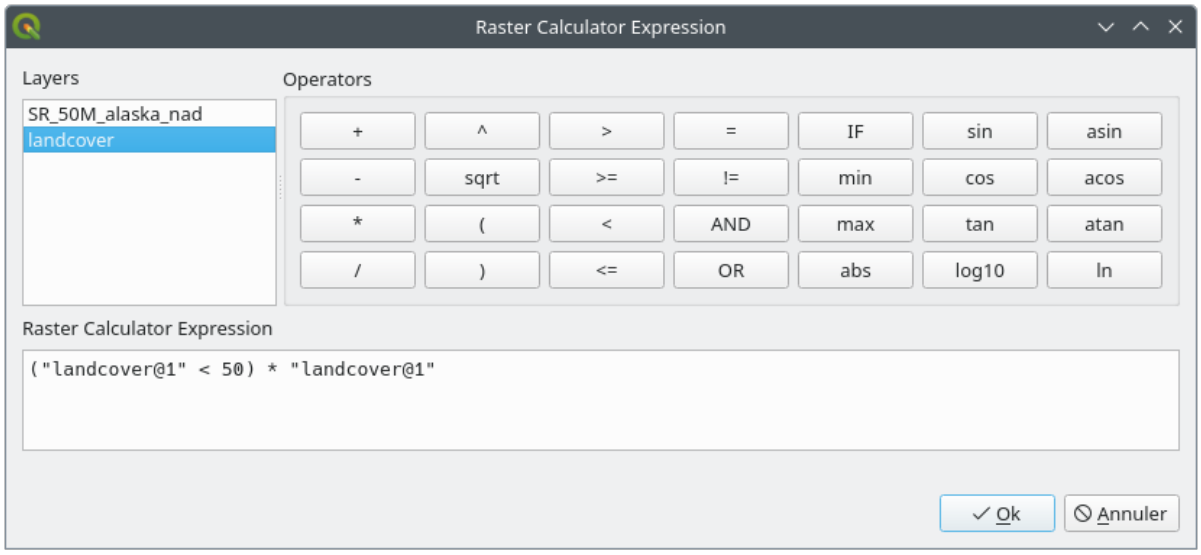


Fig. 13.26: Raster Expression Calculator

- *Layers*: Shows the list of all raster layers loaded in the legend. These can be used to fill the expression box (double click to add). Raster layers are referred by their name and the number of the band: `layer_name@band_number`. For instance, the first band from a layer named `DEM` will be referred as `DEM@1`.
- *Operators*: contains a number of calculation operators for pixels manipulation:
  - Arithmetical: +, −, *, `sqrt`, `abs`, `ln`, …
  - Trigonometric: `sin`, `cos`, `tan`, …
  - Comparison: =, !=, <, >=, …
  - Logical: `IF`, `AND`, `OR`, `(`, `)`
  - Statistical: `min`, `max`

To add an operator to the raster calculator expression box, click the appropriate button.

- *Raster calculator expression* is the area in which the expression is composed

Examples

Convert elevation values from meters to feet

Creating an elevation raster in feet from a raster in meters, you need to use the conversion factor for meters to feet: 3.28. The expression is:

```
"elevation@1" * 3.28
```

Using a mask

If you want to mask out parts of a raster – say, for instance, because you are only interested in elevations above 0 meters – you can use the following expression to create a mask and apply the result to a raster in one step.

```
("elevation@1" >= 0) * "elevation@1"
```

In other words, for every cell greater than or equal to 0 the conditional expression evaluates to 1, which keeps the original value by multiplying it by 1. Otherwise the conditional expression evaluates to 0, which sets the raster value to 0. This creates the mask on the fly.

Classify a Raster

If you want to classify a raster – say, for instance into two elevation classes, you can use the following expression to create a raster with two values 1 and 2 in one step.

```
("elevation@1" < 50) * 1 + ("elevation@1" >= 50) * 2
```

In other words, for every cell less than 50 set its value to 1. For every cell greater than or equal 50 set its value to 2.

Or you can use the `IF` operator.

```
if ( elevation@1 < 50 , 1 , 2 )
```

CHAPTER

FOURTEEN

WORKING WITH MESH DATA

14.1 What's a mesh?

A mesh is an unstructured grid usually with temporal and other components. The spatial component contains a collection of vertices, edges and/or faces, in 2D or 3D space:

- **vertices** - XY(Z) points (in the layer's coordinate reference system)
- **edges** - connect pairs of vertices
- **faces** - a face is a set of edges forming a closed shape - typically a triangle or a quadrilateral (quad), rarely polygons with more vertices

Relying on the above, mesh layers can thus have different types of structure:

- 1D Meshes: consist of vertices and edges. An edge connects two vertices and can have assigned data (scalars or vectors) on it. The 1D mesh network can be for example used for modelling of an urban drainage system.
- 2D meshes: consist of faces with triangles, regular or unstructured quads.
- 3D layered meshes: consist of multiple stacked 2D unstructured meshes each extruded in the vertical direction (levels) by means of a vertical coordinate. The vertices and faces have the same topology in each vertical level. The mesh definition (vertical level extrusion) could in general change in time. The data is usually defined in volume centres or by some parametric function.

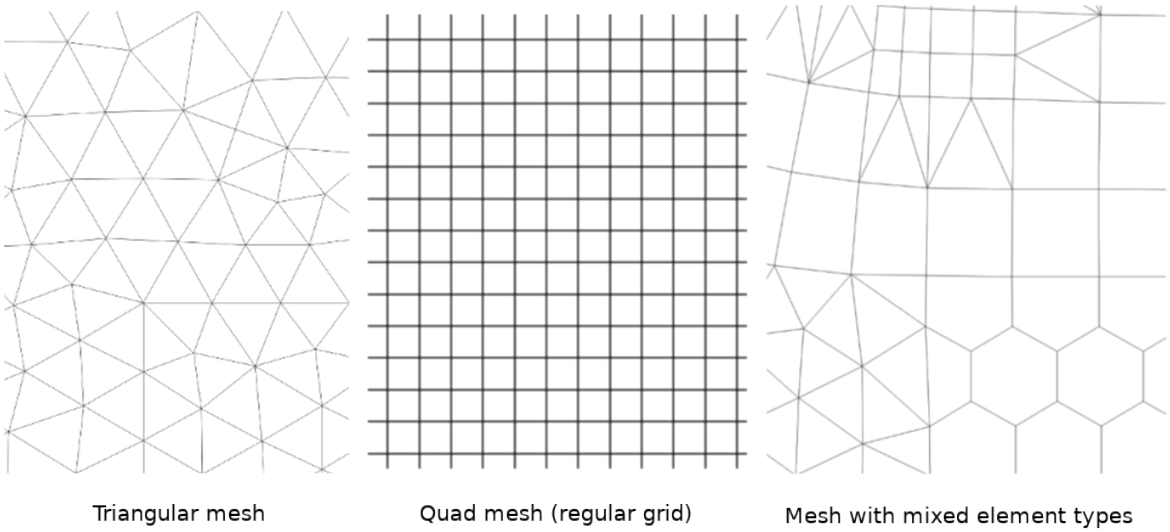


Fig. 14.1: Different mesh types

Mesh provides information about the spatial structure. In addition, the mesh can have datasets (groups) that assign a value to every vertex. For example, having a triangular mesh with numbered vertices as shown in the image below:

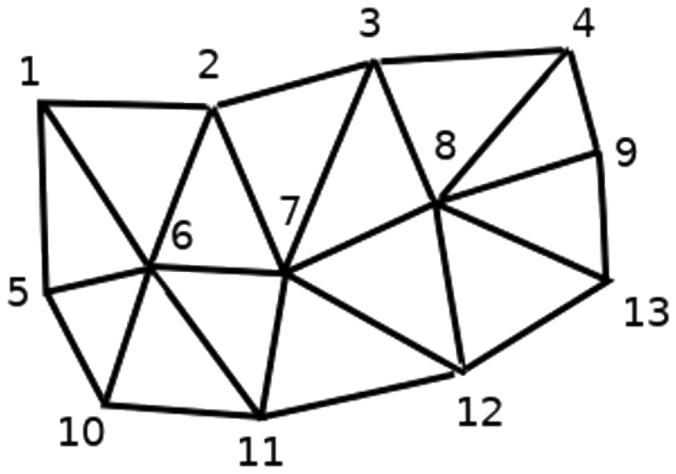


Fig. 14.2: Triangular grid with numbered vertices

Each vertex can store different datasets (typically multiple quantities), and those datasets can also have a temporal dimension. Thus, a single file may contain multiple datasets.

The following table gives an idea about the information that can be stored in mesh datasets. Table columns represent indices of mesh vertices, each row represents one dataset. Datasets can have different datatypes. In this case, it stores wind velocity at 10m at a particular moments in time (t1, t2, t3).

In a similar way, the mesh dataset can also store vector values for each vertex. For example, wind direction vector at the given time stamps:

Table 14.1: Example of mesh dataset

10 metre wind	1	2	3	...
10 metre speed at time=t1	17251	24918	32858	…
10 metre speed at time=t2	19168	23001	36418	…
10 metre speed at time=t3	21085	30668	17251	…
…	…	…	…	…
10m wind direction time=t1	[20,2]	[20,3]	[20,4.5]	…
10m wind direction time=t2	[21,3]	[21,4]	[21,5.5]	…
10m wind direction time=t3	[22,4]	[22,5]	[22,6.5]	…
…	…	…	…	…

We can visualize the data by assigning colors to values (similarly to how it is done with *Singleband pseudocolor* raster rendering) and interpolating data between vertices according to the mesh topology. It is common that some quantities are 2D vectors rather than being simple scalar values (e.g. wind direction). For such quantities it is desirable to display arrows indicating the directions.

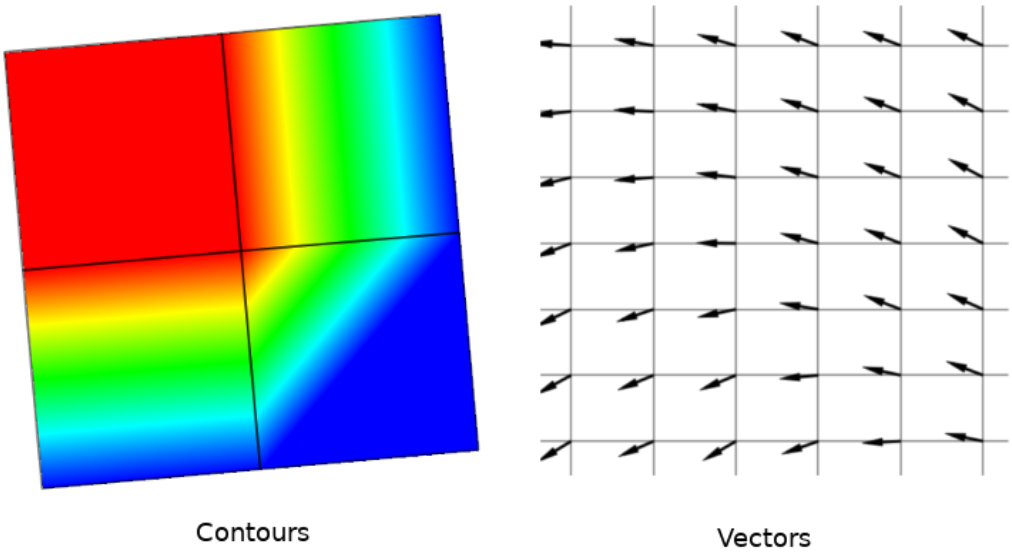


Fig. 14.3: Possible visualisation of mesh data

14.2 Supported formats

QGIS accesses mesh data using the MDAL drivers, and natively supports a variety of formats. Whether QGIS can edit a mesh layer depends on the format and the mesh structure type.

To load a mesh dataset into QGIS, use the *Mesh* tab in the *Data Source Manager* dialog. Read *Loading a mesh layer* for more details.

14.3 Mesh Dataset Properties

The *Layer Properties* dialog for a mesh layer provides general settings to manage dataset groups of the layer and their rendering (active dataset groups, symbology, 2D and 3D rendering). It also provides information about the layer.

To access the *Layer Properties* dialog:

- In the *Layers* panel, double-click the layer or right-click and select *Properties…* from the pop-up menu;
- Go to *Layer* ► *Layer Properties…* menu when the layer is selected.

The mesh *Layer Properties* dialog provides the following sections:

Table 14.2: Tabs of the Mesh Layer Properties

Information	*Source*	*Symbology*[1]
3D View[1]	*Temporal*	*Labels*[1]
Elevation	*Rendering*	*Metadata*

[1] Also available in the *Layer styling panel*

Note

Most of the properties of a mesh layer can be saved to or loaded from a `.qml` using the *Style* menu at the bottom of the dialog. More details at *Managing Custom Styles*.

14.3.1 Information Properties

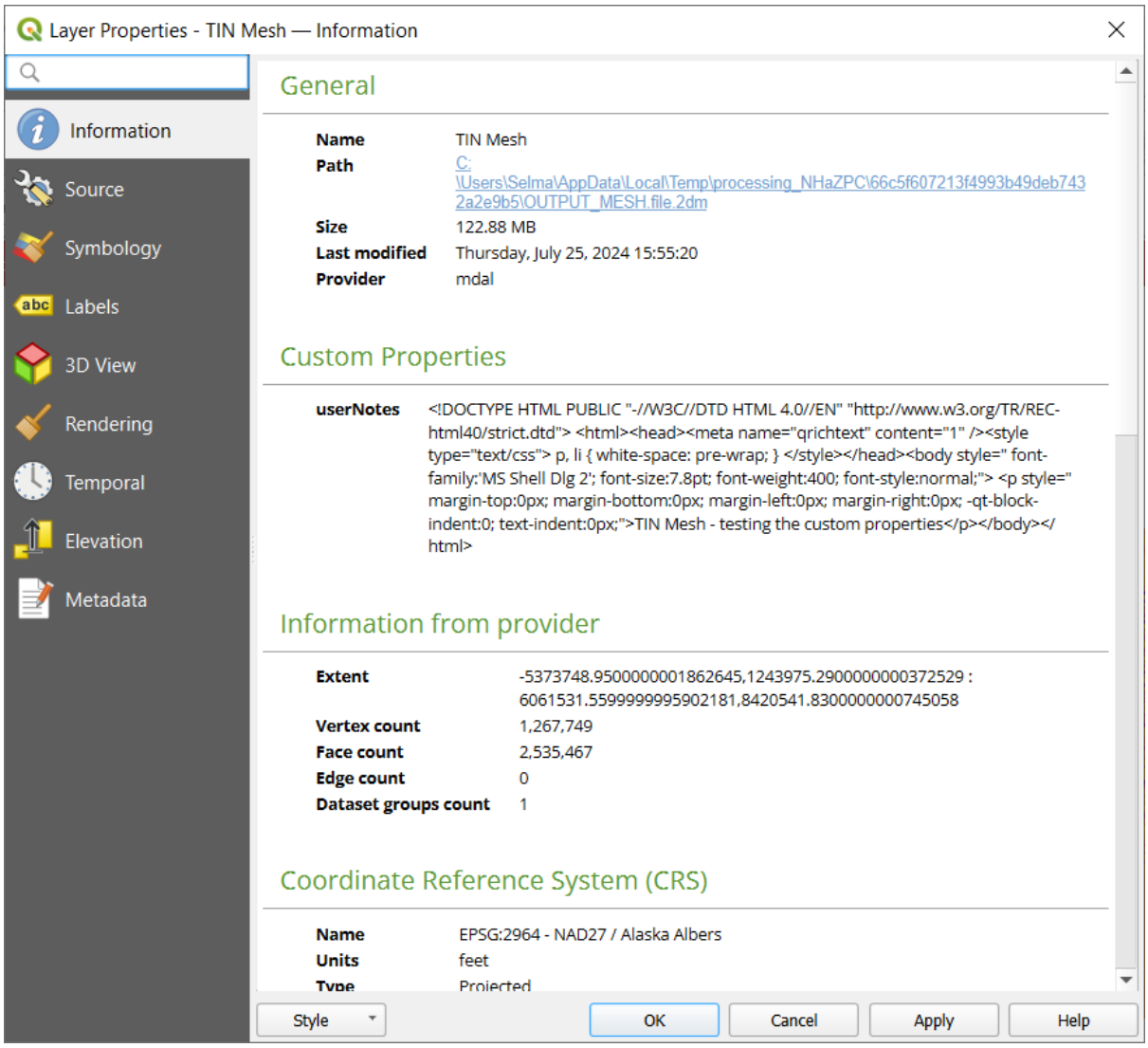


Fig. 14.4: Mesh Layer Information Properties

The *Information* tab is read-only and represents an interesting place to quickly grab summarized information and metadata on the current layer. Provided information are:

- general such as name in the project, source path, list of auxiliary files, last save time and size, the used provider
- custom properties, used to store in the active project additional information about the layer. Default custom properties may include *layer notes*. More properties can be created and managed using PyQGIS, specifically through the setCustomProperty() method.
- based on the provider of the layer: extent, vertex, face, edges and/or dataset groups count
- the Coordinate Reference System: name, units, method, accuracy, reference (i.e. whether it's static or dynamic)
- extracted from filled *metadata*: access, extents, links, contacts, history…

14.3.2 Source Properties

The *Source* tab displays basic information about the selected mesh, including:

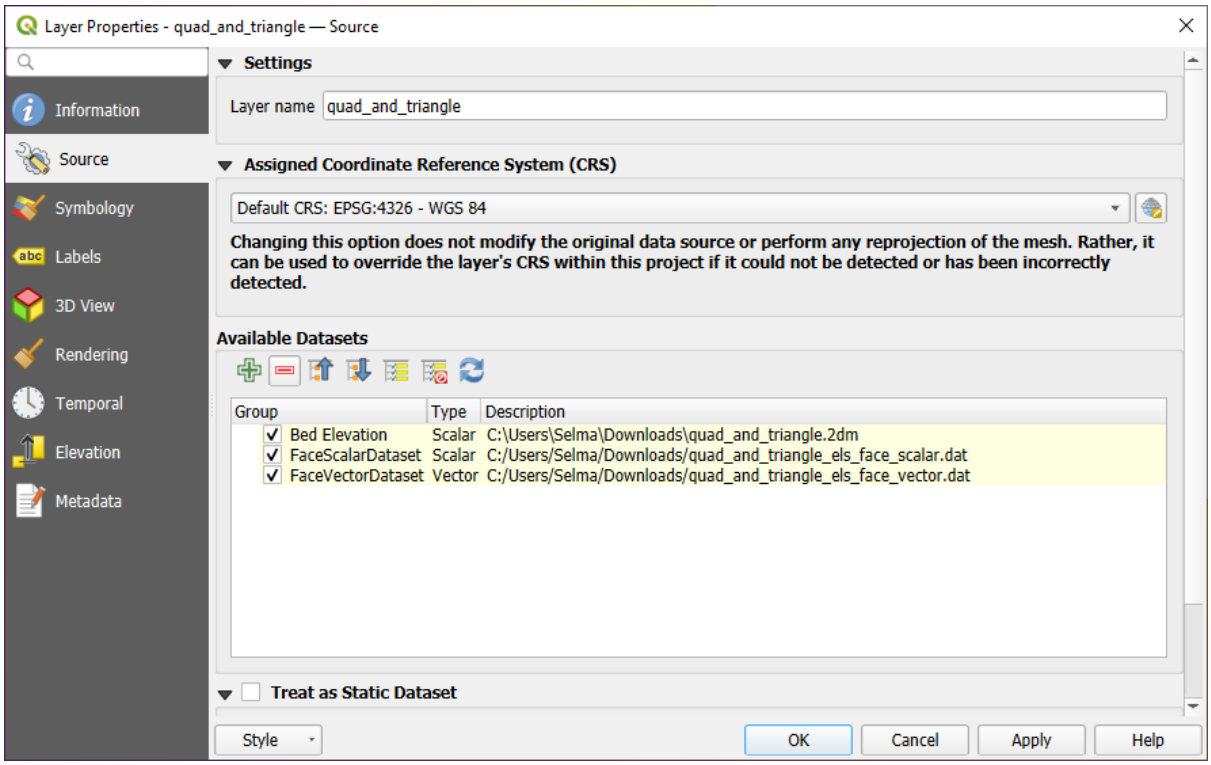


Fig. 14.5: Mesh Layer Source Properties

- the layer name to display in the *Layers* panel
- setting the Coordinate Reference System: Displays the layer's *Assigned Coordinate Reference System (CRS)*.

  You can change the layer's CRS by selecting a recently used one in the drop-down list or clicking on *Select CRS* button (see *Coordinate Reference System Selector*). Use this process only if the CRS applied to the layer is wrong or if none was applied.
- The *Available datasets* frame lists all the dataset groups (and subgroups) in the mesh layer, with their type and description in a tree view. Both regular datasets (i.e. their data is stored in the file) and virtual datasets (which are *calculated on the fly*) are listed.
  - Use the Assign Extra Dataset to Mesh button to add more groups to the current mesh layer. You can add dataset group to a mesh layer with the same name, but not from the same URI. Dataset group names are automatically renamed to "Original Name_Number".
  - Use Remove Extra Dataset from Mesh to remove additional datasets groups from the mesh layer. Note that only dataset groups not associated with the mesh source file can be removed.
  - Collapse all and Expand all the dataset tree, in case of embedded groups
  - If you are interested in few datasets, you can uncheck the others and make them unavailable in the project
  - Double-click over a name and you can rename the dataset.
  - Reset to defaults: checks all the groups and renames them back to their original name in the provider.
  - Right-click over a virtual dataset group and you can:
    - *Remove dataset group* from the project

    * *Save dataset group as…* a file on disk, to any supported format. The new file is kept assigned to the current mesh layer in the project.

- Checking the *Treat as static dataset* group allows to ignore the *map temporal navigation* properties while rendering the mesh layer. For each active dataset group (as selected in *Symbology* ► *Datasets* tab), you can:
  - set to *None*: the dataset group is not displayed at all
  - *Display dataset*: e.g., for the "bed elevation" dataset which is not time aware
  - extract a particular date time: the dataset matching the provided time is rendered and stays fixed during map navigation.

14.3.3 Symbology Properties

Click the *Symbology* button to activate the dialog. Symbology properties are divided into several tabs:

- *Datasets*
- *Contours*
- *Vectors*
- *Rendering*
- *Stacked mesh averaging method*

Datasets

The tab Datasets is the main place to control and set which datasets will be used for the layer. It presents the following items:

- *Groups* available in the mesh dataset, with whether they provide:
  - scalar dataset
  - or vector dataset: by default, each vector dataset has a scalar dataset representing its magnitude automatically generated.

  Click on the icon next to the dataset name to select the group and type of data to represent.
- *Selected dataset group(s) metadata*, with details on:
  - the mesh type: edges or faces
  - the data type: vertices, edges, faces or volume
  - whether it's of vector type or not
  - the original name in the mesh layer
  - the unit, if applicable
- *blending mode* available for the selected datasets.

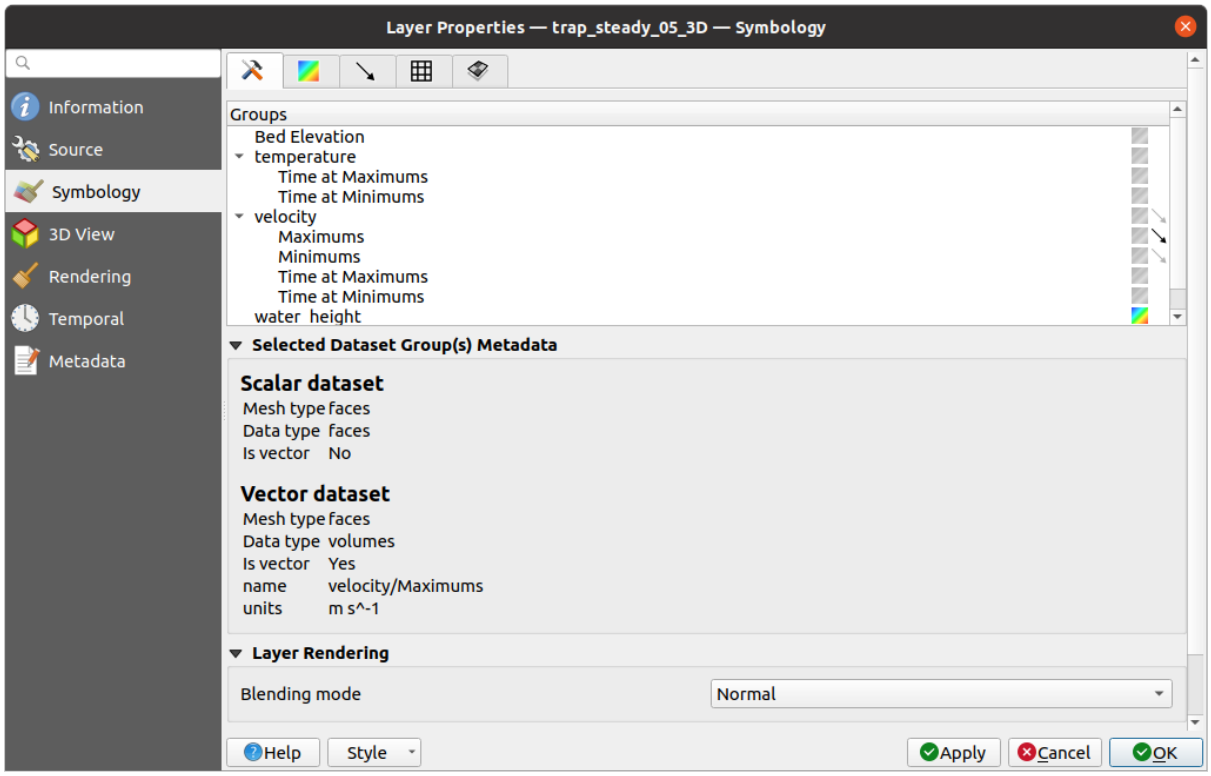


Fig. 14.6: Mesh Layer Datasets

You can apply symbology to the selected vector and/or scalar group using the next tabs.

Contours Symbology

> **Note**
>
> The Contours tab can be activated only if a scalar dataset has been selected in the *Datasets* tab.

In the Contours tab you can see and change the current visualization options of contours for the selected group, as shown in Fig. 14.7 below:

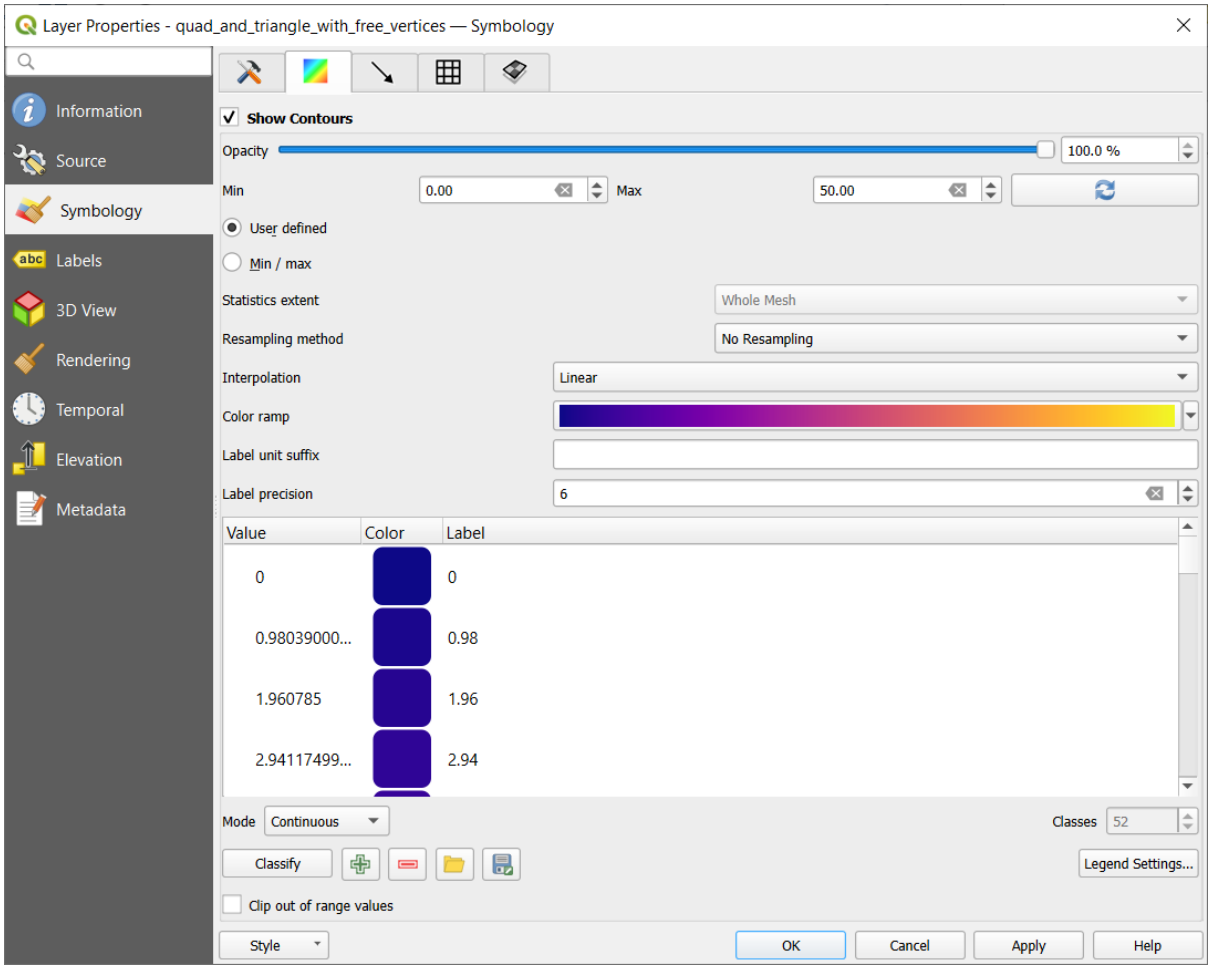


Fig. 14.7: Styling Contours in a Mesh Layer

- For 1D mesh, set the *Stroke width* of the edges. This can be a fixed size for the whole dataset, or vary along the geometry (more details with the *interpolated line renderer*)
- Use the slider or the spinbox to set the *Opacity* of the current group, if of a 2D mesh type.
- *User Defined* allows you to enter the range of values you want to represent on the current group: use Load to fetch the min and max values of the current group or enter custom values if you want to exclude some.
- Select *Min/Max* to set the renderer's minimum and maximum values based on the chosen extent.
- *Statictics extent* can be *Whole mesh*, *Current canvas* or *Updated canvas*. *Updated canvas* means that min/max values used for the rendering will change with the canvas extent (dynamic stretching).
- For 2D/3D meshes, select the *Resampling method* to interpolate the values on the surrounding vertices to the faces (or from the surrounding faces to the vertices) using the *Neighbour average* method. Depending on whether the dataset is defined on the vertices (respectively on the faces), QGIS defaults this setting to *None* (respectively *Neighbour average*) method in order to use values on vertices and keep the default rendering smooth.
- Classify the dataset using the *color ramp shader* classification.

Vectors Symbology

Note

The Vectors tab can be activated only if a vector dataset has been selected in the *Datasets* tab. Click on the Vectors icon on the right side of the *Datasets* tab.

In the Vectors tab you can see and change the current visualization options of vectors for the selected group, as shown in Fig. 14.8:

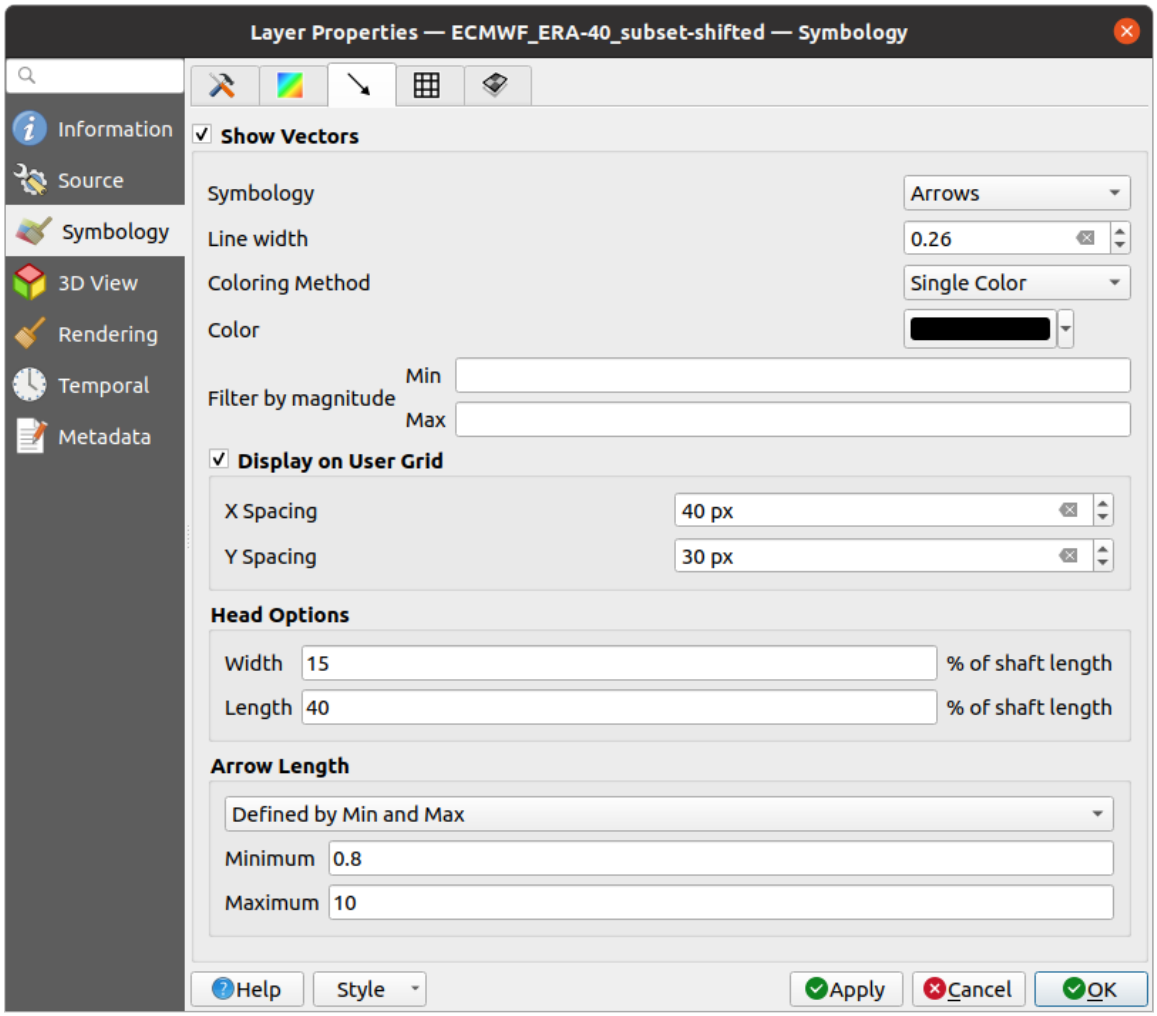


Fig. 14.8: Styling Vectors in a Mesh Layer with arrows

Mesh vector dataset can be styled using various types of *Symbology*:

- **Arrows**: vectors are represented with arrows at the same place as they are defined in the raw dataset (i.e. on the nodes or center of elements) or on a user-defined grid (hence, they are evenly distributed). The arrow length is proportional to the magnitude of the arrow as defined in the raw data but can be scaled by various methods.
- **Streamlines**: vectors are represented with streamlines seeded from start points. The seeding points can start from the vertices of the mesh, from a user grid or randomly.
- **Traces**: a nicer animation of the streamlines, the kind of effect you get when you randomly throws sand in the water and see where the sand items flows.

- **Wind barbs**: vectors are represented with wind barbs, a common way to represent wind speed and direction.

Available properties depend on the selected symbology as shown in the following table.

Table 14.3: Availability and meaning of the vectors symbology properties

Label	Description and Properties	Arrow	Streamlines	Traces	Wind Barbs
Line width	Width of the vector representation	☑	☑	☑	☑
Coloring method	• a *Single color* assigned to all vectors • or a variable color based on vectors magnitude, using a *Color ramp shader*	☑	☑	☑	☑
Filter by magnitude	Only vectors whose length for the selected dataset falls between a *Min* and *Max* range are displayed	☑	☑		☑
Display on user grid	Places the vector on a grid with custom *X spacing* and *Y spacing* and interpolates their length based on neighbours	☑	☑		☑
Head options	*Length* and *Width* of the arrow head, as a percentage of its shaft length	☑			
Arrow length	• **Defined by Min and Max**: You specify the minimum and maximum length for the arrows, QGIS will interpolate their size	☑			

Rendering

In the tab Rendering tab, QGIS offers possibilities to display and customize the mesh structure. *Line width* and *Line color* can be set to represent:

- the edges for 1D meshes
- For 2D meshes:
  - *Native mesh rendering*: shows original faces and edges from the layer
  - *Triangular mesh rendering*: adds more edges and displays the faces as triangles

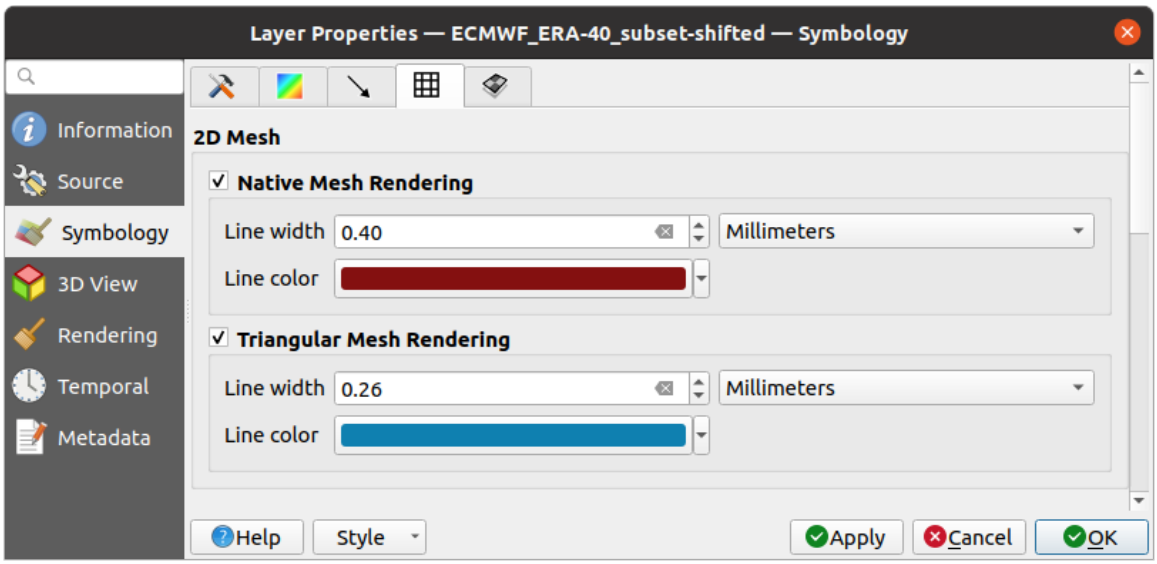


Fig. 14.9: 2D Mesh Rendering

Stacked mesh averaging method

3D layered meshes consist of multiple stacked 2D unstructured meshes each extruded in the vertical direction (`levels`) by means of a vertical coordinate. The vertices and faces have the same topology in each vertical level. Values are usually stored on the volumes that are regularly stacked over base 2d mesh. In order to visualise them on 2D canvas, you need to convert values on volumes (3d) to values on faces (2d) that can be shown in mesh layer. The Stacked mesh averaging method provides different averaging/interpolation methods to handle this.

You can select the method to derive the 2D datasets and corresponding parameters (level index, depth or height values). For each method, an example of application is shown in the dialog but you can read more on the methods at https://fvwiki.tuflow.com/Depth_Averaging_Results.

14.3.4 Labels Properties

The *Labels* tab offers you dynamic labeling of mesh vertices and native mesh faces based on geometric properties and custom expressions. This dialog can also be accessed from the *Layer Styling* panel.

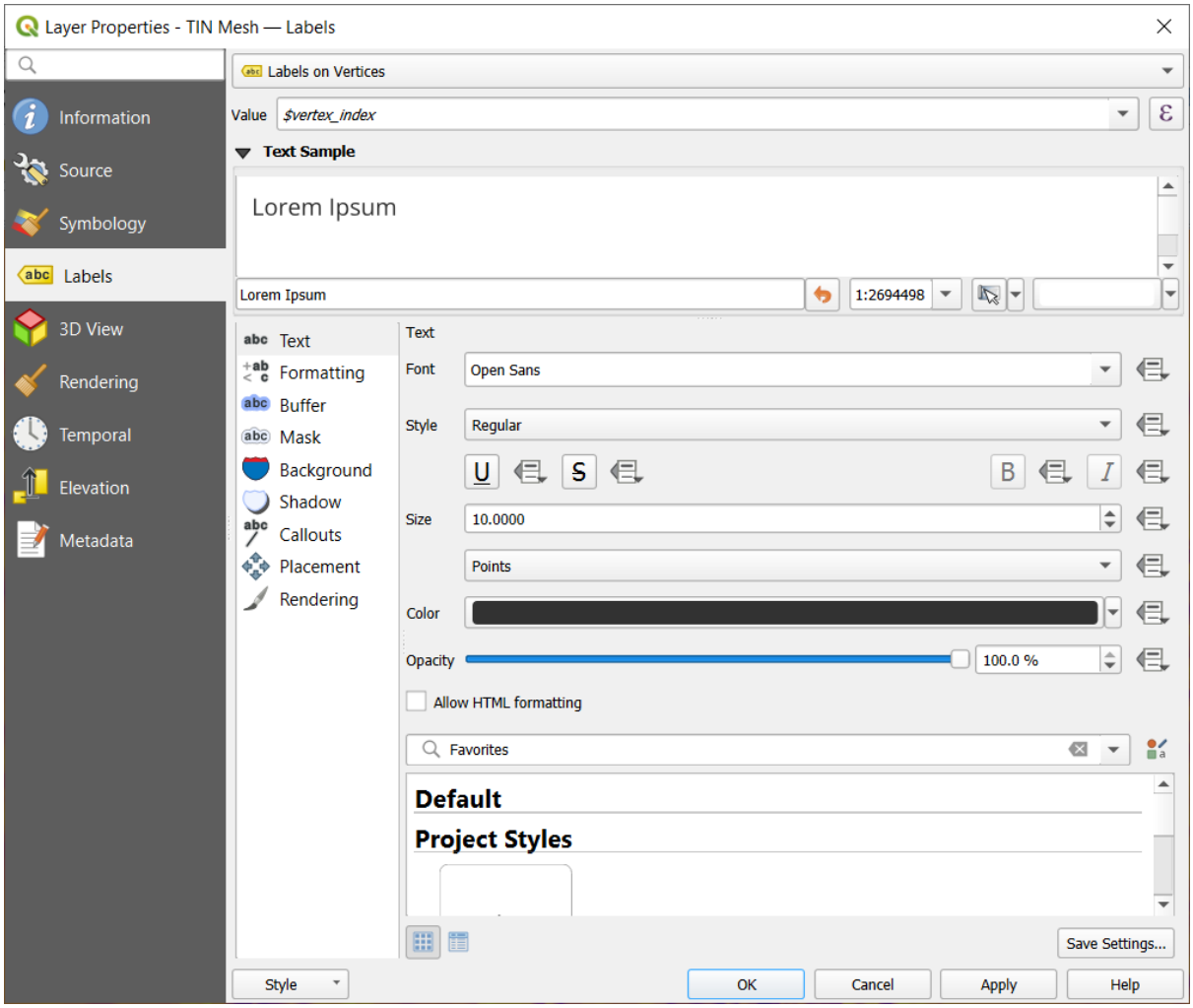


Fig. 14.10: Mesh Labels Properties

Choose the labeling method from the drop-down list. Available methods are:

- *No labels*: the default value, showing no labels from the layer.
- *Labels on vertices*: labels the mesh vertices as a point feature.
- *Labels on faces*: labels the mesh faces as a polygon feature.

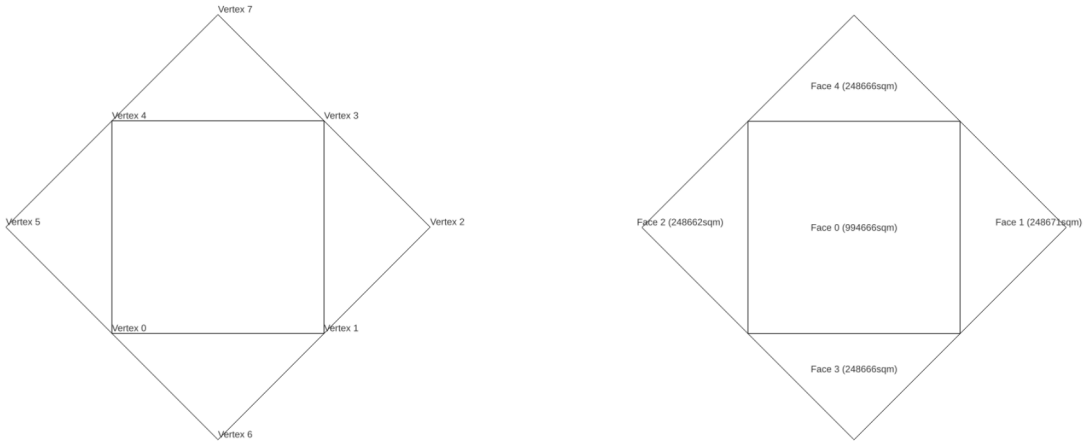


Fig. 14.11: Mesh Labeling methods

Once the labeling method has been selected, open Expression dialog and choose a *mesh function*. You can combine meshes functions with other functions from the *list* to modify the label.

Below are displayed options to customize the labels, under various tabs:

- *Text*
- *Formatting*
- *Buffer*
- *Mask*
- *Background*
- *Shadow*
- *Callouts*
- *Placement*
- *Rendering*

Description of how to set each property is exposed at *Setting a label*.

14.3.5 3D View Properties

Mesh layers can be used as *terrain in a 3D map view* based on their vertices Z values. From the *3D View* properties tab, it's also possible to render the mesh layer's dataset in the same 3D view. Therefore, the vertical component of the vertices can be set equal to dataset values (for example, level of water surface) and the texture of the mesh can be set to render other dataset values with color shading (for example velocity).

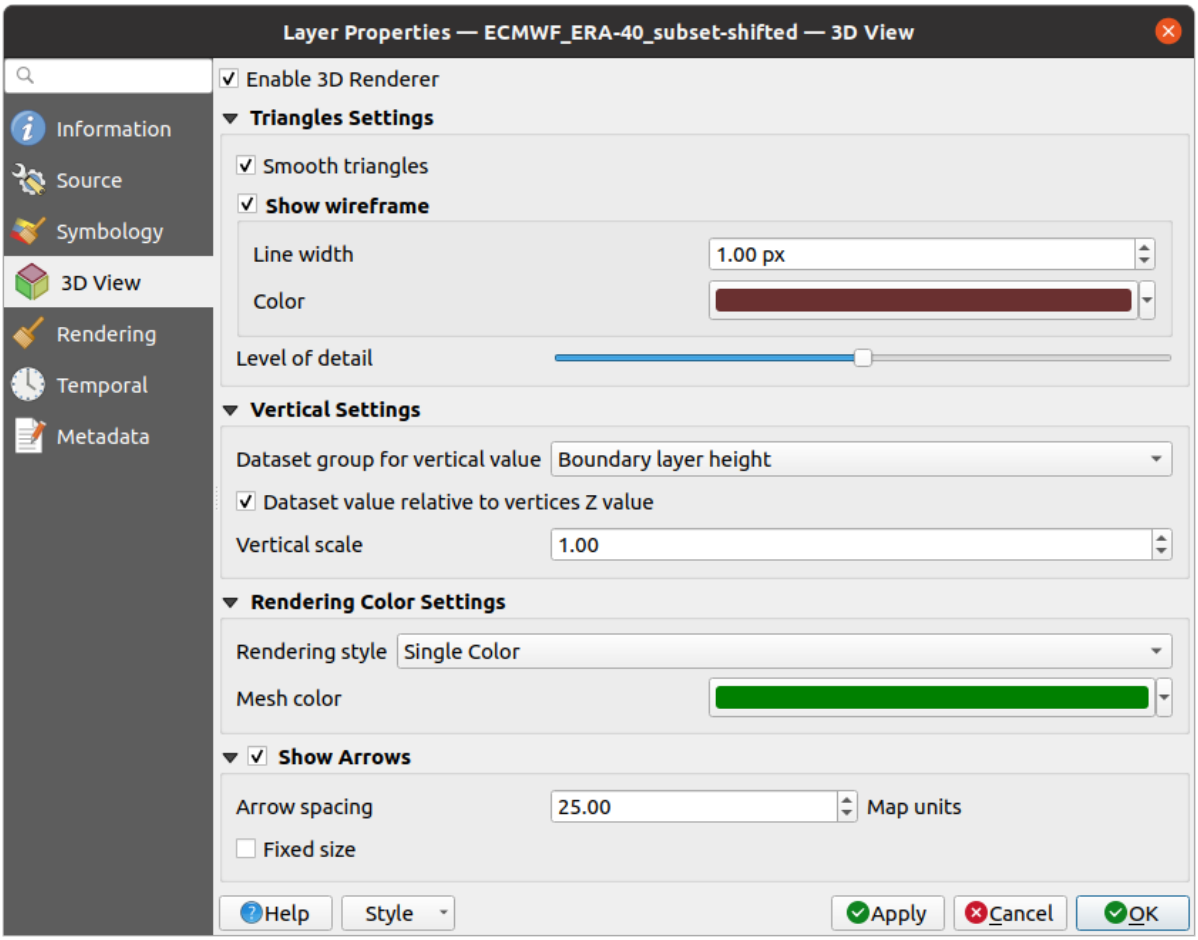


Fig. 14.12: Mesh dataset 3D properties

Check *Enable 3D Renderer* and you can edit following properties:

- Under *Triangle settings*
  - *Smooth triangles*: Angles between consecutive triangles are smoothed for a better 3D rendering
  - *Show wireframe* whose you can set the *Line width* and *Color*
  - *Level of detail*: Controls how *simplified* the mesh layer to render should be. On the far right, it is the base mesh, and the more you go left, the more the layer is simplified and is rendered with less details. This option is only available if the *Simplify mesh* option under the *Rendering* tab is activated.
- *Vertical settings* to control behavior of the vertical component of vertices of rendered triangles:
  - *Dataset group for vertical value*: the dataset group that will be used for the vertical component of the mesh
  - *Dataset value relative to vertices Z value*: whether to consider the dataset values as absolute Z coordinate or relative to the vertices native Z value
  - *Vertical scale*: the scale factor to apply to the dataset Z values
- *Rendering color settings* with a *Rendering style* that can be based on the color ramp shader set in *Contours Symbology* (*2D contour color ramp shader*) or as a *Single color* with an associated *Mesh color*
- *Show arrows*: displays arrows on mesh layer dataset 3D entity, based on the same vector dataset group used in the *vector 2D rendering*. They are displayed using the 2D color setting. It's also possible to define the *Arrow spacing* and, if it's of a *Fixed size* or scaled on magnitude. This spacing setting defines also the max size of arrows because arrows can't overlap.

14.3.6 Rendering Properties

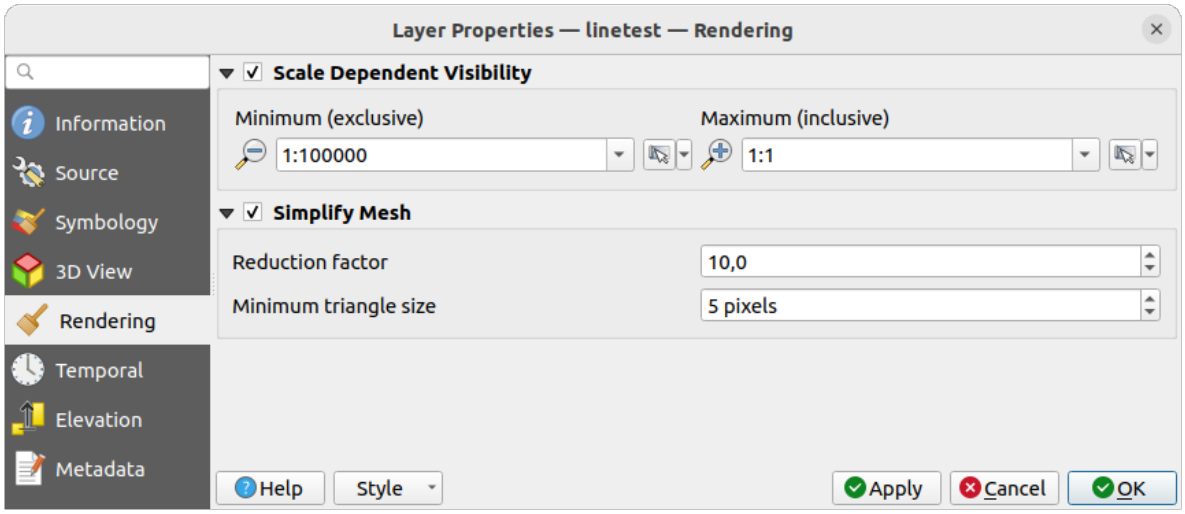


Fig. 14.13: Mesh rendering properties

Under the *Scale dependent visibility* group box, you can set the *Maximum (inclusive)* and *Minimum (exclusive)* scale, defining a range of scale in which mesh elements will be visible. Out of this range, they are hidden. The Set to current canvas scale button helps you use the current map canvas scale as boundary of the range visibility. See *Visibility Scale Selector* for more information.

Note

You can also activate scale dependent visibility on a layer from within the *Layers* panel: right-click on the layer and in the contextual menu, select *Set Layer Scale Visibility*.

As mesh layers can have millions of faces, their rendering can sometimes be very slow, especially when all the faces are displayed in the view whereas they are too small to be viewed. To speed up the rendering, you can simplify the mesh layer, resulting in one or more meshes representing different *levels of detail* and select at which level of detail you would like QGIS to render the mesh layer. Note that the simplify mesh contains only triangular faces.

From the *Rendering* tab, check *Simplify mesh* and set:

- a *Reduction factor*: Controls generation of successive levels of simplified meshes. For example, if the base mesh has 5M faces, and the reduction factor is 10, the first simplified mesh will have approximately 500 000 faces, the second 50 000 faces, the third 5000,… If a higher reduction factor leads quickly to simpler meshes (i.e. with triangles of bigger size), it produces also fewer levels of detail.
- *Minimum triangle size*: the average size (in pixels) of the triangles that is permitted to display. If the average size of the mesh is lesser than this value, the rendering of a lower level of details mesh is triggered.

14.3.7 Temporal Properties

The *Temporal* tab provides options to control the rendering of the layer over time. It allows to dynamically display temporal values of the enabled dataset groups. Such a dynamic rendering requires the *temporal navigation* to be enabled over the map canvas.

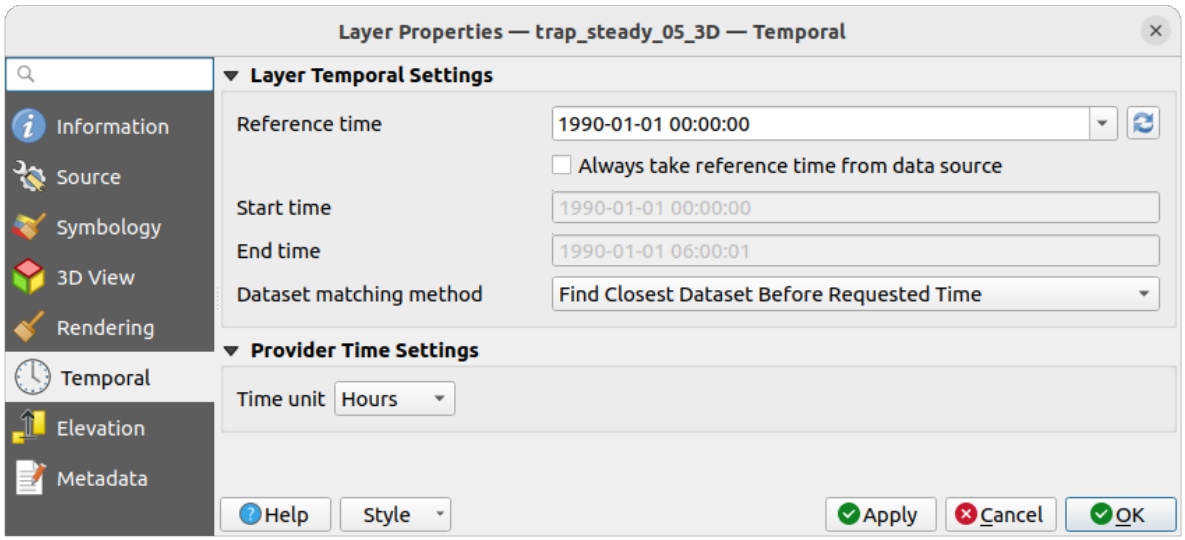


Fig. 14.14: Mesh Temporal properties

Layer temporal settings

- *Reference time* of the dataset group, as an absolute date time. By default, QGIS parses the source layer and returns the first valid reference time in the layer's dataset group. If unavailable, the value will be set by the project time range or fall back to the current date. The *Start time* and *End time* to consider are then calculated based on the internal timestamp step of the dataset.

  It is possible to set a custom *Reference time* (and then the time range), and revert the changes using the Reload from provider button. With *Always take reference time from data source* checked, you ensure that the time properties are updated from the file each time the layer is reloaded or the project reopened.

- *Dataset matching method*: determines the dataset to display at the given time. Options are *Find closest dataset before requested time* or *Find closest dataset from requested time (after or before)*.

Provider time settings

- *Time unit* extracted from the raw data, or user defined. This can be used to align the speed of the mesh layer with other layers in the project during map time navigation. Supported units are *Seconds*, *Minutes*, *Hours* and *Days*.

14.3.8 Elevation Properties

The *Elevation* tab provides options to control the layer elevation properties within a *3D map view* and *2D map view* and its appearance in the *profile tool charts*. Specifically, you can configure how heights from your dataset are interpreted:

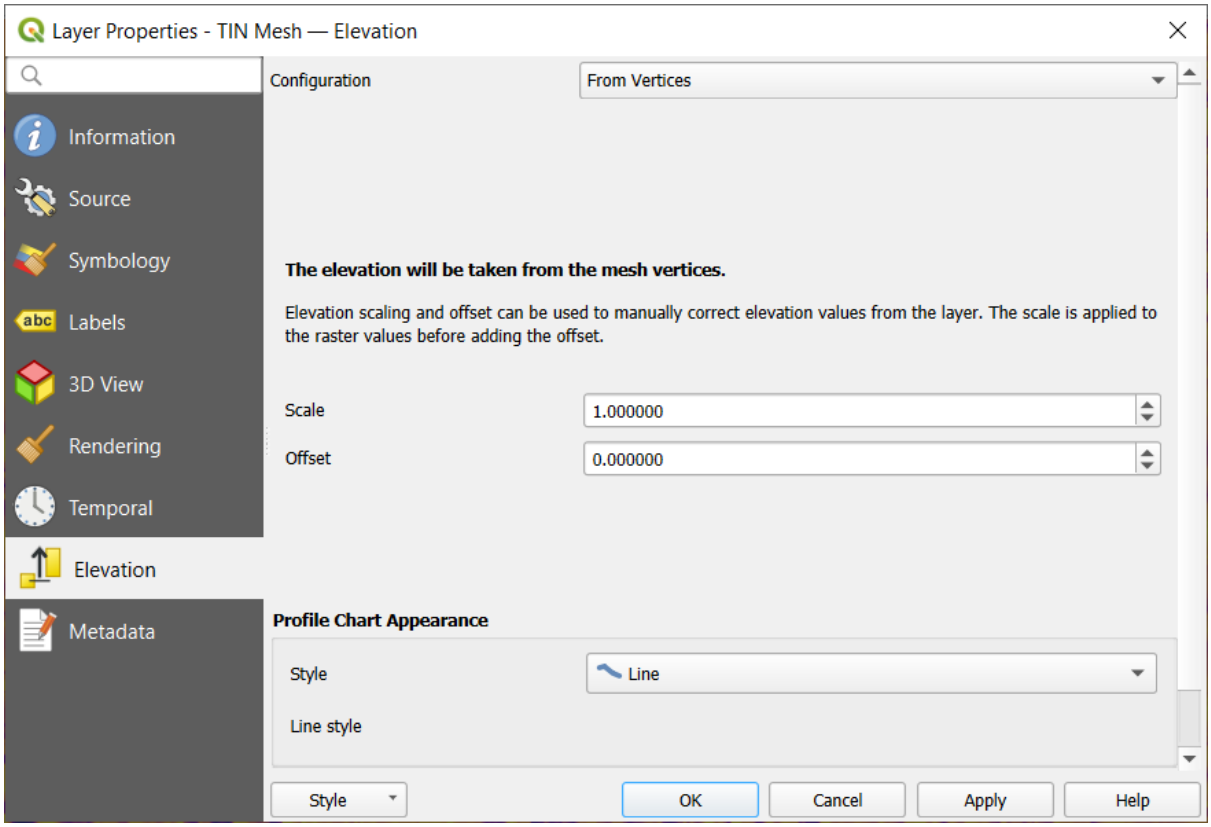


Fig. 14.15: Mesh Elevation properties

- *From vertices*: how the mesh layer vertices Z values should be interpreted as terrain elevation. You can apply a *Scale* factor and an *Offset*. This setting is available for *3D map view* and *profile tool charts*.
- *Fixed Elevation Range*: the mesh layer is linked to a fixed elevation range. This mode is applicable when a layer has either a single fixed elevation or a range (slice) of elevation values. If a range is specified, mesh values will be extruded over this range. You can set the *Lower* and *Upper* elevation range values for the layer, and specify whether the lower or upper *Limits* are inclusive or exclusive. When enabled, the layer will only be visible in *elevation filtered 2D maps* when the layer's range is included in the map's Z range.
- *Fixed Elevation Range Per Group*: each group in the mesh layer is associated with a fixed elevation range. This mode can be used when a layer has elevation data exposed through different dataset groups. This feature is exposed as a user-editable table for dataset groups with lower and upper values. You can either populate the lower and upper values manually or use an ε *Expression* to auto-fill all group values based on an expression. When enabled, the layer will be filtered in the *2D map view*, displaying only values within the map filter ranges. In *3D map view* or *profile tool charts*, the full extent of the layer will be displayed, ignoring the filtering.
- *Profile Chart Appearance*: controls the rendering of the mesh elements elevation in the profile chart. The profile *Style* can be set as:
  - a *Line* with a specific *Line style*
  - an elevation surface rendered using a fill symbol either above (*Fill above*) or below (*Fill below*) the elevation curve line. The surface symbology is represented using:
    - a *Fill style*
    - and a *Limit*: the maximum (respectively minimum) altitude determining how high the fill surface will be

14.3.9 Metadata Properties

The  *Metadata* tab provides you with options to create and edit a metadata report on your layer. See *Metadata* for more information.

14.4 Editing a mesh layer

QGIS allows to *create a mesh layer* from scratch or based on an existing one. You can create/modify the geometries of the new layer whom you can assign datasets afterwards. It's also possible to edit an existing mesh layer. Because the editing operation requires a frames-only layer, you will be asked to either remove any associated datasets first (make sure you have them available if they still are necessary) or create a copy (only geometries) of the layer.

> **Note**
>
> QGIS does not allow to digitize edges on mesh layers. Only vertices and faces are mesh elements that can be created. Also not all supported mesh formats can be edited in QGIS (see permissions).

14.4.1 Overview of the mesh digitizing tools

To interact with or edit a base mesh layer element, following tools are available.

Table 14.4: Tools for mesh digitizing

Label	Purpose	Location
 Current Edits	Access to save, rollback or cancel changes in all or selected layers simultaneously	*Digitizing* toolbar
 Toggle to Edit	Turn on/off the layer in edit mode	*Digitizing* toolbar
 Save Edits	Save changes done to the layer	*Digitizing* toolbar
 Undo	Undo the last change(s) - `Ctrl+Z`	*Digitizing* toolbar
 Redo	Redo the last undone action(s) - `Ctrl+Shift+Z`	*Digitizing* toolbar
 Enable Advanced Digitizing tools	Turn on/off the *Advanced Digitizing Panel*	*Advanced Digitizing* toolbar
 Reindex Faces and Vertices	Recreate index and renumber the mesh elements for optimization	*Mesh* menu
 Digitize Mesh Elements	Select/Create vertices and faces	*Mesh Digitizing* toolbar
 Select Mesh Elements by Polygon	Select vertices and faces overlapped by a drawn polygon	*Mesh Digitizing* toolbar
 Select Mesh Elements by Expressio	Select vertices and faces using an expression	*Mesh Digitizing* toolbar
 Transform Vertices Coordinates	Modify coordinates of a selection of vertices	*Mesh Digitizing* toolbar
 Force by Selected Geometries	Split faces and constrain Z value using a linear geometry	*Mesh Digitizing* toolbar

14.4.2 Exploring the Z value assignment logic

When a mesh layer is turned into edit mode, a *Vertex Z value* widget opens at the top right of the map canvas. By default, its value corresponds to the *Default Z value* set in *Settings* ► *Options* ► *Digitizing* tab. When there are selected vertices, the widget displays the average Z value of the selected vertices.

During editing, the *Vertex Z value* is assigned to new vertices. It is also possible to set a custom value: edit the widget, press `Enter` and you will override the default value and make use of this new value in the digitizing process. Click the  icon in the widget to reset its value to the Options default value.

Rules of assignment

This is configured in the drop-down menu of the *Digitize mesh elements* tool as *New vertex Z value* option. Following methods are available:

Table 14.5: Z value assignment methods

Method	Description
Project terrain	Z value is always taken or calculated from the *project reference terrain*. • If it is **Flat terrain**, the value corresponds to the *Terrain height* set in the project properties. • If it is a **Mesh** layer, the value is interpolated from the surrounding vertices of the mesh. • If it is a **DEM (raster layer)**, the value is assigned from the raster pixel value. • If there is no terrain under the new vertex, the value falls back to the *Default Z value* set in *digitizing options*.
Z widget	Z value is taken from the *Vertex Z value* widget or the *Advanced Digitizing Panel Z* widget (if it is in Locked state).
Prefer mesh, then Z widget	Interpolates the Z value from the active mesh layer if the vertex is on an edge or face, otherwise falls back to the *Vertex Z value* or *Advanced Digitizing Panel Z* widget (if it is in Locked state).
Prefer mesh, then terrain	Interpolates the Z value from the active mesh layer if the vertex is on the edge or face, otherwise uses the project reference terrain value.

The following detailed logic describes the behavior of the **Prefer mesh, then Z widget** strategy.

Table 14.6: Matrix of Z value assignment to new vertex

Vertex creation	Are there selected vertices in mesh layer?	Source of assigned value	Assigned Z Value
"Free" vertex, not connected to any face or edge of a face	No	*Vertex Z value*	Default or user defined
		Advanced Digitizing Panel (if *z* widget is in Locked state)	*z* widget if in Locked state
	Yes	*Vertex Z value*	Average of the selected vertices or user-defined
Vertex on an edge	—	Mesh layer	Interpolated from the edge's vertices
Vertex on a face	—	Mesh layer	Interpolated from the face's vertices
Vertex snapped to a 2D vector feature	—	*Vertex Z value*	Default or user defined
Vertex snapped to a 3D vector vertex	—	Vector layer	Vertex
Vertex snapped to a 3D vector segment	—	Vector layer	Interpolated along the vector segment

Note

The *Vertex Z value* widget is deactivated if the *Advanced Digitizing Panel* is enabled and no mesh element is selected. The latter's *z* widget then rules the Z value assignment.

Modifying Z value of existing vertices

To modify the Z value of vertices, the most straightforward way is:

1. Select one or many vertices. The *Vertex Z value* widget will display the average height of the selection.
2. Change the value in the widget.
3. Press `Enter`. The entered value is assigned to the vertices and becomes the default value of next vertices.

Another way to change the Z value of a vertex is to move and snap it on a vector layer feature with the Z value capability. If more than one vertex are selected, the Z value can't be changed in this way.

The *Transform mesh vertices* dialog also provides means to modify the Z value of a selection of vertices (along with their X or Y coordinates).

14.4.3 Selecting mesh elements

To select mesh elements, you can use the following tools:

- Digitize Mesh Elements to select different mesh elements, see more at *Using Digitize Mesh Elements*.
- Select Mesh Elements by Polygon to select different mesh elements by polygon, see more at *Using Select Mesh Elements by Polygon*.
- Select Mesh Elements by Expression to select different mesh elements by expression, you can choose to *Select by vertices* or *Select by faces*, see more at *Using Select Mesh Elements by Expression*.
- Select Isolated Vertices to select all vertices that are not part of any mesh face.

-  Select All Vertices to select all vertices of the mesh layer.

Using *Digitize Mesh Elements*

Activate the  Digitize Mesh Elements tool. Hover over an element and it gets highlighted, allowing you to select it.

- Click on a vertex, and it is selected.
- Click on the small square at the center of a face or an edge, and it gets selected. Connected vertices are also selected. Conversely, selecting all the vertices of an edge or a face also selects that element.
- Drag a rectangle to select overlapping elements (a selected face comes with all their vertices). Press `Alt` key if you want to select only completely contained elements.
- To add elements to a selection, press `Shift` while selecting them.
- To remove an element from the selection, press `Ctrl` and reselect it. A deselected face will also deselect all their vertices.

Using *Select Mesh Elements by Polygon*

Activate the  Select Mesh Elements by Polygon tool and:

- Draw a polygon (left-click to add vertex, `Backspace` to undo last vertex, `Esc` to abort the polygon and right-click to validate it) over the mesh geometries. Any partially overlapping vertices and faces will get selected. Press `Alt` key while drawing if you want to select only completely contained elements.
- Right-click over the geometry of a vector layer's feature, select it in the list that pops up and any partially overlapping vertices and faces of the mesh layer will get selected. Use `Alt` while drawing to select only completely contained elements.
- To add elements to a selection, press `Shift` while selecting them.
- To remove an element from the selection, press `Ctrl` while drawing over the selection polygon.

Using *Select Mesh Elements by Expression*

Another tool for mesh elements selection is  Select Mesh Elements by Expression. When pressed, the tool opens the mesh *expression selector dialog* from which you can:

1. Select the method of selection:
   - *Select by vertices*: applies the entered expression to vertices, and returns matching ones and their eventually associated edges/faces
   - *Select by faces*: applies the entered expression to faces, and returns matching ones and their associated edges/vertices
2. Write the expression of selection. Depending on the selected method, available functions in the *Meshes group* will be filtered accordingly.
3. Run the query by setting how the selection should behave and pressing:
   -  *Select*: replaces any existing selection in the layer
   -  *Add to current selection*
   -  *Remove from current selection*

14.4.4 Modifying mesh elements

Adding vertices

To add vertices to a mesh layer:

1. Press the  Digitize mesh elements button
2. A *Vertex Z value* widget is always available on the top right corner of the map canvas during digitizing. It allows setting or overriding the Z coordinate assigned to new vertices, depending on the selected *Z value assignment method*.
3. You can choose other method for assigning Z values to new vertices from the drop-down menu in the  Digitize mesh elements tool.
4. Then double-click:
   - outside a face: adds a "free vertex", that is a vertex not linked to any face. This vertex is represented by a red dot when the layer is in editing mode.
   - on the edge of existing face(s): adds a vertex on the edge, splits the touching face(s) into triangles connected to the new vertex.
   - inside a face: splits the face into triangles whose edges connect the surrounding vertices to the new vertex.

     With the *Refine neighboring faces when adding vertices* option enabled in the  Digitize mesh elements button drop-down menu, a check is applied to triangular faces that share at least one vertex with the face the new vertex is added to. If their edges do not satisfy Delaunay triangulation rules, they are flipped accordingly.

Adding faces

To add faces to a mesh layer:

1. Press the  Digitize mesh elements button
2. A *Vertex Z value* widget appears on the top right corner of the map canvas. Set this value to the Z coordinate you would like to assign to the subsequent vertices.
3. Hover over a vertex and click the small triangle that appears next to it.
4. Move the cursor to the next vertex position; you can snap to existing vertex or left-click to add a new one.
5. Proceed as above to add as many vertices as you wish for the face. Press `Backspace` button to undo the last vertex.
6. While moving the mouse, a rubberband showing the shape of the face is displayed. If it is shown in green, then the expected face is valid and you can right-click to add it to the mesh. If it is red, the face is not valid (e.g. because it self-intersects, overlaps an existing face or vertex, creates a hole, …) and can't be added. You'd need to undo some vertices and fix the geometry.
7. Press `Esc` to abort the face digitizing.
8. Right-click to validate the face.

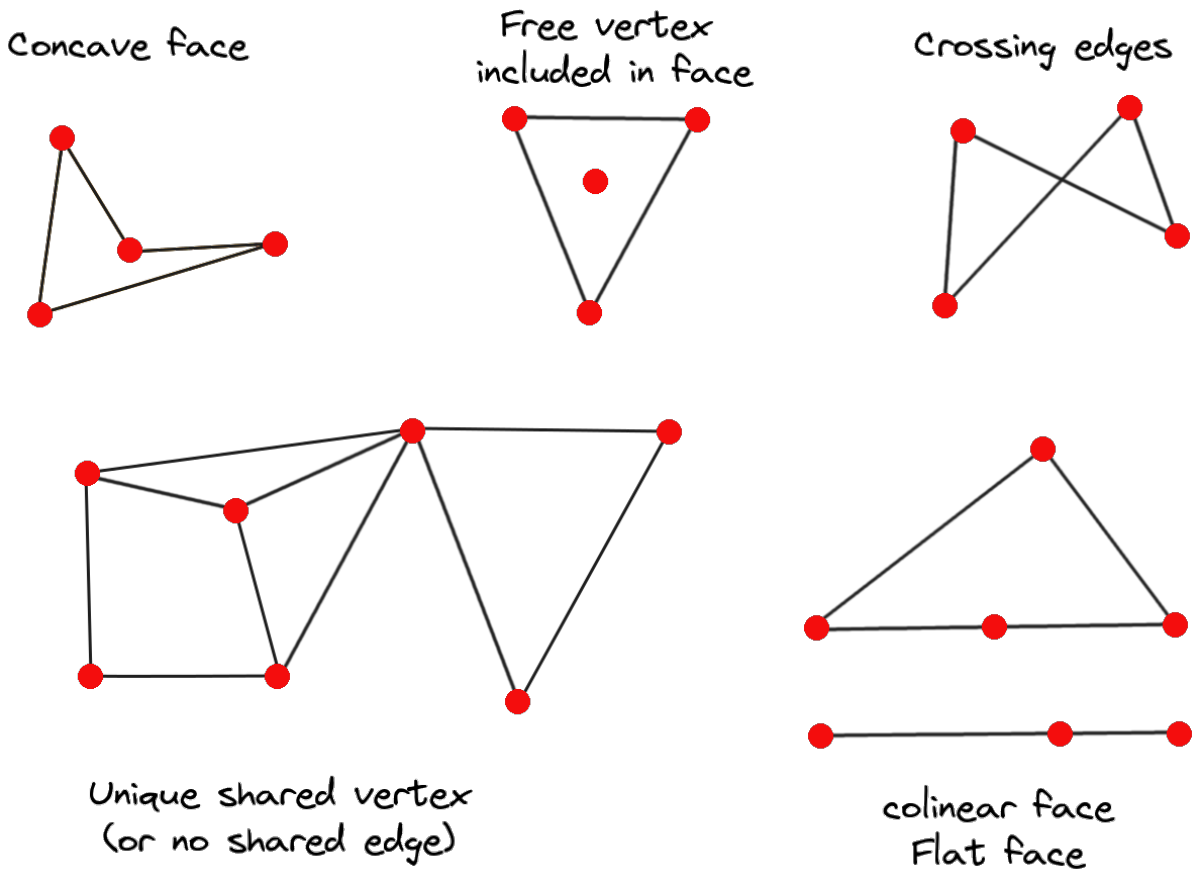


Fig. 14.16: Examples of invalid mesh

Removing mesh elements

1. *Select the target elements*
2. Enable the  Digitize mesh elements tool
3. Right-click and select:
   - *Remove Selected Vertices and Fill Hole(s)* or press `Ctrl+Del`: removes vertices and linked faces and fills the hole(s) by triangulating from the neighbor vertices
   - *Remove Selected Vertices Without Filling Hole(s)* or press `Ctrl+Shift+Del`: removes vertices and linked faces and do not fill hole(s)
   - *Remove Selected Face(s)* or press `Shift+Del`: removes faces but keeps the vertices

These options are also accessible from the contextual menu when hovering over a single item without selecting.

Moving mesh elements

To move vertices and faces of a mesh layer:

1. *Select the target elements*
2. Enable the  Digitize mesh elements tool
3. To start moving the element, click on a vertex or the centroid of a face/edge
4. Move the cursor to the target location (snapping to vector features is supported).
5. If the new location does not generate an *invalid mesh*, the moved elements appear in green. Click again to release them at this location. Faces whose vertices are all selected are translated, their neighbors are reshaped accordingly.

Transforming mesh vertices

The  Transform Vertices Coordinates tool gives a more advanced way to move vertices, by editing their X, Y and/or Z coordinates thanks to expressions.

1. Select the vertices you want to edit the coordinates
2. Press  Transform Vertices Coordinates. A dialog opens with a mention of the number of selected vertices. You can still add or remove vertices from the selection.
3. Depending on the properties you want to modify, you need to check the *X coordinate*, *Y coordinate* and/or *Z value*.
4. Then enter the target position in the box, either as a numeric value or an expression (using the  Expression dialog)
5. With the  Import Coordinates of the Selected Vertex pressed, the X, Y and Z boxes are automatically filled with its coordinates whenever a single vertex is selected. You can adjust Z value by using  *Get the Z value from the project terrain* button and in that case the Z box is deactivated. A convenient and quick way to adjust vertices individually.
6. Press *Preview Transform* to simulate the vertices new location and preview the mesh with transformation.
   - If the preview is green, transformed mesh is valid and you can apply the transformation.
   - If the preview is red, the transformed mesh is invalid and you can not apply the transformation until it is corrected.
7. Press *Apply Transform* to modify the selected coordinates for the set of vertices.

Reshaping mesh geometry

The contextual menu

1. Enable the  Digitize mesh elements
2. Select mesh item(s), or not
3. Hover over a mesh element, it gets highlighted.
4. Right-click and you can:
   - *remove the item(s)*
   - *Split Selected Face(s)* (*Split Current Face*): splits the face you are hovering over or each selected quad mesh faces into two triangles
   - *Delaunay Triangulation with Selected vertices*: builds triangular faces using selected free vertices.

- *Refine Selected Face(s)* (*Refine Current Face*): splits the face into four faces, based on vertices added at the middle of each edge (a triangle results into triangles, a quad into quads). Also triangulates adjacent faces connected to the new vertices.

The edge markers

When the Digitize mesh elements is active and you hover over an edge, the edge is highlighted and it is possible to interact with it. Depending on the context, following markers may be available:

- a **square**, at the center of the edge: click on it to select extremity vertices.
- a **cross** if the two faces on either side can be merged: click on it to delete the edge and merge the faces.
- a **circle** if the edge is between two triangles: Click on it to flip the edge, i.e. connect it instead to the two other "free" vertices of the faces

The *Force by Selected Geometries* tool

The Force by Selected Geometries tool provides advanced ways to apply break lines using lines geometry. A break line will force the mesh to have edges along the line. Note that the break line will not be considered persistent once the operation is done; resulting edges will not act as constraints anymore and can be modified like any other edge. This can be used for example to locally modify a mesh layer with accurate lines, as river banks or border of road embankments.

1. Enable the Force by Selected Geometries tool
2. Indicate the geometry to use as "forcing line"; it can be:
   - picked from a line or polygon feature in the map canvas: right-click over the vector feature and select it from the list in the contextual menu.
   - a virtual line drawn over the mesh frame: left-click to add vertices, right-click for validation. Vertices Z value is set through the *Vertex Z value* widget or the *z* widget if the *Advanced Digitizing Panel* is on. If the line is snapped to a mesh vertex or a 3D vector feature's vertex or segment, the new vertex takes the snapped element Z value.

Mesh faces that overlap the line geometry or the polygon's boundary will be affected in a way that depends on options you can set from the Force by Selected Geometries tool drop-down menu:

- *Add new vertex on intersecting edges*: with this option, a new vertex is added each time the forcing line intersect an edge. This option leads to split along the line each encountered faces.

  Without this option, encountered faces are removed and replaced by faces coming from a triangulation with only the existing vertices plus the vertices of the forcing lines (new vertices are also added on the boundary edge intersecting the forcing lines).

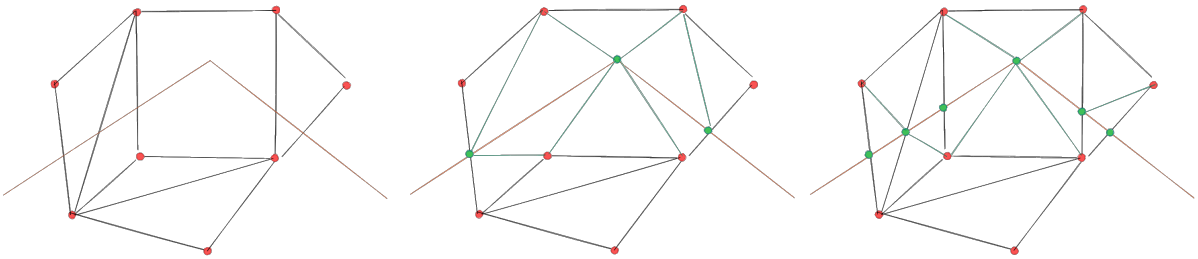


Fig. 14.17: Force Mesh using a line geometry - Results without (middle) and with (right) new vertex on edges intersection

- *Interpolate Z value from*: set how the new vertices Z value is calculated. It can be from:
  - the *Mesh* itself: the new vertices Z value is interpolated from vertices of the face they fall within
  - or the *Forcing line*: if the line is defined by a 3D vector feature or a drawn line then the new vertices Z value is derived from its geometry. In case of 2D line feature, the new vertices Z value is the *Vertex Z value*.
- *Tolerance*: when an existing mesh vertex is closer to the line than the tolerance value, do not create new vertex on the line but use the existing vertex instead. The value can be set in *Meters at Scale* or in *Map Units* (more details at *Unit Selector*).

14.4.5 Reindexing meshes

During edit, and in order to allow quick undo/redo operations, QGIS keeps empty places for deleted elements, which may lead to growing memory use and inefficient mesh structuring. The *Mesh* ► *Reindex Faces and Vertices* tool is designed to remove these holes and renumber the indices of faces and vertices so that they are continuous and somewhat reasonably ordered. This optimizes relation between faces and vertices and increases the efficiency of calculation.

Note

The *Reindex Faces and Vertices* tool saves the layer and clear the undo/redo stacks, disabling any rollback.

14.5 Mesh Calculator

The *Mesh Calculator* tool from the top *Mesh* menu allows you to perform arithmetic and logical calculations on existing dataset groups to generate a new dataset group (see Fig. 14.18).

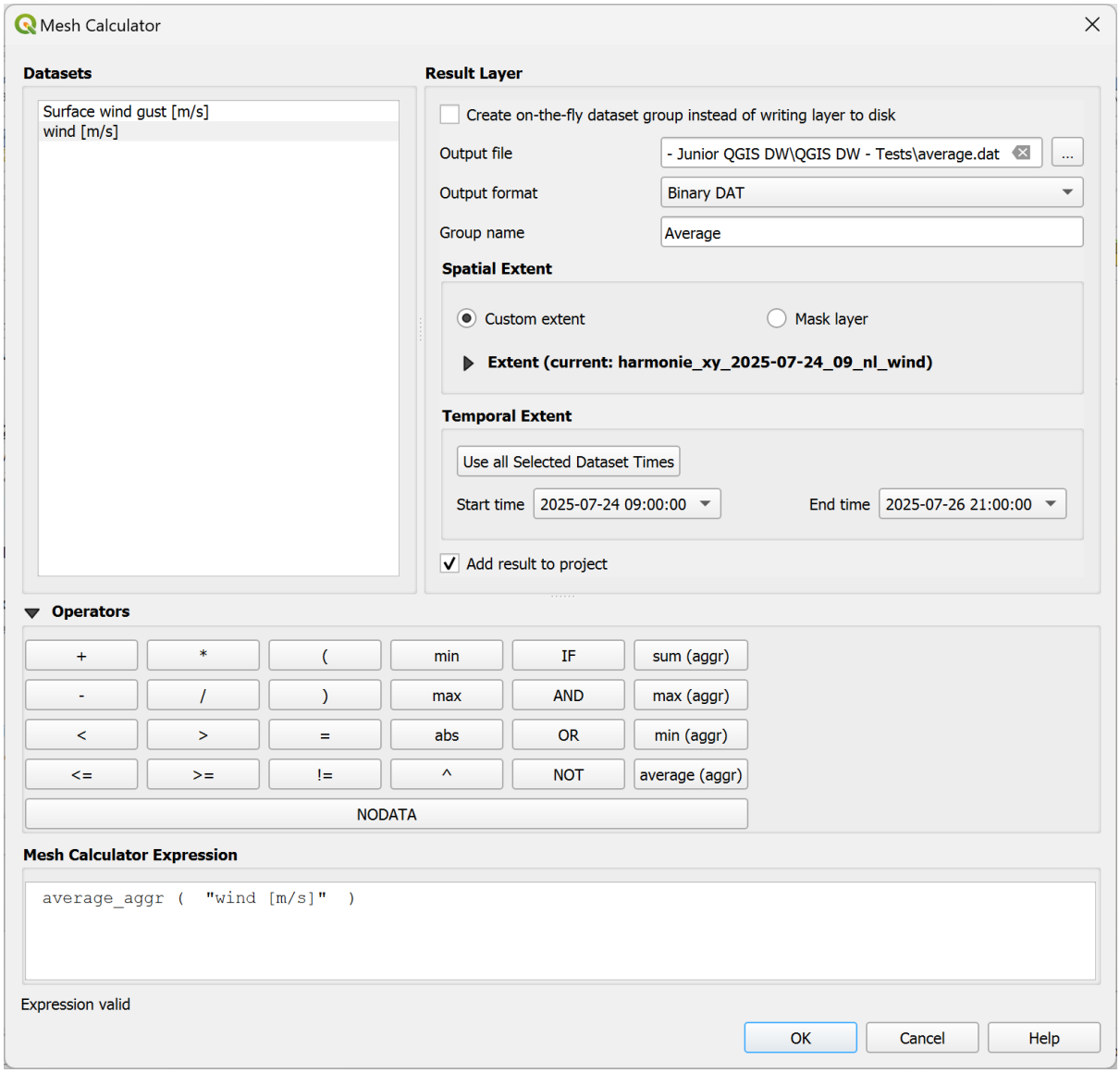


Fig. 14.18: Mesh Calculator

The *Datasets* list contains all dataset groups in the active mesh layer. To use a dataset group in an expression, double click its name in the list and it will be added to the *Mesh calculator expression* field. You can then use the operators to construct calculation expressions, or you can just type them into the box.

The *Result Layer* helps you configure properties of the output layer:

- *Create on-the-fly dataset group instead of writing layer to disk*:
  - If unchecked, the output is stored on disk as a new plain file. An *Output File* path and an *Output Format* are required.
  - If checked, a new dataset group will be added to the mesh layer. Values of the dataset group are not stored in memory but each dataset is calculated when needed with the formula entered in the mesh calculator. That virtual dataset group is saved with the project, and if needed, it can be removed or made persistent in file from the layer *Source* properties tab.

  In either case, you should provide a *Group Name* for the output dataset group.
- The *Spatial extent* to consider for calculation can be:
  - defined as a *Custom extent* using the *extent selector* widget.

  - defined by a polygon layer (*Mask layer*) of the project: the polygon features geometry are used to clip the mesh layer datasets

- The *Temporal extent* to take into account for datasets can be set with the *Start time* and *End time* options, selected from the existing dataset groups timesteps. They can also be filled using the *Use all selected dataset times* button to take the whole range.

- With the  *Add result to project* checkbox, the result layer will automatically be added to the legend area and can be visualized. Checked by default.

The *Operators* section contains all available operators. To add an operator to the mesh calculator expression box, click the appropriate button. Mathematical calculations (`+`, `-`, `*`, ...) and statistical functions (`min`, `max`, `sum (aggr)`, `average (aggr)`, ...) are available. Conditional expressions (`=`, `!=`, `<`, `>=`, `IF`, `AND`, `NOT`, ...) return either 0 for false and 1 for true, and therefore can be used with other operators and functions. The `NODATA` value can also be used in the expressions.

The *Mesh Calculator Expression* widget shows and lets you edit the expression to execute.

CHAPTER

FIFTEEN

WORKING WITH VECTOR TILES

15.1 What are Vector Tiles?

Vector tiles are packets of geographic data, packaged into pre-defined roughly-square shaped “tiles” for transfer over the web. They combine pre-rendered raster map tiles and vector map tiles. The vector tile server returns vector map data, which has been clipped to the boundaries of each tile, instead of a pre-rendered map image. The clipped tiles represent the zoom-levels of the vector tile service, derived from a pyramid approach. Using this structure, the data-transfer is reduced in comparison to un-tiled vector maps. Only data within the current map view, and at the current zoom level need to be transferred. Also, compared to a tiled raster map, data transfer is also greatly reduced, as vector data is typically much smaller than a rendered bitmap. Vector tiles do not have any styling information assigned so QGIS needs to apply a cartographic style in order to display the data.

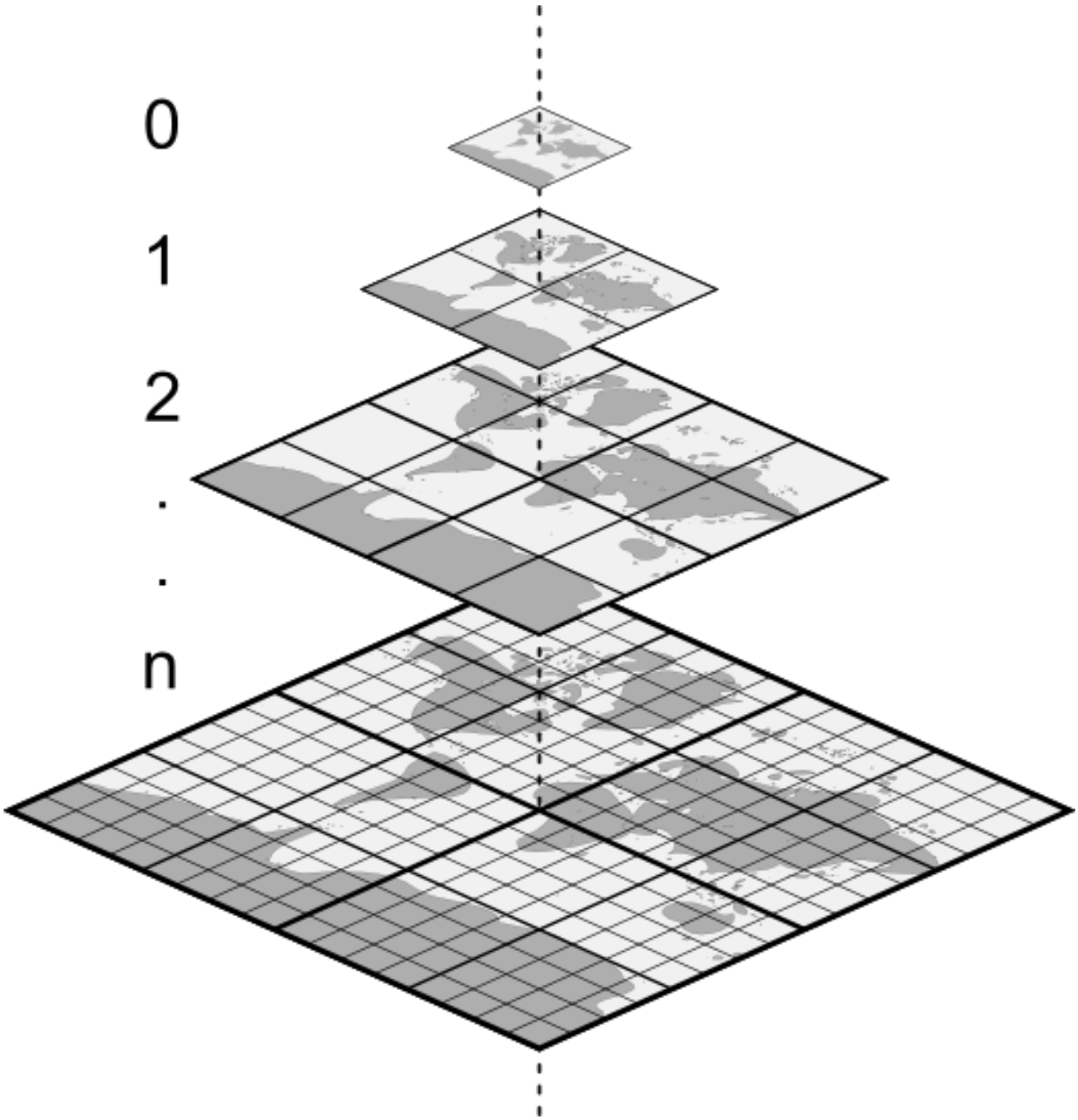


Fig. 15.1: Pyramid structure of vector tiles with zoom-levels

15.2 Supported Formats

There is support for vector tiles through:

- remote sources (HTTP/S) - with XYZ template - `type=xyz&url=http://example.com/{z}/{x}/{y}.pbf`
- local files - with XYZ template - e.g. `type=xyz&url=file:///path/to/tiles/{z}/{x}/{y}.pbf`
- local MBTiles database - e.g. `type=mbtiles&url=file:///path/to/file.mbtiles`

To load a vector tiles dataset into QGIS, use the *Vector Tile* tab in the *Data Source Manager* dialog. Read *Using Vector Tiles services* for more details.

15.3 Vector Tiles Dataset Properties

The vector tiles *Layer Properties* dialog provides the following sections:

Information	*Source*	*Symbology*[1]
Labels[1]	*Rendering*	*Metadata*

[1] Also available in the *Layer styling panel*

15.3.1 Information Properties

The *Information* tab is read-only and represents an interesting place to quickly grab summarized information and metadata on the current layer. Provided information are:

- based on the provider of the layer: name, URI, source type and path, number of zoom levels
- custom properties, used to store in the active project additional information about the layer. More properties can be created and managed using PyQGIS, specifically through the setCustomProperty() method.
- the Coordinate Reference System: name, units, method, accuracy, reference (i.e. whether it's static or dynamic)
- picked from the *filled metadata*: access, extents, links, contacts, history…

15.3.2 Source Properties

The *Source* tab displays basic information about the selected vector tile, including:

- the *Layer name* to display in the *Layers Panel*;
- the *Coordinate Reference System*: Displays the layer's *Coordinate Reference System (CRS)*. You can change the layer's CRS, by selecting a recently used one in the drop-down list or clicking on the Select CRS button (see *Coordinate Reference System Selector*). Use this process only if the layer CRS is wrong or not specified.

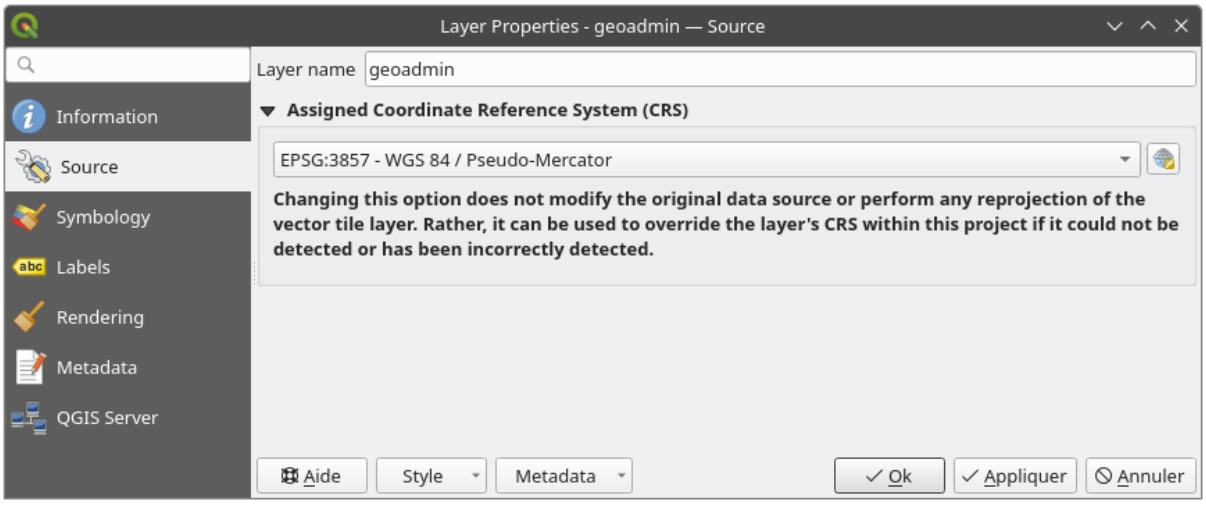


Fig. 15.2: Vector Tiles Properties - Source Dialog

15.3.3 Symbology and Labels Properties

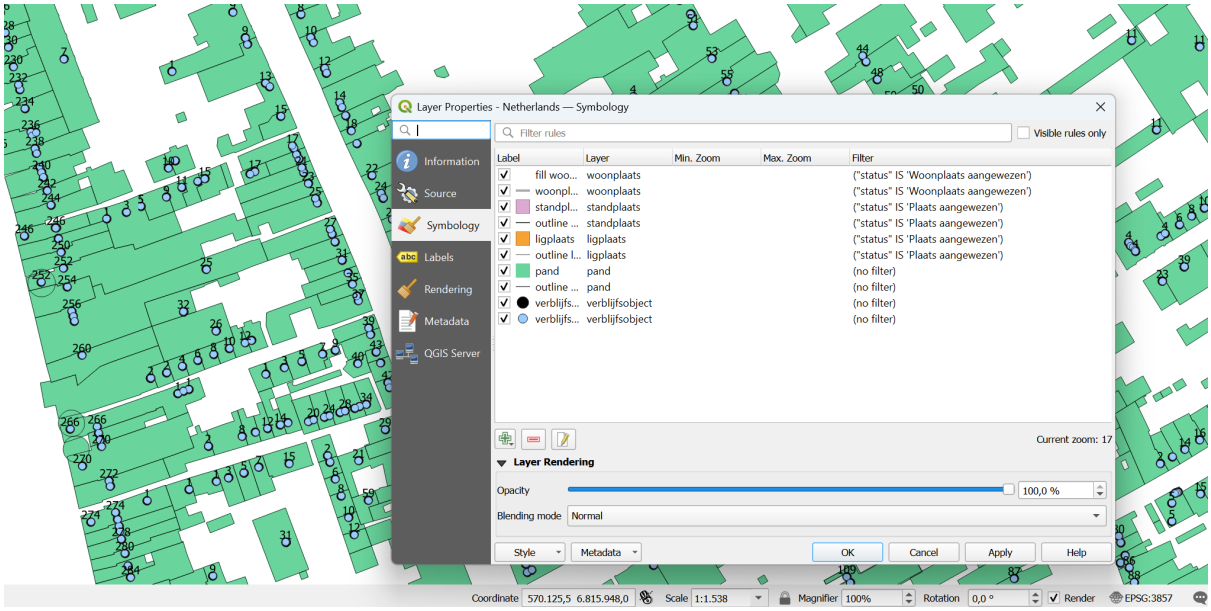


Fig. 15.3: Vector Tile Layer Symbology

Setting rules

As vector tiles consist of point, line and polygon geometries, the respective symbols are available. To apply a cartographic style (with symbology and/or labels), you can either:

- Use a *Style URL* when creating the *Vector Tiles Connection*. The symbology will be shown immediately in the *Symbology* tab after the layer is loaded in QGIS.
- Or build your own symbology and labeling in the corresponding tabs of the layer properties. By default, QGIS assigns an identical symbol to the features based on their geometry type.

In both cases, setting a style for a vector tile relies on a set of *rules* applied to the features, indicating:

- a vector *symbol* or *label*
- a *Label*, a title for comprehensive identification of the rule
- the name of a particular *Layer* the rule should apply to, if not applied to `(all layers)`
- a *Min. Zoom* and a *Max. Zoom*, for the range of display. Symbology and labeling can be dependent on the zoom level.
- a *Filter*, a QGIS expression to identify the features to apply the style to

Each rule is added pressing the Add rule button and selecting the type of symbols (*Marker*, *Line*, *Fill*) corresponding to the features geometry type. You can as well Remove selected rules or Edit current rule.

At the bottom the *Current Zoom* is shown. Check the *Visible rules only* option at the top of the dialog to filter the list of rules to only those that are visible at the current zoom level. This makes it easier to work with complex vector styling and to locate troublesome rules. The *Filter rules* text box also helps you easily find a rule, by searching the *Label*, *Layer* and *Filter* fields.

In Fig. 15.3 we set up style for the OpenStreetMap `landuse` layer. For better visibility most of the rules are deselected.

Layer rendering

From the *Symbology* tab, you can also set some options that invariably act on all features of the layer:

- *Opacity*: You can make the underlying layer in the map canvas visible with this tool. Use the slider to adapt the visibility of your vector layer to your needs. You can also make a precise definition of the percentage of visibility in the menu beside the slider.
- *Blending mode*: You can achieve special rendering effects with these tools that you may previously only know from graphics programs. The pixels of your overlaying and underlaying layers are mixed through the settings described in *Blending Modes*.

Styles

Available at the bottom of most of the tabs, the *Styles* ► menu provides shortcuts to save, load, create, switch styles to apply to the vector tiles. Vector tiles can have their style saved from QGIS as *QML* files and they can be imported as:

- *QML* files (*QML - The QGIS Style File Format*)
- *MapBox GL Json* style configuration files

More details at *Save and Share Layer Properties*.

15.3.4 Rendering Properties

Under *Scale dependent visibility*, you can set the *Maximum (inclusive)* and *Minimum (exclusive)* scales, defining a range of scales in which features will be visible. Out of this range, they are hidden. The Set to current canvas scale button helps you use the current map canvas scale as boundary of the range visibility. See *Visibility Scale Selector* for more information.

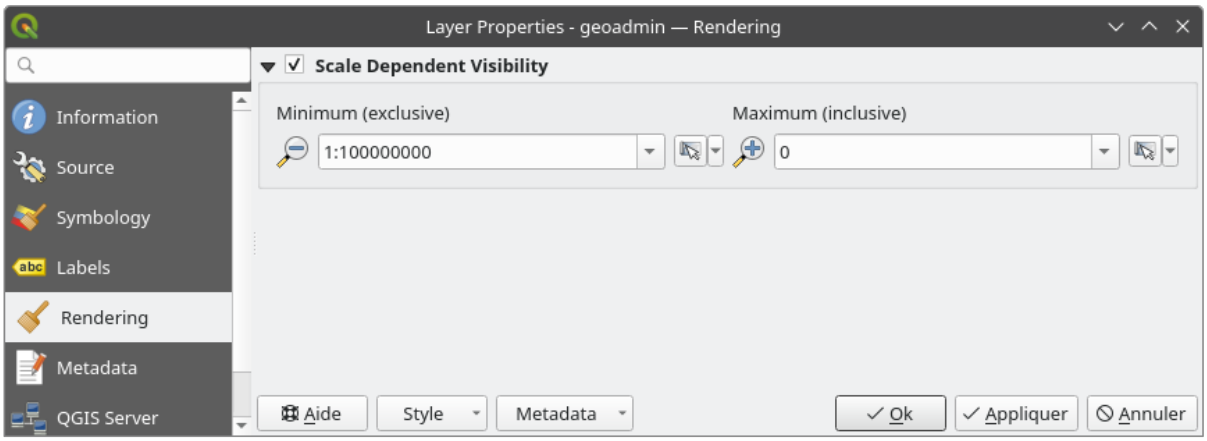


Fig. 15.4: Vector Tiles Properties - Rendering Dialog

15.3.5 Metadata Properties

The  *Metadata* tab provides you with options to create and edit a metadata report on your layer. See *Metadata* for more information.

CHAPTER

SIXTEEN

WORKING WITH POINT CLOUDS

16.1 Introduction to Point Clouds

What is A Point Cloud?

A point cloud is a three-dimensional image of a space made up of many individual of data points (up to billions, even trillions). Each of the points has an x, y and z coordinate. Depending on the capture method, point clouds usually also have additional attributes that came from the capture, such as color values or intensity. These attributes can be used, for instance, to display point clouds in different colors. In QGIS, a point cloud can be used to generate a three-dimensional image of the landscape (or of another space).

Supported Formats

QGIS supports the data formats Entwine Point Tile (EPT) and LAS/LAZ. To work with point clouds, QGIS always saves the data in EPT. EPT is a storage format that consists of several files stored in a common folder. To allow quick access to the data, EPT uses indexing. For more information on the EPT format, see entwine homepage

If the data is in LAS or LAZ format, QGIS will convert it to EPT when it is loaded for the first time. Depending on the size of the file, this may take some time. In this process, a subfolder is created in the folder in which the LAS/LAZ file is located according to the scheme `ept_` + `name_LAS/LAZ_file`. If such a subfolder already exists, QGIS loads the EPT immediately (which leads to a reduced loading time).

Worth Knowing

In QGIS it is not (yet) possible to edit point clouds. If you want to manipulate your point cloud, you can use CloudCompare, an open source point cloud processing tool. Also the Point Data Abstraction Library (PDAL - similar to GDAL) offers you options to edit point clouds (PDAL is command line only).

Due to the large number of data points, it is not possible to display an attribute table of point clouds in QGIS. However, the  *Identify tool* supports point clouds, so you can display all attributes, even of a single data point.

If you want to create a new layer, in the same or another supported format, from an existing point cloud layer, see *Creating new layers from an existing layer*.

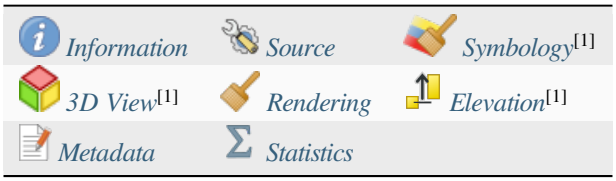
16.2 Point Clouds Properties

The *Layer Properties* dialog for a point cloud layer offers general settings for the layer and its rendering. It also provides information about the layer.

To access the *Layer Properties* dialog:

- In the *Layers* panel, double-click the layer or right-click and select *Properties…* from the context menu;
- Go to *Layer* ► *Layer Properties…* menu when the layer is selected.

The point cloud *Layer Properties* dialog provides the following sections:



[1] Also available in the *Layer styling panel*

Note

Most of the properties of a point cloud layer can be saved to or loaded from a `.qml` file using the *Style* menu at the bottom of the properties dialog. More details at *Save and Share Layer Properties*

16.2.1 Information Properties

The *Information* tab is read-only and represents an interesting place to quickly grab summarized information and metadata on the current layer. Provided information are:

- General such as name in the project, source path, last save time and size, the used provider
- custom properties, used to store in the active project additional information about the layer. Default custom properties may include *layer notes*. More properties can be created and managed using PyQGIS, specifically through the setCustomProperty() method.
- Based on the provider of the layer: extent and number of points
- The Coordinate Reference System: name, units, method, accuracy, reference (i.e. whether it's static or dynamic)
- Metadata delivered by the provider: creation date, version, data format, scale X/Y/Z, …
- Picked from the *Metadata* tab (where they can be edited): access, extents, links, contacts, history…

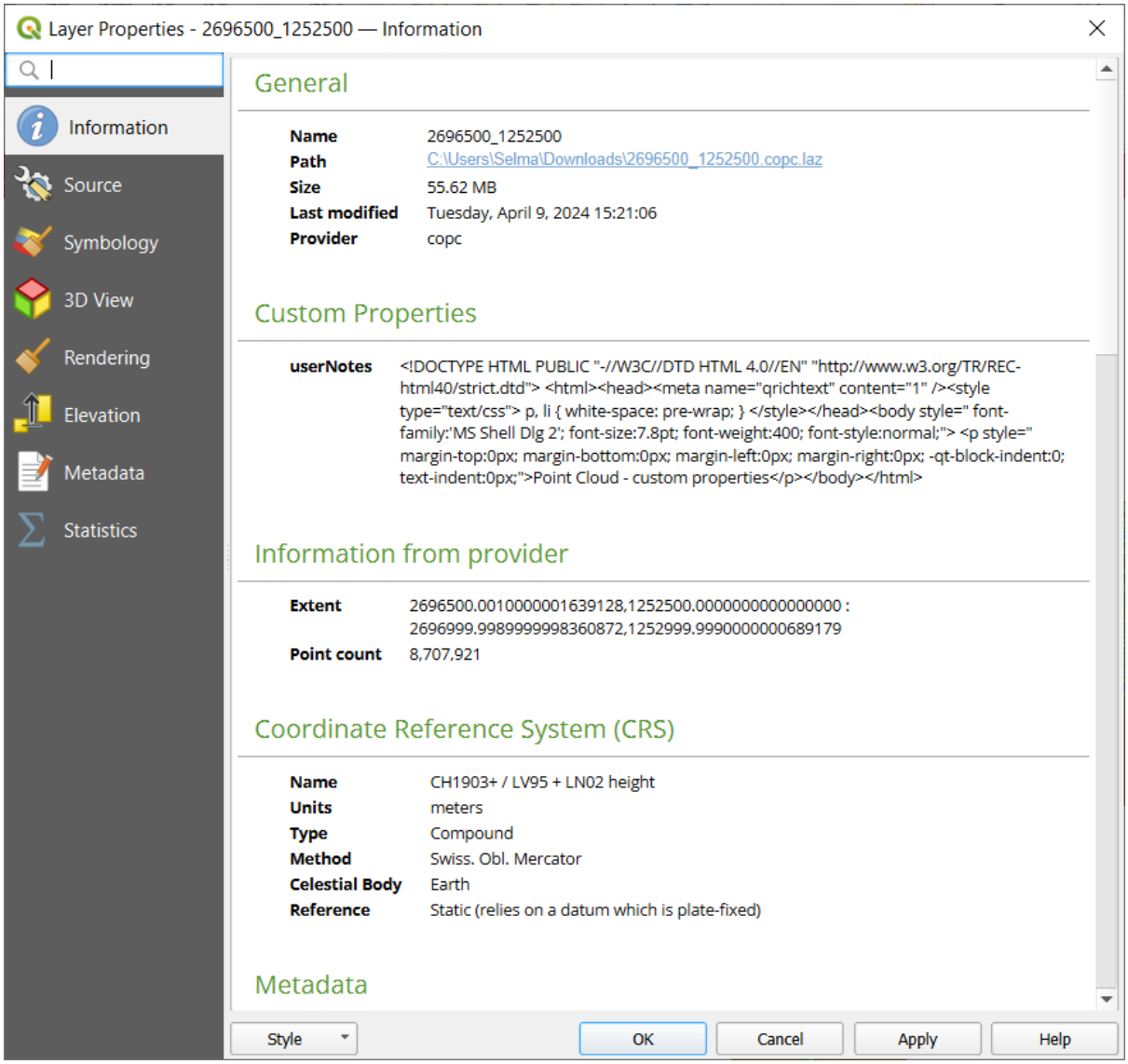


Fig. 16.1: Point cloud information tab

16.2.2 Source Properties

In the  *Source* tab you can see and edit basic information about the point cloud layer:

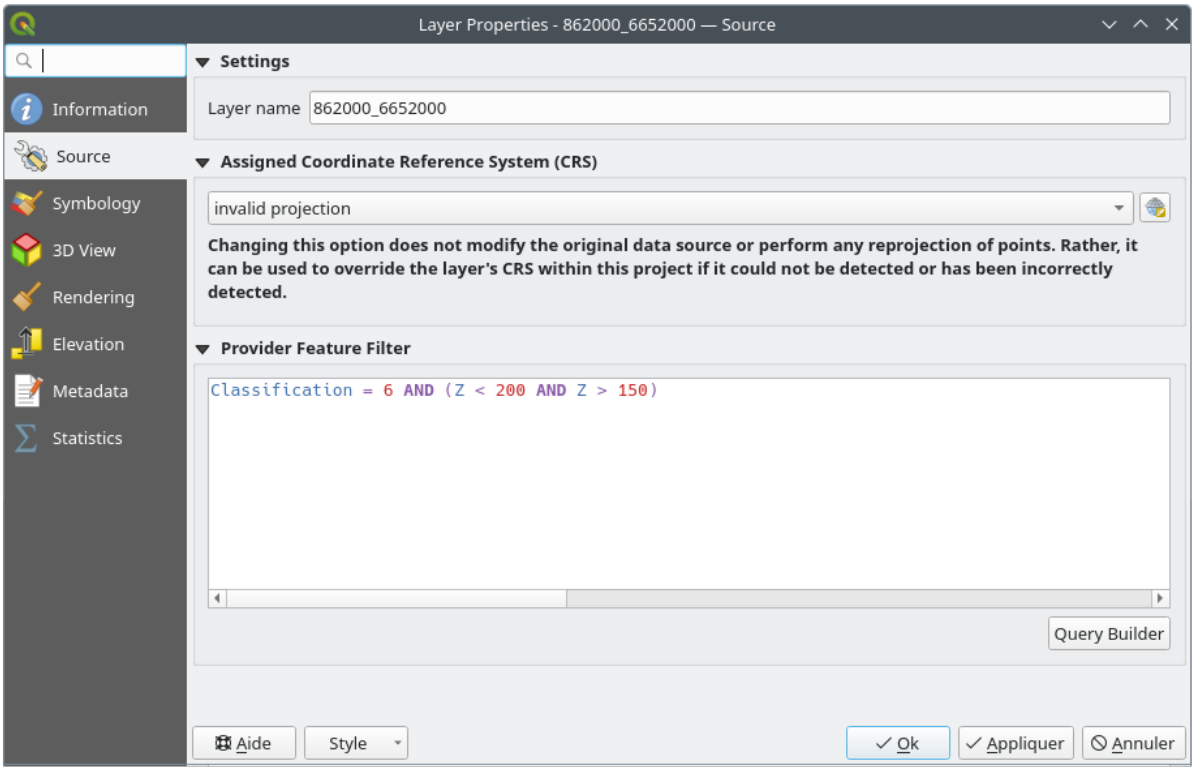


Fig. 16.2: Point cloud source tab

- *Settings*: Set a Layer name different from the layer filename that will be used to identify the layer in the project (in the Layers Panel, with expressions, in print layout legend,…)
- *Assigned Coordinate Reference System (CRS)*: Here you can change the layer's assigned *Coordinate Reference System*, selecting a recently used one in the drop-down list or clicking on set Projection Select CRS button (see *Coordinate Reference System Selector*). Use this process only if the CRS applied to the layer is a wrong one or if none was applied.
- *Provider Feature Filter*: allows to restrict data to use in the current project to a subset, using functions and expression at the PDAL data provider level. Press the *Query Builder* button at the bottom to start setting the filter.

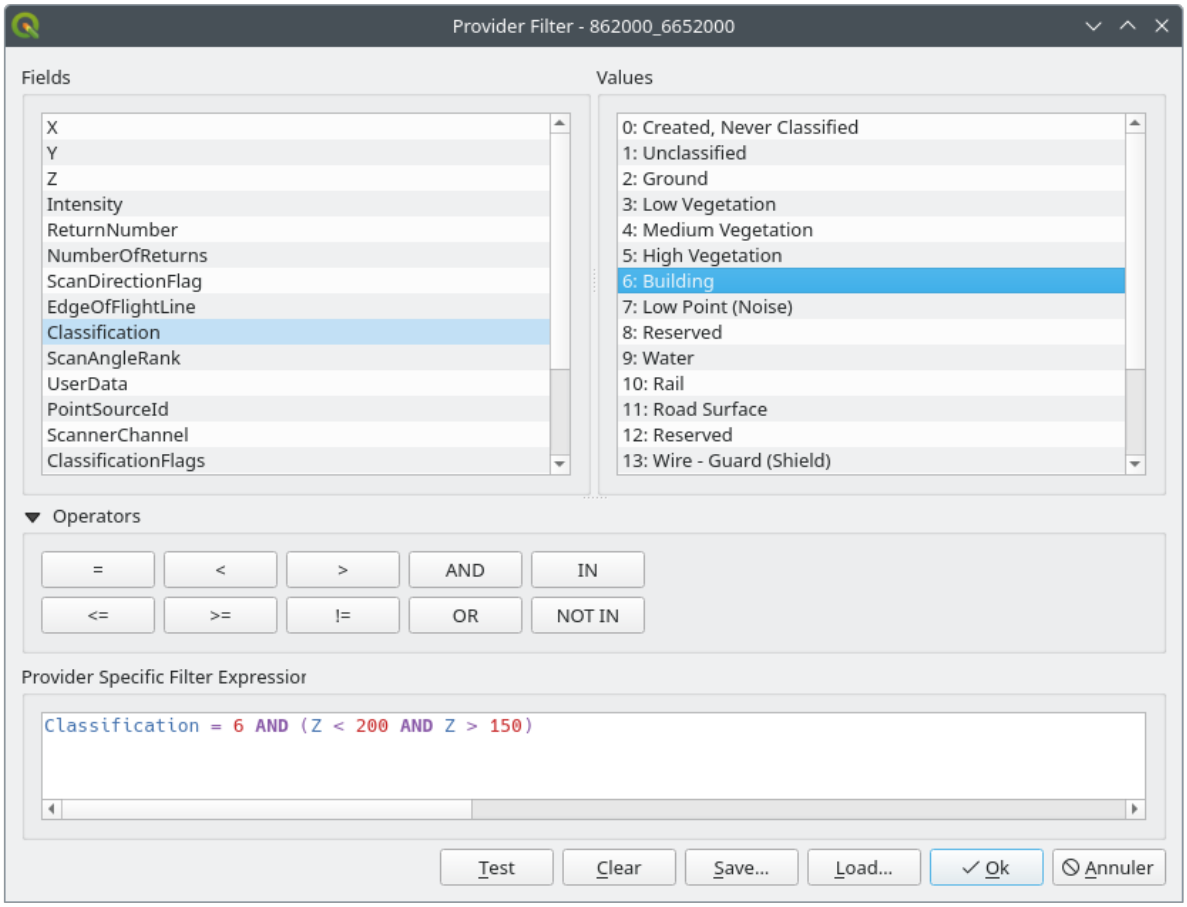


Fig. 16.3: Filtering point cloud layer to buildings at specific altitude

In the bottom part of the dialog, you can type a *Provider specific filter expression*. Such an expression can be built thanks to:

- *Fields*: the list contains all the attributes of the point cloud layer. To add an attribute to the expression, double-click its name or just type it into the text box.
- *Values*: frame lists values or statistics of the currently selected attribute, as a `key:value` pair. To add a value to the expression field, double click its name in the list: depending on the type of attribute, either the key or the value will be added to the expression. Or type the value in the expression text box.
- *Operators*: this toolbar contains all usable operators. To add an operator to the expression field, click the appropriate button. Relational operators (= , > , …) and logical operators (`AND`, `OR`, …) are available.

The *Test* button helps you check the syntax of your query. Use the *Clear* button to wipe the query and revert the layer to its original state (i.e., fully load all the points in the layer). It is possible to *Save…* the query as a `.QQF` file, or *Load…* the query from a file into the dialog.

When a filter is applied, QGIS treats the resulting subset as if it were the entire layer. For example if you applied the *filter above* for filtering buildings, you can not e.g. display, query, save or edit points that are of vegetation classification because they are not part of the subset.

Tip

Filtered layers are indicated in the Layers Panel

In the *Layers* panel, a filtered layer is listed with a Filter icon next to it indicating the query used when the mouse hovers over the icon. Double-click the icon to open the *Query Builder* dialog for edit. This can

also be achieved through the *Layer* ► *Filter…* menu.

16.2.3 Symbology Properties

In the  *Symbology* tab the settings for the rendering of the point cloud are made. In the upper part, the settings of the different feature renderers can be found. In the lower part, there are sections with which general settings for the entire layer can be made and which apply over feature renderers.

Feature Rendering types

There are different options for rendering point clouds that can be selected using the drop-down menu at the top of the *Symbology* tab (see Fig. 16.4):

-  *Extent Only*: Only a bounding box of the extent of the data is displayed; convenient for overviewing the data extent. As usual, the *Symbol widget* helps you configure any properties (color, stroke, opacity, sub-layers, …) you'd like for the box.
-  *Attribute by Ramp*: The data is drawn over a color gradient. See *Attribute by Ramp Renderer*
-  *RGB*: Draw the data using red, green and blue color values. See *RGB Renderer*
-  *Classification*: The data is drawn using different colors for different classes. See *Classification Renderer*

When a point cloud is loaded, QGIS follows a logic to select the best renderer:

- if the dataset contains color information (red, green, blue attributes), the RGB renderer will be used
- else if the dataset contains a `Classification` attribute, the classified renderer will be used
- else it will fall back to rendering based on Z attribute

If you do not know the attributes of the point cloud, the  *Statistics tab* provides a good overview of which attributes are contained in the point cloud and in which ranges the values are located.

For each renderer, you can improve the data display adjusting the *point symbol size* or enabling *surface triangulation*.

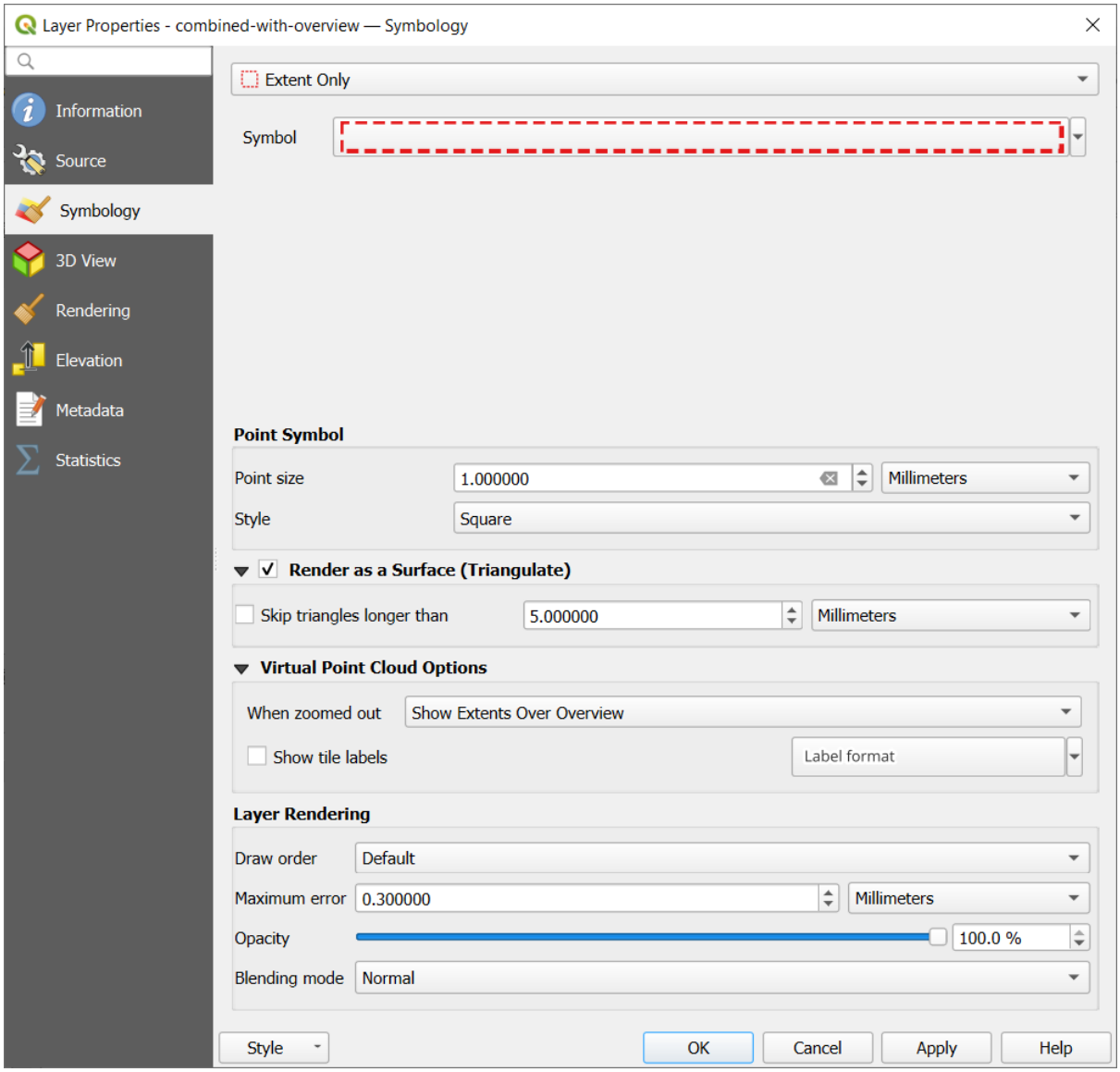


Fig. 16.4: Point cloud symbology tab

Attribute by Ramp Renderer

With *Attribute by Ramp*, the data can be displayed by numerical values over a color gradient. Such numerical values can be, for example, an existing intensity attribute or the Z-value. Depending on a minimum and a maximum value, the other values are spread to the color gradient via interpolation. The distinct values and their assignment to a certain color are called “color map” and are shown in the table. There are various setting options, which are described below the figure.

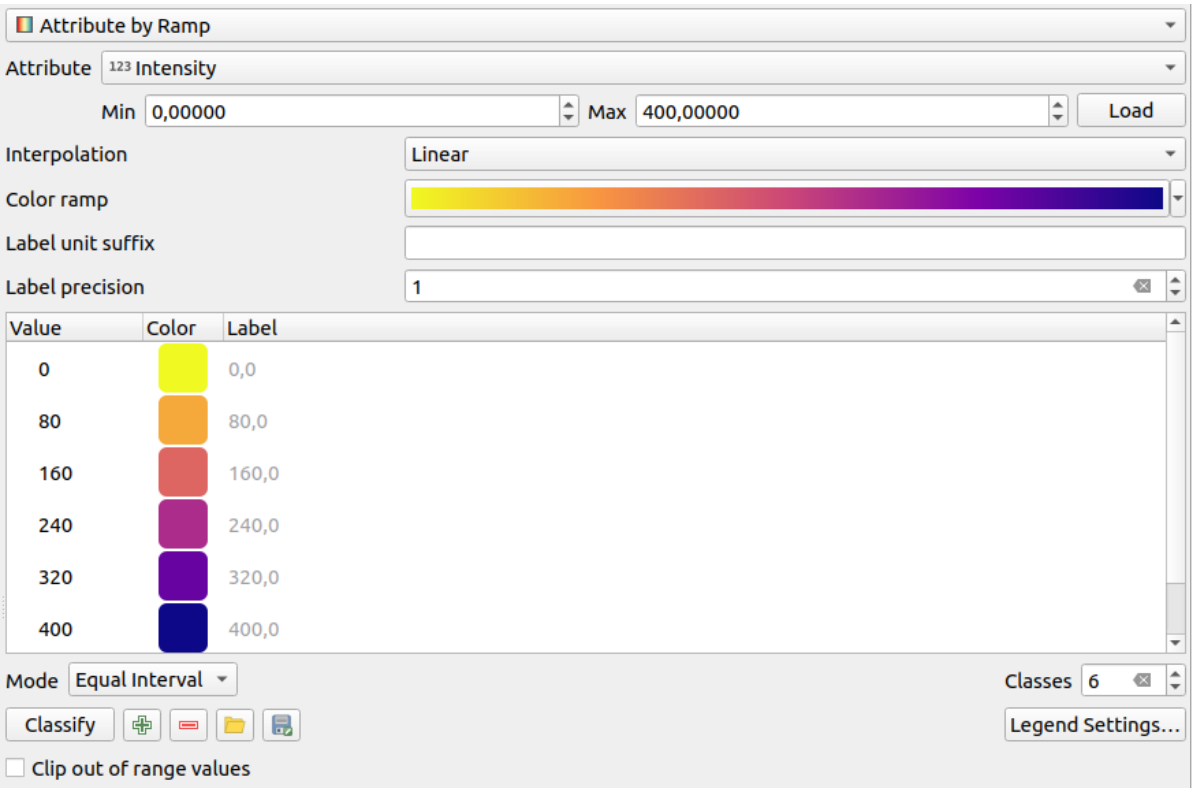


Fig. 16.5: Point cloud symbology tab: Attribute by Ramp

- *Min* and *Max* define the range that is applied to the color ramp: the *Min* value represents the left, the *Max* value the right end of the color ramp, the values in between are interpolated. By default QGIS detects the minimum and the maximum from the selected attribute but they can be modified. Once you have changed the values, you can restore the defaults by clicking on the *Load* button.
- The *Interpolation* entry defines how values are assigned their color:
  - *Discrete* (a <= symbol appears in the header of the *Value* column): The color is taken from the closest color map entry with equal or higher value
  - *Linear* The color is linearly interpolated from the color map entries above and below the pixel value, meaning that to each dataset value corresponds a unique color
  - *Exact* (a = symbol appears in the header of the *Value* column): Only pixels with value equal to a color map entry are applied a color; others are not rendered.
- The *Color ramp* widget helps you select the color ramp to assign to the dataset. As usual with *this widget*, you can create a new one and edit or save the currently selected one.
- The *Label unit suffix* adds a label after the value in the legend, and the *Label precision* controls the number of decimals to display.

The classification *Mode* helps you define how values are distributed across the classes:

- *Continuous*: Classes number and color are fetched from the color ramp stops; limits values are set following stops distribution in the color ramp (you can find more information on stops in *Setting a Color Ramp*).
- *Equal interval*: The number of classes is set by the *Classes* field at the end of the line; limits values are defined so that the classes all have the same magnitude.

The classes are determined automatically and shown in the color map table. But you can also edit these classes manually:

- Double clicking in a *Value* in the table lets you modify the class value

- Double clicking in the *Color* column opens the *Color Selector* widget, where you can select a color to apply for that value
- Double clicking in the *Label* column to modify the label of the class
- Right-clicking over selected rows in the color table shows a contextual menu to *Change Color…* and *Change Opacity…* for the selection

Below the table there are the options to restore the default classes with *Classify* or to manually Add values or Delete selected values from the table.

Since a customized color map can be very complex, there is also the option to Load an existing color map or to Save it for use in other layers (as a `txt` file).

If you have selected *Linear* for *Interpolation*, you can also configure:

- *Clip out of range values* By default, the linear method assigns the first class (respectively the last class) color to values in the dataset that are lower than the set *Min* (respectively greater than the set *Max*) value. Check this setting if you do not want to render those values.
- *Legend settings*, for display in the *Layers* panel and in the *layout legend*. Customization works the same way as with a raster layer (find more details at *Customize raster legend*).

RGB Renderer

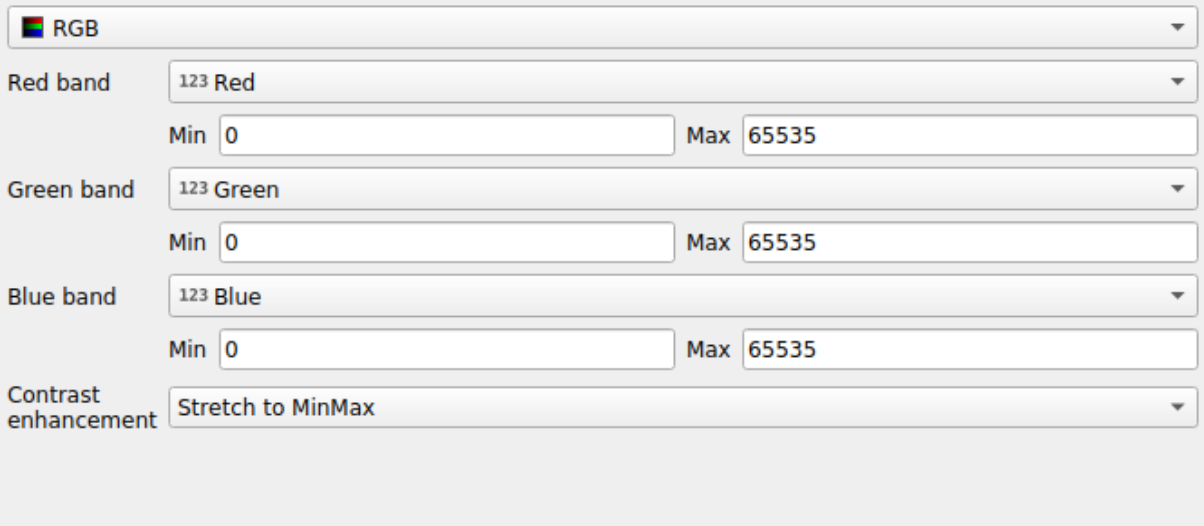


Fig. 16.6: The point cloud RGB renderer

With the *RGB* renderer, three selected attributes from the point cloud will be used as the red, green and blue component. If the attributes are named accordingly, QGIS selects them automatically and fetches *Min* and *Max* values for each band and scales the coloring accordingly. However, it is also possible to modify the values manually.

A *Contrast enhancement* method can be applied to the values: *No Enhancement*, *Stretch to MinMax*, *Stretch and Clip to MinMax* and *Clip to MinMax*

Note

The *Contrast enhancement* tool is still under development. If you have problems with it, you should use the default setting *Stretch to MinMax*.

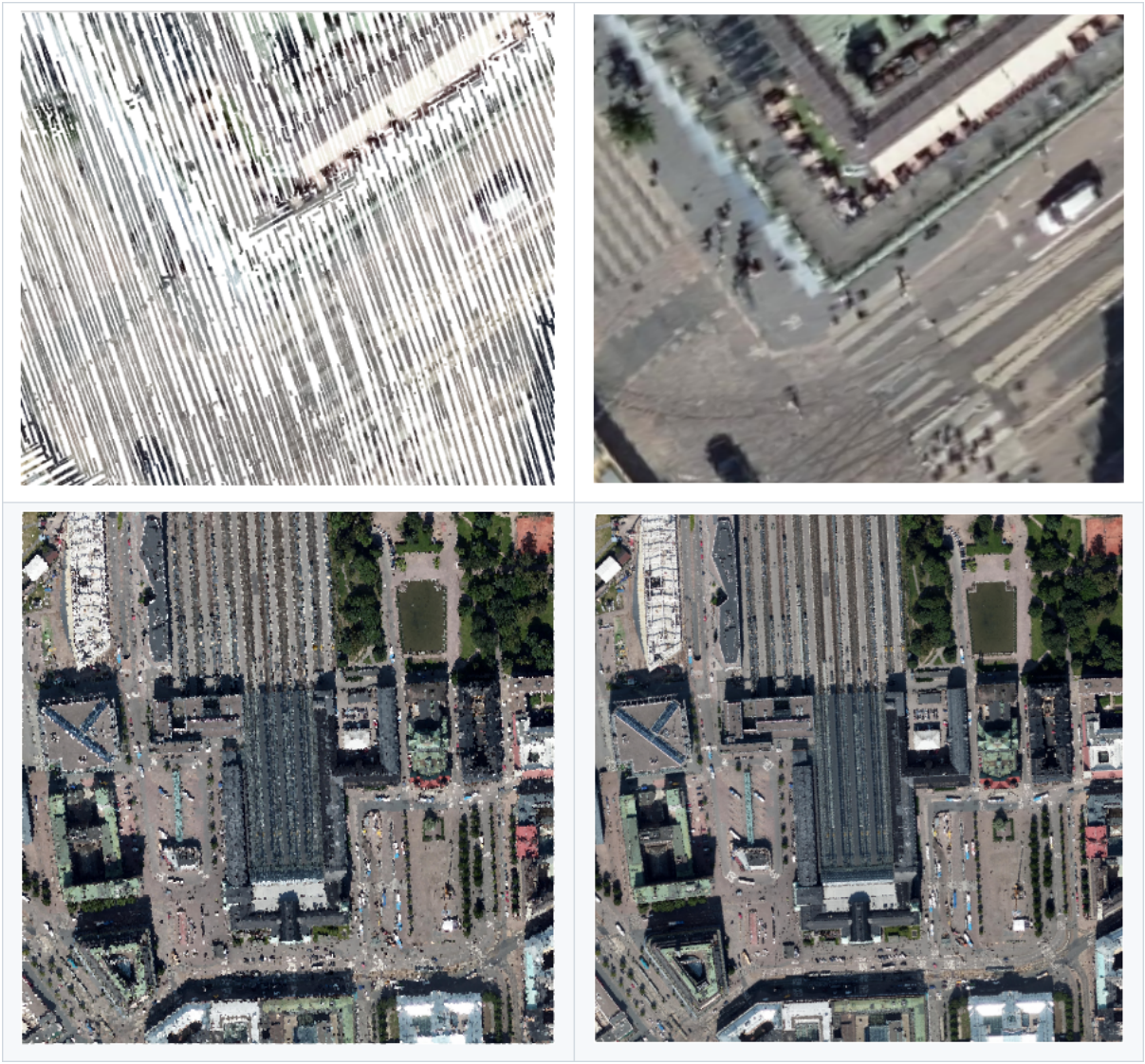


Fig. 16.7: Example of RGB renderer (left) combined with surface triangulation option (right)

Classification Renderer

In the *Classification* rendering, the point cloud is shown differentiated by color on the basis of an attribute. Any type of attribute can be used (numeric, string, …). Point cloud data often includes a field called `Classification`. This usually contains data determined automatically by post-processing, e.g. about vegetation. With *Attribute* you can select the field from the attribute table that will be used for the classification. By default, QGIS uses the definitions of the LAS specification (see table 'ASPRS Standard Point Classes' in the PDF on ASPRS home page). However, the data may deviate from this schema; in case of doubt, you have to ask the person or institution from which you received the data for the definitions.

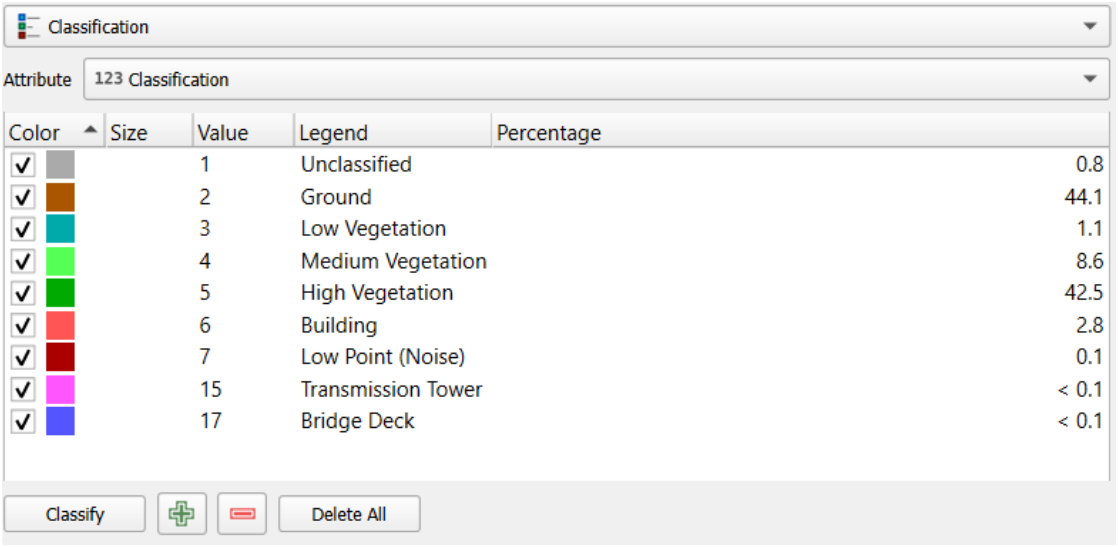


Fig. 16.8: The point cloud classification renderer

In the table all used values are displayed with the corresponding color and legend. At the beginning of each row there is a check box; if it is unchecked, this value is no longer shown on the map. With double click in the table, you can modify:

- the *Color*: the *Color Selector* widget opens
- the *Size*: assigning a size of `0` to a category will revert it to use the default *point size* set for the layer
- the *Value*
- the *Legend*
- the *Percentage* column will show you the representation of the category within the layer.

You can also right-click one or more rows to open a context menu with the options to change the color, opacity or point size.

Below the table there are buttons with which you can change the default classes generated by QGIS:

- With the *Classify* button the data can be classified automatically: all values that occur in the attributes and are not yet present in the table are added
- With Add and Delete, values can be added or removed manually
- *Delete All* removes all values from the table

Hint

In the *Layers* panel, you can right-click over a class leaf entry of a layer to quickly configure visibility of the corresponding features.

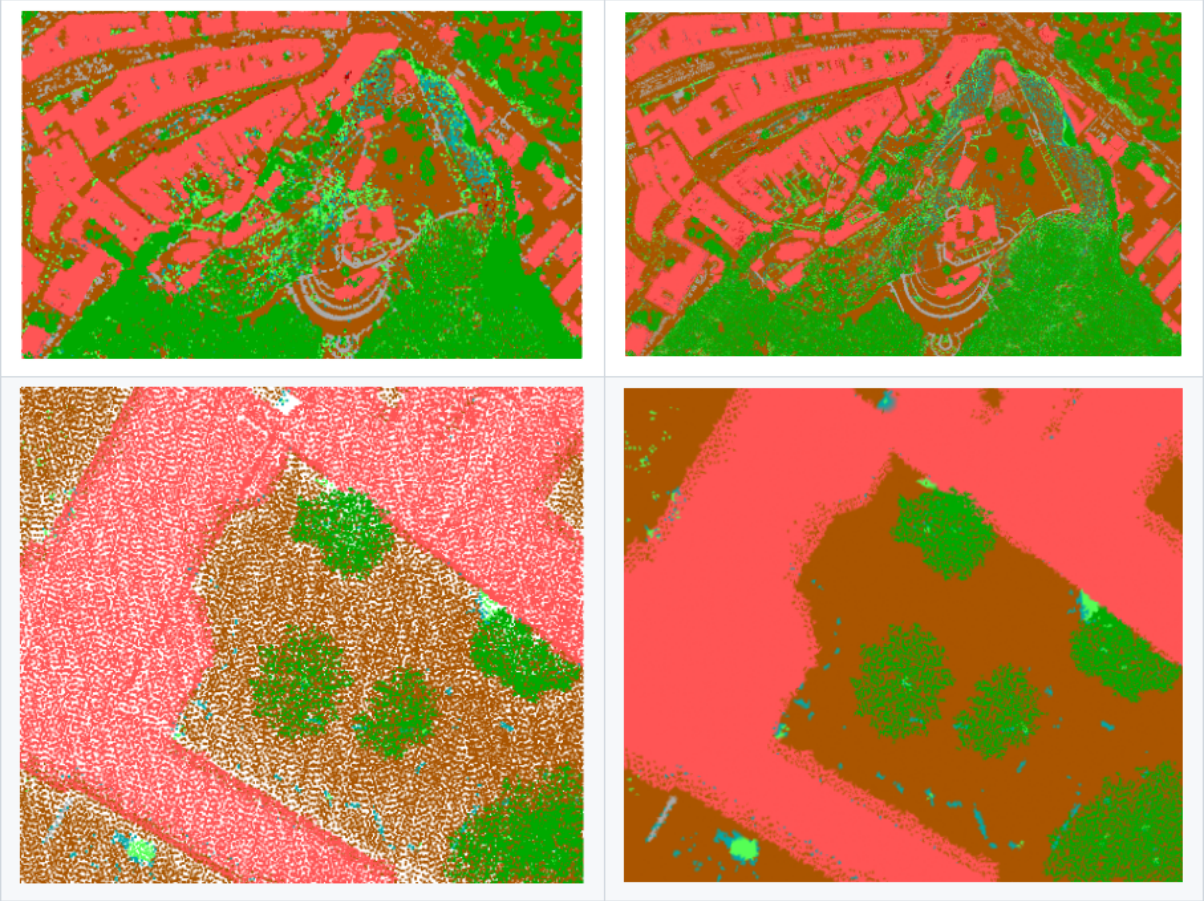


Fig. 16.9: Example of classification renderer (left) combined with surface triangulation option (right)

Point Symbol

Under *Point Symbol*, the size and the unit (e.g. millimeters, pixels, inches) with which each data point is displayed can be set. Either *Circle* or *Square* can be selected as the style for the points.

Render as a surface (Triangulate)

Check *Render as surface (Triangulate)* to enable the triangulation of the point cloud layer in the 2D view. This option allows rendering triangles instead of points. Each point keeps its color for interpolation in the triangle. You can control the horizontal length of computed triangles:

By checking the *Skip triangles longer than* option and setting up the threshold value, you can control the maximum lenght of a side of the triangles to consider in the horizontal plan. This can be particularly useful if you want to identify actual holes in the data.

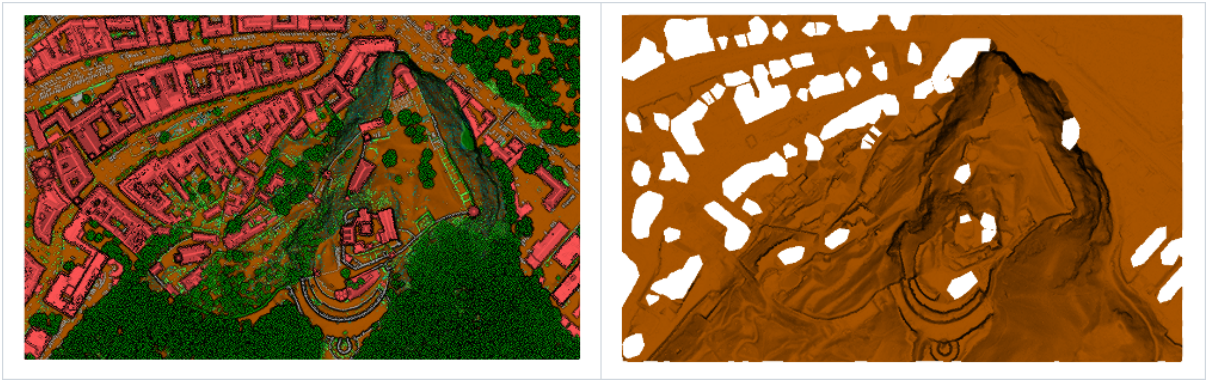


Fig. 16.10: Rendering data as a surface with map shading (left) and with map shading, filtering large triangles (right)

Virtual Point Cloud Options

The *Virtual Point Cloud Options* are available only when the layer is a *virtual point cloud (VPC)*. QGIS renders the overview of the VPC when zoomed out, if an overview is present. This provides a seamless transition from the overview to the detailed display of individual point clouds as you zoom in.

You can control how the VPC is displayed when zoomed out using the available options:

- *Show Extents Only*: Only the extents of the underlying point clouds are displayed.
- *Show Overview Only*: Only the overview is displayed (if available).
- *Show Extents Over Overview*: The extents are displayed on top of the overview.

You can also choose to *Show tile labels* to display the tile names and set the label format.

Layer Rendering

In the *Layer Rendering* section you have the following options to modify the rendering of the layer:

- *Draw order*: allows to control whether point clouds rendering order on 2d map canvas should rely on their Z value. It is possible to render :
  - with the *Default* order in which the points are stored in the layer,
  - from *Bottom to top* (points with larger Z values cover lower points giving the looks of a true ortho photo),
  - or from *Top to bottom* where the scene appears as viewed from below.
- *Maximum error*: Point clouds usually contains more points than are needed for the display. By this option you set how dense or sparse the display of the point cloud will be (this can also be understood as 'maximum allowed gap between points'). If you set a large number (e.g. 5 mm), there will be visible gaps between points. Low value (e.g. 0.1 mm) could force rendering of unnecessary amount of points, making rendering slower (different units can be selected).
- *Opacity*: You can make the underlying layer in the map canvas visible with this tool. Use the slider to adapt the visibility of your layer to your needs. You can also make a precise definition of the percentage of visibility in the menu beside the slider.
- *Blending mode*: You can achieve special rendering effects with this tool. The pixels of your overlaying and underlying layers are mixed through the settings described in *Blending Modes*.
- *Eye dome lighting*: this applies shading effects to the map canvas for a better depth rendering. Rendering quality depends on the *draw order* property; the *Default* draw order may give sub-optimal results. Following parameters can be controlled:
  - *Strength*: increases the contrast, allowing for better depth perception

  - *Distance*: represents the distance of the used pixels off the center pixel and has the effect of making edges thicker.

16.2.4 3D View Properties

In the  *3D View* tab you can make the settings for the rendering of the point cloud in 3D maps.

3D Rendering modes

Following options can be selected from the drop down menu at the top of the tab:

- *No Rendering*: Data are not displayed
- *Follow 2D Symbology*: Syncs features rendering in 3D with *symbology assigned in 2D*
-  *Single Color*: All points are displayed in the same *color* regardless of attributes
-  *Attribute by Ramp*: Interpolates a given attribute over a color ramp and assigns to features their matching color. See *Attribute by Ramp Renderer*.
-  *RGB*: Use different attributes of the features to set the Red, Green and Blue color components to assign to them. See *RGB Renderer*.
-  *Classification*: differentiates points by color on the basis of an attribute. See *Classification Renderer*.

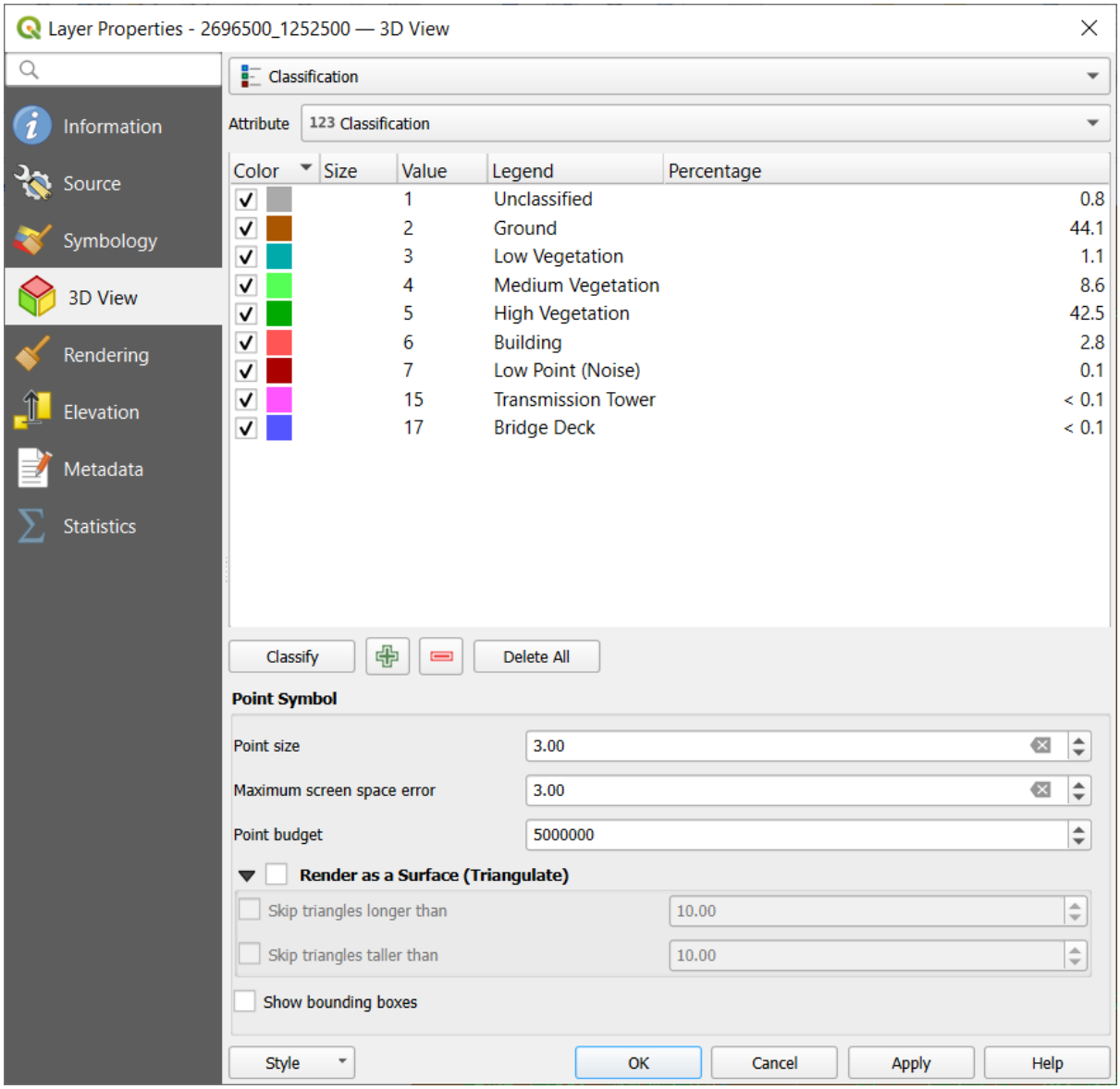


Fig. 16.11: The point cloud 3D view tab with the classification renderer

3D Point Symbol

In the lower part of the *3D View* tab you can find the *Point Symbol* section. Here you can make general settings for the entire layer which are the same for all renderers. There are the following options:

- *Point size*: The size (in pixels) with which each data point is displayed can be set
- *Maximum screen space error*: By this option you set how dense or sparse the display of the point cloud will be (in pixels). If you set a large number (e.g. 10), there will be visible gaps between points; low value (e.g. 0) could force rendering of unnecessary amount of points, making rendering slower (you can find more details at *Symbology* *Maximum error*).
- *Point budget*: To avoid long rendering, you can set the maximum number of points that will be rendered
- Check *Render as surface (Triangulate)* to render the point cloud layer in the 3D view with a solid surface obtained by triangulation. You can control dimensions of the computed triangles:
  - *Skip triangles longer than* a threshold value: sets in the horizontal plan, the maximum length of a side

of the triangles to consider

  - *Skip triangles taller than* a threshold value: sets in the vertical plan, the maximum height of a side of the triangles to consider

- *Show bounding boxes*: Especially useful for debugging, shows bounding boxes of nodes in hierarchy

16.2.5 Rendering Properties

Under the *Scale dependent visibility* group box, you can set the *Maximum (inclusive)* and *Minimum (exclusive)* scale, defining a range of scale in which features will be visible. Out of this range, they are hidden. The Set to current canvas scale button helps you use the current map canvas scale as boundary of the range visibility. See *Visibility Scale Selector* for more information.

Note

You can also activate scale dependent visibility on a layer from within the *Layers* panel: right-click on the layer and in the contextual menu, select *Set Layer Scale Visibility*.

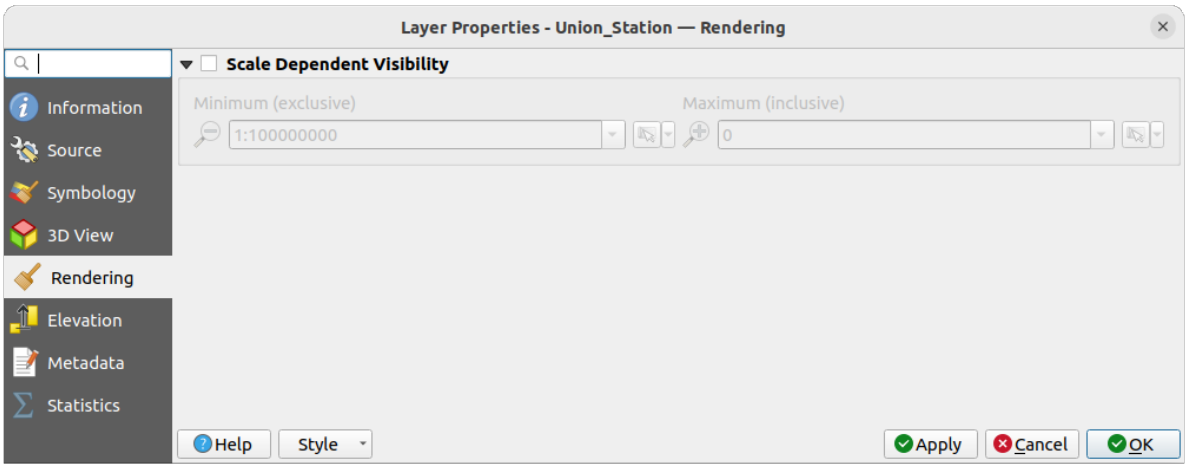


Fig. 16.12: The point cloud rendering tab

16.2.6 Elevation Properties

In the *Elevation* tab, you can set corrections for the Z-values of the data. This may be necessary to adjust the elevation of the data in 3D maps and its appearance in the *profile tool charts*. There are following options:

- *Vertical Reference System*: If the CRS of your point cloud layer is a compound one (including a Z dimension), then the vertical CRS used for the layer will be automatically derived from the vertical component of the layer's CRS. In this case, you cannot manually set a different vertical CRS, and the option to change it will be disabled. If your point cloud layer uses a horizontal (2D) CRS (though uncommon), you can manually select a specific vertical CRS by clicking on the Select CRS. Vertical reference systems are supported for point cloud layers in:
  - *Elevation profiles*
  - *Identify Tool Results*
  - *3D map views*
- Under *Elevation* group:

  - You can set a *Scale*: If `10` is entered here, a point that has a value Z = `5` is displayed at a height of `50`.
  - An *offset* to the z-level can be entered. This is useful to match different data sources in its height to each other. By default, the lowest z-value contained in the data is used as this value. This value can also be restored with the Refresh button at the end of the line.

- Under *Profile Chart Accuracy*, the *Maximum error* helps you control how dense or sparse the points will be rendered in the elevation profile. Larger values result in a faster generation with less points included.
- Under *Profile Chart Appearance*, you can control the point display:
  - *Point size*: the size to render the points with, in supported units (millimeters, map units, pixels, …)
  - *Style*: whether to render the points as *Circle* or *Square*
  - Apply a single *Color* to all the points visible in the profile view
  - Check *Respect layer's coloring* to instead show the points with the color assigned via their *2D symbology*
  - *Apply opacity by distance from curve effect*, reducing the opacity of points which are further from the profile curve

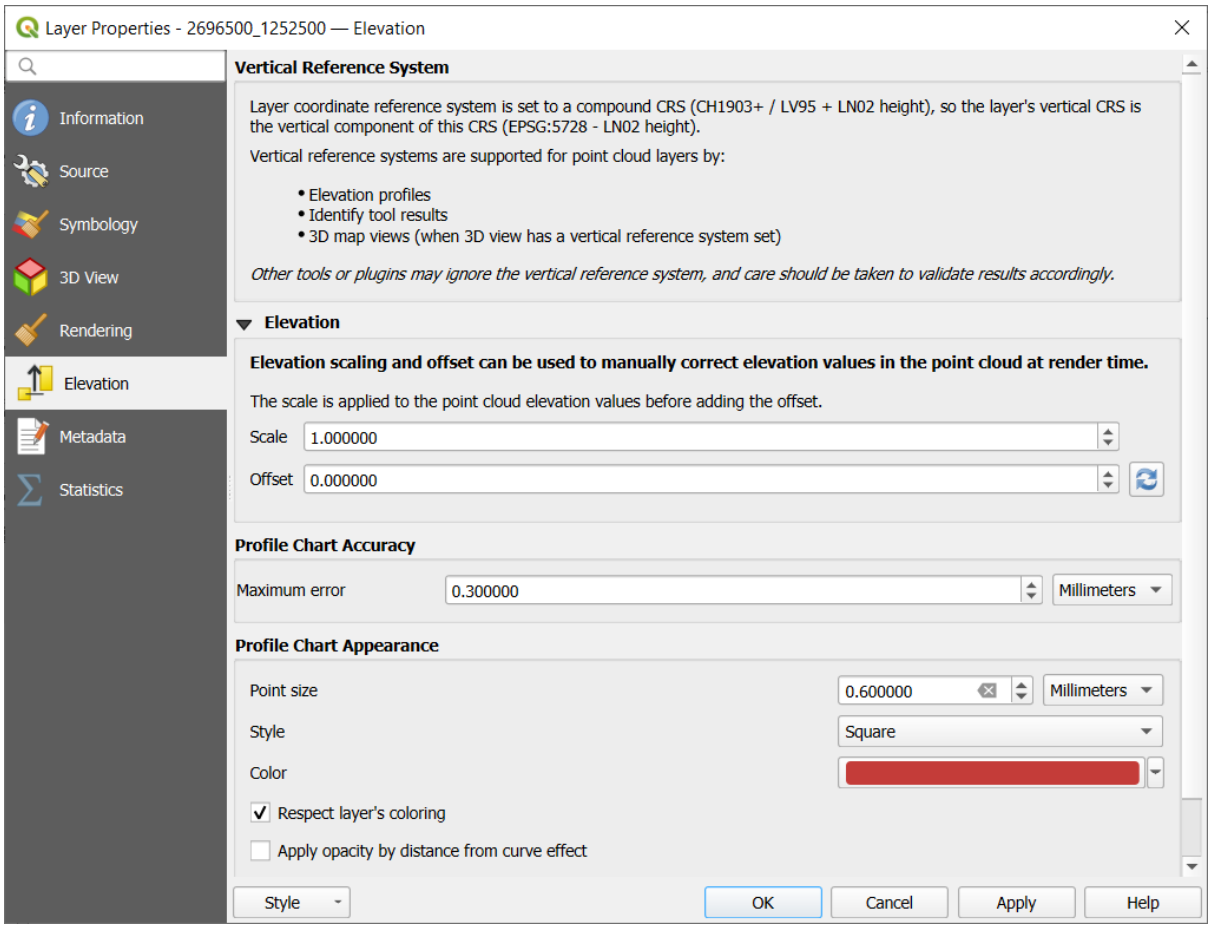


Fig. 16.13: The point cloud elevation tab

16.2.7 Metadata Properties

The *Metadata* tab provides you with options to create and edit a metadata report on your layer. See *Metadata* for more information.

16.2.8 Statistics Properties

In the *Statistics* tab you can get an overview of the attributes of your point cloud and their distribution.

At the top you will find the section *Attribute Statistics*. Here all attributes contained in the point cloud are listed, as well as some of their statistical values: *Minimum*, *Maximum*, *Mean*, *Standard Deviation*

If there is an attribute *Classification*, then there is another table in the lower section. Here all values contained in the attribute are listed, as well as their absolute *Count* and relative *%* abundance.

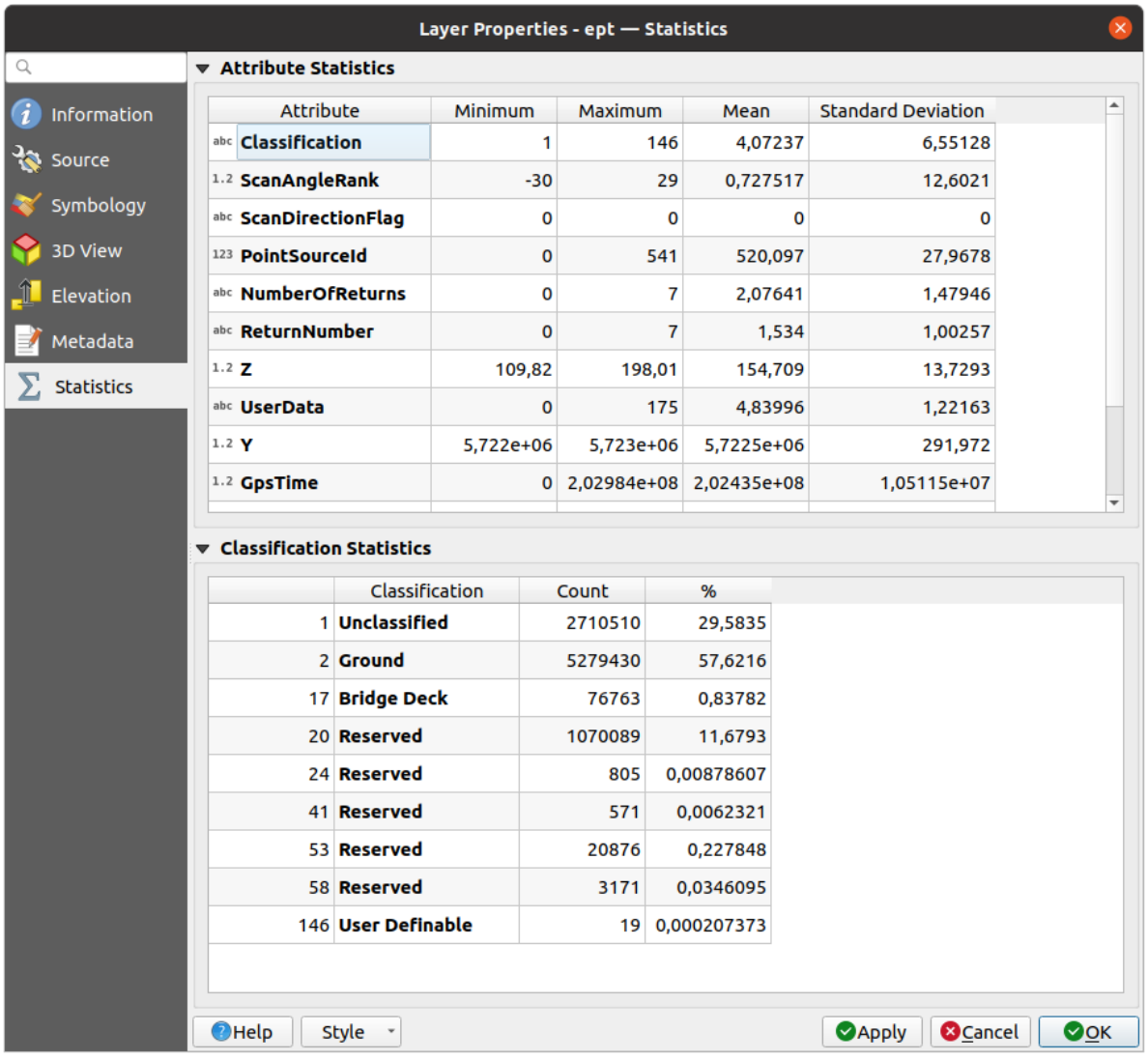


Fig. 16.14: The point cloud statistics tab

16.3 Virtual point cloud

Lidar surveys of larger areas are often multi-terabyte datasets with many billions of points. Having such large datasets represented as a single point cloud file is not practical due to the difficulties of storage, transfer, display and analysis. Point cloud data are therefore typically stored and distributed split into square tiles (e.g. `1km x 1km`), each tile having a more manageable file size (e.g. ~200 MB when compressed).

Tiling of data solves the problems with size of data, but it introduces issues when processing or viewing an area of interest that does not fit entirely into a single tile. Users need to develop workflows that take into account multiple tiles and special care needs to be taken to deal with data near edges of tiles to avoid unwanted artefacts in outputs. Similarly, when viewing point cloud data, it becomes cumbersome to load many individual files and apply the same symbology.

Here is an example of several point cloud tiles loaded in QGIS. Each tile is styled based on min/max Z values of the tile, creating visible artefacts on tile edges. The styling has to be adjusted for each layer separately:

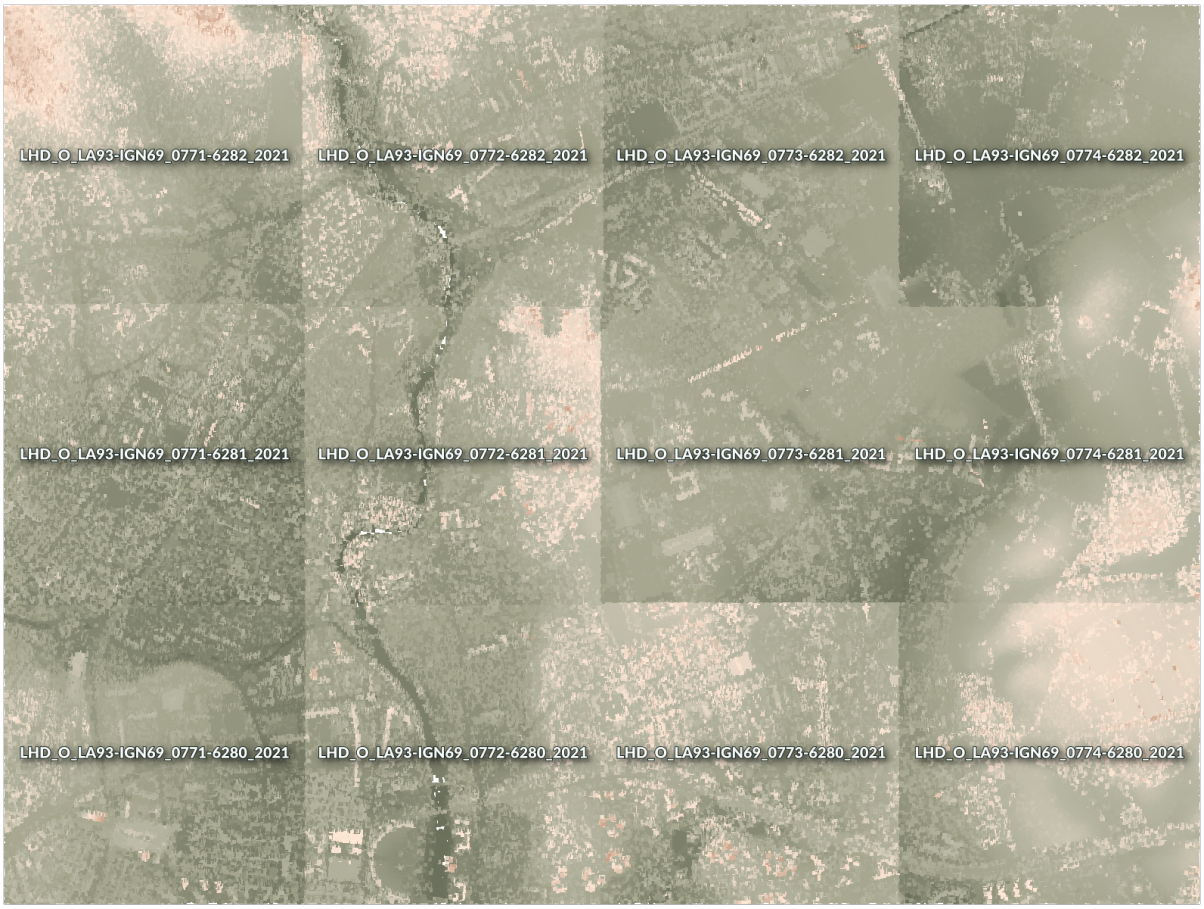


Fig. 16.15: Individual point cloud tiles loaded, with artefacts on edges

In the GIS world, many users are familiar with the concept of virtual rasters. A virtual raster is a file that simply references other raster files with actual data. In this way, GIS software then treats the whole dataset comprising many files as a single raster layer, making the display and analysis of all the rasters listed in the virtual file much easier.

Borrowing the concept of virtual rasters from GDAL, **virtual point cloud (VPC)** is a file format that references other point cloud files. Software supporting virtual point clouds handles the whole tiled dataset as a single data source.

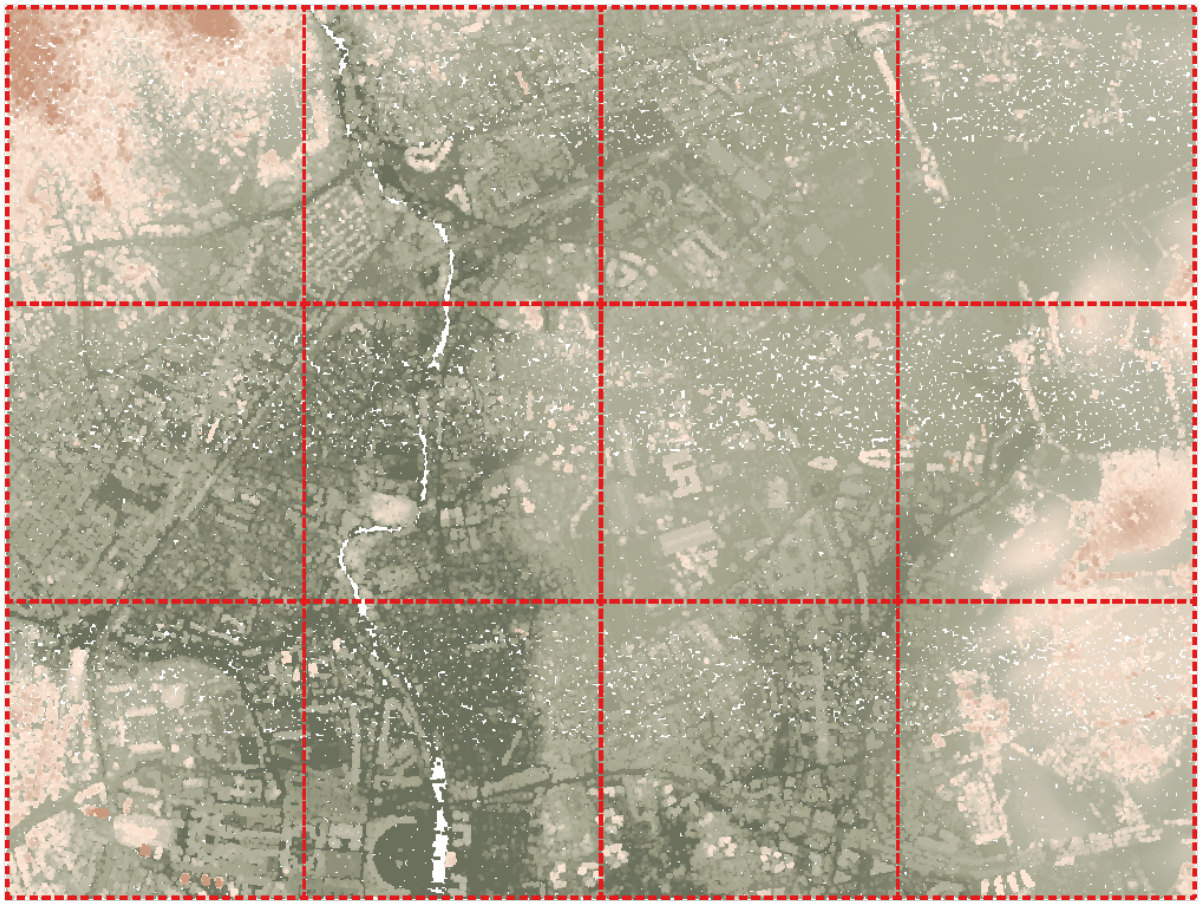


Fig. 16.16: The virtual point cloud

Displaying and manipulating virtual point cloud is much more fluent and easy.

At the core, a virtual point cloud file is a simple JSON file with `.vpc` extension, containing references to actual data files (e.g. `.LAS`, `.LAZ` or `.COPC` files) and additional metadata extracted from the files. Even though it is possible to write VPC files by hand, it is strongly recommended to create them using an automated tool, such as:

- The Processing *Build virtual point cloud (VPC)* algorithm
- The `build_vpc` command of PDAL wrench tool

For more details, please refer to the VPC specification that also contains best practices and optional extensions (such as overviews).

CHAPTER

SEVENTEEN

WORKING WITH 3D TILES

17.1 What are 3D Tiles?

3D tiles are specification for streaming and rendering large-scale 3D geospatial datasets. They use a hierarchical structure to efficiently manage and display 3D content, optimising performance by dynamically loading appropriate levels of detail. This technology is widely used in urban planning, architecture, simulation, gaming, and virtual reality, providing a standardised and interoperable solution for visualising complex geographical data.



Fig. 17.1: Example of 3D tiles

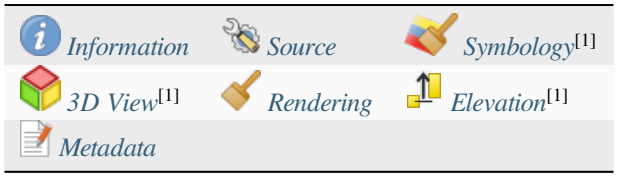
Currently, QGIS supports two formats of 3D tiles:

- Cesium 3D tiles, used primarily for complex 3D models of buildings or whole cities. Such datasets can be provided by cloud-based platforms such as Cesium Ion or Google (*Photorealistic 3D Tiles*).
- Quantized Mesh tiles, used for terrain elevation data.

See *Using 3D tiled scene services* for instructions on how to add these data sources to QGIS.

17.2 3D Tiles Properties

The 3D tiles *Layer Properties* dialog provides the following sections:



[1] Also available in the *Layer styling panel*

17.2.1 Information Properties

The *Information* tab is read-only and represents an interesting place to quickly grab summarized information and metadata on the current layer. Provided information are:

- based on the provider of the layer: name, URL, source type and path, number of zoom levels
- custom properties, used to store in the active project additional information about the layer. More properties can be created and managed using PyQGIS, specifically through the setCustomProperty() method.
- the Coordinate Reference System: name, units, method, accuracy, reference (i.e. whether it's static or dynamic)
- picked from the *filled metadata*: access, extents, links, contacts, history…

17.2.2 Source Properties

The *Source* tab displays basic information about the selected 3D tile, including:

- the *Layer name* to display in the *Layers Panel*;
- the *Coordinate Reference System*: Displays the layer's *Coordinate Reference System (CRS)*. You can change the layer's CRS, by selecting a recently used one in the drop-down list or clicking on the Select CRS button (see *Coordinate Reference System Selector*). Use this process only if the layer CRS is wrong or not specified.

17.2.3 Symbology Properties

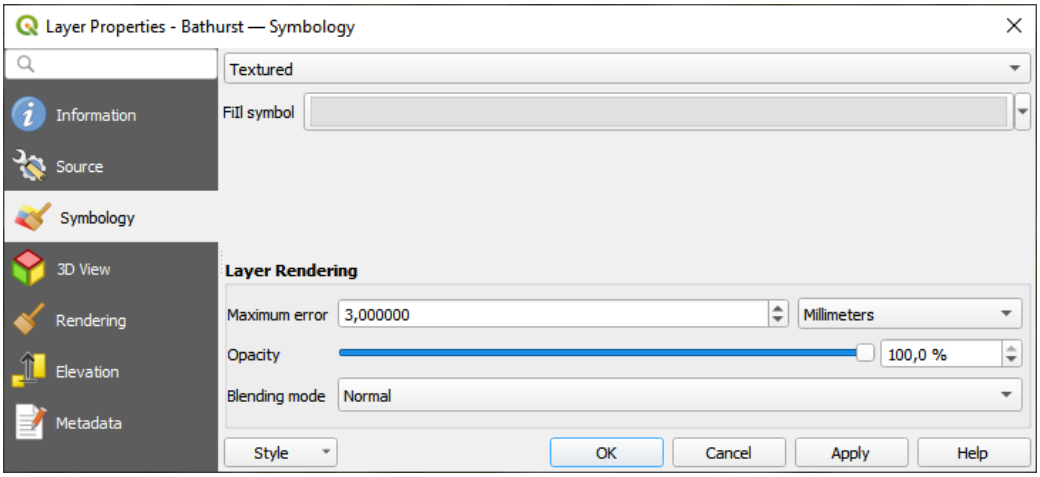


Fig. 17.2: 3D Tile Layer Symbology

By default, the layer is styled using texture, but you can change it to see the wireframe mesh behind the scene by choosing *Wireframe* in the drop-down menu. You can also, change the mesh fill and line symbols similar to the vector polygons. Checking *Use texture colors* will render each mesh element with the average value of the full texture. This is a good option to try when dealing with a large dataset and want to get a quick overview of the data.

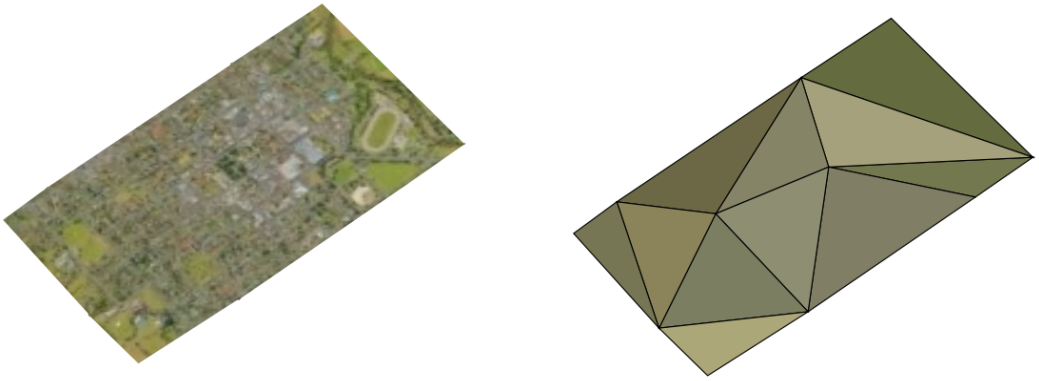


Fig. 17.3: 3D Tiles - textured and wireframe

From the *Symbology* tab, you can also set some options that invariably act on all features of the layer:

- *Maximum error*: This parameter determines the level of detail displayed in the 3D model. Similar to point clouds, 3D tiles often contain more information than necessary for visual representation. By adjusting this setting, you control the balance between display density and rendering speed. A larger value (e.g., 5 mm) may introduce noticeable gaps between elements, while a smaller value (e.g., 0.1 mm) could lead to the rendering of an excessive number of details, potentially slowing down the rendering process. Different units can be selected to tailor the setting to your specific needs.
- *Opacity*: Adjusts the visibility of the underlying layer on the map canvas using this tool. Use slider to tailor the visibility of your scene layer according to your preferences. Alternatively, specify the exact percentage of visibility through the menu next to the slider.
- *Blending mode*: You can achieve special rendering effects with these tools that you may previously only know from graphics programs. The pixels of your overlaying and underlaying layers are mixed through the settings described in *Blending Modes*.

17.2.4 3D View Properties

- *Maximum screen space error*: Determines the threshold for swapping terrain tiles with more detailed ones (and vice versa) - i.e. how soon the 3D view will use higher quality tiles. Lower numbers mean more details in the scene at the expenses of increased rendering complexity.
- *Show bounding boxes*: Shows 3D bounding boxes of the terrain tiles (useful for troubleshooting terrain issues).

To view the data you can open *New 3D map view*.

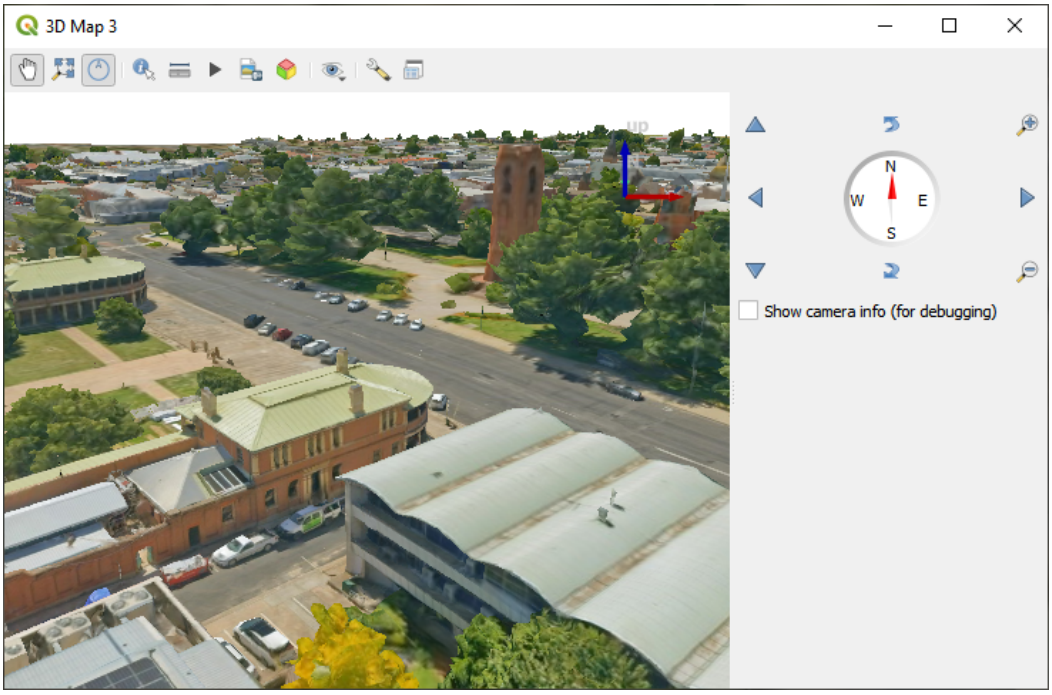


Fig. 17.4: 3D Map View

17.2.5 Rendering Properties

Under *Scale dependent visibility*, you can set the *Maximum (inclusive)* and *Minimum (exclusive)* scales, defining a range of scales in which features will be visible. Out of this range, they are hidden. The Set to current canvas scale button helps you use the current map canvas scale as boundary of the range visibility. See *Visibility Scale Selector* for more information.

17.2.6 Elevation Properties

The *Elevation* tab provides options to control the layer elevation properties within a *3D map view*. Specifically, you can set:

- *Elevation Surface*: how the 3D layer vertices Z values should be interpreted as terrain elevation. You can apply a *Scale* factor and an *Offset*.

17.2.7 Metadata Properties

The *Metadata* tab provides you with options to create and edit a metadata report on your layer. See *Metadata* for more information.

CHAPTER

EIGHTEEN

LAYING OUT THE MAPS

With Print Layouts and Reports you can create maps and atlases, and print them or save them as image, PDF or SVG files.

18.1 Overview of the Print Layout

The print layout provides growing layout and printing capabilities. It allows you to add elements such as the QGIS 2D or 3D map canvas, text labels, images, legends, scale bars, basic shapes, arrows, attribute and simple tables, elevation profiles and HTML frames… You can size, group, align, position and rotate each element and adjust their properties to create your layout.

The layout can be printed or exported to image formats, PostScript, PDF or to SVG. You can save the layout as a template and load it again in another session. Finally, generating several maps based on a template can be done through the atlas generator.

18.1.1 Sample Session for beginners

Before you start to work with the print layout, you need to load some raster or vector layers in the QGIS map canvas and adapt their properties to suit your own convenience. After everything is rendered and symbolized to your liking, click the  New Print Layout icon in the *Project* toolbar or choose *Project* ►  *New Print Layout*. You will be prompted to choose a title for the new layout.

To demonstrate how to create a map please follow the next instructions.

1. On the left side, select the  Add map toolbar button and draw a rectangle on the canvas holding down the left mouse button. Inside the drawn rectangle the QGIS map view to the canvas.
2. Select the  Add scalebar toolbar button and click with the left mouse button on the print layout canvas. A scalebar will be added to the canvas.
3. Select the  Add legend toolbar button and draw a rectangle on the canvas holding down the left mouse button. Inside the drawn rectangle the legend will be drawn.
4. Select the  Select/Move item icon to select the map on the canvas and move it a bit.
5. While the map item is still selected you can also change the size of the map item. Click while holding down the left mouse button, in a white little rectangle in one of the corners of the map item and drag it to a new location to change its size.
6. Click the *Item Properties* panel on the left down side and find the setting for the orientation. Change the value of the setting *Map orientation* to '15.00°'. You should see the orientation of the map item change.
7. Now, you can print or export your print layout to image formats, PDF or to SVG with the export tools in *Layout* menu.

8. Finally, you can save your print layout within the project file with the Save Project button.

You can add multiple elements to the print layout. It is also possible to have more than one map view or legend or scale bar in the print layout canvas, on one or several pages. Each element has its own properties and, in the case of the map, its own extent. If you want to remove any elements from the layout canvas, you can do that with the `Delete` or the `Backspace` key.

18.1.2 The Layout Manager

The *Layout Manager* is the main window to manage print layouts in the project. It gives you an overview of existing print layouts and reports in the project and offers tools to:

- search for a layout;
- add new print layout or new report from scratch, template or duplicating an existing one;
- rename or delete any of them;
- open them in the project.

To open the layout manager dialog:

- from the main QGIS dialog, select *Project* ► *Layout Manager…* menu or click on the Layout Manager button in the *Project Toolbar*;
- from a print layout or report dialog, select *Layout* ► *Layout Manager…* menu or click on the Layout Manager button in the *Layout Toolbar*.

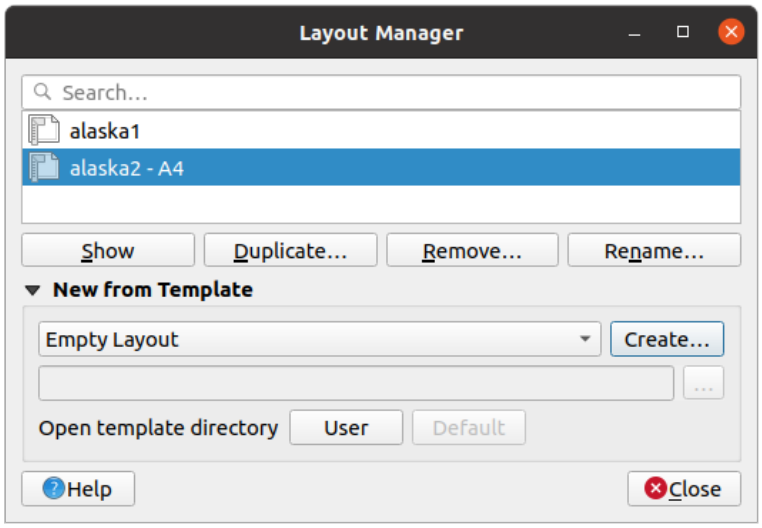


Fig. 18.1: The Print Layout Manager

The layout manager lists in its upper part all the available print layouts or reports in the project with tools to:

- show the selection: you can select multiple reports and/or print layout(s) and open them in one-click. Double-click a name also opens it;
- duplicate the selected print layout or report (available only if one item is selected): it creates a new dialog using the selected one as template. You'll be prompted to choose a new title for the new layout;
- rename the report or layout (available only if one item is selected): you'll be prompted to choose a new title for the layout;
- remove the layout: the selected print layout(s) will be deleted from the project.

In the lower part, it's possible to create new print layouts or reports from scratch or a template. By default, QGIS will look for templates in the *user profile* folder and the application template directories (accessible with the two buttons at the bottom of the frame) but also in any folder declared as *Path(s) to search for extra print templates* in *Settings* ► *Options* ► *Layouts*. Found templates are listed in the combobox. Select an item and press the *Create* button to generate a new report or print layout.

You can also use layout templates from a custom folder; in that case, select *specific* in the templates drop-down list, browse to the template and press *Create*.

> **Tip**
>
> **Creating template-based print layouts from Browser panel**
>
> Drag-and-drop a print layout template `.qpt` file from any file browser onto the map canvas or double-click it in the *Browser panel* generates a new print layout from the template.

18.1.3 Menus, tools and panels of the print layout

Opening the print layout provides you with a blank canvas that represents the paper surface when using the print option. Initially you find buttons on the left beside the canvas to add print layout items: 2D or 3D map canvases, text labels, images, legends, scale bars, basic shapes, arrows, attribute or simple tables, HTML frames, elevation profiles,… In this toolbar you also find buttons to navigate, zoom in on an area and pan the view on the layout as well as buttons to select any layout item and to move the extents of the map items.

Fig. 18.2 shows the initial view of the print layout before any elements are added.

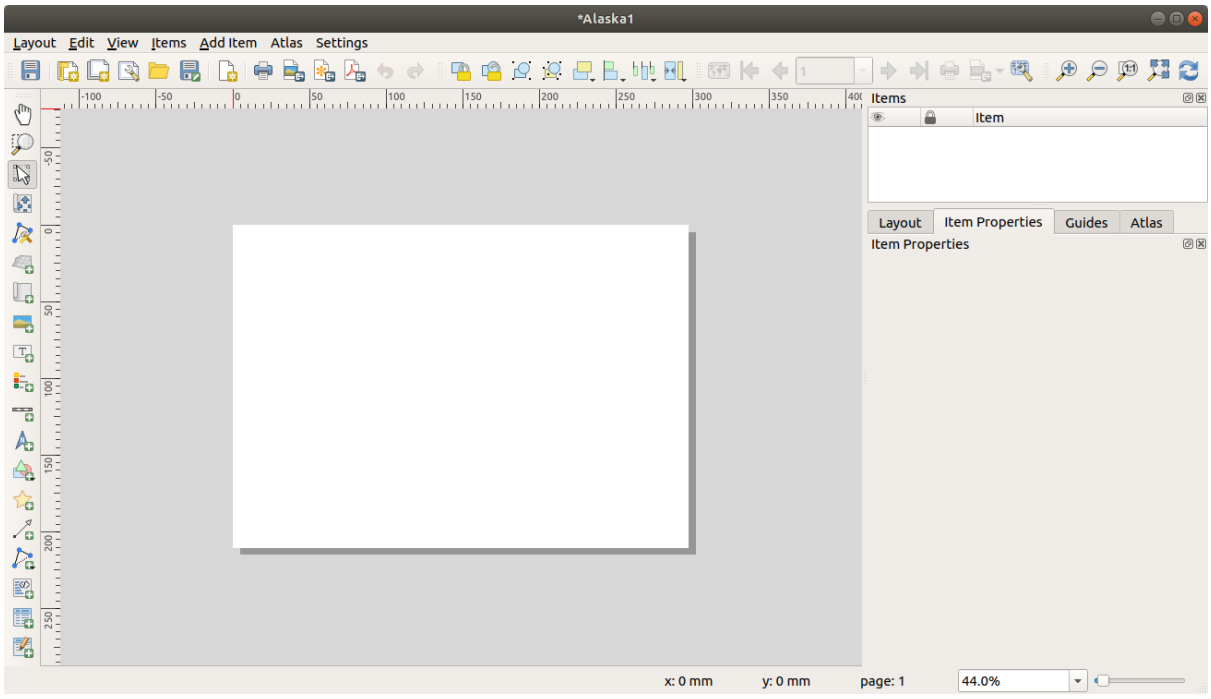


Fig. 18.2: Print Layout

On the right beside the canvas you find two set of panels. The upper one holds the panels *Items* and *Undo History* and the lower holds the panels *Layout*, *Item properties* and *Atlas generation*.

- The *Items* panel provides a list of all the print layout items added to the canvas and ways to globally interact with them (see *The Items Panel* for more information).
- The *Undo History* panel displays the history of all changes applied to the layout. With a mouse click, it is possible to undo and redo layout steps back and forth to a certain status.

- The *Layout* panel allows you to set general parameters to apply to the layout when exporting or working within (see *The Layout Panel* for more details);
- The *Item Properties* panel displays the properties for the selected item. Click the  Select/Move item icon to select an item (e.g., legend, scale bar or label) on the canvas. Then click the *Item Properties* panel and customize the settings for the selected item (see *Layout Items* for detailed information on each item settings).
- The *Atlas* panel allows you to enable the generation of an atlas for the current layout and gives access to its parameters (see *Generate an Atlas* for detailed information on atlas generation usage).

In the bottom part of the print layout window, you can find a status bar with mouse position, current page number, a combo box to set the zoom level, the number of selected items if applicable and, in the case of atlas generation, the number of features.

In the upper part of the print layout window, you can find menus and other toolbars. All print layout tools are available in menus and as icons in a toolbar.

The toolbars and the panels can be switched off and on using the right mouse button over any toolbar, or through *View* ► *Toolbars* ► or *View* ► *Panels* ►.

Menus and Tools

Layout menu

The *Layout* provides action to manage the layout:

- Save the project file directly from the print layout window.
- Create a new and blank print layout with  *New Layout…*.
-  *Duplicate Layout…* : Create a new print layout by duplicating the current one.
- Remove the current layout with  *Delete Layout…*.
- Open the  *Layout Manager…*.
- *Layouts* ► : Open an existing print layout.

Once the layout is designed, with  *Save as Template* and  *Add Items from Template* icons, you can save the current state of a print layout session as a `.qpt` template file and load its items again in another session/print layout.

In the *Layout* menu, there are also powerful ways to share geographical information produced with QGIS that can be included in reports or published. These tools are  *Export as Image…*,  *Export as PDF…*,  *Export as SVG…* and  *Print…*.

Below is a list of all the available tools in this menu with some convenient information.

Table 18.1: "Layout" Menu Tools

Tool	Shortcut	Toolbar	Reference
Save Project	`Ctrl+S`	*Layout*	*Introducing QGIS projects*
New Layout	`Ctrl+N`	*Layout*	*The Layout Manager*
Duplicate Layout		*Layout*	*The Layout Manager*
Delete Layout			
Layout Manager…		*Layout*	*The Layout Manager*
Layouts ►			
Layout Properties…			*The Layout Panel*
Rename Layout…			
Page Properties…			*Working with the page properties*
Add Pages…		*Layout*	*Working with the page properties*
Add Items from Template		*Layout*	*Creating a layout item*
Save as Template…		*Layout*	*The Layout Manager*
Export as Image…		*Layout*	*Export as Image*
Export as SVG…		*Layout*	*Export as SVG*
Export as PDF…		*Layout*	*Export as PDF*
Printer Page Setup…	`Ctrl+Shift+P`		
Print…	`Ctrl+P`	*Layout*	*Creating an Output*
Close	`Ctrl+Q`		

Edit menu

The *Edit* menu offers tools to manipulate print layout items. It includes common actions like selection tools, Copy/Cut/Paste and undo/redo (see *The Undo History Panel: Revert and Restore actions*) functionality for the items in the layout.

When using the Paste action, the elements will be pasted according to the current mouse position. Using the *Edit* ► *Paste in Place* action or pressing `Ctrl+Shift+V` will paste the items into the current page, at the same position they were in their initial page. It ensures to copy/paste items at the same place, from page to page.

Below is a list of all the available tools in this menu with some convenient information.

Table 18.2: "Edit" Menu Tools

Tool	Shortcut	Toolbar	Reference
Undo (last change)	`Ctrl+Z`	*Layout*	*The Undo History Panel: Revert and Restore actions*
Redo (last reverted change)	`Ctrl+Y`	*Layout*	*The Undo History Panel: Revert and Restore actions*
Delete	`Del`		
Cut	`Ctrl+X`		
Copy	`Ctrl+C`		
Paste	`Ctrl+V`		
Paste in place	`Ctrl+Shift+V`		
Select All	`Ctrl+A`		
Deselect all	`Ctrl+Shift+A`		
Invert Selection			
Select Next Item Below	`Ctrl+Alt+[`		
Select Next Item above	`Ctrl+Alt+]`		
Pan Layout	`P`	*Toolbox*	
Zoom	`Z`	*Toolbox*	
Select/Move Item	`V`	*Toolbox*	*Interacting with layout items*
Move Content	`C`	*Toolbox*	*The Map Item*
Edit Nodes Item		*Toolbox*	*The Node-Based Shape Items*

View menu

The *View* menu gives access to navigation tools and helps to configure general behavior of the print layout. Beside the common zoom tools, you have means to:

- Refresh view (if you find the view in an inconsistent state);
- enable a *grid* you could snap items to when moving or creating them. Grids setting is done in *Settings* ► *Layout Options…* or in the *Layout Panel*;
- enable *guides* you could snap items to when moving or creating them. Guides are red lines that you can create by clicking in the ruler (above or at the left side of the layout) and drag and drop to the desired location;
- *Smart Guides*: uses other layout items as guides to dynamically snap to as you move or reshape an item;
- *Clear Guides* to remove all current guides;
- *Show Bounding box* around the items to better identify your selection;
- *Show Rules* around the layout;
- *Show Pages* or set up pages to transparent. Often layout is used to create non-print layouts, e.g. for inclusion in presentations or other documents, and it's desirable to export the composition using a totally transparent background. It's sometimes referred to as "infinite canvas" in other editing packages.

In the print layout, you can change the zoom level using the mouse wheel or the slider and combo box in the status bar. If you need to switch to pan mode while working in the layout area, you can hold the `Spacebar` or the mouse wheel. With `Ctrl+Spacebar`, you can temporarily switch to Zoom In mode, and with `Ctrl+Alt+Spacebar`, to Zoom Out mode.

Panels and toolbars can be enabled from the *View* ► menu. To maximise the space available to interact with a composition you can check the *View* ► *Toggle Panel Visibility* option or press `Ctrl+Tab`; all panels are hidden and only previously visible panels are restored when unchecked.

It's also possible to switch to a full screen mode to have more space to interact with by pressing `F11` or using *View* ► *Toggle Full Screen*.

Table 18.3: "View" Menu Tools

Tool	Shortcut	Toolbar	Reference
Refresh	`F5`	*Navigation*	
Preview ►			
Zoom In	`Ctrl++`	*Navigation*	
Zoom Out	`Ctrl+-`	*Navigation*	
Zoom to 100%	`Ctrl+1`	*Navigation*	
Zoom Full	`Ctrl+0`	*Navigation*	
Zoom to Width			
Show Grid	`Ctrl+'`		*Guides and Grid*
Snap to Grid	`Ctrl+Shift+'`		*Guides and Grid*
Show Guides	`Ctrl+;`		*Guides and Grid*
Snap to Guides	`Ctrl+Shift+;`		*Guides and Grid*
Smart Guides	`Ctrl+Alt+;`		
Manage Guides…			*The Guides Panel*
Clear Guides			*The Guides Panel*
Show Rulers	`Ctrl+R`		
Show Bounding Boxes	`Ctrl+Shift+B`		
Show Pages			
Toolbars ►			*Panels and Toolbars*
Panels ►			*Panels and Toolbars*
Toggle Full Screen	`F11`		*View*
Toggle Panel Visibility	`Ctrl+Tab`		*View*

Items menu

The *Items* helps you configure items' position in the layout and the relations between them (see *Interacting with layout items*).

Table 18.4: "Items" Menu Tools

Tool	Shortcut	Toolbar	Reference
 *Group*	`Ctrl+G`	*Actions*	*Grouping items*
 *Ungroup*	`Ctrl+Shift+G`	*Actions*	*Grouping items*
 *Raise*	`Ctrl+]`	*Actions*	*Alignment and distribution*
 *Lower*	`Ctrl+[`	*Actions*	*Alignment and distribution*
 *Bring to Front*	`Ctrl+Shift+]`	*Actions*	*Alignment and distribution*
 *Send to Back*	`Ctrl+Shift+[`	*Actions*	*Alignment and distribution*
 *Lock Selected Items*	`Ctrl+L`	*Actions*	*Locking items*
 *Unlock All*	`Ctrl+Shift+L`	*Actions*	*Locking items*
Align Items ►		*Actions*	*Alignment and distribution*
Distribute Items ►		*Actions*	*Moving and resizing items*
Resize ►		*Actions*	*Moving and resizing items*

Add Item menu

These are tools to create layout items. Each of them is deeply described in *Layout Items* chapter.

Table 18.5: “Add Item” Menu Tools

Tool	Toolbar	Reference
Add Map	*Toolbox*	*The Map Item*
Add 3D Map	*Toolbox*	*The 3D Map Item*
Add Picture	*Toolbox*	*The Picture Item*
Add Label	*Toolbox*	*The Label Item*
Add Dynamic Text ►		*The Label Item*
Add Legend	*Toolbox*	*The Legend Item*
Add Scale Bar	*Toolbox*	*The Scale Bar Item*
Add North Arrow	*Toolbox*	*The North Arrow Item*
Add Shape ►	*Toolbox*	*The Regular Shape Item*
► *Add Rectangle*	*Toolbox*	*The Regular Shape Item*
► *Add Ellipse*	*Toolbox*	*The Regular Shape Item*
► *Add Triangle*	*Toolbox*	*The Regular Shape Item*
Add Marker	*Toolbox*	*The Marker Item*
Add Arrow	*Toolbox*	*The Arrow Item*
Add Node Item ►	*Toolbox*	*The Node-Based Shape Items*
► *Add Polygon*	*Toolbox*	*The Node-Based Shape Items*
► *Add Polyline*	*Toolbox*	*The Node-Based Shape Items*
Add HTML	*Toolbox*	*The HTML Frame Item*
Add Attribute Table	*Toolbox*	*The attribute table item*
Add Fixed Table	*Toolbox*	*The fixed table item*
Add Elevation Profile	*Toolbox*	*The Elevation Profile Item*

Atlas menu

Table 18.6: "Atlas" Menu Tools

Tool	Shortcut	Toolbar	Reference
Preview Atlas	`Ctrl+ALt+/`	*Atlas*	*Preview and generate an atlas*
First Feature	`Ctrl+<`	*Atlas*	*Preview and generate an atlas*
Previous Feature	`Ctrl+,`	*Atlas*	*Preview and generate an atlas*
Next Feature	`Ctrl+.`	*Atlas*	*Preview and generate an atlas*
Last feature	`Ctrl+>`	*Atlas*	*Preview and generate an atlas*
Print Atlas…		*Atlas*	*Preview and generate an atlas*
Export Atlas as Images…		*Atlas*	*Preview and generate an atlas*
Export Atlas as SVG…		*Atlas*	*Preview and generate an atlas*
Export Atlas as PDF…		*Atlas*	*Preview and generate an atlas*
Atlas Settings		*Atlas*	*Generate an Atlas*

Settings Menu

The *Settings* ► *Layout Options…* menu is a shortcut to *Settings* ► *Options* ► *Layouts* menu of QGIS main canvas. Here, you can set some options that will be used as default on any new print layout:

- *Layout defaults* let you specify the default font to use;
- With *Grid appearance*, you can set the grid style and its color. There are three types of grid: **Dots**, **Solid** lines and **Crosses**;
- *Grid and guide defaults* defines spacing, offset and tolerance of the grid (see *Guides and Grid* for more details);
- *Layout Paths*: to manage list of custom paths to search print templates.

The *Settings* ► *Keyboard Shortcuts…* menu allows you to use the *shortcuts manager* in the print layout interface.

Contextual menus

Depending on where you right-click in the print layout dialog, you open a contextual menu with various features:

- Right-click on the menu bar or any toolbar and you get the list of layout panels and toolbars you can enable or disable in one-click.
- Right-click over a ruler and you can *Show Guides*, *Snap to Guides*, *Manage Guides…* opening the *Guides panel* or *Clear Guides*. It's also possible to hide the rulers.
- Right-click in the print layout canvas and:
  - You'll be able to *Undo* and *Redo* recent changes, or *Paste* any copied item (only available if no item is selected).
  - If you click over a page, you can additionally access the current *Page Properties* panel or *Remove Page*.
  - If you click on a selected item then you can cut or copy it as well as open the *Item Properties* panel.
  - If more than one item are selected, then you can either group them and/or ungroup if at least one group is already in the selection.

- Right-click inside a text box or spinbox widget of any layout panel provides edit options to manipulate its content.

The Layout Panel

In the *Layout* panel, you can define the global settings of your print layout.

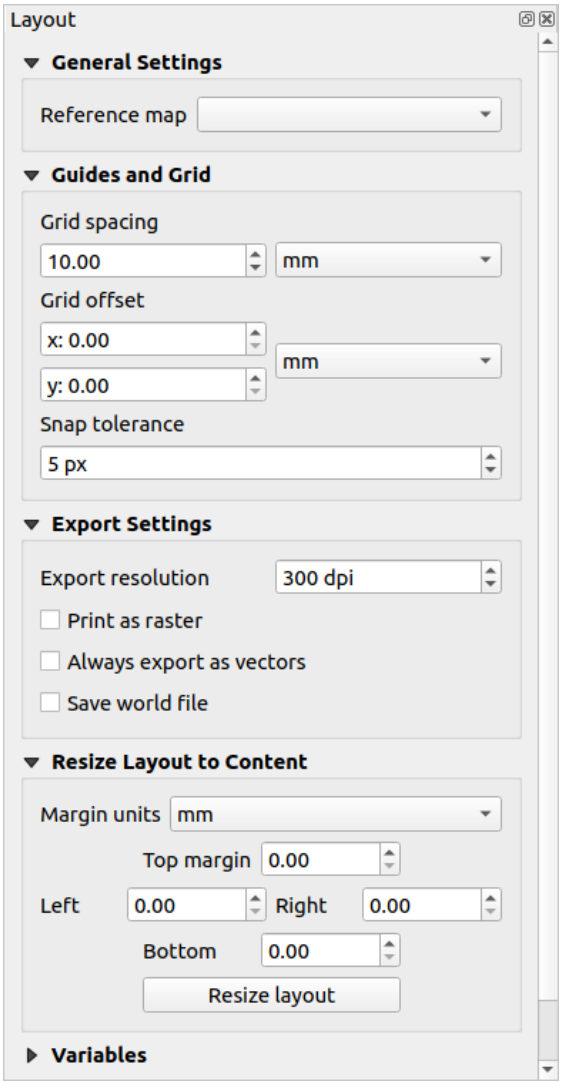


Fig. 18.3: Layout Settings in the Print Layout

General settings

In a print layout, you can use more than one map item. The *Reference map* represents the map item to use as the layout's master map. It's assigned as long as there's a map item in the layout. The layout will use this map in any of their properties and variables calculating units or scale. This includes exporting the print layout to georeferenced formats.

Moreover, new layout items such as scale bar, legend or north arrow have by default their settings (orientation, displayed layers, scale, …) bound to the map item they are drawn over, and fall back to the reference map if there is no overlapping map.

Guides and Grid

You can put some reference marks on your paper sheet to help you accurately place some items. These marks can be:

- simple horizontal or vertical lines (called **Guides**) put at the position you want (see *The Guides Panel* for guides creation).
- or regular **Grid**: a network of horizontal and vertical lines superimposed over the layout.

Settings like *Grid spacing* or *Grid offset* can be adjusted in this group as well as the *Snap tolerance* to use for items. The tolerance is the maximum distance below which the mouse cursor is snapped to a grid or a guide, while moving, resizing or creating an item.

Whether grid or guides should be shown is set in *View* menu. There, you can also decide if they might be used to snap layout items. When both a grid line and a guide line are within tolerance of a point, guides will always take precedence - since they have been manually set (hence, assumption that they have been explicitly placed at highly desirable snapping locations, and should be selected over the general grid).

> **Note**
>
> In the *Settings* ► *Layout Options* menu, you can also set the grid and guides parameters exposed above. However, these options will only apply as defaults to new print layouts.

Export settings

You can define a resolution to use for all exported maps in *Export resolution*. This setting can then be overridden each time you export a map.

Because of some advanced rendering options (*blending mode*, *effects*…), a layout item may need rasterization in order to be exported correctly. QGIS will individually rasterize it without forcing every other item to also be rasterized. This allows printing or saving as PostScript or PDF to keep items as much as possible as vectors, e.g. a map item with layer opacity won't force labels, scale bars, etc to be rasterized too. You can however:

- force all the items to be rasterized checking the *Print as raster* box;
- or use the opposite option, i.e. *Always export as vectors*, to force the export to keep items as vectors when exported to a compatible format. Note that in some cases, this could cause the output to look different to layout.

Where the format makes it possible (e.g., `.TIF`, `.PDF`) exporting a print layout results by default in a georeferenced file (based on the *Reference map* item in the *General settings* group). For other formats, georeferenced output requires you to generate a world file by checking *Save world file*. The world file is created beside the exported map(s), has the name of the page output with the reference map item and contains information to georeference it easily.

Resize layout to content

Using the *Resize page* tool in this group, you create a unique page composition whose extent covers the current contents of the print layout (with some optional *margins* around the cropped bounds).

Note that this behavior is different from the *crop to content* option in that all the items are placed on a real and unique page in replacement of all the existing pages.

Variables

The *Variables* lists all the variables available at the layout's level (which includes all global and project's variables).

It also allows the user to manage layout-level variables. Click the button to add a new custom layout-level variable. Likewise, select a custom layout-level variable from the list and click the button to remove it.

More information on variables usage in the *General Tools* section.

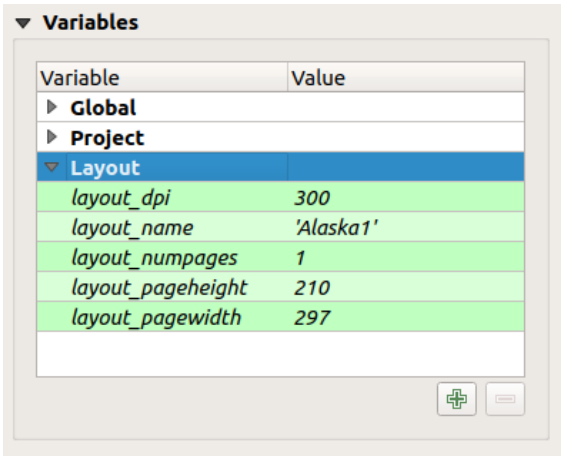


Fig. 18.4: Variables Editor in the Print Layout

Working with the page properties

A layout can be composed of several pages. For instance, a first page can show a map canvas, and a second page can show the attribute table associated with a layer, while a third one shows an HTML frame linking to your organization website. Or you can add many types of items on each page.

Adding a new page

Futhermore, a layout can be made using different size and/or orientation of pages. To add a page, select the *Add Pages…* tool from the *Layout* menu or *Layout Toolbar*. The *Insert Pages* dialog opens and you are asked to fill:

- the number of pages to insert;
- the position of the page(s): before or after a given page or at the end of the print layout;
- The *Page size*: it could be of a preset format page (`A4`, `B0`, `Legal`, `Letter`, `ANSI A`, `Arch A` and their derivatives as well as a resolution type, such as `1920x1080` or `1024x768`) with associated *Orientation* (Portrait or Landscape).

  The page size can also be of a `custom` format; In that case, you'd need to enter its *Width* and *Height* (with locked size ratio if needed) and select the unit to use among `mm`, `cm`, `px`, `pt`, `in`, `ft`… Conversion of entered values is automatically applied when switching from one unit to another.

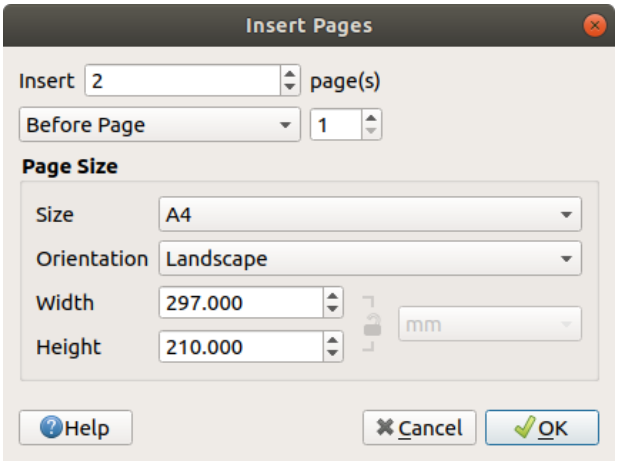


Fig. 18.5: Creating a new page in the Print Layout

Updating page properties

Any page can be later customized through the Page *Item Properties* panel. To access a page's properties, either:

- Left-click on an empty section of the page
- Right-click on a page and select *Page Properties...*.
- Go to *Layout* ► *Page properties...* while hovering over the page

The *Item Properties* panel opens with settings such as:

- for layouts with several pages, it displays the active page number and the count of available pages
- the *Page size* frame described above. You can modify each property using the data defined override options (see *Explore Data-defined override buttons with atlas* for a use case);
- the *Background* of the current page using the *color* or *symbol* you want.
- the *Apply to all Pages* button updates the current page properties to the other existing pages in the layout;
- the *Exclude page from exports* to control whether the current page with its content should be included in the *layout output*;

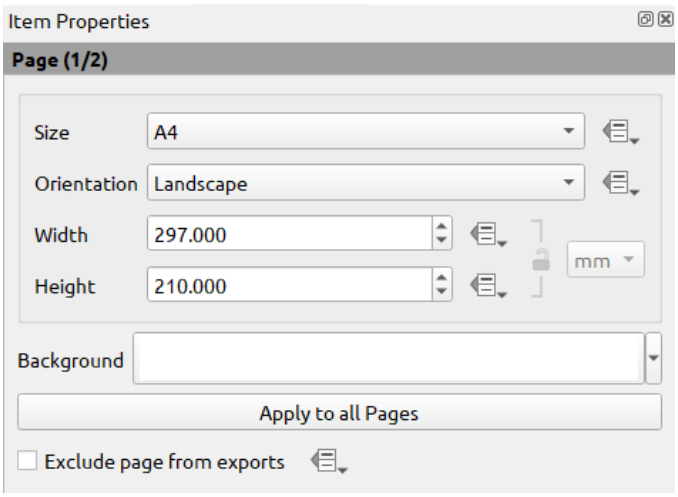


Fig. 18.6: Page properties dialog

The Guides Panel

Guides are vertical or horizontal line references you can place on a layout page to assist you on items placement, when creating, moving or resizing them. To be active, guides require the *View* ► *Show Guides* and *View* ► *Snap to Guides* options to be checked. To create a guide, there are two different methods:

- if the *View* ► *Show Rulers* option is set, drag out a ruler and release the mouse button within the page area, at the desired position.
- for more precision, use the *Guides* panel from the *View* ► *Toolbox* ► or by selecting *Manage guides for page…* from the page's contextual menu.

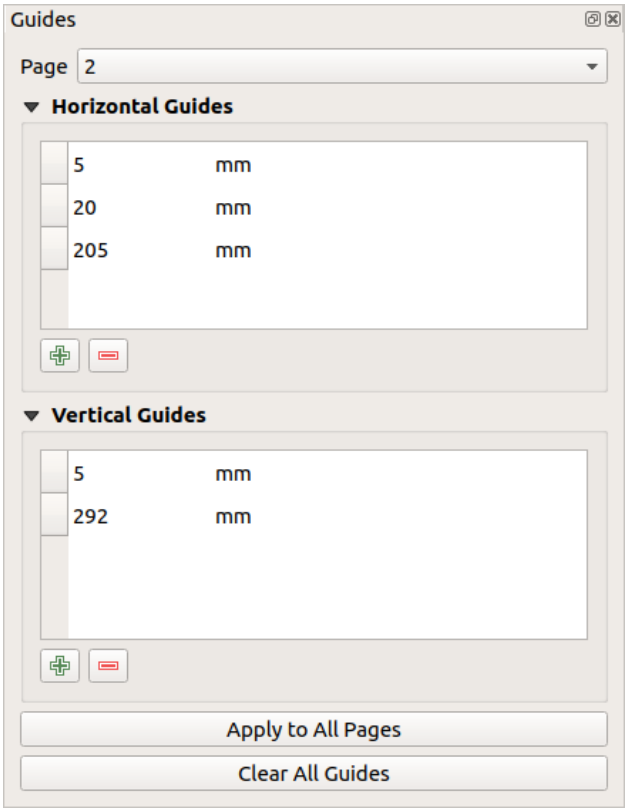


Fig. 18.7: The Guides panel

The *Guides* panel allows creation of snap lines at specific locations:

1. Select the *Page* you'd like to add the guides to
2. Click the Add new guide button and enter the coordinates of the horizontal or vertical line. The origin is at the top left corner. Different units are available for this.

   The panel also allows adjusting the position of existing guides to exact coordinates: double-click and replace the value.
3. The *Guides* panel lists only the items for the current page. It allows creation or removal of guides only in the current page. However, you can use the *Apply to All Pages* button to replicate the guide configuration of the current page to the other pages in the layout.
4. To delete a guide, select it and press the Remove selected guide button. Use *Clear All Guides* to remove all the guides in the current page.

> **Tip**
>
> **Snapping to existing layout items**
>
> Other than guides and grids, you can use existing items as snapping references when moving, resizing or creating new items; these are called **smart guides** and require *View* ► *Smart Guides* option to be checked. Anytime the mouse pointer is close to an item's bound, a snapping cross appears.

The Items Panel

The *Items* panel offers some options to manage selection and visibility of items. All the items added to the print layout canvas (including *items group*) are shown in a list and selecting an item makes the corresponding row selected in the list as well as selecting a row does select the corresponding item in the print layout canvas. This is thus a handy way to select an item placed behind another one. Note that a selected row is shown as bold. Multiple items selection is also possible holding `Shift` for contiguous items, and `Ctrl` key for non-contiguous ones.

For any available item, you can:

-  set it visible or not;
-  lock or unlock its position;
- sort its Z position. You can move up and down each item in the list with a click and drag. The upper item in the list will be brought to the foreground in the print layout canvas. By default, a newly created item is placed in the foreground.
- change the item ID by double-clicking the text;
- right-click an item and select whether to copy or delete it or open its *properties panel*.

Once you have found the correct position for an item, you can lock it by ticking the box in  column. Locked items are **not** selectable on the canvas. Locked items can be unlocked by selecting the item in the *Items* panel and unchecking the tickbox or you can use the icons on the toolbar.

The Undo History Panel: Revert and Restore actions

During the layout process, it is possible to revert and restore changes. This can be done with the revert and restore tools available in the *Edit* menu, the *Layout* toolbar or the contextual menu any time you right-click in the print layout area:

-  Revert last change
-  Restore last change

This can also be done by mouse click within the *Undo history* panel (see Fig. 18.8). The History panel lists the last actions done within the print layout. Select the point you want to revert to and once you do a new action all the actions done after the selected one will be removed.

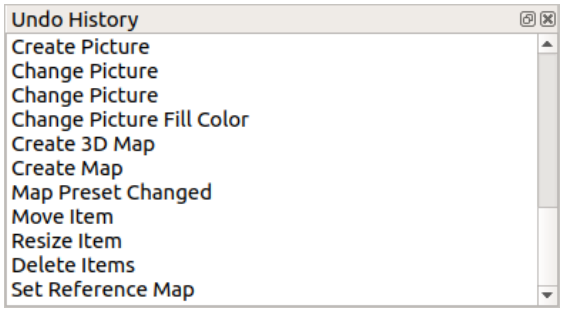


Fig. 18.8: Undo History in the Print Layout

18.2 Layout Items

18.2.1 Layout Items Common Options

QGIS provides a large set of items to layout a map. They can be of map, legend, scale bar, picture, table, north arrow, image type… They however share some common options and behavior that are exposed below.

Creating a layout item

Items can be created using different tools, either from scratch or based on existing items.

To create a layout item from scratch:

1. Select the corresponding tool either from the *Add Item* menu or the *Toolbox* bar.
2. Then:
   - Click on the page and fill the size and placement information requested in the *New Item Properties* dialog that pops up (for details, see *Position and Size*);

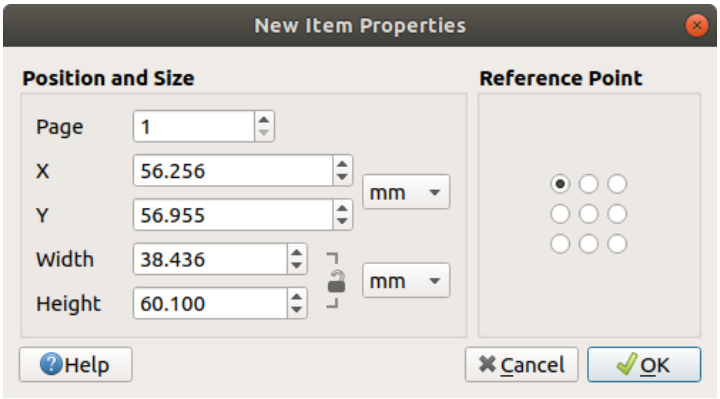


Fig. 18.9: New Item properties dialog

   - Or click-and-drag to define the initial size and placement of the item. You can rely on *grids and guides* snapping for a better position.

> **Note**
>
> Because they can have particular shapes, drawing node or arrow items does not work with one-click nor click-and-drag methods; you need to click and place each node of the item. See *The Node-Based Shape Items* for more details.

You can also:

1. Select an existing item with the  Select/Move item button from the *Toolbox* toolbar
2. Use the contextual menu or the *Edit* menu tools to copy/cut the item and paste it at the mouse position as a new item.

   You can also use the *Paste in Place* (`Ctrl+Shift+V`) command to duplicate an item from one page to another and place it in the new page at the same coordinates as the original.

Moreover, you can create items using a print layout template (for details, see *The Layout Manager*) through the *Layout* ► *Add Items from Template…* command.

> **Tip**
>
> **Add layout items using the file browser**
>
> From your file browser or using the *Browser* panel, drag-and-drop a print layout template (`.qpt` file) onto a print layout dialog and QGIS automatically adds all items from that template to the layout.

Interacting with layout items

Each item inside the print layout can be moved and resized to create a perfect layout. For both operations the first step is to activate the  Select/Move item tool and click on the item.

You can select multiple items with the  Select/Move item button: click and drag over the items or hold the `Shift` button and click on each of the items you want. To deselect an item, click on it holding the `Shift` button.

Each time there's a selection, count of selected items is displayed on the status bar. Inside the *Edit* menu, you can find actions to select all the items, clear all selections, invert the current selection and more…

Moving and resizing items

Unless *View* ► *Show Bounding Boxes* option is unchecked, a selected item will show squares on its boundaries; moving one of them with the mouse will resize the item in the corresponding direction. While resizing, holding `Shift` will maintain the aspect ratio. Holding `Alt` will resize from the item center.

To move a layout item, select it with the mouse and move while holding the left button. If you need to constrain the movements to the horizontal or vertical axis, hold the `Shift` button on the keyboard while moving the mouse. You can also move a selected item using the `Arrow keys` on the keyboard; if the movement is too slow, you can speed it up by holding `Shift`. If you need better precision, use the *Position and size* properties, or grid/guides snapping as explained above for item's creation.

Resizing or moving several items at once is made the same way as for a single item. QGIS however provides some advanced tools to automatically resize a selection of items following different rules:

- each item height matches the  tallest or the  shortest selected item;
- each item width matches the  widest or the  narrowest selected item;
- resizes items to  squares: each item is enlarged to shape a square.

Likewise, there are *tools* available to organize multiple items position by distributing equidistantly:

- edges (left, right, top or bottom) of items;
- centers of items horizontally or vertically;
- spacing between items horizontally or vertically.

Grouping items

Grouping items allows you to manipulate a set of items like a single one: you can easily resize, move, delete, copy the items as a whole.

To create a group of items, select more than one and press the . *Group* button on the *Items* menu or the *Actions* toolbar, or from the right-click menu. A row named `Group` is added to the *Items* panel and can be locked or hidden like any other *Items panel's object*. Grouped items are **not individually** selectable on the canvas; use the Items panel for direct selection and access the item's properties panel.

Locking items

Once you have found the correct position for an item, you can lock it by using the *Lock selected items* button in the *Items* menu or the *Actions* toolbar, or ticking the box next to the item in the *Items* panel. Locked items are **not** selectable on the canvas.

Locked items can be unlocked by selecting the item in the *Items* panel and unchecking the tickbox or you can use the icons on the toolbar.

Alignment and distribution

Tools for raising or lowering the Z position of items in the layout are inside the Raise selected items pull-down menu. Choose an element on the print layout canvas and select the matching functionality to raise or lower the selected element over the other elements. This order is shown in the *Items* panel. You can also raise or lower objects in the *Items* panel by clicking and dragging an object's label in this list.

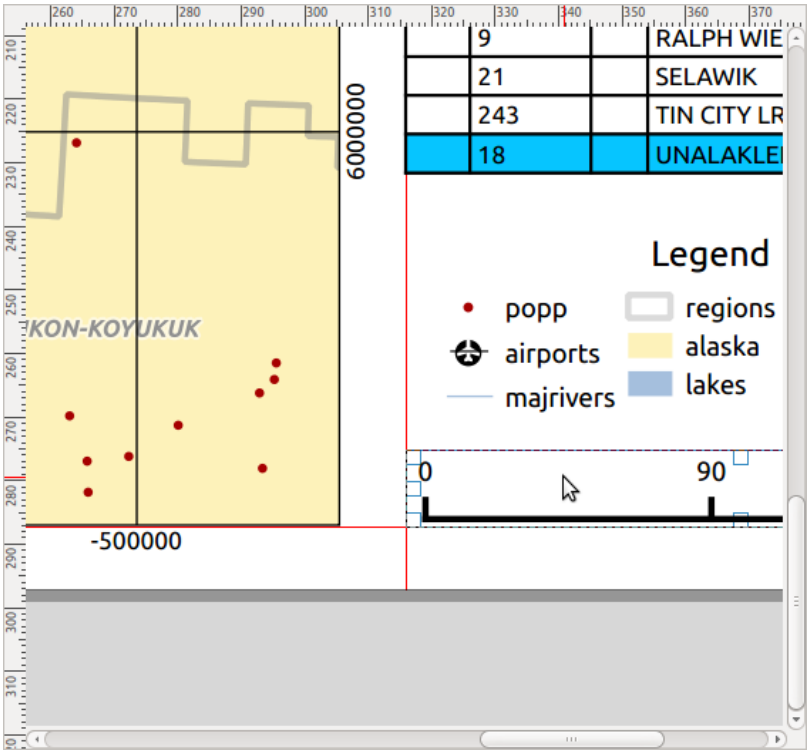


Fig. 18.10: Alignment helper lines in the print layout

There are several alignment options available within the Align selected items pull-down menu (see Fig. 18.10). To use an alignment function, you first select the elements and then click on one of the alignment icons:

- *Align Left* or *Align Right*;
- *Align Top* or *Align Bottom*;
- *Align Center* horizontally or *Align Center Vertical*.

All selected elements will then be aligned to their common bounding box. When moving items on the layout canvas, alignment helper lines appear when borders, centers or corners are aligned.

Another way to improve layout items placement is by adjusting the spacing between them, over the layout page. This can be done by selecting the items and press the *Distribute Left Edges* drop-down menu to:

- *Distribute Left Edges* or *Distribute Right Edges* of items equidistantly
- *Distribute Top edges* or *Distribute Bottom Edges* of items equidistantly
- *Distribute Horizontal Centers* or *Distribute Vertical Centers* of items equidistantly
- Add equal space between items: *Distribute Horizontal Spacing Equally* or *Distribute Vertical Spacing Equally*

Items Common Properties

Layout items have a set of common properties you will find at the bottom of the *Item Properties* panel: Position and size, Rotation, Frame, Background, Item ID, Variables and Rendering (see Fig. 18.11).

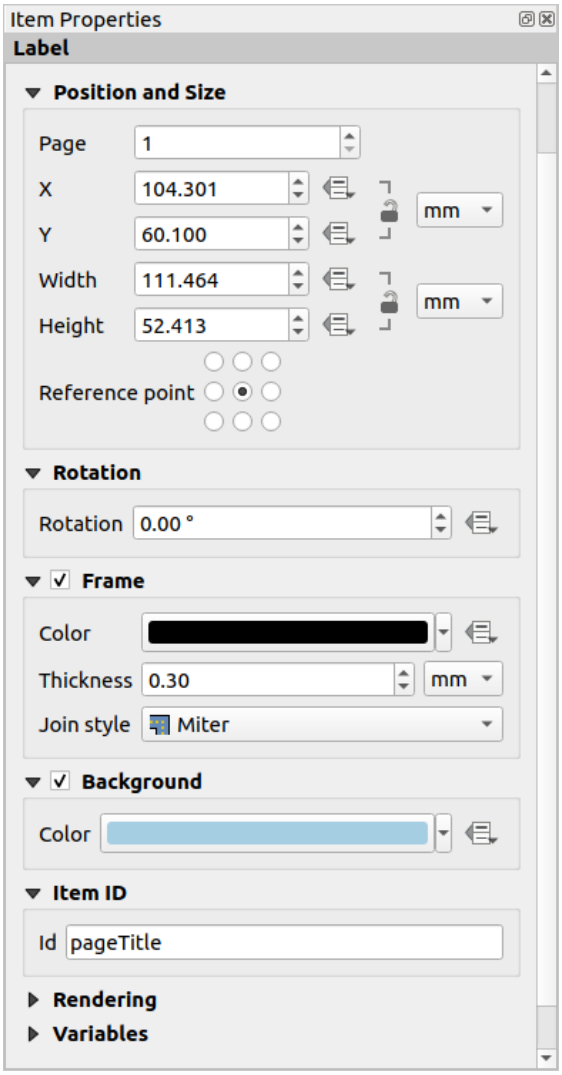


Fig. 18.11: Common Item Properties groups

Note

The Data defined override icon next to most of the options means that you can associate that property with a layer, features attributes, geometry or with any other layout item's property, using *expressions* or *variables*. For more information see *Data defined override setup*.

- The *Position and size* group lets you define the size and position of the frame which contains the item (see *Position and Size* for more information).
- The *Rotation* sets the rotation of the element (in degrees).
- The *Frame* shows or hides the frame around the item. Use the *Color*, *Thickness* and *Join style* widgets to adjust those properties.

- In the *Background* group you can create or pick a background *Color*. Transparency can be adjusted through altering the alpha field settings.
- Use the *Item ID* to create a relationship to other print layout items. This is used with QGIS server and other potential web clients. You can set an ID on an item (for example, a map or a label), and then the web client can send data to set a property (e.g., label text) for that specific item. The GetProjectSettings command will list the items and IDs which are available in a layout.
- *Rendering* mode helps you set whether and how the item can be displayed: you can, for instance, apply *blending mode*, adjust the opacity of the item or *Exclude item from exports*.

Position and Size

Extending the features of the *New Item Properties* dialog with data-defined capabilities, this group allows you to place the items accurately.

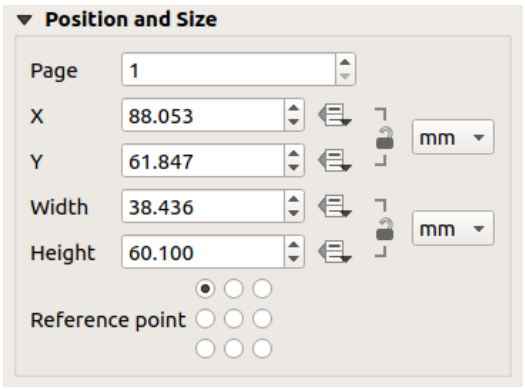


Fig. 18.12: Position and size

- the actual number of the page to place the item on;
- the reference point of the item;
- the *X* and *Y* coordinates of the *Reference point* of the item on the chosen page. The ratio between these values can be locked by clicking on the button. Changes made to a value using the widget or the Select/Move item tool will be reflected in both of them;
- the *Width* and *Height* of the item bounding box. As for coordinates, the ratio between width and height can be locked.

Rendering mode

QGIS allows advanced rendering for layout items just like vector and raster layers.

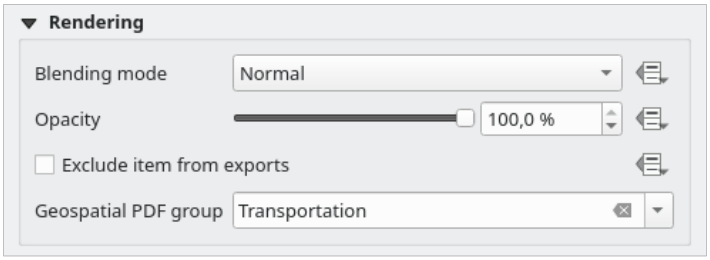


Fig. 18.13: Rendering mode

- *Blending mode*: With this tool you can achieve effects which would otherwise only be achieved using graphic rendering software. The pixels of your overlaying and underlaying items can be mixed according to the mode set (see *Blending Modes* for description of each effect).
- *Opacity*: You can make the underlying item in the layout visible with this tool. Use the slider to adapt the visibility of your item to your needs. You can also make a precise definition of the percentage of visibility in the menu beside the slider.
- *Exclude item from exports*: Combined with data-defined expressions, you can decide to make an item invisible in some exports. After activating this checkbox, the item will not be included in export to PDF, print etc..
- *Geospatial PDF group*: When set, a matching layer tree group will be created in the exported Geospatial PDF and the item will only be visible when this group is checked. This allows content to be selectively displayed as a group by viewers of the Geospatial PDF. E.g., it can allow extra layout content such as descriptive labels or legends to only be shown when layers from the group are visible, making the file export much more flexible.

Variables

The *Variables* lists all the variables available at the layout item's level (which includes all global, project and layout's variables). Layout map items alhave an additional *Map settings* section for variables that provide easy access to values like the map's scale, extent, and so on.

In *Variables*, it's also possible to manage layout item level variables. Click the button to add a new custom variable. Likewise, select any custom item-level variable from the list and click the button to remove it.

More information on variables usage in the *Storing values in Variables* section.

18.2.2 The Map Item

The map item is the main frame that displays the map you've designed in the map canvas. Use the *Add Map* tool following *items creation instructions* to add a new map item that you can later manipulate the same way as exposed in *Interacting with layout items*.

By default, a new map item shows the current status of the *map canvas* with its extent and visible layers. You can customize it thanks to the *Item Properties* panel. Other than the *items common properties*, this feature has the following functionalities:

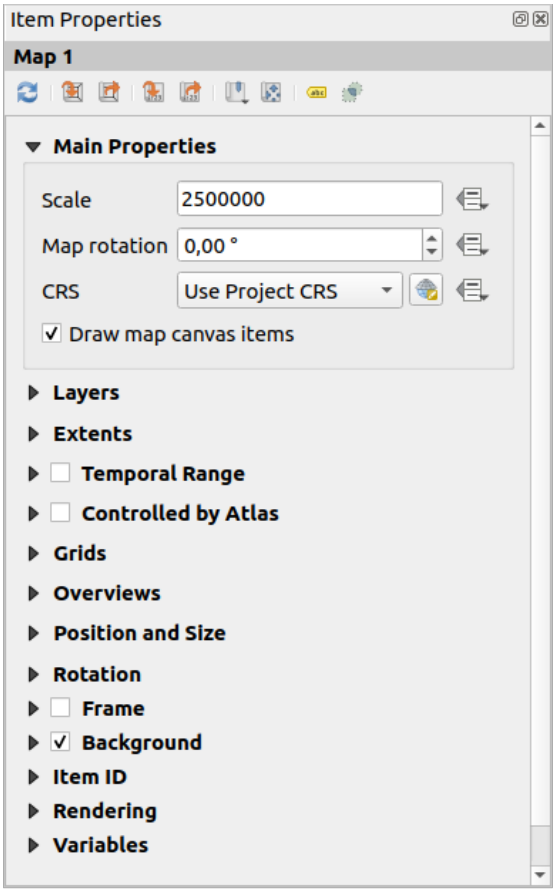


Fig. 18.14: Map Item Properties Panel

The Toolbar

The Map *Item Properties* panel embeds a toolbar with the following functionalities:

- Update map preview
- Set map canvas to match main canvas extent
- View current map extent in main canvas
- Set map scale to match main canvas scale
- Set main canvas to match current map scale
- Bookmarks: set the map item extent to match an existing spatial bookmark
- Interactively edit map extent: pan and zoom interactively within the map item
- Labeling settings: control feature label behaviour (placement, visibility…) in the layout map item extent:
  - set a *Margin from map edges*, a data definable distance from the map item's limits inside which no label should be displayed
  - *Allow truncated labels on edges of map*: controls whether labels which fall partially outside of the map item allowed extent should be rendered. If checked, these labels will be shown (when there's no way to place them fully within the visible area). If unchecked then partially visible labels will be skipped.

  - *Label blocking items*: allows other layout items (such as scalebars, north arrows, inset maps, etc) to be marked as a blockers for the map labels in the **active** map item. This prevents any map labels from being placed under those items - causing the labeling engine to either try alternative placement for these labels or discard them altogether.

    If a *Margin from map edges* is set, the map labels are not placed closer than the specified distance from the checked layout items.

  - *Show unplaced labels*: can be used to determine whether labels are missing from the layout map (e.g. due to conflicts with other map labels or due to insufficient space to place the label) by highlighting them in a *predefined color*.

- Clipping settings: allows to clip the map item to the atlas feature and to shape and polygon items:

  - *Clip to atlas feature*: you can determine that the layout map item will be clipped automatically to the current *atlas feature*.

    There are different clipping modes available:

    * *Clip During Render Only*: applies a painter based clip, so that portions of vector features which sit outside the atlas feature become invisible
    * *Clip Feature Before Render*: applies the clip before rendering features, so borders of features which fall partially outside the atlas feature will still be visible on the boundary of the atlas feature
    * *Render Intersecting Features Unchanged*: renders all features which intersect the current atlas feature, but without clipping their their geometry.

    You can *Force labels inside atlas feature*. If you don't want to *Clip all layers* to the atlas feature you can use the *Clip selected layers* option.

  - *Clip to item*: it is possible to change the shape of the map item by using a *shape* or *polygon* item from the print layout. When you enable this option the map will be automatically clipped to the selected shape in the combobox. Again, the above mentioned clipping modes are available and labels can be forced to display only inside the clipping shape.

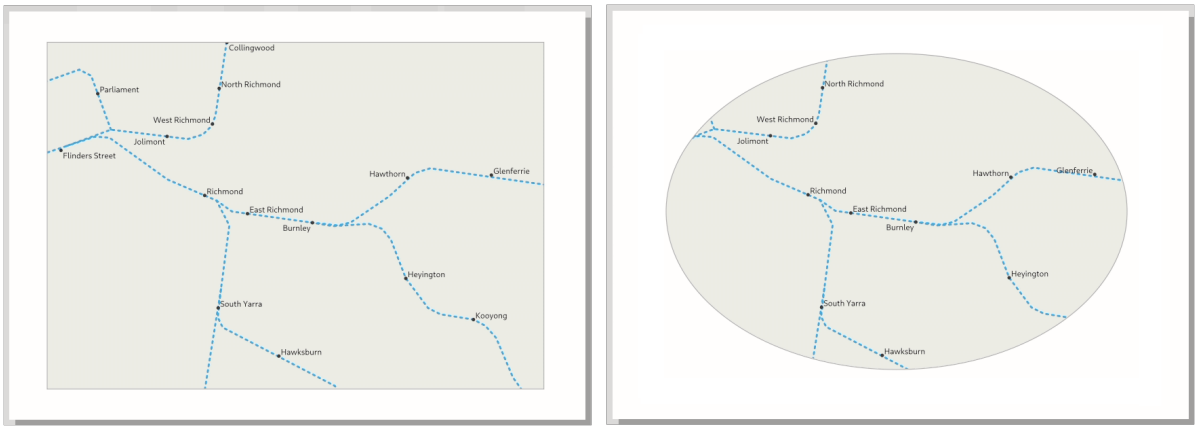


Fig. 18.15: Clipping a layout map item to shapes

Main properties

In the *Main properties* group (see Fig. 18.14) of the map *Item Properties* panel, available options are:

- The *Update Preview* button to refresh the map item rendering if the view in map canvas has been modified. Note that most of the time, the map item refresh is automatically triggered by the changes;
- The *Scale* to manually set the map item scale;
- The *Map rotation* allows you to rotate the map item content clockwise in degrees. The rotation of the map canvas can be imitated here;
- The *CRS* allows you to display the map item content in any *CRS*. It defaults to `Use project CRS`;
- *Draw map canvas items* lets you show in the print layout *annotations* that are placed on the main map canvas.

Layers

By default, map item appearance is synced with the map canvas rendering meaning that toggling visibility of the layers or modifying their style in the *Layers Panel* is automatically applied to the map item. Because, like any other item, you may want to add multiple map items to a print layout, there's a need to break this synchronization in order to allow showing different areas, layer combinations, at different scales… The *Layers* properties group (see Fig. 18.16) helps you do that.

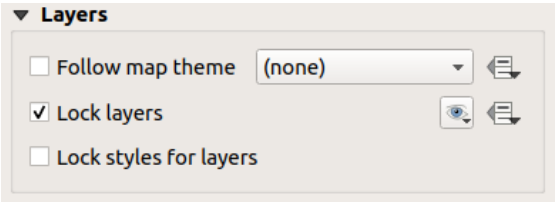


Fig. 18.16: Map Layers group

If you want to keep the map item consistent with an existing *map theme*, check *Follow map theme* and select the desired theme in the drop-down list. Any changes applied to the theme in QGIS' main window (using the replace theme function) will automatically affect the map item. If a map theme is selected, the *Lock styles for layers* option is disabled because *Follow map theme* also updates the style (symbology, labels, diagrams) of the layers.

To lock the layers shown in a map item to the current map canvas visibility, check *Lock layers*. When this option is enabled, any changes on the layers' visibility in QGIS' main window will not affect the layout's map item. Nevertheless, style and labels of locked layers are still refreshed according to QGIS' main window. You can prevent this by using *Lock styles for layers*.

Instead of using the current map canvas, you can also lock the layers of the map item to those of an existing map theme: select a map theme from the Set layer list from a map theme drop-down button, and the *Lock layers* is activated. The set of visible layers in the map theme is from now on used for the map item until you select another map theme or uncheck the *Lock layers* option. You then may need to refresh the view using the Refresh view button of the *Navigation* toolbar or the *Update Preview* button seen above.

Note that, unlike the *Follow map theme* option, if the *Lock layers* option is enabled and set to a map theme, the layers in the map item will not be refreshed even if the map theme is updated (using the replace theme function) in QGIS' main window.

Locked layers in the map item can also be *data-defined*, using the icon beside the option. When used, this overrides the selection set in the drop-down list. You need to pass a list of layers separated by `|` character. The following example locks the map item to use only layers `layer 1` and `layer 2`:

```
concat ('layer 1', '|', 'layer 2')
```

Extents

The *Extents* group of the map item properties panel provides the following functionalities (see Fig. 18.17):

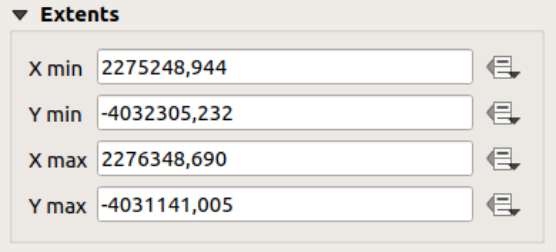


Fig. 18.17: Map Extents group

The **Extents** area displays `X` and `Y` coordinates of the area shown in the map item. Each of these values can be manually replaced, modifying the map canvas area displayed and/or map item size. The extent can also be modified using tools at the top of the map item panel such as:

- Set map canvas to match main canvas extent
- Set map scale to match main canvas scale

You can also alter a map item extent using the Move item content tool: click-and-drag within the map item to modify its current view, keeping the same scale. With the tool enabled, use the mouse wheel to zoom in or out, modifying the scale of the shown map. Combine the movement with `Ctrl` key pressed to have a smaller zoom.

Elevation range

The *Elevation range* setting in the map item properties allows you to filter the content of certain layers based on a specific elevation range. While all layers remain visible, those that support elevation filtering (currently point clouds and raster DEMs) will have their data filtered. Only the parts of these layers that fall within the elevation range set by the *Lower* and *Upper* values will be displayed.

The *Elevation range* can be data-defined. This means you can set different elevation ranges for different features in the *atlas* or *report*.

Temporal range

The *Temporal range* group of the map item properties panel provides the options to control layers rendering in the map item based on a temporal range. Only layers whose temporal properties overlap with the time range set by the *Start* and *End* dates are displayed in the map item.

The associated data-defined widgets help make the time range dynamic, and allow outputting temporal *atlases*, i.e. automated maps with fixed spatial extent and whose contents vary based on time. For example, using as coverage layer a csv file with a start and end pair of fields and a number of rows representing date ranges, enable both the temporal range and control by atlas in the map item properties and hit atlas export.

Controlled by atlas

The *Controlled by atlas* group properties is available only if an *atlas* is active in the print layout. Check this option if you want the map item being ruled by the atlas; when iterating over the coverage layer, the map item extent is panned/zoomed to the atlas feature following:

- *Margin around features*: zooms to the feature at the best scale, keeping around each a margin representing a percentage of the map item width or height. The margin can be the same for all features or *set variable*, e.g., depending on map scale;
- *Predefined scale (best fit)*: zooms to the feature at the project *predefined scale* where the atlas feature best fits;
- *Fixed scale*: atlas features are panned from one to another, keeping the same scale of the map item. Ideal when working with features of same size (e.g., a grid) or willing to highlight size differences among atlas features.

Grids

With grids, you can add, over your map, information relative to its extent or coordinates, either in the map item projection or a different one. The *Grids* group provides the possibility to add several grids to a map item.

- With the and buttons you can add or remove a selected grid;
- With the and buttons you can move up and down a grid in the list, hence move it on top or bottom of another one, over the map item.

Double-click the added grid to rename it.

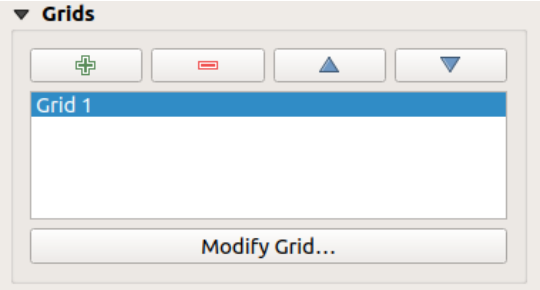


Fig. 18.18: Map Grids Dialog

To modify a grid, select it and press the *Modify Grid…* button to open the *Map Grid Properties* panel and access its configuration options.

Grid Appearance

In the *Map Grid Properties* panel, check *Grid enabled* to show the grid on the map item.

As grid type, you can specify to use a:

- **Solid**: shows a line across the grid frame. The *Line style* can be customized using *color* and *symbol* selector widget;
- **Cross**: displays segment at the grid lines intersection for which you can set the *Line style* and the *Cross width*;
- **Markers**: only displays customizable markers symbol at grid lines intersection;
- or **Frame and annotations only**.

Other than the grid type, you can define:

- the *CRS* of the grid: by default, it will follow the map item CRS. Press Select CRS button to set it to a different CRS.
- the *Interval* type to use for the grid references:
  - **Map Units**: you set a distance within the map (in the unit of the grid CRS) between consecutive grid references in the *X* and *Y* directions. The number of grid ticks will vary depending on the map scale.
  - choosing **Fit Segment Width** will dynamically select the grid interval based on the map extent to a "pretty" interval. That optimal interval is calculated within a range of distances whose *Minimum* and *Maximum* values can be customized.
  - With **Millimeters** or **Centimeters**, you set a distance on the paper between consecutive grid references in the *X* and *Y* directions. The number of grid ticks will be the same whatever the map scale.
- the *Offset* from the map item edges, in the *X* and/or the *Y* direction
- and the *Blend mode* of the grid (see *Blending Modes*) when compatible.

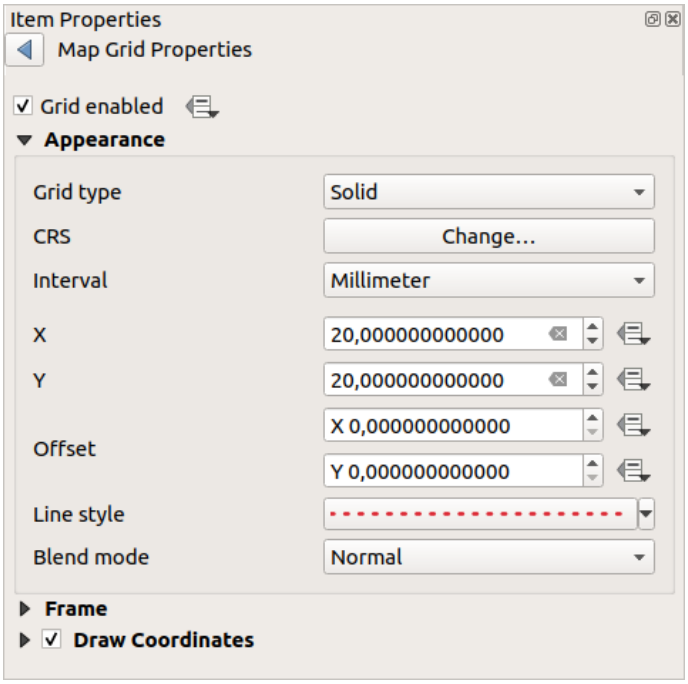


Fig. 18.19: Grid Appearance Dialog

Grid Frame

There are different options to style the frame that holds the map. The following options are available: `No Frame`, `Zebra`, `Zebra (nautical)`, `Interior ticks`, `Exterior ticks`, `Interior and Exterior ticks`, `Line border` and `Line border (nautical)`.

When compatible, it's possible to set the *Frame size*, a *Frame margin*, the *Frame line thickness* with associated color and the *Frame fill colors*.

Using `Latitude/Y only` and `Longitude/X only` values in the divisions section you can prevent a mix of latitude/Y and longitude/X coordinates showing on each side when working with rotated maps or reprojected grids. Also you can choose to set visible or not each side of the grid frame.

When the map item extent is rotated (from the *Main properties* group) or the grid has a different CRS applied, grid lines may not be orthogonal to the map item sides. This can result in bad looking of the grid when styled with interior

and/or exterior ticks. Checking *Follow grid rotation* will align the ticks with grid lines. Moreover, you can adjust some more properties:

- *Ticks alignment*: The interior and/or exterior ticks will be parallel to their corresponding grid line. Their alignment can be:
  - **Orthogonal**: ticks on the same side end at one line, parallel to the side. This can result e.g. in some ticks getting longer when with a low angle to the frame.
  - **Fixed length**: all ticks have the same length, so they may not align
- *Skip below angle*: prevents displaying ticks for grid lines intersecting the frame border below a specified threshold
- *Margin from map corner*: prevents displaying ticks too close to the map corners, because they could overlap and/or be out of bounds.

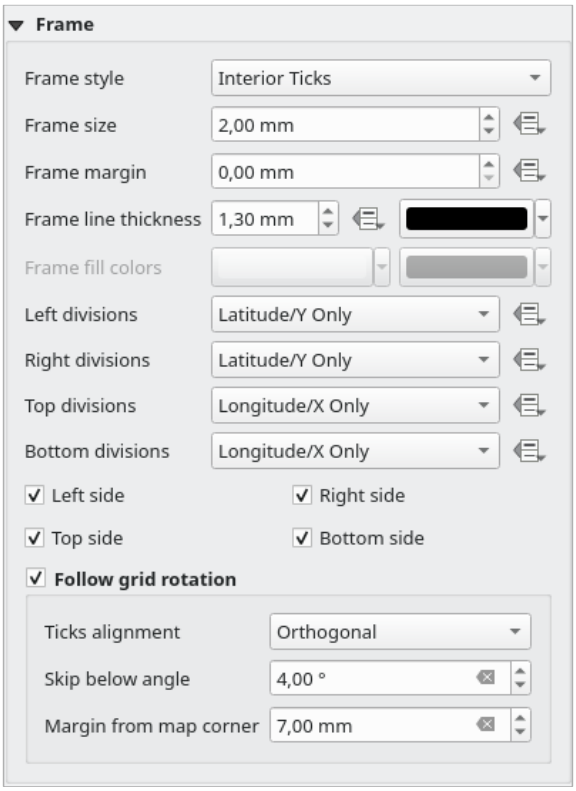


Fig. 18.20: Grid Frame Dialog

Coordinates

The *Draw coordinates* checkbox allows you to add coordinates to the map frame. Displayed values relate to the chosen *grid interval* unit. You can choose the annotation numeric format, the options range from decimal to degrees, minute and seconds, with or without suffix, aligned or not and a custom format using the expression dialog.

For each of the *Left*, *Right*, *Top* and *Bottom* sides of the grid frame, you can indicate:

- whether to render the coordinates: **Show all**, **Show latitude/Y only**, **Show longitude/X only**, **Disabled**. Showing only Latitude/Y or Longitude/X values in the divisions helps prevent a mix of latitude/Y and longitude/X coordinates showing on each side when working with rotated maps or reprojected grids.
- the relative position of the text to the grid frame: **Outside frame** or **Inside frame**
- the placement and orientation of the annotation:

- **Horizontal**
- **Vertical ascending**, **Vertical descending**
- **Boundary direction**
- **Above tick**, **On tick**, **Under tick** when a tick-based frame is used

You can also define the *Font properties* (font, size, color, buffer,…) the *Distance to the map frame* and the *Coordinate precision* (number of decimals) for the drawn annotations.

Follow grid rotation: available when the map extent is rotated or the grid is reprojected, it helps you adjust the annotations placement. Depending on the selected placement mode, the annotations are also rotated:

- *Annotations alignment*: it can be **Orthogonal** or of **Fixed length**
- *Skip below angle*: prevents displaying annotations for grid lines intersecting the frame border below a specified threshold
- *Margin from map corner*: prevents displaying annotations too close to the map corners, because they could overlap and/or be out of bounds.

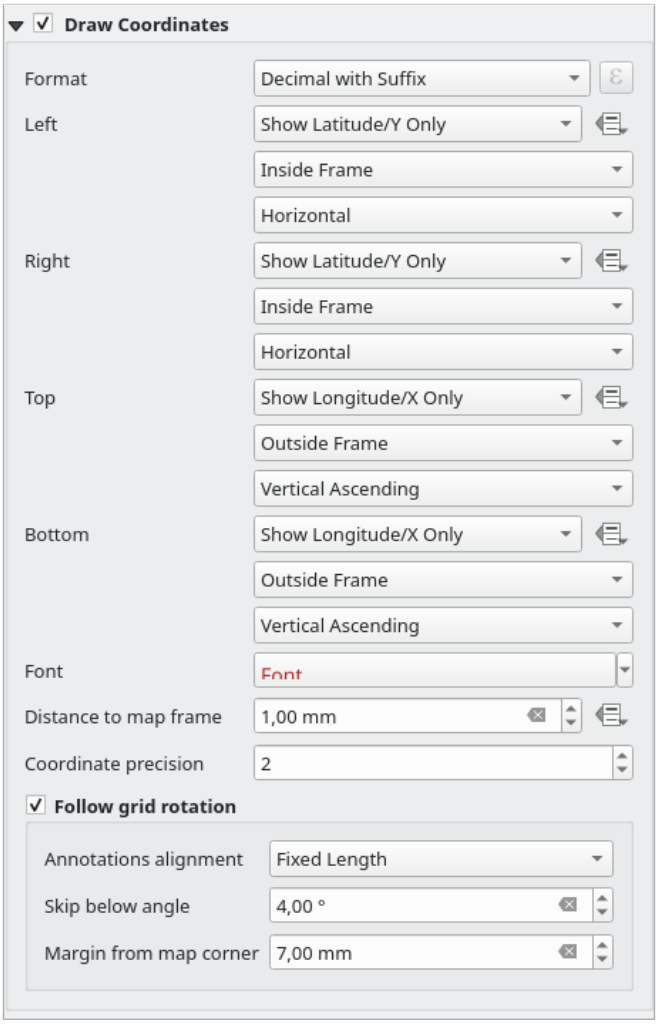


Fig. 18.21: Grid Draw Coordinates dialog

Overviews

Sometimes you may have more than one map in the print layout and would like to locate the study area of one map item on another one. This could be for example to help map readers identify the area in relation with its larger geographic context shown in the second map.

The *Overviews* group of the map panel helps you create the link between two different maps extent and provides the following functionalities:

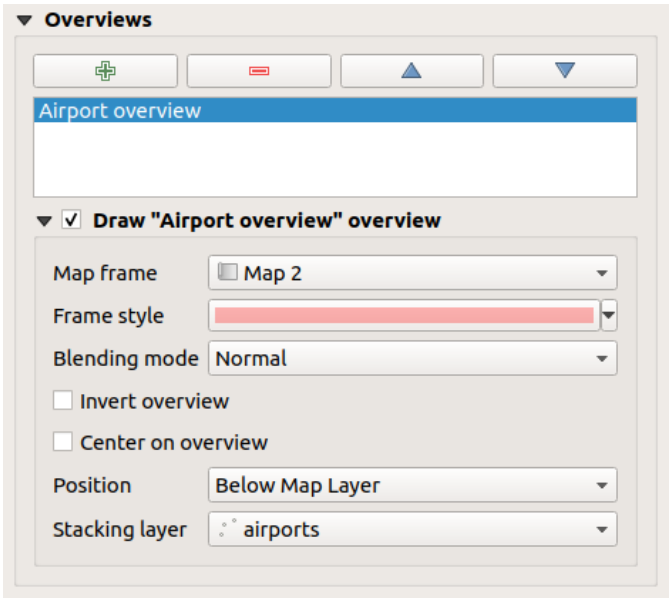


Fig. 18.22: Map Overviews group

To create an overview, select the map item on which you want to show the other map item's extent and expand the *Overviews* option in the *Item Properties* panel. Then press the button to add an overview.

Initially this overview is named 'Overview 1' (see Fig. 18.22). You can:

- Rename it with a double-click
- With the and buttons, add or remove overviews
- With the and buttons, move an overview up and down in the list, placing it above or below other overviews in the map item (when they are at the same *stack position*).

Then select the overview item in the list and check the *Draw "<name_overview>" overview* to enable the drawing of the overview on the selected map frame. You can customize it with:

- The *Map frame* selects the map item whose extents will be shown on the present map item.
- The *Frame Style* uses the *symbol properties* to render the overview frame.
- The *Blending mode* allows you to set different transparency blend modes.
- The *Invert overview* creates a mask around the extents when activated: the referenced map extents are shown clearly, whereas the rest of the map item is blended with the frame fill color (if a fill color is used).
- The *Center on overview* pans the map item content so that the overview frame is displayed at the center of the map. You can only use one overview item to center, when you have several overviews.
- The *Position* controls exactly where in the map item's layer stack the overview will be placed, e.g. allowing an overview extent to be drawn below some feature layers such as roads whilst drawing it above other background layers. Available options are:

- *Below map*
- *Below map layer* and *Above map layer*: place the overview frame below and above the geometries of a layer, respectively. The layer is selected in the *Stacking layer* option.
- *Below map labels*: given that labels are always rendered above all the feature geometries in a map item, places the overview frame above all the geometries and below any label.
- *Above map labels*: places the overview frame above all the geometries and labels in the map item.

18.2.3 The 3D Map Item

The 3D Map item is used to display a *3D map view*. Use the  *Add 3D Map* button, and follow *items creation instructions* to add a new 3D Map item that you can later manipulate the same way as demonstrated in *Interacting with layout items*.

By default, a new 3D Map item is empty. You can set the properties of the 3D view and customize it in the *Item Properties* panel. In addition to the *common properties*, this feature has the following functionalities (Fig. 18.23):

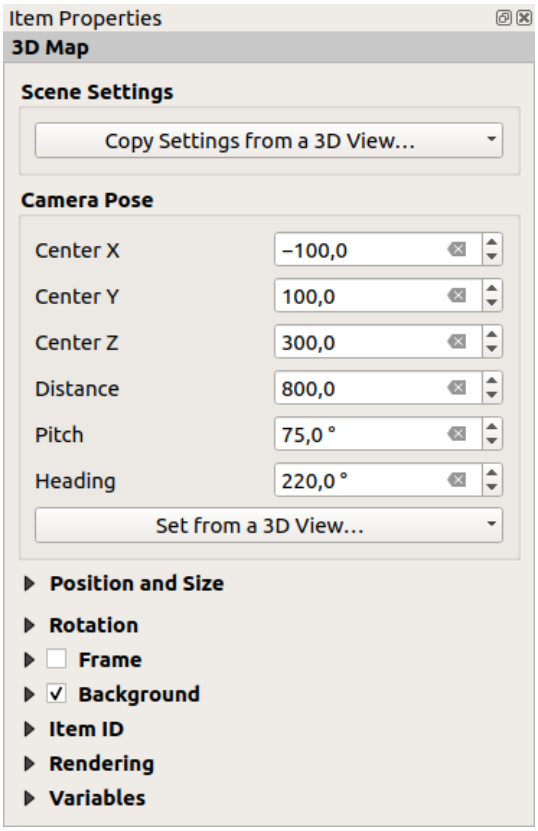


Fig. 18.23: 3D Map Item Properties

Scene settings

Press *Copy Settings from a 3D View…* to choose the 3D map view to display.

The 3D map view is rendered with its current configuration (layers, terrain, lights, camera position and angle…).

Camera pose

- *Center X* sets the X coordinate of the point the camera is pointing at
- *Center Y* sets the Y coordinate of the point the camera is pointing at
- *Center Z* sets the Z coordinate of the point the camera is pointing at
- *Distance* sets the distance from the camera center to the point the camera is pointing at
- *Pitch* sets the rotation of the camera around the X-axis (vertical rotation). Values from 0 to 360 (degrees). 0°: terrain seen straight from above; 90°: horizontal (from the side); 180°: straight from below; 270°: horizontal, upside down; 360°: straight from above.
- *Heading* sets the rotation of the camera around the Y-axis (horizontal rotation - 0 to 360 degrees). 0°/360°: north; 90°: west; 180°: south; 270°: east.

The *Set from a 3D View…* pull-down menu lets you populate the items with the parameters of a 3D View.

18.2.4 The Label Item

The *Label* item is a tool that helps decorate your map with texts that would help to understand it; it can be the title, author, data sources or any other information… You can add a label with the  *Add Label* tool following *items creation instructions* and manipulate it the same way as exposed in *Interacting with layout items*.

By default, the label item provides a default text that you can customize using its *Item Properties* panel. Other than the *items common properties*, this feature has the following functionalities (see Fig. 18.24):

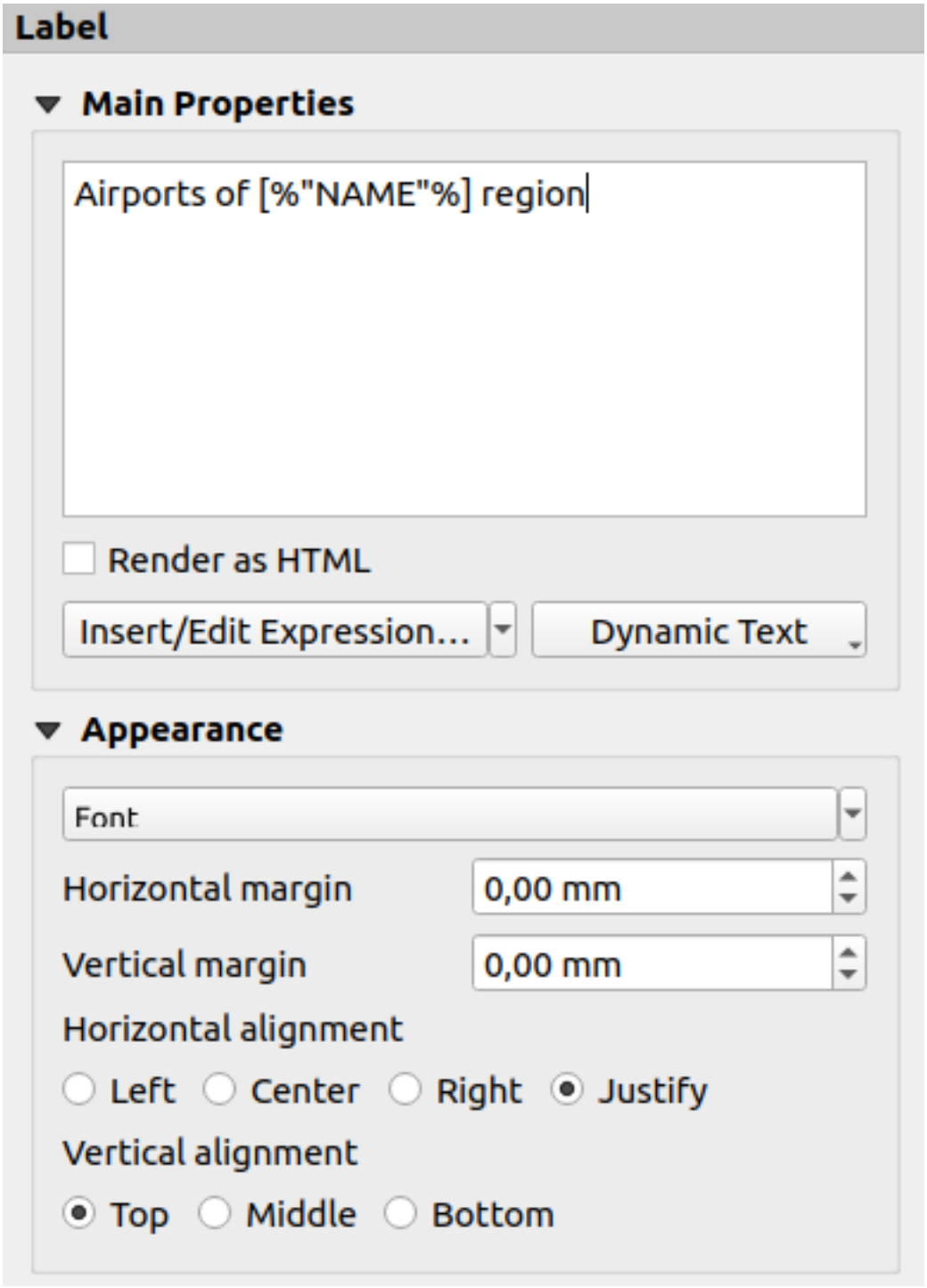


Fig. 18.24: Label Item Properties Panel

Main properties

The *Main properties* group is the place to provide the text of the label. The text can be static, dynamic with *expression* functions and variables, and/or formatted with HTML. Dynamic parts of a label need to be surrounded by `[%` and `%]` in order to be interpreted and evaluated as such.

- To use expressions in labels, you can click on *Insert/Edit Expression…* button, write your formula as usual and when the dialog is applied, QGIS automatically adds the surrounding characters.

  > **Hint**
  >
  > When you click the *Insert/Edit Expression…* button with no selection made in the textbox, the new expression will be appended to the existing text. However, if you want to modify an existing expression, the behavior changes based on your selection: If some text is selected and the selection is within an expression (between '[%' and '%]'), or if no text is selected but the cursor is inside an expression, the whole expression will be selected. In all other cases, the selection will remain as is.

  Because maps are usually filled with some common textual information (date, author, title, page number, …), QGIS provides a direct access to the corresponding expressions or variables: press the *Dynamic text* button to select and insert them into your label.

  > **Tip**
  >
  > The top menu *Add Item* ► *Add Dynamic Text* ► can be used to create a new label item filled with the selected predefined expression.

  It's possible to turn a dynamic label into static: press the drop-down arrow next to the *Insert/Edit Expression…* button and select *Convert to Static*. Any dynamic parts of the label's contents will be evaluated and replaced with their current values. You can then manually tweak the resulting text when needed.

- Labels can be interpreted as HTML code: check *Render as HTML*. You can now insert HTML tags or styles, URL, a clickable image that links to a web page, or something more complex…

The following code combines HTML rendering with expressions, for an advanced labeling and will output Fig. 18.25:

```
<html>
 <head>
   <style>
      /* Define some custom styles, with attribute-based size */
      name {color:red; font-size: [% ID %]px; font-family: Verdana; text-shadow:␣
↪grey 1px 0 10px;}
      use {color:blue;}
   </style>
 </head>

 <body>
   <!-- Information to display -->
   <u>Feature Information</u>
   <ul style="list-style-type:disc">
     <li>Feature Id: [% ID %]</li>
     <li>Airport: <name>[% NAME %]</name></li>
     <li>Main use: <use>[% USE %]</use></li>
   </ul>
   Last check: [% concat( format_date( "control_date", 'yyyy-MM-dd'), ' by <b><i>',
↪ @user_full_name, '</i></b>' ) %]

   <!-- Insert an image -->
   <p align=center><img src="path/to/logos/qgis-logo-made-with-color.svg" alt=
↪"QGIS icon" style="width:80px;height:50px;"</p>
```

(continues on next page)

(continued from previous page)

```
 </body>
</html>
```



Fig. 18.25: Leveraging a label with HTML styling

Appearance

- Define font and style of the text by clicking on the *Font* button. In the *Label Font* menu you can use some of the options for *Formatting the label text*.
- You can specify different horizontal and vertical margins in mm. This is the margin from the edge of the layout item. The label can be positioned outside the bounds of the label e.g. to align label items with other items. In this case you have to use negative values for the margin.
- Using the text alignment is another way to position your label. It can be:
  - *Left*, *Center*, *Right* or *Justify* for *Horizontal alignment*
  - and *Top*, *Vertical Center*, *Bottom* for *Vertical alignment*.

> **Tip**
>
> **Resize the layout label item to fit its contents**
>
> By default, the size of a layout label item does not depend on the contained text. It may be too large with useless blank space or too small, hiding part of the text. To automatically resize a label so it fits the text inside it perfectly, double-click one of the square handles on the label. The label will adjust its size to match the text, either expanding or shrinking as needed, while the opposite handle of the label remains fixed in place. This allows you to quickly ensure the label fits the text without manually resizing it. So e.g., double-clicking the left-middle handle causes the left side of the label to move.

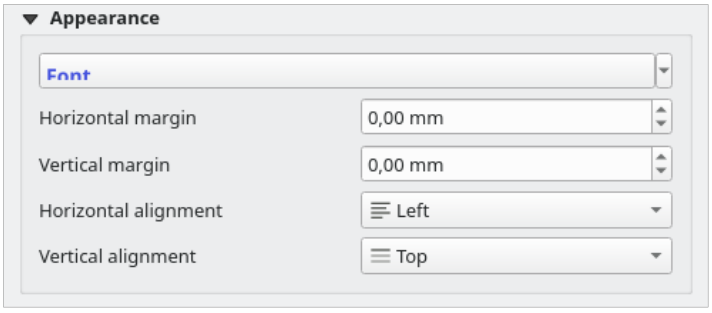


Fig. 18.26: Label Item Appearance Properties

Exploring expressions in a label item

Below some examples of expressions you can use to populate the label item with interesting information - remember that the code, or at least the calculated part, should be surrounded by `[%` and `%]` in the *Main properties* frame:

- Display a title with the current atlas feature value in "field1":

```
'This is the map for ' || "field1"
```

  or, written in the *Main properties* section:

```
This is the map for [% "field1" %]
```

- Add a pagination for processed atlas features (eg, `Page 1/10`):

```
concat( 'Page ', @atlas_featurenumber, '/', @atlas_totalfeatures )
```

- Return the name of the airports of the current atlas region feature, based on their common attributes:

```
aggregate( layer := 'airports',
           aggregate := 'concatenate',
           expression := "NAME",
           filter := fk_regionId = attribute( @atlas_feature, 'ID' ),
           concatenator := ', '
         )
```

  Or, if an *attributes relation* is set:

```
relation_aggregate( relation := 'airports_in_region_relation',
                    aggregate := 'concatenate',
                    expression := "NAME",
                    concatenator := ', '
                  )
```

- Return the name of the airports contained in the current atlas region feature, based on their spatial relationship:

```
aggregate( layer := 'airports',
           aggregate := 'concatenate',
           expression := "NAME",
           filter := contains( geometry( @parent ), $geometry ),
           concatenator := ', '
         )
```

  OR:

```
array_to_string( array:= overlay_contains( layer := 'airports',
                                           expression := "NAME" ),
                 delimiter:= ', '
               )
```

- Return the lower X coordinate of the `Map 1` item's extent:

```
x_min( map_get( item_variables( 'Map 1' ), 'map_extent' ) )
```

- Retrieve the name of the layers in the current layout `Map 1` item, and formats in one name by line:

```
array_to_string(
 array_foreach(
  map_get( item_variables( 'Map 1' ), 'map_layers' ), -- retrieve the layers␣
↪list
  layer_property( @element, 'name' ) -- retrieve each layer name
 ),
```

(continues on next page)

(continued from previous page)

```
 '\n' -- converts the list to string separated by breaklines
)
```

- Display the list of layers with their license strings (usage rights) in a layout `Map 1` item. You need to fill the layers' *Access metadata* properties first.

```
array_to_string( map_credits( 'Map 1', true ) )
```

18.2.5 The Legend Item

The *Legend* item is a box or a table that explains the meanings of the symbols used on the map. A legend is then bound to a map item. You can add a legend item with the  *Add Legend* tool following *items creation instructions* and manipulate it the same way as exposed in *Interacting with layout items*.

By default, the legend item displays all available layers and can be refined using its *Item Properties* panel. Other than the *items common properties*, this feature has the following functionalities (see Fig. 18.27):

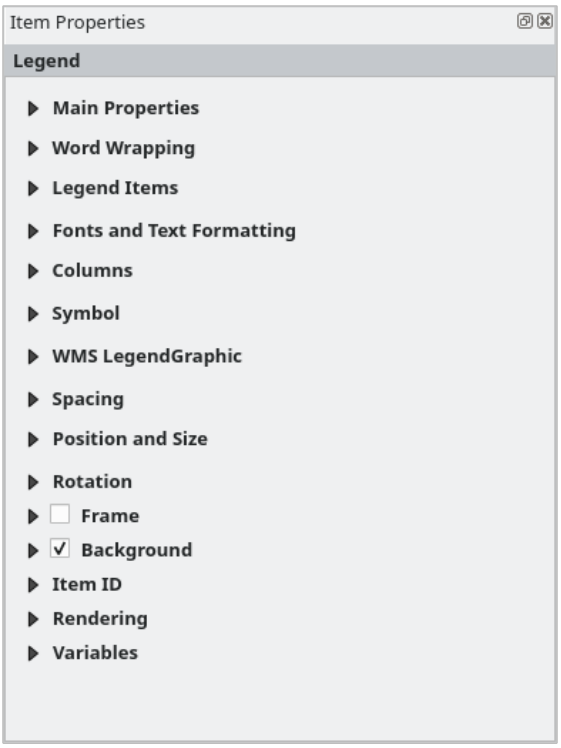


Fig. 18.27: Legend Item Properties Panel

Main properties

The *Main properties* group of the legend *Item Properties* panel provides the following functionalities (see Fig. 18.28):

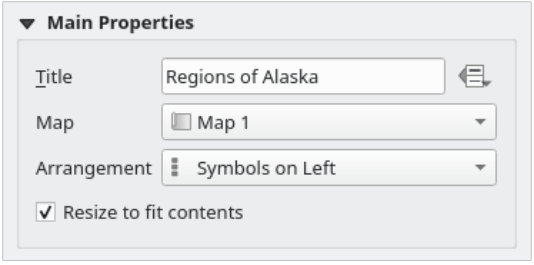


Fig. 18.28: Legend Main properties group

In Main properties you can:

- Change the *Title* of the legend. It can be made dynamic using the *data-defined override* setting, useful for example when generating an atlas;
- Choose which *Map* item the current legend will refer to. By default, the map over which the legend item is drawn is picked. If none, then it falls back to the *reference map*.

> **Note**
>
> *Variables* of the linked map item (@map_id, @map_scale, @map_extent…) are also accessible from data-defined properties of the legend.

- Set the symbols and text placement in the legend: the *Arrangement* can be *Symbols on left* or *Symbols on right*. The default value depends on the locale in use (right-to-left based or not).
- Use *Resize to fit contents* to control whether or not a legend should be automatically resized to fit its contents. If unchecked, then the legend will never resize and instead just stick to whatever size the user has set. Any content which doesn't fit the size is cropped out.

Word wrapping

The *Word wrapping* group of the legend *Item Properties* panel provides the following functionalities:

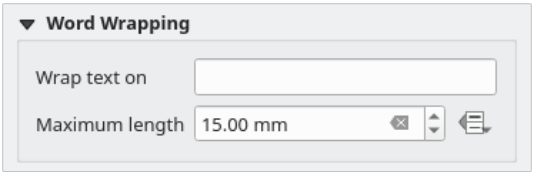


Fig. 18.29: Legend's word wrapping group

- Wrap the text of the legend on a given character: each time the character appears, it's replaced with a line break;
- Allow legend text to be automatically wrapped after a set line length (in millimeters), preventing very wide auto-generated legends. The *Maximum length* can be set as a static or data-defined value, convenient for dynamic layouts which adjust legend size based on for example page orientation or displayed features, or atlases where you may want to tweak the legend appearance on different pages.

  Examples:

- Control legend width based on layout page orientation

```
IF( @layout_pageheight >= @layout_pagewidth, 35, 80 )
```

- Set the legend column width when a specific layer is displayed in the linked map

```
CASE WHEN
  array_contains(
    map_get( item_variables('legend_1'), 'map_layer_ids' ),
    'a_specific_layer_id' )
THEN 60
END
```

Legend items

The *Legend items* group of the legend *Item Properties* panel provides the following functionalities (see Fig. 18.30):

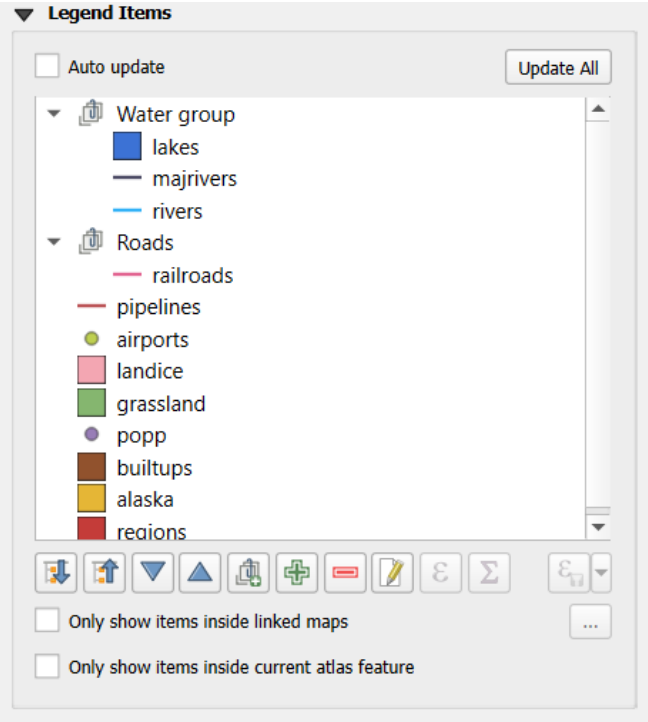


Fig. 18.30: Legend Items group

- The legend will be updated automatically if *Auto update* is checked. When *Auto update* is unchecked this will give you more control over the legend items. All the icons below the legend items list will be activated.
- The legend items window lists all legend items and allows you to change item order, group layers, remove and restore items in the list, edit layer names and symbology and add a filter.
  - Use the Expand all and Collapse all buttons to respectively expand or collapse all the groups and subgroups in the legend tree. Ensure that the *Auto update* is unchecked to use these buttons.
  - The item order can be changed using the and buttons or with 'drag-and-drop' functionality. The order can not be changed for WMS legend graphics.
  - Use the button to add a legend group.
  - Use the button to add layers and button to remove groups, layers or symbol classes.

  - The  button is used to edit the layer, group name or title. First you need to select the legend item. Double-clicking the item also opens the text box to rename it.
  - The  button uses expressions to customize each symbol label of the selected layer (see *Data-define the legend labels*)
  - The  button adds a feature count for each class of vector layer.
  - The  Filter legend by expression helps you filter which of the legend items of a layer will be displayed, i.e. using a layer that has different legend items (e.g., from a rule-based or categorized symbology), you can specify a boolean expression to remove from the legend tree, styles that have no feature satisfying a condition. Note that the features are nevertheless kept and shown in the layout map item.

  While the default behavior of the legend item is to mimic the *Layers* panel tree, displaying the same groups, layers and classes of symbology, right-click any item offers you options to hide layer's name or raise it as a group or subgroup. In case you have made some changes to a layer, you can revert them by choosing *Reset to defaults* from the contextual menu of the legend entry.

  After changing the symbology in the QGIS main window, you can click on *Update All* to adapt the changes in the legend element of the print layout.

- With the  *Only show items inside linked maps*, only the legend items visible in the linked map will be listed in the legend. If you have more than one map you can click on … and select other maps from your layout. This tool remains available when  *Auto-update* is active.
- While generating an atlas with polygon features, you can filter out legend items that lie outside the current atlas feature. To do that, check the  *Only show items inside current atlas feature* option.

Data-define the legend labels

 allows you to add *expressions* to each symbol label of a given layer. New variables (`@symbol_label`, `@symbol_id` and `@symbol_count`) help you interact with the legend entry.

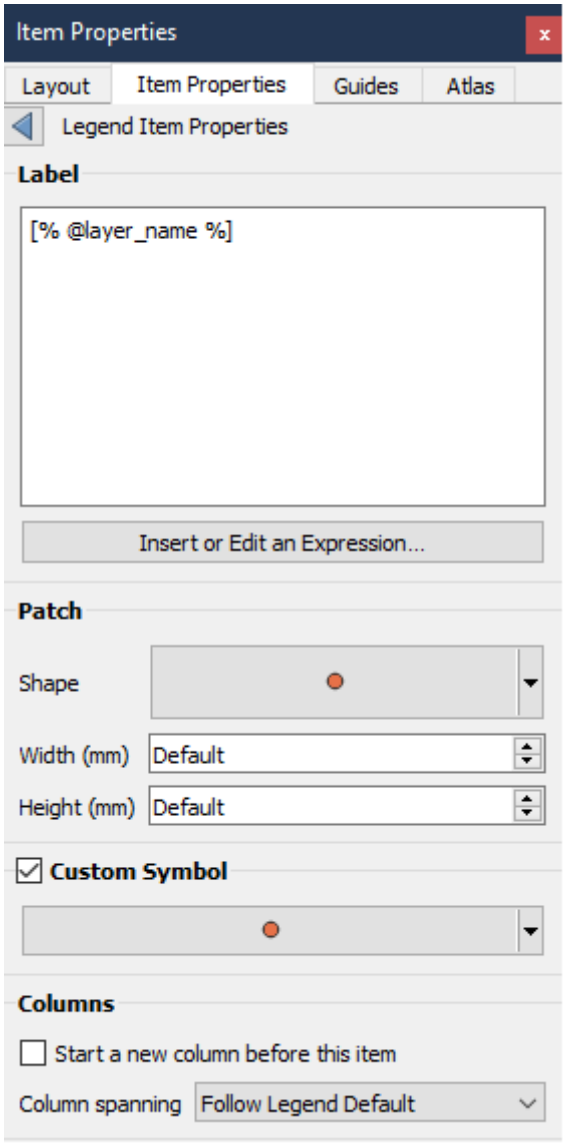
For example, given a `regions` layer categorized by its `type` field, you can append to each class in the legend their number of features and total area, e.g. `Borough (3) - 850ha`:

1. Select the layer entry in the legend tree
2. Press the  button, opening the *Expression String Builder* dialog
3. Enter the following expression (*assuming symbol labels have not been edited*):

```
format( '%1 (%2) - %3ha',
        @symbol_label,
        @symbol_count,
        round( aggregate(@layer, 'sum', $area, filter:= "type"=@symbol_label)/
↪10000 )
      )
```

4. Press *OK*

Customizing legend items



Legend items can also be customized individually in the *Legend Items Properties*. But these customization can only be done with *Auto update* disabled.

Double-clicking on an item or pressing Edit selected item properties allows for futher customization.

Label

For all item types it allows to modify the label text by typing in or by inserting expressions using the *Insert or Edit an Expression*. Expressions can also be added directly anywhere in the item's label by using the [% expression %] notation.

Columns

The Legend Item Property also allows you to control the column splitting behaviour by forcing the column split to occur after a specific item or all symbols of a layer. Automatic splitting of a layer and its child can also be allowed or blocked on a layer-basis in this widget.

Patch

For items with a symbol, the Legend Item Property allows you to specify the maximum height and width that a symbol can occupy.

For vector symbols, you can specify a custom shape for the symbol. The shapes are usually defined by an expression to represent the geometry in a simple plane, but those symbols can also be saved in the style manager and imported later. The default symbol for each geometry type can also be controlled via the style manager.

Custom Symbol

A custom symbol can also be specified for vector symbols. This can be useful to tweak the render of a specific symbol, to enhance it in the legend or have a symbol independent from its true symbol preview. This custom symbol will override the legend symbol, but will take into account the symbol *Patch* specified.

Fonts and text formatting

The *Fonts and text formatting* group of the legend *Item Properties* panel provides the following functionalities:

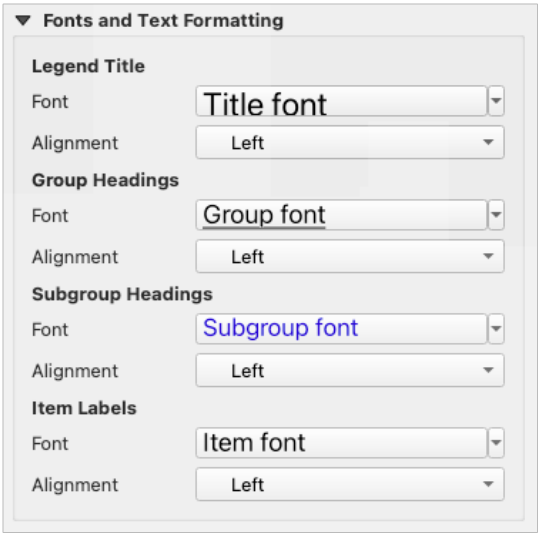


Fig. 18.31: Legend Fonts properties

- You can change the font of the legend title, group, subgroup and item (feature) in the legend item using the *font selector* widget, providing all the capabilities of *text formatting* (font spacing control, mixed HTML formatting, coloring, blending, background, text buffer, shadow, …)
- For each of these levels you can set the text *Alignment*: it can be *Left* (default for left-to-right based locales), *Center* or *Right* (default for right-to-left based locales).

Columns

Under the *Columns* group of the legend *Item Properties* panel, legend items can be arranged over several columns:

- Set the number of columns in the *Count* 1,00 field. This value can be made dynamic e.g., following atlas features, legend contents, the frame size…
- *Equal column widths* sets how legend columns should be adjusted.
- The *Split layers* option allows a categorized or a graduated layer legend to be divided between columns.

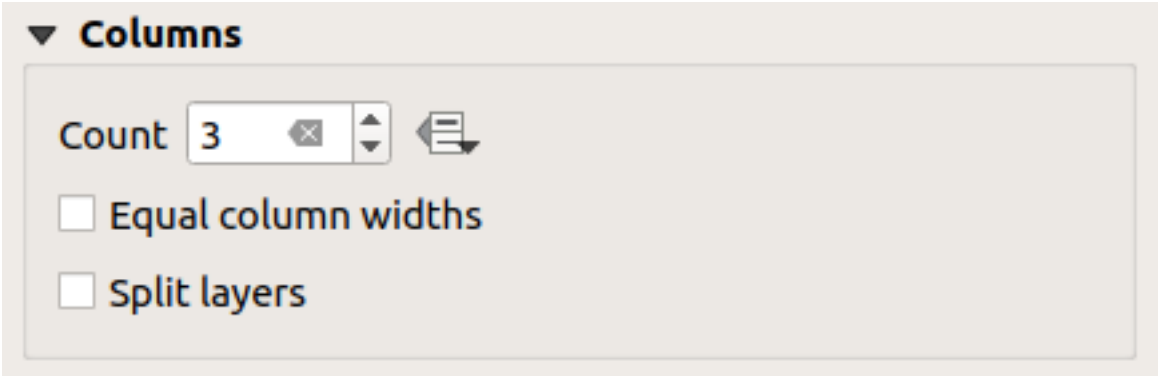


Fig. 18.32: Legend Columns settings

Symbol

The *Symbol* group of the legend *Item Properties* panel configures the size of symbols displayed next to the legend labels. You can:

- Set the *Symbol width* and *Symbol height*
- Set the markers' *Min symbol size* and *Max symbol size*: `0.00mm` means there is no value set.
- *Draw stroke for raster symbols*: this adds an outline to the symbol representing the band color of the raster layer; you can set both the *Stroke color* and *Tickness*.

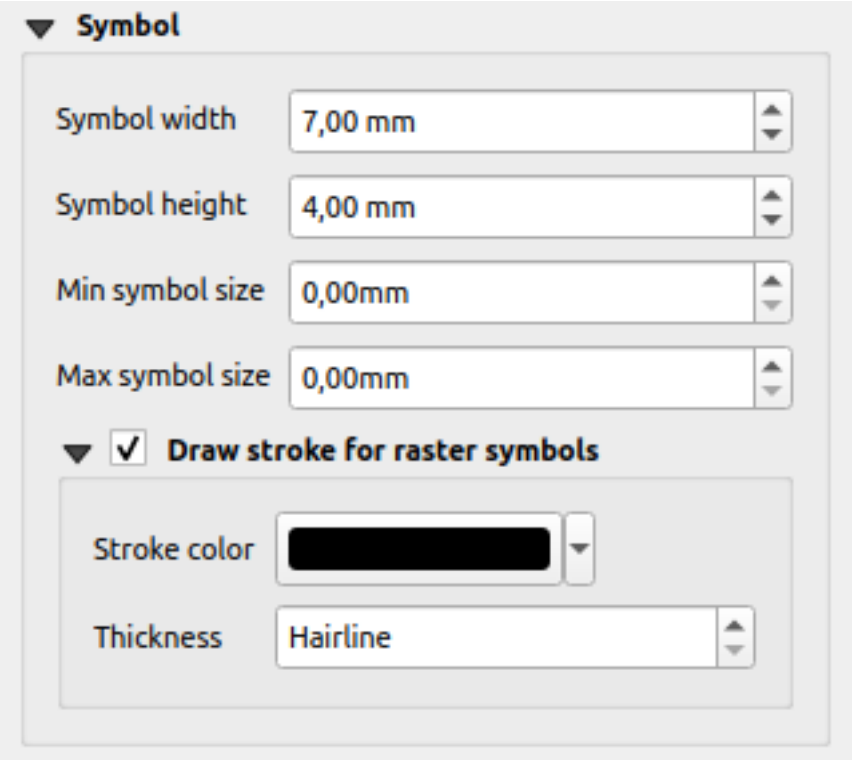


Fig. 18.33: Legend Symbol configuration

WMS LegendGraphic

The *WMS LegendGraphic* section of the legend *Item Properties* panel provide the following functionalities (see Fig. 18.34):

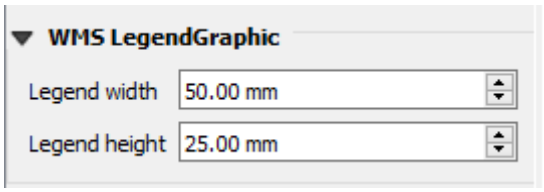
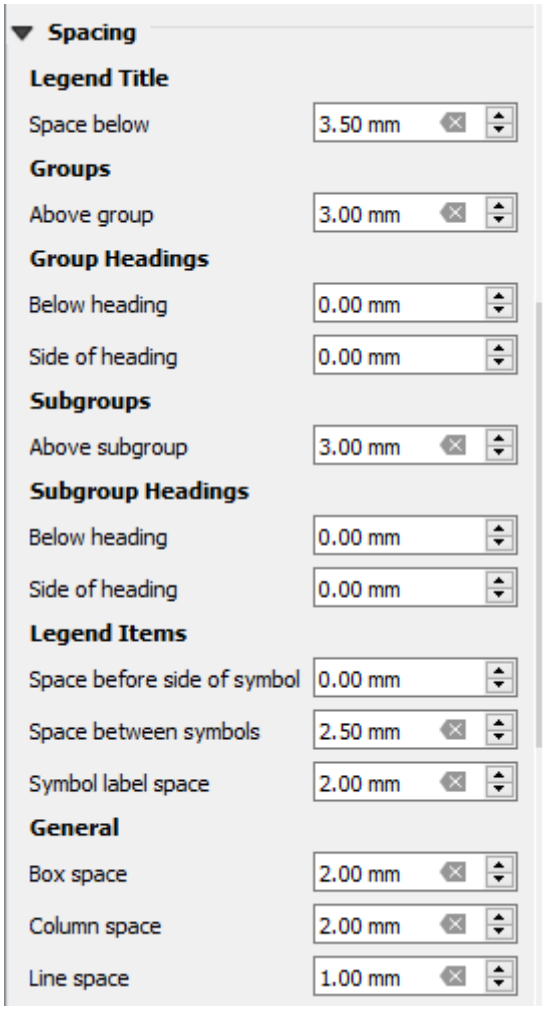


Fig. 18.34: WMS LegendGraphic

When you have added a WMS layer and you insert a legend item, a request will be sent to the WMS server to provide a WMS legend. This Legend will only be shown if the WMS server provides the GetLegendGraphic capability. The WMS legend content will be provided as a raster image.

WMS LegendGraphic is used to be able to adjust the *Legend width* and the *Legend height* of the WMS legend raster image.

Spacing



The *Spacing* section allows you to customize the spacing within the legend. Spacing can greatly help denote the groupement of items in the legend and their relation.

Spacing around and before title, groups, subgroups, symbols, labels, boxes, columns and lines can be customized through this dialog.

18.2.6 The Scale Bar Item

Scale bars provide a visual indication of the size of features, and distance between features, on the map item. A scale bar item requires a map item. Use the *Add Scale Bar* tool following *items creation instructions* to add a new scale bar item that you can later manipulate the same way as exposed in *Interacting with layout items*.

By default, a new scale bar item shows the scale of the map item over which it is drawn. If there is no map item below, the *reference map* is used. You can customize it in the *Item Properties* panel. Other than the *items common properties*, this feature has the following functionalities (see Fig. 18.35):

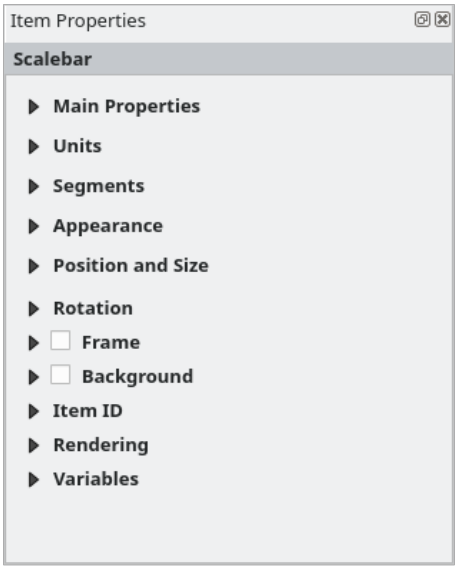


Fig. 18.35: Scale Bar Item Properties Panel

Main properties

The *Main properties* group of the scale bar *Item Properties* panel provides the following functionalities (see Fig. 18.36):



Fig. 18.36: Scale Bar Main properties group

1. First, choose the map the scale bar will be attached to
2. Then, choose the style of the scale bar. Available styles are:
   - **Single box** and **Double box** styles, which contain one or two lines of boxes alternating colors;
   - **Middle**, **Up** or **Down** line ticks;

- **Stepped line** style that draws a stepped line representation of a scalebar
- **Hollow** style that draws a single box with alternating color for the segments, with horizontal lines through alternating segments
- **Numeric**, where the scale ratio is printed (e.g., `1:50000`).

3. Set properties as appropriate

Units

The *Units* group of the scale bar *Item Properties* panel provides the functionalities to set the units of display and some text formatting (see Fig. 18.37):

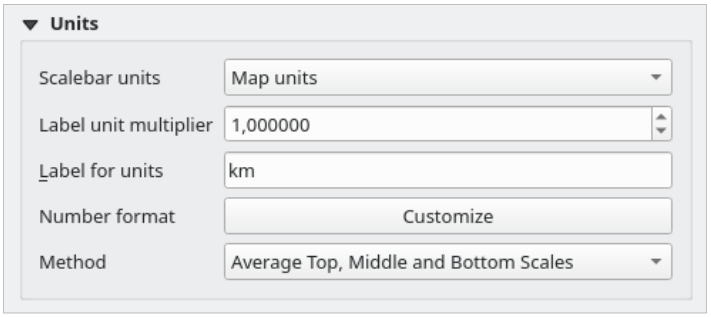


Fig. 18.37: Scale Bar Units group

- Select the units you want to use with *Scalebar units*. There are many possible choices: **Map Units** (the default one), **Meters**, **Feet**, **Miles** or **Nautical Miles**… and some derivatives. Units conversion is handled automatically.
- The *Label unit multiplier* specifies how many scale bar units per labeled unit. Eg, if your scale bar units are set to "meters", a multiplier of 1000 will result in the scale bar labels in "kilometers".
- The *Label for units* field defines the text used to describe the units of the scale bar, eg `m` or `km`. This should be matched to reflect the multiplier above.
- Press *Customize* next to *Number format* to have control over all the formatting properties for the numbers in the scale bar, including thousand separators, decimal places, scientific notation, etc. (see *Number Formatting* for more details). Very useful in the case of making maps for audiences outside of the current QGIS locale, or when you would like to vary the style from the locale defaults (e.g. adding thousands separators when the locale default is to hide them).
- Select the *Method* for scale bar calculation: Depending on the map CRS and extent, the distance reported in the scale bar may not be the same measured over the map. This option, which defaults to the *project's scale calculation method*, helps you specify the one you would like to use for this particular scale bar.

Segments

The *Segments* group of the scale bar *Item Properties* panel provides the functionalities to configure the number and size of segments and subdivisions (see Fig. 18.38):

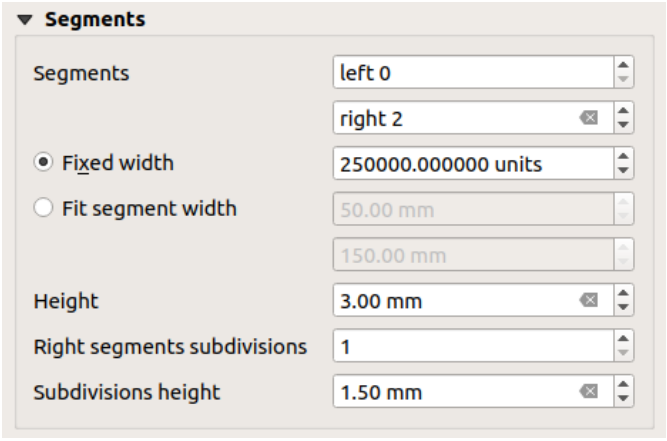


Fig. 18.38: Scale Bar Segments group

- You can define the number of *Segments* that will be drawn at the left and right sides of the `0` of the scale bar:
  - number of subdivisions of a unique segment on the *Left* side
  - number of segments on the *Right* side
- You can set the width of a segment or a range for the total length of the scale bar:
  - Set how long a segment will be in your scale bar units (*Fixed width*)
  - Or limit the total scale bar size in `mm` with *Fit segment width* option. In the latter case, each time the map scale changes, the scale bar is resized (and its label updated) to fit between the upper and lower range set.
- *Height* is used to define the height of the bar.
- *Right segment subdivisions* is used to define the number of sections the right-side segments of the scale bar can have (for *Line Ticks Down*, *Line Ticks Middle* and *Line Ticks Up* scale bar styles) .
- *Subdivision height* is used to define the height of the subdivision segment.

Appearance

The *Appearance* group of the scale bar *Item Properties* panel provides the following functionalities:

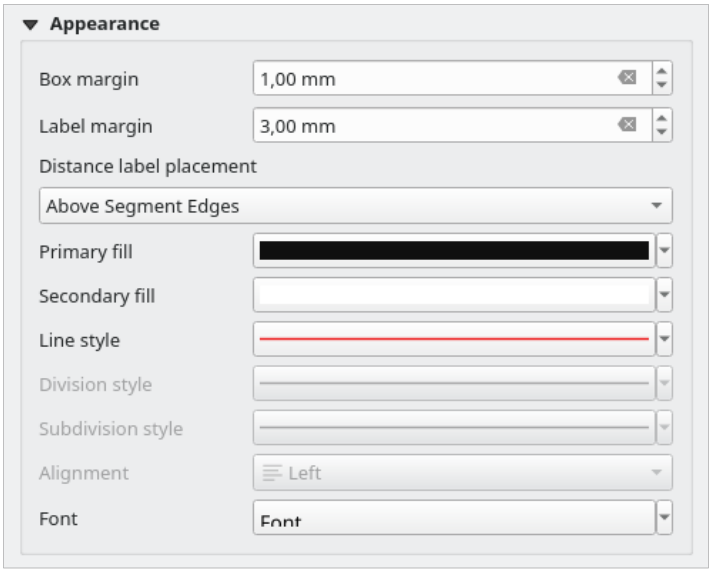


Fig. 18.39: Scale Bar Appearance group

You can define how the scale bar will be displayed in its frame.

- *Box margin* : space between text and frame borders
- *Label margin* : space between text and scale bar drawing
- *Distance label placement* defines the position of the text relative to the scale bar segments: *Above Segment Edges*, *Above Segment Centers*, *Below Segment Edges* or *Below Segment Centers*
- *Primary fill* and *Secondary fill* of the scale bar drawing using *fill symbols properties* (color, opacity, patterns, effects…) — for *Single Box*, *Double Box* and *Hollow* styles
- *Line style* of the scale bar drawing using *line symbols properties* (color, stroke, join, cap style, patterns, effects…) — for all but *Numeric* style
- *Division style* and *Subdivision style* respectively for division and subdivision segments in *Line Ticks Up*, *Line Ticks Middle* and *Line Ticks Down* scale bar styles using *line symbols properties* (color, stroke, join, cap style, patterns, effects…)
- *Alignment* puts text on the left, center or right side of the frame (only for *Numeric* scale bar style)
- *Font* to set the *properties* (size, font, color, letter spacing, shadow, background…) of the scale bar label.

Since most of the display properties of the scale bar rely on symbols whose properties can be data-defined, it's possible to render data-defined scale bars.

Example: The following code applied to the bold property of the scale labels will display numbers in bold when they are a multiple of 500:

```
-- returns True (or 1) if the value displayed on the bar
-- is a multiple of 500

@scale_value % 500 = 0
```

18.2.7 The Table Items

You can use table items to decorate and explain your map:

- *Attribute table*: automatically extracts a subset of the attributes of a layer, based on predefined rules
- *Fixed table*: allows for creation of tables with contents manually entered (i.e. spreadsheet style), so that you can create completely custom tables.

The attribute table item

Any layer in the project can have its attributes shown in the print layout. Use the  *Add Attribute Table* tool following *items creation instructions* to add a new table item that you can later manipulate the same way as exposed in *Interacting with layout items*.

By default, a new attribute table item loads first rows of the first (alphabetically sorted) layer, with all the fields. You can however customize the table thanks to its *Item Properties* panel. Other than the *items common properties*, this feature has the following functionalities (see Fig. 18.40):

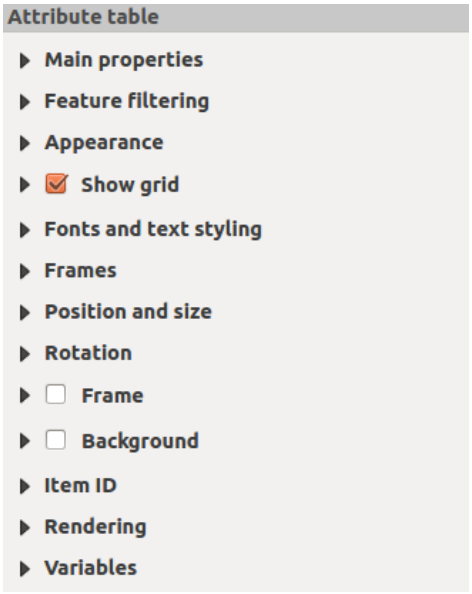


Fig. 18.40: Attribute table Item Properties Panel

Main properties

The *Main properties* group of the attribute table provides the following functionalities (see Fig. 18.41):

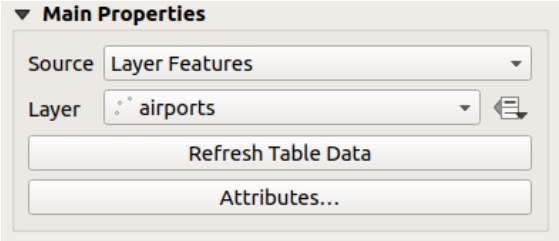


Fig. 18.41: Attribute table Main properties Group

- For *Source* you can by default only select **Layer features** allowing you to select a *Layer* from the vector layers loaded in the project.

  The Data-defined override button near the layer list allows you to dynamically change the layer which is used to populate the table, e.g. you could fill the attribute table with different layer attributes per atlas page. Note that the table structure used (Fig. 18.44) is the one of the layer shown in the *Layer* drop-down list and it is left intact, meaning that setting a data defined table to a layer with different field(s) will result in empty column(s) in the table.

  In case you activate the *Generate an atlas* option in the *Atlas* panel (see *Generate an Atlas*), there are two additional *Source* possible:

  - **Current atlas feature** (see Fig. 18.42): you won't see any option to choose the layer, and the table item will only show a row with the attributes from the current feature of the atlas coverage layer.
  - and **Relation children** (see Fig. 18.43): an option with the relation names will show up. This feature can only be used if you have defined a *relation* using your atlas coverage layer as parent, and the table will show the children rows of the atlas coverage layer's current feature.

- The button *Refresh Table Data* can be used to refresh the table when the actual contents of the table has changed.

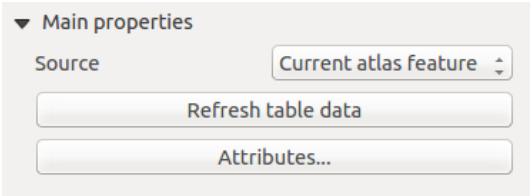


Fig. 18.42: Attribute table Main properties for 'Current atlas feature'

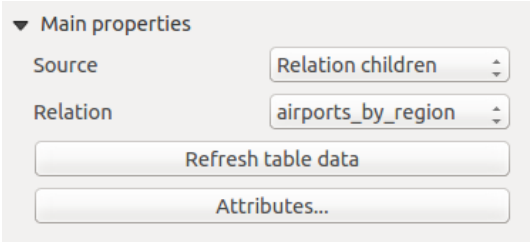


Fig. 18.43: Attribute table Main properties for 'Relation children'

- The button *Attributes…* starts the *Select Attributes* dialog, (see Fig. 18.44) that can be used to change the visible contents of the table. The upper part of the window shows the list of the attributes to display and the lower part helps you sort the data.

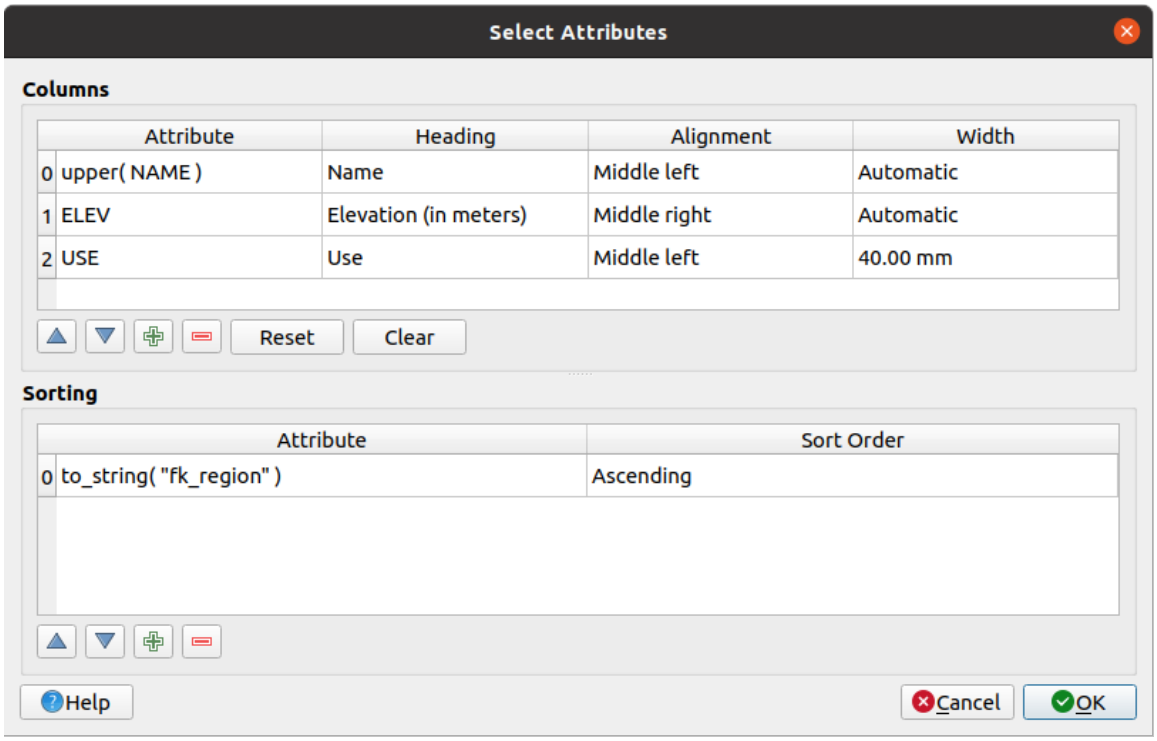


Fig. 18.44: Attribute table Select attributes Dialog

In the *Columns* section you can:

- Move attributes up or down the list by selecting the rows and then using the and buttons to shift the rows. Multiple rows can be selected and moved at any one time.
- Add an attribute with the button. This will add an empty row at the bottom of the table where you can select a field to be the attribute value or create an attribute via a regular expression.

- Remove an attribute with the button. Multiple rows can be selected and removed at any one time.
- Reset the attribute table back to its default state with the *Reset* button.
- Clear the table using the *Clear* button. This is useful when you have a large table but only want to show a small number of attributes. Instead of manually removing each row, it may be quicker to clear the table and add the rows needed.
- Cell headings can be altered by adding the custom text in the *Heading* column.
- Cell alignment can be managed with the *Alignment* column which will dictate the texts position within the table cell.
- Cell width can be manually managed by adding custom values to the *width* column.

In the *Sorting* section you can:

- Add an attribute to sort the table with: press the button and a new empty row is added. Insert a field or an expression in the *Attribute* column and set the *Sort order* to **Ascending** or **Descending**.
- Select a row in the list and use the and buttons to change the sort priority on attribute level. Selecting a cell in the *Sort Order* column helps you change the sorting order of the attribute field.
- Use the button to remove an attribute from the sorting list.

Feature filtering

The *Feature filtering* group of the attribute table provides the following functionalities (see Fig. 18.45):

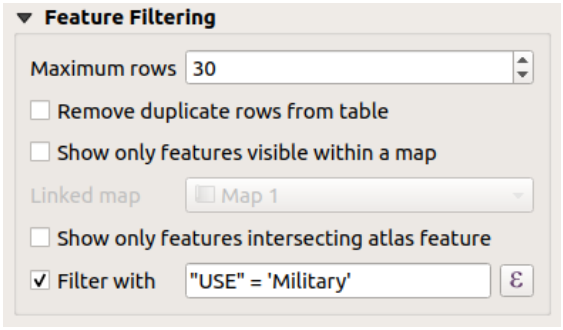


Fig. 18.45: Attribute table Feature filtering Group

You can:

- Define the *Maximum rows* to be displayed.
- Activate *Remove duplicate rows from table* to show unique records only.
- Activate *Show only visible features within a map* and select the corresponding *Linked map* whose visible features attributes will be displayed.
- Activate *Show only features intersecting Atlas feature* is only available when *Generate an atlas* is activated. When activated it will show a table with only the features which intersect the current atlas feature.
- Activate *Filter with* and provide a filter by typing in the input line or insert a regular expression using the given expression button. A few examples of filtering statements you can use when you have loaded the airports layer from the Sample dataset:
  - `ELEV > 500`

- `NAME = 'ANIAK'`
- `NAME NOT LIKE 'AN%'`
- `regexp_match( attribute( $currentfeature, 'USE' ) , '[i]')`

The last regular expression will include only the airports that have a letter 'i' in the attribute field 'USE'.

Appearance

The *Appearance* group of the attribute table provides the following functionalities (see Fig. 18.46):

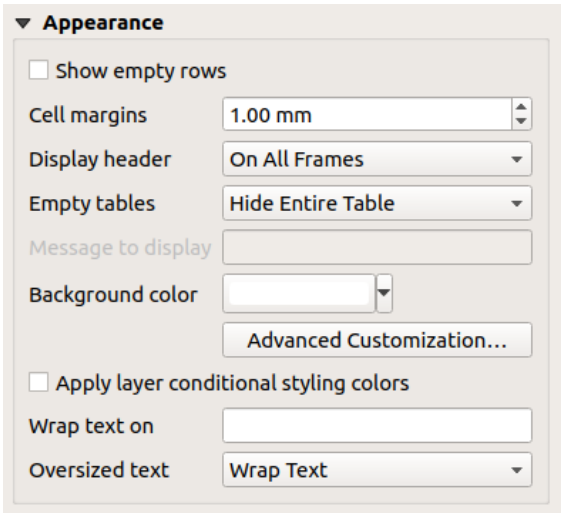


Fig. 18.46: Attribute table appearance Group

- Click *Show empty rows* to fill the attribute table with empty cells. This option can also be used to provide additional empty cells when you have a result to show!
- With *Cell margins* you can define the margin around text in each cell of the table.
- With *Display header* you can select from a list one of 'On first frame', 'On all frames' default option, or 'No header'.
- The option *Empty table* controls what will be displayed when the result selection is empty.
  - **Draw headers only**, will only draw the header except if you have chosen 'No header' for *Display header*.
  - **Hide entire table**, will only draw the background of the table. You can activate *Don't draw background if frame is empty* in *Frames* to completely hide the table.
  - **Show set message**, will draw the header and adds a cell spanning all columns and display a message like 'No result' that can be provided in the option *Message to display*
- The option *Message to display* is only activated when you have selected **Show set message** for *Empty table*. The message provided will be shown in the table in the first row, when the result is an empty table.
- With *Background color* you can set the background color of the table using the *color selector* widget. The *Advanced customization* option helps you define different background colors for each cell (see Fig. 18.47)

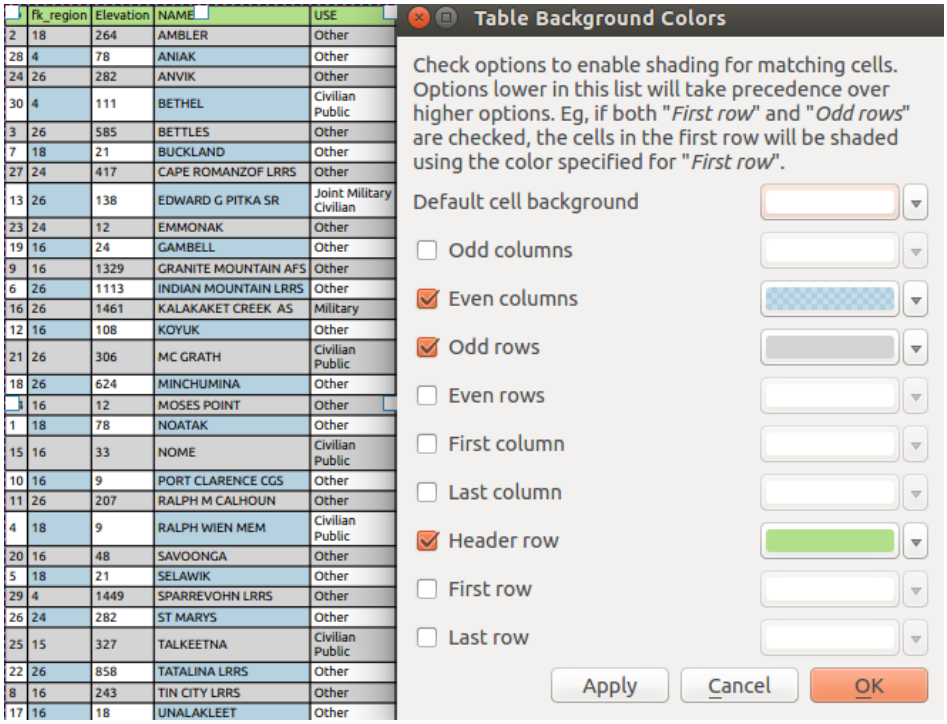


Fig. 18.47: Attribute table Advanced Background Dialog

- *Apply layer conditional styling colors*: the *conditional table formatting* present in the layer is applied inside the layout attribute table (background color, font family and properties such as bold, italic, strikeout, underline, color, …). Conditional formatting rules take precedence over other layout table formatting settings, e.g. they will override other cell background color settings such as alternating row colors.
- With the *Wrap text on* option, you can define a character on which the cell content will be wraped each time it is met
- With *Oversized text* you define the behavior when the width set for a column is smaller than its content's length. It can be **Wrap text** or **Truncate text**.

> **Note**
>
> More properties of the attribute table item are described in the *Tables common functionalities* section.

The fixed table item

Additional information about the map can be inserted manually into a table by choosing *Add Fixed Table* and by following *items creation instructions* to add a new table item that you can later manipulate the same way as exposed in *Interacting with layout items*.

By default, an empty table with two minimized columns and rows appears in the map layout. You have to customize the table in the *Item Properties* panel. Other than the *items common properties*, this feature has the following functionalities:

Main properties

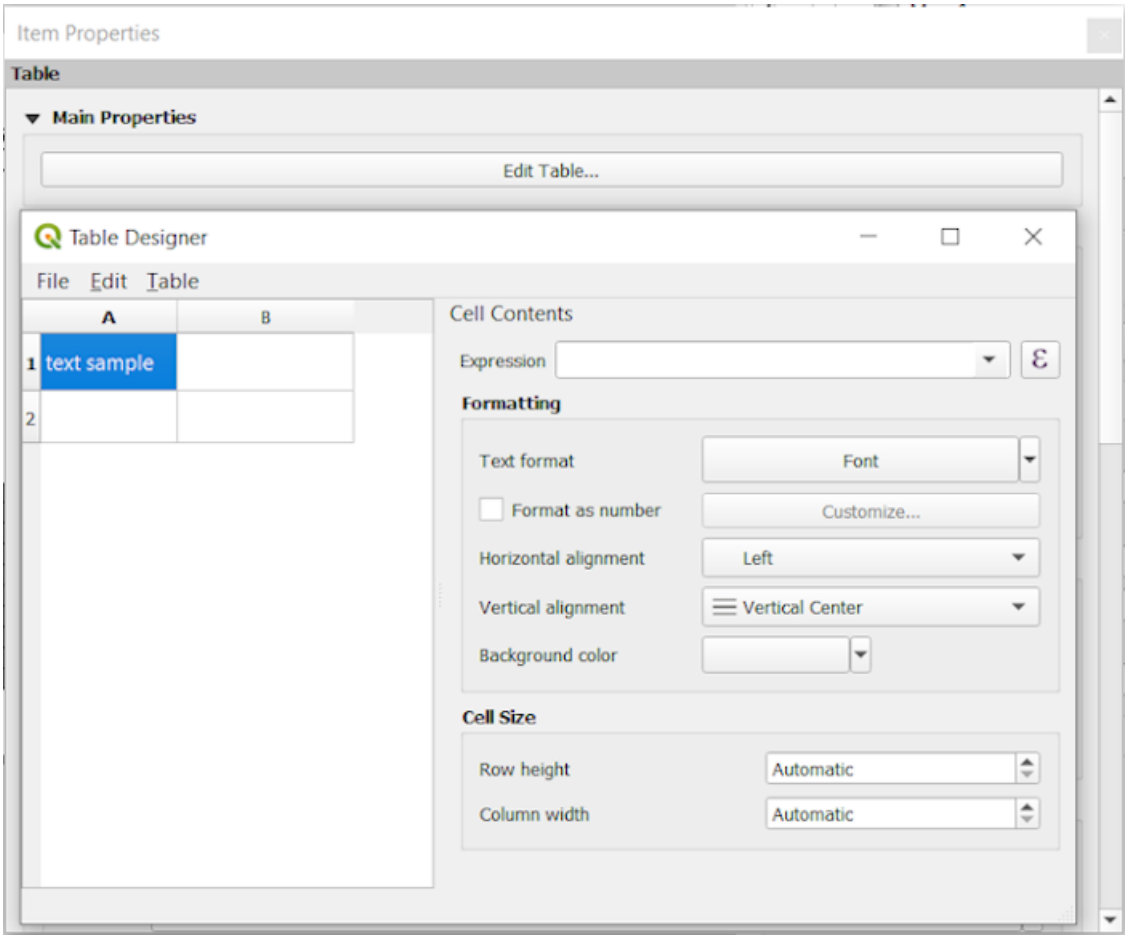


Fig. 18.48: Fixed table Item Properties Panel with Table designer

In *Main properties*, press *Edit table…* to open the *Table designer* dialog where you can build your table (double-clicking the table item will also open the editor dialog):

- On the left hand of the dialog, a table frame displays the active table item with its content and settings. Click or double-click on a cell and write a text to modify its contents. Press `Ctrl+Enter` to add linebreak (for a multiline text). The logic for selecting multiple cells in the table is the same as in the vector layer *attribute table*. Grab and move a row or column separator to resize the row above or the column before. A double-click on the separator will resize the row above or the column before to fit their contents. A right-click over column or row headers shows a contextual menu to insert columns or rows around the selection or to delete them.
- Through the menus on top of the dialog, it is possible to:
  - *File* ► *Import Content From Clipboard*: it overrides contents of the table.
  - *File* ► *Close editor*: it closes the *Table Designer* dialog.
  - From the *Edit* menu, you can work with selection functionalities for rows and columns:
    - *Select all* cells in the table
    - *Select columns* or *Select Rows*: when cells are selected in the table, you can extend the selection respectively to their column(s) or row(s)
    - *Clear Cells*: deletes the content of the selected cells
  - The *Table* menu is the place where you design the structure of the table. You can:

    - *Insert rows* ► above or below the selection
    - *Insert columns* ► before or after the selection
    - *Delete Rows* or *Delete Columns* of selected cells
    - *Merge selected cells*: select multiple cells in a rectangular shape and you can merge them and concatenate their contents into the top left cell (whose styling is also applied to the merged cell, unless there is a last row/column cell in the selection).
    - *Split selected cells*: select a merged cell and you can split it back to its individual cells. The current text is kept in the top left cell, and the other cells are filled with their contents before they get merged. They are also reapplied their original styling.
    - *Include Header Row* whose styling can only be controlled from the *Fonts and text styling* widget
- Select the cells and, on the right of the table frame, you can provide the *Cell Contents*:
  - Either manually enter the contents of each cell, or use an *Expression* to automatically populate it.
  - Under the *Formatting* group, set specific options that apply to the selection and take precedence over the *global table content styling*:
    - by choosing the *Text format*
    - by *Format as number* (several formats are available)
    - by defining the *Horizontal alignment* and the *Vertical alignment*
    - by choosing a *Background color*
  - Define the *Cell Size* with *Row height* and *Column width*.

Appearance

The *Appearance* group of the fixed table provides the following functionalities:

- Click *Show empty rows* to fill the attribute table with empty cells.
- With *Cell margins* you can define the margin around text in each cell of the table.
- With *Display header* you can select from a list one of 'On first frame', 'On all frames' default option, or 'No header'.
- With *Background color* you can set the background color of the table using the *color selector* widget. The *Advanced customization* option helps you define different background colors for each cell.
- With *Oversized text* you define the behavior when the width set for a column is smaller than its content's length. It can be **Wrap text** or **Truncate text**.

> **Note**
>
> More properties of the fixed table item are described in the *Tables common functionalities* section.

Tables common functionalities

Show grid

The *Show grid* group of the table items provides the following functionalities (see Fig. 18.49):

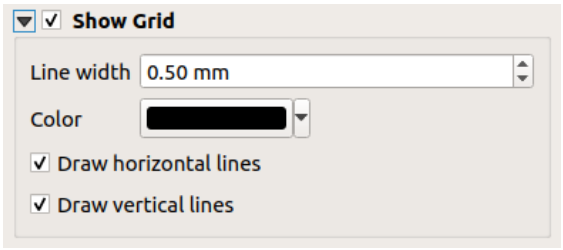


Fig. 18.49: Attribute table Show grid Group

- Activate *Show grid* when you want to display the grid, the outlines of the table cells. You can also select to either *Draw horizontal lines* or *Draw vertical lines* or both.
- With *Line width* you can set the thickness of the lines used in the grid.
- The *Color* of the grid can be set using the color selection widget.

Fonts and text styling

The *Fonts and text styling* group of the table items provides the following functionalities (see Fig. 18.50):



Fig. 18.50: Attribute table Fonts and text styling Group

- You can define *Font* properties for *Table heading* and *Table contents*, using the advanced *text settings* widget (with buffer, shadow, paint effects, transparence, background, coloring, …). Note that these changes do not affect the cells that have custom font assigned, either from the *Appearance* section or the *Table Designer* dialog. Only cells with the default rendering are overwritten.
- For *Table heading* you can additionally set the *Alignment* to `Follow column alignment` or override this setting by choosing `Left`, `Center` or `Right`. The column alignment is set using the *Select Attributes* dialog (see Fig. 18.44).

Frames

The *Frames* group of the table item properties provides the following functionalities (see Fig. 18.51):

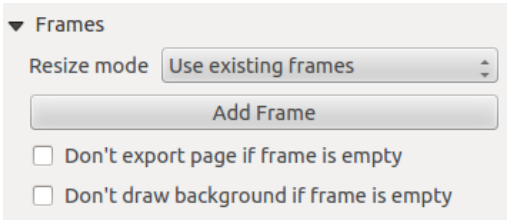


Fig. 18.51: Attribute table Frames Group

- With *Resize mode* you can select how to render the attribute table contents:
  - `Use existing frames` displays the result in the first frame and added frames only.
  - `Extend to next page` will create as many frames (and corresponding pages) as necessary to display the full selection of attribute table. Each frame can be moved around on the layout. If you resize a frame, the resulting table will be divided up between the other frames. The last frame will be trimmed to fit the table.
  - `Repeat until finished` will also create as many frames as the *Extend to next page* option, except all frames will have the same size.
- Use the *Add Frame* button to add another frame with the same size as selected frame. The result of the table that will not fit in the first frame will continue in the next frame when you use the Resize mode `Use existing frames`.
- Activate *Don't export page if frame is empty* prevents the page to be exported when the table frame has no contents. This means all other layout items, maps, scalebars, legends etc. will not be visible in the result.
- Activate *Don't draw background if frame is empty* prevents the background to be drawn when the table frame has no contents.

18.2.8 The Marker, Picture and North Arrow Items

- *The Picture Item*
  - *Main properties*
  - *Size and placement*
  - *Image rotation*
- *The North Arrow Item*
- *The Marker Item*

Along with the map or legend items in the print layout, you may want to decorate your realization with images or annotations. QGIS provides different tools to achieve this:

- the *picture item*: decorates the layout with an image raster or SVG file (e.g. logos, pictures, north arrows, …)
- the *north arrow item*: a picture item predefined with a north arrow image
- the *marker item*: decorates the layout with QGIS vector *symbols*. It can be used to place markers over a map item or for creation of advanced custom legends.

The Picture Item

You can add a picture by dragging it from your file manager onto the canvas, pasting it directly into the layout by using `Ctrl+V` or *Edit* ► *Paste* and by using the Add Picture, following *items creation instructions*. Then you can manipulate it, as explained in *Interacting with layout items*.

When using Add Picture, the picture item will be a blank frame that you can customize using its *Item Properties* panel. Other than the *items common properties*, this feature has the following functionalities:

Main properties

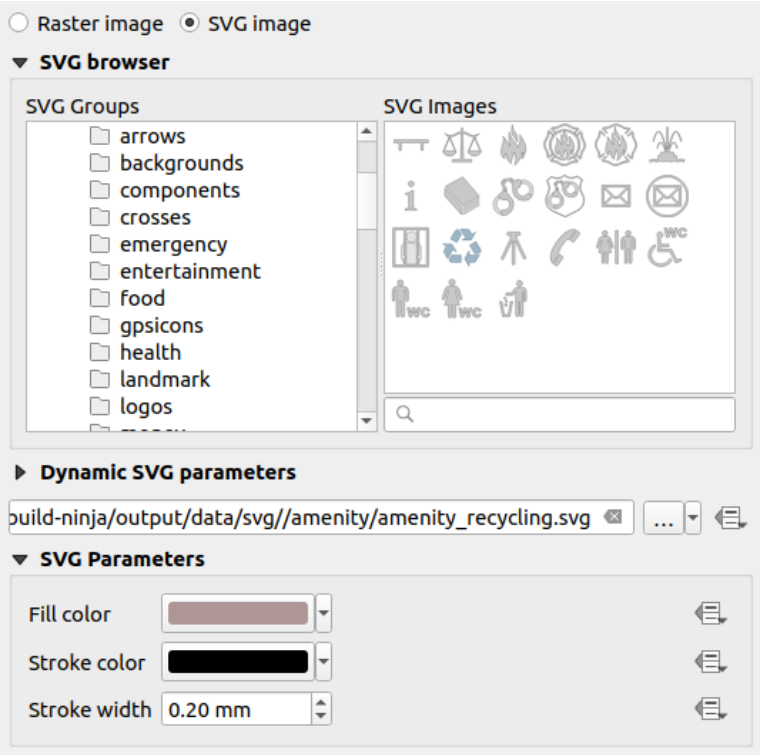


Fig. 18.52: Picture Item Properties panel

The picture item supports two types of images:

- *Raster Image*: a file selector widget can be used to fetch the data. Use the … Browse button to select a file on your computer or enter the path directly in the text field. You can even provide a remote URL that points to a picture. The associated image can also be *embedded* in the layout.

  Use the data defined override button to set the image source from a feature attribute or using a regular expression.

- *SVG Image*: using by default the SVG libraries provided in *Settings* ► *Options* ► *System* ► *SVG Paths*. You can however use any other file, and the file selection follows the same rules as for the raster image. The SVG parameters can as well be set dynamic.

  The QGIS provided (default) `.SVG` files are customizable, meaning that you can easily apply other *Fill color*, *Stroke color* (including opacity) and *Stroke width* than the original, using their corresponding feature in the *SVG Parameters* group. These properties can also be *data-defined*.

  If you add an `.SVG` file that does not enable these properties, you may need to add the following tags to the file in order to add support e.g. for transparency:

  – *fill-opacity="param(fill-opacity)"*

- *stroke-opacity="param(outline-opacity)"*

More details at *Parametrizable SVG*.

Note

Drag-and-drop an image file (raster or SVG) into the layout page will create a layout picture item with corresponding settings.

Size and placement



Fig. 18.53: Layout pictures size and placement properties

With the *Resize mode* option, you can set how the image is displayed when the frame is resized:

- `Zoom`: enlarges/reduces the image to the frame while maintaining the aspect ratio of picture
- `Stretch`: stretches the image to fit inside the frame
- `Clip`: use this mode for raster images only, it sets the size of the image to the original image size without scaling, and the frame is used to clip the image. So only the part of the image that is inside the frame will be visible.
- `Zoom and resize frame`: enlarges the image to fit the frame, and then resizes frame to fit the resulting image dimensions
- `Resize frame to image size`: sets the size of the frame to match the original size of the image (no scaling)

Depending on the selected *Resize mode*, the *Placement* and *Image rotation* options may be disabled. *Placement* lets you select the position of the image inside its frame (top/middle/bottom and left/center/right).

Image rotation

Images can be rotated with the *Image rotation* field. Activating the *Sync with map* checkbox synchronizes the rotation of the image with the rotation applied to the selected map item. This is a convenient feature to make any picture behave as a north arrow. The *North alignment* can be:

- **Grid north**: the direction of a grid line which is parallel to the central meridian of the national/local grid
- **True north**: direction of a meridian of longitude.

You can also apply a declination *Offset* to the picture rotation.

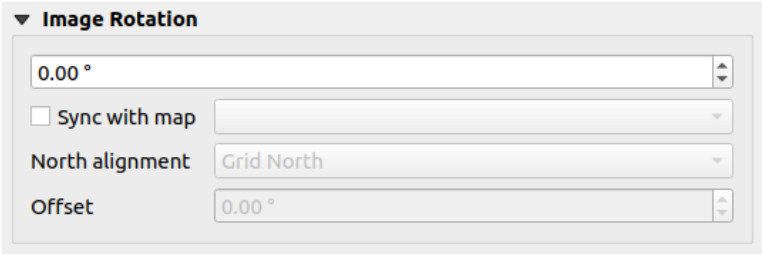


Fig. 18.54: Layout pictures image rotation properties

The North Arrow Item

You can add a north arrow with the Add North Arrow button, following *items creation instructions* and manipulate it the same way as exposed in *Interacting with layout items*.

Since north arrows are images, the *North Arrow* item has the same properties as the *picture item*. The main differences are:

- A default north arrow is used when adding the item, instead of a blank frame
- The north arrow item is synced with a map item by default: the *Sync with map* property is the map over which the north arrow item is drawn. If none, it falls back to the *reference map*.

Note

Many of the north arrows do not have an 'N' added in the north arrow. This is done on purpose, since there are languages that do not use an 'N' for North.

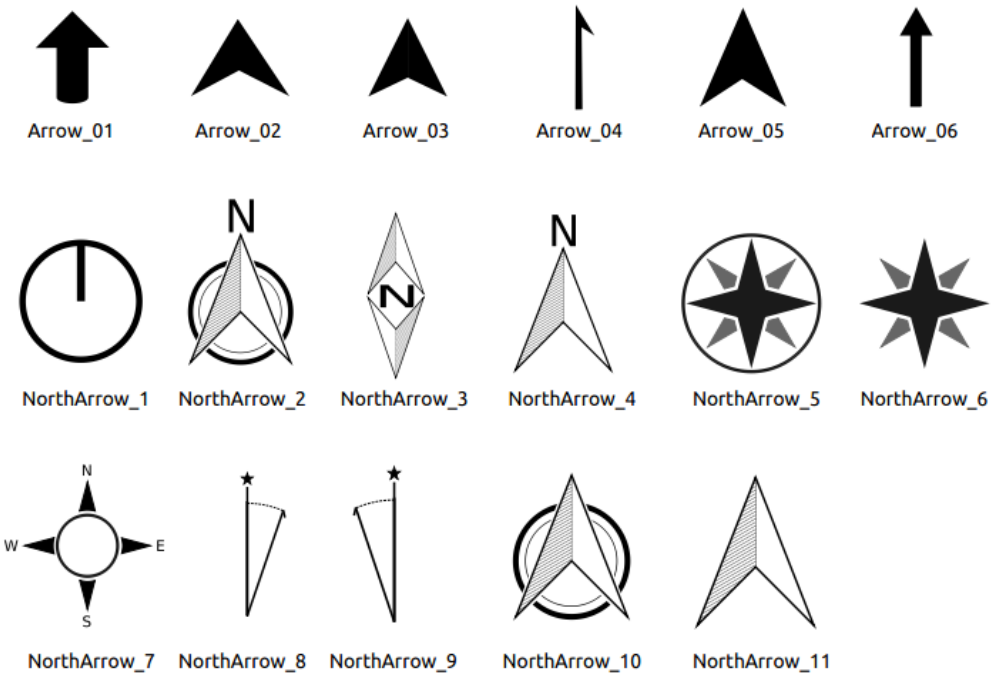


Fig. 18.55: North arrows available for selection in provided SVG library

The Marker Item

To add a marker item, select the Add Marker button, and click on the page. A default point marker symbol is added. Then you can manipulate it, as explained in *Interacting with layout items*. But note that unlike most of the other items, you resize the item given that its size is controlled by the embedded symbols properties.

The marker item can be customized from the *Item Properties* panel. Other than the *items common properties*, you can also:

- modify the *Symbol*, relying on all the symbol *widget capabilities*
- sync the marker item rotation with the map's (see *Image rotation*), acting as a north arrow. The map rotation is added to any existing marker symbol level rotation (so .e.g if you have to rotate the triangle marker 90° to get it pointing straight up, it will still work nicely in north arrow mode!)

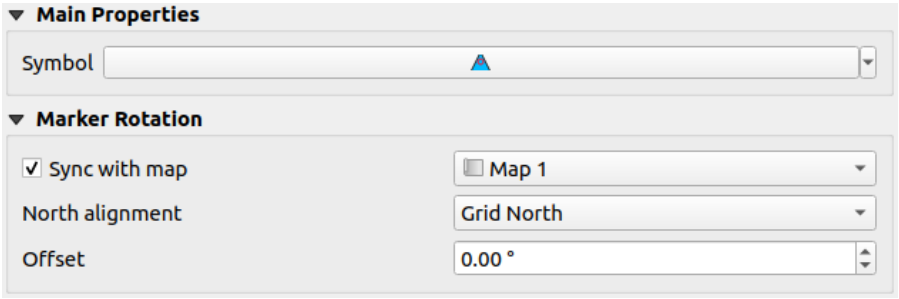


Fig. 18.56: The marker item custom properties

18.2.9 The Elevation Profile Item

The Elevation Profile item is used to display an *elevation profile view* in a layout. Use the *Add Elevation Profile* button, and follow *items creation instructions* to add a new elevation profile item that you can later manipulate the same way as demonstrated in *Interacting with layout items*.

A new elevation profile item has default settings that make it render an empty chart. You can customize its properties in the *Item Properties* panel. In addition to the *common properties*, this feature has the following functionalities:

The Elevation profile *Item Properties* panel embeds a top toolbar with the following functionalities:

- Update elevation profile to refresh the item rendering
- Copy from elevation profile: a drop-down menu to select an elevation profile view from. The view settings are applied to the layout elevation profile item, and can be later modified.

Layers

Under the *Layers* group, check in the tree view the layers you would like to render in the profile item. Remember to properly configure the *Elevation* properties of the selected layers.

Profile curve

- *Controlled by atlas*: the *profile curve* will be taken from the current atlas feature and the elevation profile view updated as you walk through the atlas features. This is currently supported for an active layout atlas or report, using a line geometry type coverage layer.
- The *Tolerance* distance, which can be data-defined, helps you control how far a feature of the visible layers should be from the profile curve in order to display in the layout elevation view. Only point features are currently returned.

Chart ranges

A layout elevation profile item does not necessarily display the full extent of the elevation profile view it is based on. You can limit the area to render, providing:

- on the X axis, the *Minimum distance* and *Maximum distance* from the profile curve starting point
- On the Y axis, the *Minimum elevation* and *Maximum elevation*

Distance and elevation axes

The *Distance axis* and *Elevation axis* groups give options to tweak the grid over the elevation profile item, respectively on the X and Y axes:

- the distance display *Unit*, allowing to override the map canvas unit
- the graduation on the axis with both a *Major interval* and *Minor interval*
- the line symbols to apply to the corresponding *Major grid lines* and *Minor grid lines*
- how regular the graduation items should be labeled (*Label interval*) as well as their *Label format* and *Label font*
- the *Distance labels*: configures whether the distance unit symbol should be placed next to *Every value*, *First value*, *Last value*, *First and last values*, or skipped (*None*)

Chart area

Under *Chart area*, you can configure the rendering of the area in which the elevation profile plot is actually displayed:

- a *Background* fill symbol
- a *Border* line symbol
- the margins from the elevation profile item border

18.2.10 The HTML Frame Item

It is possible to add a frame that displays the contents of a website or even create and style your own HTML page and display it! You can add a picture with the *Add HTML* following *items creation instructions* and manipulate it the same way as exposed in *Interacting with layout items*. Note that the HTML scale is controlled by the layout export resolution at the time the HTML frame is created.

The HTML item can be customized using its *Item Properties* panel. Other than the *items common properties*, this feature has the following functionalities (see Fig. 18.57):

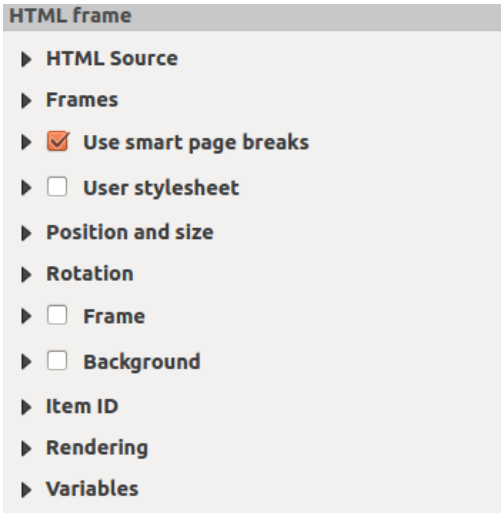


Fig. 18.57: HTML Frame, the Item Properties Panel

HTML Source

The *HTML Source* group of the HTML frame *Item Properties* panel provides the following functionalities (see Fig. 18.58):

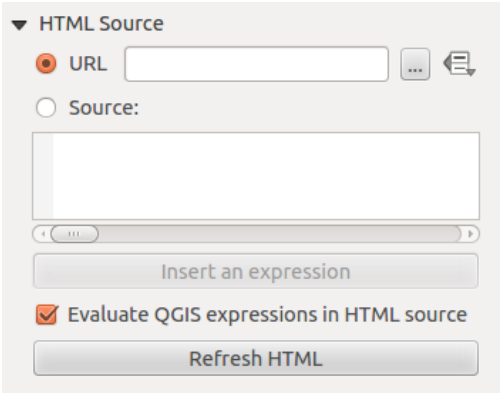


Fig. 18.58: HTML frame, the HTML Source properties

- In *URL* you can enter the URL of a webpage you copied from your Internet browser or select an HTML file using the … Browse button. There is also the option to use the Data-defined override button, to provide a URL from the contents of an attribute field of a table or using a regular expression.
- In *Source* you can enter text in the textbox with some HTML tags or provide a full HTML page.
- The *Insert or Edit an Expression…* button can be used to add an expression like `[%Year($now)%]` in the Source textbox to display the current year. This button is only activated when radiobutton *Source* is selected. After inserting the expression click somewhere in the textbox before refreshing the HTML frame, otherwise you will lose the expression.
- Activate *Evaluate QGIS expressions in HTML code* to see the result of the expression you have included, otherwise you will see the expression instead.
- Use the *Refresh HTML* button to refresh the HTML frame(s) and see the result of changes.

Frames

The *Frames* group of the HTML frame *Item Properties* panel provides the following functionalities (see Fig. 18.59):

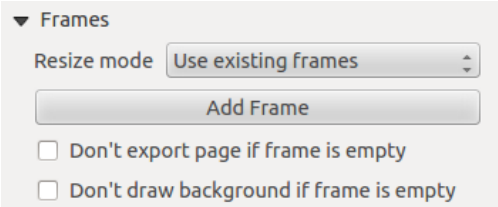


Fig. 18.59: HTML frame, the Frames properties

- With *Resize mode* you can select how to render the HTML contents:
  - `Use existing frames` displays the result in the first frame and added frames only.
  - `Extend to next page` will create as many frames (and corresponding pages) as necessary to render the height of the web page. Each frame can be moved around on the layout. If you resize a frame, the webpage will be divided up between the other frames. The last frame will be trimmed to fit the web page.
  - `Repeat on every page` will repeat the upper left of the web page on every page in frames of the same size.
  - `Repeat until finished` will also create as many frames as the `Extend to next page` option, except all frames will have the same size.
- Use the *Add Frame* button to add another frame with the same size as selected frame. If the HTML page does not fit in the first frame it will continue in the next frame when you use *Resize mode* or *Use existing frames*.
- Activate *Don't export page if frame is empty* prevents the page from being exported when the frame has no HTML contents. This means all other layout items, maps, scale bars, legends etc. will not be visible in the result.
- Activate *Don't draw background if frame is empty* prevents the HTML frame being drawn if the frame is empty.

Use smart page breaks and User style sheet

The *Use smart page breaks* dialog and *User style sheet* dialog of the HTML frame *Item Properties* panel provides the following functionalities (see Fig. 18.60):

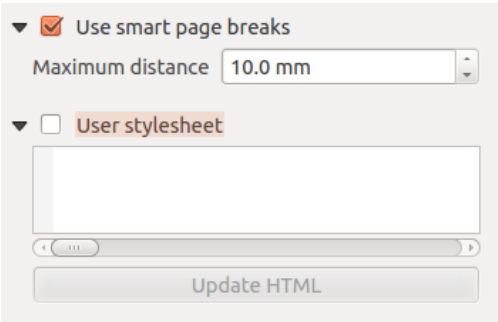


Fig. 18.60: HTML frame, Use smart page breaks and User style sheet properties

- Activate *Use smart page breaks* to prevent the html frame contents from breaking mid-way a line of text so it continues nice and smooth in the next frame.
- Set the *Maximum distance* allowed when calculating where to place page breaks in the html. This distance is the maximum amount of empty space allowed at the bottom of a frame after calculating the optimum break

location. Setting a larger value will result in better choice of page break location, but more wasted space at the bottom of frames. This is only used when *Use smart page breaks* is activated.

- Activate *User style sheet* to apply HTML styles that often is provided in cascading style sheets. An example of style code is provided below to set the color of `<h1>` header tag to green and set the font and font size of text included in paragraph tags `<p>`.

```
h1 {color: #00ff00;
}
p {font-family: "Times New Roman", Times, serif;
   font-size: 20px;
}
```

- Use the *Update HTML* button to see the result of the style sheet settings.

18.2.11 The Shape Items

QGIS provides a couple of tools to draw regular or more complex shapes over the print layout.

> **Note**
>
> Unlike other print layout items, you can not style the frame nor the background color of the shapes bounding frame (set to transparent by default).

The Regular Shape Item

The *Shape* item is a tool that helps to decorate your map with regular shapes like triangle, rectangle, ellipse… You can add a regular shape using the Add Shape tool which gives access to particular tools like Add Rectangle, Add Ellipse and Add Triangle. Once you have selected the appropriate tool, you can draw the item following *items creation instructions*. Like other layout items, a regular shape can be manipulated the same way as exposed in *Interacting with layout items*.

> **Note**
>
> Holding down the `Shift` key while drawing the basic shape with the click and drag method helps you create a perfect square, circle or triangle.

The default shape item can be customized using its *Item Properties* panel. Other than the *items common properties*, this feature has the following functionalities (see Fig. 18.61):

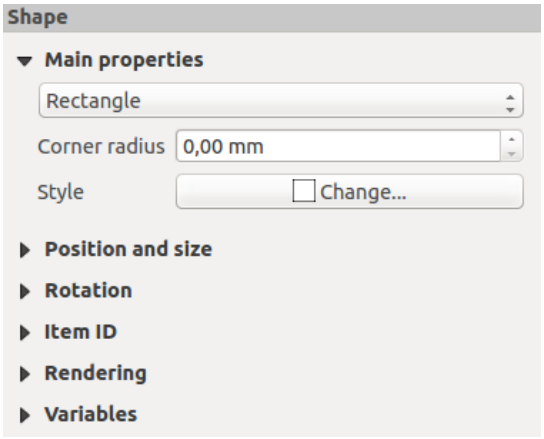


Fig. 18.61: Shape Item Properties Panel

The *Main properties* group shows and allows you to switch the type of the shape item (**Ellipse**, **Rectangle** or **Triangle**) inside the given frame.

You can set the style of the shape using the advanced *symbol* and *color* selector widget…

For the rectangle shape, you can set in different units the value of the *Corner radius* to round of the corners.

The Node-Based Shape Items

While the *Add Shape* tool provides way to create simple and predefined geometric item, the *Add Node Item* tool helps you create a custom and more advanced geometric item. For polylines or polygons, you can draw as many lines or sides as you want and vertices of the items can be independently and directly manipulated using the *Edit Nodes Item*. The item itself can be manipulated as exposed in *Interacting with layout items*.

To add a node-based shape:

1. Click the Add Node Item icon
2. Select either Add Polygon or Add Polyline tool
3. Perform consecutive left clicks to add nodes of your item. If you hold down the `Shift` key while drawing a segment, it is constrained to follow an orientation multiple of 45°.
4. When you're done, right-click to terminate the shape.

You can customize the appearance of the shape in the *Item Properties* panel.

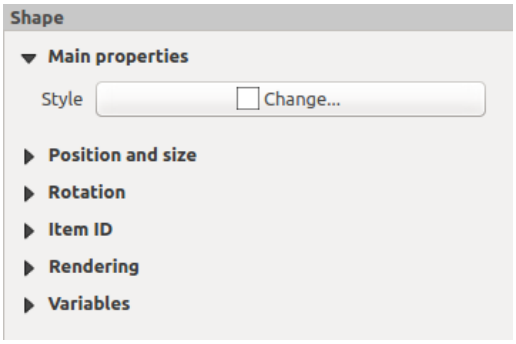


Fig. 18.62: Polygon Node Shape Item Properties Panel

In the *Main properties*, you can set the style of the shape using the advanced *symbol* and *color* selector widget…

For polyline node items, you can also parameterize the *Line markers* i.e. add:

- start and/or end markers with options:
  - *None*: draws a simple polyline.
  - *Arrow*: adds a regular triangular arrow head that you can customize.
  - *SVG* marker: uses an `SVG` file as arrow head of the item.
- customize the arrow head:
  - *Arrow stroke color*: sets the stroke color of the arrow head.
  - *Arrow fill color*: sets the fill color of the arrow head.
  - *Arrow stroke width*: sets the stroke width of the arrow head.
  - *Arrow head width*: sets the size of the arrow head.

SVG images are automatically rotated with the line. Stroke and fill colors of QGIS predefined SVG images can be changed using the corresponding options. Custom SVG may require some tags following this *instruction*.

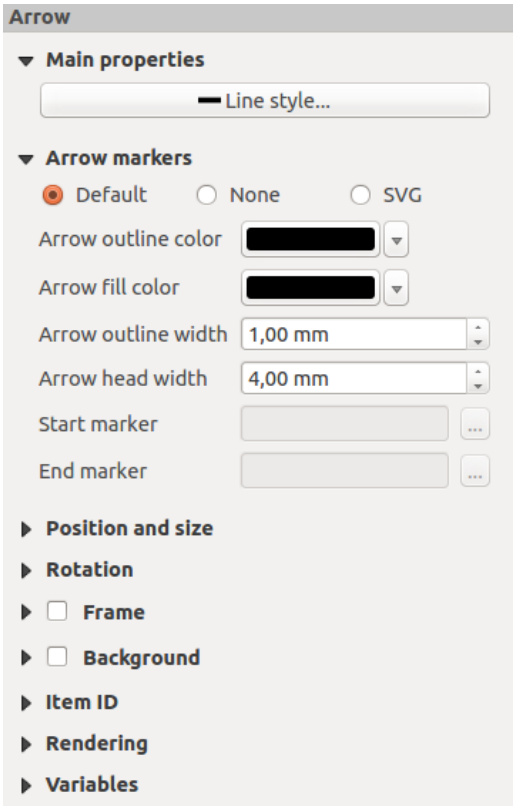


Fig. 18.63: Polyline Node Shape Item Properties Panel

The Arrow Item

The Add Arrow tool is a shortcut to create an arrow-enabled polyline by default and thus has the same properties and behavior as a *polyline node item*.

Actually, the arrow item can be used to add a simple arrow, for example, to show the relation between two different print layout items. However, to create a north arrow, the *image item* should be considered first as it gives access to a set of north arrows in `.SVG` format that you can sync with a map item so that it rotates automatically with it.

Editing a node item geometry

A specific tool is provided to edit node-based shapes through Edit Nodes Item. Within this mode, you can select a node by clicking on it (a marker is displayed on the selected node). A selected node can be moved either by dragging it or by using the arrow keys. Moreover, in this mode, you are able to add nodes to an existing shape: double-click on a segment and a node is added at the place you click. Finally, you can remove the currently selected node by hitting the `Del` key.

18.3 Creating an Output

Fig. 18.64 shows an example print layout including all the types of layout items described in the previous section.

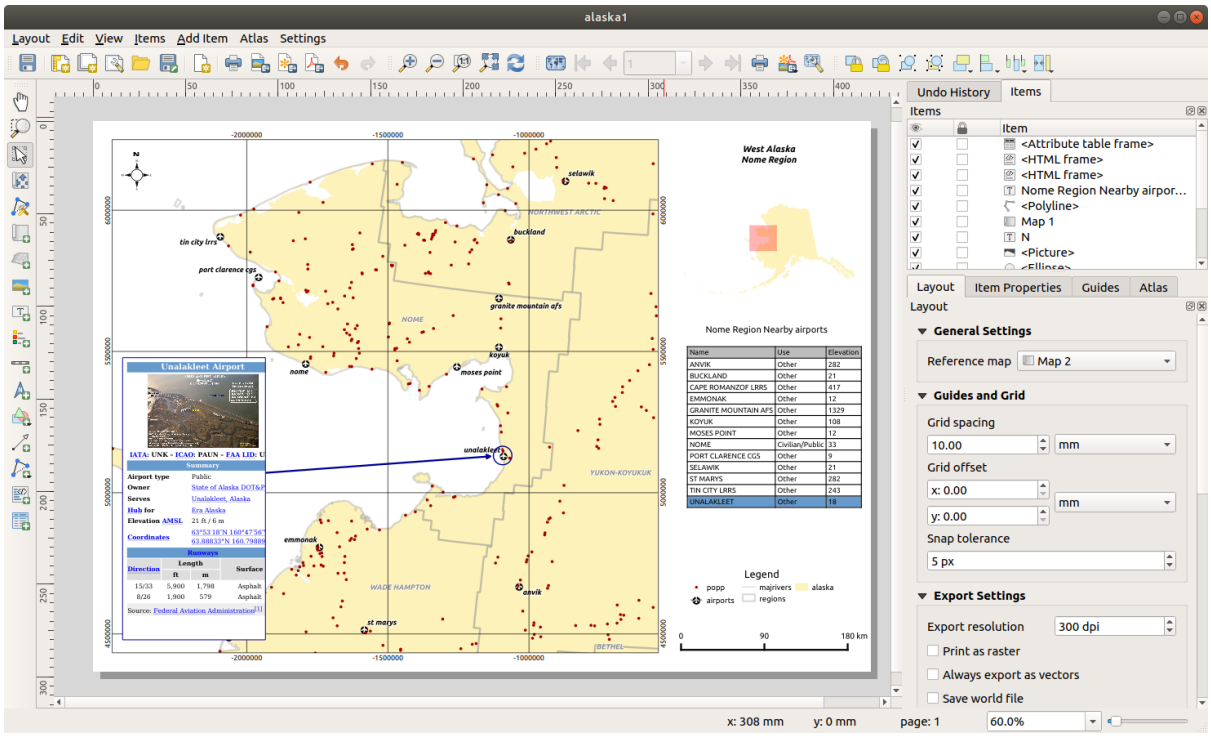


Fig. 18.64: Print Layout with map view, legend, image, scale bar, coordinates, text and HTML frame added

From the *Layout* menu or toolbar, you can output the print layout to different file formats, and it is possible to modify the resolution (print quality) and paper size:

- The Print icon allows you to print the layout to a connected printer or a PostScript file, depending on the installed printer drivers.

- The  Export as image icon exports the print layout image formats such as `PNG`, `BMP`, `TIF`, `JPG`, and many others…
- The  Export as SVG icon saves the print layout as an `SVG` (Scalable Vector Graphic).
- The  Export as PDF icon saves the defined print layout directly as a `PDF` (Portable Document Format) file.

18.3.1 Export settings

Whenever you export a print layout, there are a selection of export settings QGIS needs to check in order to produce the most appropriate output. These configurations are:

- The *Export settings* of the *Layout* panel, such as *Export resolution*, *Print as raster Always export as vectors* or *Save world file*
- *Exclude page from exports* in the *page item properties* panel
- *Exclude item from exports* in the *item properties* panel

Moreover, a number of predefined checks are automatically applied to the layout. Currently these checks include testing that scalebars are correctly linked to map items, and that map overview items are also correctly linked to a map. If the checks fail, you are shown a nice warning advising you of the issue.

18.3.2 Export as Image

To export a layout as an image:

1. Click the  Export as image icon
2. Select the image format, the folder and filename (e.g. `myill.png`) to use. If the layout contains more than one page, each page will be exported to a file with the given filename with the page number appended (e.g. `myill_2.png`).
3. In the next (*Image Export Options*) dialog:
   - You can override the print layout *Export resolution* and the exported page dimensions (as set in *Layout* panel).
   - For JPEG and JPG image formats, you can set the *JPEG Quality* of the output image.
   - Image rendering can also be improved with the *Enable antialiasing* option.
   - If you want to export your layout as a **georeferenced image** (e.g., to share with other projects), check the  *Generate world file* option, and an *ESRI World File* with the same name as the exported image, but a different extension (`.tfw` for TIFF, `.pnw` for PNG, `jgw` for JPEG, …) will be created when exporting. This option can also be checked by default in the *layout panel*.

     > **Note**
     >
     > For multi-page output, only the page that contains the *reference map* will get a world file (assuming that the *Generate world file* option is checked).

   - By checking  *Crop to content* option, the image output by the layout will include the minimal area enclosing all the items (map, legend, scale bar, shapes, label, image…) of each page of the composition:
     - If the composition includes a single page, then the output is resized to include EVERYTHING on the composition. The page can then be reduced or extended to all items depending on their position (on, above, below, left or right of the page).

  - In case of a multi-page layout, each page will be resized to include items in its area (left and right sides for all pages, plus top for the first page and bottom for the last page). Each resized page is exported to a separate file.

  The *Crop to content* dialog also lets you add margins around the cropped bounds.

- By checking *Open file after exporting* the exported file will automatically open in the default image viewer.

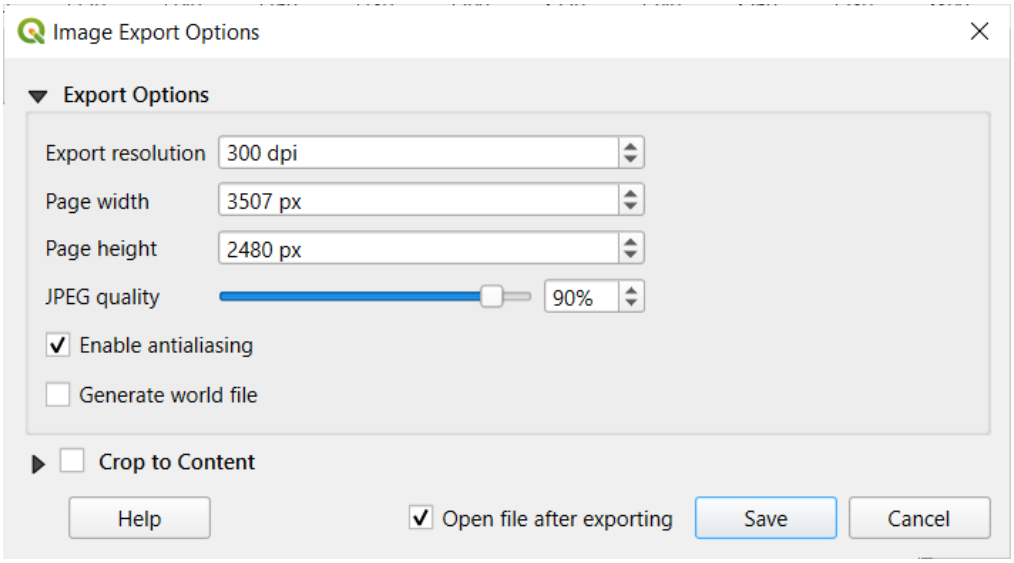


Fig. 18.65: Image Export Options

Tip

Use image formats that support transparency when items extend beyond the paper extent

Layout items may be placed outside the paper extent. When exporting with the *Crop to content* option, the resulting image may therefore extend beyond the paper extent. Since the background outside of the paper extent will be transparent, for image formats that do not support transparency (e.g. `BMP` and `JPG`) the transparent background will be rendered as full black, "corrupting" the image. Use transparency-compatible formats (e.g. `TIFF` and `PNG`) in such cases.

Note

When supported by the format (e.g. `PNG`) and the underlying Qt library, the exported image may include *project metadata* (author, title, date, description…)

18.3.3 Export as SVG

To export a layout as SVG:

1. Click the Export as SVG icon
2. Fill in the path and filename (used as a base name for all the files in case of multi-page composition, as for image export)
3. In the next *SVG Export Options* dialog, you can override the layout default *export settings* or configure new ones:

- *Export map layers as SVG groups*: exported items are grouped within layers whose name matches the layer names from QGIS, making it much easier to understand the contents of the document.
- *Always export as vectors*: some rendering options require items to be rasterized for a better rendering. Check this option to keep the objects as vectors with the risk that the appearance of the output file may not match the print layout preview (for more details, see *Export settings*).
- *Export RDF metadata* of the document such as the title, author, date, description…
- *Simplify geometries to reduce output file size*: this avoids exporting ALL geometry vertices, which can result in a ridiculously complex and large export file size that could fail to load in other applications. Geometries will be simplified while exporting the layout in order to remove any redundant vertices which are not discernably different at the export resolution (e.g. if the export resolution is `300 dpi`, vertices that are less than `1/600 inch` apart will be removed).
- Set the *Text export*: controls whether text labels are always or preferably exported as *text or outline objects*.
- Apply *Crop to content* *option*
- *Disable tiled raster layer exports*: When exporting files, QGIS uses a built-in raster layer tiled rendering that saves memory. Sometimes, this can cause visible "seams" in the rasters for generated files. Checking this option would fix that, at the cost of a higher memory usage during exports.
- By checking *Open file after exporting* the exported file will automatically open in the default SVG viewer.

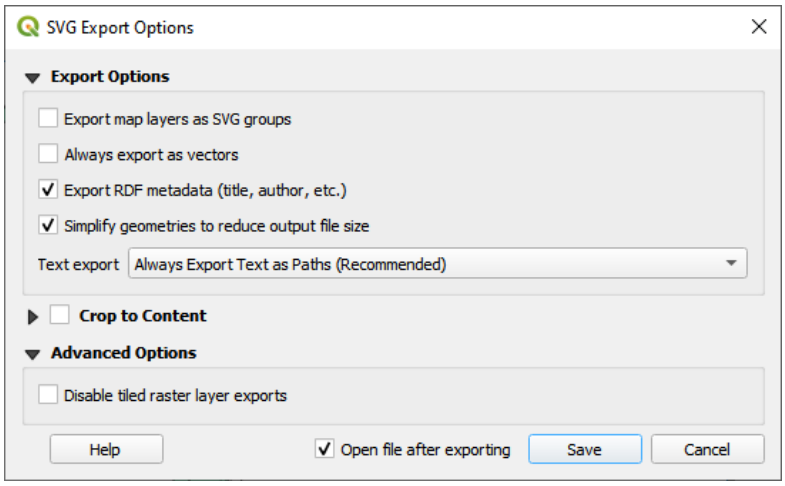


Fig. 18.66: SVG Export Options

Note

Currently, the SVG output is very basic. This is not a QGIS problem, but a problem with the underlying Qt library. This will hopefully be sorted out in future versions.

18.3.4 Export as PDF

To export a layout as PDF:

1. Click the  Export as PDF icon
2. Fill in the path and filename: unlike for image and SVG export, all the pages in the layout are exported to a single PDF file.
3. In the next *PDF Export Options* dialog, you can override the layout default *export settings* or configure new ones:
   - ☐ *Always export as vectors*: some rendering options require items to be rasterized for a better rendering. Check this option to keep the objects as vectors with the risk that the appearance of the output file may not match the print layout preview (for more details, see *Export settings*).

     Note

     Mind opacity settings for vector layers

     When you set a layer-wide opacity (or opacity for any vector symbology) to less than 100%, it forces a rasterized rendering of those objects during PDF exports. Depending on the size of the layer, this may considerably increase the PDF file size.

   - ☑ *Append georeference information*: available only if the *reference map*, from which the information is taken, is on the first page.
   - ☑ *Export RDF metadata* of the document such as the title, author, date, description…
   - Set the *Text export*: controls whether text labels are always or preferably exported as *text or outline objects*.
   - Control the PDF *Image compression* using:
     - *Lossy (JPEG)*, which is the default compression mode
     - or *Lossless*, which creates bigger files in most cases, but is much more suitable for printing outputs or for post-production in external applications (requires Qt 5.13 or later).
   - ☐ *Create Geospatial PDF*: Generate a georeferenced PDF file.
   - ☐ *Disable tiled raster layer exports*: When exporting files, QGIS uses tiled based rendering that saves memory. Sometimes, this can cause visible "seams" in the rasters for generated files. Checking this option would fix that, at the cost of a higher memory usage during exports.
   - ☑ *Simplify geometries to reduce output file size*: Geometries will be simplified while exporting the layout by removing vertices that are not discernably different at the export resolution (e.g. if the export resolution is `300 dpi`, vertices that are less than `1/600 inch` apart will be removed). This can reduce the size and complexity of the export file (very large files can fail to load in other applications).
   - By checking ☑ *Open file after exporting* the exported file will automatically open in the default PDF viewer.

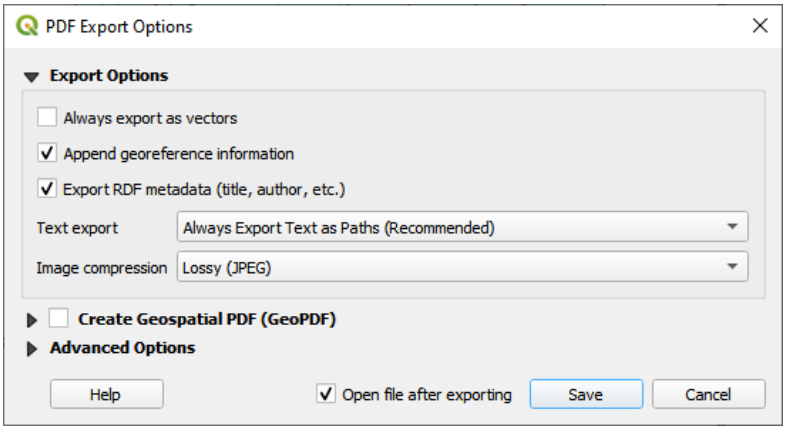


Fig. 18.67: PDF Export Options

Note

Geospatial PDF export is supported, and a number of Geospatial PDF specific options are available:

- *Format* (Geospatial PDF format - there are some variations),
- *Include multiple map themes* (specify map themes to include),
- *Include vector feature information* (choose the layers and group them into logical PDF groups).

Note

Exporting a print layout to formats that supports georeferencing (e.g. `PDF` and `TIFF`) creates a georeferenced output by default.

18.3.5 Generate an Atlas

Atlas functions allow you to create map books in an automated way. Atlas uses the features of a table or vector layer (*Coverage layer*) to create an output for each feature (**atlas feature**) in the table / layer. The most common usage is to zoom a map item to the current atlas feature. Further use cases include:

- a map item showing, for another layer, only features that share the same attribute as the atlas feature or are within its geometry.
- a label or HTML item whose text is replaced as features are iterated over
- a table item showing attributes of associated *parent or children* features of the current atlas feature…

For each feature, the output is processed for all pages and items according to their exports settings.

Tip

Use variables for more flexibility

QGIS provides a large panel of functions and *variables*, including atlas related ones, that you can use to manipulate the layout items, but also the symbology of the layers, according to atlas status. Combining these features gives you a lot of flexibility and helps you easily produce advanced maps.

To enable the generation of an atlas and access atlas parameters, refer to the *Atlas* panel. This panel contains the following (see Fig. 18.68):

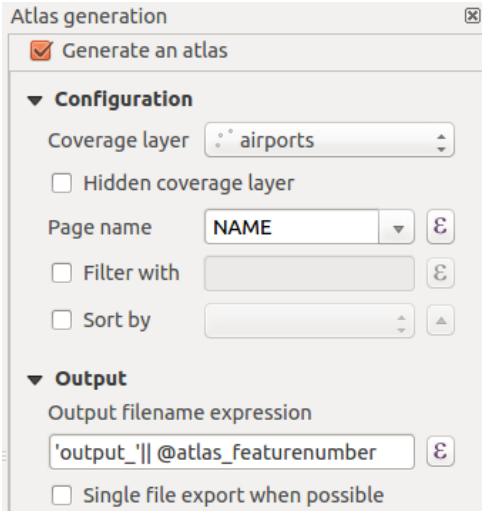


Fig. 18.68: Atlas Panel

- *Generate an atlas* enables or disables atlas generation.
- *Configuration*
  - A *Coverage layer* combo box that allows you to choose the table or vector layer containing the features to iterate over.
  - An optional *Hidden coverage layer* that, if checked, will hide the coverage layer (but not the other layers) during the generation.
  - An optional *Page name* combo box to specify the name for the feature page(s). You can select a field of the coverage layer or set an *expression*. If this option is empty, QGIS will use an internal ID, according to the filter and/or the sort order applied to the layer.
  - An optional *Filter with* text area that allows you to specify an expression for filtering features from the coverage layer. If the expression is not empty, only features that evaluate to `True` will be processed.
  - An optional *Sort by* that allows you to sort features of the coverage layer (and the output), using a field of the coverage layer or an expression. The sort order (either ascending or descending) is set by the two-state *Sort direction* button that displays an up or a down arrow.
- *Output* - this is where the output of the atlas can be configured:
  - An *Output filename expression* textbox that is used to generate a filename for each atlas feature. It is based on expressions. is meaningful only for rendering to multiple files.
  - A *Single file export when possible* that allows you to force the generation of a single file if this is possible with the chosen output format (`PDF`, for instance). If this field is checked, the value of the *Output filename expression* field is meaningless.
  - An *Image export format* drop-down list to select the output format when using the Export atlas as Images… button.

Control map by atlas

The most common usage of atlas is with the map item, zooming to the current atlas feature, as iteration goes over the coverage layer. This behavior is set in the *Controlled by atlas* group properties of the map item. See *Controlled by atlas* for different settings you can apply on the map item.

Customize labels with expression

In order to adapt labels to the feature the atlas iterates over, you can include expressions. Make sure that you place the expression part (including functions, fields or variables) between `[%` and `%]` (see *The Label Item* for more details).

For example, for a city layer with fields `CITY_NAME` and `ZIPCODE`, you could insert this:

```
The area of [% concat( upper(CITY_NAME), ',', ZIPCODE, ' is ',
format_number($area/1000000, 2) ) %] km2
```

or, another combination:

```
The area of [% upper(CITY_NAME)%],[%ZIPCODE%] is
[%format_number($area/1000000,2) %] km2
```

The information `[% concat( upper(CITY_NAME), ',', ZIPCODE, ' is ', format_number($area/1000000, 2) ) %]` is an expression used inside the label. Both expressions would result in the following type of label in the generated atlas:

```
The area of PARIS,75001 is 1.94 km2
```

Explore Data-defined override buttons with atlas

There are several places where you can use a Data defined override button to override the selected setting. This is particularly useful with atlas generation. See *Data defined override setup* for more details on this widget.

For the following examples the `Regions` layer of the QGIS sample dataset is used and selected as *Coverage layer* for the atlas generation. We assume that it is a single page layout containing a map item and a label item.

When the height (north-south) of a region extent is greater than its width (east-west), you should use *Portrait* instead of *Landscape* orientation to optimize the use of paper. With a Data Defined Override button you can dynamically set the paper orientation.

Right-click on the page and select *Page Properties* to open the panel. We want to set the orientation dynamically, using an expression depending on the region geometry, so press the button of field *Orientation*, select *Edit…* to open the *Expression string builder* dialog and enter the following expression:

```
CASE WHEN bounds_width(@atlas_geometry) > bounds_height(@atlas_geometry)
THEN 'Landscape' ELSE 'Portrait' END
```

Now if you *preview the atlas*, the paper orients itself automatically, but item placements may not be ideal. For each Region you need to reposition the location of the layout items as well. For the map item you can use the button of its *Width* property to set it dynamic using the following expression:

```
@layout_pagewidth - 20
```

Likewise, use the button of the *Height* property to provide the following expression to constrain map item size:

```
@layout_pageheight - 20
```

To ensure the map item is centered in the page, set its *Reference point* to the upper left radio button and enter `10` for its *X* and *Y* positions.

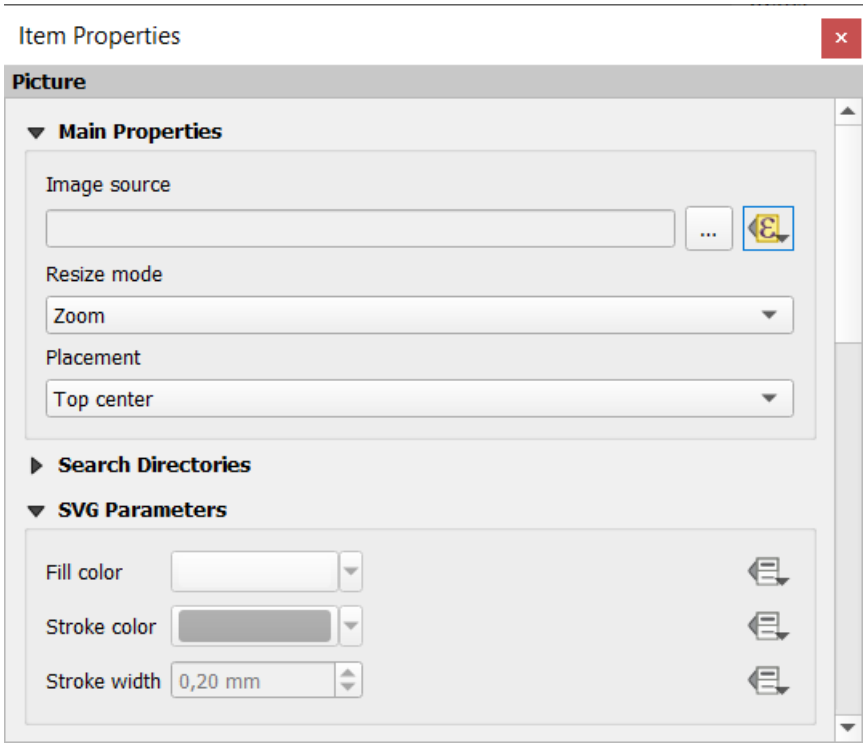
Let's add a title above the map in the center of the page. Select the label item and set the horizontal alignment to  *Center*. Next move the label to the right position, choose the middle button for the *Reference point*, and provide the following expression for field *X*:

```
@layout_pagewidth / 2
```

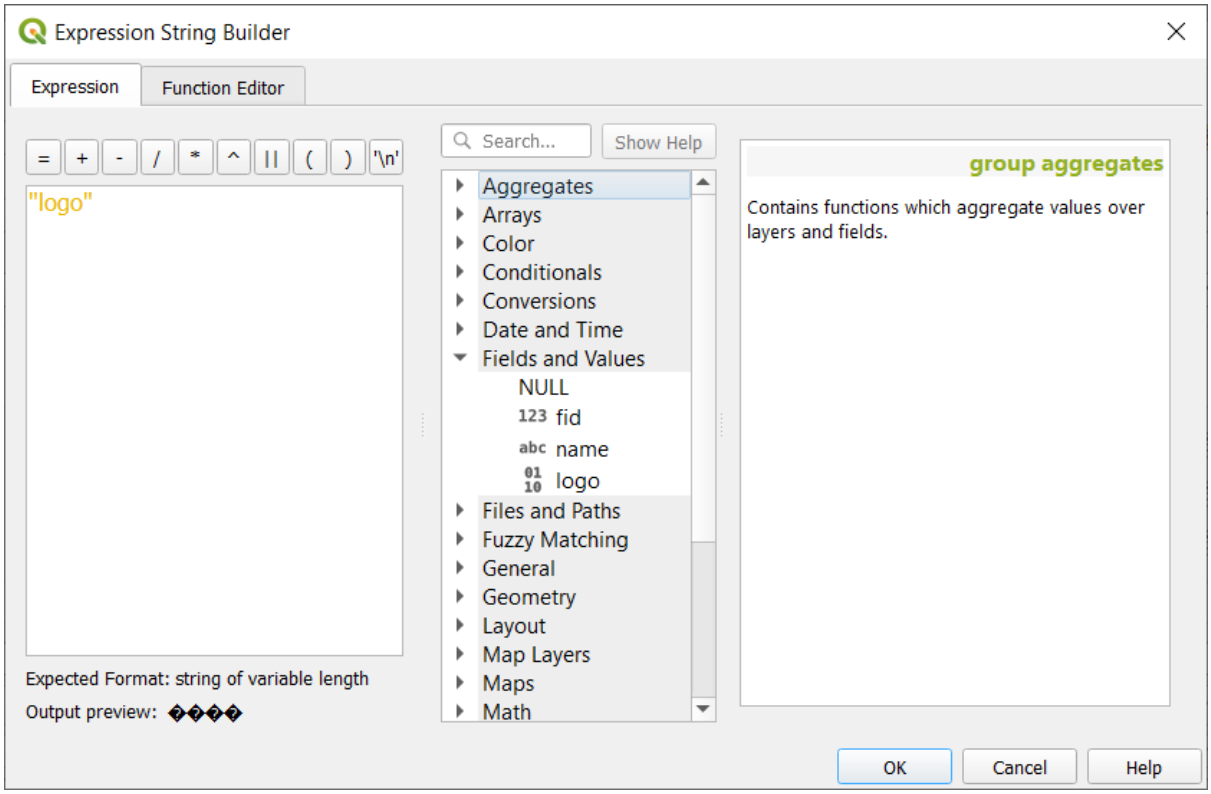
For all other layout items you can set the position in a similar way so they are correctly positioned both for portrait and landscape. You can also do more tweaks such as customizing the title with feature attributes (see *Customize labels with expression* example), changing images, resizing the number of legend columns number according to page orientation, …

The information provided here is an update of the excellent blog (in English and Portuguese) on the Data Defined Override options Multiple_format_map_series_using_QGIS_2.6 .

Another example for using data-defined override buttons is the usage of a dynamic picture. For the following examples we use a geopackage layer containing a BLOB field called `logo` with the field type binary (see *Creating a new GeoPackage layer*). For every feature there is defined a different picture so that the atlas can iterate over as described in *Preview and generate an atlas*. All you need to do is add a picture in the print layout and go to its *Item properties* in the atlas context. There you can find a data-defined override button in the *Image source* section of the *Main Properties*.



In the following window choose *Edit* so that the *Expression String Builder* opens. From the *Fields and values* section you can find the BLOB field that was defined in the geopackage layer. Double-click the field name `logo` and click *OK*.



The atlas iterates over the entries in the BLOB field provided that you choose the geopackage layer as *Coverage layer* (further instructions you can find in *Preview and generate an atlas*).

These are just two examples of how you can use some advanced settings with atlas.

Preview and generate an atlas



Fig. 18.69: Atlas Preview toolbar

Once the atlas settings have been configured, and layout items (map, table, image…) linked to it, you can create a preview of all the pages by choosing *Atlas* ► *Preview Atlas* or clicking the Preview Atlas icon. You can then use the arrows to navigate through all the features:

- First feature
- Previous feature
- Next feature
- Last feature

You can also use the combo box to select and preview a specific feature. The combo box shows atlas feature names according to the expression set in the atlas *Page name* option.

As for simple compositions, an atlas can be generated in different ways (see *Creating an Output* for more information - just use tools from the *Atlas* menu or toolbar instead of the *Layout* menu.

This means that you can directly print your compositions with *Atlas* ► *Print Atlas*. You can also create a PDF using *Atlas* ► *Export Atlas as PDF…*: You will be asked for a directory to save all the generated PDF files, except if the *Single file export when possible* has been selected. In that case, you'll be prompted to give a filename.

With *Atlas* ► *Export Atlas as Images…* or *Atlas* ► *Export Atlas as SVG…* tool, you're also prompted to select a folder. Each page of each atlas feature composition is exported to the image file format set in *Atlas* panel or to SVG.

Note

With multi-page output, an atlas behaves like a layout in that only the page that contains the *General settings* will get a world file (for each feature output).

Tip

Print a specific atlas feature

If you want to print or export the composition of only one feature of the atlas, simply start the preview, select the desired feature in the drop-down list and click on *Layout* ► *Print* (or *Export…* to any supported file format).

Use project defined relations for atlas creation

For users with HTML and Javascript knowledge it is possible to operate on GeoJSON objects and use project defined relations from the QGIS project. The difference between this approach and using expressions directly inserted into the HTML is that it gives you a full, unstructured GeoJSON feature to work with. This means that you can use existing Javascript libraries and functions that operate on GeoJSON feature representations.

The following code includes all related child features from the defined relation. Using the JavaScript `setFeature` function it allows you to make flexible HTML which represents relations in whatever format you like (lists, tables, etc). In the code sample, we create a dynamic bullet list of the related child features.

```
// Declare the two HTML div elements we will use for the parent feature id
// and information about the children
<div id="parent"></div>
<div id="my_children"></div>

<script type="text/javascript">
   function setFeature(feature)
   {
     // Show the parent feature's identifier (using its "ID" field)
     document.getElementById('parent').innerHTML = feature.properties.ID;
     //clear the existing relation contents
     document.getElementById('my_children').innerHTML = '';
     feature.properties.my_relation.forEach(function(child_feature) {
     // for each related child feature, create a list element
     // with the feature's name (using its "NAME" field)
       var node = document.createElement("li");
       node.appendChild(document.createTextNode(child_feature.NAME));
       document.getElementById('my_children').appendChild(node);
     });
   }
</script>
```

During atlas creation there will be an iteration over the coverage layer containing the parent features. On each page, you will see a bullet list of the related child features following the parent's identifier.

18.4 Creating a Report

This section will help you set up a report in QGIS.

18.4.1 What is it?

By definition, a GIS report is a document containing information organized in a narrative way, containing maps, text, graphics, tables, etc. A report can be prepared ad hoc, periodic, recurring, regular, or as required. Reports may refer to specific periods, events, occurrences, subjects or locations.

In QGIS, a *Report* is an extension of a *Layouts*.

Reports allow users to output their GIS projects in a simple, quick and structured way.

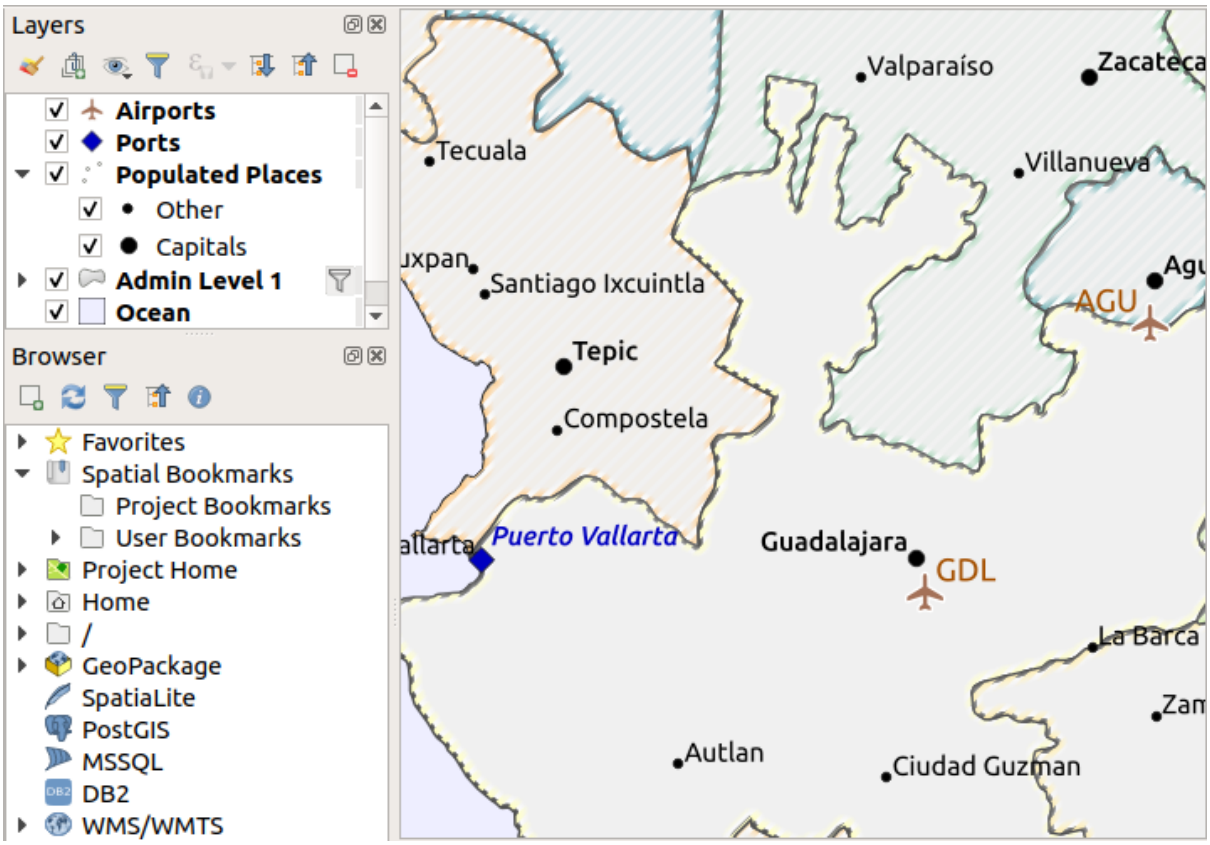
A report can be created with *Project* ► *New Report* or inside the *Project* ► *Layout Manager*.

> **Note**
>
> The maps in QGIS reports behave in the same way as maps in print layouts and atlases. We will concentrate on the specifics of QGIS reports. For details on map handling, see the sections on *print layouts* and *atlases*.

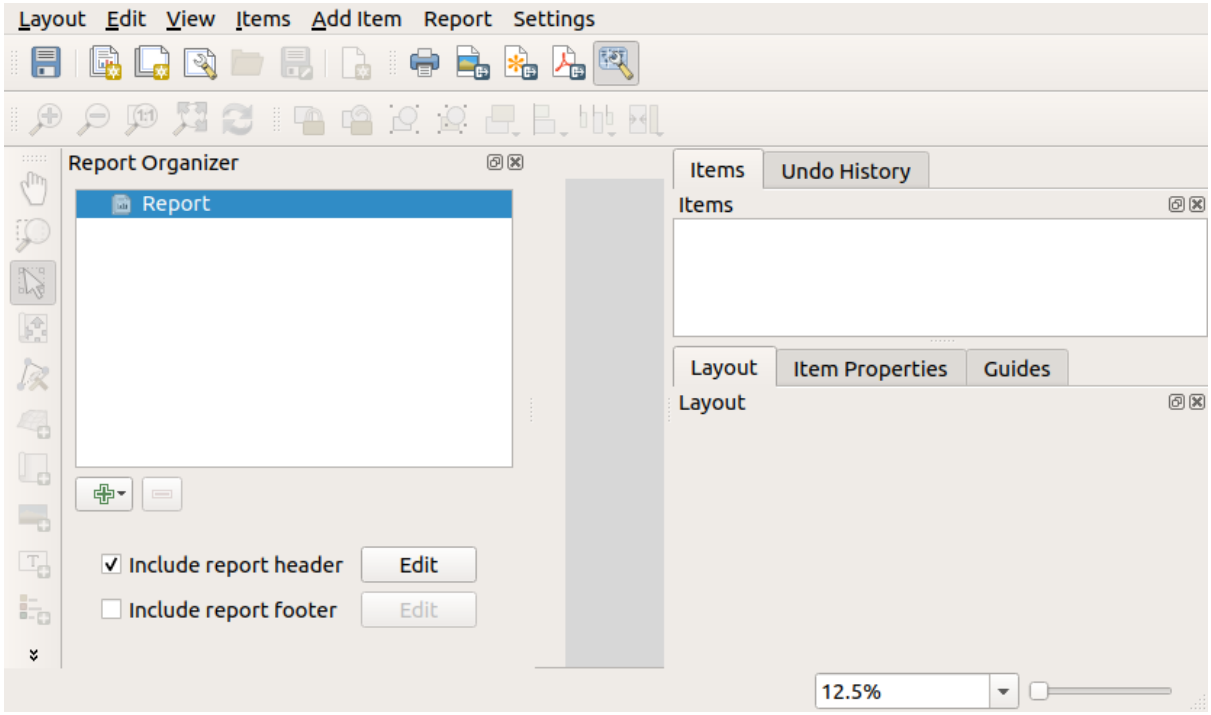
18.4.2 Get started

In the *Layout Manager* dialog a report can be created through *New from template* by selecting the dropdown option *Empty Report* and hitting the *Create…* button.

For this example, we use some administrative boundaries, populated places, ports and airports from the Natural Earth dataset (1:10M).

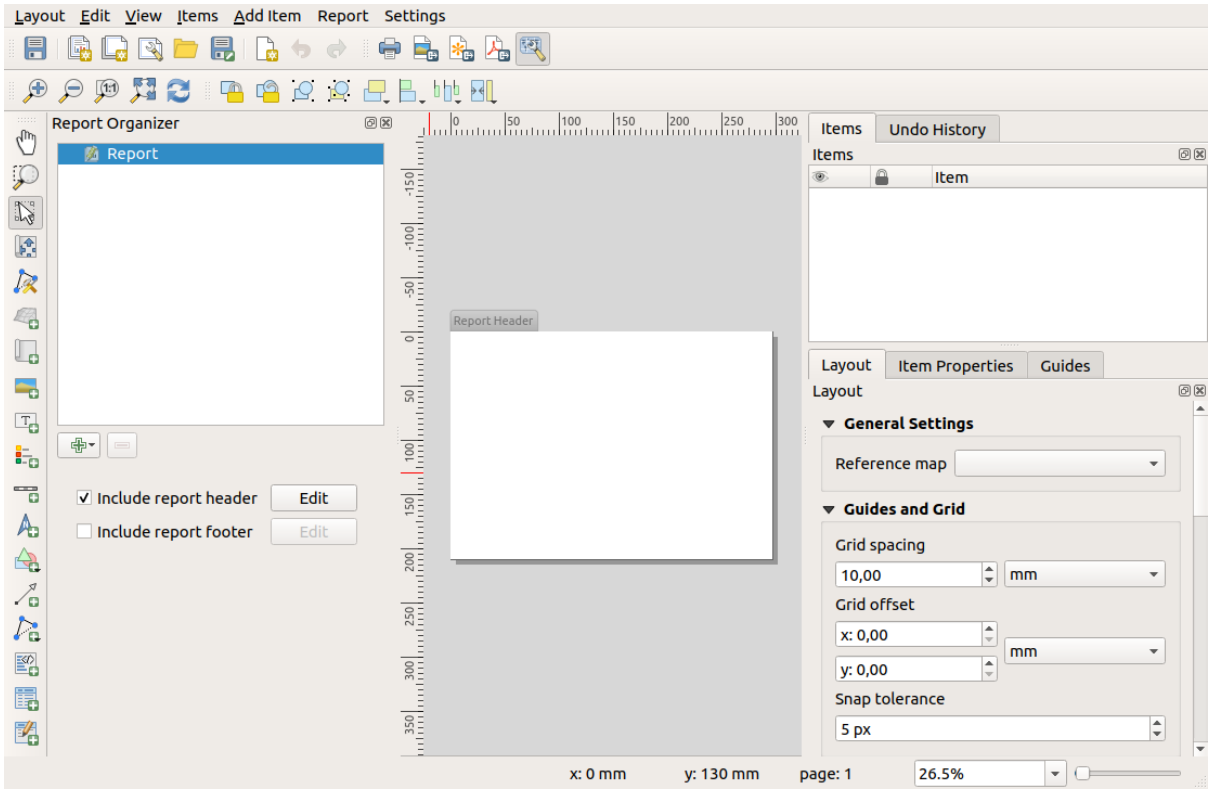


Using the *Project* ► *New Report* command, we create a blank report. Initially, there is not much to look at – the dialog which is displayed looks much like the print layout designer, except for the *Report Organizer* panel to the left:



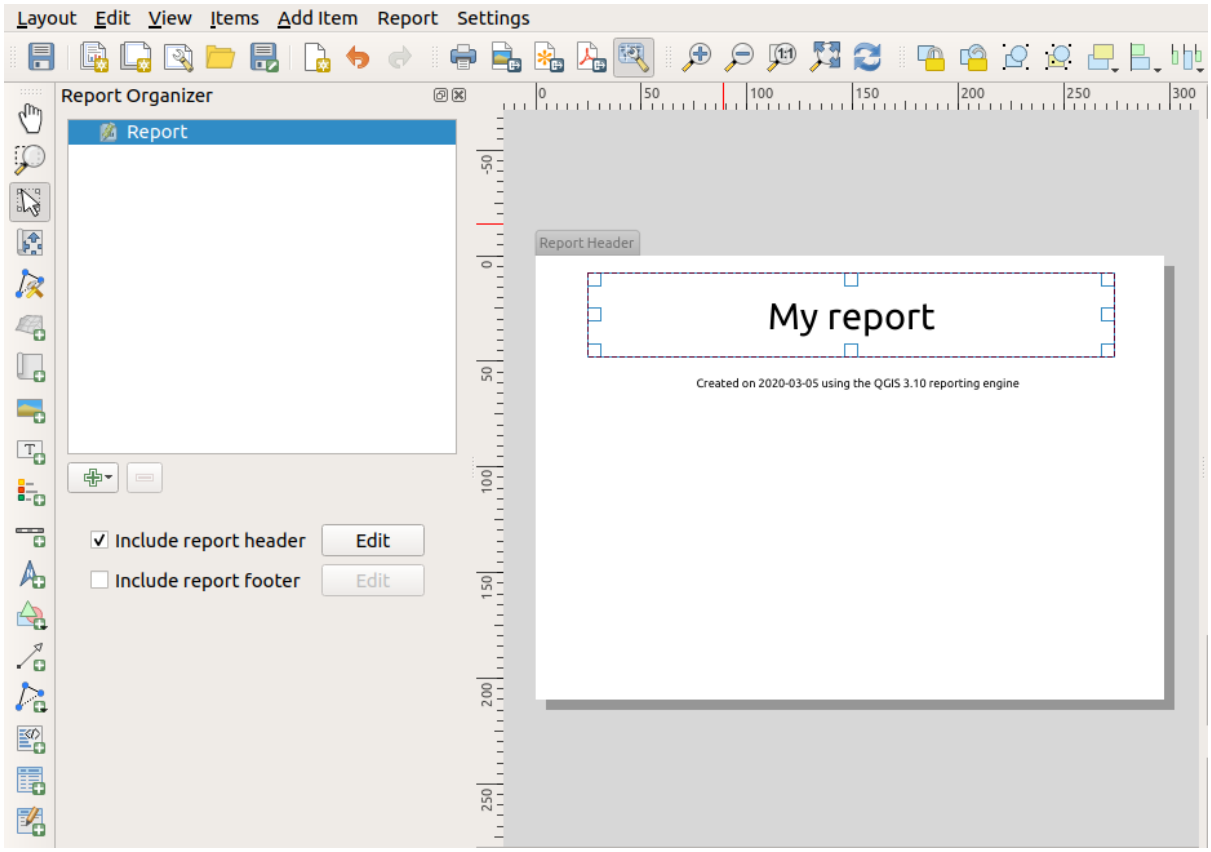
18.4.3 Layout Report Workspace

QGIS reports can consist of multiple, nested sections. In our new blank report we initially only have the main report section. The only options for this report section is *Include report header* and *Include report footer*. If we enable these options, a header will be included as the first page(s) (individual parts of reports can be multi-page if desired) in the report, and a footer will constitute the last page(s). Enable the header (*Include report header*), and hit the *Edit* button next to it:

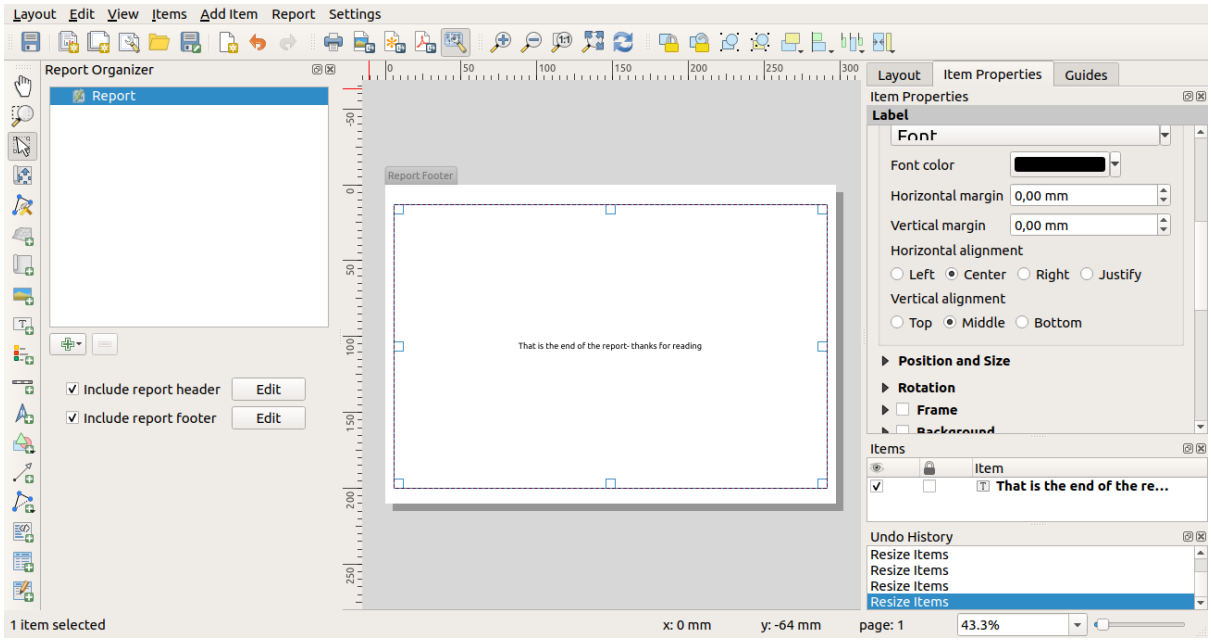


A few things happen as a result. Firstly, an edit pencil is shown next to *Report* in the *Report Organizer*, indicating that the report section is currently being edited in the designer. We also see a new page with a small *Report Header* title. The page has *landscape* orientation by default, but this (and other properties of the page) can be changed by right-clicking on the page and choosing *Page properties*. This will bring up the *Item properties* tab for the page, and page *Size*, *Width*, *Height*, and more can be specified.

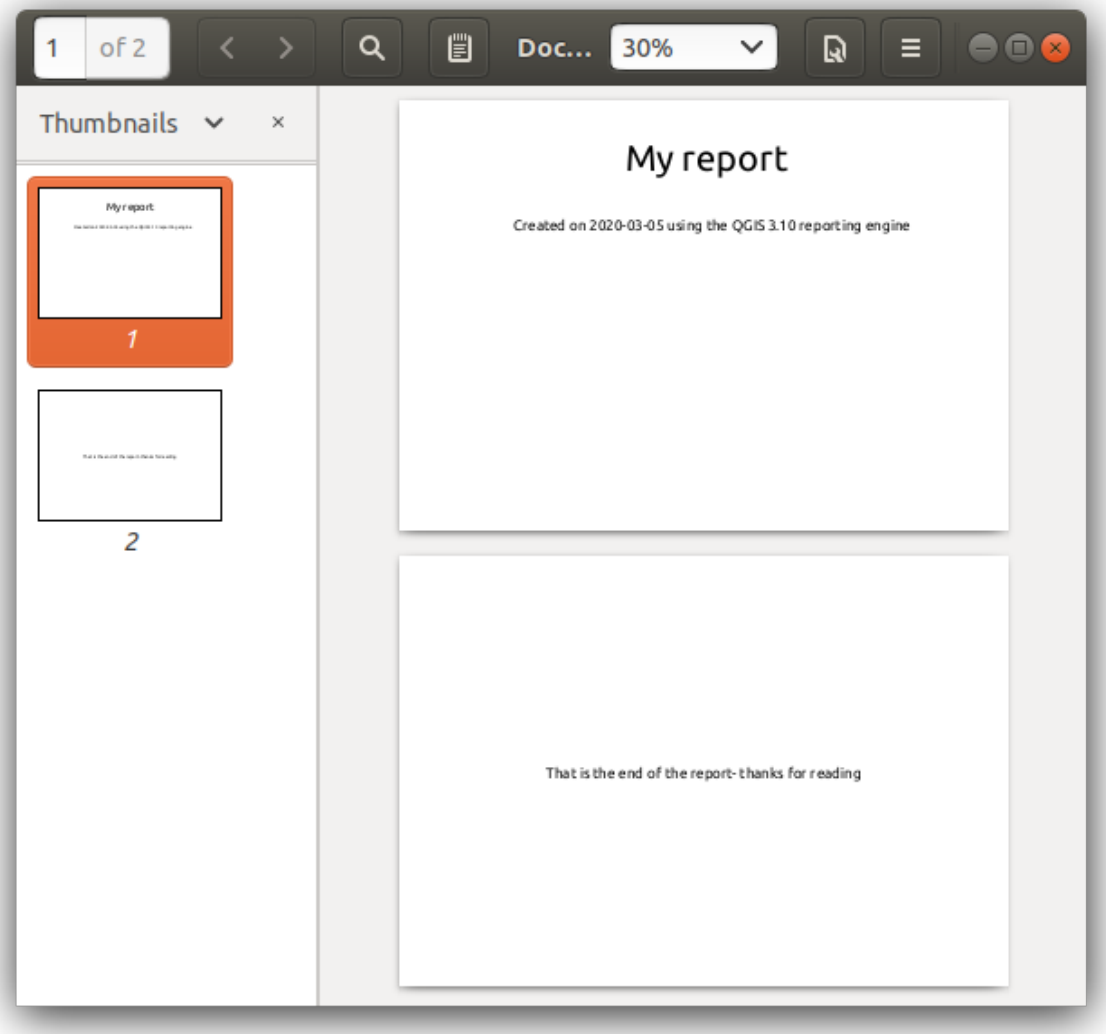
In QGIS reports, every component of the report is made up of individual layouts. They can be created and modified using the same tools as for standard print layouts – so you can use any desired combination of labels, pictures, maps, tables, etc. Let us add some items to our report header to demonstrate:



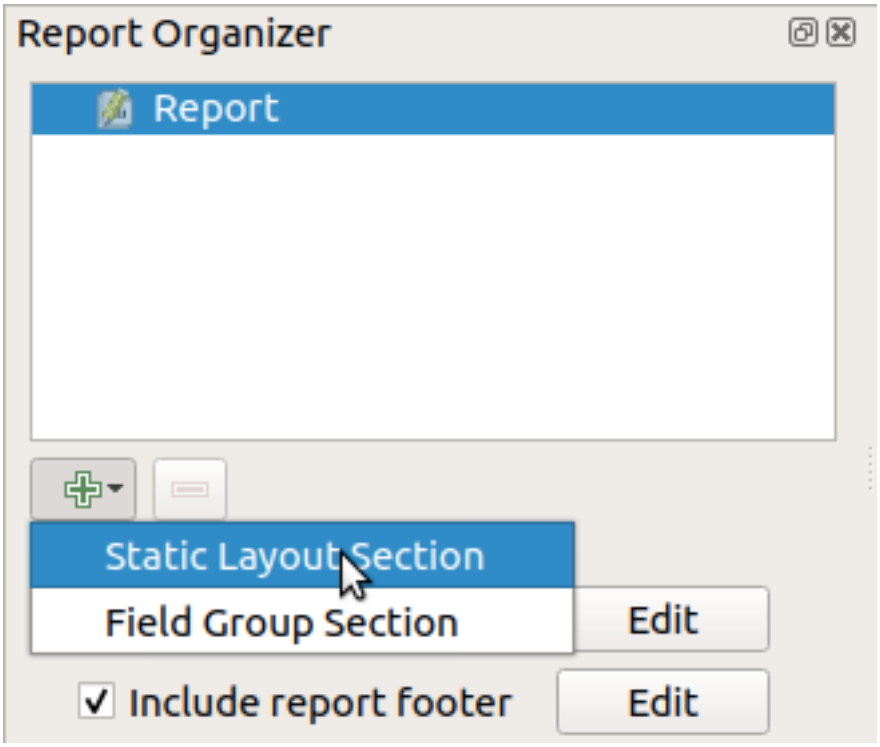
We will also create a simple footer for the report by checking the *Include report footer* option and hitting *Edit*.



Before proceeding further, let us export this report and see what we get. Exporting is done from the *Report* menu – in this case we select *Export Report as PDF…* to render the whole report to a PDF file. Here is the not-very-impressive result – a two page PDF consisting of our header and footer:



Let us make things more interesting. By hitting the Add Section button in the *Report Organizer*, we are given a choice of new sections to add to our report.

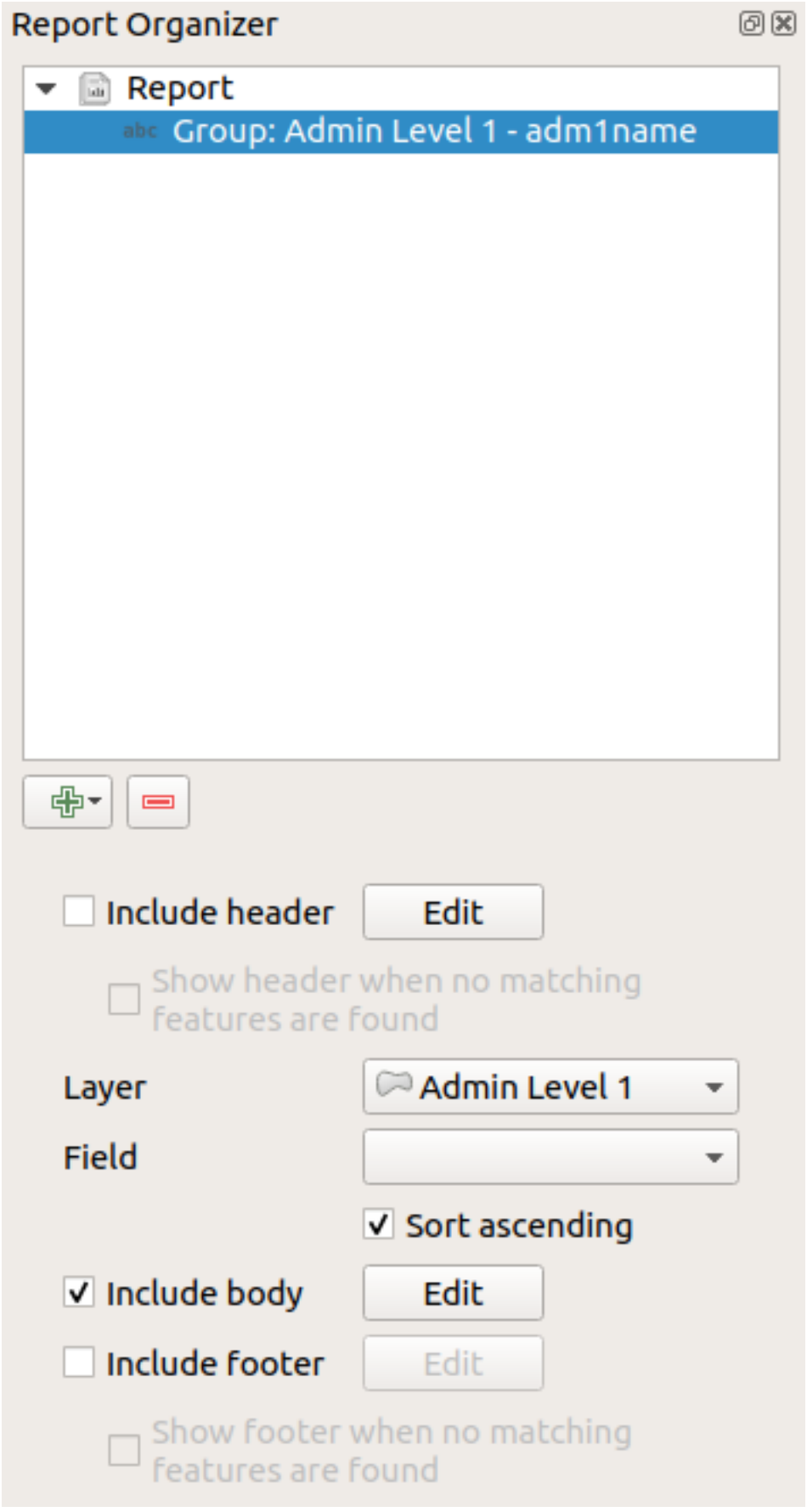


There are two options: *Static Layout Section* and *Field Group Section*.

The *Add Static Layout Section* is a single, static body layout. This can be used to embed static layouts mid-way through a report.

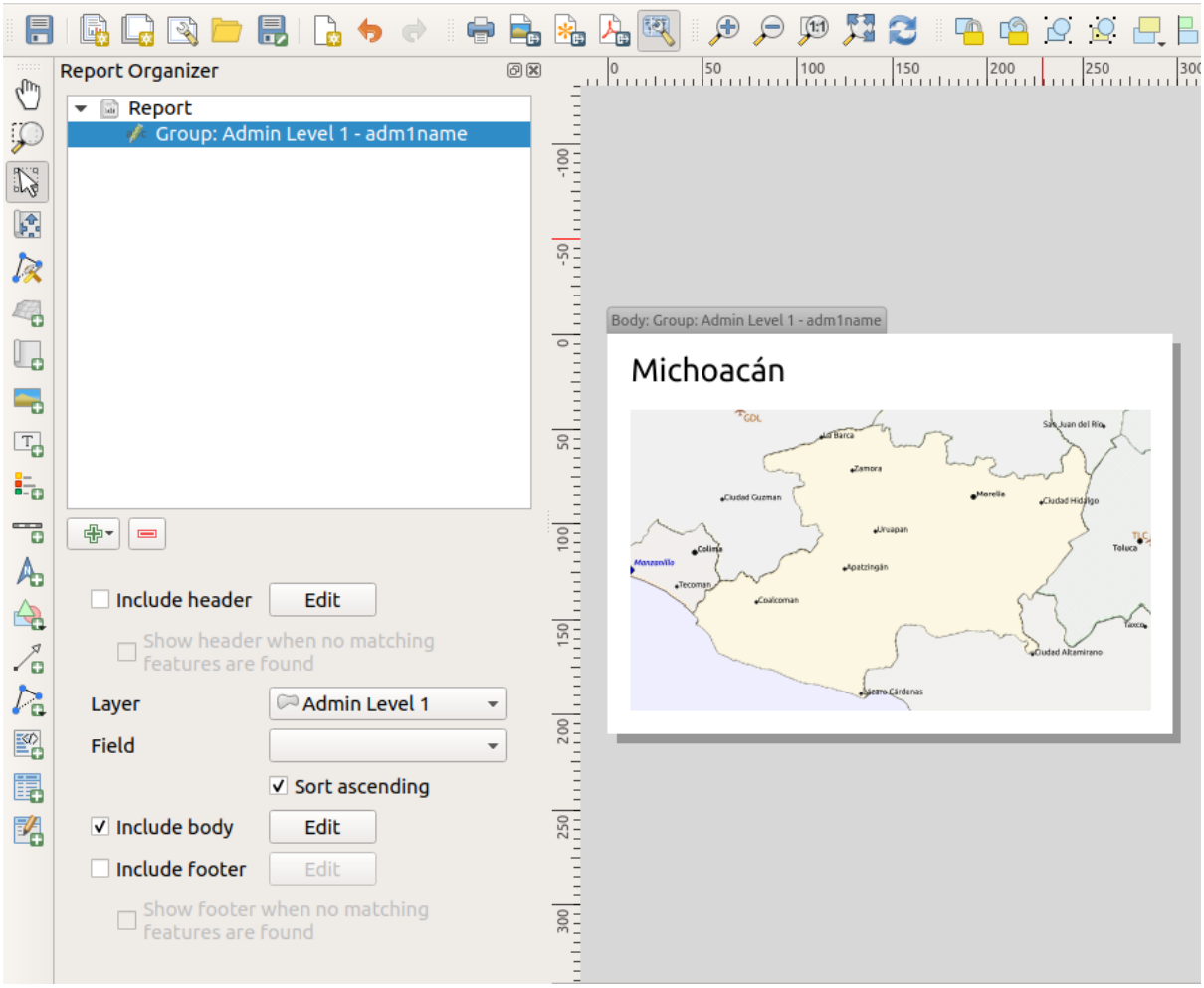
The *Field Group Section* repeats its body layout for every feature of a layer. The features are sorted by the selected grouping feature (with an option for ascending/descending sort). If a field group section has child sections (e.g. another field group section with a different field), then only features with unique values for the group feature are iterated over. This allows nested reports.

For now we will add a *Field Group Section* to our report. At its most basic level, you can think of a *Field Group Section* as the equivalent of a *print atlas*: you select a layer to iterate over, and the report will insert a section for each feature found. Selecting the new *Field Group Section* reveals a number of new related settings:



In this case we've setup our Field Group so that we iterate over all the states from the *Admin Level 1* layer, using the

values from the *adm1name* field. The same options to include header and footer are present, together with a new option to include a *body* for this section. We'll do that, and edit the body:

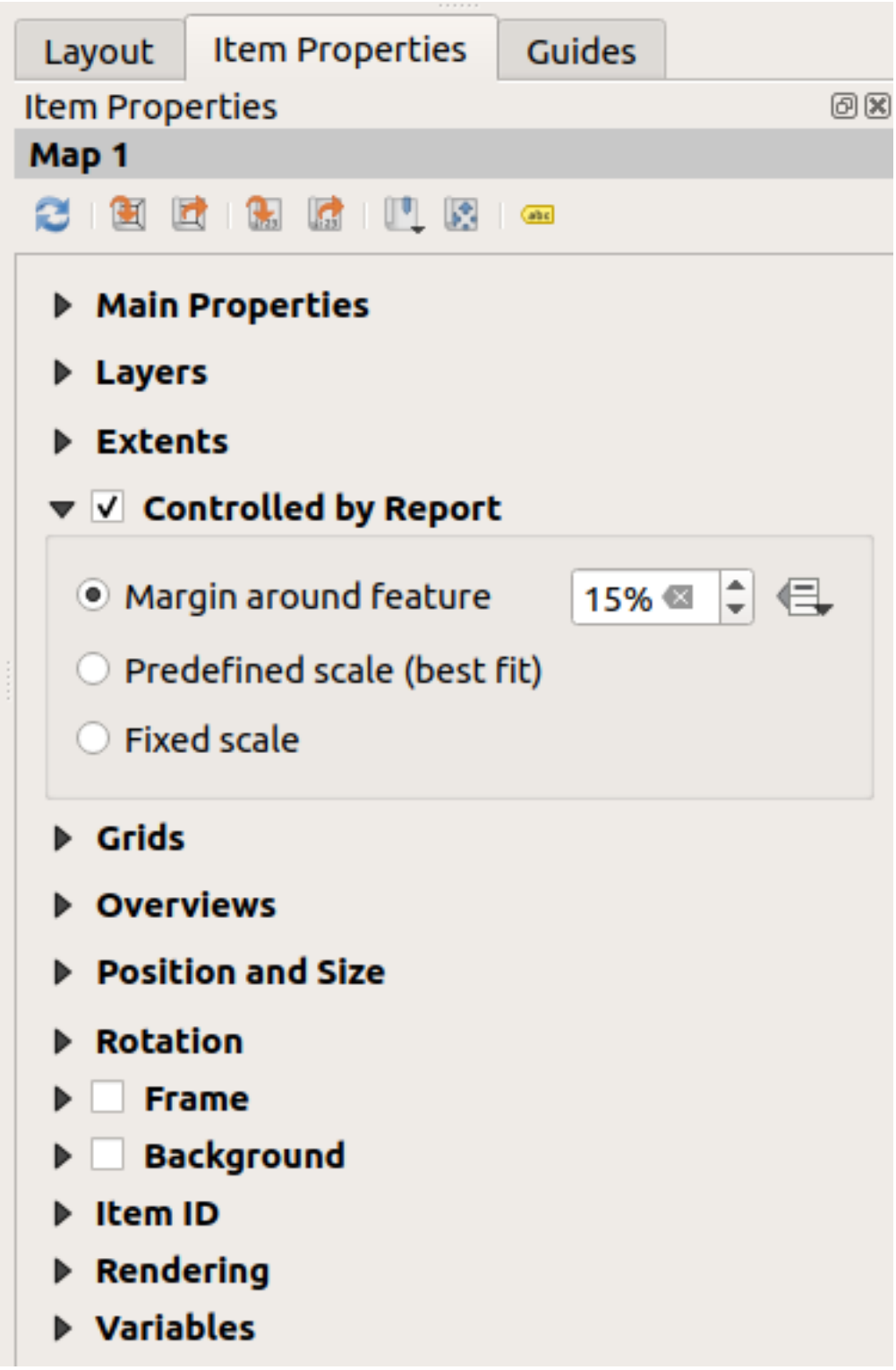


Our body now consists of a map and a label showing the name of the state. To include the name of the state, we selected *Add Item* ► *Add Label* and data defined the text under *Main Properties* with the help of *Insert or Edit an Expression*….

The result was the following expression (*name* is the name of the attribute in the *Admin Level 1* layer that contains the name of the state):

```
[% "name" %]
```

The map is set to follow the current report feature (enabled by checking *Controlled by Report* – just like a map item in an atlas will follow the current atlas feature when *Controlled by Atlas* is checked):



If we went ahead and exported our report now, we'd get something like this:

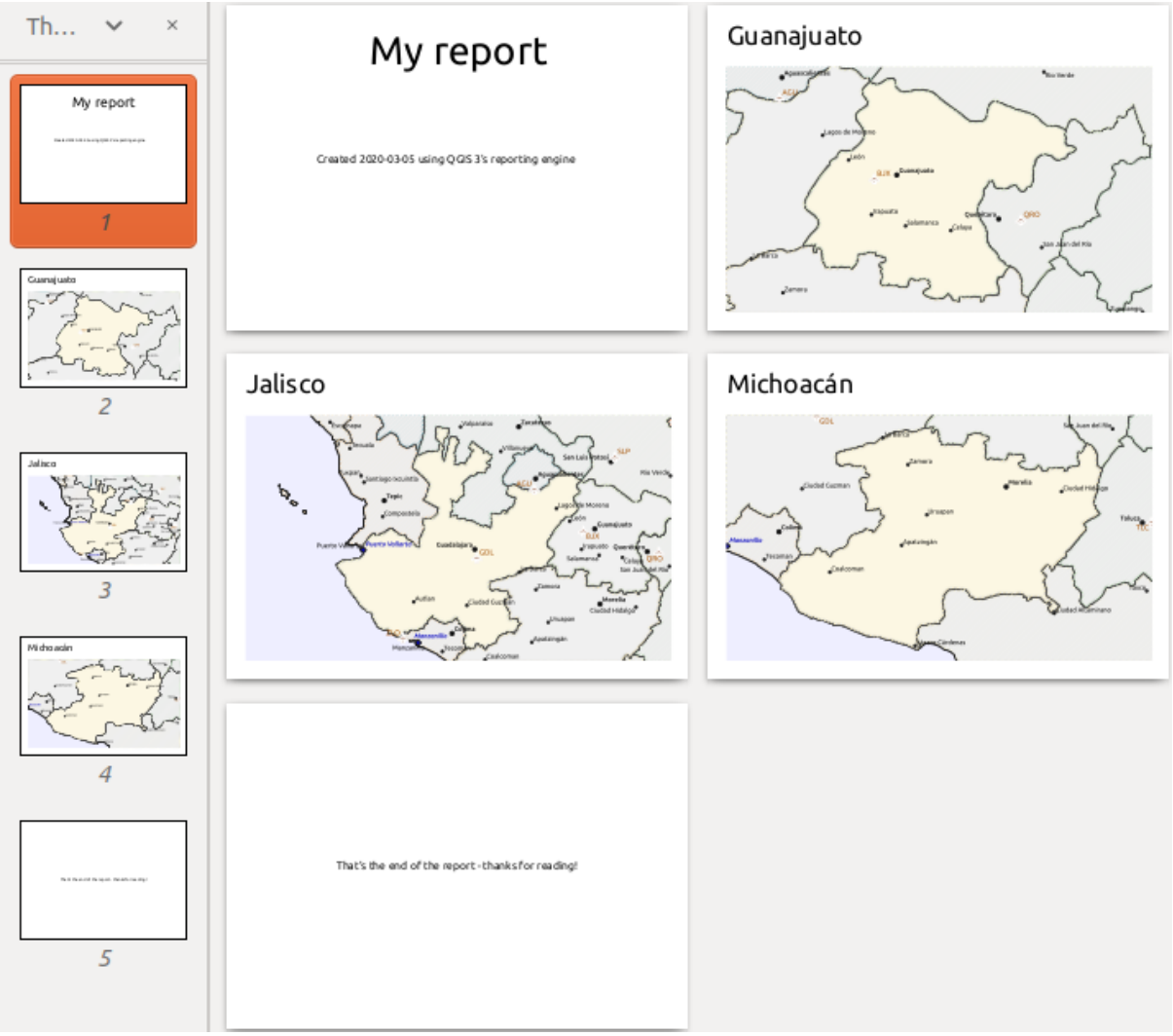
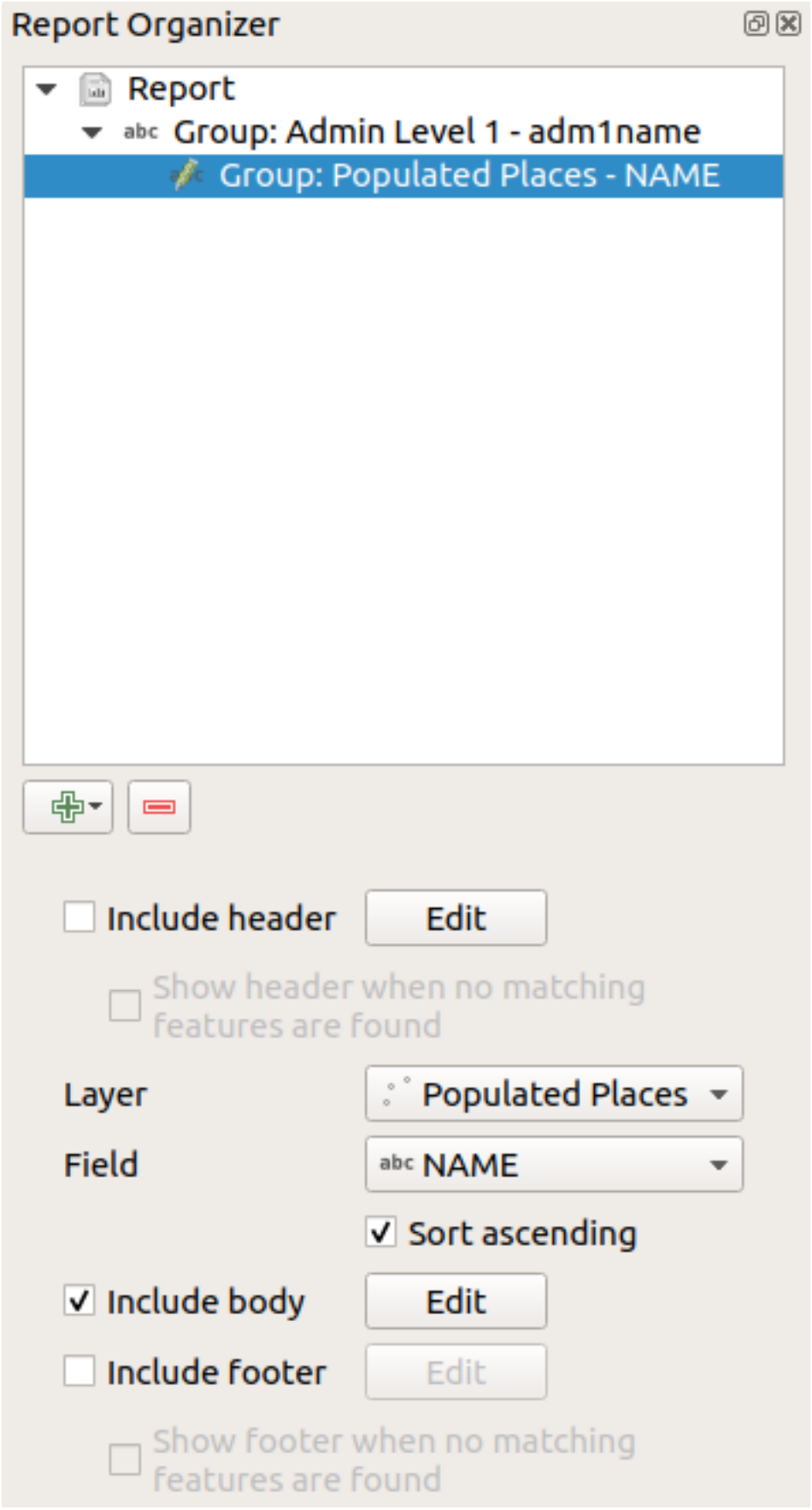


Fig. 18.70: The report header, a page for each state, and the report footer.

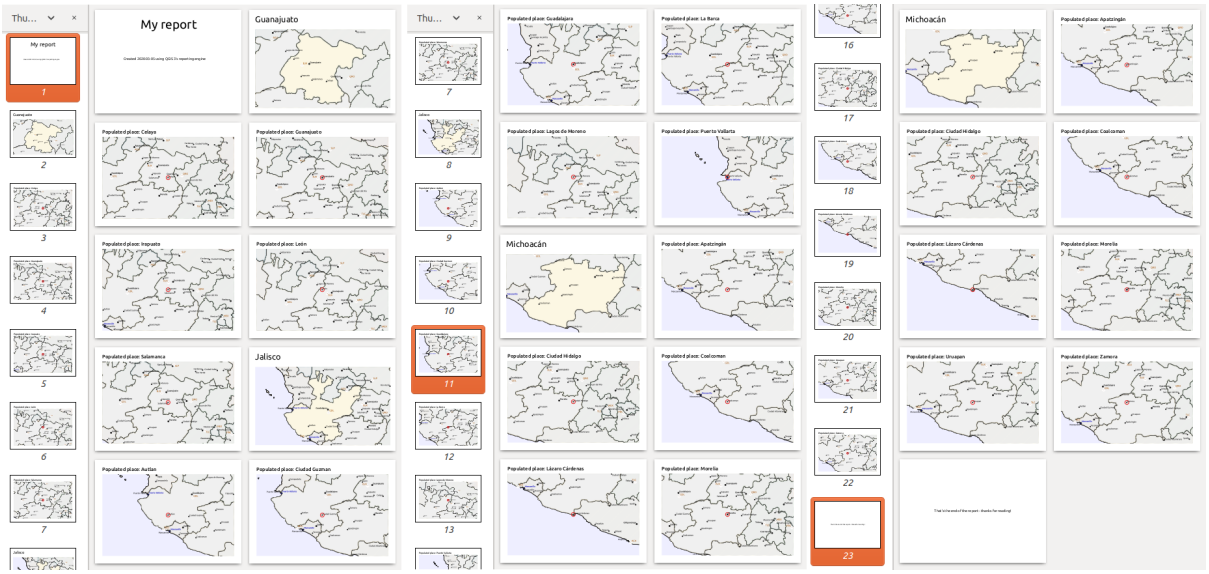
So more or less an atlas, but with a header and footer page.

Let us make things more interesting by adding a subsection to our state group. We do this by first selecting the *Admin Level 1* field group in the organizer, then hitting the Add Field button and adding a new *Field Group Section*:



When iterating over the features of a *Field Group Section*, the features will be filtered to match the defining field

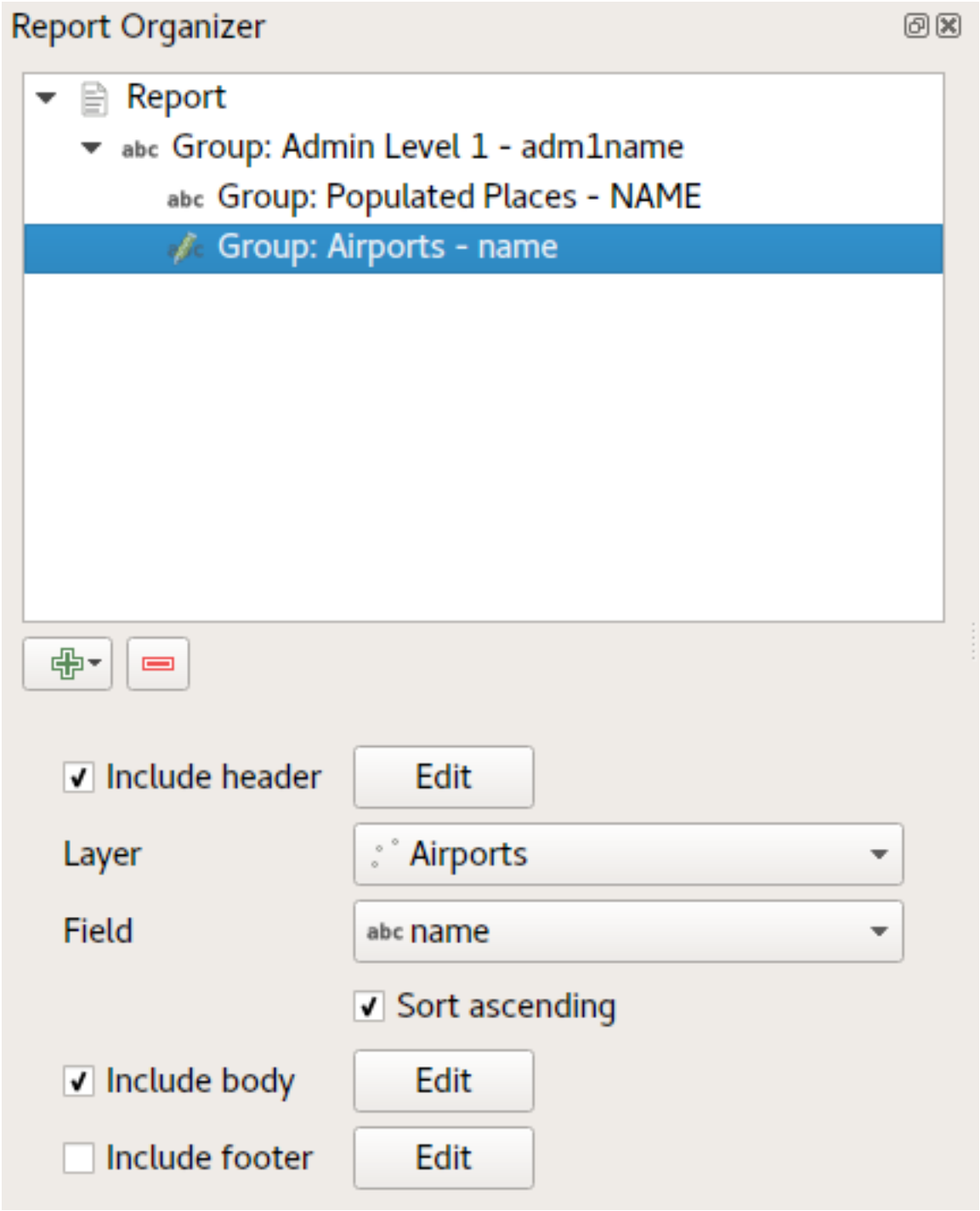
of its parent group (`adm1name` in this case). Here, the subsection we added will iterate over a *Populated Places* layer, including a body section for each place encountered. The magic here is that the *Populated Places* layer has an attribute with the same name as the defining field in the parent layer, *adm1name*, tagging each place with the state it is contained within (if you're lucky your data will already be structured like this – if not, run the *Join Attributes by Location* Processing algorithm and create your own field). When we export this report, QGIS will grab the first state from the *Admin Level 1* layer, and then iterate over all the *Populated Places* with a matching *adm1name* value. Here's what we get:



Here we created a basic body for the Populated Places group, including a map of the place and a table of some place attributes. So our report is now a report header, a page for the first state, followed by a page for every populated place within that state, then the rest of the states with their populated places, and finally the report footer. If we were to add a header for the Populated Places group, it would be included just before listing the populated places for each state, as shown in the illustration below.

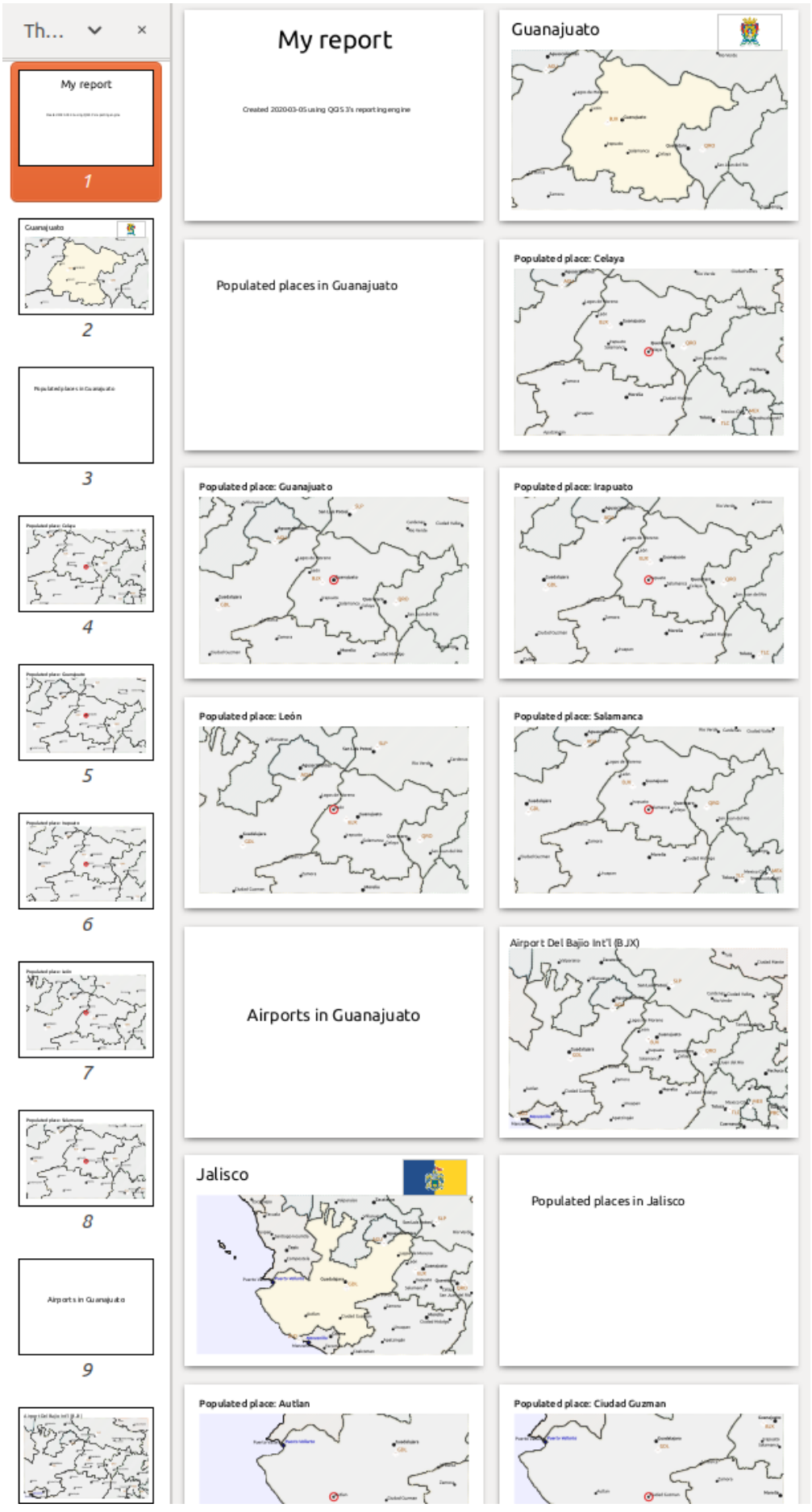
Similarly, a footer for the Populated Places group would be inserted after the final place for each state is included.

In addition to nested subsections, subsections in a report can also be included consecutively. If we add a second subsection to the *Admin Level 1 group* for *Airports*, then (if the *Airports* layer has an attribute *adm1name* that can link it to the parent group) our report will first list ALL the populated places for each state, followed by all the airports within that state, before proceeding to the next state.



The key point here is that our *Airports group* is a subsection of the *Admin Level 1 group* – not the *Populated Places group*.

In this case our report would be structured like this (note that state flags have also been included - the procedure for adding feature specific pictures in this way is described below):


Th...
My report
1
2
3
4
5
6
7
8
9
My report
Created 2020-03-05 using QGIS 3's reporting engine
Guanajuato
Populated places in Guanajuato
Populated place: Celaya
Populated place: Guanajuato
Populated place: Irapuato
Populated place: León
Populated place: Salamanca
Airports in Guanajuato
Airport Del Bajio Int'l (BJX)
Jalisco
Populated places in Jalisco
Populated place: Autlan
Populated place: Ciudad Guzman

Including pictures in a report

Pictures can be quite useful in reports, and QGIS allows pictures in both the static and dynamic parts of a report. Pictures are added in the same way as for standard print layouts, and for the static report parts (and static pictures in dynamic parts) there is not more to it.

But if you want illustrations that are tailored to the report features, your layer must have an attribute that can be used to define the picture to include.

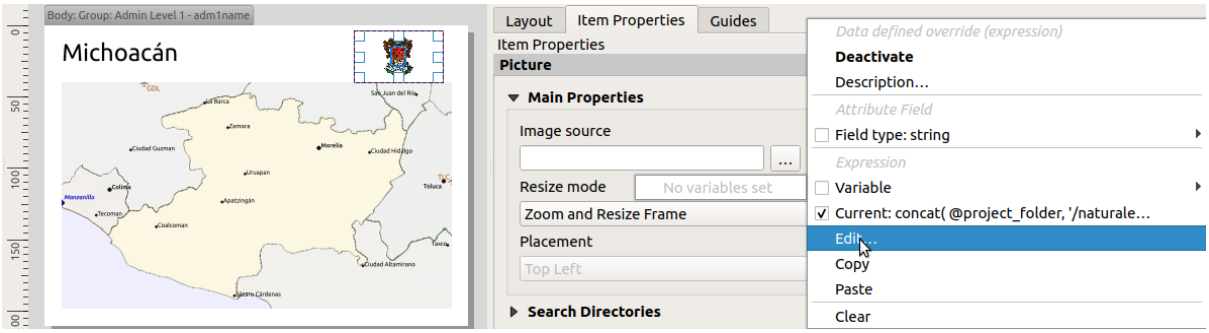
QGIS depends on absolute file names for images in reports.

For dynamic pictures, you first add a picture to the body part of the group, as usual. In the *Item properties* of the picture, you set the *Image Source* using the Data defined override button, and either select an attribute that contains the absolute path of the images or *Edit...* (to enter an expression that generates the absolute image path).

Below is an example expression that uses string concatenation to specify the absolute path to the pictures, using the directory where the project file is located (`@project_path`) and an attribute (`adm1name`) from which the file name is generated (in this case by transforming the string in the `adm1name` attribute to uppercase, and appending '_flag.png'):

```
concat(@project_folder, '/naturalearth/pictures/' ,
       upper("adm1name"), '_flag.png')
```

This means that the pictures are located in the `naturalearth/pictures` subdirectory of the project file directory.



Highlighting the current report feature in a map

In the above report, the report features are emphasized in the maps using highlighting (state) and circles (populated places). To emphasize the report features in the maps (apart from placing them at the centre of the maps), you must data define the style using a comparison between its `@id` and the `@atlas_featureid`, as for atlases.

For instance, if you would like to use a thicker line / border for the report feature than the other features you can data define the line width:

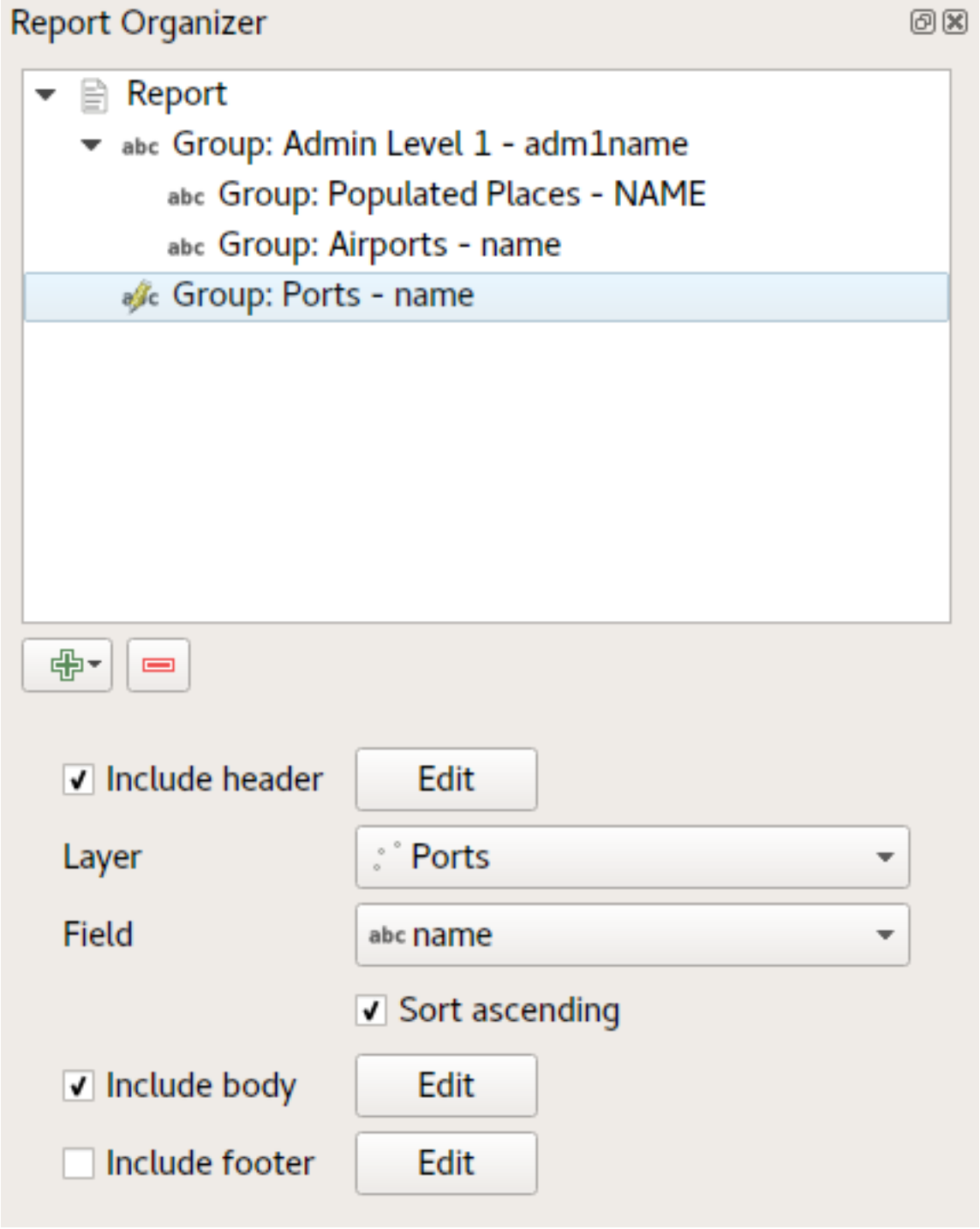
```
if($id=@atlas_featureid, 2.0, 0.1)
```

The report feature will get a 2 units wide polygon outline, while all other features will get a 0.1 units wide line. It is also possible to data define the colour (non-transparent dark magenta for the report feature and semi-transparent light gray for the other features):

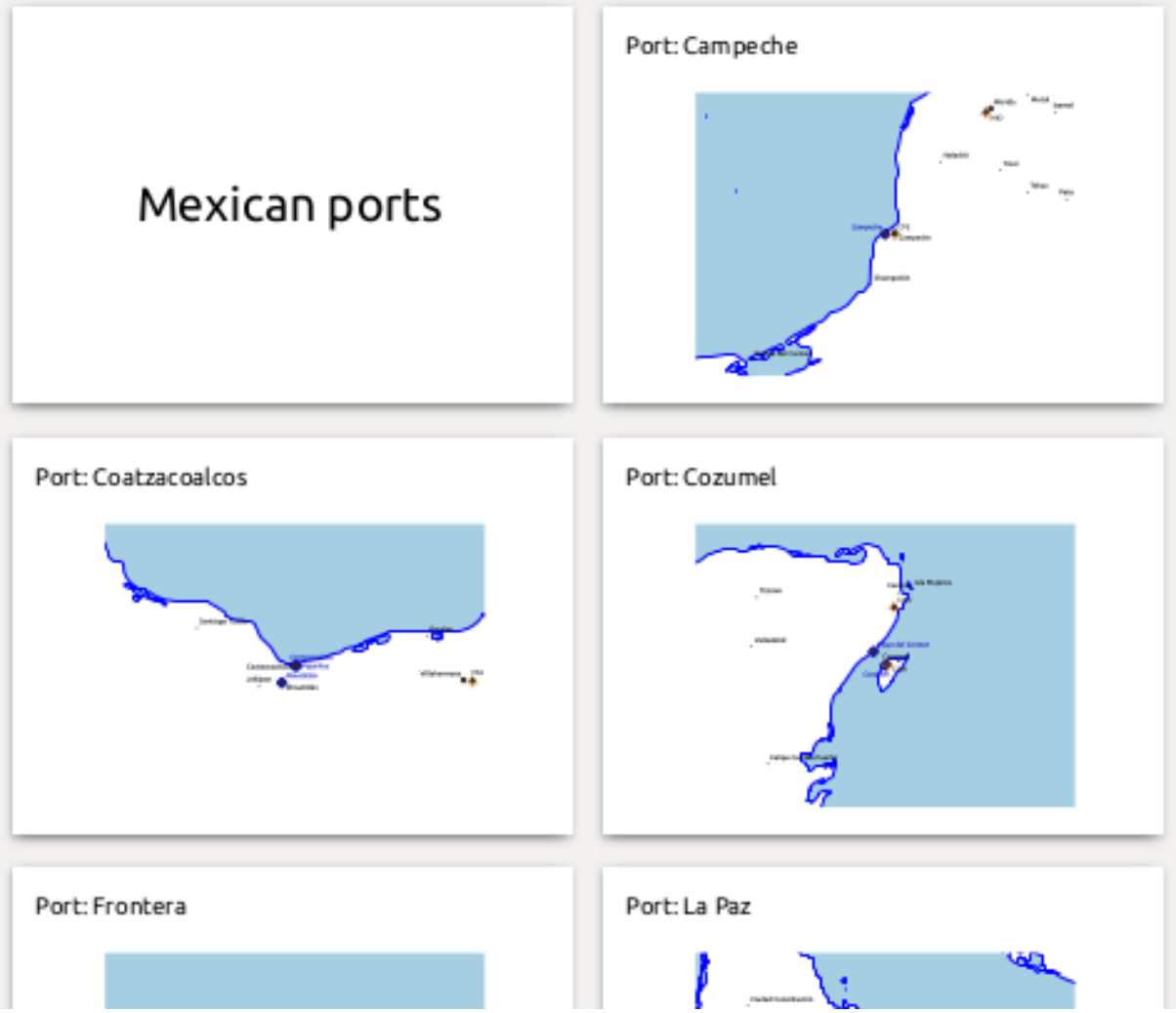
```
if($id=@atlas_featureid, '#FF880088', '#88CCCCCC')
```

More level 1 groups

Combining nested and consecutive sections, together with section headers and footers allows for tons of flexibility. For instance, in the below report we add another field group as a child of the main report for the *Ports* layer. Now, after listing the states together with their populated places and airports, we'll get a summary list of all the ports in the region:



This results in the last part of our report exporting as:



18.4.4 Export settings

When you export a report (*Report* ► *Export Report as Images… / SVG… / PDF…*), you will be asked for a file name, and then you get the opportunity to tune the export settings to get the most appropriate output.

As you see, reports in QGIS are extremely powerful and flexible!

> **Note**
>
> The current information was adapted from a North Road blog, Exploring Reports in QGIS 3.0 - the Ultimate Guide!

CHAPTER

NINETEEN

WORKING WITH OGC / ISO PROTOCOLS

The Open Geospatial Consortium (OGC) is an international organization with membership of more than 300 commercial, governmental, nonprofit and research organizations worldwide. Its members develop and implement standards for geospatial content and services, GIS data processing and exchange.

Describing a basic data model for geographic features, an increasing number of specifications are developed by OGC to serve specific needs for interoperable location and geospatial technology, including GIS. Further information can be found at https://www.ogc.org/.

Important OGC specifications supported by QGIS are:

- **WMS** — Web Map Service (*WMS/WMTS Client*)
- **WMTS** — Web Map Tile Service (*WMS/WMTS Client*)
- **WFS** — Web Feature Service (*WFS and WFS-T Client*)
- **WFS-T** — Web Feature Service - Transactional (*WFS and WFS-T Client*)
- **WCS** — Web Coverage Service (*WCS Client*)
- **WPS** — Web Processing Service
- **CSW** — Catalog Service for the Web
- **SFS** — Simple Features for SQL (*PostgreSQL Database*)
- **GML** — Geography Markup Language
- **SensorThings API** — SensorThings API (*SensorThings*)

OGC services are increasingly being used to exchange geospatial data between different GIS implementations and data stores. QGIS can deal with the above specifications as a client, being **SFS** (through support of the PostgreSQL data provider, see section *PostgreSQL Database*).

You can also share your maps and data through the WMS, WMTS, WFS, WFS-T and WCS protocols using a webserver with QGIS Server, UMN MapServer or GeoServer installed.

19.1 WMS/WMTS Client

19.1.1 Overview of WMS Support

QGIS currently can act as a WMS client that understands WMS 1.1, 1.1.1 and 1.3 servers. In particular, it has been tested against publicly accessible servers such as DEMIS.

A WMS server acts upon requests by the client (e.g., QGIS) for a raster map with a given extent, set of layers, symbolization style, and transparency. The WMS server then consults its local data sources, rasterizes the map, and sends it back to the client in a raster format. For QGIS, this format would typically be JPEG or PNG.

WMS is generically a REST (Representational State Transfer) service rather than a full-blown Web service. As such, you can actually take the URLs generated by QGIS and use them in a web browser to retrieve the same images that

QGIS uses internally. This can be useful for troubleshooting, as there are several brands of WMS server on the market and they all have their own interpretation of the WMS standard.

WMS layers can be added quite simply, as long as you know the URL to access the WMS server, you have a serviceable connection to that server, and the server understands HTTP as the data transport mechanism.

Additionally, QGIS will cache your WMS responses (i.e. images) for 24h as long as the GetCapabilities request is not triggered. The GetCapabilities request is triggered every time the *Connect* button in the *WMS/WMTS* dialog is used to retrieve the WMS server capabilities. This is an automatic feature meant to optimize project loading time. If a project is saved with a WMS layer, the corresponding WMS tiles will be loaded from the cache the next time the project is opened as long as they are not older than 24h.

19.1.2 Overview of WMTS Support

QGIS can also act as a WMTS client. WMTS is an OGC standard for distributing tile sets of geospatial data. This is a faster and more efficient way of distributing data than WMS because with WMTS, the tile sets are pre-generated, and the client only requests the transmission of the tiles, not their production. A WMS request typically involves both the generation and transmission of the data. A well-known example of a non-OGC standard for viewing tiled geospatial data is Google Maps.

In order to display the data at a variety of scales close to what the user might want, the WMTS tile sets are produced at several different scale levels and are made available for the GIS client to request them.

This diagram illustrates the concept of tile sets:

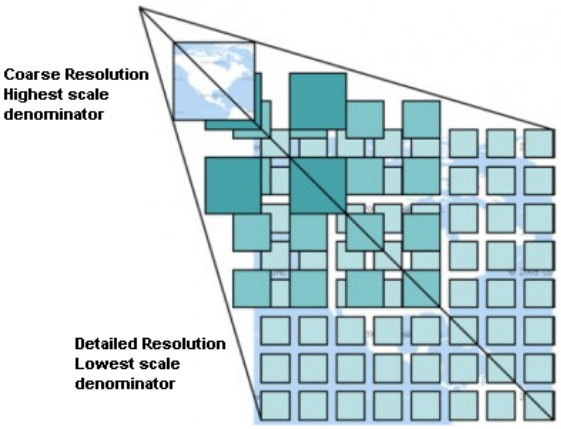


Fig. 19.1: Concept of WMTS tile sets

The two types of WMTS interfaces that QGIS supports are via Key-Value-Pairs (KVP) and RESTful. These two interfaces are different, and you need to specify them to QGIS differently.

1. In order to access a **WMTS KVP** service, a QGIS user must open the WMS/WMTS interface and add the following string to the URL of the WMTS tile service:

   ```
   "?SERVICE=WMTS&REQUEST=GetCapabilities"
   ```

   An example of this type of address is:

   ```
   https://opencache.statkart.no/gatekeeper/gk/gk.open_wmts?service=WMTS&
   ↪request=GetCapabilities
   ```

   For testing the topo2 layer in this WMTS works nicely. Adding this string indicates that a WMTS web service is to be used instead of a WMS service.

2. The **RESTful WMTS** service takes a different form, a straightforward URL. The format recommended by the OGC is:

```
{WMTSBaseURL}/1.0.0/WMTSCapabilities.xml
```

This format helps you to recognize that it is a RESTful address. A RESTful WMTS is accessed in QGIS by simply adding its address in the WMS setup in the URL field of the form. An example of this type of address for the case of an Austrian basemap is:

```
https://maps.wien.gv.at/basemap/1.0.0/WMTSCapabilities.xml
```

Note

You can still find some old services called WMS-C. These services are quite similar to WMTS (i.e., same purpose but working a little bit differently). You can manage them the same as you do WMTS services. Just add `?tiled=true` at the end of the url. See https://wiki.osgeo.org/wiki/Tile_Map_Service_Specification for more information about this specification.

When you read WMTS, you can often think WMS-C also.

19.1.3 Selecting WMS/WMTS Servers

The first time you use the WMS/WMTS feature in QGIS, there are no servers defined.

You then need to create connections to the server you are targeting:

1. Go to the *WMS/WMTS* tab of the *Data Source Manager* dialog, either by:
   - clicking the Open Data Source Manager button (or pressing `Ctrl+L`) and enabling the tab
   - clicking the Add WMS layer button on the *Manage Layers* toolbar
   - or selecting *Layer* ► *Add Layer* ► *Add WMS/WMTS Layer…* menu
2. Press *New* from the *Layers* tab. The *Create a New WMS/WMTS Connection…* dialog appears.

   Tip

   Right-click the *WMS/WMTS* entry from within the *Browser panel* and select *New Connection…* also opens the *Create a New WMS/WMTS Connection…* dialog.

3. Then enter the parameters to connect to your desired WMS server, as listed below:

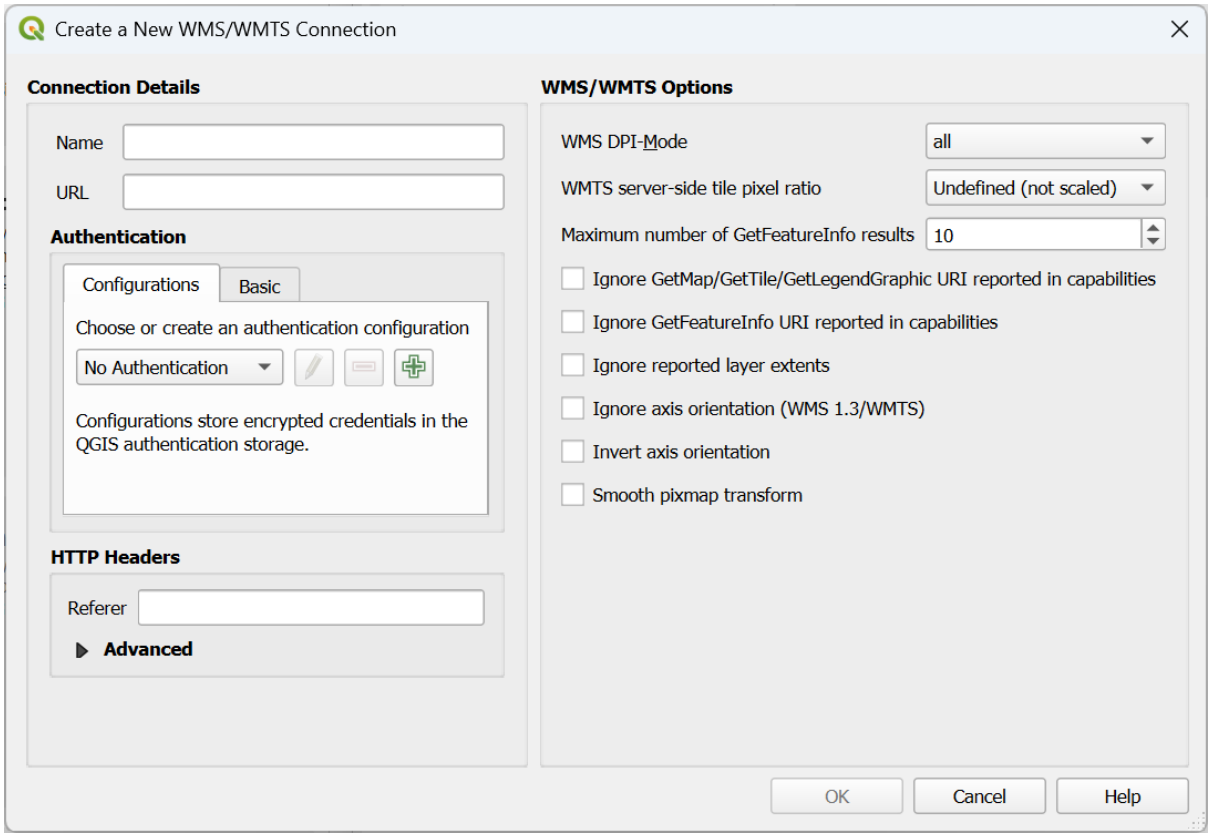


Fig. 19.2: Creating a connection to a WMS server

- *Name*: A name for the connection. This name will be used in the Server Connections drop-down box so that you can distinguish it from other WMS servers.
- *URL*: URL of the server providing the data. This must be a resolvable host name – the same format as you would use to open a telnet connection or ping a host, i.e. the base URL only. For example, you shouldn't have fragments such as `request=GetCapabilities` or `version=1.0.0` in your URL.
- *Authentication* (optional): using a *stored configuration* or a basic authentication with *Username* and *Password*.

> **Warning**
>
> Entering **username** and **password** in the *Authentication* tab will keep unprotected credentials in the connection configuration. Those **credentials will be visible** if, for instance, you shared the project file with someone. Therefore, it's advisable to save your credentials in an *Authentication configuration* instead (*Configurations* tab). See *Authentication System* for more details.

- HTTP *Referer*
- *WMS DPI-Mode*: Available options are **all**, **off**, **QGIS**, **UMN** and **GeoServer**
- *WMTS server-side tile pixel ratio*: When rendering WMTS layers, allows to scale up or down the tiles based on the device screen DPI. Available options are *Undefined (not scaled)*, *Standard (96 DPI)* and *High (192 DPI)*.
- *Maximum number of GetFeatureInfo results*: specifies a default value for the maximum number of results returned per layer by a GetFeatureInfo request using this connection (see FEATURE_COUNT parameter). Default value is `10`. Set to `0` to use server default value (usually `1`): no FEATURE_COUNT parameter will be added to the request.

- ☐ *Ignore GetMap/GetTile/GetLegendGraphic URI reported in capabilities*: if checked, use given URI from the *URL* field above.
- ☐ *Ignore GetFeatureInfo URI reported in capabilities*: if checked, use given URI from the *URL* field above.
- ☐ *Ignore reported layer extents*: because the extent reported by raster layers may be smaller than the actual area which can be rendered (notably for WMS servers with symbology which takes more space than the data extent), check this option to avoid cropping raster layers to their reported extents, resulting in truncated symbols on the borders of these layers.
- ☐ *Ignore axis orientation (WMS 1.3/WMTS)*
- ☐ *Invert axis orientation*
- ☐ *Smooth pixmap transformation*

4. Press *OK*

Once the new WMS/WMTS server connection has been created, it will be preserved for future QGIS sessions. Note that it is also possible to *Load* the connection parameters from a `.XML` file or *Save* them to a `.XML` file.

If you need to set up a proxy server to be able to receive WMS services from the internet, you can add your proxy server in the options. Choose *Settings* ► *Options* and click on the *Network* tab. There, you can add your proxy settings and enable them by setting ☑ *Use proxy for web access*. Make sure that you select the correct proxy type from the *Proxy type* [...▼] drop-down menu.

19.1.4 Loading WMS/WMTS Layers

Once you have successfully filled in your parameters, you can use the *Connect* button to retrieve the capabilities of the selected server. This includes the image encoding, layers, layer styles and projections. Since this is a network operation, the speed of the response depends on the quality of your network connection to the WMS server. While downloading data from the WMS server, the download progress is visualized in the lower left corner of the main QGIS dialog.

Your screen should now look a bit like Fig. 19.3, which shows the response provided by a WMS server.

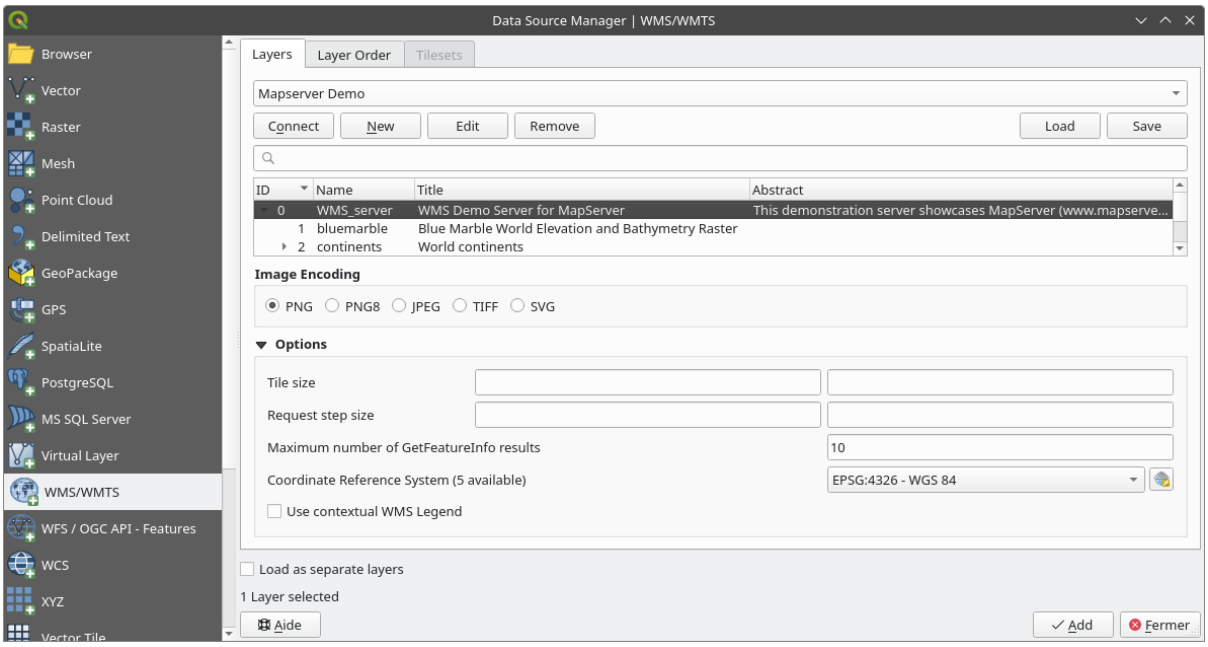


Fig. 19.3: Dialog for adding a WMS server, with filter on available layers

The upper part of the *Layers* tab of the dialog shows a tree structure that can include layer groups embedding layers with their associated image style(s) served by the server. Each item can be identified by:

- an *ID*
- a *Name*
- a *Title*
- and an *Abstract*.

The list can be filtered using the widget in the top right corner.

Image Encoding

The *Image encoding* section lists the formats that are supported by both the client and server. Choose one depending on your image accuracy requirements.

> **Tip**
>
> **Image Encoding**
>
> You will typically find that a WMS server offers you the choice of JPEG or PNG image encoding. JPEG is a lossy compression format, whereas PNG faithfully reproduces the raw raster data.
>
> Use JPEG if you expect the WMS data to be photographic in nature and/or you don't mind some loss in picture quality. This trade-off typically reduces by five times the data transfer requirement compared with PNG.
>
> Use PNG if you want precise representations of the original data and you don't mind the increased data transfer requirements.

Options

The Options area of the dialog provides means to configure the WMS requests. You can define:

- *Tile size* if you want to set tile sizes (e.g., 256x256) to split up the WMS request into multiple requests.
- *Request step size*: if you want to reduce the effect of cut labels at tile borders, increasing the step size creates larger requests, fewer tiles and fewer borders. The default value is 2000.
- The *Maximum number of GetFeatureInfo results*: specifies the maximum number of results returned by a GetFeatureInfo request, for the layer(s) being loaded. Allows to override the *connection's default value* for specific layers.
- Each WMS layer can be presented in multiple CRSs, depending on the capability of the WMS server. If you select a WMS from the list, a field with the default projection provided by the web server appears. Press the Select CRS widget to replace the default projection of the WMS with another CRS supported by the WMS server.

  A dialog similar to the one shown in Fig. 6.4 will appear. The main difference with the WMS version of the dialog is that only those CRSs supported by the WMS server will be shown.
- Finally you can activate *Use contextual WMS Legend* if the WMS Server supports this feature. Then only the relevant legend for your current map view extent will be shown and thus will not include legend items for items you can't see in the current map.

You can select several layers at once, but only one image style per layer. When several layers are selected, they will be combined at the WMS server and transmitted to QGIS in one go, as a single layer. The default name is a slash (/) separated list of their original title. You can however opt to *Load as separate layers*.

Layer Order

The *Layer Order* tab lists the selected layers available from the current connected WMS server.

WMS layers rendered by a server are overlaid in the order listed in the *Layers* tab, from top to bottom of the list. If you want to change the overlay order, you can use the *Up* and *Down* buttons of the *Layer Order* tab.

Transparency

The *Global transparency* setting from the *Layer Properties* is hard coded to be always on, where available.

19.1.5 Tilesets

When using WMTS (Cached WMS) services you are able to browse through the *Tilesets* tab given by the server. Additional information like tile size, formats and supported CRS are listed in this table.

```
# example of WMTS service

https://opencache.statkart.no/gatekeeper/gk/gk.open_wmts?service=WMTS&
↪request=GetCapabilities
```

Selecting a layer to load, it is also possible to apply an *Interpretation method*, converting it into a raster layer of single band float type, ready for styling using QGIS usual *raster renderers*.

In combination with this feature, you can use the tile scale slider by selecting *View* ► *Panels* (or *Settings* ► *Panels*), then choosing *Tile Scale Panel*. This gives you the available scales from the tile server with a nice slider docked in.

19.1.6 Using the Identify Tool

Once you have added a WMS server, and if any layer from a WMS server is queryable, you can then use the Identify tool to select a pixel on the map canvas. A query is made to the WMS server for each selection made. The results of the query are returned in plain text. The formatting of this text is dependent on the particular WMS server used.

Format selection

If multiple output formats are supported by the server, a combo box with supported formats is automatically added to the identify results dialog and the selected format may be stored in the project for the layer.

GML format support

The Identify tool supports WMS server response (GetFeatureInfo) in GML format (it is called Feature in the QGIS GUI in this context). If "Feature" format is supported by the server and selected, results of the Identify tool are vector features, as from a regular vector layer. When a single feature is selected in the tree, it is highlighted in the map and it can be copied to the clipboard and pasted to another vector layer. See the example setup of the UMN Mapserver below to support GetFeatureInfo in GML format.

```
# in layer METADATA add which fields should be included and define geometry␣
↪(example):

"gml_include_items"   "all"
"ows_geometries"      "mygeom"
"ows_mygeom_type"     "polygon"

# Then there are two possibilities/formats available, see a) and b):

# a) basic (output is generated by Mapserver and does not contain XSD)
# in WEB METADATA define formats (example):
"wms_getfeatureinfo_formatlist" "application/vnd.ogc.gml,text/html"

# b) using OGR (output is generated by OGR, it is sent as multipart and contains␣
↪XSD)
# in MAP define OUTPUTFORMAT (example):
OUTPUTFORMAT
    NAME "OGRGML"
```

(continues on next page)

(continued from previous page)

```
    MIMETYPE "ogr/gml"
    DRIVER "OGR/GML"
    FORMATOPTION "FORM=multipart"
END

# in WEB METADATA define formats (example):
"wms_getfeatureinfo_formatlist" "OGRGML,text/html"
```

19.1.7 Viewing Properties

Once you have added a WMS server, you can view its properties by right-clicking on it in the legend and selecting *Properties*. The WMS/WMTS layer properties is much like the raster layer properties so you will find detailed description at *Raster Properties Dialog*. However, there are some differences, which will be explained below.

Information properties

Metadata Tab

The tab *Metadata* displays a wealth of information about the WMS server, generally collected from the capabilities statement returned from that server. Many definitions can be gleaned by reading the WMS standards (see OPENGEOSPATIAL-CONSORTIUM in *Literature and Web References*), but here are a few handy definitions:

- **Server Properties**
  - **WMS Version** — The WMS version supported by the server.
  - **Image Formats** — The list of MIME-types the server can respond with when drawing the map. QGIS supports whatever formats the underlying Qt libraries were built with, which is typically at least `image/png` and `image/jpeg`.
  - **Identity Formats** — The list of MIME-types the server can respond with when you use the Identify tool. Currently, QGIS supports the `text-plain` type.
- **Layer Properties**
  - **Selected** — Whether or not this layer was selected when its server was added to this project.
  - **Visible** — Whether or not this layer is selected as visible in the legend (not yet used in this version of QGIS).
  - **Can Identify** — Whether or not this layer will return any results when the Identify tool is used on it.
  - **Can be Transparent** — Whether or not this layer can be rendered with transparency. This version of QGIS will always use transparency if this is `Yes` and the image encoding supports transparency.
  - **Can Zoom In** — Whether or not this layer can be zoomed in by the server. This version of QGIS assumes all WMS layers have this set to `Yes`. Deficient layers may be rendered strangely.
  - **Cascade Count** — WMS servers can act as a proxy to other WMS servers to get the raster data for a layer. This entry shows how many times the request for this layer is forwarded to peer WMS servers for a result.
  - **Fixed Width, Fixed Height** — Whether or not this layer has fixed source pixel dimensions. This version of QGIS assumes all WMS layers have this set to nothing. Deficient layers may be rendered strangely.
  - **WGS 84 Bounding Box** — The bounding box of the layer, in WGS 84 coordinates. Some WMS servers do not set this correctly (e.g., UTM coordinates are used instead). If this is the case, then the initial view of this layer may be rendered with a very 'zoomed-out' appearance by QGIS. The WMS webmaster should be informed of this error, which they may know as the WMS XML elements `LatLonBoundingBox`, `EX_GeographicBoundingBox` or the CRS:84 `BoundingBox`.
  - **Available in CRS** — The projections that this layer can be rendered in by the WMS server. These are listed in the WMS-native format.

– **Available in style** — The image styles that this layer can be rendered in by the WMS server.

Temporal properties

Raster *temporal properties* (namely *Dynamic Temporal Control*) can be set for WMS and WMTS layers. By default, when a time-dimension enabled WMS or WMTS layer is added to the project, it is indicated in the *Layers* panel with the Temporal Layer icon next to it. Its *Temporal* properties default to the *Automatic* temporal mode, meaning that the layer will follow the temporal controller's current time range by default.

You can then opt to show a specific static time value for the layer by unchecking *Dynamic Temporal Control* and picking an option under *Static WMS-T Temporal Range*:

- *Server default*
- *Predefined date* with a server exposing data for non-contiguous temporal ranges or *Predefined range* with a server exposing a range of available dates. A *Start date* and *End date* are necessary in the latter case. Their expected formatting can be deduced from the reference time option (see below). depending on whether the provider has data for contiguous period or not
- *Follow project's temporal range* as defined in the project's properties dialog

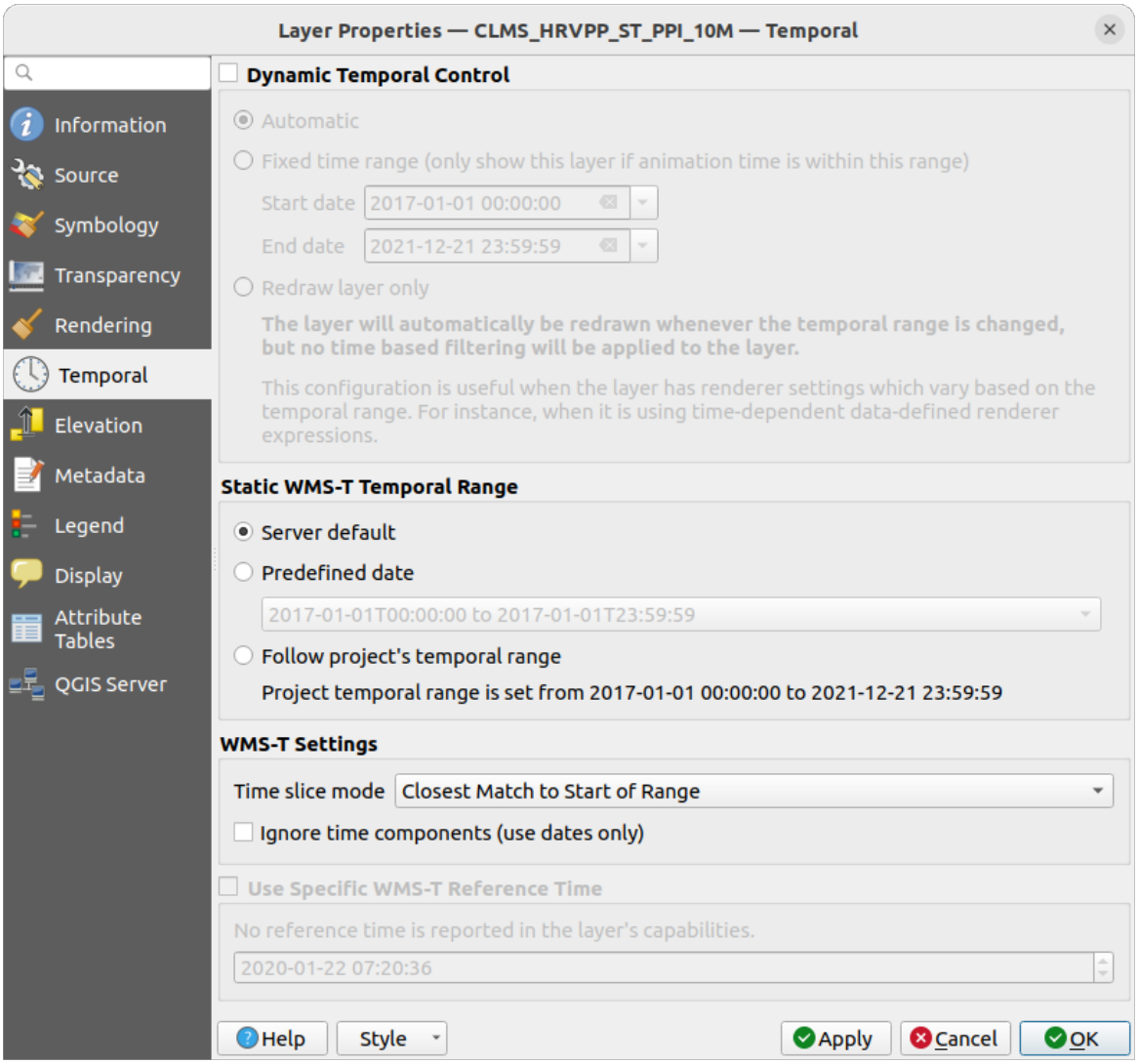


Fig. 19.4: Temporal properties of a WMTS layer

Whatever temporal data control is in use, there are some *WMS-T Settings* to help display the correct temporal data:

- *Time slice mode* which can be:
  - *Use whole temporal range*
  - *Match to start of range*
  - *Match to end of range*
  - *Closest match to start of range*
  - *Closest match to end of range*
- *Ignore time components (use dates only)*: If checked, the time component of temporal queries will be discarded and only the date component will be used in server requests.

You can also  *Use Specific WMS-T Reference Time* picked from times reported in the layer's capabilities. Convenient for servers which expose a non-contiguous set of date time instances (instead of a range of dates).

QGIS Server properties

In addition to the raster layer *properties*, publishing a WMS/WMTS layer with QGIS Server will display following options:

- *WMS Print layer*: Allows to set alternative WMS layer that should be used for printing (in the GetProjectSettings reply). Convenient for WMTS layers that are generally not suitable for printing.
-  *Publish WMS/WMTS data source uri*: Allows a web client to fetch the WMS/WMTS data directly
-  *Advertise as background layer*

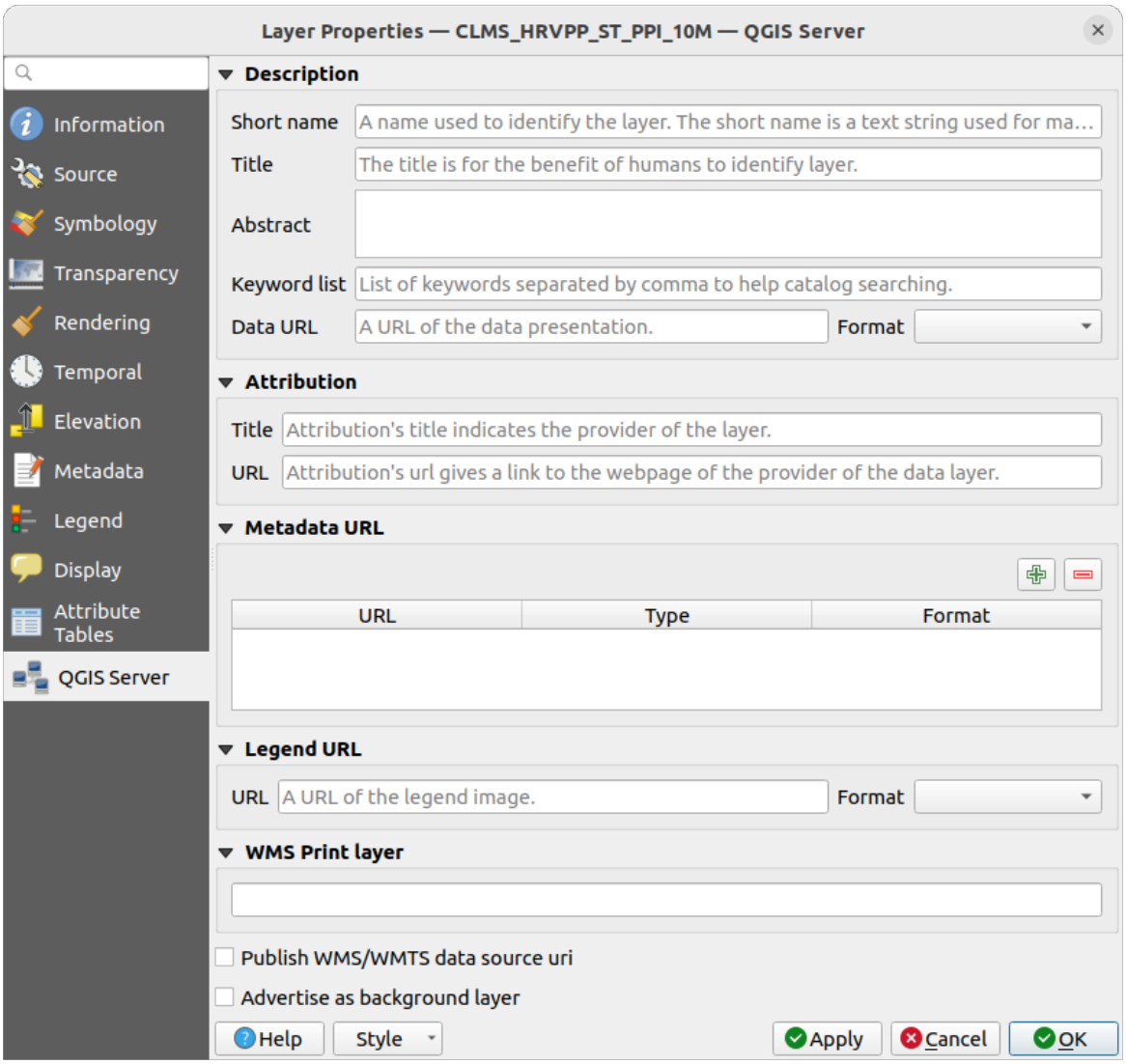


Fig. 19.5: QGIS Server properties of a WMS/WMTS layer

19.1.8 Show WMS legend graphic in table of contents and layout

The QGIS WMS data provider is able to display a legend graphic in the table of contents' layer list and in the print layout. The WMS legend will be shown only if the WMS server has GetLegendGraphic capability and the layer has getCapability url specified, so you additionally have to select a styling for the layer.

If a legendGraphic is available, it is shown below the layer. It is little and you have to click on it to open it in real dimension (due to QgsLegendInterface architectural limitation). Clicking on the layer's legend will open a frame with the legend at full resolution.

In the print layout, the legend will be integrated at it's original (downloaded) dimension. Resolution of the legend graphic can be set in the item properties under *Legend* ► *WMS LegendGraphic* to match your printing requirements.

The legend will display contextual information based on your current scale. The WMS legend will be shown only if the WMS server has GetLegendGraphic capability and the layer has getCapability url specified, so you have to select a styling.

19.2 WCS Client

A Web Coverage Service (WCS) provides access to raster data in forms that are useful for client-side rendering, as input into scientific models, and for other clients. The WCS may be compared to the WFS and the WMS. As WMS and WFS service instances, a WCS allows clients to choose portions of a server's information holdings based on spatial constraints and other query criteria.

QGIS has a native WCS provider and supports both version 1.0 and 1.1 (which are significantly different), but currently it prefers 1.0, because 1.1 has many issues (i.e., each server implements it in a different way with various particularities).

The native WCS provider handles all network requests and uses all standard QGIS network settings (especially proxy). It is also possible to select cache mode ('always cache', 'prefer cache', 'prefer network', 'always network'), and the provider also supports selection of time position, if temporal domain is offered by the server.

Loading a WCS Layer

To be able to load a WCS Layer, first create a connection to the WCS server:

1. Open the *Data Source Manager* dialog by pressing the Open Data Source Manager button
2. Enable the *WCS* tab
3. Click on *New…* to open the *Create a New WCS Connection* dialog

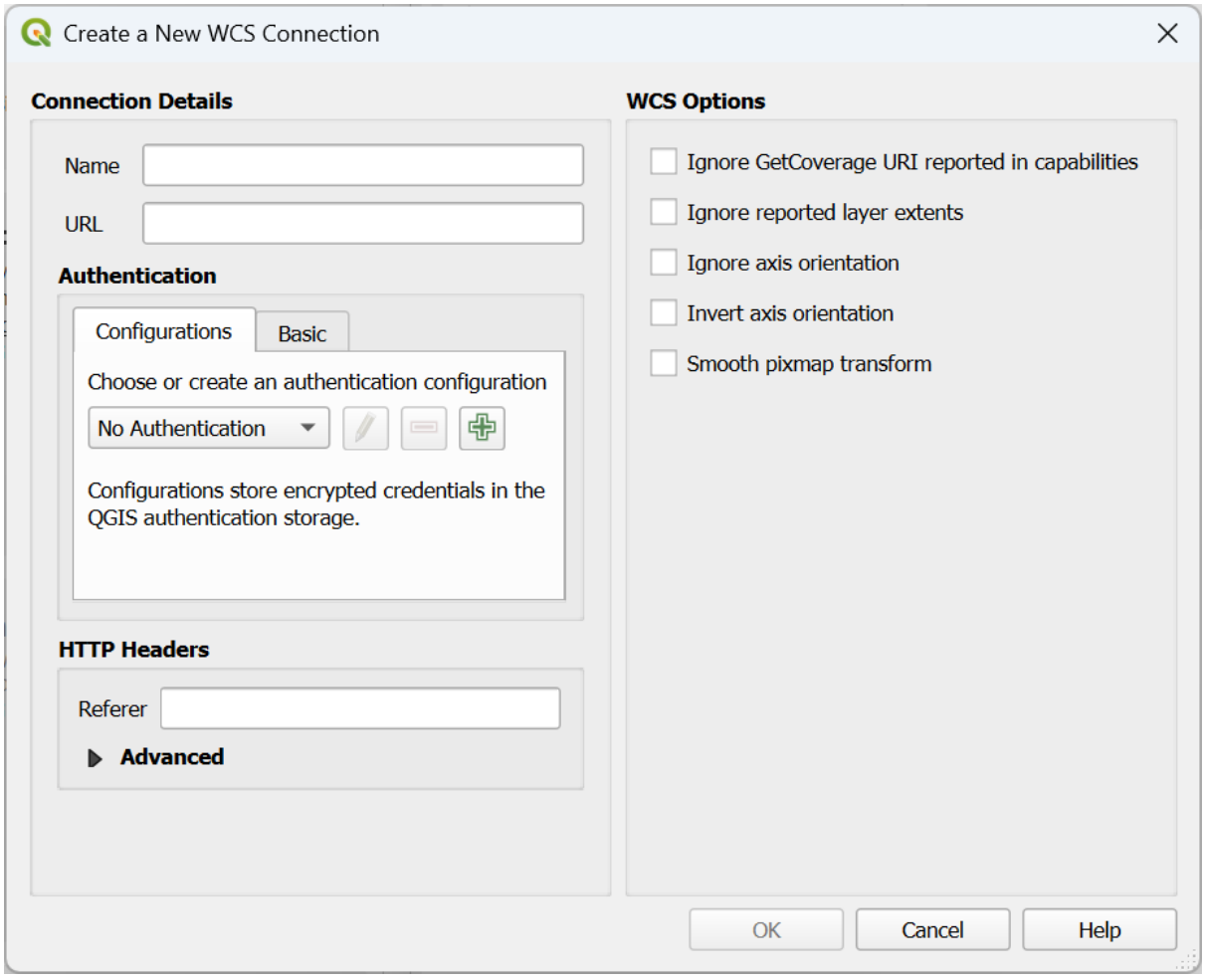


Fig. 19.6: Creating a connection to a WCS server

- *Name*: A name for the connection. This name will be used in the Server Connections drop-down box so that you can distinguish it from other WCS servers.
- *URL*: URL of the server providing the data. This must be a resolvable host name – the same format as you would use to open a telnet connection or ping a host, i.e. the base URL only. For example, you shouldn't have fragments such as `request=GetCapabilities` or `version=1.0.0` in your URL.
- *Authentication* (optional): using a *stored configuration* or a basic authentication with *Username* and *Password*.

> **Warning**
>
> Entering **username** and **password** in the *Authentication* tab will keep unprotected credentials in the connection configuration. Those **credentials will be visible** if, for instance, you shared the project file with someone. Therefore, it's advisable to save your credentials in an *Authentication configuration* instead (*Configurations* tab). See *Authentication System* for more details.

- HTTP *Referer*
- ☐ *Ignore GetCoverage URI reported in capabilities*: if checked, use given URI from the *URL* field above.
- ☐ *Ignore reported layer extents*: because the extent reported by raster layers may be smaller than the actual area which can be rendered (notably for WCS servers with symbology which takes more space than the data extent), check this option to avoid cropping raster layers to their reported extents, resulting in truncated symbols on the borders of these layers.
- ☐ *Ignore axis orientation*
- ☐ *Invert axis orientation*
- ☐ *Smooth pixmap transformation*

4. Press *OK* to create the connection.

Note that any proxy settings you may have set in your preferences are also recognized. Also note that it is possible to *Load* the connection parameters from a `.XML` file or *Save* them to a `.XML` file.

Now we are ready to load WCS layers from the above connection.

19.3 WFS and WFS-T Client

In QGIS, a WFS layer behaves pretty much like any other vector layer. You can identify and select features, and view the attribute table. QGIS supports WFS 1.0.0, 1.1.0, 2.0 and OGC API - Features (OAPIF), including editing (through WFS-T). QGIS also supports background download and progressive rendering, on-disk caching of downloaded features and version autodetection.

Layers of servers implementing OGC API - Features - Part 4: Create, Replace, Update and Delete can be turned into editing mode to allow creating, modifying and deleting features. Note that each created/modified/deleted feature requires a dedicated network request, so performance might suffer in case of simultaneous modification of hundreds or more features at a time.

In general, adding a WFS layer is very similar to the procedure used with WMS. There are no default servers defined, so you have to add your own. You can find WFS servers by using the *MetaSearch plugin* or your favourite web search engine. There are a number of lists with public URLs, some of them maintained and some not.

Loading a WFS Layer

As an example, we use the Gateway Geomatics WFS server and display a layer.

```
https://demo.gatewaygeomatics.com/cgi-bin/wfs_gateway?REQUEST=GetCapabilities&
↪VERSION=1.0.0&SERVICE=WFS
```

To be able to load a WFS Layer, first create a connection to the WFS server:

1. Open the *Data Source Manager* dialog by pressing the Open Data Source Manager button
2. Enable the *WFS / OGC API - Features* tab
3. Click on *New…* to open the *Create a New WFS Connection* dialog
4. In the *Connection Details* group, enter `Gateway Geomatics` as name
5. Enter the URL (see above)
6. Optionally, enter the *Authentication* settings

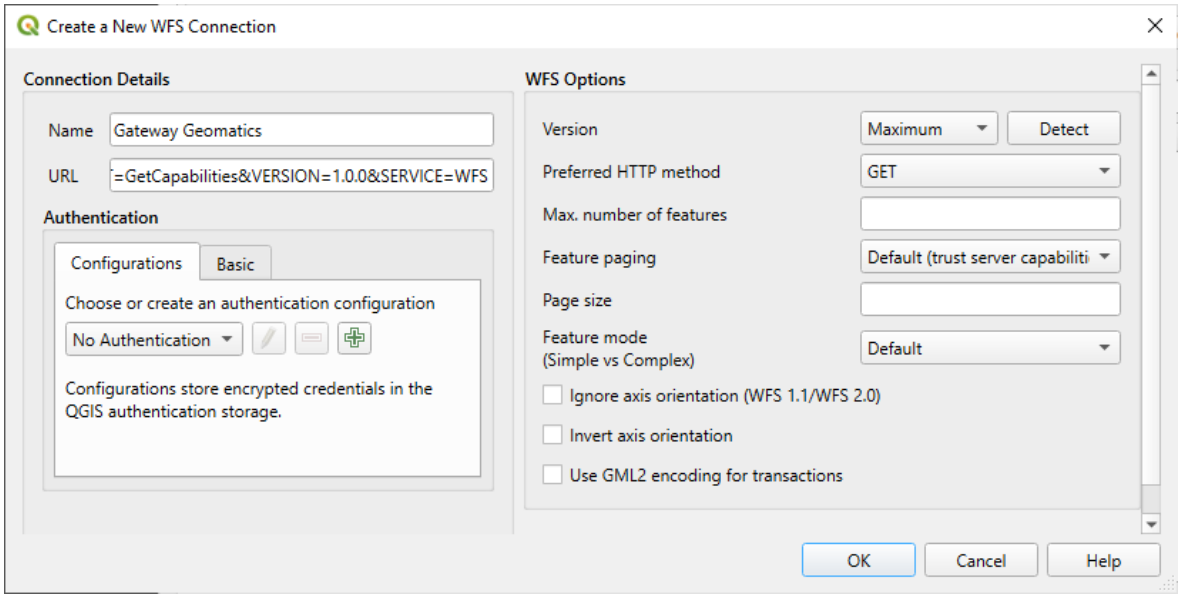


Fig. 19.7: Creating a connection to a WFS server

> **Note**
>
> In case of an OGC API - Features (OAPIF), the URL to provide should be the landing page, i.e., the main page from which it is possible to navigate to all the available service endpoints.

7. In the *WFS Options* group, you can:
   - Indicate the WFS version of the server. If unknown, press the *Detect* button to automatically retrieve it.
   - Select the *Preferred HTTP method* to use for requests. The default is *GET*, but you can also select *POST*.
   - Define the *Maximum number of features* retrieved in a single GetFetFeature request. If empty, no limit is set.
   - *Feature mode (simple vs complex)*: helps determine how to handle servers serving data with simple or complex GML schemas. Available options are:
     - *Default*: if the server supports editing capabilities, this is the same as using *Simple Features*. If not, it fallbacks to using *Complex Features* mode.
     - *Simple Features*: returns data as simple features only
     - *Complex Features*: uses the *OGR GMLAS <https://gdal.org/en/stable/drivers/vector/gmlas.html>*, potentially returning complex features, depending on the data schema, where nested data structures are exposed by QGIS as JSON.

- Depending on the WFS version, indicate whether to:
  - Enable *Feature paging* and specify the maximum number of features to retrieve with *Page size*. If no limit is defined, then the server default is applied.
  - Force to *Ignore axis orientation (WFS 1.1/WFS 2.0)*
  - *Invert axis orientation*.
  - *Use GML2 encoding for transactions*.

> **Warning**
>
> Entering **username** and **password** in the *Authentication* tab will keep unprotected credentials in the connection configuration. Those **credentials will be visible** if, for instance, you shared the project file with someone. Therefore, it's advisable to save your credentials in an *Authentication configuration* instead (*Configurations* tab). See *Authentication System* for more details.

8. Press *OK* to create the connection.

Note that any proxy settings you may have set in your preferences are also recognized. Also note that it is possible to *Load* the connection parameters from a `.XML` file or *Save* them to a `.XML` file.

Now we are ready to load WFS layers from the above connection.

1. Choose 'Gateway Geomatics' from the *Server Connections* drop-down list.
2. Click *Connect*
3. Select the *Parks* layer in the list
4. You can also choose whether to:
   - *Only request features overlapping the view extent*
   - *Change…* the layer's CRS to any other supported by the service
   - or build a query to specify particular features to retrieve from the service: double-click on the row of the layer to open the *SQL Query Composer* dialog. That dialog provides widgets to write an advanced SQL query relying on available tables and columns of the service, with sorting and filtering and a bunch of SQL functions, spatial predicates and operators.

     The query you build will appear after validation in the *SQL* column within the *WFS / OGC API - Features* table, and the filtered layer will display the icon next to it in the *Layers* panel. It is thus possibe to adjust the query at any moment.
5. Click *Add* to add the layer to the map.

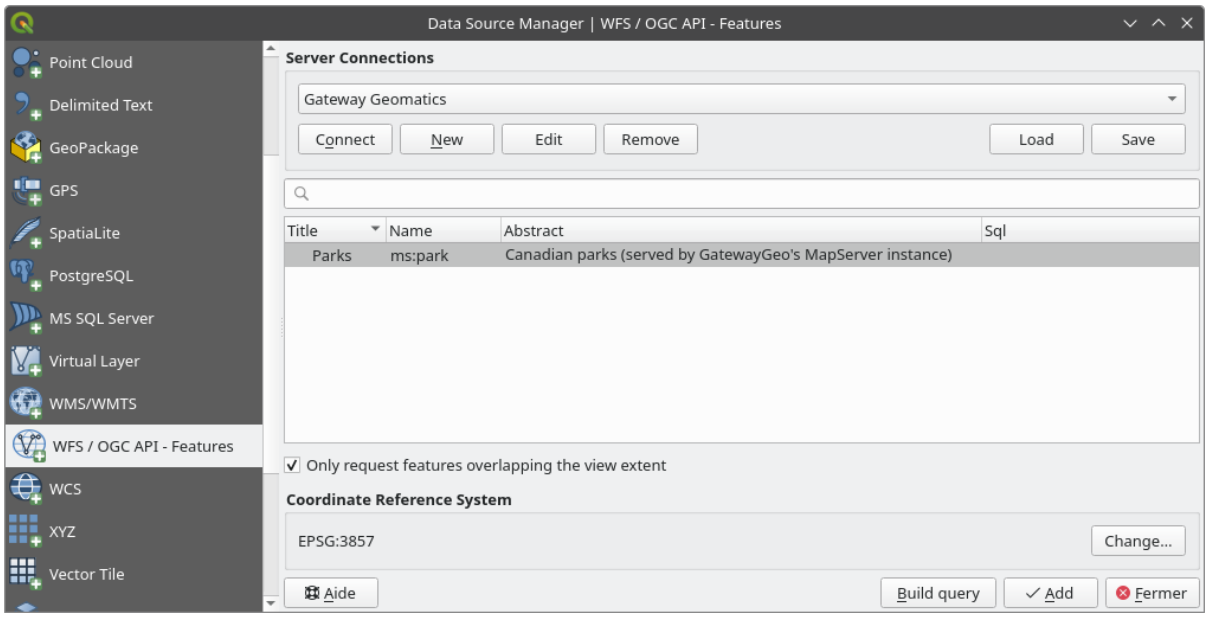


Fig. 19.8: Adding a WFS layer

You'll notice the download progress is visualized in the lower left of the QGIS main window. Once the layer is loaded, you can identify and select a couple of features and view the attribute table.

19.4 SensorThings

QGIS supports connections to OGC SensorThings API, a standard providing an open and unified framework to interconnect IoT sensing devices, data, and applications over the Web. It is an open standard addressing the syntactic and semantic interoperability of the Internet of Things. It is based on the Observations and Measurements data model, a standardized model for observations, and for features involved in sampling when making observations.

19.4.1 Setting connection

To add SensorThings data to QGIS use the *SensorThings* tab in the *Data Source Manager* dialog.

To establish a new connection, press *New* (or *New SensorThings Connection* from the Browser panel) and provide *Name* and *URL*. Advanced options, such as *authentication* and a *Referer*, can also be configured.

Press *OK* to establish the connection. Then you will be able to:

- *Edit* the SensorThings connection settings
- *Remove* the SensorThings connection

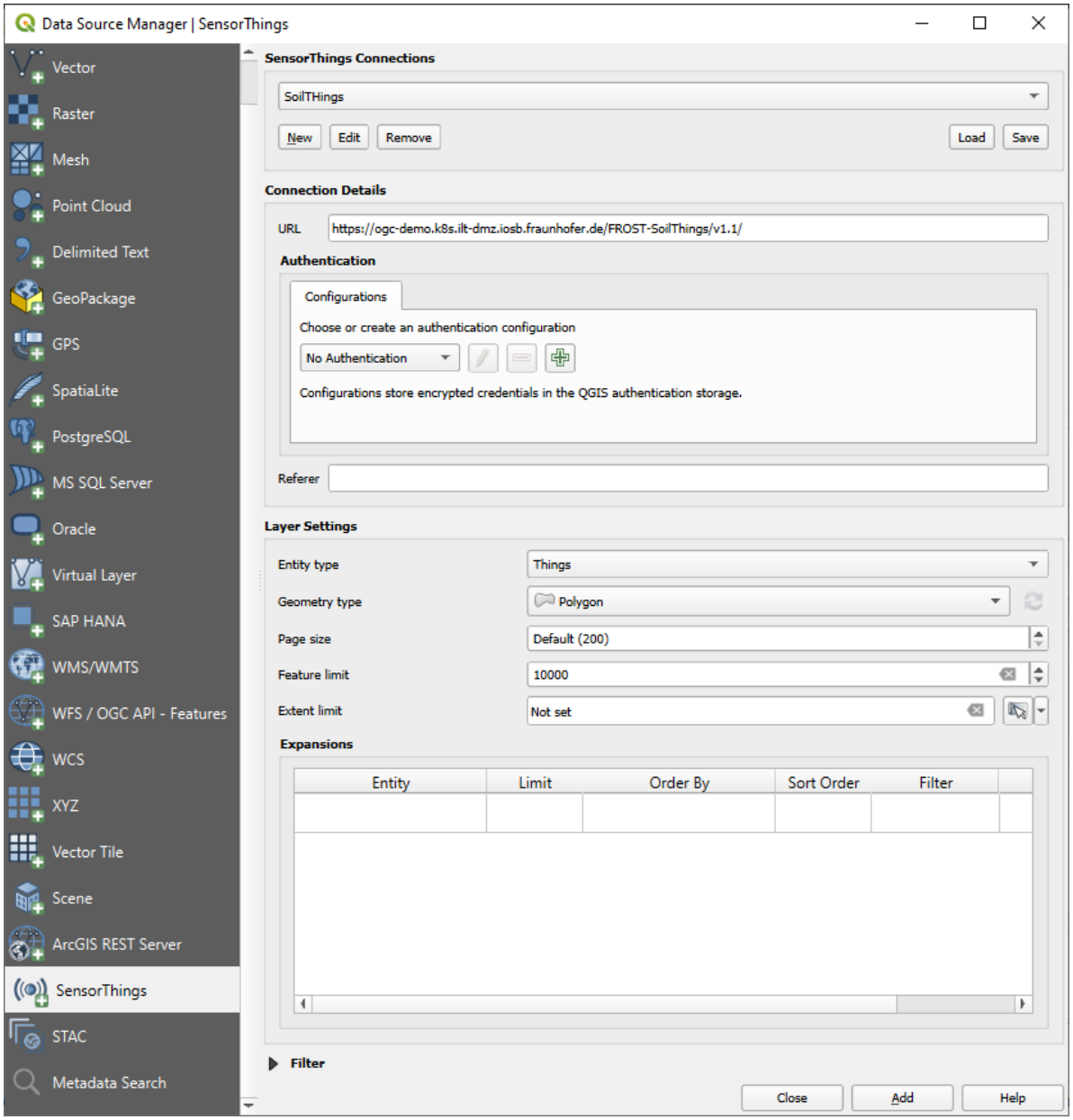


Fig. 19.9: SensorThings Connection dialog

Configurations can be saved to an `.XML` file (*Save*) through the *SensorThings* entry in *Data Source Manager* dialog or its contextual menu in the *Browser* panel (*Save Connections*). Likewise, configurations can be added from a file (*Load*).

19.4.2 Loading SensorThings data

Relations between layers (so-called entities) stored in a SensorThings dataset are expressed in the diagram below.

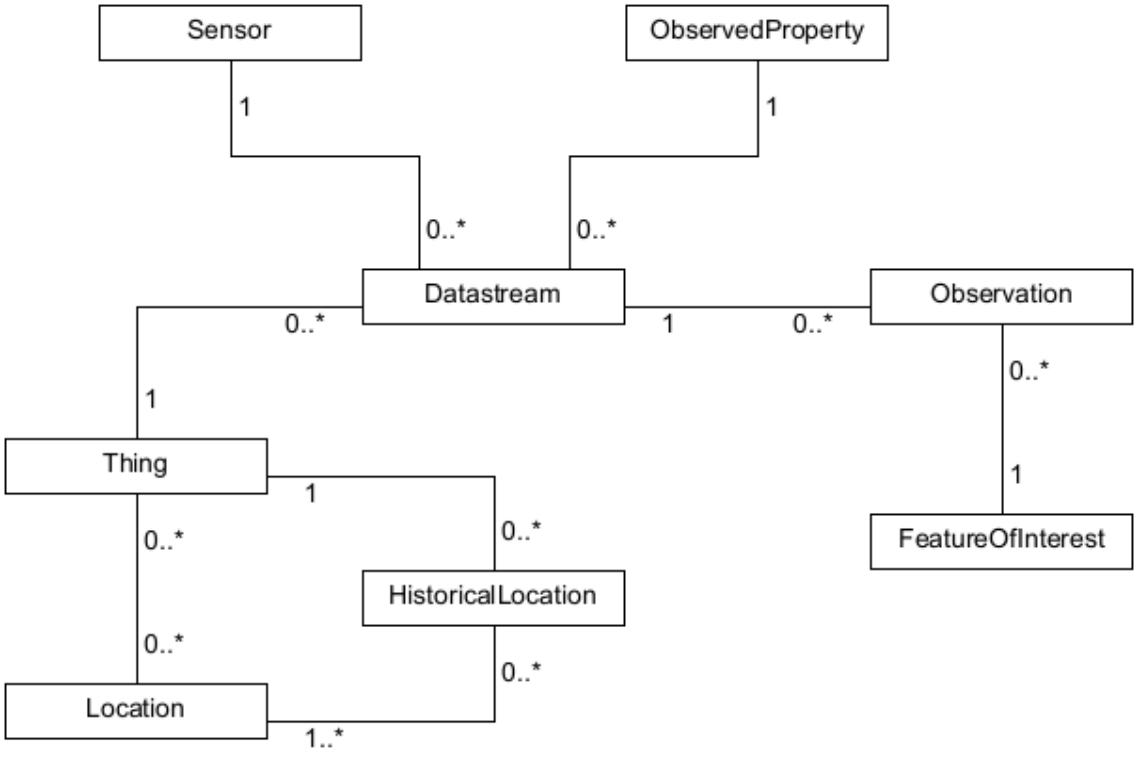


Fig. 19.10: Data model Observations and Measurements (Source: Sensor Web Tutorial by SIST network)

Any type of entity can be loaded in QGIS, but not all are spatial data. To load an entity, there are *Layer Settings* that can be configured:

- *Entity Type*: the entity to load from the data model as layer in QGIS
- *Geometry Type*: the geometry type of the selected entity to load. Press Check available types to limit the list to the actually supported geometry types.
- *Page Size*
- *Feature Limit* sets a maximum number of features to request from the service
- *Extent Limit* sets a maximum extent limit for the layer, so that only features within the extent are requested
- *Expansions*: The data model of SensorThings provides a mechanism of expansion of the results to related entities, similar to how tables are joined together in a relational database. Using this approach, you can expand the selected layer to include data from other items. This will flatten the relationship, creating as many parent features as children, and additional properties are added as columns in the attribute table.

Use *Filter* to build a query to filter the data, using SensorThings filter syntax.

> **Note**
>
> The above settings and filtering options are also available for update in the layer properties dialog, *Source* tab, once loaded in QGIS.

Press *Add* to load the selected entity type as layer in QGIS.

19.4.3 Working with a vector layer from SensorThings

A SensorThings layer is loaded in QGIS as a vector layer. As such, it displays the same tabs in the *layer properties* and allows same feature interactions using the selection or identify tools. There are however some specificities you should consider while working with SensorThings data.

Because of the data model, the result property of a SensorThings Observation is a string field. In case you want to use its numerical representation in for example a graduated style, use an expression to convert the value to real and try() in case this fails (e.g., `try( to_real("Observation_result"), Null)`).

In case you want to create a chart of the observations at one or more locations, you can install the QGIS plugin *Data Plotly*.

1. Now select the observations at a point location in the map view.
2. Open the plotly panel and activate the *Use only selected features* checkbox.
3. Select on the x-column a date-time property and on the y-column the *Observation_result*. This will plot the observations at that location over time.
4. Verify to filter by a single Observed Property.
5. Notice that the chart changes as soon as you select other locations on the map.

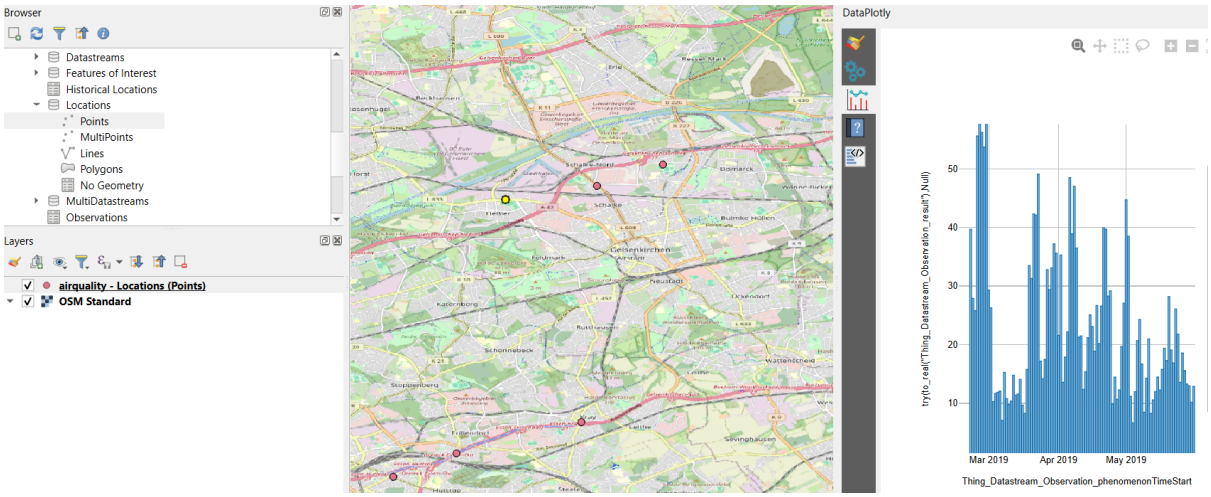


Fig. 19.11: Use Data plotly to plot the air quality observations at a location

19.5 STAC (SpatioTemporal Asset Catalogs)

STAC (SpatioTemporal Asset Catalog) is a specification for describing geospatial data in a consistent and accessible way. It defines a standard structure for organizing and indexing spatial-temporal assets such as satellite imagery, drone photos, or sensor data so that you can search, preview, and use these datasets across different platforms and tools.

STAC enables QGIS users to connect to and browse geospatial datasets, either from static catalogs or dynamic APIs, and download or stream assets such as imagery or other spatial data.

More information at https://stacspec.org/en.

Key components of STAC in QGIS include:

- **STAC Items**: The basic unit of a STAC catalog. Each item represents one geospatial asset or a group of related assets (e.g. image bands) at a specific time and location. It includes metadata such as geometry, datetime, links to assets (e.g., TIFFs, JSON, COG), and properties.
- **STAC Collections**: A grouping of items that share common characteristics and metadata (e.g. satellite mission). Collections may define spatial and temporal extents, licensing, and keywords.
- **STAC Catalogs**: A hierarchical container that organizes items and/or collections. Catalogs allow navigation of STAC datasets but do not necessarily include search capabilities.
- **STAC API**: An implementation of the STAC specification that allows querying and filtering of STAC items using spatial and temporal based filters, as well as searching within specific collections. STAC APIs follow the OGC API - Features pattern and support dynamic access to datasets.

There is an important distinction between static STAC catalogs and STAC API endpoints:

- **Static STAC Catalogs**: are collections of JSON files without search capabilities. They can be browsed via the Browser panel.
- **STAC API**: provide search capabilities and can be accessed in QGIS through both the Browser panel and the Data Source Manager.

19.5.1 Setting connection

STAC connections can be added in QGIS using either the *Browser panel* or the *Data Source Manager*:

- **Browser Panel:** In the *Browser panel*, right-click on the *STAC* entry and select *New STAC Connection…*. In the dialog that appears, enter a *Name* for the connection, the *URL* of the STAC catalog and optionally fill in *Authentication* credentials and a *Referer*. Then click *OK*.

  Use this method for browsing static STAC catalogs that do not support search or filtering.

- **Data Source Manager:** For STAC APIs you can use *Data Source Manager* dialog.

  Open the *Data Source Manager*, choose the *STAC* tab and click the *New* button. Fill in the *Name* and *URL* fields, and (optional) the *Authentication* credentials and a *Referer*. Press *OK* and then *Connect* to establish the connection, after that you will be able to:

  - *Edit* the STAC connection settings
  - *Remove* the STAC connection

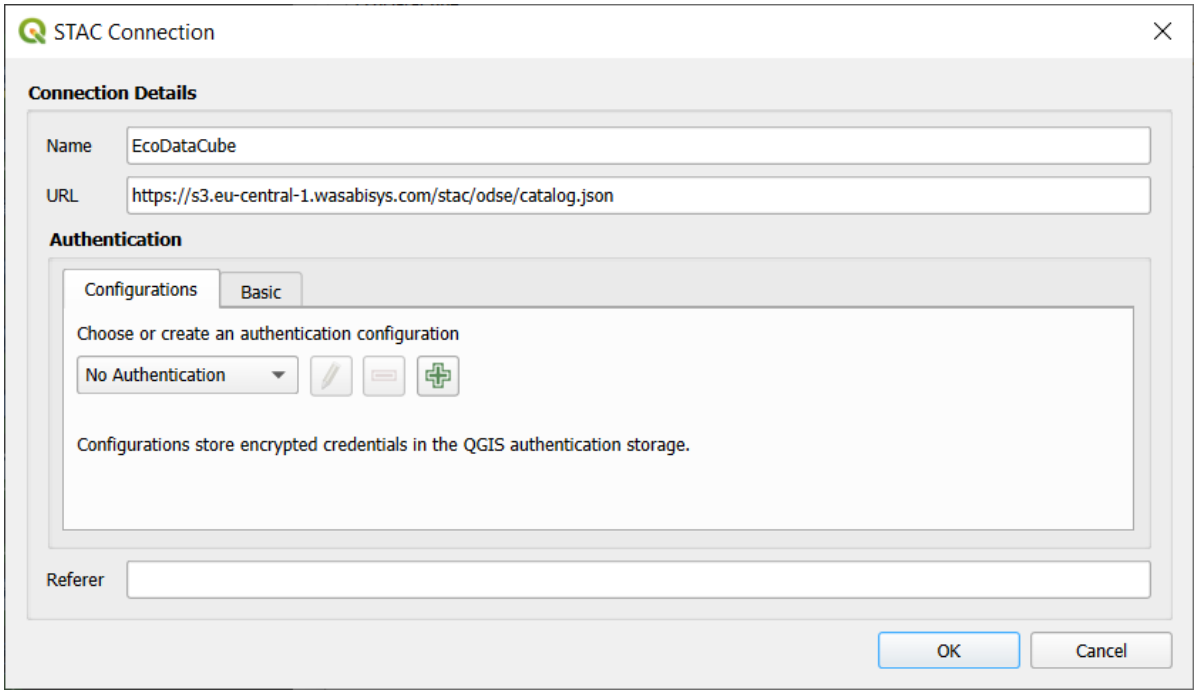


Fig. 19.12: Creating a connection to a STAC server

19.5.2 Browsing STAC Catalogs

Static STAC catalogs are displayed as hierarchical structures within the *Browser panel*. Once connected, the STAC catalog appears under *STAC* in the *Browser* panel. You can expand the catalog node to see its *Collections*. Expanding a collection reveals the individual *Items* it contains.

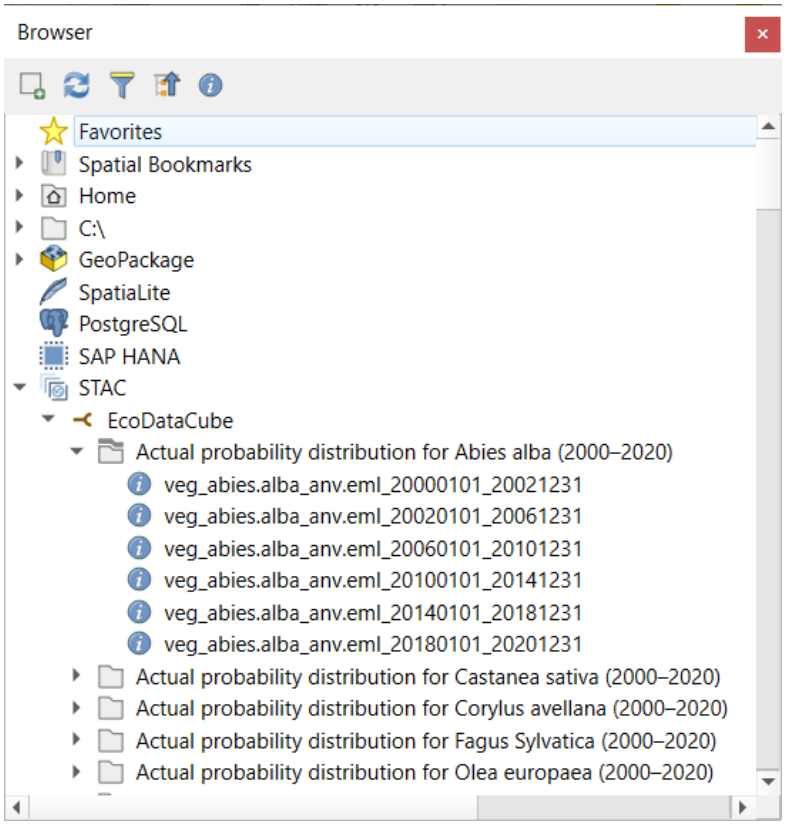


Fig. 19.13: STAC connection expanded in the Browser, showing Collections and Items

Right-click any STAC *Item* and choose *Details…* to view its metadata. The details panel shows the item's JSON content and a map of its coverage. If an item's asset is a cloud-optimized format (e.g. a COG), you can add it directly to the map canvas. Otherwise, *Download Assets…* to save it locally before use.

19.5.3 Filtering and Searching STAC Items

For STAC API endpoints, QGIS supports spatial and temporal filtering, as well as limiting searches to specific collections.

To search for items:

1. Open *Data Source Manager* and go to the *STAC*
2. Select the STAC connection you created earlier and click *Filters…*

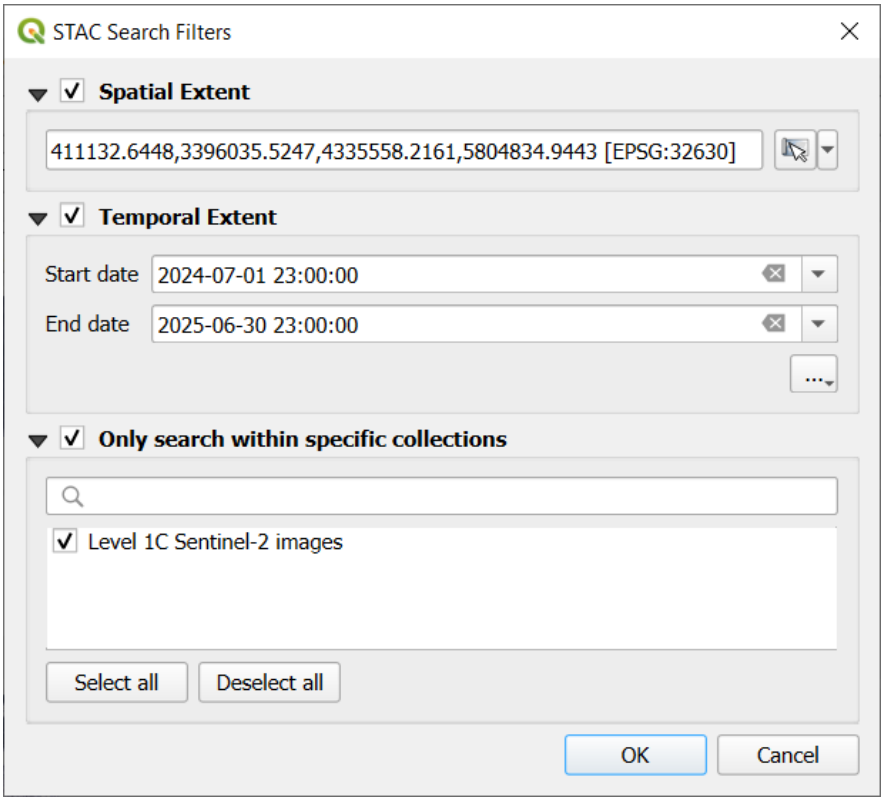


Fig. 19.14: Define spatial and temporal filters for a STAC catalog search

3. In the filter dialog, you can:
   - Enable *Spatial Extent* to restrict results to a bounding box or drawn area.
   - Enable *Temporal Extent* and define a date or range.
   - Limit to specific collections using *Only search within specific collections*
4. After applying filters, results will be listed in the main *Data Source Manager* panel.
5. Optionally check *Show Footprints* to display the footprints of the results on the map.

Right-click a result item to access actions:

- *Zoom to Item*
- *Pan to Item*
- *Download Assets*
- *Details…*

If the item's asset requires download, use the *Download Assets* option.

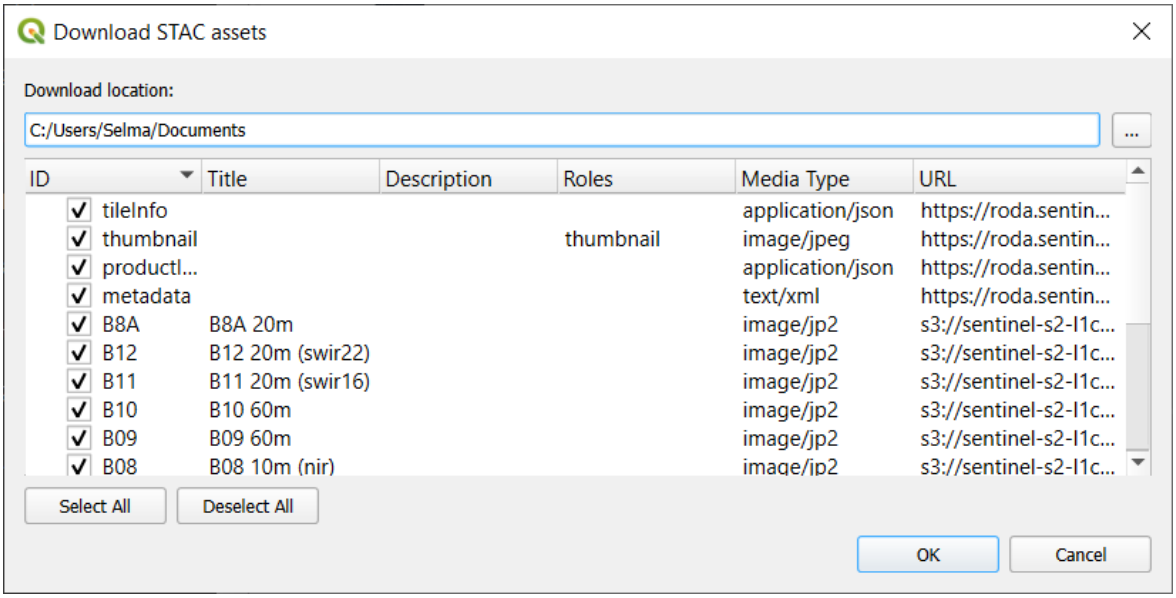


Fig. 19.15: Download STAC item assets

Downloaded assets include the main dataset and any auxiliary files such as thumbnails or style files. After download, use standard QGIS tools (e.g., *Add Raster Layer…*) to load and display the data.

CHAPTER

TWENTY

WORKING WITH GPS DATA

20.1 Introducing GNSS/GPS Data

20.1.1 What is GPS?

GPS, the **Global Positioning System**, is a satellite-based system that allows anyone with a GPS receiver to find their exact position anywhere in the world. GPS is used as an aid in navigation, for example in airplanes, in boats and by hikers. The GPS receiver uses the signals from the satellites to calculate its latitude, longitude and (sometimes) elevation. Most receivers also have the capability to store:

- locations (known as **waypoints**)
- sequences of locations that make up a planned **route**
- and a **track** log of the receiver's movement over time.

Waypoints, routes and tracks are the three basic feature types in GPS data. QGIS displays waypoints in point layers, while routes and tracks are displayed in linestring layers.

> **Note**
>
> QGIS supports also GNSS receivers. But we keep using the term GPS in this documentation.

20.1.2 Transferring or loading GPS data

Loading a GPX file

There are dozens of different file formats for storing GPS data. The format that QGIS uses is called GPX (GPS eXchange format), which is a standard interchange format that can contain any number of waypoints, routes and tracks in the same file.

To load a `GPX` file:

1. Open the *GPS* tab in the *Data Source Manager* dialog, i.e.:
   - Click the  Open Data Source Manager button on the toolbar (or press `Ctrl+L`) and enable the target tab
   - Or select *Layer* ► *Add Layer* ►  *Add GPX Layer…*
2. Use the … Browse button next to the *GPX dataset* option to select the GPX file
3. Use the check boxes to select the *Feature types* you want to load from the file. Each feature type (*Waypoints*, *Tracks* or *Routes*) will be loaded in a separate layer.

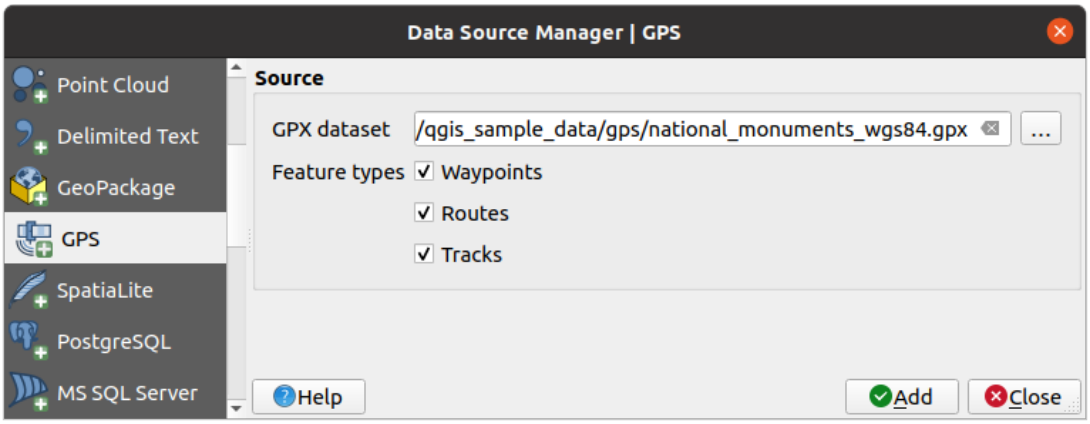


Fig. 20.1: Loading GPS Data dialog

Loading to or from a device

There are lots of different types of GPS devices and formats. Since QGIS uses GPX files, you need a way to convert other GPS file formats to GPX. QGIS can do that using the free program GPSBabel. GPSBabel can help you convert waypoints, tracks, and routes between popular GPS receivers such as Garmin or Magellan and mapping programs like Google Earth or Basecamp. Literally hundreds of GPS receivers and programs are supported. It can also transfer GPS data between your computer and a GPS device.

Under *Settings* ► *Options* ► *GPS* ► *GPSBabel*, QGIS allows you to define your own device type and set parameters of conversion that could later be used by the *Processing GPS algorithms*.

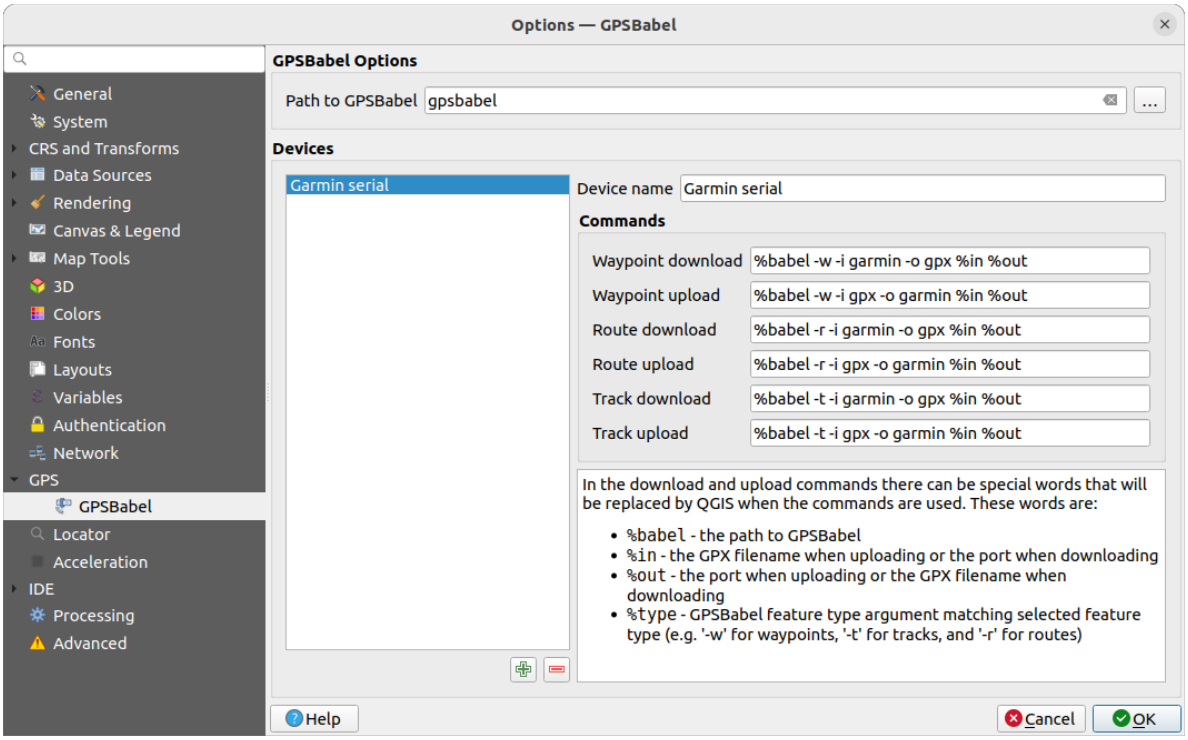


Fig. 20.2: GPS Babel settings

1. First you have to define the *Path to GPSBabel* binaries.
2. Then you may want to add your device. You can update devices list using Add new device or Remove device

button.

3. For each device:
   - you provide a *Device name*
   - you configure different *Commands* QGIS will use while interacting with it, such as:
     - *Waypoint download* from the device
     - *Waypoint upload* to the device
     - *Route download* from the device
     - *Route upload* to the device
     - *Track download* from the device
     - *Track upload* to the device

     While the commands are usually GPSBabel commands, you can also use any other command line program that can create a GPX file. QGIS will replace the keywords `%type`, `%in`, and `%out` when it runs the command.

     As an example, if you create a device type with the download command `gpsbabel %type -i garmin -o gpx %in %out` and then use it to download waypoints from port `/dev/ttyS0` to the file `output.gpx`, QGIS will replace the keywords and run the command `gpsbabel -w -i garmin -o gpx /dev/ttyS0 output.gpx`.

     Read the GPSBabel manual for the command line options that may be specific to your use case.

Once you have created a new device type, it will appear in the device lists for the GPS download and upload algorithms.

Note

GPS units allow you to store data in different coordinate systems. When downloading a GPX file (from your GPS unit or a web site) and then loading it in QGIS, be sure that the data stored in the GPX file uses WGS 84 (latitude/longitude). QGIS expects this, and it is the official GPX specification. See GPX 1.1 Schema Documentation.

20.2 Live GPS tracking

QGIS can help you do field mapping with a GPS receiver. Such a live tracking operation is done using the *GPS Toolbar*. Some *device configuration* may be needed before connecting QGIS and the GPS receiver.

20.2.1 GPS toolbar

The *GPS Toolbar* provides the main tools to control a live tracking session. It can be activated from *View* ► *Toolbars* ► *GPS*. It follows the state of the project, GPS, and current GPS track, and enables actions only when they make sense. Once a device is detected by QGIS, you will be able to interact with it:

-  Connect to GPS
-  Disconnect from GPS
-  Recenter map on GPS location: the map will immediately recenter on the current GPS position regardless of the *automatic recentering parameter*.

-  Set destination layer for GPS digitized features: By default, QGIS will *Follow active layer* for feature digitizing, meaning that GPS digitizing tools will adapt to the layer selected in *Layers* panel and created features will be stored in it accordingly. While this may be useful in some contexts, it also implies careful interaction with other layers to avoid storing inadvertently features in undesired layers. This option allows you to explicitly indicate a layer for the data storage during the live tracking session, and you could switch depending on your needs. Exposed tools will adapt as well to the selected layer type.

  The GPS destination layer is made automatically editable when a feature is being created, and the user is notified accordingly.

- actions for feature creation:
  -  Add vertex to GPS track using GPS current location
  -  Create a new point feature at current GPS location
  -  Create a new line feature using the current GPS track
  -  Create a new polygon feature using the current GPS track
  -  Reset track
-  Show GPS information: opens the *GPS Information* panel
- a display box for quick access to some GPS information components you may need:
  -  Show location
  -  Show altitude (Geoid)
  -  Show altitude (WGS 84 ellipsoid)
  -  Show ground speed
  -  Show bearing
  -  Show total track length
  -  Show distance from start of track
- The  Settings button opens a drop-down menu with common settings which are expected to be modified mid-session:
  -  *Show location marker*
  -  *Show bearing line*
  -  *Rotate map to match GPS direction*
  - Options to control map recentering:
    -  *Always recenter map* when the GPS position is offset from the map canvas center by a certain distance (as a ratio of the map canvas extent)
    -  *Recenter map when leaving extent*
    -  *Never recenter*
  -  *Automatically add track vertices* whenever new locations are received from the GPS device.
  -  *Automatically save added features*: features created from GPS locations are immediately committed to their target layers (skipping the usual layer edit buffer)
  - *Time stamp destination* to adjust the field for time fix storage

- *Log to GeoPackage/SpatiaLite…*: When activated, the user will be prompted to select an existing GeoPackage/SpatiaLite file or enter a new file name. A `gps_points` and `gps_tracks` table will be created in the file with a predefined structure.

  All incoming GPS messages will be logged to the `gps_points` layer, along with speed, bearing, altitude and accuracy information from the GPS.

  When the GPS is disconnected (or QGIS closed), the entire recorded GPS track will be added to the `gps_tracks` table (along with some calculated information like track length, start and end times).
- *Log NMEA sentences…*: enables logging all raw NMEA strings from the device to a text file
- *GPS settings…* to access the GPS *global options* dialog

> **Tip**
>
> **Live status bar information on position**
>
> When a GPS device is connected and the user moves the cursor over the map canvas, a live status bar message displays the distance and bearing from the cursor to the GPS position. Project distance and bearing settings are respected in this display. On a touch screen device, use a tap-and-hold event to trigger the live status bar message.

20.2.2 GPS Information Panel

For full monitoring of a live GPS tracking in QGIS, you may need to enable the *GPS Information Panel* (*View* ► *Panels* ► or press `Ctrl+0`).

In the top right corner of the *GPS Information Panel*, you press *Connect* to initiate connection between QGIS and a plugged-in GPS receiver, or *Disconnect* them.

In the top left of the panel, following buttons are accessible:

- Position: live details on the GPS position and sensors
- Signal: signal strength of satellite connections
- Settings: drop-down menu to *live tracking options* you may need to modify during a session

Position and additional attributes

In the Position tab, if the GPS is receiving signals from satellites, you will see your position in latitude, longitude and altitude together with additional attributes.

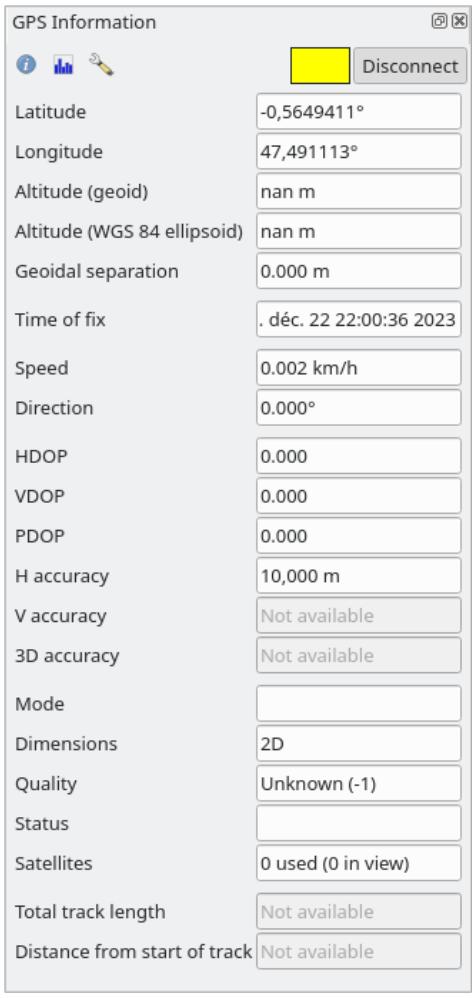


Fig. 20.3: GPS tracking position and additional attributes

- *Latitude*
- *Longitude*
- *Altitude (Geoid)*: Altitude/elevation above or below the mean sea level
- *Altitude (WGS 84 ellipsoid)*: Altitude/elevation above or below the WGS-84 Earth ellipsoid
- *Geoidal separation*: Difference between the WGS-84 Earth ellipsoid and mean sea level (geoid), − means mean sea level is below ellipsoid
- *Time of fix*
- *Speed*: Ground speed
- *Direction*: Bearing measured in degrees clockwise from true north to the direction of travel
- *HDOP*: Horizontal dilution of precision
- *VDOP*: Vertical dilution of precision
- *PDOP*: Dilution of precision
- *H accuracy*: Horizontal accuracy in meters

- *V accuracy*: Vertical accuracy in meters
- *3D accuracy*: 3D Root Mean Square (RMS) in meters
- *Mode*: GPS receiver configuration 2D/3D mode, can be `automatic` or `manual`
- *Dimensions*: Position fix dimension, can be `2D`, `3D` or `No fix`
- *Quality*: Positioning quality indicator
- *Status*: Position fix status, can be `Valid` or `Invalid`
- *Satellites*: Count of satellites used in obtaining the fix
- *Total track length*: Total distance of current GPS track
- *Distance from start of track*: Direct distance from first vertex in current GPS track to latest vertex

Signal

With  Signal tab, you can see the signal strength of the satellites you are receiving signals from.

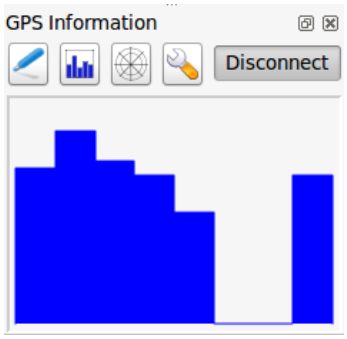


Fig. 20.4: GPS tracking signal strength

20.2.3 Connect to a Bluetooth GPS for live tracking

With QGIS you can connect a Bluetooth GPS for field data collection. To perform this task you need a GPS Bluetooth device and a Bluetooth receiver on your computer.

At first you must let your GPS device be recognized and paired to the computer. Turn on the GPS, go to the Bluetooth icon on your notification area and search for a New Device.

On the right side of the Device selection mask make sure that all devices are selected so your GPS unit will probably appear among those available. In the next step a serial connection service should be available, select it and click on *Configure* button.

Remember the number of the COM port assigned to the GPS connection as resulting by the Bluetooth properties.

After the GPS has been recognized, make the pairing for the connection. Usually the authorization code is `0000`.

Now open *GPS information* panel and switch to GPS options screen. Select the COM port assigned to the GPS connection and click the *Connect*. After a while a cursor indicating your position should appear.

If QGIS can't receive GPS data, then you should restart your GPS device, wait 5-10 seconds then try to connect again. Usually this solution work. If you receive again a connection error make sure you don't have another Bluetooth receiver near you, paired with the same GPS unit.

20.2.4 Examples of GPS devices connection in QGIS

Using GPSMAP 60cs

MS Windows

Easiest way to make it work is to use a middleware (freeware, not open) called GPSGate.

Launch the program, make it scan for GPS devices (works for both USB and BT ones) and then in QGIS:

1. In *Settings* ► *Options* ► *GPS*, detect the plugged-in device. You can use the  *Autodetect* mode.
2. Press *Connect* in the *GPS Information* panel

Ubuntu/Mint GNU/Linux

As for Windows the easiest way is to use a server in the middle, in this case GPSD, so

1. Install the program

   ```
   sudo apt install gpsd
   ```

2. Then load the `garmin_gps` kernel module

   ```
   sudo modprobe garmin_gps
   ```

3. And then connect the unit and check with `dmesg` the actual device being used by the unit, for example `/dev/ttyUSB0`.
4. Now you can launch gpsd

   ```
   gpsd /dev/ttyUSB0
   ```

5. And finally connect with the QGIS live tracking tool.

Using BTGP-38KM datalogger (only Bluetooth)

Using GPSD (under Linux) or GPSGate (under Windows) is effortless.

Using BlueMax GPS-4044 datalogger (both BT and USB)

MS Windows

The live tracking works for both USB and BT modes, by using GPSGate or even without it, just use the  *Autodetect* mode, or point the tool to the right port.

Ubuntu/Mint GNU/Linux

For USB

The live tracking works both with GPSD

```
gpsd /dev/ttyACM3
```

or without it, by connecting the QGIS live tracking tool directly to the device (for example `/dev/ttyACM3`).

For Bluetooth

The live tracking works both with GPSD

```
gpsd /dev/rfcomm0
```

or without it, by connecting the QGIS live tracking tool directly to the device (for example `/dev/rfcomm0`).

CHAPTER

TWENTYONE

AUTHENTICATION SYSTEM

21.1 Authentication System Overview

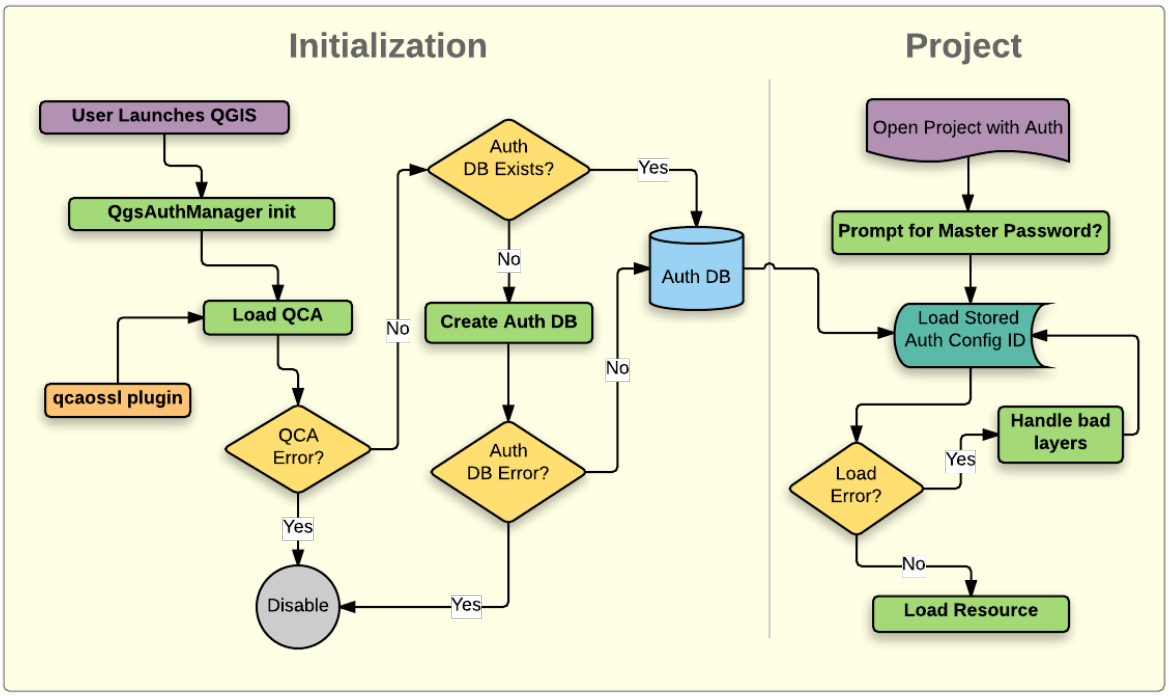


Fig. 21.1: Anatomy of authentication system

21.1.1 Authentication database

The authentication system stores authentication configurations in an SQLite database file located, by default, at `<profile directory>/qgis-auth.db`.

This authentication database can be moved between QGIS installations without affecting other current QGIS user preferences, as it is completely separate from normal QGIS settings. A configuration ID (a random 7-character alphanumeric string) is generated when initially storing a configuration to the database. This represents the configuration, thereby allowing the ID to be stored in plain text application components, (such as project, plugin, or settings files) without disclosure of its associated credentials.

> **Note**
>
> The parent directory of the *qgis-auth.db* can be set using the following environment variable, `QGIS_AUTH_DB_DIR_PATH`, or set on the command line during launch with the `--authdbdirectory` option.

21.1.2 Custom authentication databases

QGIS can be configured to use a custom authentication database instead of the above mentioned default SQLite one: any database suppported by the Qt SQL module can be used (e.g. PostgreSQL, MySQL, etc), provided that the corresponding Qt SQL driver is available in the system.

This can be useful in scenarios where a user wants to share the same authentication database between multiple QGIS installations, or when a user wants to use a different authentication database than the default SQLite one or when a centralized authentication database is used by QGIS server.

The only way to configure a custom authentication database is by setting the `QGIS_AUTH_DB_URI` environment variable to the URI of the connection, the URI is in the form of `driver://username:password@hostname:port/database?options`.

Where:

- `driver` is the name of the Qt SQL driver to use, e.g. `QPSQL` for PostgreSQL, `QMYSQL` for MySQL, etc.
- `username` is the username to use to connect to the database
- `password` is the password to use to connect to the database
- `hostname` is the hostname of the database server
- `port` is the port of the database server
- `database` is the name of the database to use
- `options` are the options to pass to the driver, e.g. `sslmode=require` for PostgreSQL

> **Note**
>
> The `schema` can be specified in the URI options, e.g. `QPSQL://username:password@hostname:port/database?schema=schema_name`
>
> The database must exist before starting QGIS, and the user must have the necessary permissions to connect to the database and to create the required tables if they do not exist.

> **Warning**
>
> The password in the URI is stored in plain text in the environment variable, so it is recommended to use a passwordless user or a user with limited permissions to connect to the database.

> **Warning**
>
> Any database not based on SQLite is considered to be read-only (this can be changed by Python plugins if necessary).

This is an advanced feature, designed to allow one or more custom authentication databases or even custom Python implementations of credentials storages to be used by QGIS.

The system is also designed to allow for multiple authentication databases to be used but there is currently no user facing interface to manage multiple credential storages so its usage requires manual configuration and management, typically from a Python plugin.

21.1.3 Master password

To store or access sensitive information within the database, a user must define a *master password*. A new master password is requested and verified when initially storing any encrypted data to the database. When sensitive information is accessed, the user is prompted for the master password. The password is then cached for the remainder of the session (until application is quit), unless the user manually chooses an action to clear its cached value. Some instances of using the authentication system do not require input of the master password, such as when selecting an existing authentication configuration, or applying a configuration to a server configuration (such as when adding a WMS layer).

You can choose to save the password in the `Wallet/Keyring` of your computer.

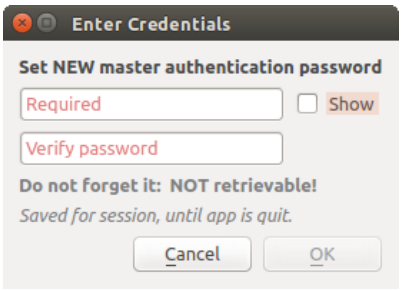


Fig. 21.2: Input new master password

Note

A path to a file containing the master password can be set using the following environment variable, `QGIS_AUTH_PASSWORD_FILE`.

Managing the master password

Once set, the master password can be reset; the current master password will be needed prior to resetting. During this process, there is an option to generate a complete backup of the current database.

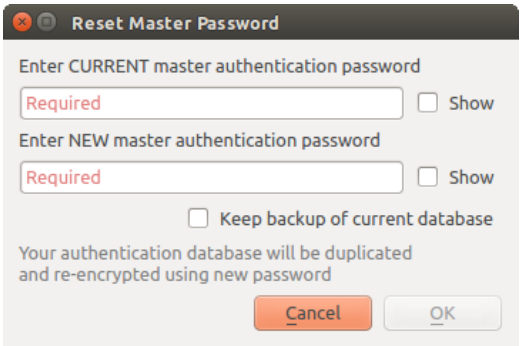


Fig. 21.3: Resetting master password

If the user forgets the master password, there is no way to retrieve or override it. There is also no means of retrieving encrypted information without knowing the master password.

If a user inputs their existing password incorrectly three times, the dialog will offer to erase the database.

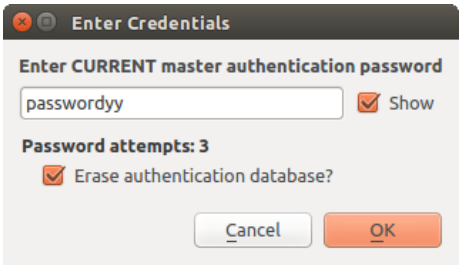


Fig. 21.4: Password prompt after three invalid attempts

21.1.4 Authentication Configurations

You can manage authentication configurations from *Configurations* in the *Authentication* tab of the QGIS Options dialog (*Settings* ► *Options*).

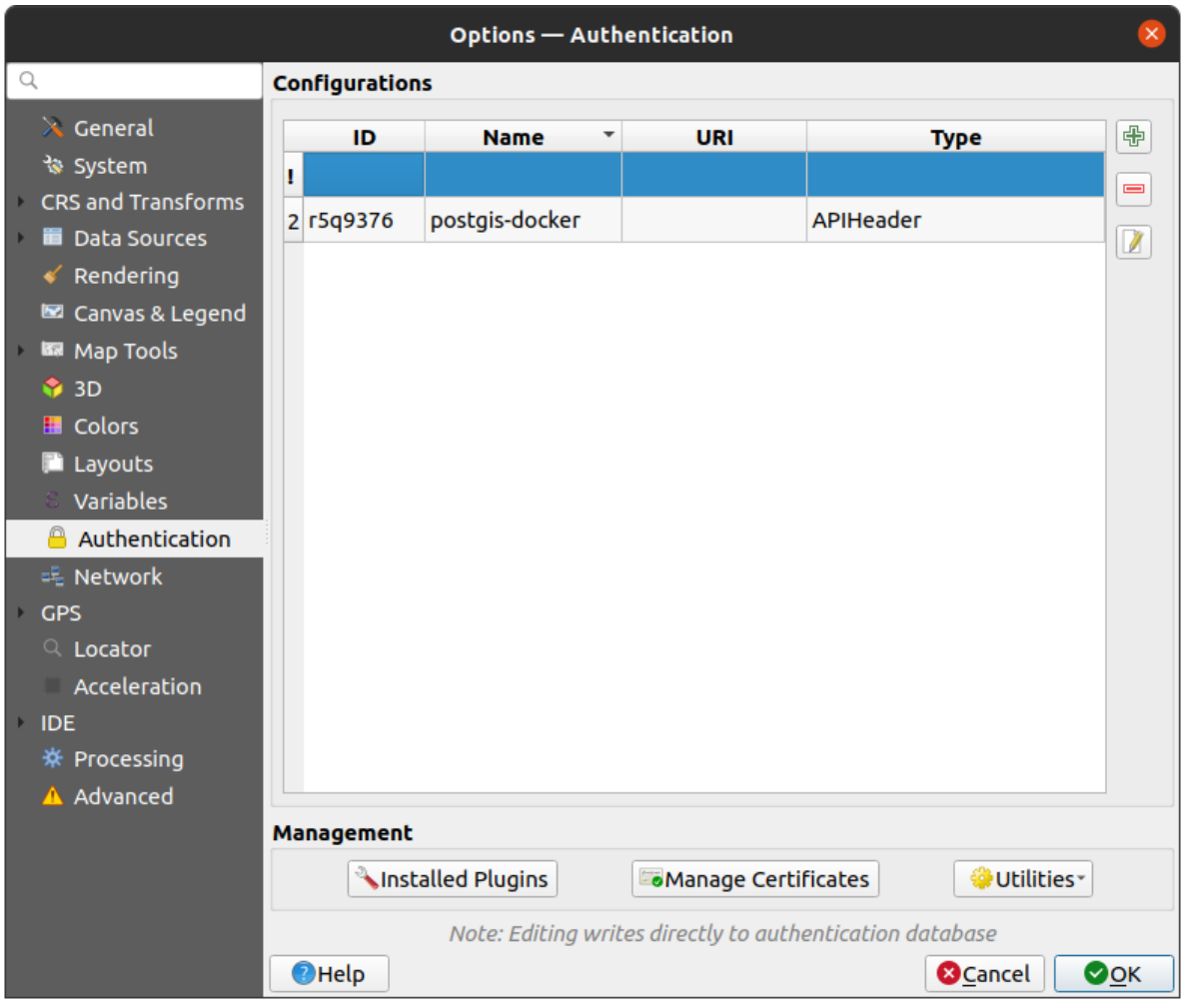


Fig. 21.5: Configurations editor

Use the button to add a new configuration, the button to remove configurations, and the button to modify existing ones.

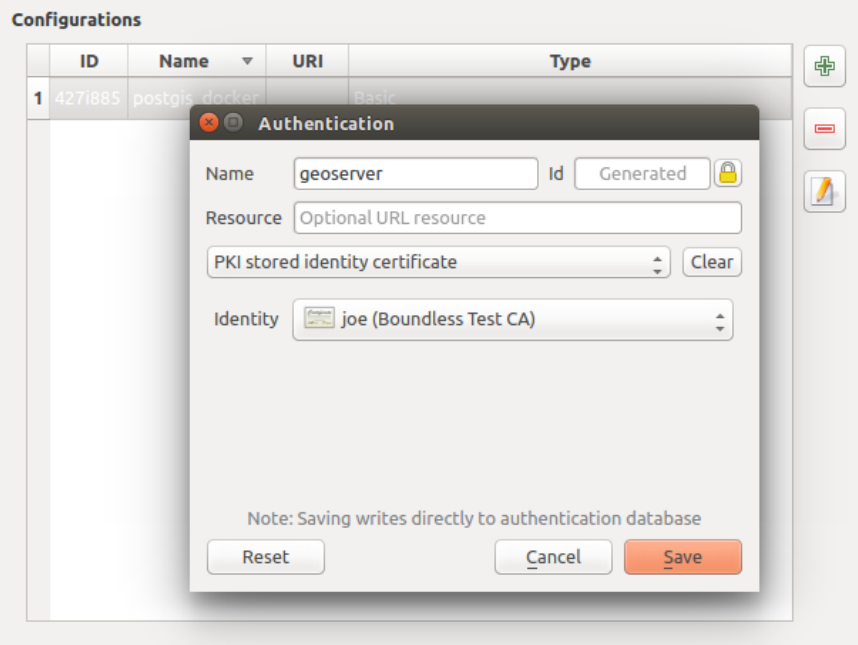


Fig. 21.6: Adding config from within Configuration editor

The same type of operations for authentication configuration management (Add, Edit and Remove) can be done when configuring a given service connection, such as configuring an OWS service connection. For that, there are action buttons within the configuration selector for fully managing configurations found within the authentication database. In this case, there is no need to go to the *configurations* in *Authentication* tab of QGIS options unless you need to do more comprehensive configuration management.

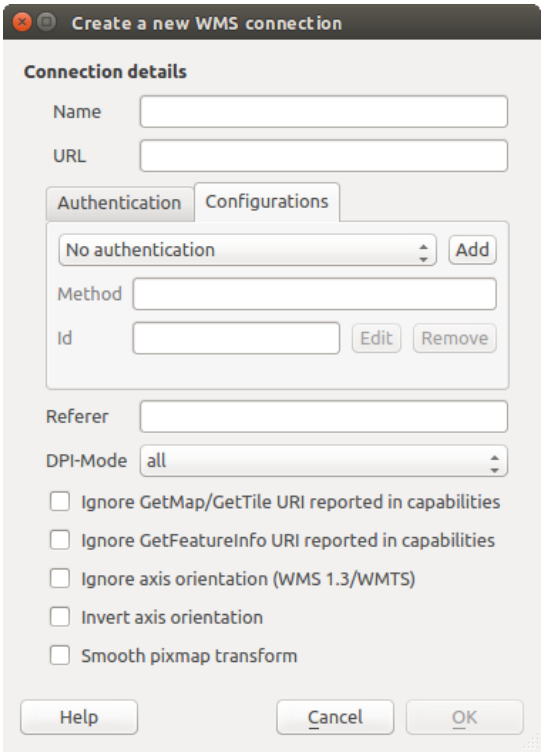


Fig. 21.7: WMS connection dialog showing *Add*, *Edit*, and *Remove* authentication configuration buttons

When creating or editing an authentication configuration, the info required is a name, an authentication method

and any other info that the authentication method requires (see more about the available authentication types in *Authentication Methods*).

21.1.5 Authentication Methods

Available authentications are provided by C++ plugins much in the same way data provider plugins are supported by QGIS. The method of authentication that can be selected is relative to the access needed for the resource/provider, e.g. HTTP(S) or database, and whether there is support in both QGIS code and a plugin. As such, some authentication method plugins may not be applicable everywhere an authentication configuration selector is shown. A list of available authentication method plugins and their compatible resource/providers can be accessed going to *Settings* ► *Options* and, in the *Authentication* tab, click the *Installed Plugins* button.

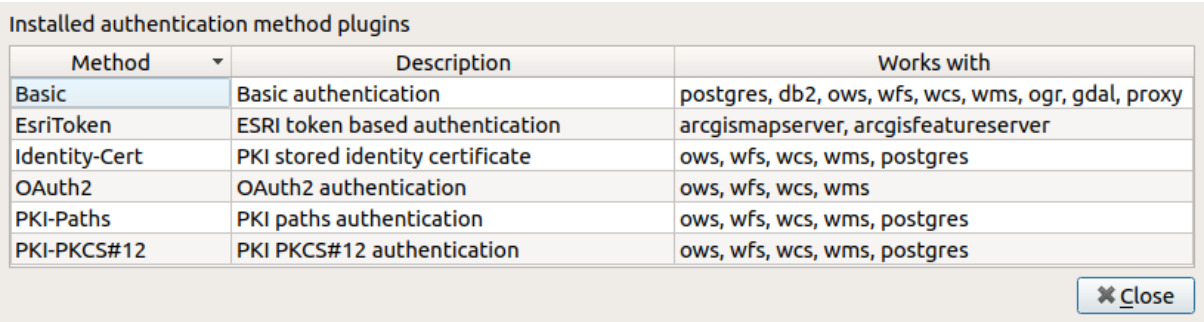


Fig. 21.8: Available method plugins list

Plugins can be created for new authentication methods that do not require QGIS to be recompiled. Since the support for plugins is currently C++-only, QGIS will need to be restarted for the new dropped-in plugin to become available to the user. Ensure your plugin is compiled against the same target version of QGIS if you intend to add it to an existing target install.

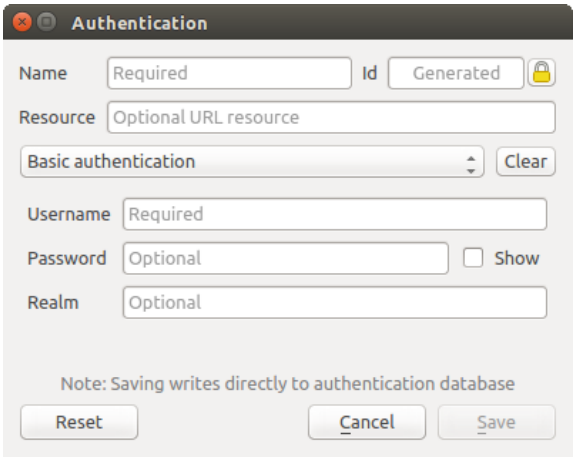


Fig. 21.9: Basic HTTP authentication configs

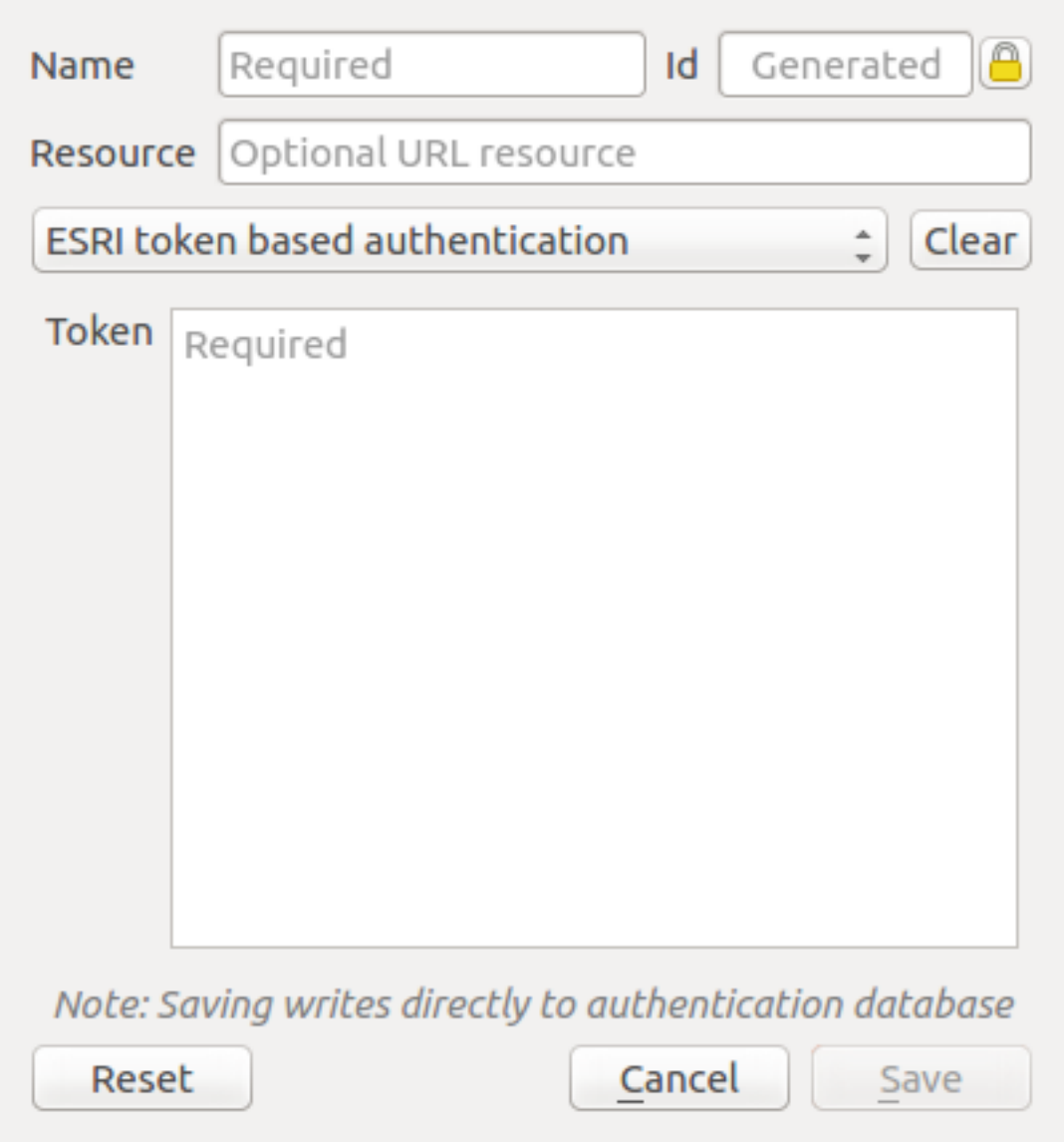


Fig. 21.10: ESRI Token authentication configs

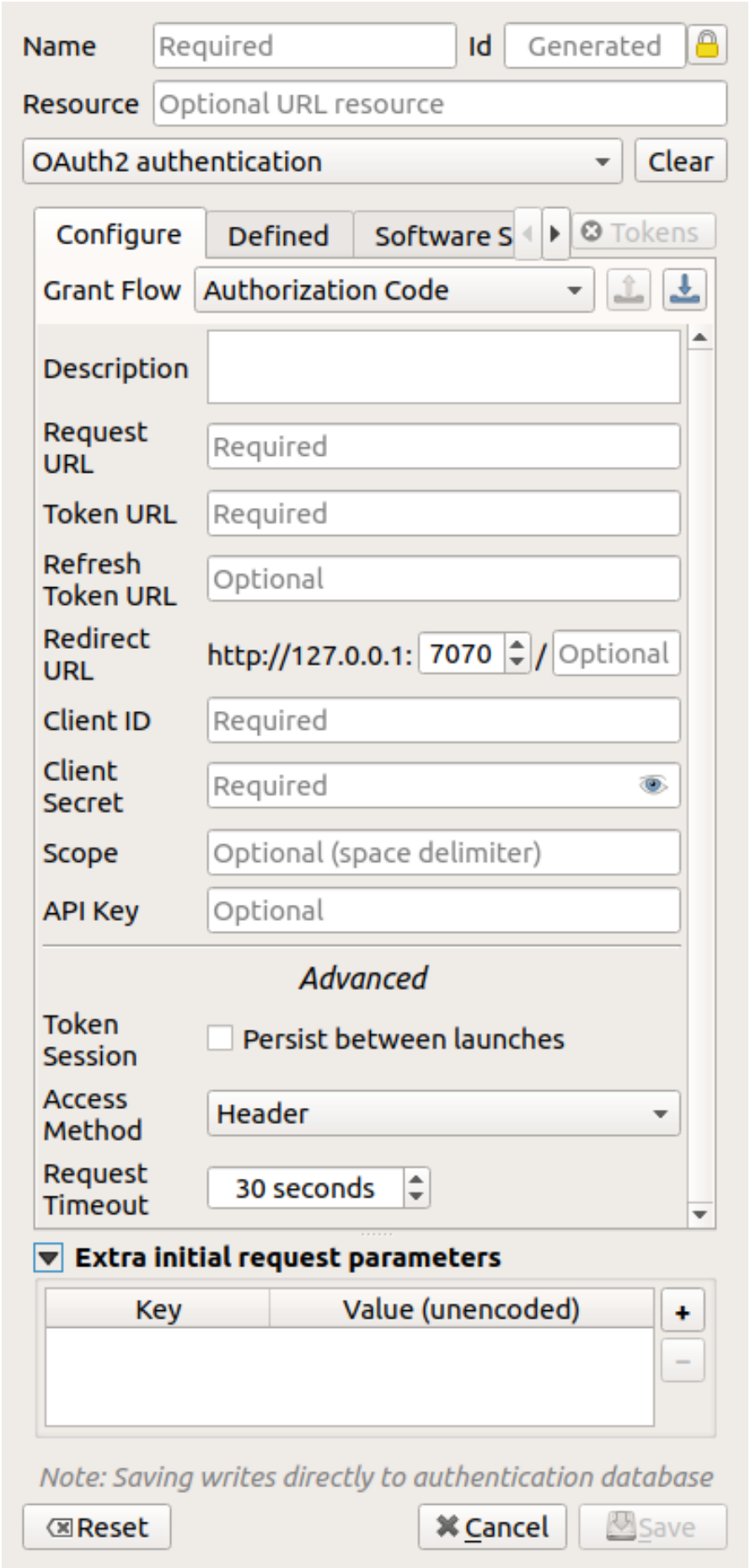


Fig. 21.11: OAuth2 authentication configs

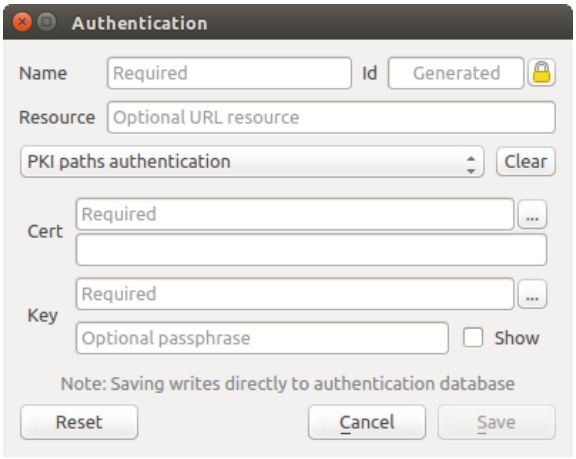


Fig. 21.12: PKI paths authentication configs

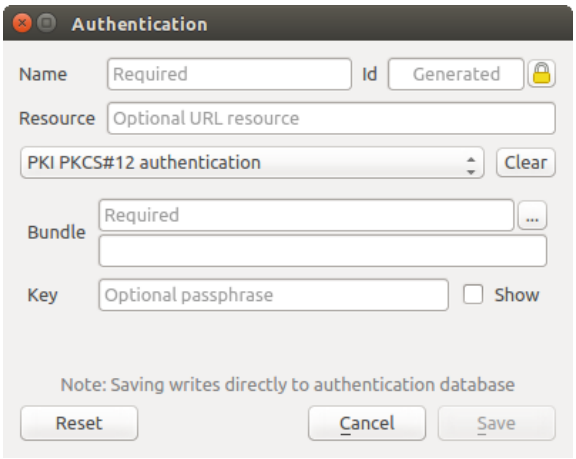


Fig. 21.13: PKI PKCS#12 file paths authentication configs

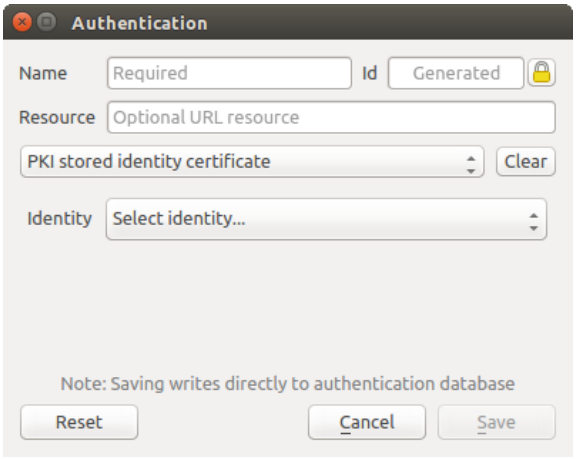


Fig. 21.14: Stored Identity authentication configs

Note

The Resource URL is currently an *unimplemented* feature that will eventually allow a particular configuration to be auto-chosen when connecting to resources at a given URL.

21.1.6 Master Password and Auth Config Utilities

Under the Options menu (*Settings* ► *Options*) in the *Authentication* tab, there are several utility actions to manage the authentication database and configurations:

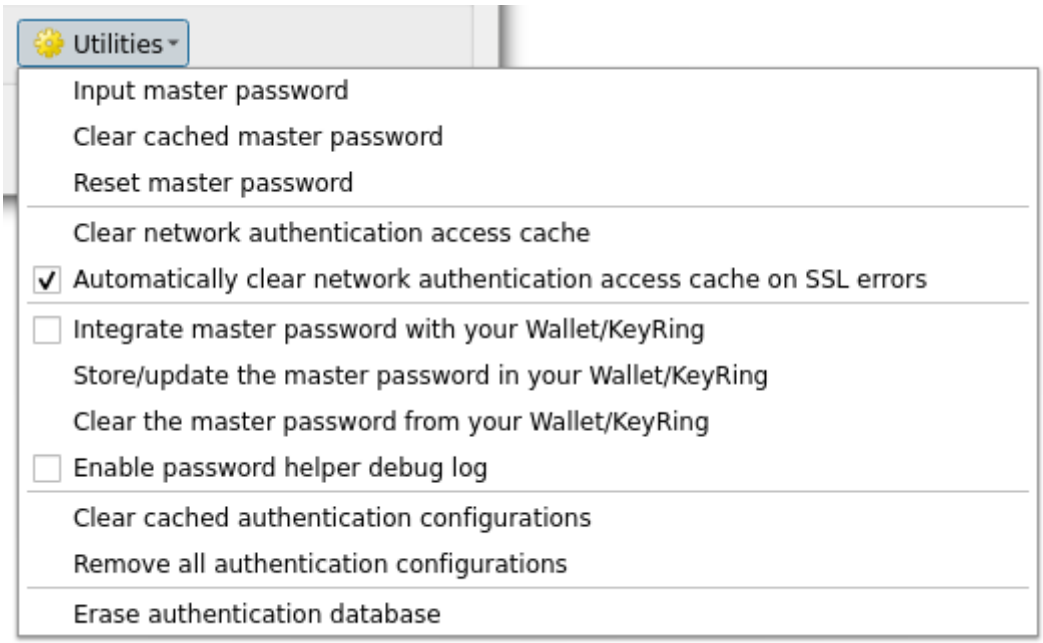


Fig. 21.15: Utilities menu

- **Input master password**: opens the master password input dialog, independent of performing any authentication database command
- **Clear cached master password**: unsets the master password if it has been set
- **Reset master password**: opens a dialog to change the master password (the current password must be known) and optionally back up the current database
- **Clear network authentication access cache**: clears the authentication cache of all connections
- **Automatically clear network authentication access cache on SSL errors**: the connection cache stores all authentication data for connections, also when the connection fails. If you change authentication configurations or certification authorities, you should clear the authentication cache or restart QGIS. When this option is checked, the authentication cache will be automatically cleared every time an SSL error occurs and you choose to abort the connection
- **Integrate master password with your Wallet/Keyring**: adds the master password to your personal Wallet/Keyring
- **Store/update the master password in your Wallet/Keyring**: updates the changed master password in your Wallet/Keyring
- **Clear the master password from your Wallet/Keyring**: deletes the master password from your Wallet/Keyring
- **Enable password helper debug log**: enables a debug tool that will contain all the log information of the authentication methods

- **Clear cached authentication configurations**: clears the internal lookup cache for configurations, used to speed up network connections. This does not clear QGIS's core network access manager's cache, which requires a relaunch of QGIS.
- **Remove all authentication configurations**: clears the database of all configuration records, without removing other stored records.
- **Erase authentication database**: schedules a backup of the current database and complete rebuild of the database table structure. The actions are scheduled for a later time, to ensure that other operations, like project loading, do not interrupt the operation or cause errors due to a temporarily missing database.

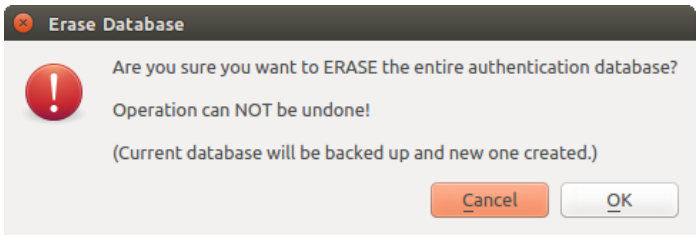


Fig. 21.16: DB erase verification menu

21.1.7 Using authentication configurations

Typically, an authentication configuration is selected in a configuration dialog for a network services (such as WMS). However, the selector widget can be embedded anywhere authentication is needed or in non-core functionality, like in third-party PyQGIS or C++ plugins.

When using the selector, *No authentication* is displayed in the pop-up menu control when nothing is selected, when there are no configurations to choose from, or when a previously assigned configuration can no longer be found in the database. The *Type* and *Id* fields are read-only and provide a description of the authentication method and the config's ID respectively.

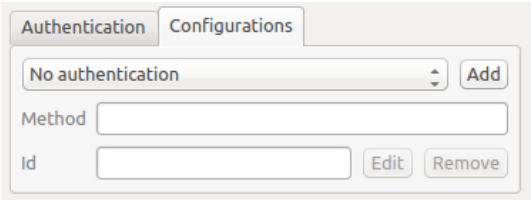


Fig. 21.17: Authentication configuration selector with no authentication

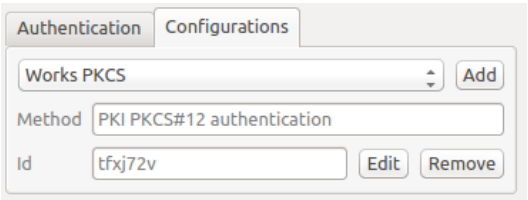


Fig. 21.18: Authentication configuration selector with selected config

21.1.8 Python bindings

All classes and public functions have sip bindings, except `QgsAuthCrypto`, since management of the master password hashing and auth database encryption should be handled by the main app, and not via Python.

Once the master password is entered, the API is open to access authentication configs in the authentication database, similar to how Firefox works. However, no wall against PyQGIS access has been defined. This may lead to issues where a user downloads/installs a malicious PyQGIS plugin or standalone app that gains access to authentication credentials.

21.2 User Authentication Workflows

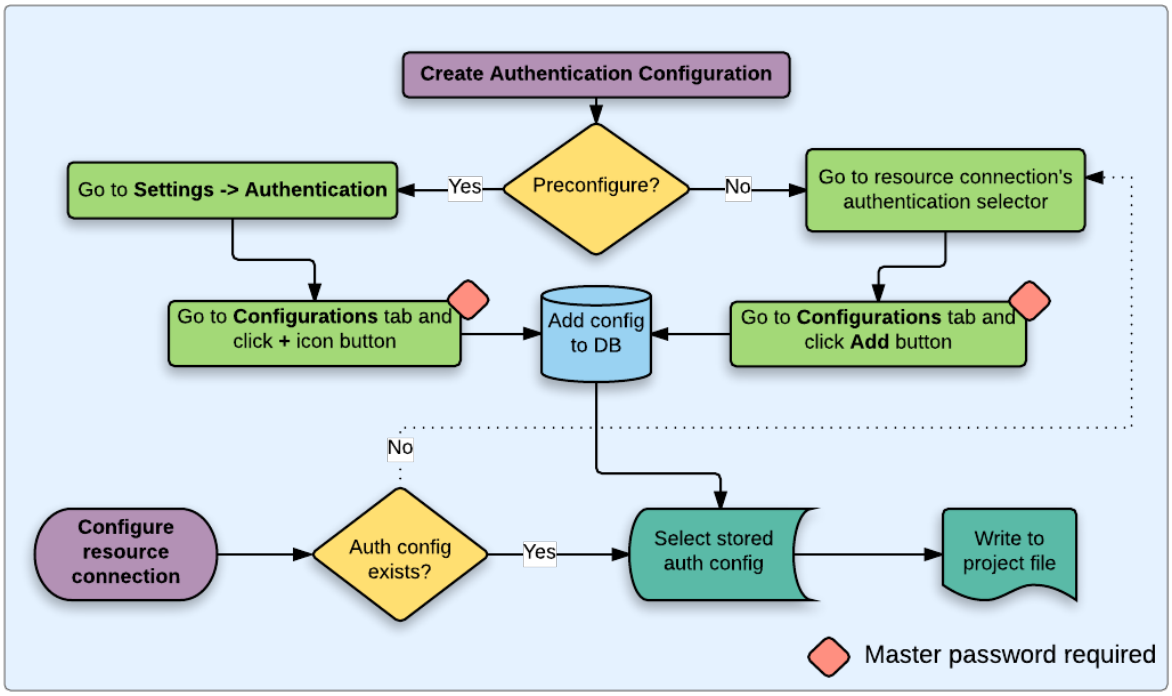


Fig. 21.19: Generic user workflow

21.2.1 HTTP(S) authentication

One of the most common resource connections is via HTTP(S), e.g. web mapping servers, and authentication method plugins often work for these types of connections. Method plugins have access to the HTTP request object and can manipulate both the request as well as its headers. This allows for many forms of internet-based authentication. When connecting via HTTP(S) using the standard username/password authentication method will attempt HTTP BASIC authentication upon connection.

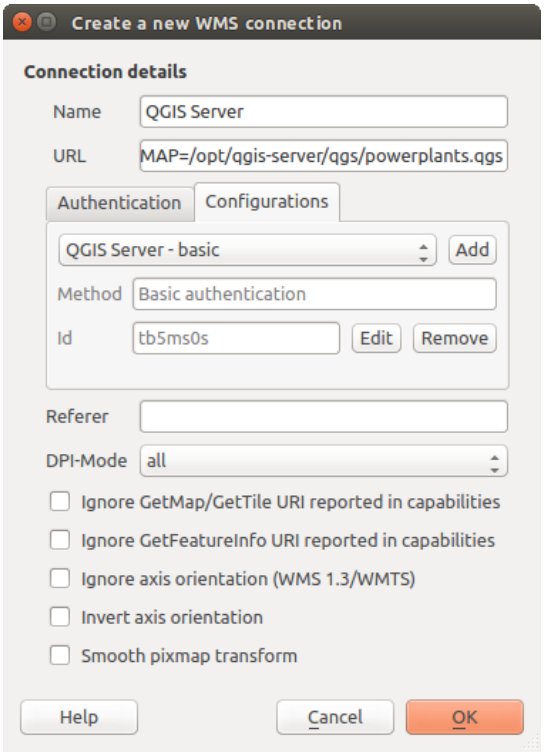


Fig. 21.20: Configuring a WMS connection for HTTP BASIC

21.2.2 Database authentication

Connections to database resources are generally stored as `key=value` pairs, which will expose usernames and (optionally) passwords, if *not* using an authentication configuration. When configuring with the auth system, the `key=value` will be an abstracted representation of the credentials, e.g. `authfg=81t21b9`.

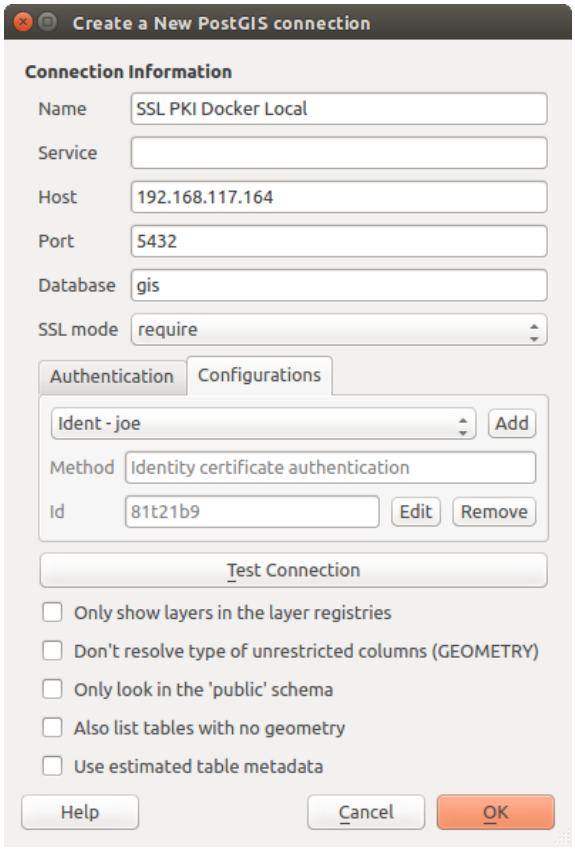


Fig. 21.21: Configuring a Postgres SSL-with-PKI connection

21.2.3 PKI authentication

When configuring PKI components within the authentication system, you have the option of importing components into the database or referencing component files stored on your filesystem. The latter may be useful if such components change frequently, or where the components will be replaced by a system administrator. In either instance you will need to store any passphrase needed to access private keys within the database.

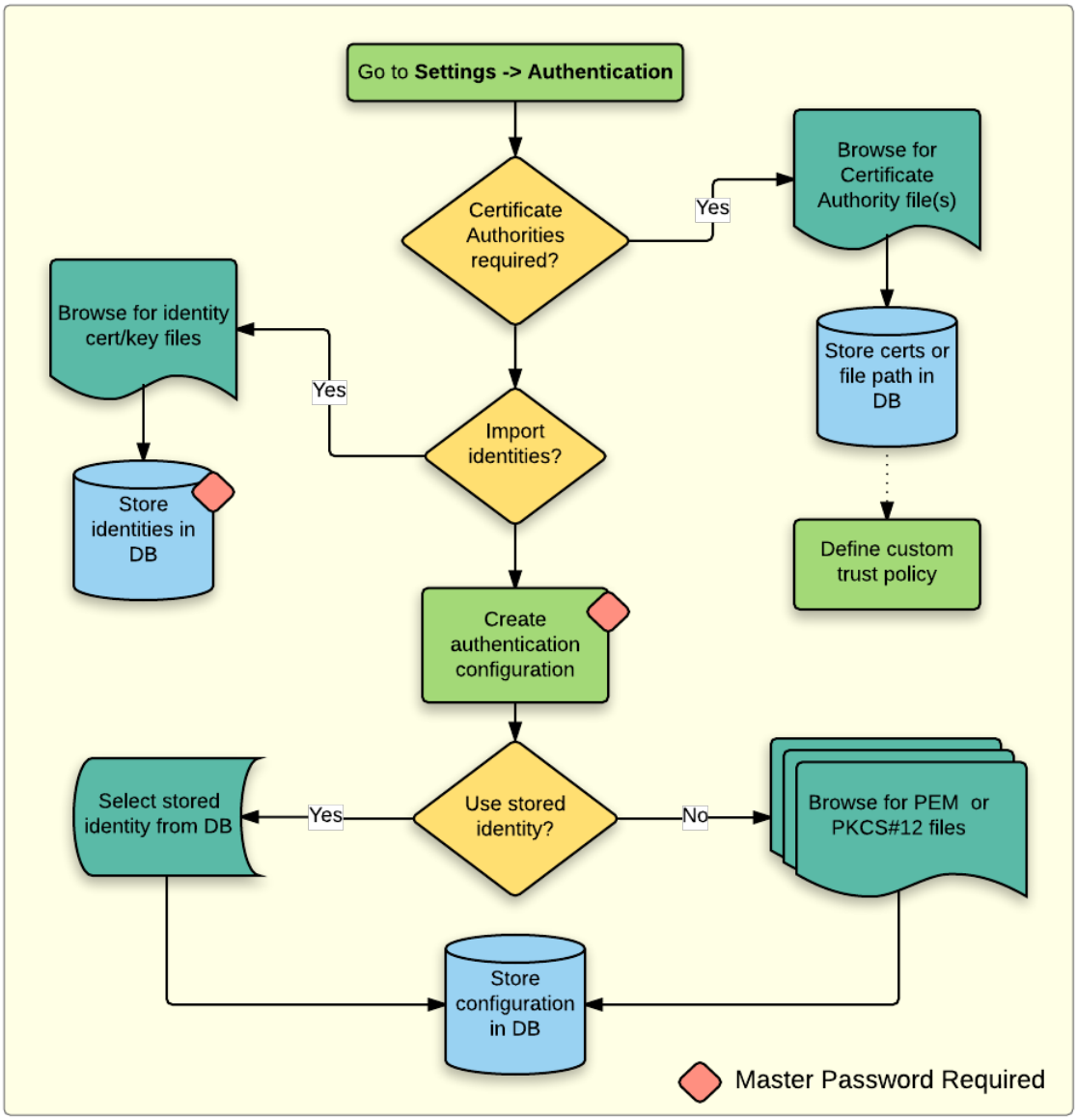


Fig. 21.22: PKI configuration workflow

All PKI components can be managed in separate editors within the **Certificate Manager**, which can be accessed in the *Authentication* tab in QGIS *Options* dialog (*Settings* ► *Options*) by clicking the *Manage Certificates* button.

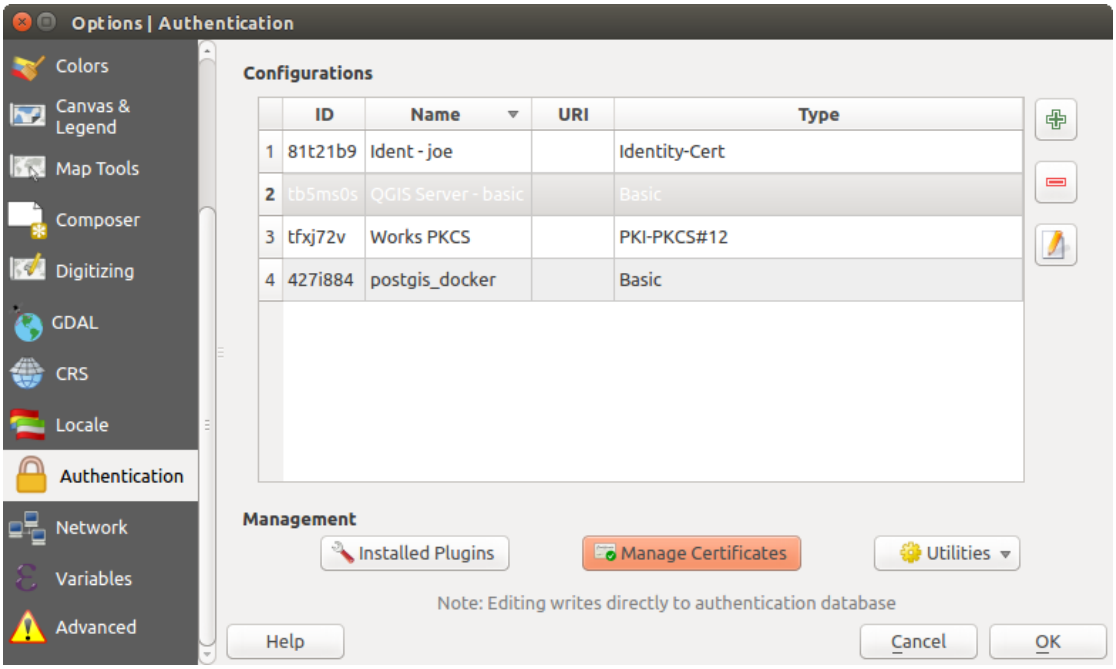


Fig. 21.23: Opening the Certificate Manager

In the *Certificate Manager*, there are editors for **Identities**, **Servers** and **Authorities**. Each of these are contained in their own tabs, and are described below in the order they are encountered in the workflow chart above. The tab order is relative to frequently accessed editors once you are accustomed to the workflow.

> **Note**
>
> Because all authentication system edits write immediately to the authentication database, there is no need to click the *Options* dialog *OK* button for any changes to be saved. This is unlike other settings in the Options dialog.

Authorities

You can manage available Certificate Authorities (CAs) from the **Authorities** tab in the **Certificate manager** from the **Authentication** tab of the QGIS **Options** dialog.

As referenced in the workflow chart above, the first step is to import or reference a file of CAs. This step is optional, and may be unnecessary if your PKI trust chain originates from root CAs already installed in your operating system (OS), such as a certificate from a commercial certificate vendor. If your authenticating root CA is not in the OS's trusted root CAs, it will need to be imported or have its file system path referenced. (Contact your system administrator if unsure.)

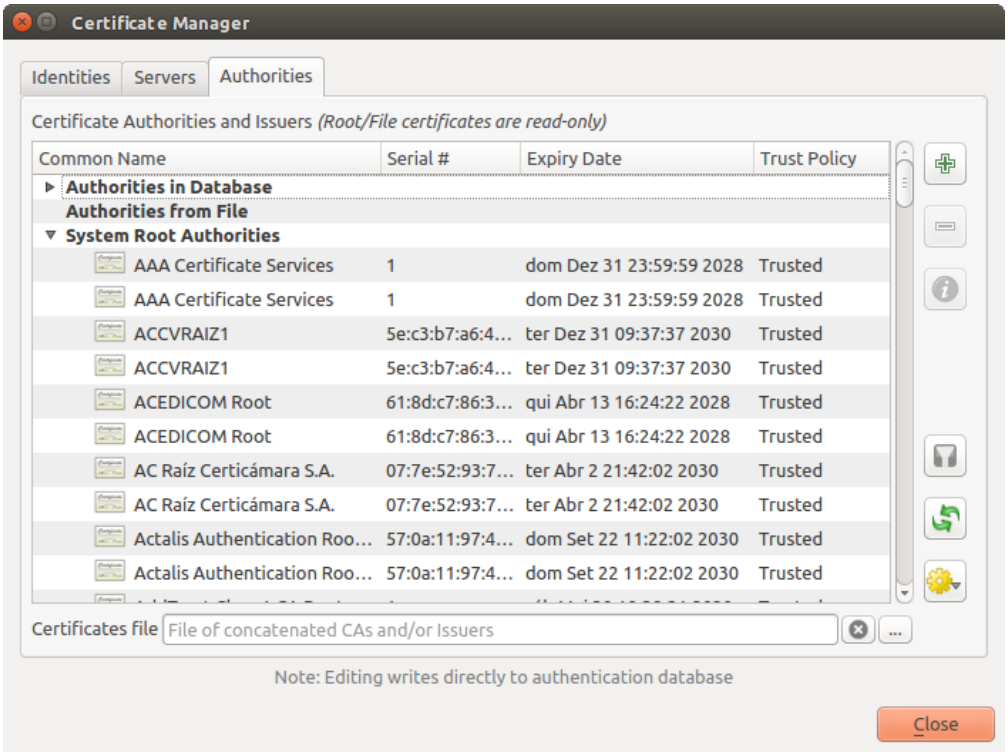


Fig. 21.24: Authorities editor

By default, the root CAs from your OS are available; however, their trust settings are not inherited. You should review the certificate trust policy settings, especially if your OS root CAs have had their policies adjusted. Any certificate that is expired will be set to untrusted and will not be used in secure server connections, unless you specifically override its trust policy. To see the QGIS-discoverable trust chain for any certificate, select it and click the Show information for certificate.

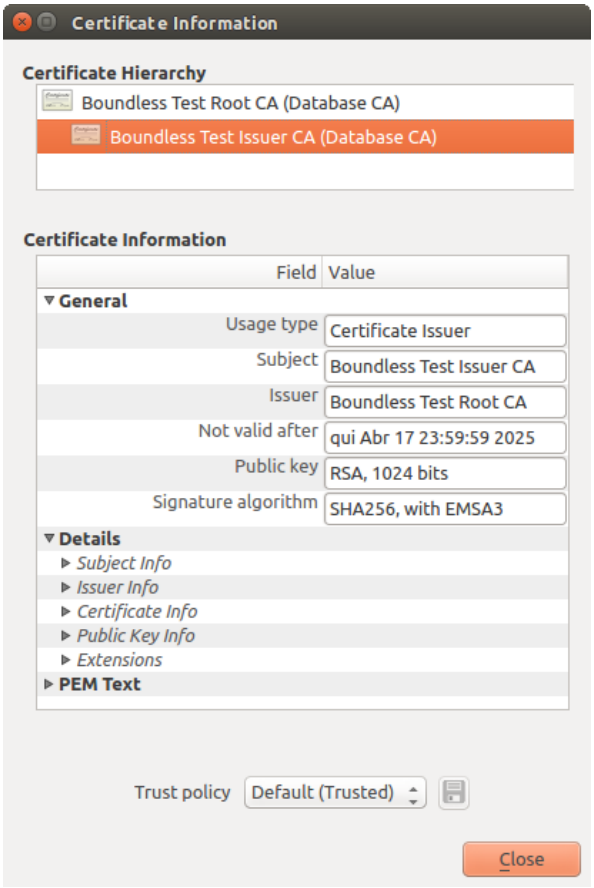


Fig. 21.25: Certificate info dialog

You can edit the *Trust policy*  for any selected certificate within the chain. Any change in trust policy to a selected certificate will not be saved to the database unless the  Save certificate trust policy change to database button is clicked *per* selected certification. Closing the dialog will **not** apply the policy changes.

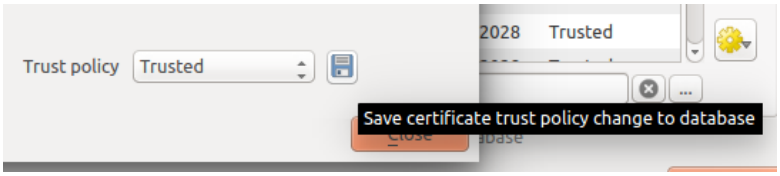


Fig. 21.26: Saving the trust policy changes

You can review the filtered CAs, both intermediate and root certificates, that will be trusted for secure connections or change the default trust policy by clicking the  **Options** button.

Warning

Changing the default trust policy may result in problems with secure connections.

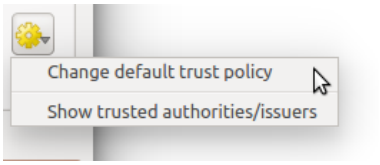


Fig. 21.27: Authorities options menu

You can import CAs or save a file system path from a file that contains multiple CAs, or import individual CAs. The standard PEM format for files that contain multiple CA chain certifications has the root cert at the bottom of the file and all subsequently signed child certificates above, towards the beginning of the file.

The CA certificate import dialog will find all CA certificates within the file, regardless of order, and also offers the option to import certificates that are considered invalid (in case you want to override their trust policy). You can override the trust policy upon import, or do so later within the **Authorities** editor.

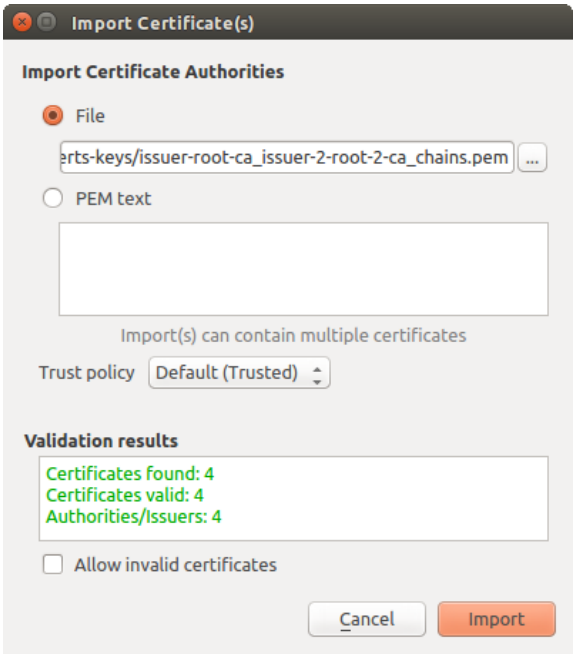


Fig. 21.28: Import certificates dialog

Note

If you are pasting certificate information into the *PEM text* field, note that encrypted certificates are not supported.

Identities

You can manage available client identity bundles from the *Identities* tab in the *Certificate manager* from the **Authentication** tab of the QGIS **Options** dialog. An identity is what authenticates you against a PKI-enabled service and usually consists of a client certificate and private key, either as separate files or combined into a single "bundled" file. The bundle or private key is often passphrase-protected.

Once you have any Certificate Authorities (CAs) imported you can optionally import any identity bundles into the authentication database. If you do not wish to store the identities, you can reference their component file system paths within an individual authentication configuration.

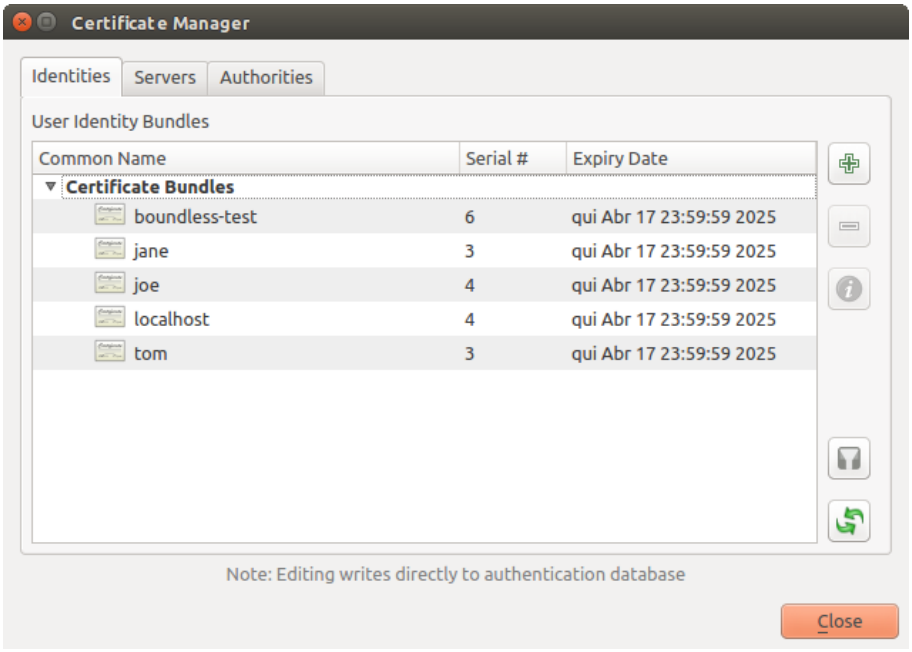


Fig. 21.29: Identities editor

When importing an identity bundle, it can be passphrase-protected or unprotected, and can contain CA certificates forming a trust chain. Trust chain certifications will not be imported here; they can be added separately under the *Authorities* tab.

Upon import the bundle's certificate and private key will be stored in the database, with the key's storage encrypted using the QGIS master password. Subsequent usage of the stored bundle from the database will only require input of the master password.

Personal identity bundles consisting of PEM/DER (.pem/.der) and PKCS#12 (.p12/.pfx) components are supported. If a key or bundle is passphrase-protected, the password will be required to validate the component prior to import. Likewise, if the client certificate in the bundle is invalid (for example, its effective date has not yet started or has elapsed) the bundle can not be imported.

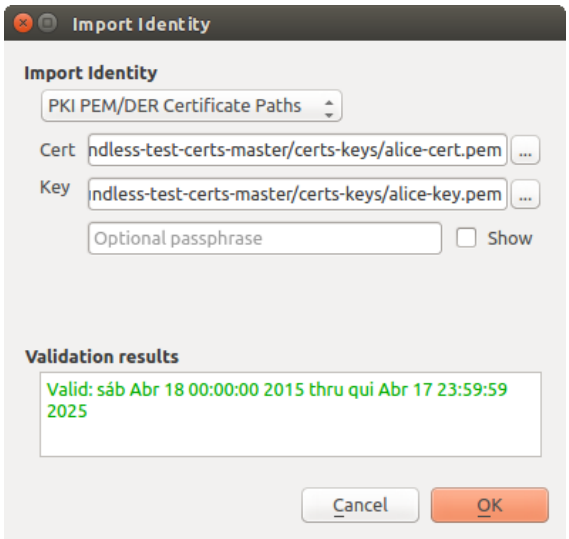


Fig. 21.30: PEM/DER identity import

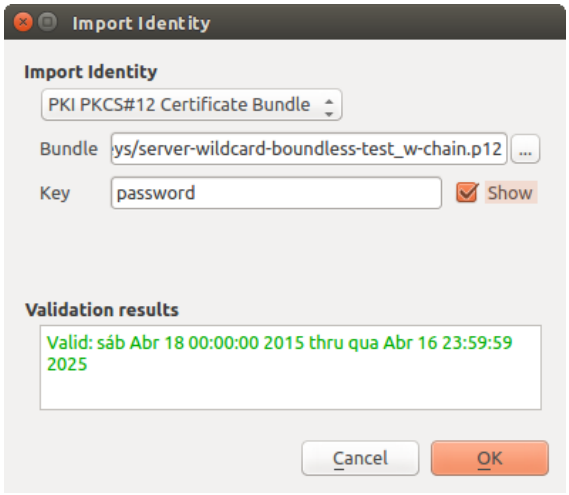


Fig. 21.31: PKCS#12 identity import

21.2.4 Handling bad layers

Occasionally, the authentication configuration ID that is saved with a project file is no longer valid, possibly because the current authentication database is different than when the project was last saved, or due to a credentials mismatch. In such cases the *Handle bad layers* dialog will be presented upon QGIS launch.

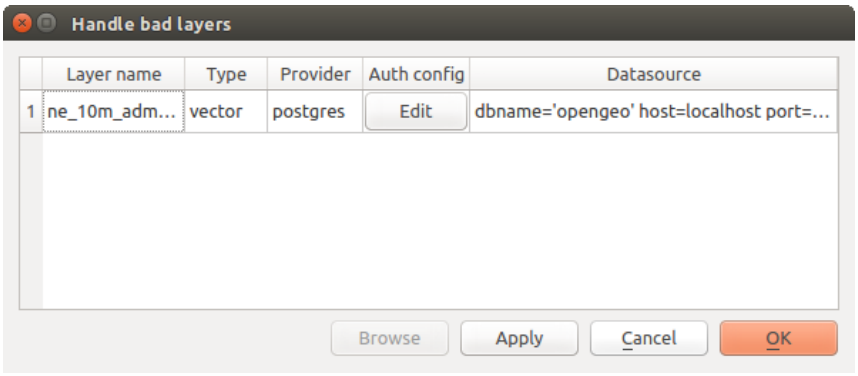


Fig. 21.32: Handle bad layers with authentication

If a data source is found to have an authentication configuration ID associated with it, you will be able to edit it. Doing so will automatically edit the data source string, much in the same way as opening the project file in a text editor and editing the string.

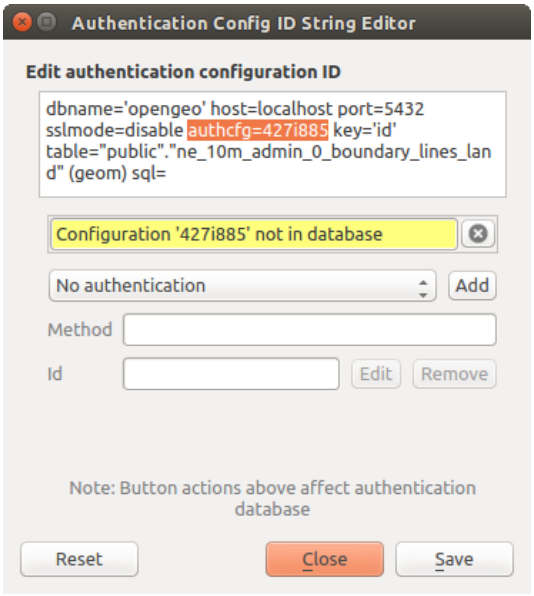


Fig. 21.33: Edit bad layer's authentication config ID

21.2.5 Changing authentication config ID

Occasionally, you will need to change the authentication configuration ID that is associated with accessing a resource. There are instances where this is useful:

- **Resource auth config ID is no longer valid**: This can occur when you have switched auth databases add need to *align* a new configuration to the ID already associated with a resource.
- **Shared project files**: If you intended to share projects between users, e.g. via a shared file server, you can *predefine* a 7-character (containing **a-z** and/or **0-9**) that is associated with the resource. Then, individual users change the ID of an authentication configuration that is specific to their credentials of the resource. When the project is opened, the ID is found in the authentication database, but the credentials are different per user.

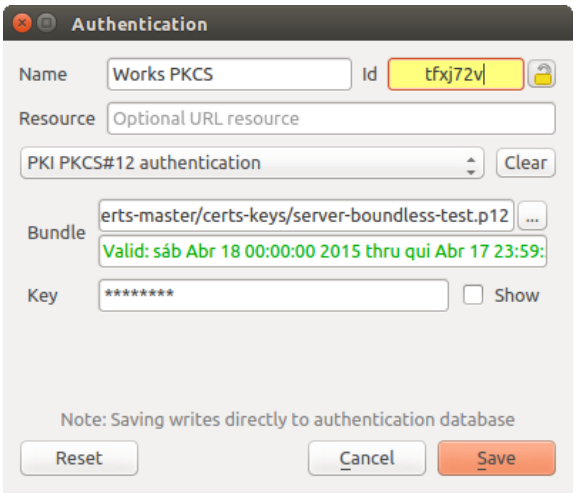


Fig. 21.34: Changing a layer's authentication config ID (unlocked yellow text field)

Warning

Changing the auth config ID is considered an advanced operation and should only be done with full knowledge as

to why it is necessary. This is why there is a lock button that needs clicked, to unlock the ID's text field prior to editing the ID.

21.2.6 QGIS Server support

When using a project file, with layers that have authentication configurations, as a basis for a map in QGIS Server, there are a couple of additional setup steps necessary for QGIS to load the resources:

- Authentication database needs to be available
- Authentication database's master password needs to be available

When instantiating the authentication system, Server will create or use `qgis-auth.db` file in the active *user profile*, or the directory defined by the `QGIS_AUTH_DB_DIR_PATH` environment variable. It may be that the Server's user has no HOME directory, in which case, use the environment variable to define a directory that the Server's user has read/write permissions and is not located within the web-accessible directories.

To pass the master password to Server, write it to the first line of file at a path on the file system readable by the Server processes user and defined using the `QGIS_AUTH_PASSWORD_FILE` environment variable. Ensure to limit the file as only readable by the Server's process user and to not store the file within web-accessible directories.

Note

`QGIS_AUTH_PASSWORD_FILE` variable will be removed from the Server environment immediately after accessing.

21.2.7 SSL server exceptions

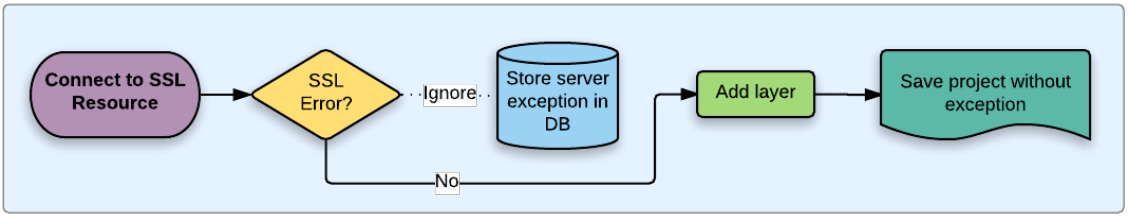


Fig. 21.35: SSL server exception

You can manage SSL server configurations and exceptions from the **Servers** tab in the **Authentication** section of the QGIS **Options** dialog.

Sometimes, when connecting to an SSL server, there are errors with the SSL "handshake" or the server's certificate. You can ignore those errors or create an SSL server configuration as an exception. This is similar to how web browsers allow you to override SSL errors, but with more granular control.

Warning

You should not create an SSL server configuration unless you have complete knowledge of the entire SSL setup between the server and client. Instead, report the issue to the server administrator.

> **Note**
>
> Some PKI setups use a completely different CA trust chain to validate client identities than the chain used to validate the SSL server certificate. In such circumstances, any configuration created for the connecting server will not necessarily fix an issue with the validation of your client identity, and only your client identity's issuer or server administrator can fix the issue.

You can pre-configure an SSL server configuration by clicking the button. Alternatively, you can add a configuration when an SSL error occurs during a connection and you are presented with an **SSL Error** dialog (where the error can be ignored temporarily or saved to the database and ignored):

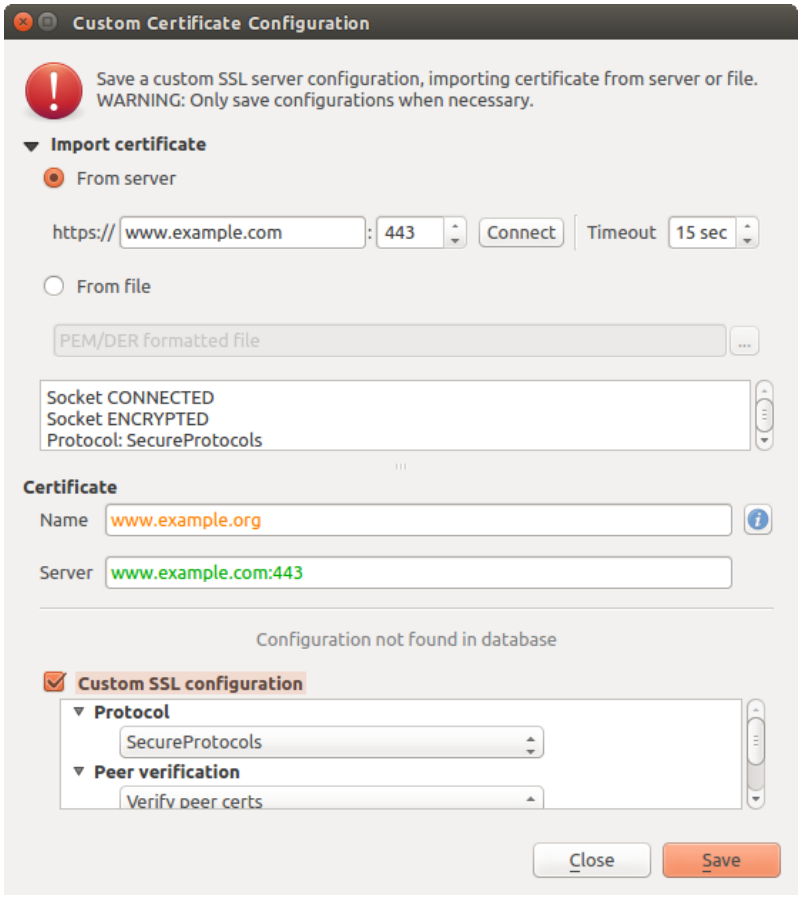


Fig. 21.36: Manually adding configuration

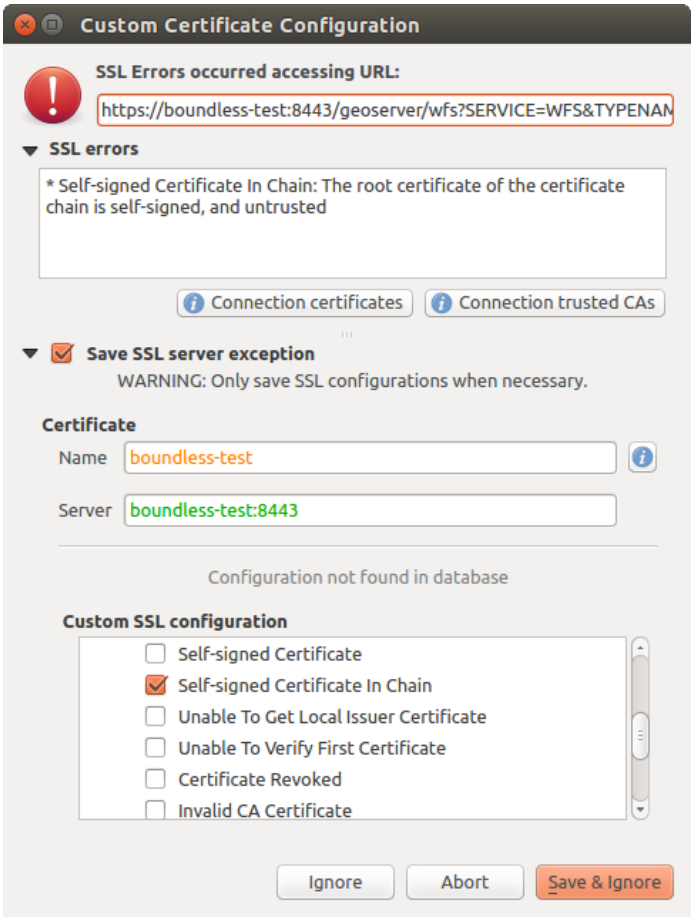


Fig. 21.37: Adding configuration during SSL error

Once an SSL configuration is saved to the database, it can be edited or deleted.

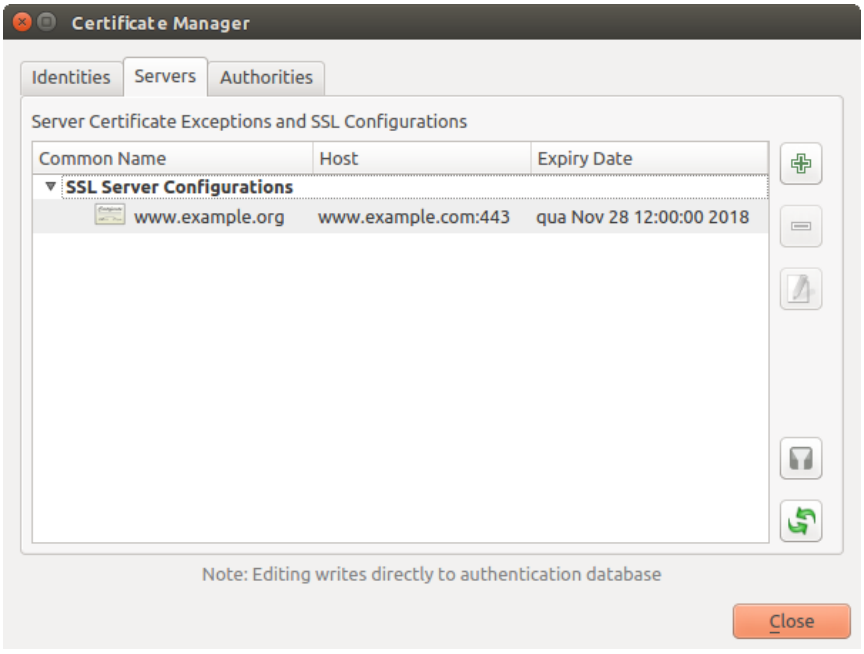


Fig. 21.38: Existing SSL configuration

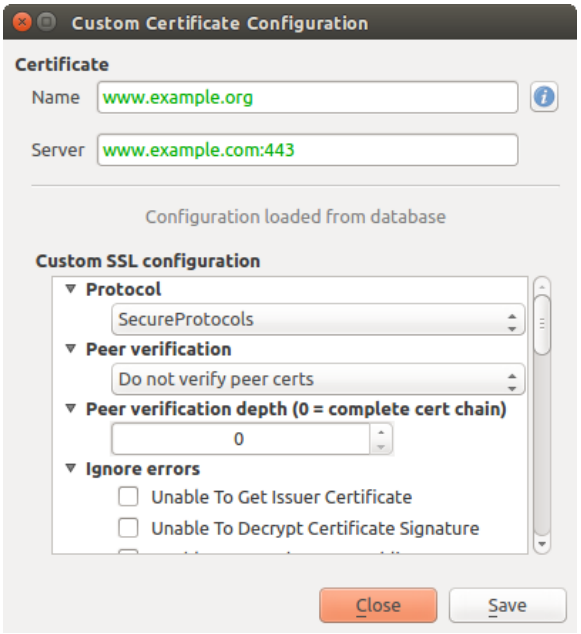


Fig. 21.39: Editing an existing SSL configuration

If you want to pre-configure an SSL configuration and the import dialog is not working for your server's connection, you can manually trigger a connection via the **Python Console** by running the following code (replace `https://bugreports.qt-project.org` with the URL of your server):

```
from qgis.PyQt.QtNetwork import QNetworkRequest
from qgis.PyQt.QtCore import QUrl
from qgis.core import QgsNetworkAccessManager

req = QNetworkRequest(QUrl('https://bugreports.qt-project.org'))
reply = QgsNetworkAccessManager.instance().get(req)
```

This will open an SSL error dialog if any errors occur, where you can choose to save the configuration to the database.

CHAPTER

TWENTYTWO

GRASS GIS INTEGRATION

GRASS integration provides access to GRASS GIS databases and functionalities (see GRASS-PROJECT in *Literature and Web References*). The integration consists of two parts: provider and plugin. The provider allows to browse, manage and visualize GRASS raster and vector layers. The plugin can be used to create new GRASS locations and mapsets, change GRASS region, create and edit vector layers and analyze GRASS 2-D and 3-D data with more than 400 GRASS modules. In this section, we'll introduce the provider and plugin functionalities and give some examples of managing and working with GRASS data.

The provider supports GRASS version 6 and 7, the plugin supports GRASS 6 and 7 (starting from QGIS 2.12). QGIS distribution may contain provider/plugin for either GRASS 6 or GRASS 7 or for both versions at the same time (binaries have different file names). Only one version of the provider/plugin may be loaded on runtime however.

22.1 Demo dataset

As an example, we will use the QGIS Alaska dataset (see section *Downloading sample data*). It includes a small sample GRASS `LOCATION` with three vector layers and one raster elevation map. Create a new folder called `grassdata`, download the QGIS 'Alaska' dataset `qgis_sample_data.zip` from https://download.qgis.org/downloads/data/ and unzip the file into `grassdata`.

More sample GRASS `LOCATIONs` are available at the GRASS website.

22.2 Loading GRASS raster and vector layers

If the provider is loaded in QGIS, the location item with GRASS  icon is added in the browser tree under each folder item which contains GRASS location. Go to the folder `grassdata` and expand location `alaska` and mapset `demo`.

You can load GRASS raster and vector layers like any other layer from the browser either by double click on layer item or by dragging and dropping to map canvas or legend.

Tip

GRASS Data Loading

If you don't see GRASS location item, verify in *Help* ► *About* ► *Providers* if GRASS vector provider is loaded.

22.3 Importing data into a GRASS LOCATION via drag and drop

This section gives an example of how to import raster and vector data into a GRASS mapset.

1. In QGIS browser navigate to the mapset you want to import data into.
2. In QGIS browser find a layer you want to import to GRASS, note that you can open another instance of the browser (*Browser Panel (2)*) if source data are too far from the mapset in the tree.
3. Drag a layer and drop it on the target mapset. The import may take some time for larger layers, you will see animated icon in front of new layer item until the import finishes.

When raster data are in different CRS, they can be reprojected using an *Approximate* (fast) or *Exact* (precise) transformation. If a link to the source raster is created (using `r.external`), the source data are in the same CRS and the format is known to GDAL, the source data CRS will be used. You can set these options in the *Browser* tab in *GRASS Options*.

If a source raster has more bands, a new GRASS map is created for each layer with **.<band number>** suffix and group of all maps with icon is created. External rasters have a different icon .

22.4 Managing GRASS data in QGIS Browser

- Copying maps: GRASS maps may be copied between mapsets within the same location using drag and drop.
- Deleting maps: Right click on a GRASS map and select *Delete* from context menu.
- Renaming maps: Right click on a GRASS map and select *Rename* from context menu.

22.5 GRASS Options

GRASS options may be set in *GRASS Options* dialog, which can be opened by right clicking on the location or mapset item in the browser and then choosing *GRASS Options*.

22.6 Starting the GRASS plugin

To use GRASS functionalities in QGIS, you must select and load the GRASS plugin using the Plugin Manager. To do this, go to the menu *Plugins* ► *Manage and Install Plugins…*, select *GRASS* and click *OK*.

The following main features are provided with the GRASS menu (*Plugins* ► *GRASS*) when you start the GRASS plugin:

- Open Mapset
- New Mapset
- Close Mapset
- Open GRASS Tools
- Display Current GRASS Region
- GRASS Options

22.7 Opening GRASS mapset

A GRASS mapset must be opened to get access to GRASS Tools in the plugin (the tools are disabled if no mapset is open). You can open a mapset from the browser: right click on mapset item and then choose *Open mapset* from context menu.

22.8 GRASS LOCATION and MAPSET

GRASS data are stored in a directory referred to as GISDBASE. This directory, often called `grassdata`, must be created before you start working with the GRASS plugin in QGIS. Within this directory, the GRASS GIS data are organized by projects stored in subdirectories called `LOCATIONs`. Each `LOCATION` is defined by its coordinate system, map projection and geographical boundaries. Each `LOCATION` can have several `MAPSETs` (subdirectories of the `LOCATION`) that are used to subdivide the project into different topics or sub-regions, or as workspaces for individual team members (see Neteler & Mitasova 2008 in *Literature and Web References*). In order to analyze vector and raster layers with GRASS modules, you generally have to import them into a GRASS `LOCATION`. (This is not strictly true – with the GRASS modules `r.external` and `v.external` you can create read-only links to external GDAL-supported datasets without importing them. This is not the usual way for beginners to work with GRASS, therefore this functionality will not be described here.)

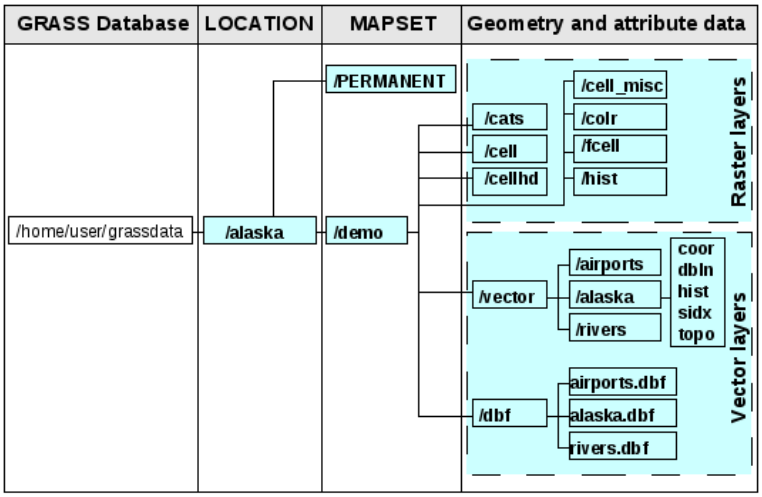


Fig. 22.1: GRASS data in the alaska LOCATION

22.9 Importing data into a GRASS LOCATION

See section *Importing data into a GRASS LOCATION via drag and drop* to find how data can be easily imported by dragging and dropping in the browser.

This section gives an example of how to import raster and vector data into the 'alaska' GRASS `LOCATION` provided by the QGIS 'Alaska' dataset in traditional way, using standard GRASS modules. Therefore, we use the landcover raster map `landcover.img` and the vector GML file `lakes.gml` from the QGIS 'Alaska' dataset (see *Downloading sample data*).

1. Start QGIS and make sure the GRASS plugin is loaded.
2. In the GRASS toolbar, click the Open MAPSET icon to bring up the *MAPSET* wizard.
3. Select as GRASS database the folder `grassdata` in the QGIS Alaska dataset, as `LOCATION` 'alaska', as `MAPSET` 'demo' and click *OK*.
4. Now click the Open GRASS tools icon. The GRASS Toolbox (see section *The GRASS Toolbox*) dialog appears.

5. To import the raster map `landcover.img`, click the module `r.in.gdal` in the *Modules Tree* tab. This GRASS module allows you to import GDAL-supported raster files into a GRASS `LOCATION`. The module dialog for `r.in.gdal` appears.
6. Browse to the folder `raster` in the QGIS 'Alaska' dataset and select the file `landcover.img`.
7. As raster output name, define `landcover_grass` and click *Run*. In the *Output* tab, you see the currently running GRASS command `r.in.gdal -o input=/path/to/landcover.img output=landcover_grass`.
8. When it says **Successfully finished**, click *View Output*. The `landcover_grass` raster layer is now imported into GRASS and will be visualized in the QGIS canvas.
9. To import the vector GML file `lakes.gml`, click the module `v.in.ogr` in the *Modules Tree* tab. This GRASS module allows you to import OGR-supported vector files into a GRASS `LOCATION`. The module dialog for `v.in.ogr` appears.
10. Browse to the folder `gml` in the QGIS 'Alaska' dataset and select the file `lakes.gml` as OGR file.
11. As vector output name, define `lakes_grass` and click *Run*. You don't have to care about the other options in this example. In the *Output* tab you see the currently running GRASS command `v.in.ogr -o dsn=/path/to/lakes.gml output=lakes\_grass`.
12. When it says **Succesfully finished**, click *View Output*. The `lakes_grass` vector layer is now imported into GRASS and will be visualized in the QGIS canvas.

22.9.1 Creating a new GRASS LOCATION

As an example, here is the sample GRASS `LOCATION alaska`, which is projected in the Albers Equal Area projection using feet as units. This sample GRASS `LOCATION alaska` will be used for all examples and exercises in the following GRASS-related sections. It is useful to download and install the dataset on your computer (see *Downloading sample data*).

1. Start QGIS and make sure the GRASS plugin is loaded.
2. Visualize the `alaska.shp` shapefile (see section *Loading a layer from a file*) from the QGIS Alaska dataset (see *Downloading sample data*).
3. In the GRASS toolbar, click on the  New mapset icon to bring up the *MAPSET* wizard.
4. Select an existing GRASS database (GISDBASE) folder `grassdata`, or create one for the new `LOCATION` using a file manager on your computer. Then click *Next*.
5. We can use this wizard to create a new `MAPSET` within an existing `LOCATION` (see section *Adding a new MAPSET*) or to create a new `LOCATION` altogether. Select  *Create new location* (see Fig. 22.2).
6. Enter a name for the `LOCATION` – we used 'alaska' – and click *Next*.
7. Define the projection by clicking on the radio button  *Projection* to enable the projection list.
8. We are using Albers Equal Area Alaska (feet) projection. Since we happen to know that it is represented by the EPSG ID 2964, we enter it in the search box. (Note: If you want to repeat this process for another `LOCATION` and projection and haven't memorized the EPSG ID, click on the  CRS Status icon in the lower right-hand corner of the status bar (see section *Working with Projections*)).
9. In *Filter*, insert 2964 to select the projection.
10. Click *Next*.
11. To define the default region, we have to enter the `LOCATION` bounds in the north, south, east, and west directions. Here, we simply click on the button *Set Current QGIS Extent*, to apply the extent of the loaded layer `alaska.shp` as the GRASS default region extent.
12. Click *Next*.

13. We also need to define a MAPSET within our new LOCATION (this is necessary when creating a new LOCATION). You can name it whatever you like - we used 'demo'. GRASS automatically creates a special MAPSET called PERMANENT, designed to store the core data for the project, its default spatial extent and coordinate system definitions (see Neteler & Mitasova 2008 in *Literature and Web References*).
14. Check out the summary to make sure it's correct and click *Finish*.
15. The new LOCATION, 'alaska', and two MAPSETs, 'demo' and 'PERMANENT', are created. The currently opened working set is 'demo', as you defined.
16. Notice that some of the tools in the GRASS toolbar that were disabled are now enabled.

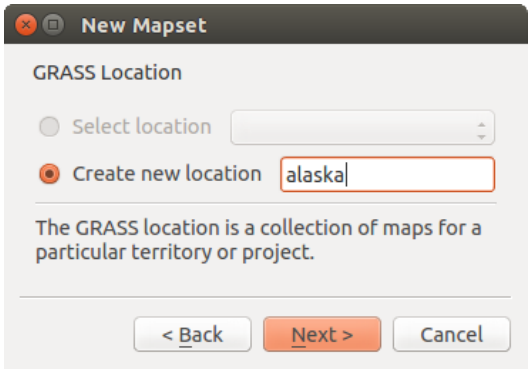


Fig. 22.2: Creating a new GRASS LOCATION or a new MAPSET in QGIS

If that seemed like a lot of steps, it's really not all that bad and a very quick way to create a LOCATION. The LOCATION 'alaska' is now ready for data import (see section *Importing data into a GRASS LOCATION*). You can also use the already-existing vector and raster data in the sample GRASS LOCATION 'alaska', included in the QGIS 'Alaska' dataset *Downloading sample data*, and move on to section *The GRASS vector data model*.

22.9.2 Adding a new MAPSET

A user has write access only to a GRASS MAPSET which he or she created. This means that besides access to your own MAPSET, you can read maps in other users' MAPSETs (and they can read yours), but you can modify or remove only the maps in your own MAPSET.

All MAPSETs include a WIND file that stores the current boundary coordinate values and the currently selected raster resolution (see Neteler & Mitasova 2008 in *Literature and Web References*, and section *The GRASS region tool*).

1. Start QGIS and make sure the GRASS plugin is loaded.
2. In the GRASS toolbar, click on the  New mapset icon to bring up the *MAPSET* wizard.
3. Select the GRASS database (GISDBASE) folder grassdata with the LOCATION 'alaska', where we want to add a further MAPSET called 'test'.
4. Click *Next*.
5. We can use this wizard to create a new MAPSET within an existing LOCATION or to create a new LOCATION altogether. Click on the radio button  *Select location* (see Fig. 22.2) and click *Next*.
6. Enter the name test for the new MAPSET. Below in the wizard, you see a list of existing MAPSETs and corresponding owners.
7. Click *Next*, check out the summary to make sure it's all correct and click *Finish*.

22.10 The GRASS vector data model

It is important to understand the GRASS vector data model prior to digitizing. In general, GRASS uses a topological vector model. This means that areas are not represented as closed polygons, but by one or more boundaries. A boundary between two adjacent areas is digitized only once, and it is shared by both areas. Boundaries must be connected and closed without gaps. An area is identified (and labelled) by the **centroid** of the area.

Besides boundaries and centroids, a vector map can also contain points and lines. All these geometry elements can be mixed in one vector and will be represented in different so-called 'layers' inside one GRASS vector map. So in GRASS, a layer is not a vector or raster map but a level inside a vector layer. This is important to distinguish carefully. (Although it is possible to mix geometry elements, it is unusual and, even in GRASS, only used in special cases such as vector network analysis. Normally, you should prefer to store different geometry elements in different layers.)

It is possible to store several 'layers' in one vector dataset. For example, fields, forests and lakes can be stored in one vector. An adjacent forest and lake can share the same boundary, but they have separate attribute tables. It is also possible to attach attributes to boundaries. An example might be the case where the boundary between a lake and a forest is a road, so it can have a different attribute table.

The 'layer' of the feature is defined by the 'layer' inside GRASS. 'Layer' is the number which defines if there is more than one layer inside the dataset (e.g., if the geometry is forest or lake). For now, it can be only a number. In the future, GRASS will also support names as fields in the user interface.

Attributes can be stored inside the GRASS `LOCATION` as dBase, SQLite3 or in external database tables, for example, PostgreSQL, MySQL, Oracle, etc.

Attributes in database tables are linked to geometry elements using a 'category' value.

'Category' (key, ID) is an integer attached to geometry primitives, and it is used as the link to one key column in the database table.

> **Tip**
>
> **Learning the GRASS Vector Model**
>
> The best way to learn the GRASS vector model and its capabilities is to download one of the many GRASS tutorials where the vector model is described more deeply. See https://grass.osgeo.org/learn/manuals/ for more information, books and tutorials in several languages.

22.11 Creating a new GRASS vector layer

To create a new GRASS vector layer, select one of following items from mapset context menu in the browser:

- New Point Layer
- New Line Layer
- New Polygon Layer

and enter a name in the dialog. A new vector map will be created and layer will be added to canvas and editing started. Selecting type of the layer does not restrict geometry types which can be digitized in the vector map. In GRASS, it is possible to organize all sorts of geometry types (point, line and polygon) in one vector map. The type is only used to add the layer to the canvas, because QGIS requires a layer to have a specific type.

It is also possible to add layers to existing vector maps selecting one of the items described above from context menu of existing vector map.

In GRASS, it is possible to organize all sorts of geometry types (point, line and area) in one layer, because GRASS uses a topological vector model, so you don't need to select the geometry type when creating a new GRASS vector. This is different from shapefile creation with QGIS, because shapefiles use the Simple Feature vector model (see section *Creating new vector layers*).

22.12 Digitizing and editing a GRASS vector layer

GRASS vector layers can be digitized using the standard QGIS digitizing tools. There are however some particularities, which you should know about, due to

- GRASS topological model versus QGIS simple feature
- complexity of GRASS model
  - multiple layers in single maps
  - multiple geometry types in single map
  - geometry sharing by multiple features from multiple layers

The particularities are discussed in the following sections.

Save, discard changes, undo, redo

> **Warning**
>
> All the changes done during editing are immediately written to vector map and related attribute tables.

Changes are written after each operation, it is however, possible to do undo/redo or discard all changes when closing editing. If undo or discard changes is used, original state is rewritten in vector map and attribute tables.

There are two main reasons for this behaviour:

- It is the nature of GRASS vectors coming from conviction that user wants to do what he is doing and it is better to have data saved when the work is suddenly interrupted (for example, blackout)
- Necessity for effective editing of topological data is visualized information about topological correctness, such information can only be acquired from GRASS vector map if changes are written to the map.

Toolbar

The 'Digitizing Toolbar' has some specific tools when a GRASS layer is edited:

Table 22.1: GRASS Digitizing Tools

Icon	Tool	Purpose
	New Point	Digitize new point
	New Line	Digitize new line
	New Boundary	Digitize new boundary
	New Centroid	Digitize new centroid (label existing area)
	New Closed Boundary	Digitize new closed boundary

> **Tip**
>
> **Digitizing polygons in GRASS**
>
> If you want to create a polygon in GRASS, you first digitize the boundary of the polygon. Then you add a centroid (label point) into the closed boundary. The reason for this is that a topological vector model links the attribute information of a polygon always to the centroid and not to the boundary.

Category

Category, often called cat, is sort of ID. The name comes from times when GRASS vectors had only singly attribute "category". Category is used as a link between geometry and attributes. A single geometry may have multiple categories and thus represent multiple features in different layers. Currently it is possible to assign only one category per layer using QGIS editing tools. New features have automatically assigned new unique category, except boundaries. Boundaries usually only form areas and do not represent linear features, it is however possible to define attributes for a boundary later, for example in different layer.

New categories are always created only in currently being edited layer.

It is not possible to assign more categories to geometry using QGIS editing, such data are properly represented as multiple features, and individual features, even from different layers, may be deleted.

Attributes

Attributes of currently edited layer can only be modified. If the vector map contains more layers, features of other layers will have all attributes set to '<not editable (layer #)>' to warn you that such attribute is not editable. The reason is, that other layers may have and usually have different set of fields while QGIS only supports one fixed set of fields per layer.

If a geometry primitive does not have a category assigned, a new unique category is automatically assigned and new record in attribute table is created when an attribute of that geometry is changed.

Tip

If you want to do bulk update of attributes in table, for example using 'Field Calculator' (*Using the Field Calculator*), and there are features without category which you don't want to update (typically boundaries), you can filter them out by setting 'Advanced Filter' to `cat is not null`.

Editing style

The topological symbology is essential for effective editing of topological data. When editing starts, a specialized 'GRASS Edit' renderer is set on the layer automatically and original renderer is restored when editing is closed. The style may be customized in layer properties 'Style' tab. The style can also be stored in project file or in separate file as any other style. If you customize the style, do not change its name, because it is used to reset the style when editing is started again.

Tip

Do not save project file when the layer is edited, the layer would be stored with 'Edit Style' which has no meaning if layer is not edited.

The style is based on topological information which is temporarily added to attribute table as field 'topo_symbol'. The field is automatically removed when editing is closed.

Tip

Do not remove 'topo_symbol' field from attribute table, that would make features invisible because the renderer is based on that column.

Snapping

To form an area, vertices of connected boundaries must have **exactly** the same coordinates. This can be achieved using snapping tool only if canvas and vector map have the same CRS. Otherwise, due conversion from map coordinates to canvas and back, the coordinate may become slightly different due to representation error and CRS transformations.

Tip

Use layer's CRS also for canvas when editing.

Limitations

Simultaneous editing of multiple layers within the same vector at the same time is not supported. This is mainly due to the impossibility of handling multiple undo stacks for a single data source.

On Linux and macOS only one GRASS layer can be edited at time. This is due to a bug in GRASS which does not allow to close database drivers in random order. This is being solved with GRASS developers.

Tip

GRASS Edit Permissions

You must be the owner of the GRASS `MAPSET` you want to edit. It is impossible to edit data layers in a `MAPSET` that is not yours, even if you have write permission.

22.13 The GRASS region tool

The region definition (setting a spatial working window) in GRASS is important for working with raster layers. Vector analysis is by default not limited to any defined region definitions. But all newly created rasters will have the spatial extension and resolution of the currently defined GRASS region, regardless of their original extension and resolution. The current GRASS region is stored in the `$LOCATION/$MAPSET/WIND` file, and it defines north, south, east and west bounds, number of columns and rows, horizontal and vertical spatial resolution.

It is possible to switch on and off the visualization of the GRASS region in the QGIS canvas using the Display current GRASS region button.

The region can be modified in 'Region' tab in 'GRASS Tolls' dock widget. Type in the new region bounds and resolution, and click *Apply*. If you click on *Select the extent by dragging on canvas* you can select a new region interactively with your mouse on the QGIS canvas dragging a rectangle.

The GRASS module `g.region` provides a lot more parameters to define an appropriate region extent and resolution for your raster analysis. You can use these parameters with the GRASS Toolbox, described in section *The GRASS Toolbox*.

22.14 The GRASS Toolbox

The Open GRASS Tools box provides GRASS module functionalities to work with data inside a selected GRASS `LOCATION` and `MAPSET`. To use the GRASS Toolbox you need to open a `LOCATION` and `MAPSET` that you have write permission for (usually granted, if you created the `MAPSET`). This is necessary, because new raster or vector layers created during analysis need to be written to the currently selected `LOCATION` and `MAPSET`.

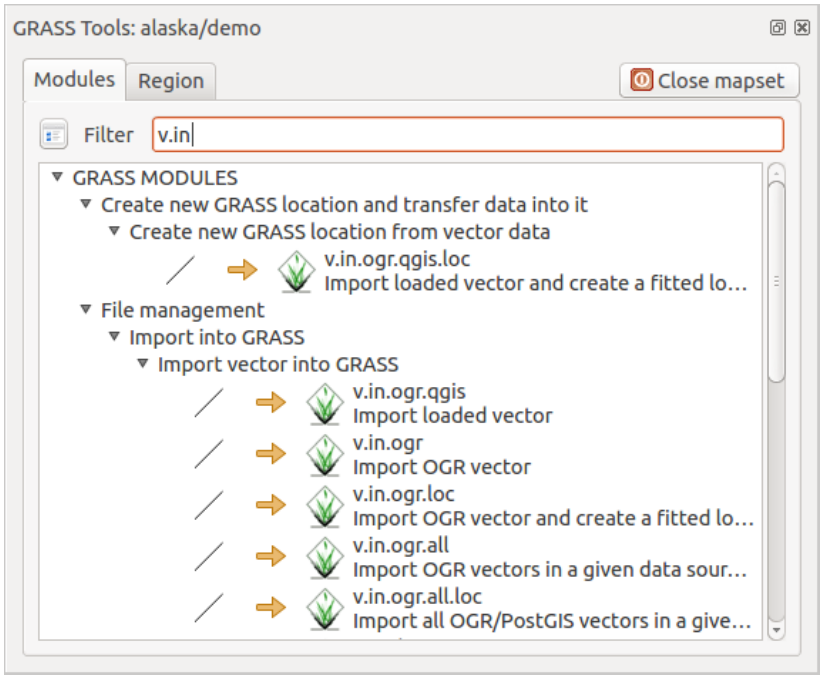


Fig. 22.3: GRASS Toolbox and Module Tree

22.14.1 Working with GRASS modules

The GRASS shell inside the GRASS Toolbox provides access to almost all (more than 300) GRASS modules in a command line interface. To offer a more user-friendly working environment, about 200 of the available GRASS modules and functionalities are also provided by graphical dialogs within the GRASS plugin Toolbox.

A complete list of GRASS modules available in the graphical Toolbox in QGIS version 3.40 is available in the GRASS wiki at https://grasswiki.osgeo.org/wiki/GRASS-QGIS_relevant_module_list.

It is also possible to customize the GRASS Toolbox content. This procedure is described in section *Customizing the GRASS Toolbox*.

As shown in Fig. 22.3, you can look for the appropriate GRASS module using the thematically grouped *Modules Tree* or the searchable *Modules List* tab.

By clicking on a graphical module icon, a new tab will be added to the Toolbox dialog, providing three new sub-tabs: *Options*, *Output* and *Manual*.

Options

The *Options* tab provides a simplified module dialog where you can usually select a raster or vector layer visualized in the QGIS canvas and enter further module-specific parameters to run the module.

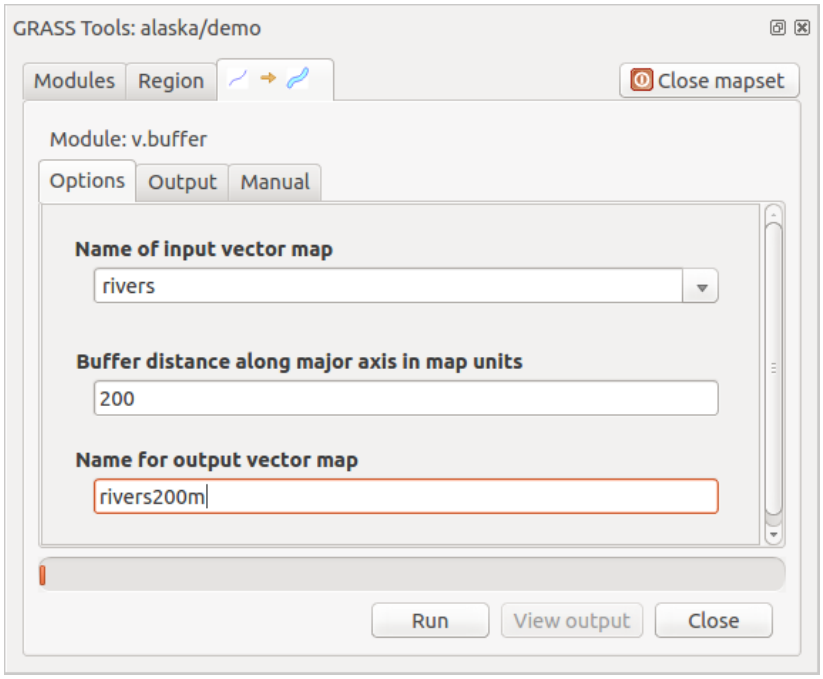


Fig. 22.4: GRASS Toolbox Module Options

The provided module parameters are often not complete to keep the dialog simple. If you want to use further module parameters and flags, you need to start the GRASS shell and run the module in the command line.

A new feature since QGIS 1.8 is the support for a *Show Advanced Options* button below the simplified module dialog in the *Options* tab. At the moment, it is only added to the module `v.in.ascii` as an example of use, but it will probably be part of more or all modules in the GRASS Toolbox in future versions of QGIS. This allows you to use the complete GRASS module options without the need to switch to the GRASS shell.

Output

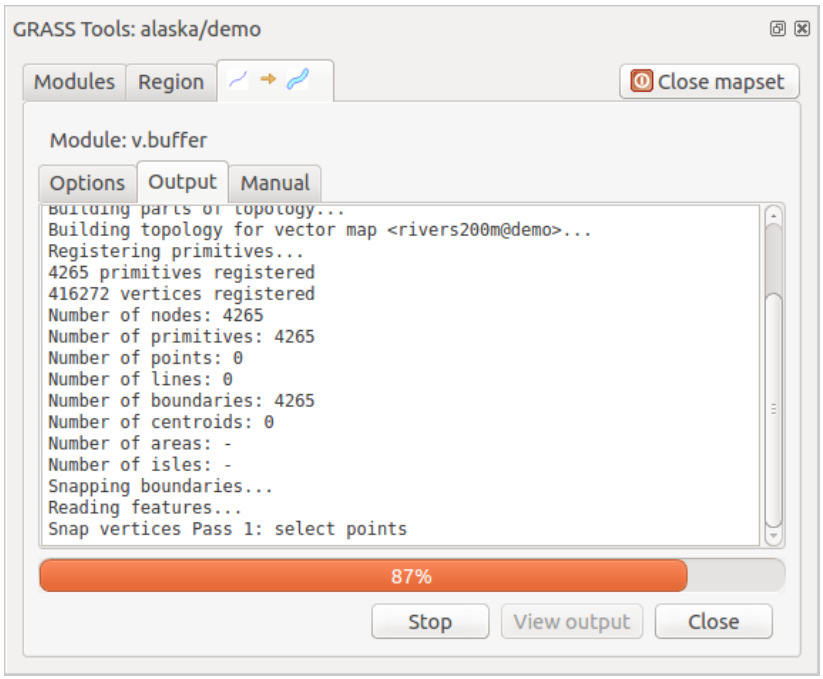


Fig. 22.5: GRASS Toolbox Module Output

The *Output* tab provides information about the output status of the module. When you click the *Run* button, the module switches to the *Output* tab and you see information about the analysis process. If all works well, you will finally see a `Successfully finished` message.

Manual

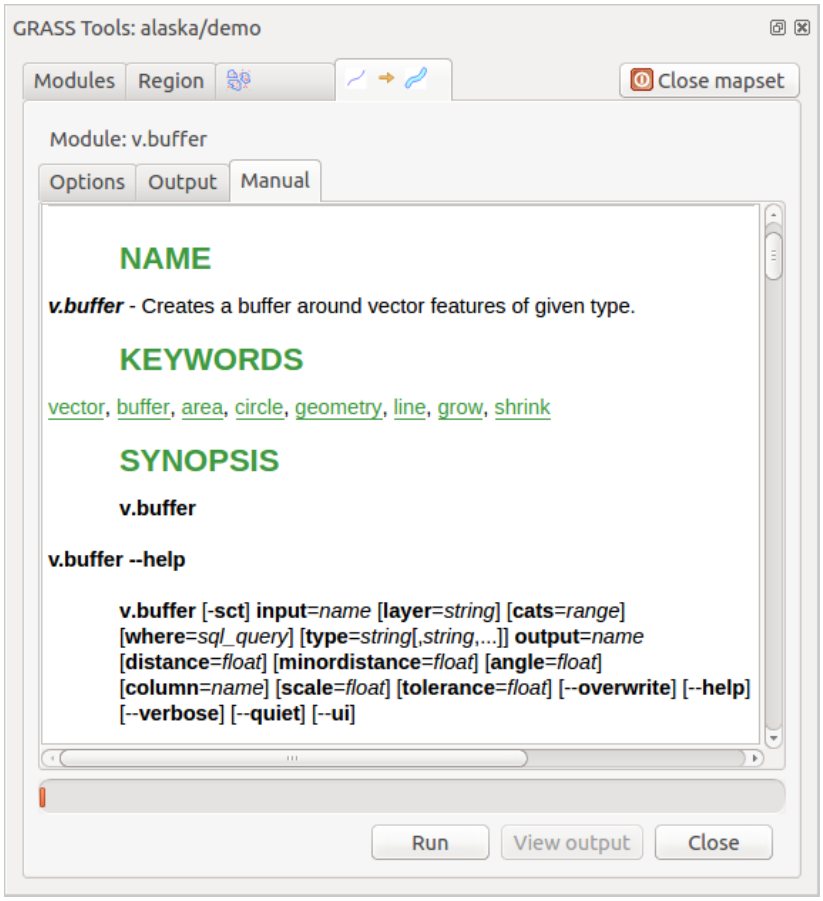


Fig. 22.6: GRASS Toolbox Module Manual

The *Manual* tab shows the HTML help page of the GRASS module. You can use it to check further module parameters and flags or to get a deeper knowledge about the purpose of the module. At the end of each module manual page, you see further links to the `Main Help index`, the `Thematic index` and the `Full index`. These links provide the same information as the module `g.manual`.

Tip

Display results immediately

If you want to display your calculation results immediately in your map canvas, you can use the 'View Output' button at the bottom of the module tab.

22.14.2 GRASS module examples

The following examples will demonstrate the power of some of the GRASS modules.

Creating contour lines

The first example creates a vector contour map from an elevation raster (DEM). Here, it is assumed that you have the Alaska `LOCATION` set up as explained in section *Importing data into a GRASS LOCATION*.

- First, open the location by clicking the  Open mapset button and choosing the Alaska location.
- Now open the Toolbox with the  Open GRASS tools button.
- In the list of tool categories, double-click *Raster* ► *Surface Management* ► *Generate vector contour lines*.
- Now a single click on the tool **r.contour** will open the tool dialog as explained above (see *Working with GRASS modules*).
- In the *Name of input raster map* enter `gtopo30`.
- Type into the *Increment between Contour levels*  the value 100. (This will create contour lines at intervals of 100 meters.)
- Type into the *Name for output vector map* the name `ctour_100`.
- Click *Run* to start the process. Wait for several moments until the message `Successfully finished` appears in the output window. Then click *View Output* and *Close*.

Since this is a large region, it will take a while to display. After it finishes rendering, you can open the layer properties window to change the line color so that the contours appear clearly over the elevation raster, as in *The Vector Properties Dialog*.

Next, zoom in to a small, mountainous area in the center of Alaska. Zooming in close, you will notice that the contours have sharp corners. GRASS offers the **v.generalize** tool to slightly alter vector maps while keeping their overall shape. The tool uses several different algorithms with different purposes. Some of the algorithms (i.e., Douglas Peuker and Vertex Reduction) simplify the line by removing some of the vertices. The resulting vector will load faster. This process is useful when you have a highly detailed vector, but you are creating a very small-scale map, so the detail is unnecessary.

> **Tip**
>
> **The simplify tool**
>
> Note that QGIS has a *Vector* ► *Geometry Tools* ► *Simplify geometries* tool that works just like the GRASS **v.generalize** Douglas-Peuker algorithm.

However, the purpose of this example is different. The contour lines created by `r.contour` have sharp angles that should be smoothed. Among the **v.generalize** algorithms, there is Chaiken's, which does just that (also Hermite splines). Be aware that these algorithms can **add** additional vertices to the vector, causing it to load even more slowly.

- Open the GRASS Toolbox and double-click the categories *Vector* ► *Develop map* ► *Generalization*, then click on the **v.generalize** module to open its options window.
- Check that the 'ctour_100' vector appears as the *Name of input vector*.
- From the list of algorithms, choose Chaiken's. Leave all other options at their default, and scroll down to the last row to enter in the field *Name for output vector map* 'ctour_100_smooth', and click *Run*.
- The process takes several moments. Once `Successfully finished` appears in the output windows, click *View Output* and then *Close*.

- You may change the color of the vector to display it clearly on the raster background and to contrast with the original contour lines. You will notice that the new contour lines have smoother corners than the original while staying faithful to the original overall shape.



Fig. 22.7: GRASS module v.generalize to smooth a vector map

Tip

Other uses for r.contour

The procedure described above can be used in other equivalent situations. If you have a raster map of precipitation data, for example, then the same method will be used to create a vector map of isohyetal (constant rainfall) lines.

Creating a Hillshade 3-D effect

Several methods are used to display elevation layers and give a 3-D effect to maps. The use of contour lines, as shown above, is one popular method often chosen to produce topographic maps. Another way to display a 3-D effect is by hillshading. The hillshade effect is created from a DEM (elevation) raster by first calculating the slope and aspect of each cell, then simulating the sun's position in the sky and giving a reflectance value to each cell. Thus, you get sun-facing slopes lighted; the slopes facing away from the sun (in shadow) are darkened.

- Begin this example by loading the `gtopo30` elevation raster. Start the GRASS Toolbox, and under the Raster category, double-click to open *Spatial analysis* ► *Terrain analysis*.
- Then click **r.shaded.relief** to open the module.
- Change the *azimuth angle* 1,00 270 to 315.
- Enter `gtopo30_shade` for the new hillshade raster, and click *Run*.
- When the process completes, add the hillshade raster to the map. You should see it displayed in grayscale.
- To view both the hillshading and the colors of the `gtopo30` together, move the hillshade map below the `gtopo30` map in the table of contents, then open the *Properties* window of `gtopo30`, switch to the *Transparency* tab and set its transparency level to about 25%.

You should now have the `gtopo30` elevation with its colormap and transparency setting displayed **above** the grayscale hillshade map. In order to see the visual effects of the hillshading, turn off the `gtopo30_shade` map, then turn it back on.

Using the GRASS shell

The GRASS plugin in QGIS is designed for users who are new to GRASS and not familiar with all the modules and options. As such, some modules in the Toolbox do not show all the options available, and some modules do not appear

at all. The GRASS shell (or console) gives the user access to those additional GRASS modules that do not appear in the Toolbox tree, and also to some additional options to the modules that are in the Toolbox with the simplest default parameters. This example demonstrates the use of an additional option in the **r.shaded.relief** module that was shown above.

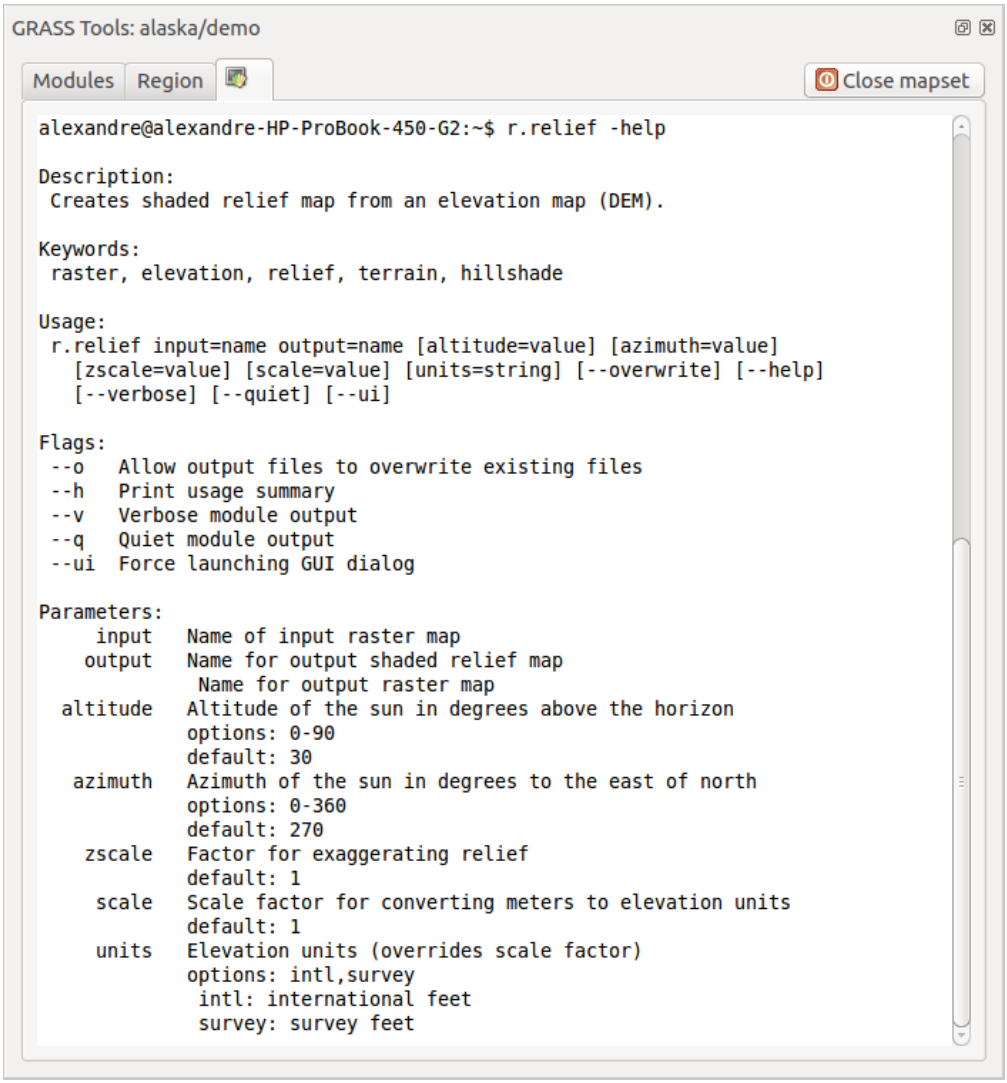


Fig. 22.8: The GRASS shell, r.shaded.relief module

The module **r.shaded.relief** can take a parameter `zmult`, which multiplies the elevation values relative to the X-Y coordinate units so that the hillshade effect is even more pronounced.

- Load the `gtopo30` elevation raster as above, then start the GRASS Toolbox and click on the GRASS shell. In the shell window, type the command `r.shaded.relief map=gtopo30 shade=gtopo30_shade2 azimuth=315 zmult=3` and press `Enter`.
- After the process finishes, shift to the *Browse* tab and double-click on the new `gtopo30_shade2` raster to display it in QGIS.
- As explained above, move the shaded relief raster below the `gtopo30` raster in the table of contents, then check the transparency of the colored `gtopo30` layer. You should see that the 3-D effect stands out more strongly compared with the first shaded relief map.

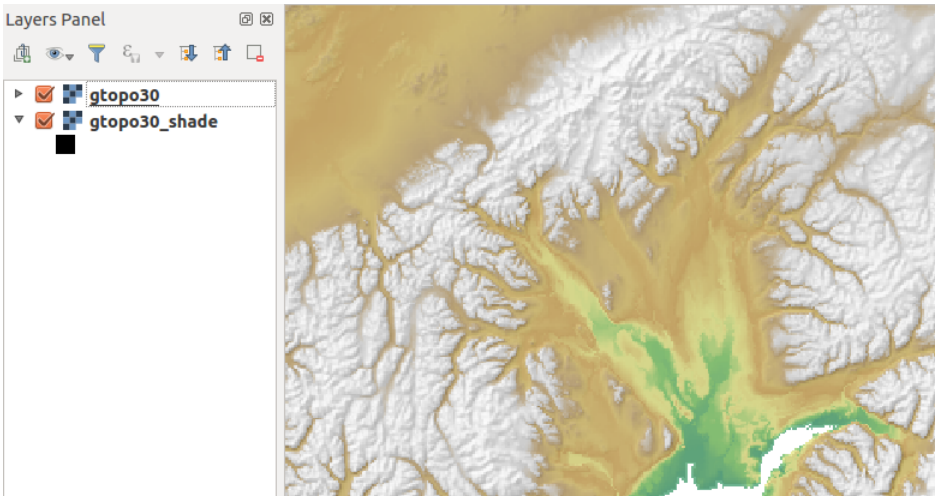


Fig. 22.9: Displaying shaded relief created with the GRASS module r.shaded.relief

Raster statistics in a vector map

The next example shows how a GRASS module can aggregate raster data and add columns of statistics for each polygon in a vector map.

- Again using the Alaska data, refer to *Importing data into a GRASS LOCATION* to import the `shapefiles/trees.shp` file into GRASS.
- Now an intermediate step is required: centroids must be added to the imported trees map to make it a complete GRASS area vector (including both boundaries and centroids).
- From the Toolbox, choose *Vector* ► *Manage features*, and open the module **v.centroids**.
- Enter as the *output vector map* 'forest_areas' and run the module.
- Now load the `forest_areas` vector and display the types of forests - deciduous, evergreen, mixed - in different colors: In the layer *Properties* window, *Symbology* tab, choose from *Legend type*  'Unique value' and set the *Classification field* to 'VEGDESC'. (Refer to the explanation of the symbology tab in *Symbology Properties* of the vector section.)
- Next, reopen the GRASS Toolbox and open *Vector* ► *Vector update* by other maps.
- Click on the **v.rast.stats** module. Enter `gtopo30` and `forest_areas`.
- Only one additional parameter is needed: Enter *column prefix* `elev`, and click *Run*. This is a computationally heavy operation, which will run for a long time (probably up to two hours).
- Finally, open the `forest_areas` attribute table, and verify that several new columns have been added, including `elev_min`, `elev_max`, `elev_mean`, etc., for each forest polygon.

22.14.3 Customizing the GRASS Toolbox

Nearly all GRASS modules can be added to the GRASS Toolbox. An XML interface is provided to parse the pretty simple XML files that configure the modules' appearance and parameters inside the Toolbox.

A sample XML file for generating the module `v.buffer` (v.buffer.qgm) looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE qgisgrassmodule SYSTEM "http://mrcc.com/qgisgrassmodule.dtd">

<qgisgrassmodule label="Vector buffer" module="v.buffer">
        <option key="input" typeoption="type" layeroption="layer" />
```

(continues on next page)

(continued from previous page)

```
        <option key="buffer"/>
        <option key="output" />
</qgisgrassmodule>
```

The parser reads this definition and creates a new tab inside the Toolbox when you select the module.

CHAPTER

TWENTYTHREE

QGIS PROCESSING FRAMEWORK

23.1 Introduction

This chapter introduces the QGIS processing framework, a geoprocessing environment that can be used to call native and third-party algorithms from QGIS, making your spatial analysis tasks more productive and easy to accomplish.

As a *Core plugin*, Processing is installed by default but you need to activate it:

1. Go to *Plugins* ► *Manage and install plugins…*
2. Click on the *Installed* tab at the left
3. Check the box next to the  *Processing* entry
4. Close the dialog.

   A *Processing* menu is now available in the top menu bar. From there you can reach the main components of this framework.

In the following sections, we will review how to use the graphical elements of this framework and make the most out of each one of them.

There are four basic elements in the framework GUI, which are used to run algorithms for different purposes. Choosing one tool or another will depend on the kind of analysis that is to be performed and the particular characteristics of each user and project. All of them (except for the batch processing interface, which is called from the toolbox or the algorithm execution dialog, as we will see) can be accessed from the *Processing* menu item (you will see more entries; the remaining ones are not used to execute algorithms and will be explained later in this chapter).

- The *Toolbox*: The main element of the GUI, it is used to execute a single algorithm or run a batch process based on that algorithm.

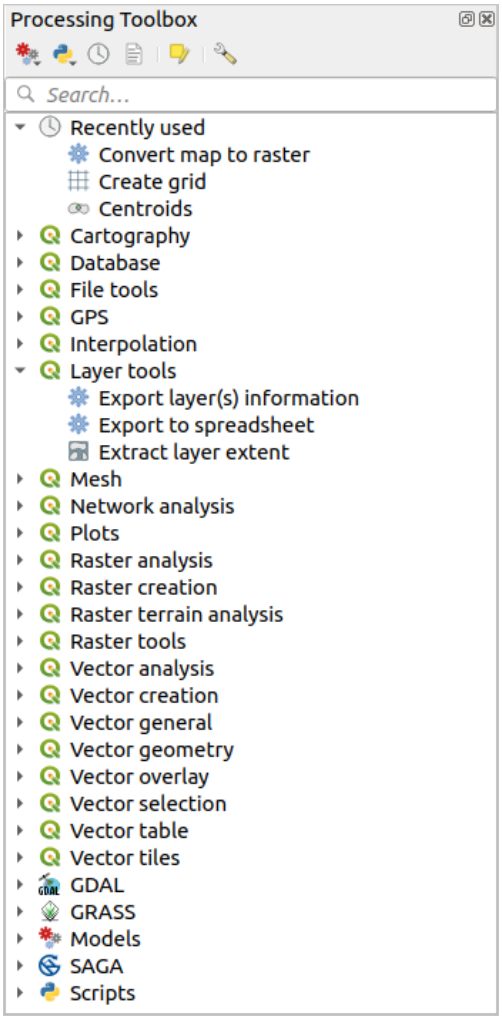


Fig. 23.1: Processing Toolbox

- The *Model Designer*: Several algorithms can be combined graphically using the modeler to define a workflow, creating a single process that involves several subprocesses.

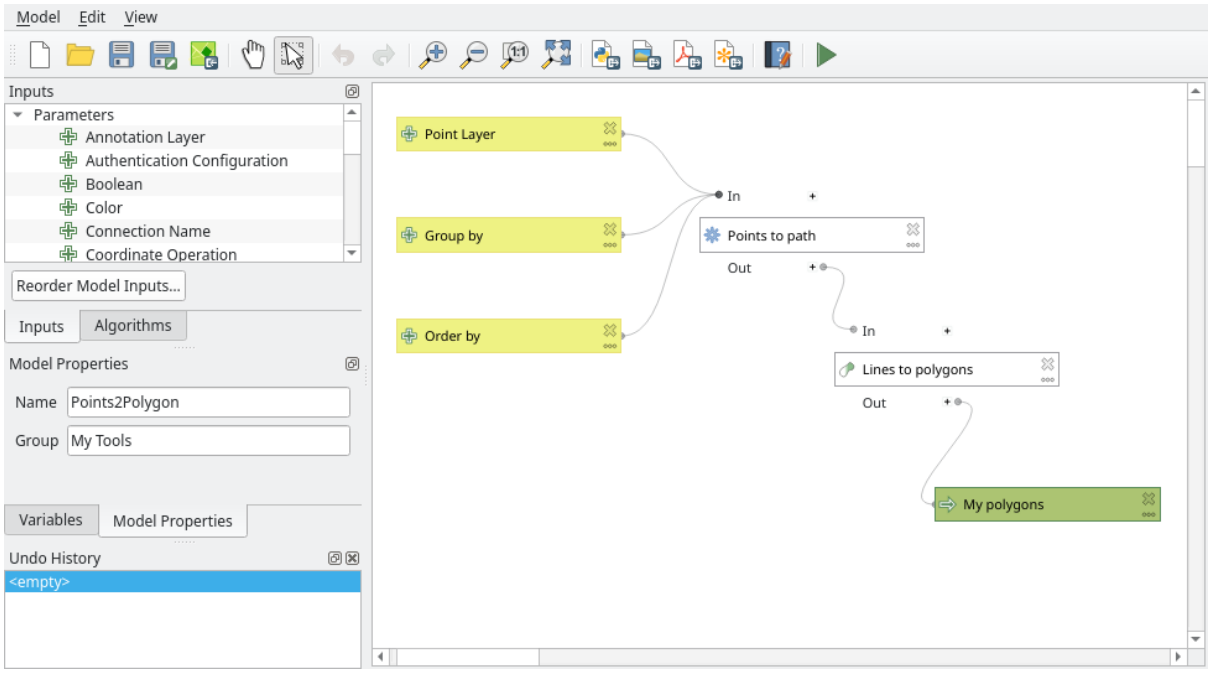


Fig. 23.2: Processing Modeler

- The *History* manager: All actions performed using any of the aforementioned elements are stored in a history file and can be later easily reproduced using the history manager.

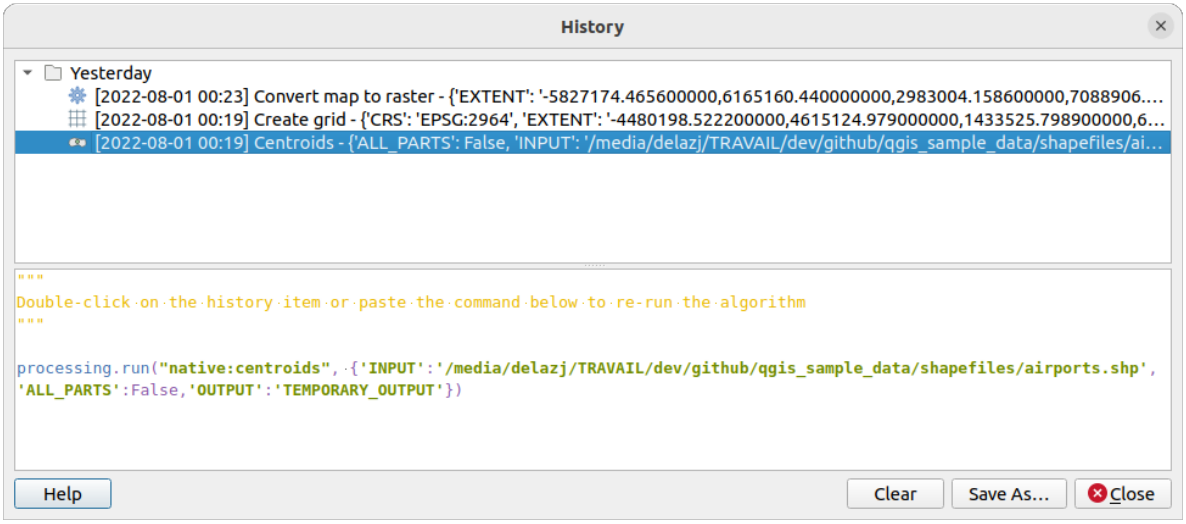


Fig. 23.3: Processing History

- The *Batch Processing* interface: This interface allows you to execute batch processes and automate the execution of a single algorithm on multiple datasets.

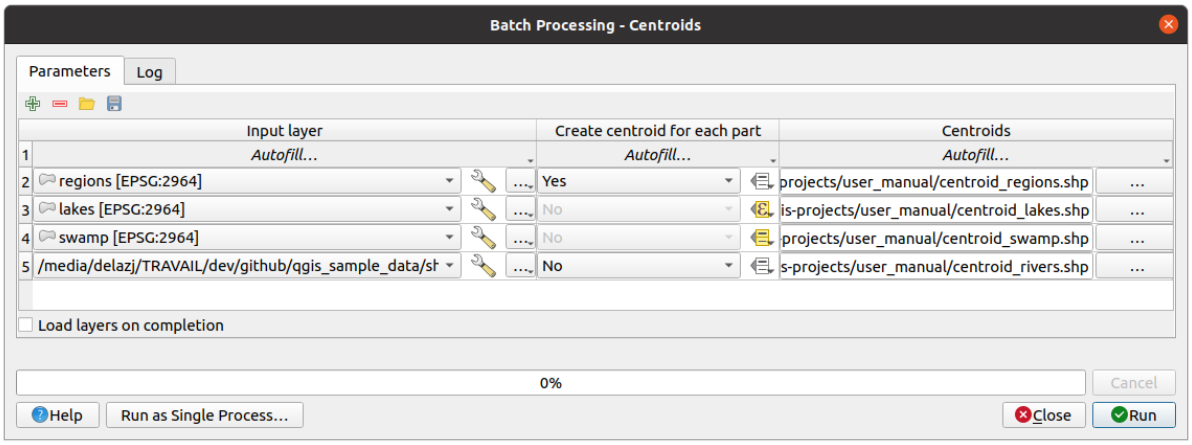


Fig. 23.4: Batch Processing interface

In the following sections, we will review each one of these elements in detail.

23.2 Configuring the Processing Framework

The Processing Options menu (*Settings*► *Options* ►  *Processing* tab) allows you to configure how algorithms work. Configuration parameters are structured in separate blocks that you can select on the left-hand side of the dialog.

23.2.1 General

The *General* block contains the default settings to control how the algorithm dialog as well as input or output parameters should behave. Some of the settings can however be overridden at the algorithm level, per *algorithm run* or for *individual parameters*.

- *Default output raster layer extension* is by default `tif`
- *Default output vector layer extension* is by default `gpkg`
- *Invalid features filtering* when executing algorithm:
  - *Do not filter (better performance)*: all the features (with valid and invalid geometries) are processed, but the result may be erroneous depending on how the geometry invalidity affects the operations
  - *Skip (ignore) features with invalid geometries*, meaning that only a subset of your dataset (the valid geometry features) will be processed
  - *Stop algorithm execution when a geometry is invalid*: you'll need to track and fix the invalid geometries if you want the algorithm to process the whole layer. Algorithms like *Check validity* or *Fix geometries* can help you achieve this.

  The *Invalid features filtering* setting can be overridden on a per-input basis, at algorithm runtime.
- *Keep dialog open after running algorithm*. Once an algorithm has finished execution and its output layers are loaded into the QGIS project, the algorithm dialog is closed. If you want to keep it open (to run the algorithm again with different parameters, or to better check the output that is written to the log tab), check this option.
- *Max Threads*
- *Output folder* for non temporary outputs: If no folder path is provided for the Processing execution outputs, this is the folder in which they will be saved. Default is `processing/outputs` under the active *user profile* directory.

- *Override temporary output folder path*: Temporary outputs are saved by default in the `tmp` folder on the machine. This option helps you set a different place for storage.
- *Pre-execution script* and *Post-execution script*. These parameters point to files that contain scripts written using the processing scripting functionality, explained in the section covering scripting and the console.
- *Prefer output filename for layer names*. The name of each resulting layer created by an algorithm is defined by the algorithm itself. In some cases, a fixed name might be used, meaning that the same output name will be used, no matter which input layer is used. In other cases, the name might depend on the name of the input layer or some of the parameters used to run the algorithm. If this checkbox is checked, the name will be taken from the output filename instead. Notice that, if the output is saved to a temporary file, the filename of this temporary file is usually a long and meaningless one intended to avoid collision with other already existing filenames.
- *Results group name*. If you want to obtain all processing result layers in a group in the *Layers* panel, set a group name for this parameter. The group may exist already or not. QGIS will add all output layers to such a group. By default, this parameter is empty, so all output layers are added to different places in the *Layers* panel, depending on the item that is active when running an algorithm. Note that output layers will be loaded to the *Layers* panel only if *Open output file after running algorithm* is checked in the algorithm dialog.
- *Show algorithms with known issues*: By default, QGIS avoids display of broken algorithms (generally from third-party providers). If checked, they will be available in the Processing toolbox, with a warning icon and a tooltip explaining they have issues. Use at your own risks.
- *Show feature count for output vector layers*. Since calculating the feature count might take some time in certain data formats, this option is off by default.
- *Show layer CRS definition in selection boxes*
- *Show tooltip when there are disabled providers*
- *Style for line layers*, *Style for point layers*, *Style for polygons layers* and *Style for raster layers* are used for setting the default rendering style for output layers (that is, layers generated by processing algorithms). Just create the style you want using QGIS, save it to a file, and then enter the path to that file in the settings so the algorithms can use it. Whenever a layer is loaded by Processing and added to the QGIS canvas, it will be rendered with that style.

  Rendering styles can be configured individually for each algorithm and each one of its outputs. Just right-click on the name of the algorithm in the toolbox and select *Edit rendering styles for outputs*. You will see a dialog like the one shown next.

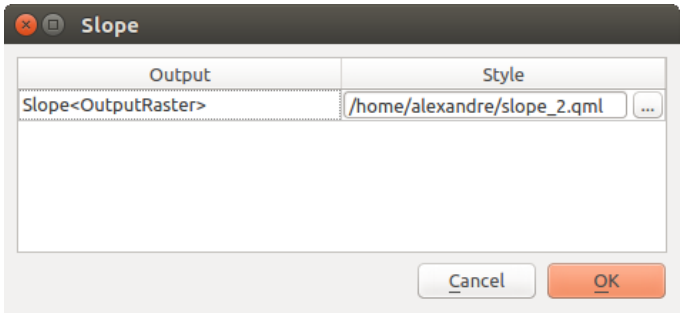


Fig. 23.5: Rendering Styles

  Select the style file (`.qml`) that you want for each output and press *OK*.
- *Warn before executing if parameter CRS's do not match*: By default, QGIS native algorithms (i.e. the ones listed under the *Menus* ► *QGIS (native C++)* group) transparently reproject the input layers to the first one's CRS before execution. Check this option to get a notification from the other tools that do not suppport reprojection, when the inputs CRS are not identical. Third-party providers are not concerned.

23.2.2 Menus

The  *Menus* block controls whether an algorithm, script or model (built-in or provided by plugins) should be made available through a dedicated menu or toolbar (along with the Processing Toolbox). For each item of each provider, you can:

- *Add button in toolbar*, making it available in the *Processing Algorithms* toolbar
- assign an *Icon* to the algorithm
- set a *Menu path*: the algorithm will then be available through an existing or a custom menu, e.g. `Vect&or/MyTopAlgorithms` will add a shortcut in a *MyTopAlgorithms* sub-menu of the default *Vector* menu.

Restart QGIS to apply the settings. At any time, your changes can be *Reset to defaults*.

23.2.3 Models and Scripts

In the  *Models* and  *Scripts* blocks, a default path, set under the active *User profile* folder, is provided for storing models and scripts respectively. As for the other options, you can modify the path, e.g., to lead to a remote or shared folder for all your users.

23.2.4 Providers

You will also find a block for algorithm  *Providers*. This is the place where installed providers expose their settings.

By default, QGIS is installed with two algorithm providers:

- *GDAL* whose algorithms you may *Activate* (or not) in this dialog. Read more about the *GDAL algorithms*.
- *GRASS* whose algorithms require the installation of the *GRASS GIS Processing Provider* from the *Plugins Manager*. You could then set some dedicated settings such as:
  -  *For raster layers, use r.external (faster) instead of r.in.gdal*
  -  *For vector layers, use v.external (faster) instead of v.in.ogr*
  - *Location of GRASS docs*: by default, and depending on your OS, QGIS utilizes a hardcoded list of paths for finding the local GRASS documentation and opening it whenever you hit the *Help* button of a GRASS algorithm dialog. Filling this option will override the default path, allowing you to use a custom location, or remote documentation.
  -  *Log console output*
  -  *Log execution commands*

Installed plugins that provide Processing algorithms may also have their provider listed in this group, with custom settings.

23.3 The Toolbox

The *Processing Toolbox* is the main element of the processing GUI, and the one that you are more likely to use in your daily work. It shows the list of all available **algorithms** grouped in different blocks called *Providers*, and custom **models** and **scripts** you can add to extend the set of tools. Hence the toolbox is the access point to run them, whether as a single process or as a batch process involving several executions of the same algorithm on different sets of inputs.

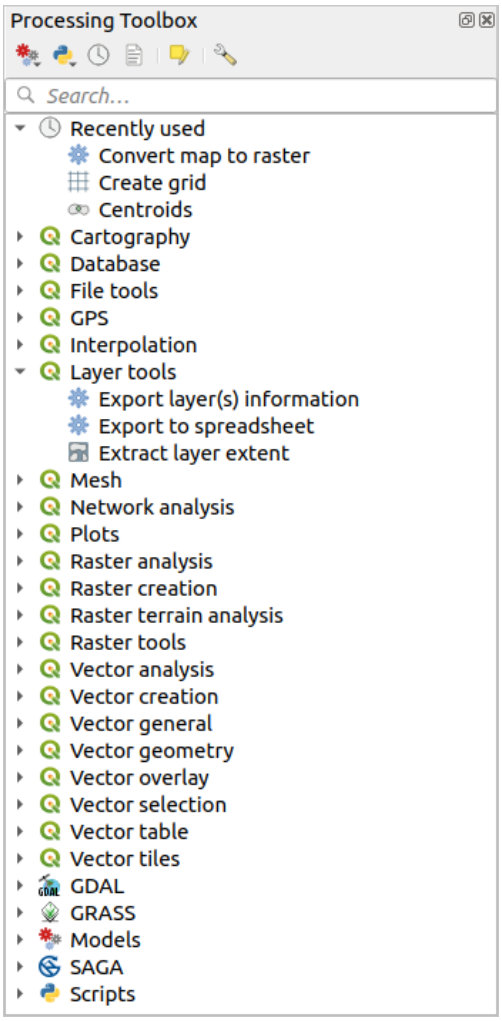


Fig. 23.6: Processing Toolbox

Providers can be (de)activated in the *Processing settings dialog*. By default, only providers that do not rely on third-party applications (that is, those that only require QGIS elements to be run) are active. Algorithms requiring external applications might need additional configuration. Configuring providers is explained in a *later chapter* in this manual.

In the upper part of the toolbox dialog, you will find a set of tools to:

- work with Models: *Create New Model…*, *Open Existing Model…* and *Add Model to Toolbox…*;
- work with Scripts: *Create New Script…*, *Create New Script from Template…*, *Open Existing Script…* and *Add Script to Toolbox…*;
- open the History panel;
- open the Results Viewer panel;

- toggle the toolbox to the *in-place modification mode* using the  Edit Features In-Place button: only the algorithms that are suitable to be executed on the active layer without outputting a new layer are displayed;
- open the  Options dialog.

Below this toolbar is a  *Search…* box to help you easily find the tools you need. You can enter any word or phrase on the text box. Notice that, as you type, the number of algorithms, models or scripts in the toolbox is reduced to just those that contain the text you have entered in their names or keywords.

The top of the algorithm list shows the most  Recently Used tools and the  Favorites tools, making it convenient to reexecute them. The  Recently Used list is there by default, showing the algorithms you have recently executed. The  Favorites list is created when you manually add an algorithm to it.

You can right-click on any algorithm in the toolbox and choose from the contextual menu to:

- *Execute…* to run the algorithm.
- *Execute as Batch Process…* to run the algorithm in batch mode.
- *Edit Rendering Styles for Outputs…* to customize the rendering styles for the algorithm's outputs.
- *Add to Favorites* to add the algorithm to the Favorites list.
- *Remove from Favorites* to remove the algorithm from the Favorites list.

Unlike the Recently Used list, the Favorites list remains static and is not affected by which algorithms were executed.

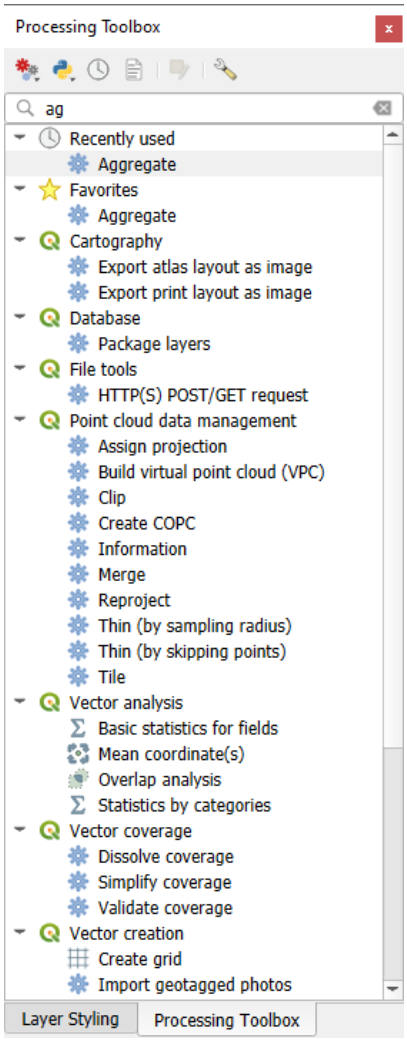


Fig. 23.7: Processing Toolbox showing search results

To execute a tool, just double-click on its name in the toolbox.

23.3.1 The algorithm dialog

Once you double-click on the name of the algorithm that you want to execute, a dialog similar to that in the Fig. 23.8 below is shown (in this case, the dialog corresponds to the `Centroids` algorithm). Dialog title will include the group name from which the algorithm originates.

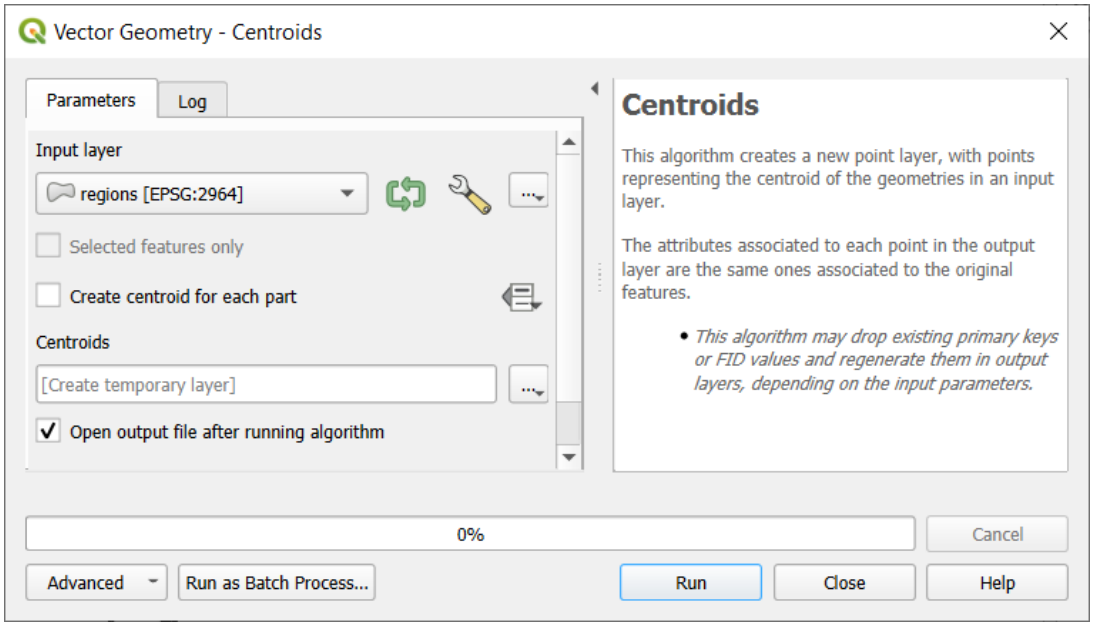


Fig. 23.8: Algorithm Dialog - Parameters

The dialog shows two tabs (*Parameters* and *Log*) on the left part, the algorithm description on the right, and a set of buttons at the bottom.

Parameter types

The *Parameters* tab is used to set the input values that the algorithm needs to be executed. It shows a list of input values and configuration parameters to be set. It of course has a different content, depending on the requirements of the algorithm to be executed, and is created automatically based on those requirements.

> **Tip**
>
> **Setting your own default values for algorithm parameters**
>
> Algorithm dialogs open with some parameters prefilled with values from QGIS installation. It is however possible to set *your own default values* for specific algorithm parameters so that they are used at algorithm startup.

Although the number and type of parameters depend on the characteristics of the algorithm, the structure is similar for all of them. The parameters found in the table can be of one of the following types.

- A **vector layer**, to select from a list of all vector layers available (currently opened) in QGIS. You can also use unloaded layers: press the … button on the widget right-hand side, and select:
  - *Select file…*: selects file on disk using the Operating System file explorer
  - *Browse for layer…*: opens the *Browser panel*, allowing to take the layers directly from database sources (PostgreSQL, SQL Server, Oracle, …), web services (WFS, AFS, …) or files on disk.



Fig. 23.9: Vector input widget

> **Tip**
>
> **Drag-and drop as convenient alternative for layer selection**
>
> Besides selecting from the drop-down menu or browsing sources, you can also drag and drop a layer from the *Layers* or *Browser* panel, or from the Operating System file explorer directly into the combobox.

> **Note**
>
> By default, the layer widget shows the CRS of the layer along with its name. If you do not want to see this additional information, you can disable this functionality in the Processing Settings dialog, unchecking the *General* ► *Show layer CRS definition in selection boxes* option.

The vector input widget also has following features:

- an iterator button: If toggled, the algorithm will be executed iteratively on each one of its features, instead of just once for the whole layer, producing as many outputs as times the algorithm is executed. This allows for automating the process when all features in a layer have to be processed separately. If the algorithm contains several input vectors you can iterate over, the iteration will be processed only on the first toggled parameter, in the order parameters are declared in the algorithm.
- Advanced options button to adjust settings to use for that specific parameter. These settings concern:
  - *Invalid feature filtering*: allows the *default method* for handling features with invalid geometries to be overridden
  - *Limit features processed*: optional limit on number of features processed from the source
  - *Feature filter*: allows to enter an expression to subset the layer dynamically when running the tool, avoiding the need for separate steps to set layer filters or create layer subsets.

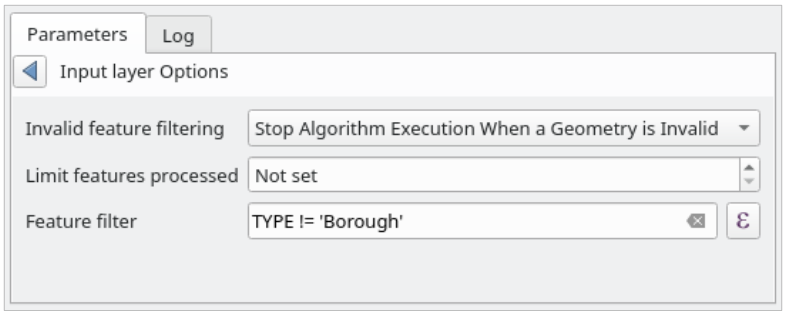


Fig. 23.10: Advanced options for vector input widget

  - It is also possible to limit the algorithm execution on the vector layer to its *Selected features only*.
- A **table**, to select from a list of all available in QGIS. Non-spatial tables are loaded into QGIS like vector layers, and use the *same widget*.
- A **raster layer**, to select from a list of all raster layers available in QGIS. The selector contains as well a … button on its right-hand side, to let you select filenames that represent layers currently not loaded in QGIS.

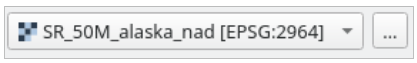


Fig. 23.11: Raster input widget

- An **option**, to choose from a selection list of possible options.

- A **numerical value**, to be introduced in a spin box. In some contexts (when the parameter applies at the feature level and not at the layer's), you will find a Data-defined override button by its side, allowing you to open the *expression builder* and enter a mathematical expression to generate variable values for the parameter. Some useful variables related to data loaded into QGIS can be added to your expression, so you can select a value derived from any of these variables, such as the cell size of a layer or the northernmost coordinate of another one.

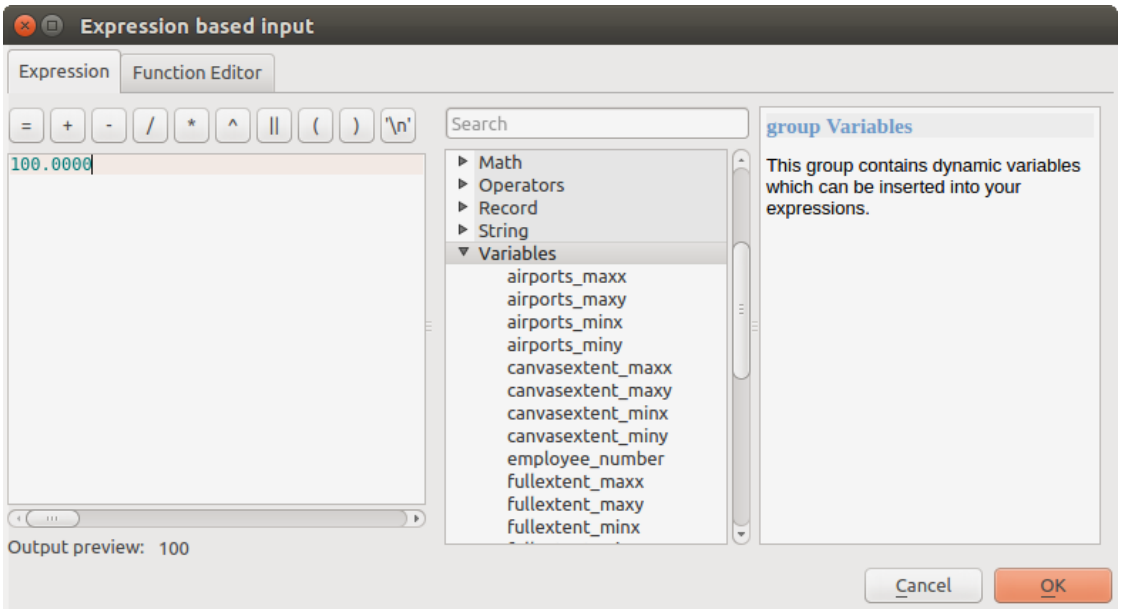


Fig. 23.12: Expression based input

- A **range**, with min and max values to be introduced in two text boxes.
- A **text string**, to be introduced in a text box.
- A **field**, to choose from the attributes table of a vector layer or a single table selected in another parameter.
- A **coordinate reference system**. You can select it among the recently used ones from the drop-down list or from the *CRS selection* dialog that appears when you click on the button on the right-hand side.

> **Tip**
>
> **Drag-and-drop as convenient shortcut for CRS selection**
>
> You can also drag a layer from the Layers panel and drop it onto the CRS widget. The widget will automatically set its CRS to match the dropped layer's.

- An **extent**, a text box defining a rectangle through its corners coordinate in the format `xmin, xmax, ymin, ymax`. Press the Set to current map canvas extent button to use the map canvas extent. Clicking the arrow on the right-hand side of the value selector, a pop-up menu will appear, giving you options to:
  - *Calculate from layer* ►: fills the text box with the coordinates of the bounding box of a layer to select among the loaded ones
  - *Calculate from layout map* ►: fills the text box with the coordinates of a map item selected from a layout in the current project
  - *Calculate from bookmark* ►: fills the text box with the coordinates of a saved bookmark
  - *Use current map canvas extent*

– *Draw on canvas*: the parameters window will hide itself, so you can click and drag onto the canvas. Once you have defined the extent rectangle, the dialog will reappear, containing the values in the extent text box.

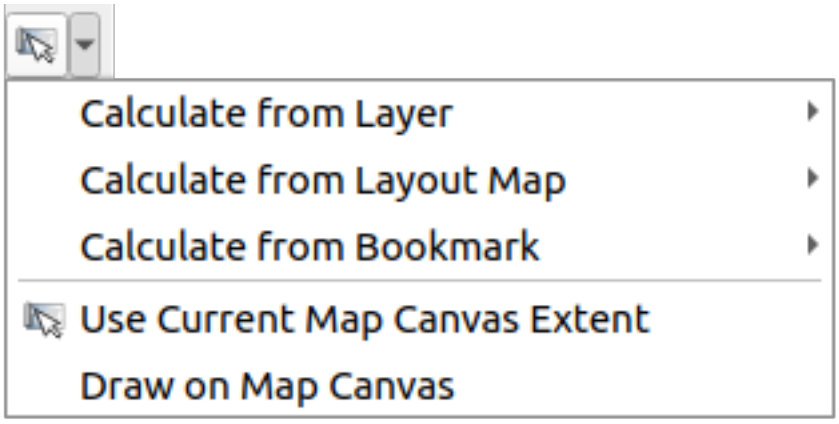


Fig. 23.13: Extent selector

- A **list of elements** (whether raster or vector layers, tables, fields) to select from. Click on the … button at the left of the option to see a dialog like the following one. Multiple selection is allowed and when the dialog is closed, number of selected items is displayed in the parameter text box widget.

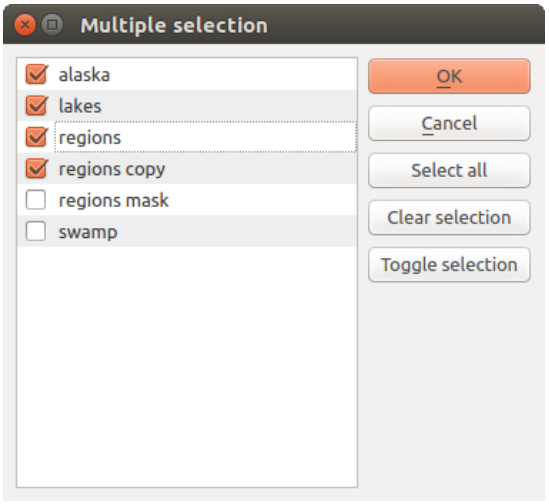


Fig. 23.14: Multiple Selection

- A **small table** to be edited by the user. These are used to define parameters like lookup tables or convolution kernels, among others.

  Click on the button on the right side to see the table and edit its values.

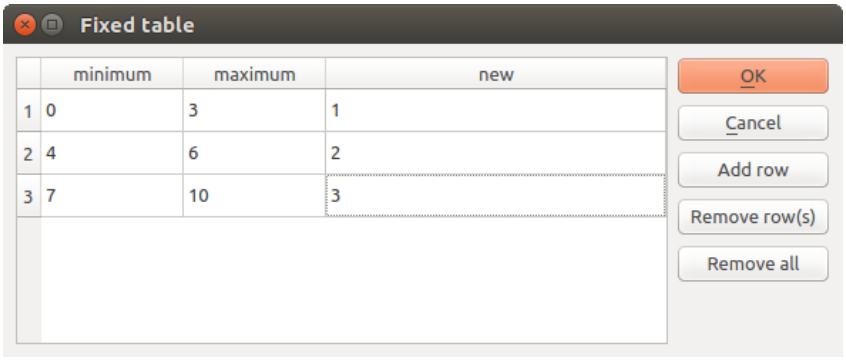


Fig. 23.15: Fixed Table

Depending on the algorithm, the number of rows can be modified or not by using the buttons on the right side of the window.

Note

Some algorithms require many parameters to run, e.g. in the *Raster calculator* you have to specify manually the cell size, the extent and the CRS. You can avoid to choose all the parameters manually when the algorithm has the `Reference layers` parameter. With this parameter you can choose the reference layer and all its properties (cell size, extent, CRS) will be used.

Logging the execution

Along with the *Parameters* tab, there is another tab named *Log* (see Fig. 23.16 below). Information provided by the algorithm during its execution is written in this tab, allowing you to track the execution as well as being aware and having more details about the algorithm as it runs. You can directly click on the names of output files, folders, or HTML files listed in the log. Doing so will open the folder containing the generated file and automatically select it. Information on algorithm execution is also output in the *View* ► *Panels* ► *Log Messages Panel*.

Notice that not all algorithms write information to the *Log* tab, and many of them might run silently without producing any output other than the final files. Check the *Log Messages Panel* in that case.

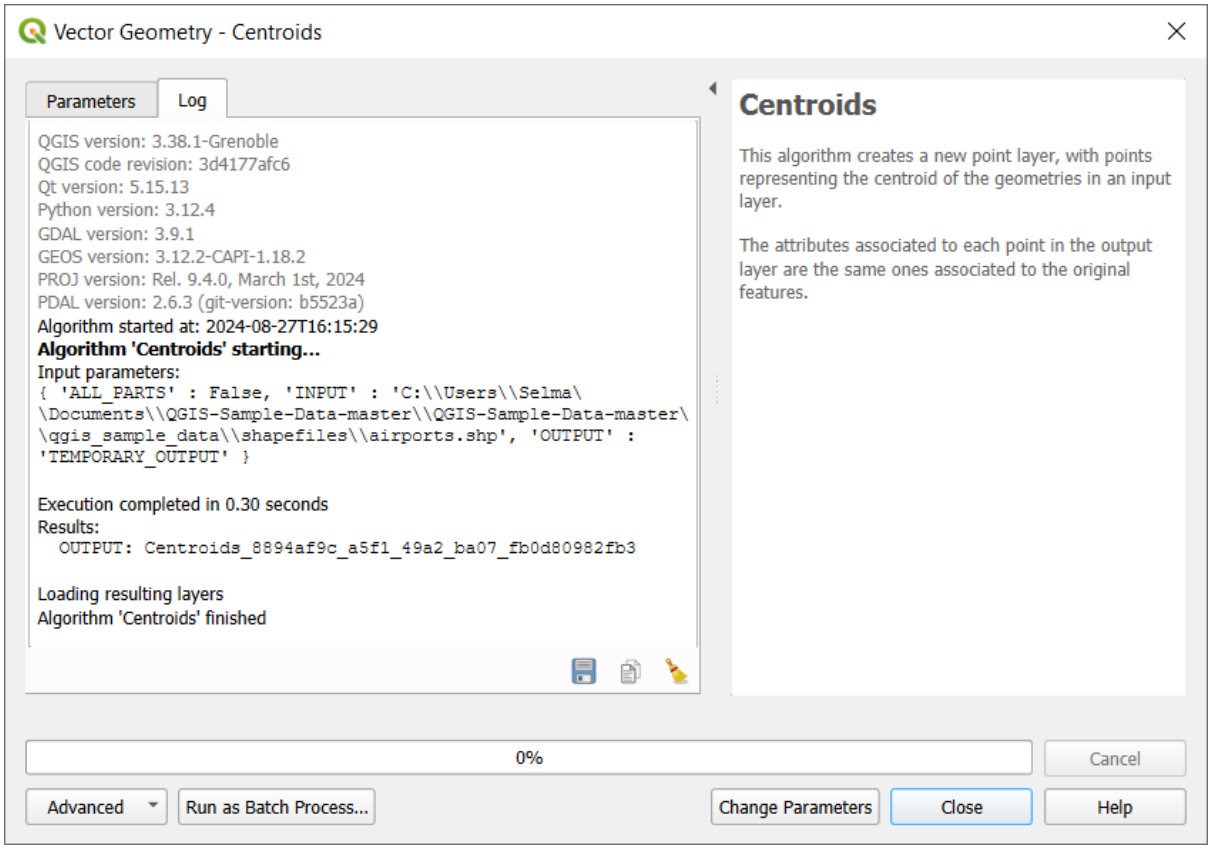


Fig. 23.16: Algorithm Dialog - Log

At the bottom of the *Log* tab you will find buttons to *Save Log to File*, *Copy Log to Clipboard* and *Clear Log*. These are particularly handy when you have checked the *Keep dialog open after running algorithm* in the *General* part of the Processing options.

Other tools

On the right hand side of the dialog you will find a short description of the algorithm, which will help you understand its purpose and its basic ideas. If such a description is not available, the description panel will not be shown.

For a more detailed help file, which might include description of every parameter it uses, or examples, you will find a *Help* button at the bottom of the dialog bringing you to the *Processing algorithms documentation* or to the provider documentation (for some third-party providers).

The *Advanced* ► menu provides functions to reuse the configuration defined in the dialog without running the algorithm:

- *Algorithm Settings...*: allows to override processing settings for the current algorithm execution. More details at *Override algorithm settings*.
- *Copy as Python Command*: allows for easy copying of the equivalent *PyQGIS command* to run the tool using the parameters defined in the dialog
- *Copy as qgis_process Command*: allows for easy generation of *qgis_process command*, including its environment settings like the distance units, area units, ellipsoid, and any tricky parameter values like GeoPackage outputs with specific layers
- *Copy as JSON*: all the settings of the command are copied in a `JSON` format, ready to be consumed by qgis_process. This is a convenient way to see the expected format of the commands, even for complex

parameters (like TIN interpolation parameters). You can store these easily and then restore them later by pasting the values.

-  *Paste Settings* in a `JSON` format

The *Run as Batch Process…* button triggers the *batch processing mode* allowing to configure and run multiple instances of the algorithm with a variety of parameters. A *Run as Single Process…* button helps you switch back from the batch mode.

When an algorithm execution finishes (either successfully or not), a new button *Change Parameters* is shown as long as the *Log* tab is active.

Override algorithm settings

Triggered from within the *Advanced* drop-down menu at the bottom of an algorithm dialog, the  *Algorithm Settings…* shows a panel allowing users to control general processing settings which apply to that algorithm execution only. It is intended to be a place where a user can override their *global processing settings* on an ad-hoc basis without having to change their usual default settings.

Settings that can be overridden are:

- *Invalid feature filtering*: unlike the existing per-parameter setting override for this, setting the handling method here will apply to **ALL inputs** for the algorithm
- *Calculation settings*, such as *Distance units* and *Area units* to use for distance/area measurements
- *Environment settings*, such as *Temporary folder* and *Number of threads to use*

A note on projections

Processing algorithm execution are always performed in the input layer coordinate reference system (CRS). Due to QGIS's on-the-fly reprojecting capabilities, although two layers might seem to overlap and match, that might not be true if their original coordinates are used without reprojecting them onto a common coordinate system. Whenever you use more than one layer as input to a *QGIS native algorithm*, whether vector or raster, the layers will all be reprojected to match the coordinate reference system of the first input layer.

This is however less true for most of the external applications whose algorithms are exposed through the processing framework as they assume that all of the layers are already in a common coordinate system and ready to be analyzed.

By default, the parameters dialog will show a description of the CRS of each layer along with its name, making it easy to select layers that share the same CRS to be used as input layers. If you do not want to see this additional information, you can disable this functionality in the Processing settings dialog, unchecking the *Show layer CRS definition in selection boxes* option.

If you try to execute an algorithm using as input two or more layers with unmatching CRSs, a warning dialog will be shown. This occurs thanks to the *Warn before executing if layer CRS's do not match* option.

You still can execute the algorithm, but be aware that in most cases that will produce wrong results, such as empty layers due to input layers not overlapping.

> **Tip**
>
> **Use Processing algorithms to do intermediate reprojection**
>
> When an algorithm can not successfully perform on multiple input layers due to unmatching CRSs, use QGIS internal algorithm such as *Reproject layer* to perform layers' reprojection to the same CRS before executing the algorithm using these outputs.

23.3.2 Data objects generated by algorithms

Data objects generated by an algorithm can be of different types. They can be a layer (vector with or without geometry, raster, mesh, point cloud, ...), stored as a plain file on disk or in a database, text-based files such as `qml` style file or graphics `.html` file, a folder, ...

The parameters dialog will contain a widget corresponding to the needed output, where you can type the output channel to use for saving it. An output channel contains the information needed to save the resulting object somewhere. In the most usual case, you will save it to a file, but in the case of vector layers, and when they are generated by native algorithms (algorithms not using external applications) you can also save to a PostgreSQL, GeoPackage or SpatiaLite database, or a memory layer.

The **output** parameter for setting the result of the algorithm execution provides a text box for filling the path to the output result, and a drop-down ... button for additional options (the availability depends on the algorithm you run):

- *Skip Output*: if you are not interested in a given output for this parameter
- *Create Temporary Layer* (`TEMPORARY_OUTPUT`): the output is stored in a vector *temporary scratch layer*.
- *Save to File...*: you will be prompted with a save file dialog, where you can select the desired file path. Supported file extensions are shown in the file format selector of the dialog, depending on the kind of output and the algorithm.

  The format of the output is defined by the filename extension. The supported formats depend on what is supported by the algorithm itself. To select a format, just select the corresponding file extension (or add it, if you are directly typing the file path instead). If the extension of the file path you entered does not match any of the supported formats, a default extension will be appended to the file path, and the file format corresponding to that extension will be used to save the layer or table. Default extensions are `.dbf` for tables, `.tif` for raster layers and `.gpkg` for vector layers. These can be modified in the setting dialog, selecting any other of the formats supported by QGIS.

  You can set a default folder for output data objects. Go to the settings dialog (you can open it from the *Settings* ► *Options* ► *Processing* menu), and in the *General* group, you will find a parameter named *Output folder*. This output folder is used as the default path in case you type just a filename with no path (i.e., `myfile.shp`) when executing an algorithm.
- *Save to a Temporary File*: if you do not enter any filename in the output text box (or select the corresponding option in the context menu), the result will be saved as a temporary file in the corresponding default file format, and it will be deleted once you exit QGIS (take care with that, in case you save your project and it contains temporary layers). The file is saved in the default``*TEMP*` folder on the machine, if *not overridden*.
- *Save to Geopackage...*: the output layer will be saved as a named table of a new or existing GeoPackage database
- *Save to Database Table...*: the output layer will be saved as a new table in a database such as GeoPackage, SpatiaLite, PostgreSQL or MS SQL Server. A connection should already exist. You'll need to provide the name of the new table and depending on the type of target database, select a schema,
- *Append to Layer...*: adds features output by the algorithm to an existing layer. Upon selecting the target layer, a field mapping panel opens, allowing to manually set how fields from the output layer are mapped to the target layer's fields.

- *Save to Directory*: Use a new or existing directory for storing the algorithm outputs
- *Save to a Temporary Directory*: creates a directory within the set `TEMP` folder for storing the algorithm outputs. The files will be deleted once you exit QGIS.

When running an algorithm that uses a vector layer in iterative mode, the entered file path is used as the base path for all generated files, which are named using the base name and appending a number representing the index of the iteration. The file extension (and format) is used for all such generated files.

Algorithms also generate graphics and text as HTML files. These results are shown at the end of the algorithm execution in the *Results Viewer* dialog. This dialog will keep the results produced by any algorithm during the current session, and can be shown at any time by selecting *Processing* ► *Results Viewer* from the QGIS main menu.

For layer-based outputs, it is possible to *Open output file after running algorithm* directly in QGIS, as a new entry in the *Layers* panel. By default, the files are opened.

23.4 The history manager

23.4.1 The processing history

Every time you execute an algorithm, information about the process is stored in the history manager. The date and time of the execution are saved, along with the parameters used, making it is easy to track and control all the work that has been developed using the Processing framework, and to reproduce it.

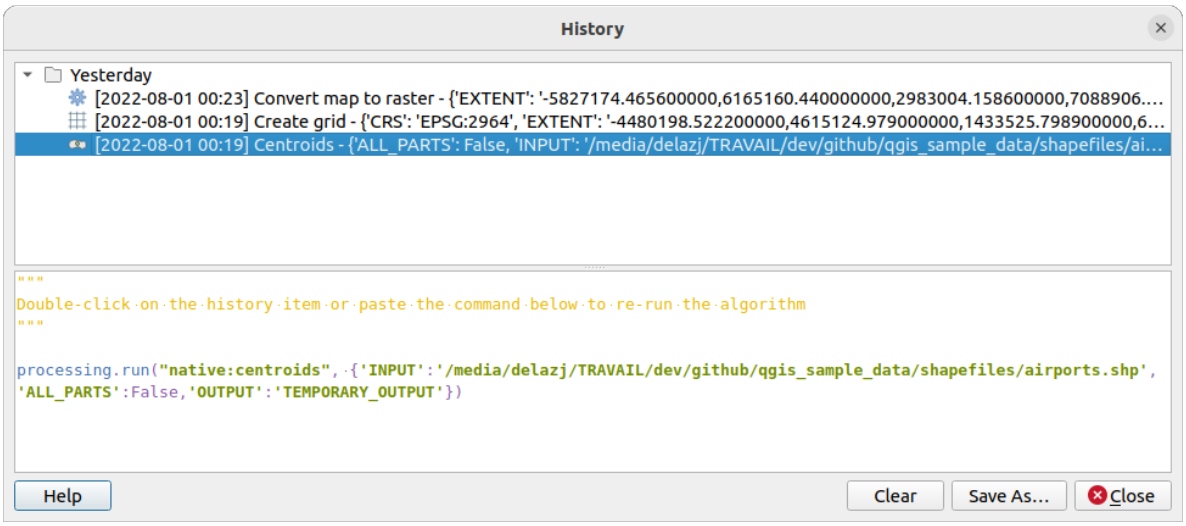


Fig. 23.17: History

Process information is kept as a command-line expression, even if the algorithm was launched from the toolbox. This makes it useful for those learning how to use the command-line interface, since they can call an algorithm using the toolbox and then check the history manager to see how it could be called from the command line.

Right-click on a row and you should be able to:

- *Copy as Python Command*: allows for easy copying of the equivalent *PyQGIS command* run from the dialog. Same as the code displayed below the commands list.
- *Copy as qgis_process Command*: allows for easy generation of *qgis_process command*, including its environment settings like the distance units, area units, ellipsoid, and any tricky parameter values like GeoPackage outputs with specific layers
- *Copy as JSON*: all the settings of the command are copied in a `JSON` format, ready to be consumed by qgis_process. This is a convenient way to see the expected format of the commands, even for complex parameters (like TIN interpolation parameters). You can store these easily and then restore them later by pasting the values over an algorithm dialog.
- *Create Test...* using the concerned algorithm and parameters, following instructions at Processing README file.

Apart from browsing the entries in the registry, you can also re-execute processes by simply double-clicking on the entry. The algorithm dialog then opens with parameters already set, and you can change any of them to fit your needs and re-run the algorithm.

The *History* dialog also provides a convenient way to contribute to the consolidation of the testing infrastructure of QGIS Processing algorithms and scripts.

23.4.2 The processing log

The history dialog only contains the execution calls, but not the information produced by the algorithm when executed. That information is written to the QGIS log (*View* ► *Panels* ► *Log Messages Panel*).

Third-party algorithms are usually executed by using their command-line interfaces, which communicate with the user via the console. Although that console is not shown, usually a full dump of it is written to the log each time you run one of those algorithms. To avoid cluttering the log with that information, you can disable it for each provider in the settings dialog.

Some algorithms, even if they can produce a result with the given input data, output comments or additional information to log when they detect potential problems with the data, in order to warn you. Make sure you check those messages in the log if you get unexpected results.

23.5 The model designer

The *model designer* allows you to create complex models using a simple and easy-to-use interface. When working with a GIS, most analysis operations are not isolated, rather part of a chain of operations. Using the model designer, that chain of operations can be wrapped into a single process, making it convenient to execute later with a different set of inputs. No matter how many steps and different algorithms it involves, a model is executed as a single algorithm, saving time and effort.

The model designer can be opened from the Processing menu (*Processing* ► *Model Designer*).

23.5.1 The model designer interface

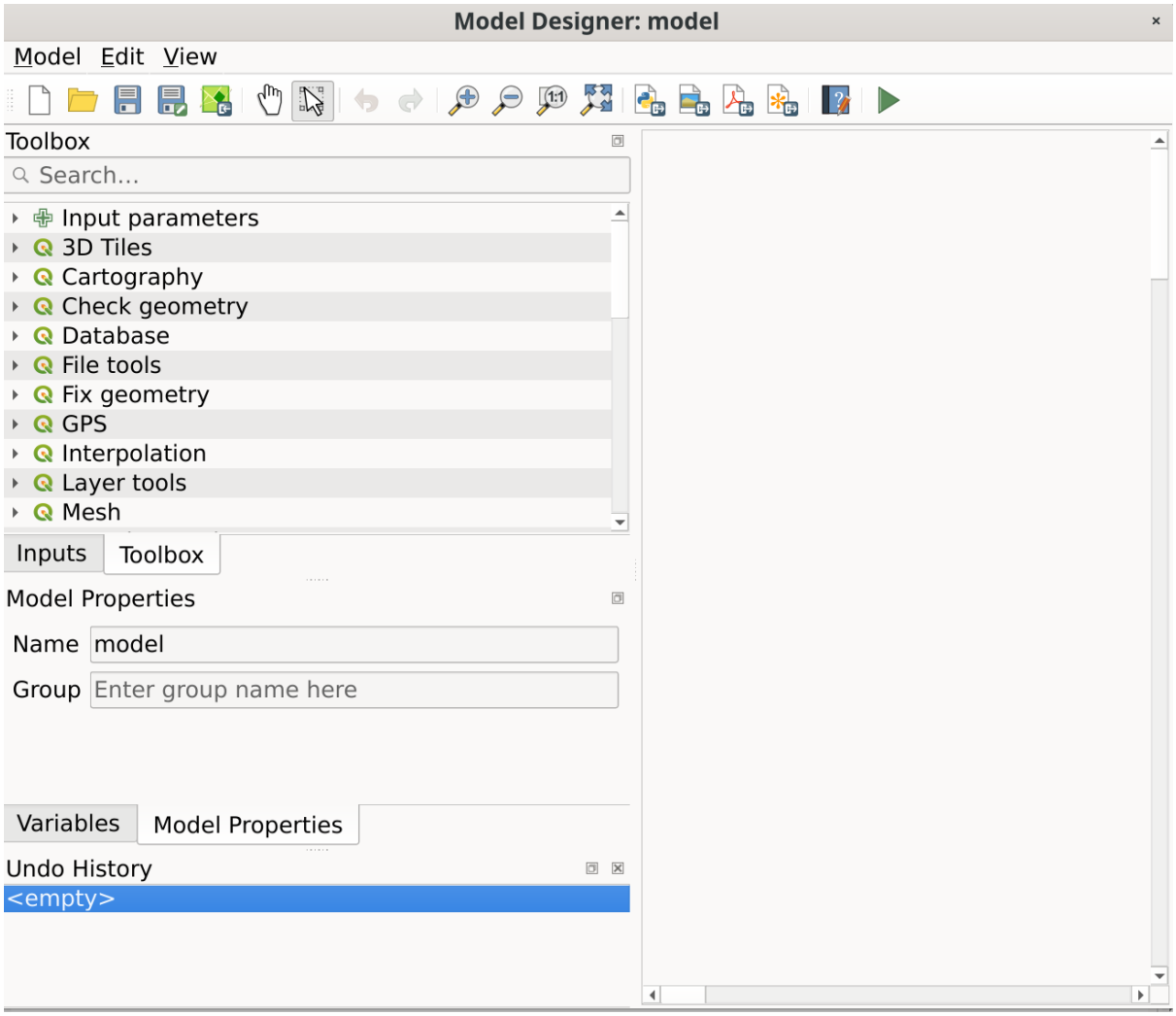


Fig. 23.18: Model designer

In its main part, the modeler has a working canvas where the structure of the model and the workflow it represents can be constructed.

At the top of the dialog, different menus and the *Navigation* toolbar give access to a variety of tools.

Model menu

Label	Shortcut	Navigation Toolbar	Description
Validate Model			Checks whether the algorithms and inputs used in the model exist. Convenient before releasing a model.
Run Model…	F5	☑	Executes the model
Run Selected Steps…	Shift+F5		Runs only the selected steps in a model, allowing to run a subset of the model
Reorder Model Inputs…			Sets the order in which inputs are presented to the user in the algorithm dialog.
Reorder Output Layers…			Sets a specific order which the outputs from the model must use when loading the results into a project.
Open Model…	Ctrl+O	☑	Opens a `.model3` file for edit or execution
Save Model	Ctrl+S	☑	Saves the model to disk as a `.model3` file
Save Model as…	Ctrl+Shi	☑	Saves the model to disk as a new `.model3` file
Save Model in project		☑	Embeds the model file in the project file, making it available when sharing the project file.
Edit Model Help…		☑	An interface to document the model, the algorithms, the parameters and outputs, as well as the author and versioning
Export ►			
► *Export as Image…*		☑	Saves the model's graphical design to an image file format (for illustration purpose)
► *Export as PDF…*			Saves the model's graphical design to a `PDF` file format (for illustration purpose)
► *Export as SVG…*			Saves the model's graphical design to an `SVG` file format (for illustration purpose)
► *Export as Script Algorithm…*		☑	Generates a python script file including the model's instructions

Edit menu

Label	Shortcut	Navigation Toolbar	Description
Select All	Ctrl+A		Selects all the model components in the designer
Snap selected components to Grid			snaps and aligns the elements into a grid
Redo	Ctrl+Y	☑	Rollback the latest canceled action. See also the *Undo/Redo* panel.
Undo	Ctrl+Z	☑	Cancel the previous change. See also the *Undo/Redo* panel.
Cut	Ctrl+X		Cuts a selection of components from the model.
Copy	Ctrl+C		Copies a selection of components from the model.
Paste	Ctrl+V		Pastes a cut or copied selection of components from a model to another or within the same model. The selected components keep their original properties and comments.
Delete selected components	Del		Removes a component from the model.
Add Group Box			Adds a box at the background of related components in order to visually group them. Particularly useful in big models to keep the workflow clean.

View menu

Label	Shortcut	Navigation Toolbar	Description
Zoom To ►			Zooms to the selected group box extent
Zoom In	Ctrl++	☑	
Zoom Out	Ctrl+-	☑	
Zoom to 100%	Ctrl+1	☑	
Zoom Full	Ctrl+0	☑	Displays all the components in the designer current canvas
Show Comments			Displays comments associated to every algorithm or input in the model designer
Enable Snapping			
Toggle Panel Visibility	Ctrl+Tab		Switches ON or OFF the *panels* in the designer

Panels

The left part of the window is a section with five panels that can be used to add new elements to the model:

1. *Inputs*: all the *input parameters* that could shape your model
2. *Toolbox*: here you can find available *Processing algorithms*. It also includes input parameters. Use *Search…* box to find algorithms or inputs.
3. *Model Properties*: specify the name (required) of the model and the group in which it will be displayed in the *Processing Toolbox*
4. *Variables*: Models can contain dedicated *variables* that are unique and only available to them. These variables can be accessed by any expression used within the model. They are useful to control algorithms within a model and control multiple aspects of the model by changing a single variable. The variables can be viewed and modified in the *Variables* panel.
5. *Undo History*: this panel will register everything that happens in the modeler, making it easy to cancel things you did wrong.

About available algorithms

Some algorithms that can be executed from the *Processing* ► *Toolbox* do not appear in the list of available algorithms when you are designing a model. To be included in a model, an algorithm must have the correct semantic. If an algorithm does not have such a well-defined semantic (for instance, if the number of output layers cannot be known in advance), then it is not possible to use it within a model, and it will not appear in the list of algorithms that you can find in the modeler dialog. On the other hand some algorithms are specific to the modeler. Those algorithms are located within the group 'Modeler Tools'.

23.5.2 Creating a model

Creating a model involves two basic steps:

1. *Definition of necessary inputs*. These inputs will be added to the parameters window, so the user can set their values when executing the model. The model itself is an algorithm, so the parameters window is generated automatically as for all algorithms available in the Processing framework.
2. *Definition of the workflow*. Using the input data of the model, the workflow is defined by adding algorithms and selecting how they use the defined inputs or the outputs generated by other algorithms in the model.

Definition of inputs

The first step is to define the inputs for the model. They are found in the *Inputs* panel on the left side of the modeler window. Hovering with the mouse over the inputs will show a tooltip with additional information. For a full list of available parameters in modeler and their correspondance for scripting, please read *Input and output types for Processing Algorithms*.

When double-clicking on an element, a dialog is shown that lets you define its characteristics. Depending on the parameter, the dialog will contain at least one element (the description, which is what the user will see when executing the model). For example, when adding a numerical value, as can be seen in the next figure, in addition to the description of the parameter, you have to set a default value and the range of valid values.

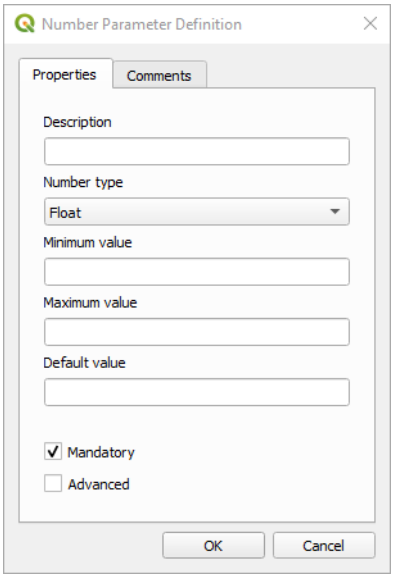


Fig. 23.19: Model Parameters Definition

You can define your input as mandatory for your model by checking the Mandatory option and by checking the Advanced checkbox you can set the input to be within the Advanced section. This is particularly useful when the model has many parameters and some of them are not trivial, but you still want to choose them.

For each added input, a new element is added to the modeler canvas.



Fig. 23.20: Model Parameters

You can also add inputs by dragging the input type from the list and dropping it at the position where you want it in the modeler canvas. If you want to change a parameter of an existing input, just double click on it, and the same dialog will pop up.

When using a model within another model, the inputs and outputs necessary will be displayed in the canvas.

Definition of the workflow

In the following example we will add two inputs and two algorithms. The aim of the model is to copy the elevation values from a DEM raster layer to a line layer using the Drape algorithm, and then calculate the total ascent of the line layer using the Climb Along Line algorithm.

In the *Inputs* tab, choose the two inputs as Vector Layer for the line and Raster Layer for the DEM. We are now ready to add the algorithms to the workflow.

Algorithms can be found in the *Toolbox* panel, grouped much in the same way as they are in the Processing toolbox.



Fig. 23.21: Model Inputs

To add an algorithm to a model, double-click on its name or drag and drop it, just like for inputs. As for the inputs you can change the description of the algorithm and add a comment. When adding an algorithm, an execution dialog

will appear, with a content similar to the one found in the execution panel that is shown when executing the algorithm from the toolbox. The following picture shows both the `Drape (set Z value from raster)` and the `Climb along line` algorithm dialogs.

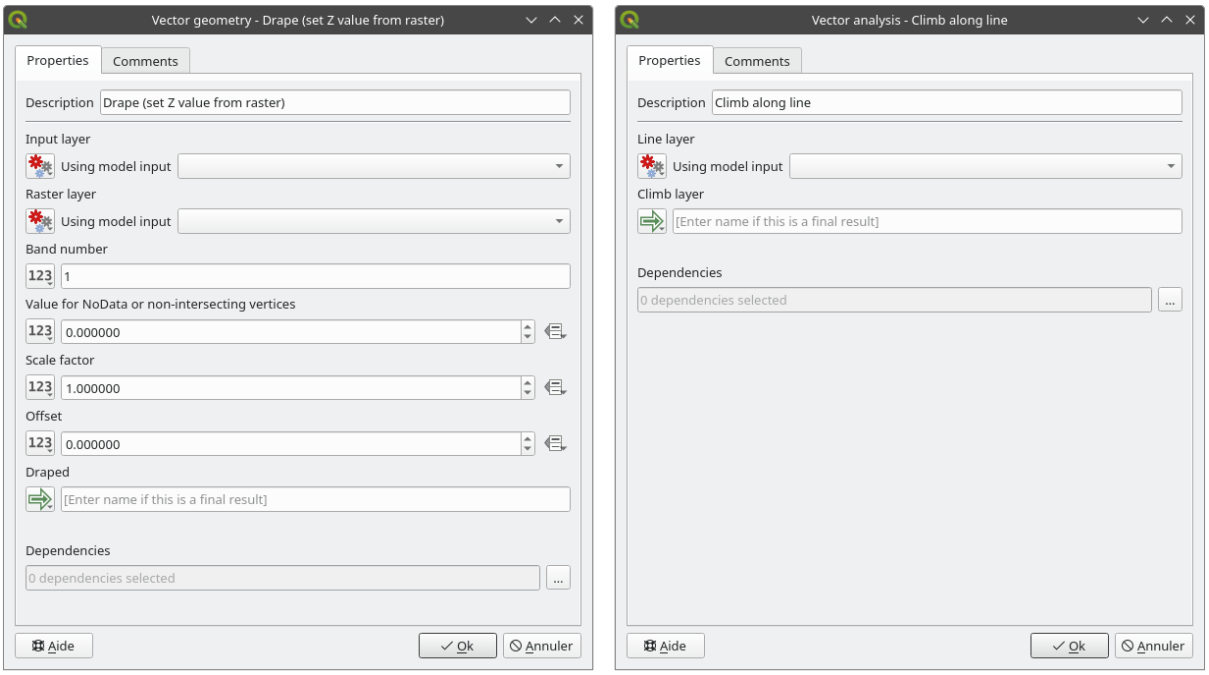


Fig. 23.22: Model Algorithm parameters

As you can see, there are however some differences. Each parameter has a drop-down menu next to it allowing to control how it will be served during the workflow:

- Value: allows you to assign a static value to the parameter. Depending on the parameter type, the widget will let you enter a number (`5.0`), a string (`mytext`), select layer(s) loaded in the QGIS project or from a folder, pick items from a list, …
- Pre-calculated Value: opens the *Expression Builder* dialog and lets you define an expression to fill the parameter. Model inputs together with some other layer statistics are available as **variables** and are listed at the top of the Search dialog of the Expression Builder. The expression is evaluated once before the child algorithm is executed and used during the execution of that algorithm.
- Model Input: allows to use an input added to the model as a parameter. Once clicked, this option will list all the suitable inputs for the parameter.
- Algorithm Output: allows to use the output of another algorithm as an input of the current algorithm. As of model inputs, this option will list all the suitable inputs for the parameter.
- The **output parameter** also has the above options in its drop-down menu:
  - add static outputs for child algorithms, e.g. always saving a child algorithm's output to a predefined geopackage or postgres layer
  - use an expression based output values for child algorithms, e.g. generating an automatic file name based on today's date and saving outputs to that file
  - use a model input, e.g. the *File/Folder* model input to specify an output file or folder
  - use another algorithm output, e.g. the output of the *Create directory* algorithm (from *Modeler tools*)
  - an addditional Model Output option makes the output of the algorithm available in the model. If a layer generated by the algorithm is only to be used as input to another algorithm, don't edit that text box.

In the following picture you can see the two input parameters defined as `Model Input` and the temporary output layer:

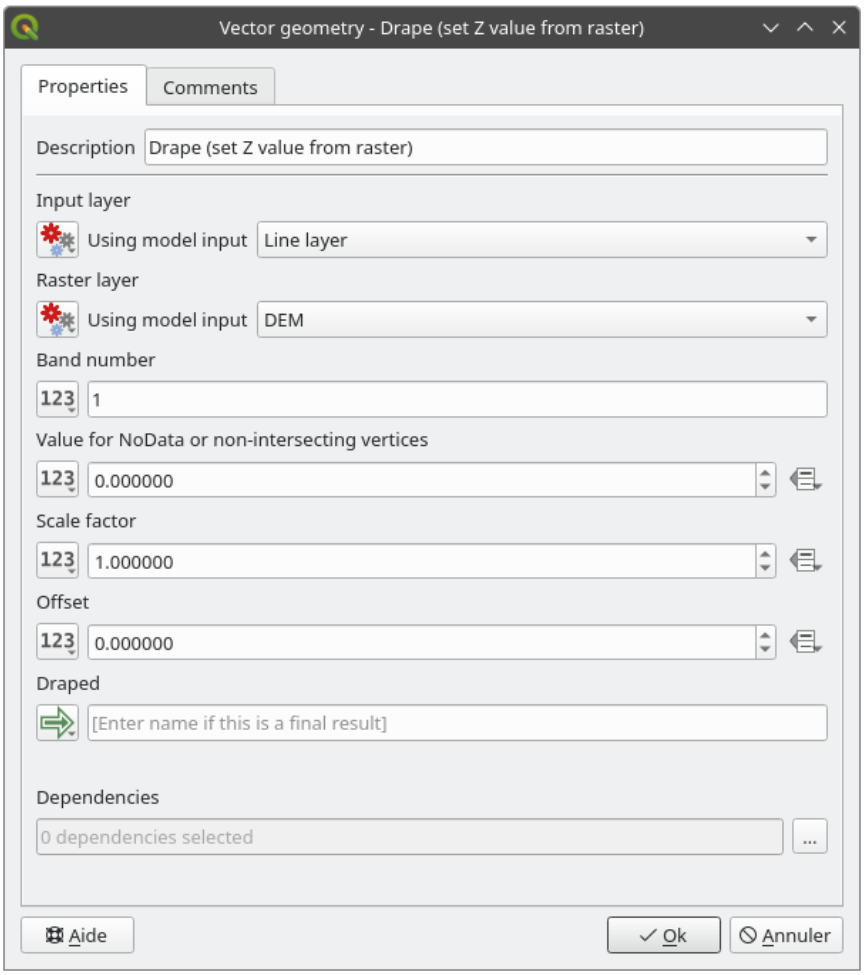


Fig. 23.23: Algorithm Input and Output parameters

You will also find an additional parameter named *Dependencies* that is not available when calling the algorithm from the toolbox. This parameter allows you to define the order in which algorithms are executed, by explicitly defining one algorithm as a *parent* of the current one. This will force the *parent* algorithm to be executed before the current one.

When you use the output of a previous algorithm as the input of your algorithm, that implicitly sets the previous algorithm as parent of the current one (and places the corresponding arrow in the modeler canvas). However, in some cases an algorithm might depend on another one even if it does not use any output object from it (for instance, an algorithm that executes a SQL sentence on a PostgreSQL database and another one that imports a layer into that same database). In that case, just select the previous algorithm in the *Dependencies* parameter and they will be executed in the correct order.

Once all the parameters have been assigned valid values, click on *OK* and the algorithm will be added to the canvas. It will be linked to the elements in the canvas (algorithms or inputs) that provide objects that are used as inputs for the algorithm.

Elements can be dragged to a different position on the canvas using the  Select/Move/Link Item tool. This is useful to make the structure of the model clearer and more intuitive. You can also resize the elements, grasping their border. This is particularly useful if the description of the input or algorithm is long. With *View* ► *Enable snapping* option checked, items resizing or displacement can be bound to a virtual grid, for a more visually structured algorithm design.

Links between elements are updated automatically and you can see a + button at the top and at the bottom of each

algorithm. Clicking the button will list all the inputs and outputs of the algorithm so you can have a quick overview.

You can create or remove connections between model components using an interactive drag-and-drop interface. Each parameter and algorithm item on the model canvas includes small circular sockets, output sockets on the right and input sockets on the left. To connect components, click and drag from an output socket (either from a model input or an algorithm) to an input socket on another algorithm. A visual edge follows the cursor and completes the connection when released. To remove a connection, drag the edge away from the input socket. The algorithm dialogs update automatically to reflect changes made using the drag-and-drop interface.

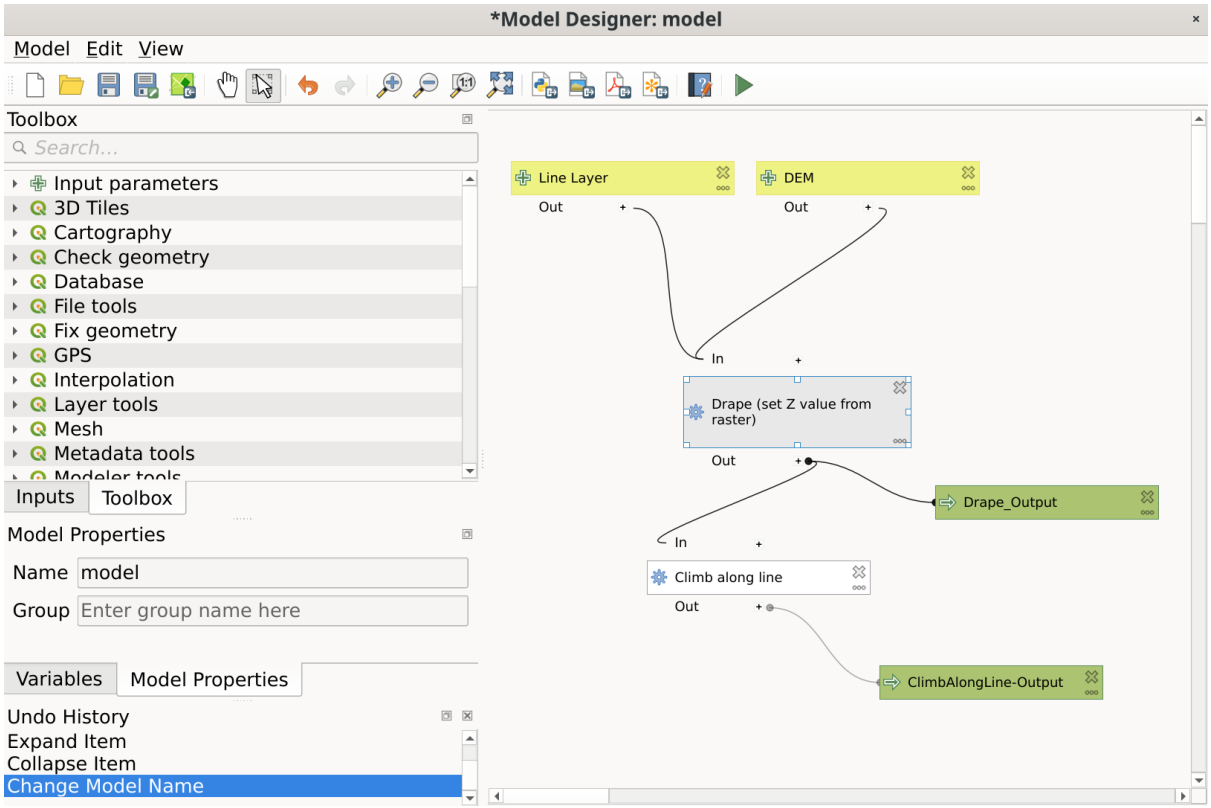


Fig. 23.24: A complete model

With the *Edit* ► *Add Group Box* tool, you can add a draggable *box* to the canvas. This feature is very useful in big models to group related elements in the modeler canvas and to keep the workflow clean. For example we might group together all the algorithms of the example:

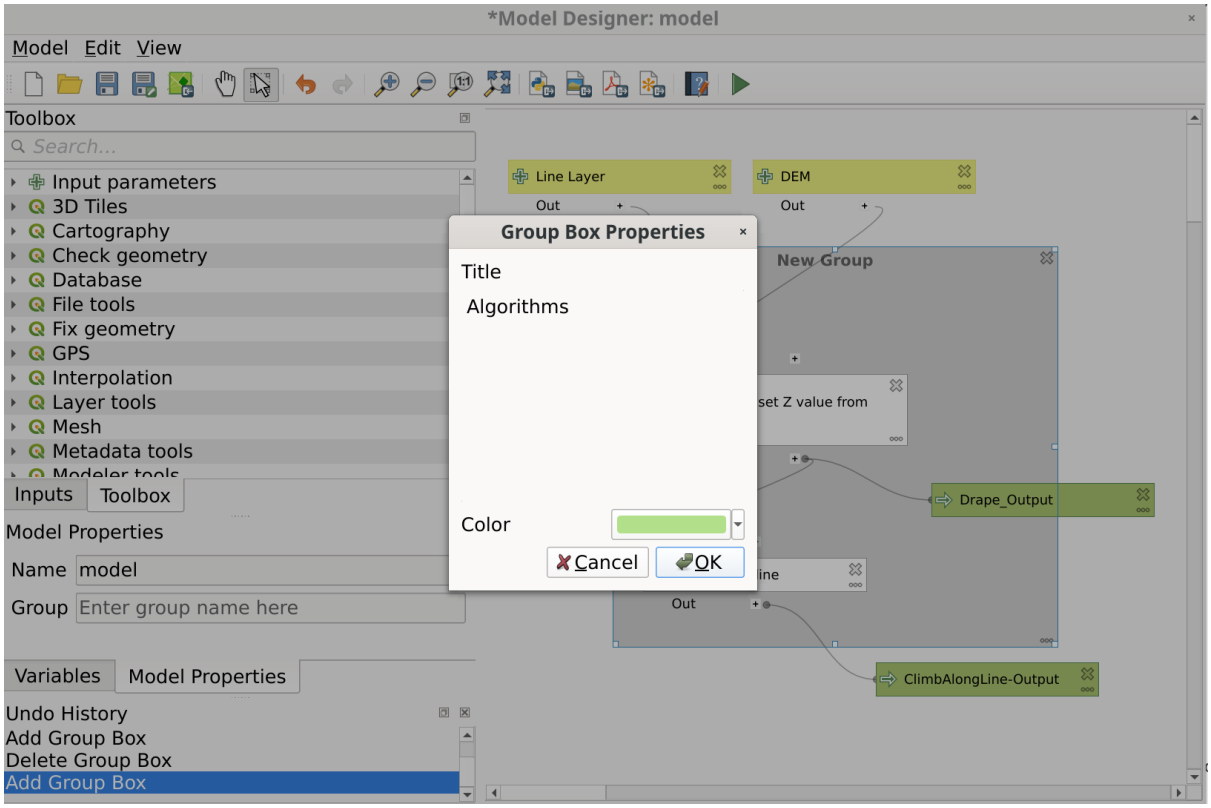


Fig. 23.25: Model Group Box

You can change the name and the color of the boxes. Group boxes are very useful when used together with *View* ► *Zoom To* ► tool, allowing you to zoom to a specific part of the model. You can also zoom in and out by using the mouse wheel.

You might want to change the order of the inputs and how they are listed in the main model dialog. At the bottom of the `Input` panel you will find the `Reorder Model Inputs...` button and by clicking on it a new dialog pops up allowing you to change the order of the inputs:

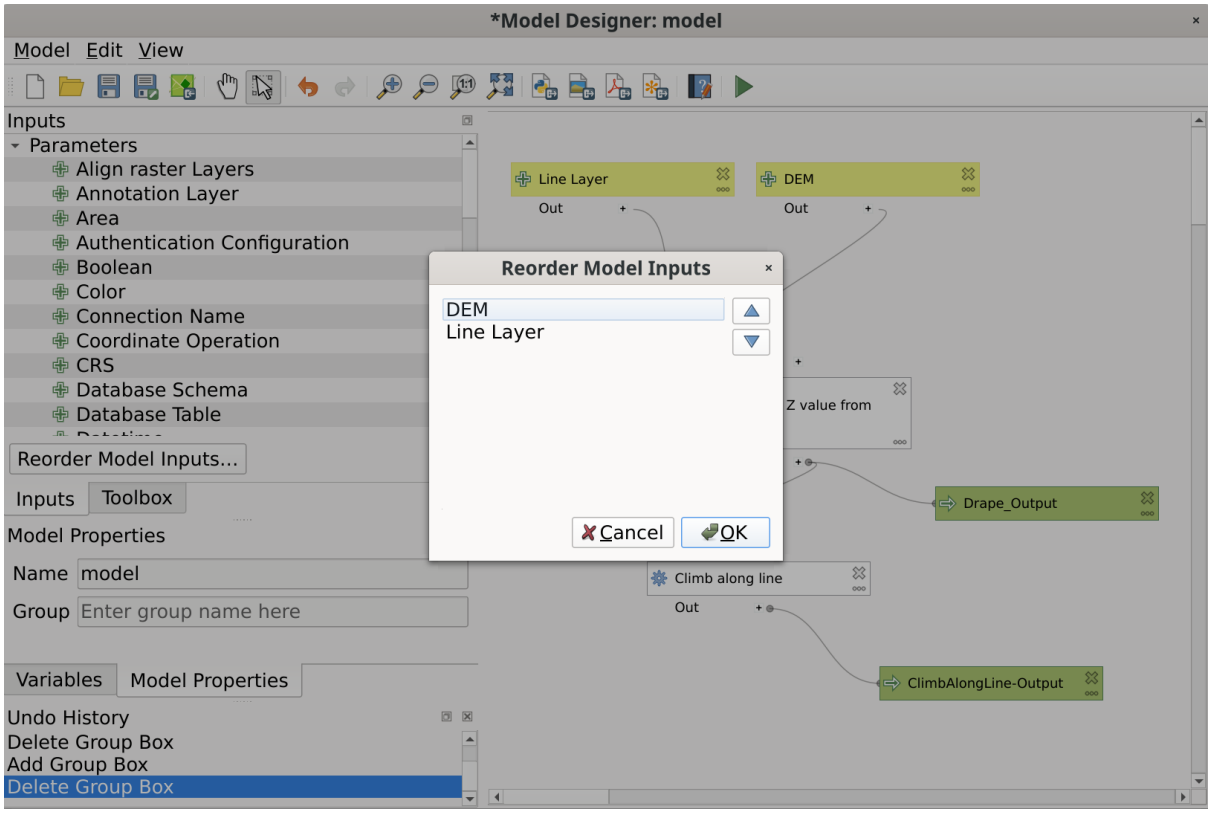


Fig. 23.26: Reorder Model Inputs

There is also the possibility to set a specific order which the outputs from the model must use when loading the results into a project. This gives the model creator a means of ensuring that layers are logically ordered on the canvas when running a model, such as placing a vector layer output over a raster layer output, or a point layer over a polygon layer. The model creator can also set an optional "Group name" for the outputs for automatically grouping outputs within the layer tree using a new group name or by adding them to an existing group. In the `Model` menu you will find the `Reorder Output Layers...` entry and by clicking on it a new dialog pops up allowing you to change the order of the output layers:

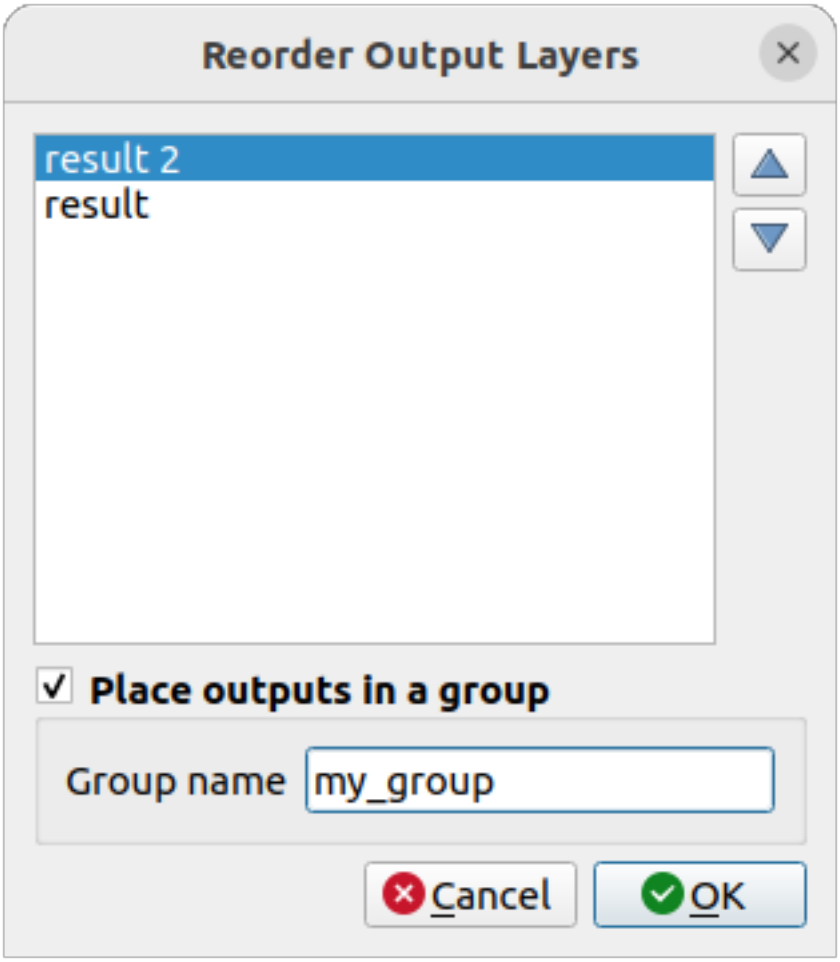


Fig. 23.27: Reorder Output Layers

Comments can also be added to inputs or algorithms present in the modeler. This can be done by going in the *Comment* tab of the item or with a right-click. In the same tab a color can be set manual for individual model comments. Comments are visible only in the modeler canvas and not in the final algorithm dialog; they can be hidden by deactivating *View* ► *Show Comments*.

Your model can be run in various ways:

- You can run the whole model by clicking on Run Model… from the toolbar, right-clicking on the model in the *Browser panel* or *Model* ► *Run Model…*: when using the editor to execute a model, any non-default values will be saved in the inputs. This means that executing the model at a later time from the editor will have the dialog prefilled with those values on any subsequent run.
- You can select elements of the model and run only that subset of the model: press the *Run Selected Steps…* option from the *Models* ► menu or from the contextual menu of a selected algorithm. The initial state will be taken from any previous executions of the model through the editor, so results from previous steps in the model are available for the selected steps. This makes it possible to fix parts of a large model, without having to constantly run the entire model to test. Especially useful when earlier steps in the model are time consuming.
- You can run a subset of the model, starting from a specific algorithm: right-click the algorithm and select *Run from Here…*. Likewise, values are taken from previous executions.
- In order to use the algorithm from the Processing Toolbox, it has to be saved and the modeler dialog closed, to allow the toolbox to refresh its contents.

Documenting your model

You need to document your model, and this can be done from the modeler itself. Click on the  Edit model help button, and a dialog like the one shown next will appear.

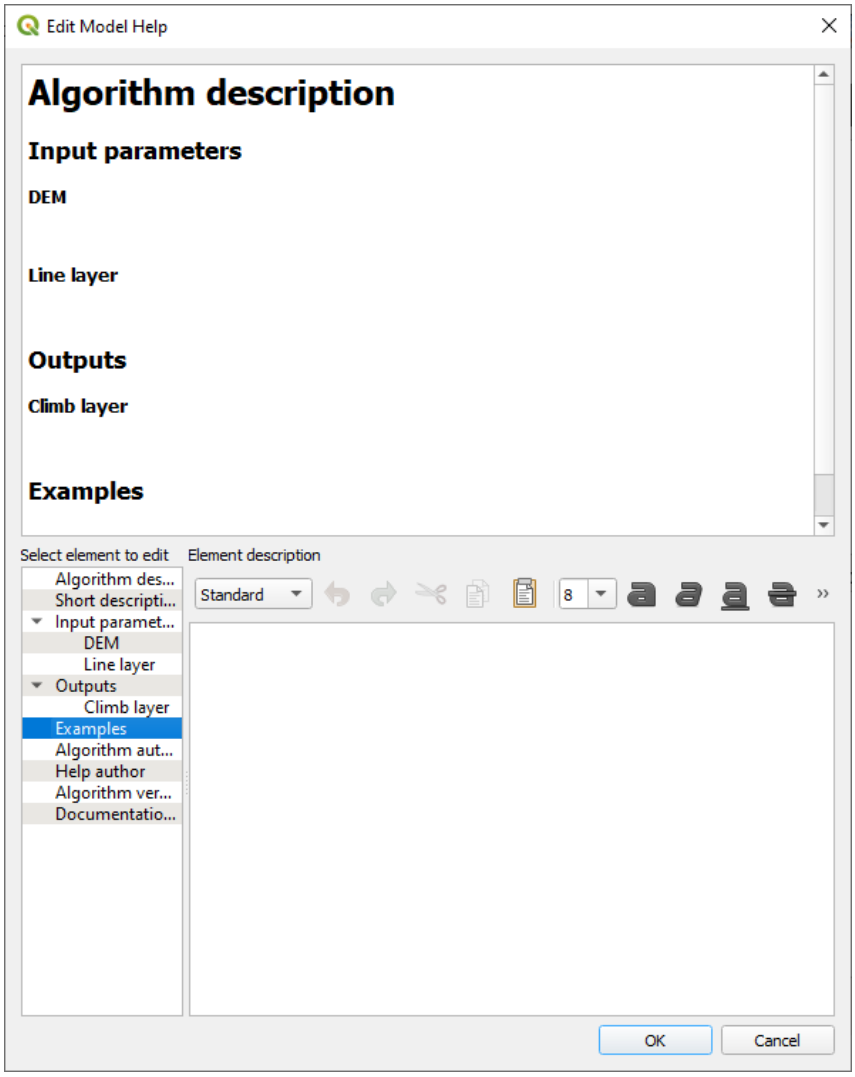


Fig. 23.28: Editing Help

On the right-hand side, you will see a simple HTML page, created using the description of the input parameters and outputs of the algorithm, along with some additional items like a general description of the model or its author. Also, there is an Example section where you can input your own custom examples to help explain the usage of the model. The first time you open the help editor, all these descriptions are empty, but you can edit them using the elements on the left-hand side of the dialog. Select an element on the upper part and then write its description in the text box below.

Model help is saved as part of the model itself.

23.5.3 Saving and loading models

Saving models

Use the Save model button to save the current model and the Open Model button to open a previously saved model. Models are saved with the `.model3` extension. If the model has already been saved from the modeler window, you will not be prompted for a filename. Since there is already a file associated with the model, that file will be used for subsequent saves.

Before saving a model, you have to enter a name and a group for it in the text boxes in the upper part of the window.

Models saved in the `models` folder (the default folder when you are prompted for a filename to save the model) will appear in the toolbox in the corresponding branch. When the toolbox is invoked, it searches the `models` folder for files with the `.model3` extension and loads the models they contain. Since a model is itself an algorithm, it can be added to the toolbox just like any other algorithm.

Models can also be saved within the project file using the Save model in project button. Models saved using this method won't be written as `.model3` files on the disk but will be embedded in the project file.

Project models are available in the *Project models* menu of the toolbox and in the *Project* ► *Models* menu item.

The models folder can be set from the Processing configuration dialog, under the *Modeler* group.

Models loaded from the `models` folder appear not only in the toolbox, but also in the algorithms tree in the *Algorithms* tab of the modeler window. That means that you can incorporate a model as a part of a bigger model, just like other algorithms.

Models will show up in the *Browser* panel and can be run from there.

Exporting a model as a Python script

As we will see in a later chapter, Processing algorithms can be called from the QGIS Python console, and new Processing algorithms can be created using Python. A quick way to create such a Python script is to create a model and then export it as a Python file.

To do so, click on the Export as Script Algorithm… in the modeler canvas or right click on the name of the model in the Processing Toolbox and choose Export Model as Python Algorithm….

Exporting a model as an image, PDF or SVG

A model can also be exported as an image, SVG or PDF (for illustration purposes) by clicking Export as image, Export as PDF or Export as SVG.

23.5.4 Editing a model

You can edit the model you are currently creating, redefining the workflow and the relationships between the algorithms and inputs that define the model.

You can also edit existing model from the *Browser panel* choosing *Edit Model…* option.

If you right-click on an algorithm in the canvas, you will see a context menu like the one shown next:

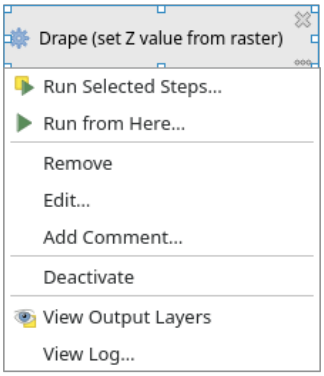


Fig. 23.29: Modeler Right Click

- Options for *running part of the model* have been exposed earlier.
- Selecting the *Remove* option will cause the selected algorithm to be removed. An algorithm can be removed only if there are no other algorithms depending on it. That is, if no output from the algorithm is used as input in a different one.
- Selecting the *Edit…* option will show the parameter dialog so you can change the inputs and parameter values. Not all input elements available in the model will appear as available inputs. Layers or values generated at a more advanced step in the workflow defined by the model will not be available if they cause circular dependencies.

  Select the new values and click on the *OK* button as usual. The connections between the model elements will change in the modeler canvas accordingly.
- The *Add comment…* allows you to add a comment to the algorithm to better describe the behavior.

Note

Right-clicking an input parameter will also allow you to *Remove*, *Edit…* and *Add comment…*

- A model can be run partially by deactivating some of its algorithms. To do it, select the *Deactivate* option in the context menu that appears when right-clicking on an algorithm element. The selected algorithm, and all the ones in the model that depend on it will be displayed in grey and will not be executed as part of the model.

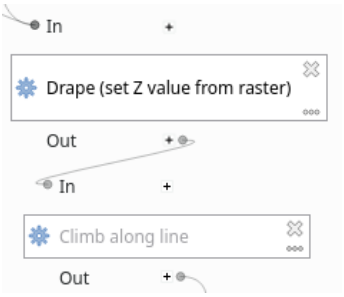


Fig. 23.30: Model with a deactivated algorithm

  When right-clicking on an algorithm that is not active, you will see an *Activate* menu option that you can use to reactivate it.
- When editing a model through the designer (and after having run that model), right-clicking any child step in the model allows to select *View Output Layers*. This will add the output layers from that step as new layers in the current QGIS project.

  This action is available for all child algorithms in the model, even if the model is not configured to use the outputs from those children as model outputs. This is designed as a helpful debugging action. If a model

fails (or gives unexpected results), you can then trace through the model and view the outputs for suspected problematic steps. It avoids the need to add temporary outputs to a model and re-run to test.

Additionally, this action is always available after running the model, even if the model itself failed (e.g., because of a misconfigured step later in the model).

- The *View Log* action helps you see the log for each child algorithm after you've closed down the algorithm dialog.

23.6 The batch processing interface

23.6.1 Introduction

All algorithms (including models) can be executed as a batch process. That is, they can be executed using not just a single set of inputs, but several of them, executing the algorithm as many times as needed. This is useful when processing large amounts of data, since it is not necessary to launch the algorithm many times from the toolbox.

To execute an algorithm as a batch process, right-click on its name in the toolbox and select the *Execute as batch process* option in the pop-up menu that will appear.

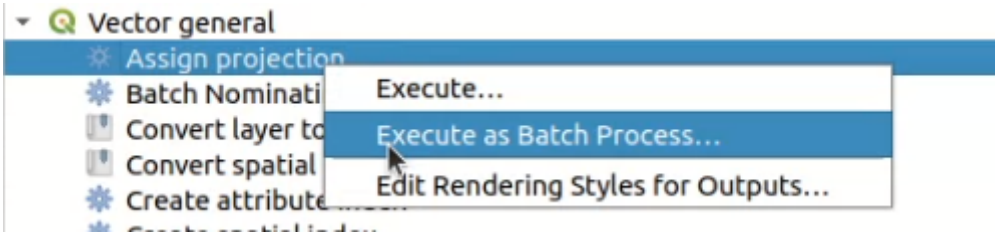


Fig. 23.31: Batch Processing from right-click

If you have the execution dialog of the algorithm open, you can also start the batch processing interface from there, clicking on the *Run as batch process...* button.

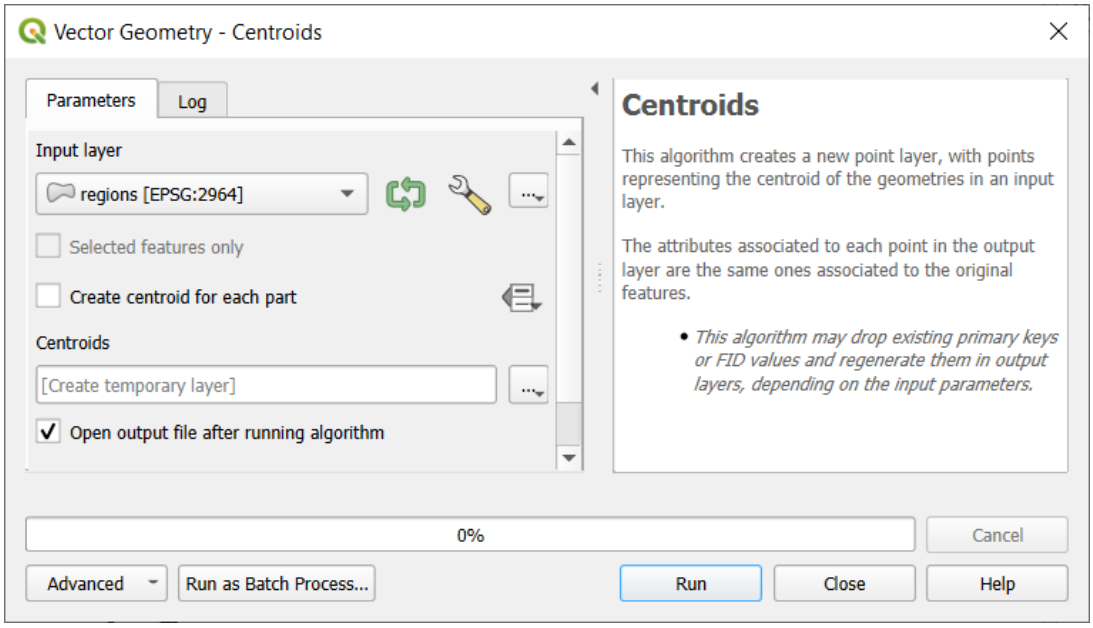


Fig. 23.32: Batch Processing From Algorithm Dialog

23.6.2 The parameters table

Executing a batch process is similar to performing a single execution of an algorithm. Parameter values have to be defined, but in this case we need not just a single value for each parameter, but a set of them instead, one for each time the algorithm has to be executed. Values are introduced using a table like the one shown next, where each row is an iteration and columns are the parameters of the algorithm.

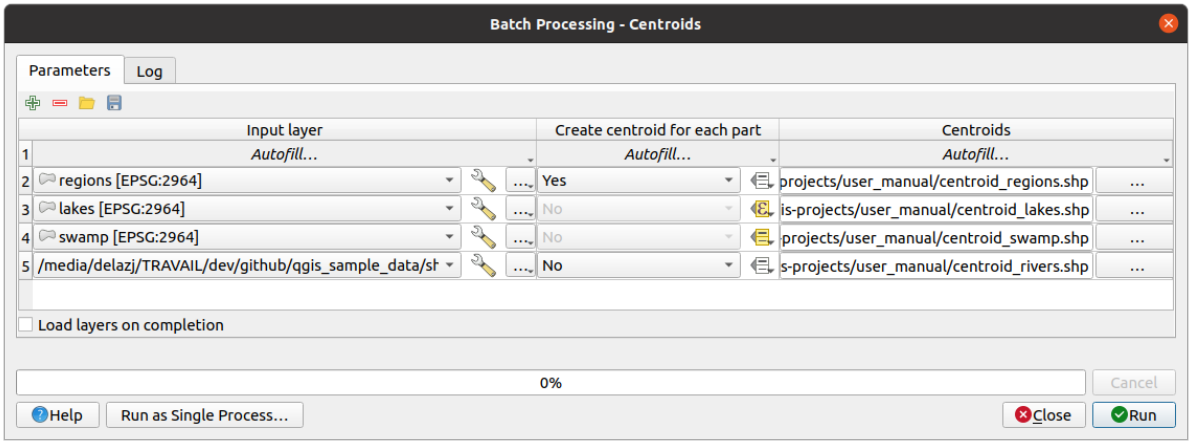


Fig. 23.33: Batch Processing

From the top toolbar, you can:

- Toggle advanced mode: Available only when the algorithm has parameters that are marked as advanced, this button allows to show or hide such parameters in the batch dialog.
- Add row: adds a new processing entry for configuration
- Remove row(s): remove selected rows from the table. Row selection is done by clicking the number at the left and allows *keyboard combination* for multi selection.
- Open a batch processing configuration file
- Save the batch processing configuration to a `.JSON` file that can be run afterwards

By default, the table contains just two rows:

- The first row displays in each cell an *Autofill...* ► drop-down menu with *options* to quickly fill the cells below. Available options depend on the parameter type.
- The second row (as well as each subsequent one) represents a single execution of the algorithm, and each cell contains the value of one of the parameters. It is similar to the parameters dialog that you see when executing an algorithm from the toolbox, but with a different arrangement.

At the bottom of the table, you can set whether to *Load layers on completion*.

Once the size of the table has been set, it has to be filled with the desired values.

23.6.3 Filling the parameters table

For most parameters, setting the value is trivial. The appropriate widget, same as in the *single process dialog*, is provided, allowing to just type the value, or select it from a list of possible values, depending on the parameter type. This also includes data-define widget, when compatible.

To automate the batch process definition and avoid filling the table cell by cell, you may want to press down the *Autofill…* menu of a parameter and select any of the following options to replace values in the column:

- *Fill Down* will take the input for the first process and enter it for all other processes.
-  *Calculate by Expression…* will allow you to create a new QGIS expression to use to update all existing values within that column. Existing parameter values (including those from other columns) are available for use inside the expression via *variables*. E.g. setting the number of segments based on the buffer distance of each layer:

```
CASE WHEN @DISTANCE > 20 THEN 12 ELSE 8 END
```

- *Add Values by Expression…* will add new rows using the values from an expression which returns an array (as opposed to *Calculate by Expression…*, which works only on existing rows). The intended use case is to allow populating the batch dialog using complex numeric series. For example adding rows for a batch buffer using the expression `generate_series(100, 1000, 50)` for distance parameter results in new rows with values 100, 150, 200, …. 1000.
- When setting a file or layer parameter, more options are provided:
  - *Add Files by Pattern…*: adds new rows to the table for files matching a *File pattern* in a folder to *Look in*. E.g. `*.shp` will add to the list all the `SHP` files in the folder. Check  *Search recursively* to also browse sub-folders.
  - *Select Files…* individually on disk
  - *Add All Files from a Directory…*
  - *Select from Open Layers…* in the active project

Output data parameter exposes the same capabilities as when executing the algorithm as a single process. Depending on the algorithm, the output can be:

- skipped, if the cell is left empty
- saved as a temporary layer: fill the cell with your chosen output name, select  *Create Temporary Layer* from the … drop-down, and remember to tick the  *Load layers on completion* checkbox. For temporary layers, the value you provide will be used as the layer name. You can use the *Autofill* options to construct that name.
- saved as a plain file (`.SHP`, `.GPKG`, `.XML`, `.PDF`, `.JPG`,…): choose *Select File/Folder…* from the … drop-down. For plain files, you can set the path using the *Autofill* options exposed beforehand.

  E.g. use *Calculate by Expression…* to set output file names to complex expressions like:

```
'/home/me/stuff/buffer_' || left(@INPUT, 30) || '_' || @DISTANCE || '.shp'
```

  You can also type the file path directly or use the file chooser dialog that appears when clicking on the accompanying *Select File/Folder…* button. Once you select the file, a new dialog is shown to allow for auto-completion of other cells in the same column (same parameter).

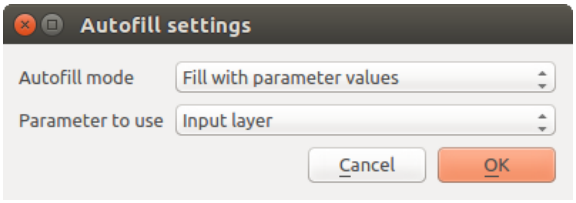


Fig. 23.34: Batch Processing Save

If the default value (*Do not autofill*) is selected, it will just put the selected filename in the selected cell from the parameters table. If any of the other options is selected, all the cells **below** the selected one will be automatically filled based on a defined criteria:

- *Fill with numbers*: incrementally appends a number to the file name
- *Fill with parameter values*: you can select a parameter whose value in the same row is appended to the file name. This is particularly useful for naming output data objects according to input ones.

- saved as a layer within a database container:

```
# Indicate a layer within a GeoPackage file
ogr:dbname='C:/Path/To/Geopackage.gpkg' table="New_Table" (geom)

# Use the "Calculate By Expression" to output to different layers in a␣
↪GeoPackage
'ogr:dbname=\'' || @project_folder || '/Buffers.gpkg\' table="' || @INPUT || '_
↪' || @DISTANCE || '" (geom)'
```

23.6.4 Executing the batch process

To execute the batch process once you have introduced all the necessary values, just click on *Run*. The *Log* panel is activated and displays details and steps of the execution process. Progress of the global batch task will be shown in the progress bar in the lower part of the dialog.

23.7 Using processing algorithms from the console

The console allows advanced users to increase their productivity and perform complex operations that cannot be performed using any of the other GUI elements of the processing framework. Models involving several algorithms can be defined using the command-line interface, and additional operations such as loops and conditional sentences can be added to create more flexible and powerful workflows.

There is not a processing console in QGIS, but all processing commands are available instead from the QGIS built-in *Python console*. That means that you can incorporate those commands into your console work and connect processing algorithms to all the other features (including methods from the QGIS API) available from there.

The code that you can execute from the Python console, even if it does not call any specific processing method, can be converted into a new algorithm that you can later call from the toolbox, the graphical modeler or any other component, just like you do with any other algorithm. In fact, some algorithms that you can find in the toolbox are simple scripts.

In this section, we will see how to use processing algorithms from the QGIS Python console, and also how to write algorithms using Python.

23.7.1 Calling algorithms from the Python console

The first thing you have to do is to import the processing functions with the following line:

```
>>> from qgis import processing
```

Now, there is basically just one (interesting) thing you can do with that from the console: execute an algorithm. That is done using the `run()` method, which takes the name of the algorithm to execute as its first parameter, and then a variable number of additional parameters depending on the requirements of the algorithm. So the first thing you need to know is the name of the algorithm to execute. That is not the name you see in the toolbox, but rather a unique command–line name. To find the right name for your algorithm, you can use the `processingRegistry`. Type the following line in your console:

```
>>> for alg in QgsApplication.processingRegistry().algorithms():
        print(alg.id(), "->", alg.displayName())
```

You will see something like this (with some extra dashes added to improve readability).

```
3d:tessellate --------------> Tessellate
gdal:aspect ----------------> Aspect
gdal:assignprojection ------> Assign projection
gdal:buffervectors ---------> Buffer vectors
gdal:buildvirtualraster ----> Build Virtual Raster
gdal:cliprasterbyextent ----> Clip raster by extent
gdal:cliprasterbymasklayer -> Clip raster by mask layer
gdal:clipvectorbyextent ----> Clip vector by extent
gdal:clipvectorbypolygon ---> Clip vector by mask layer
gdal:colorrelief -----------> Color relief
gdal:contour ---------------> Contour
gdal:convertformat ---------> Convert format
gdal:dissolve --------------> Dissolve
...
```

That's a list of all the available algorithm IDs, sorted by provider name and algorithm name, along with their corresponding names.

Once you know the command-line name of the algorithm, the next thing to do is to determine the right syntax to execute it. That means knowing which parameters are needed when calling the `run()` method.

There is a method to describe an algorithm in detail, which can be used to get a list of the parameters that an algorithm requires and the outputs that it will generate. To get this information, you can use the `algorithmHelp(id_of_the_algorithm)` method. Use the ID of the algorithm, not the full descriptive name.

Calling the method with `native:buffer` as parameter (`qgis:buffer` is an alias for `native:buffer` and will also work), you get the following description:

```
>>> processing.algorithmHelp("native:buffer")
Buffer (native:buffer)

This algorithm computes a buffer area for all the features in an
input layer, using a fixed or dynamic distance.

The segments parameter controls the number of line segments to
use to approximate a quarter circle when creating rounded
offsets.

The end cap style parameter controls how line endings are handled
in the buffer.

The join style parameter specifies whether round, miter or
beveled joins should be used when offsetting corners in a line.
```

(continues on next page)

(continued from previous page)

```
The miter limit parameter is only applicable for miter join
styles, and controls the maximum distance from the offset curve
to use when creating a mitered join.


----------------
Input parameters
----------------

INPUT: Input layer

   Parameter type: QgsProcessingParameterFeatureSource

   Accepted data types:
           - str: layer ID
           - str: layer name
           - str: layer source
           - QgsProcessingFeatureSourceDefinition
           - QgsProperty
           - QgsVectorLayer

DISTANCE: Distance

   Parameter type: QgsProcessingParameterDistance

   Accepted data types:
           - int
           - float
           - QgsProperty

SEGMENTS: Segments

   Parameter type: QgsProcessingParameterNumber

   Accepted data types:
           - int
           - float
           - QgsProperty

END_CAP_STYLE: End cap style

   Parameter type: QgsProcessingParameterEnum

   Available values:
           - 0: Round
           - 1: Flat
           - 2: Square

   Accepted data types:
           - int
           - str: as string representation of int, e.g. '1'
           - QgsProperty

JOIN_STYLE: Join style

   Parameter type: QgsProcessingParameterEnum

   Available values:
           - 0: Round
           - 1: Miter
           - 2: Bevel
```

(continues on next page)

(continued from previous page)

```
   Accepted data types:
            - int
            - str: as string representation of int, e.g. '1'
            - QgsProperty

MITER_LIMIT: Miter limit

   Parameter type: QgsProcessingParameterNumber

   Accepted data types:
            - int
            - float
            - QgsProperty

DISSOLVE: Dissolve result

   Parameter type: QgsProcessingParameterBoolean

   Accepted data types:
            - bool
            - int
            - str
            - QgsProperty

OUTPUT: Buffered

   Parameter type: QgsProcessingParameterFeatureSink

   Accepted data types:
            - str: destination vector file, e.g. 'd:/test.shp'
            - str: 'memory:' to store result in temporary memory layer
            - str: using vector provider ID prefix and destination URI,
                   e.g. 'postgres:...' to store result in PostgreSQL table
            - QgsProcessingOutputLayerDefinition
            - QgsProperty

----------------
Outputs
----------------

OUTPUT:  <QgsProcessingOutputVectorLayer>
   Buffered
```

Now you have everything you need to run any algorithm. As we have already mentioned, algorithms can be run using: `run()`. Its syntax is as follows:

```
>>> processing.run(name_of_the_algorithm, parameters)
```

Where parameters is a dictionary of parameters that depend on the algorithm you want to run, and is exactly the list that the `algorithmHelp()` method gives you.

```
>>> processing.run("native:buffer", {'INPUT': '/data/lines.shp',
                'DISTANCE': 100.0,
                'SEGMENTS': 10,
                'DISSOLVE': True,
                'END_CAP_STYLE': 0,
                'JOIN_STYLE': 0,
                'MITER_LIMIT': 10,
                'OUTPUT': '/data/buffers.shp'})
```

If a parameter is optional and you do not want to use it, then don't include it in the dictionary.

If a parameter is not specified, the default value will be used.

Depending on the type of parameter, values are introduced differently. The next list gives a quick review of how to introduce values for each type of input parameter:

- Raster Layer, Vector Layer or Table. Simply use a string with the name that identifies the data object to use (the name it has in the QGIS Table of Contents) or a filename (if the corresponding layer is not opened, it will be opened but not added to the map canvas). If you have an instance of a QGIS object representing the layer, you can also pass it as parameter.
- Enumeration. If an algorithm has an enumeration parameter, the value of that parameter should be entered using an integer value. To know the available options, you can use the `algorithmHelp()` command, as above. For instance, the `native:buffer` algorithm has an enumeration called JOIN_STYLE:

```
JOIN_STYLE: Join style

   Parameter type: QgsProcessingParameterEnum

   Available values:
           - 0: Round
           - 1: Miter
           - 2: Bevel

   Accepted data types:
           - int
           - str: as string representation of int, e.g. '1'
           - QgsProperty
```

  In this case, the parameter has three options. Notice that ordering is zero-based.

- Boolean. Use `True` or `False`.
- Multiple input. The value is a string with input descriptors separated by semicolons (`;`). As in the case of single layers or tables, each input descriptor can be the data object name, or its file path.
- Table Field from XXX. Use a string with the name of the field to use. This parameter is case-sensitive.
- Fixed Table. Type the list of all table values separated by commas (`,`) and enclosed between quotes (`"`). Values start on the upper row and go from left to right. You can also use a 2-D array of values representing the table.
- CRS. Enter the EPSG code number of the desired CRS.
- Extent. You must use a string with `xmin`, `xmax`, `ymin` and `ymax` values separated by commas (`,`).

Boolean, file, string and numerical parameters do not need any additional explanations.

Input parameters such as strings, booleans, or numerical values have default values. The default value is used if the corresponding parameter entry is missing.

For output data objects, type the file path to be used to save it, just as it is done from the toolbox. If the output object is not specified, the result is saved to a temporary file (or skipped if it is an optional output). The extension of the file determines the file format. If you enter a file extension not supported by the algorithm, the default file format for that output type will be used, and its corresponding extension appended to the given file path.

Unlike when an algorithm is executed from the toolbox, outputs are not added to the map canvas if you execute that same algorithm from the Python console using the `run()` method. That method returns a dictionary with one or more output names (the ones shown in the algorithm description) as keys and the file paths of those outputs as values:

```
>>> myresult = processing.run("native:buffer", {'INPUT': '/data/lines.shp',
                'DISTANCE': 100.0,
                'SEGMENTS': 10,
                'DISSOLVE': True,
                'END_CAP_STYLE': 0,
                'JOIN_STYLE': 0,
                'MITER_LIMIT': 10,
                'OUTPUT': '/data/buffers.shp'})
```

(continues on next page)

(continued from previous page)

```
>>> myresult['OUTPUT']
/data/buffers.shp
```

You can then load the output in the project as any common layer:

```
>>> buffered_layer = myresult['OUTPUT']
>>> QgsProject.instance().addMapLayer(buffered_layer)
```

To immediately load the processing outputs in the project, you can use the `runAndLoadResults()` method instead of `run()`.

```
>>> processing.runAndLoadResults("native:buffer", {parameters:values})
```

If you want to open an algorithm dialog from the console you can use the `createAlgorithmDialog` method. The only mandatory parameter is the algorithm name, but you can also define the dictionary of parameters so that the dialog will be filled automatically:

```
>>> my_dialog = processing.createAlgorithmDialog("native:buffer", {
                'INPUT': '/data/lines.shp',
                'DISTANCE': 100.0,
                'SEGMENTS': 10,
                'DISSOLVE': True,
                'END_CAP_STYLE': 0,
                'JOIN_STYLE': 0,
                'MITER_LIMIT': 10,
                'OUTPUT': '/data/buffers.shp'})
>>> my_dialog.show()
```

The `execAlgorithmDialog` method opens the dialog immediately:

```
>>> processing.execAlgorithmDialog("native:buffer", {
                'INPUT': '/data/lines.shp',
                'DISTANCE': 100.0,
                'SEGMENTS': 10,
                'DISSOLVE': True,
                'END_CAP_STYLE': 0,
                'JOIN_STYLE': 0,
                'MITER_LIMIT': 10,
                'OUTPUT': '/data/buffers.shp'})
```

23.7.2 Creating scripts and running them from the toolbox

You can create your own algorithms by writing Python code. Processing scripts extend `QgsProcessingAlgorithm`, so you need to add some extra lines of code to implement mandatory functions. You can find *Create new script* (clean sheet) and *Create New Script from Template* (template that includes code for mandatory functions of `QgsProcessingAlgorithm`) under the *Scripts* dropdown menu on the top of the Processing toolbox. The Processing Script Editor will open, and that's where you should type your code. Saving the script from there in the `scripts` folder (the default folder when you open the save file dialog) with a `.py` extension should create the corresponding algorithm.

The name of the algorithm (the one you will see in the toolbox) is defined within the code.

Let's have a look at the following code, which defines a Processing algorithm that performs a buffer operation with a user defined buffer distance on a vector layer that is specified by the user, after first smoothing the layer.

```
from qgis.core import (QgsProcessingAlgorithm,
       QgsProcessingParameterNumber,
       QgsProcessingParameterFeatureSource,
       QgsProcessingParameterFeatureSink)
```

(continues on next page)

(continued from previous page)

```
from qgis import processing

class algTest(QgsProcessingAlgorithm):
    INPUT_BUFFERDIST = 'BUFFERDIST'
    OUTPUT_BUFFER = 'OUTPUT_BUFFER'
    INPUT_VECTOR = 'INPUT_VECTOR'

    def __init__(self):
        super().__init__()

    def name(self):
        return "algTest"

    def displayName(self):
        return "algTest script"

    def createInstance(self):
        return type(self)()

    def initAlgorithm(self, config=None):
        self.addParameter(QgsProcessingParameterFeatureSource(
            self.INPUT_VECTOR, "Input vector"))
        self.addParameter(QgsProcessingParameterNumber(
            self.INPUT_BUFFERDIST, "Buffer distance",
            QgsProcessingParameterNumber.Double,
            100.0))
        self.addParameter(QgsProcessingParameterFeatureSink(
            self.OUTPUT_BUFFER, "Output buffer"))

    def processAlgorithm(self, parameters, context, feedback):
        #DO SOMETHING
        algresult = processing.run("native:smoothgeometry",
            {'INPUT': parameters[self.INPUT_VECTOR],
             'ITERATIONS':2,
             'OFFSET':0.25,
             'MAX_ANGLE':180,
             'OUTPUT': 'memory:'},
            context=context, feedback=feedback, is_child_algorithm=True)
        smoothed = algresult['OUTPUT']
        algresult = processing.run('native:buffer',
            {'INPUT': smoothed,
            'DISTANCE': parameters[self.INPUT_BUFFERDIST],
            'SEGMENTS': 5,
            'END_CAP_STYLE': 0,
            'JOIN_STYLE': 0,
            'MITER_LIMIT': 10,
            'DISSOLVE': True,
            'OUTPUT': parameters[self.OUTPUT_BUFFER]},
            context=context, feedback=feedback, is_child_algorithm=True)
        buffered = algresult['OUTPUT']
        return {self.OUTPUT_BUFFER: buffered}
```

After doing the necessary imports, the following `QgsProcessingAlgorithm` functions are specified:

- `name()`: The id of the algorithm (lowercase).
- `displayName()`: A human readable name for the algorithm.
- `createInstance()`: Create a new instance of the algorithm class.
- `initAlgorithm()`: Configure the parameterDefinitions and outputDefinitions.

  Here you describe the parameters and output of the algorithm. In this case, a feature source for the input, a

feature sink for the result and a number for the buffer distance.

- `processAlgorithm()`: Do the work.

  Here we first run the `smoothgeometry` algorithm to smooth the geometry, and then we run the `buffer` algorithm on the smoothed output. To be able to run algorithms from within another algorithm we have to set the `is_child_algorithm` argument to `True`. You can see how input and output parameters are used as parameters to the `smoothgeometry` and `buffer` algorithms.

There are a number of different parameter types available for input and output. You can find the full list at *Input and output types for Processing Algorithms*.

The first parameter to the constructors is the name of the parameter, and the second is the description of the parameter (for the user interface). The rest of the constructor parameters are parameter type specific.

The input can be turned into QGIS classes using the `parameterAs` functions of `QgsProcessingAlgorithm`. For instance to get the number provided for the buffer distance as a double:

```
self.parameterAsDouble(parameters, self.INPUT_BUFFERDIST, context)).
```

The `processAlgorithm` function should return a dictionary containing values for every output defined by the algorithm. This allows access to these outputs from other algorithms, including other algorithms contained within the same model.

Well behaved algorithms should define and return as many outputs as makes sense. Non-feature outputs, such as numbers and strings, are very useful when running your algorithm as part of a larger model, as these values can be used as input parameters for subsequent algorithms within the model. Consider adding numeric outputs for things like the number of features processed, the number of invalid features encountered, the number of features output, etc. The more outputs you return, the more useful your algorithm becomes!

Feedback

The `feedback` object passed to `processAlgorithm()` should be used for user feedback / interaction. You can use the `setProgress()` function of the `feedback` object to update the progress bar (0 to 100) to inform the user about the progress of the algorithm. This is very useful if your algorithm takes a long time to complete.

The `feedback` object provides an `isCanceled()` method that should be monitored to enable cancelation of the algorithm by the user. The `pushInfo()` method of `feedback` can be used to send information to the user, and `reportError()` is handy for pushing non-fatal errors to users.

Algorithms should avoid using other forms of providing feedback to users, such as print statements or logging to `QgsMessageLog`, and should always use the feedback object instead. This allows verbose logging for the algorithm, and is also thread-safe (which is important, given that algorithms are typically run in a background thread).

Handling errors

If your algorithm encounters an error which prevents it from executing, such as invalid input values or some other condition from which it cannot or should not recover, then you should raise a `QgsProcessingException`. E.g.:

```
if feature['value'] < 20:
  raise QgsProcessingException('Invalid input value {}, must be >= 20'.
↪format(feature['value']))
```

Try to avoid raising `QgsProcessingException` for non-fatal errors (e.g. when a feature has a null geometry), and instead just report these errors via `feedback.reportError()` and skip the feature. This helps make your algorithm "model-friendly", as it avoids halting the execution of an entire algorithm when a non-fatal error is encountered.

Documenting your scripts

As in the case of models, you can create additional documentation for your scripts, to explain what they do and how to use them.

`QgsProcessingAlgorithm` provides the `helpString()`, `shortHelpString()` and `helpUrl()` functions for that purpose. Specify / override these to provide more help to the user.

`shortDescription()` is used in the tooltip when hovering over the algorithm in the toolbox.

23.7.3 Pre- and post-execution script hooks

Scripts can also be used as pre- and post-execution hooks that are run before and after an algorithm is run, respectively. This can be used to automate tasks that should be performed whenever an algorithm is executed.

The syntax is identical to the syntax explained above, but an additional global variable named `alg` is available, representing the algorithm that has just been (or is about to be) executed.

In the *General* group of the processing options dialog, you will find two entries named *Pre-execution script* and *Post-execution script* where the filenames of the scripts to be run in each case can be entered.

23.8 Using processing from the command line

QGIS comes with a tool called `QGIS Processing Executor` which allows you to run Processing algorithms and models (built-in or provided by plugins) directly from the command line without starting QGIS Desktop itself.

From a command line tool, run `qgis_process` and you should get:

```
QGIS Processing Executor - 3.35.0-Master 'Master' (3.35.0-Master)
Usage: C:\OSGeo4W\apps\qgis-dev\bin\qgis_process.exe [--help] [--version] [--json]␣
↪[--verbose] [--no-python] [--skip-loading-plugins] [command] [algorithm id, path␣
↪to model file, or path to Python script] [parameters]

Options:

  --help or -h            Output the help
  --version or -v         Output all versions related to QGIS Process
  --json                  Output results as JSON objects
  --verbose               Output verbose logs
  --no-python             Disable Python support (results in faster startup)
  --skip-loading-plugins  Avoid loading enabled plugins (results in faster startup)

Available commands:

  plugins            list available and active plugins
  plugins enable     enables an installed plugin. The plugin name must be␣
↪specified, e.g. "plugins enable cartography_tools"
  plugins disable    disables an installed plugin. The plugin name must be␣
↪specified, e.g. "plugins disable cartography_tools"
  list               list all available processing algorithms
  help               show help for an algorithm. The algorithm id or a path to a␣
↪model file must be specified.
  run                runs an algorithm. The algorithm id or a path to a model file␣
↪and parameter values must be specified.
                     Parameter values are specified after -- with PARAMETER=VALUE␣
↪syntax.
                     Ordered list values for a parameter can be created by␣
↪specifying the parameter multiple times,
```

(continues on next page)

(continued from previous page)

```
                    e.g. --LAYERS=layer1.shp --LAYERS=layer2.shp
                    Alternatively, a '-' character in place of the parameters␣
↪argument indicates that the parameters should be read from STDIN as a JSON␣
↪object.
                    The JSON should be structured as a map containing at least␣
↪the "inputs" key specifying a map of input parameter values.
                    This implies the --json option for output as a JSON object.
                    If required, the ellipsoid to use for distance and area␣
↪calculations can be specified via the "--ELLIPSOID=name" argument.
                    If required, an existing QGIS project to use during the␣
↪algorithm execution can be specified via the "--PROJECT_PATH=path" argument.
                    When passing parameters as a JSON object from STDIN, these␣
↪extra arguments can be provided as an "ellipsoid" and a "project_path" key␣
↪respectively.
```

Note

Only installed plugins that advertise `hasProcessingProvider=yes` in their `metadata.txt` file are recognized and can be activated or loaded by qgis_process tool.

Hint

Before calling qgis_process on a system without window manager (e.g. a headless server), you should set:

```
export QT_QPA_PLATFORM=offscreen
```

The command `list` can be used to get a list of all available providers and algorithms.

```
qgis_process list
```

The command `help` can be used to get further information about commands or algorithms.

```
qgis_process help qgis:regularpoints
```

The command `run` can be used to run an algorithm or model. Specify the name of the algorithm or a path to a model as first parameter.

```
qgis_process run native:buffer -- INPUT=source.shp DISTANCE=2 OUTPUT=buffered.shp
```

Where a parameter accepts a list of values, set the same variable multiple times.

```
qgis_process run native:mergevectorlayers -- LAYERS=input1.shp LAYERS=input2.shp␣
↪OUTPUT=merged.shp
```

While running an algorithm a text-based feedback bar is shown, and the operation can be cancelled via `CTRL+C`.

The `run` command also supports further parameters.

- `--json` will format stdout output in a JSON structured way.
- `--ellipsoid` will set the ellipsoid to the specified one.
- `--distance_units` will use the specified distance units.
- `--area_units` will use the specified area units.
- `--project_path` will load the specified project for running the algorithm.

Complex input parameters, i.e. parameter types which are themselves specified as a dictionary type object for algorithms, are supported by qgis_process. To indicate that parameters will be specified via stdin, the qgis_process command must follow the format (with a trailing – in place of the usual arguments list).

```
qgis_process run algorithmId –
```

The JSON object must contain an "inputs" key, which is a map of the input parameter values. E.g.

```
echo "{'inputs': {'INPUT': 'my_shape.shp', 'DISTANCE': 5}}" | qgis_process run␣
↪native:buffer –
```

Additionally, extra settings like the distance units, area units, ellipsoid and project path can be included in this JSON object:

```
{
 'ellipsoid': 'EPSG:7019',
 'distance_units': 'feet',
 'area_units': 'ha',
 'project_path': 'C:/temp/my_project.qgs'
 'inputs': {'DISTANCE': 5, 'SEGMENTS': 8 ... }
}
```

Specifying input parameters via stdin implies automatically the `JSON` output format for results.

23.9 Writing new Processing algorithms as Python scripts

There are two options for writing Processing algorithms using Python.

- *Extending* `QgsProcessingAlgorithm`
- *Using the @alg decorator*

Within QGIS, you can use *Create new script* in the *Scripts* menu at the top of the *Processing Toolbox* to open the *Processing Script Editor* where you can write your code. To simplify the task, you can start with a script template by using *Create new script from template* from the same menu. This opens a template that extends `QgsProcessingAlgorithm`.

If you save the script in the `scripts` folder (the default location) with a `.py` extension, the algorithm will become available in the *Processing Toolbox*.

23.9.1 Extending QgsProcessingAlgorithm

The following code

1. takes a vector layer as input
2. counts the number of features
3. does a buffer operation
4. creates a raster layer from the result of the buffer operation
5. returns the buffer layer, raster layer and number of features

```
1 from qgis.PyQt.QtCore import QCoreApplication
2 from qgis.core import (QgsProcessing,
3                        QgsProcessingAlgorithm,
4                        QgsProcessingException,
5                        QgsProcessingOutputNumber,
6                        QgsProcessingParameterDistance,
```

(continues on next page)

(continued from previous page)

```
                       QgsProcessingParameterFeatureSource,
                       QgsProcessingParameterVectorDestination,
                       QgsProcessingParameterRasterDestination)
from qgis import processing


class ExampleProcessingAlgorithm(QgsProcessingAlgorithm):
    """
    This is an example algorithm that takes a vector layer,
    creates some new layers and returns some results.
    """

    def tr(self, string):
        """
        Returns a translatable string with the self.tr() function.
        """
        return QCoreApplication.translate('Processing', string)

    def createInstance(self):
        # Must return a new copy of your algorithm.
        return ExampleProcessingAlgorithm()

    def name(self):
        """
        Returns the unique algorithm name.
        """
        return 'bufferrasterextend'

    def displayName(self):
        """
        Returns the translated algorithm name.
        """
        return self.tr('Buffer and export to raster (extend)')

    def group(self):
        """
        Returns the name of the group this algorithm belongs to.
        """
        return self.tr('Example scripts')

    def groupId(self):
        """
        Returns the unique ID of the group this algorithm belongs
        to.
        """
        return 'examplescripts'

    def shortHelpString(self):
        """
        Returns a localised short help string for the algorithm.
        """
        return self.tr('Example algorithm short description')

    def initAlgorithm(self, config=None):
        """
        Here we define the inputs and outputs of the algorithm.
        """
        # 'INPUT' is the recommended name for the main input
        # parameter.
        self.addParameter(
            QgsProcessingParameterFeatureSource(
```

(continues on next page)

(continued from previous page)

```
                'INPUT',
                self.tr('Input vector layer'),
                types=[QgsProcessing.TypeVectorAnyGeometry]
            )
        )
        self.addParameter(
            QgsProcessingParameterVectorDestination(
                'BUFFER_OUTPUT',
                self.tr('Buffer output'),
            )
        )
        # 'OUTPUT' is the recommended name for the main output
        # parameter.
        self.addParameter(
            QgsProcessingParameterRasterDestination(
                'OUTPUT',
                self.tr('Raster output')
            )
        )
        self.addParameter(
            QgsProcessingParameterDistance(
                'BUFFERDIST',
                self.tr('BUFFERDIST'),
                defaultValue = 1.0,
                # Make distance units match the INPUT layer units:
                parentParameterName='INPUT'
            )
        )
        self.addParameter(
            QgsProcessingParameterDistance(
                'CELLSIZE',
                self.tr('CELLSIZE'),
                defaultValue = 10.0,
                parentParameterName='INPUT'
            )
        )
        self.addOutput(
            QgsProcessingOutputNumber(
                'NUMBEROFFEATURES',
                self.tr('Number of features processed')
            )
        )

    def processAlgorithm(self, parameters, context, feedback):
        """
        Here is where the processing itself takes place.
        """
        # First, we get the count of features from the INPUT layer.
        # This layer is defined as a QgsProcessingParameterFeatureSource
        # parameter, so it is retrieved by calling
        # self.parameterAsSource.
        input_featuresource = self.parameterAsSource(parameters,
                                                     'INPUT',
                                                     context)
        numfeatures = input_featuresource.featureCount()

        # Retrieve the buffer distance and raster cell size numeric
        # values. Since these are numeric values, they are retrieved
        # using self.parameterAsDouble.
        bufferdist = self.parameterAsDouble(parameters, 'BUFFERDIST',
                                            context)
```

(continues on next page)

(continued from previous page)

```
        rastercellsize = self.parameterAsDouble(parameters, 'CELLSIZE',
                                                context)
        if feedback.isCanceled():
            return {}
        buffer_result = processing.run(
            'native:buffer',
            {
                # Here we pass on the original parameter values of INPUT
                # and BUFFER_OUTPUT to the buffer algorithm.
                'INPUT': parameters['INPUT'],
                'OUTPUT': parameters['BUFFER_OUTPUT'],
                'DISTANCE': bufferdist,
                'SEGMENTS': 10,
                'DISSOLVE': True,
                'END_CAP_STYLE': 0,
                'JOIN_STYLE': 0,
                'MITER_LIMIT': 10
            },
            # Because the buffer algorithm is being run as a step in
            # another larger algorithm, the is_child_algorithm option
            # should be set to True
            is_child_algorithm=True,
            #
            # It's important to pass on the context and feedback objects to
            # child algorithms, so that they can properly give feedback to
            # users and handle cancelation requests.
            context=context,
            feedback=feedback)

        # Check for cancelation
        if feedback.isCanceled():
            return {}

        # Run the separate rasterization algorithm using the buffer result
        # as an input.
        rasterized_result = processing.run(
            'qgis:rasterize',
            {
                # Here we pass the 'OUTPUT' value from the buffer's result
                # dictionary off to the rasterize child algorithm.
                'LAYER': buffer_result['OUTPUT'],
                'EXTENT': buffer_result['OUTPUT'],
                'MAP_UNITS_PER_PIXEL': rastercellsize,
                # Use the original parameter value.
                'OUTPUT': parameters['OUTPUT']
            },
            is_child_algorithm=True,
            context=context,
            feedback=feedback)

        if feedback.isCanceled():
            return {}

        # Return the results
        return {'OUTPUT': rasterized_result['OUTPUT'],
                'BUFFER_OUTPUT': buffer_result['OUTPUT'],
                'NUMBEROFFEATURES': numfeatures}
```

Processing algorithm standard functions:

- **createInstance (mandatory)**
  Must return a new copy of your algorithm. If you change the name of the class, make sure you also

update the value returned here to match!

- **name (mandatory)**
  Returns the unique algorithm name, used for identifying the algorithm.
- **displayName (mandatory)**
  Returns the translated algorithm name.
- **group**
  Returns the name of the group this algorithm belongs to.
- **groupId**
  Returns the unique ID of the group this algorithm belongs to.
- **shortHelpString**
  Returns a localised short help string for the algorithm.
- **initAlgorithm (mandatory)**
  Here we define the inputs and outputs of the algorithm.

  `INPUT` and `OUTPUT` are recommended names for the main input and main output parameters, respectively.

  If a parameter depends on another parameter, `parentParameterName` is used to specify this relationship (could be the field / band of a layer or the distance units of a layer).
- **processAlgorithm (mandatory)**
  This is where the processing takes place.

  Parameters are retrieved using special purpose functions, for instance `parameterAsSource` and `parameterAsDouble`.

  `processing.run` can be used to run other processing algorithms from a processing algorithm. The first parameter is the name of the algorithm, the second is a dictionary of the parameters to the algorithm. `is_child_algorithm` is normally set to `True` when running an algorithm from within another algorithm. `context` and `feedback` inform the algorithm about the environment to run in and the channel for communicating with the user (catching cancel request, reporting progress, providing textual feedback). When using the (parent) algorithm's parameters as parameters to "child" algorithms, the original parameter values should be used (e.g. `parameters['OUTPUT']`).

  It is good practice to check the feedback object for cancelation as much as is sensibly possible! Doing so allows for responsive cancelation, instead of forcing users to wait for unwanted processing to occur.

  The algorithm should return values for all the output parameters it has defined as a dictionary. In this case, that's the buffer and rasterized output layers, and the count of features processed. The dictionary keys must match the original parameter/output names.

23.9.2 The @alg decorator

Using the @alg decorator, you can create your own algorithms by writing the Python code and adding a few extra lines to supply additional information needed to make it a proper Processing algorithm. This simplifies the creation of algorithms and the specification of inputs and outputs.

One important limitation with the decorator approach is that algorithms created in this way will always be added to a user's Processing Scripts provider – it is not possible to add these algorithms to a custom provider, e.g. for use in plugins.

The following code uses the @alg decorator to

1. use a vector layer as input
2. count the number of features
3. do a buffer operation
4. create a raster layer from the result of the buffer operation

5. returns the buffer layer, raster layer and number of features

```
from qgis import processing
from qgis.processing import alg
from qgis.core import QgsProject

@alg(name='bufferrasteralg', label='Buffer and export to raster (alg)',
     group='examplescripts', group_label='Example scripts')
# 'INPUT' is the recommended name for the main input parameter
@alg.input(type=alg.SOURCE, name='INPUT', label='Input vector layer')
# 'OUTPUT' is the recommended name for the main output parameter
@alg.input(type=alg.RASTER_LAYER_DEST, name='OUTPUT',
           label='Raster output')
@alg.input(type=alg.VECTOR_LAYER_DEST, name='BUFFER_OUTPUT',
           label='Buffer output')
@alg.input(type=alg.DISTANCE, name='BUFFERDIST', label='BUFFER DISTANCE',
           default=1.0)
@alg.input(type=alg.DISTANCE, name='CELLSIZE', label='RASTER CELL SIZE',
           default=10.0)
@alg.output(type=alg.NUMBER, name='NUMBEROFFEATURES',
            label='Number of features processed')

def bufferrasteralg(instance, parameters, context, feedback, inputs):
    """
    Description of the algorithm.
    (If there is no comment here, you will get an error)
    """
    input_featuresource = instance.parameterAsSource(parameters,
                                                     'INPUT', context)
    numfeatures = input_featuresource.featureCount()
    bufferdist = instance.parameterAsDouble(parameters, 'BUFFERDIST',
                                            context)
    rastercellsize = instance.parameterAsDouble(parameters, 'CELLSIZE',
                                                context)
    if feedback.isCanceled():
        return {}
    buffer_result = processing.run('native:buffer',
                                   {'INPUT': parameters['INPUT'],
                                    'OUTPUT': parameters['BUFFER_OUTPUT'],
                                    'DISTANCE': bufferdist,
                                    'SEGMENTS': 10,
                                    'DISSOLVE': True,
                                    'END_CAP_STYLE': 0,
                                    'JOIN_STYLE': 0,
                                    'MITER_LIMIT': 10
                                    },
                                   is_child_algorithm=True,
                                   context=context,
                                   feedback=feedback)
    if feedback.isCanceled():
        return {}
    rasterized_result = processing.run('qgis:rasterize',
                                   {'LAYER': buffer_result['OUTPUT'],
                                    'EXTENT': buffer_result['OUTPUT'],
                                    'MAP_UNITS_PER_PIXEL': rastercellsize,
                                    'OUTPUT': parameters['OUTPUT']
                                   },
                                   is_child_algorithm=True, context=context,
                                   feedback=feedback)
    if feedback.isCanceled():
        return {}
    return {'OUTPUT': rasterized_result['OUTPUT'],
            'BUFFER_OUTPUT': buffer_result['OUTPUT'],
```

(continues on next page)

(continued from previous page)

```
62              'NUMBEROFFEATURES': numfeatures}
```

As you can see, it involves two algorithms ('native:buffer' and 'qgis:rasterize'). The last one ('qgis:rasterize') creates a raster layer from the buffer layer that was generated by the first one ('native:buffer').

The part of the code where this processing takes place is not difficult to understand if you have read the previous chapter. The first lines, however, need some additional explanation. They provide the information that is needed to turn your code into an algorithm that can be run from any of the GUI components, like the toolbox or the model designer.

These lines are all calls to the `@alg` decorator functions that help simplify the coding of the algorithm.

- The @alg decorator is used to define the name and location of the algorithm in the Toolbox.
- The @alg.input decorator is used to define the inputs of the algorithm.
- The @alg.output decorator is used to define the outputs of the algorithm.

For existing parameters and their correspondance, read *Input and output types for Processing Algorithms*.

23.9.3 Handing algorithm output

When you declare an output representing a layer (raster or vector), the algorithm will try to add it to QGIS once it is finished.

- Raster layer output: QgsProcessingParameterRasterDestination / alg.RASTER_LAYER_DEST.
- Vector layer output: QgsProcessingParameterVectorDestination / alg.VECTOR_LAYER_DEST.

So even if the `processing.run()` method does not add the layers it creates to the user's current project, the two output layers (buffer and raster buffer) will be loaded, since they are saved to the destinations entered by the user (or to temporary destinations if the user does not specify destinations).

If a layer is created as output of an algorithm, it should be declared as such. Otherwise, you will not be able to properly use the algorithm in the modeler, since what is declared will not match what the algorithm really creates.

You can return strings, numbers and more by specifying them in the result dictionary (as demonstrated for "NUMBEROFFEATURES"), but they should always be explicitly defined as outputs from your algorithm. We encourage algorithms to output as many useful values as possible, since these can be valuable for use in later algorithms when your algorithm is used as part of a model.

23.9.4 Communicating with the user

If your algorithm takes a long time to process, it is a good idea to inform the user about the progress. You can use `feedback` (`QgsProcessingFeedback`) for this.

The progress text and progressbar can be updated using two methods: `setProgressText(text)` and `setProgress(percent)`.

You can provide more information by using `pushCommandInfo(text)`, `pushDebugInfo(text)`, `pushInfo(text)` and `reportError(text)`.

If your script has a problem, the correct way of handling it is to raise a `QgsProcessingException`. You can pass a message as an argument to the constructor of the exception. Processing will take care of handling it and communicating with the user, depending on where the algorithm is being executed from (toolbox, modeler, Python console, …)

23.9.5 Documenting your scripts

You can document your scripts by overloading the `helpString()` and `helpUrl()` methods of `QgsProcessingAlgorithm`.

23.9.6 Flags

You can override the `flags()` method of `QgsProcessingAlgorithm` to tell QGIS more about your algorithm. You can for instance tell QGIS that the script shall be hidden from the modeler, that it can be canceled, that it is not thread safe, and more.

Tip

By default, Processing runs algorithms in a separate thread in order to keep QGIS responsive while the processing task runs. If your algorithm is regularly crashing, you are probably using API calls which are not safe to do in a background thread. Try returning the QgsProcessingAlgorithm.FlagNoThreading flag from your algorithm's flags() method to force Processing to run your algorithm in the main thread instead.

23.9.7 Best practices for writing script algorithms

Here's a quick summary of ideas to consider when creating your script algorithms and, especially, if you want to share them with other QGIS users. Following these simple rules will ensure consistency across the different Processing elements such as the toolbox, the modeler or the batch processing interface.

- Do not load resulting layers. Let Processing handle your results and load your layers if needed.
- Always declare the outputs your algorithm creates.
- Do not show message boxes or use any GUI element from the script. If you want to communicate with the user, use the methods of the feedback object (`QgsProcessingFeedback`) or throw a `QgsProcessingException`.

There are already many processing algorithms available in QGIS. You can find code on the QGIS repo.

23.10 Input and output types for Processing Algorithms

Here is the list of input and output types that are supported in Processing with their corresponding alg decorator constants, their name in the modeler designer if exposed.

23.10.1 Input types

Input name	Class	Alg constant	Description
Annotation layer	`QgsProcessingParameterAnnotationLayer`	`alg.ANNOTATION_LAYE`	An annotation layer
Area	`QgsProcessingParameterArea`		A numeric parameter representing an area measure
Authentication Configuration	`QgsProcessingParameterAuthConfig`	`alg.AUTH_CFG`	Allows users to select from available authentication configurations or create new authentication configurations

continues on next page

Table 23.1 – continued from previous page

Input name	Class	Alg constant	Description
Raster band	QgsProcessingParameterBand	alg.BAND	A band of a raster layer
Boolean	QgsProcessingParameterBoolean	alg.BOOL	A boolean value
Connection Name	QgsProcessingParameterProviderConnection		A selection from available registered connections for a particular data provider
Color	QgsProcessingParameterColor	alg.COLOR	A color
Coordinate Operation	QgsProcessingParameterCoordinateOperation	alg.COORDINATE_OPER	A coordinate operation (for CRS transformations)
CRS	QgsProcessingParameterCrs	alg.CRS	A Coordinate Reference System
Database Schema	QgsProcessingParameterDatabaseSchema	alg.DATABASE_SCHEMA	A database schema
Database Table	QgsProcessingParameterDatabaseTable	alg.DATABASE_TABLE	A database table
Date	ProcessingDateTimeParameterDate	alg.DATE	A pure date
Datetime	ProcessingDateTimeParameterDateTime	alg.DATETIME	A datetime
Distance	QgsProcessingParameterDistance	alg.DISTANCE	A double numeric parameter for distance values
DXF Layers	QgsProcessingParameterDxfLayers		A list of vector layers to export as DXF file
Enum	QgsProcessingParameterEnum	alg.ENUM	An enumeration, allowing for selection from a set of predefined values
Expression	QgsProcessingParameterExpression	alg.EXPRESSION	An expression
Extent	QgsProcessingParameterExtent	alg.EXTENT	A spatial extent defined by xmin, xmax, ymin, ymax
	QgsProcessingParameterField	alg.FIELD	A field in the attribute table of a vector layer
Field Aggregates	QgsProcessingParameterAggregate		A parameter for "aggregate" configurations, which consist of a definition of desired output fields, types, and aggregate used to populate them
Fields Mapper	QgsProcessingParameterFieldMapping		A parameter for "field mapping" configurations, which consist of a definition of desired output fields, types, and expressions used to populate them
File/Folder	QgsProcessingParameterFile	alg.FILE	A filename of an existing file
	QgsProcessingParameterFileDestination	alg.FILE_DEST	A filename for a newly created output file
	QgsProcessingParameterFolderDestination	alg.FOLDER_DEST	A folder (destination folder)

continues on next page

Table 23.1 – continued from previous page

Input name	Class	Alg constant	Description
Geometry	QgsProcessingParameterGeometry	alg.GEOMETRY	A geometry
	ProcessingNumberParInteger	alg.INT	An integer
Print Layout	QgsProcessingParameterLayout	alg.LAYOUT	A layout
Print Layout Item	QgsProcessingParameterLayoutItem	alg.LAYOUT_ITEM	A layout item
Map Layer	QgsProcessingParameterMapLayer	alg.MAPLAYER	A map layer
Map Theme	QgsProcessingParameterMapTheme	alg.MAP_THEME	A project map theme
Matrix	QgsProcessingParameterMatrix	alg.MATRIX	A matrix
Mesh Dataset Groups	QgsProcessing-ParameterMesh-DatasetGroups		Dataset groups of mesh layer
Mesh Dataset Time	QgsProcessing-ParameterMesh-DatasetTime		Dataset index from a mesh layer by time setting
Mesh Layer	QgsProcessingParameterMeshLayer	alg.MESH_LAYER	A mesh layer
Multiple Input	QgsProcessingParameterMultipleLayers	alg.MULTILAYER	A set of layers
Number	ProcessingNumberParDouble	alg.NUMBER	A numerical value
Point	QgsProcessingParameterPoint	alg.POINT	A point
Point Cloud Attribute	QgsProcessingParameterPointCloudAttribute		An attribute from a point cloud layer
	QgsProcessing-ParameterPoint-CloudDestination	alg.POINTCLOUD_LAYE	A point cloud layer destination parameter, for specifying the destination path for a point cloud layer created by the algorithm
Point Cloud Layer	QgsProcessing-ParameterPoint-CloudLayer	alg.POINTCLOUD_LAYE	A point cloud layer
	QgsProcessingParameterProvider-Connection	alg.PROVIDER_CONNEC	An available connection for a database provider
Range	QgsProcessingParameterRange	alg.RANGE	A number range
Raster Layer	QgsProcessing-ParameterRaster-Layer	alg.RASTER_LAYER	A raster layer
	QgsProcessingParameterRasterDestination	alg.RASTER_LAYER_DE	A raster layer destination parameter, for specifying the destination path for a raster layer created by the algorithm
Scale	QgsProcessingParameterScale	alg.SCALE	A map scale

continues on next page

Table 23.1 – continued from previous page

Input name	Class	Alg constant	Description
	QgsProcessingParameterFeatureSink	alg.SINK	A feature sink
String	QgsProcessingParameterString	alg.STRING	A text string
TIN Creation Layers	QgsProcessingParameterTinInputLayers		Selection of multiple layers to create a TIN with vertices and/or break lines
Time	ProcessingDateTimeParameterTime	alg.TIME	A pure time
Vector Features	QgsProcessingParameterFeatureSource	alg.SOURCE	A feature source
Vector Layer	QgsProcessingParameterVectorLayer	alg.VECTOR_LAYER	A vector layer
	QgsProcessingParameterVectorDestination	alg.VECTOR_LAYER_DE	A vector layer destination parameter, for specifying the destination path for a vector layer created by the algorithm
Vector Tile Writer Layers	QgsProcessingParameterVectorTileDestination		A vector tile layer destination parameter, for specifying the destination path for a vector tile layer created by the algorithm
Volume	QgsProcessingParameterVolume		A numeric parameter representing a volume measure

23.10.2 Output types

Class	Alg constant	Description
QgsProcessingOutputBoolean	alg.BOOL	A boolean value
QgsProcessingOutputNumber	alg.DISTANCE	A double numeric parameter for distance values
QgsProcessingOutputFile	alg.FILE	A filename of an existing file
QgsProcessingOutputFolder	alg.FOLDER	A folder
QgsProcessingOutputHtml	alg.HTML	HTML
QgsProcessingOutputNumber	alg.INT	A integer
QgsProcessingOutputLayerDefinition	alg.LAYERDEF	A layer definition
QgsProcessingOutputMapLayer	alg.MAPLAYER	A map layer
QgsProcessingOutputMultipleLayers	alg.MULTILAYER	A set of layers
QgsProcessingOutputNumber	alg.NUMBER	A numerical value
QgsProcessingOutputPointCloudLayer	alg.POINTCLOUD_LAYER	A point cloud layer
QgsProcessingOutputRasterLayer	alg.RASTER_LAYER	A raster layer
QgsProcessingOutputString	alg.STRING	A text string
QgsProcessingOutputVectorLayer	alg.VECTOR_LAYER	A vector layer
QgsProcessingOutputVectorTileLayer		A vector tile layer

23.11 Configuring external applications

The processing framework can be extended using additional applications. Algorithms that rely on external applications are managed by their own algorithm providers. Additional providers can be found as separate plugins, and installed using the QGIS Plugin Manager.

This section will show you how to configure the Processing framework to include these additional applications, and it will explain some particular features of the algorithms based on them. Once you have correctly configured the system, you will be able to execute external algorithms from any component like the toolbox or the model designer, just like you do with any other algorithm.

By default, algorithms that rely on an external application not shipped with QGIS are not enabled. You can enable them in the Processing settings dialog if they are installed on your system.

23.11.1 A note for Windows users

If you are not an advanced user and you are running QGIS on Windows, you might not be interested in reading the rest of this chapter. Make sure you install QGIS in your system using the standalone installer. That will automatically install SAGA and GRASS in your system and configure them so they can be run from QGIS. All the algorithms from these providers will be ready to be run without needing any further configuration. If installing with the OSGeo4W application, make sure that you also select SAGA and GRASS for installation.

23.11.2 A note on file formats

When using external software, opening a file in QGIS does not mean that it can be opened and processed in that other software. In most cases, other software can read what you have opened in QGIS, but in some cases, that might not be true. When using databases or uncommon file formats, whether for raster or vector layers, problems might arise. If that happens, try to use well-known file formats that you are sure are understood by both programs, and check the console output (in the log panel) to find out what is going wrong.

You might for instance get trouble and not be able to complete your work if you call an external algorithm with a GRASS raster layers as input. For this reason, such layers will not appear as available to algorithms.

You should, however, not have problems with vector layers, since QGIS automatically converts from the original file format to one accepted by the external application before passing the layer to it. This adds extra processing time, which might be significant for large layers, so do not be surprised if it takes more time to process a layer from a DB connection than a layer from a Shapefile format dataset of similar size.

Providers not using external applications can process any layer that you can open in QGIS, since they open it for analysis through QGIS.

All raster and vector output formats produced by QGIS can be used as input layers. Some providers do not support certain formats, but all can export to common formats that can later be transformed by QGIS automatically. As for input layers, if a conversion is needed, that might increase the processing time.

23.11.3 A note on vector layer selections

External applications may also be made aware of the selections that exist in vector layers within QGIS. However, that requires rewriting all input vector layers, just as if they were originally in a format not supported by the external application. Only when no selection exists, or the *Use only selected features* option is not enabled in the processing general configuration, can a layer be directly passed to an external application.

In other cases, exporting only selected features is needed, which causes longer execution times.

23.11.4 Using third-party Providers

SAGA

SAGA algorithms can be run from QGIS if SAGA is included with the QGIS installation.

If you are running Windows, both the stand-alone installer and the OSGeo4W installer include SAGA.

About SAGA grid system limitations

Most SAGA algorithms that require several input raster layers require them to have the same grid system. That is, they must cover the same geographic area and have the same cell size, so their corresponding grids match. When calling SAGA algorithms from QGIS, you can use any layer, regardless of its cell size and extent. When multiple raster layers are used as input for a SAGA algorithm, QGIS resamples them to a common grid system and then passes them to SAGA (unless the SAGA algorithm can operate with layers from different grid systems).

The definition of that common grid system is controlled by the user, and you will find several parameters in the SAGA group of the settings window to do so. There are two ways of setting the target grid system:

- Setting it manually. You define the extent by setting the values of the following parameters:
  - *Resampling min X*
  - *Resampling max X*
  - *Resampling min Y*
  - *Resampling max Y*
  - *Resampling cellsize*

  Notice that QGIS will resample input layers to that extent, even if they do not overlap with it.
- Setting it automatically from input layers. To select this option, just check the *Use min covering grid system for resampling* option. All the other settings will be ignored and the minimum extent that covers all the input layers will be used. The cell size of the target layer is the maximum of all cell sizes of the input layers.

For algorithms that do not use multiple raster layers, or for those that do not need a unique input grid system, no resampling is performed before calling SAGA, and those parameters are not used.

Limitations for multi-band layers

Unlike QGIS, SAGA has no support for multi-band layers. If you want to use a multiband layer (such as an RGB or multispectral image), you first have to split it into single-banded images. To do so, you can use the 'SAGA/Grid - Tools/Split RGB image' algorithm (which creates three images from an RGB image) or the 'SAGA/Grid - Tools/Extract band' algorithm (to extract a single band).

Limitations in cell size

SAGA assumes that raster layers have the same cell size in the X and Y axis. If you are working with a layer with different values for horizontal and vertical cell size, you might get unexpected results. In this case, a warning will be added to the processing log, indicating that an input layer might not be suitable to be processed by SAGA.

Logging

When QGIS calls SAGA, it does so using its command-line interface, thus passing a set of commands to perform all the required operations. SAGA shows its progress by writing information to the console, which includes the percentage of processing already done, along with additional content. This output is filtered and used to update the progress bar while the algorithm is running.

Both the commands sent by QGIS and the additional information printed by SAGA can be logged along with other processing log messages, and you might find them useful to track what is going on when QGIS runs a SAGA algorithm. You will find two settings, namely *Log console output* and *Log execution commands*, to activate that logging mechanism.

Most other providers that use external applications and call them through the command-line have similar options, so you will find them as well in other places in the processing settings list.

R scripts and libraries

To enable R in Processing you need to install the **Processing R Provider** plugin and configure R for QGIS.

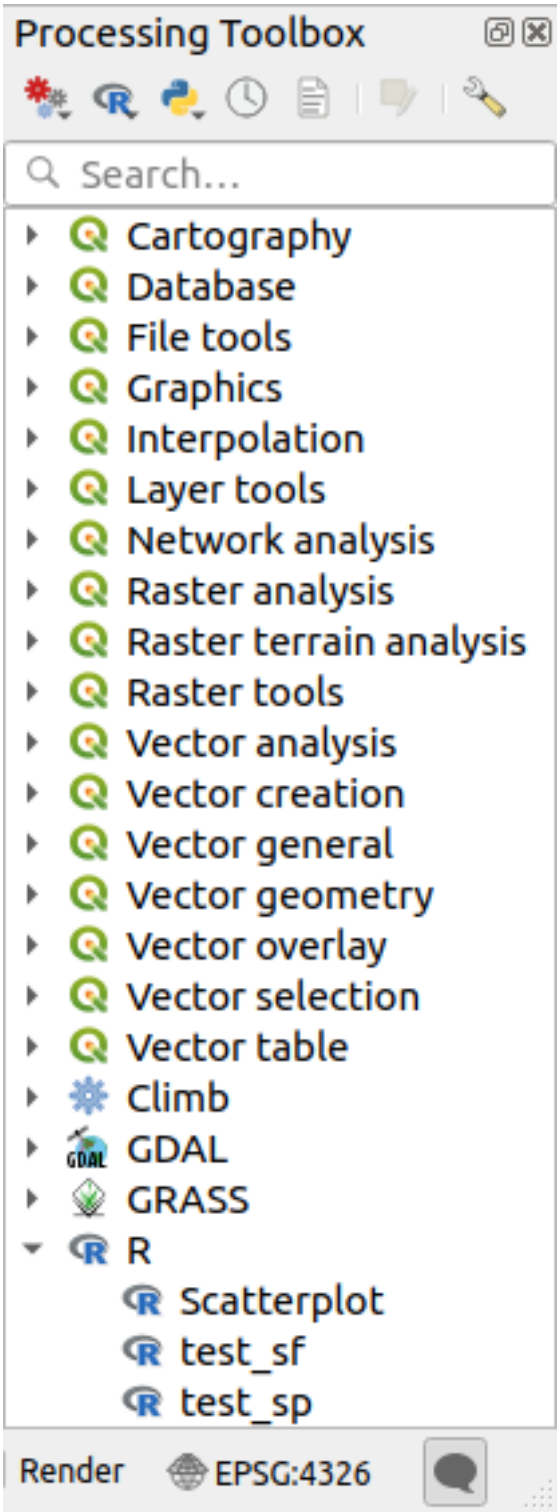
Configuration is done in *Provider* ► *R* in the *Processing* tab of *Settings* ► *Options*.

Depending on your operating system, you may have to use *R folder* to specify where your R binaries are located.

> **Note**
>
> On **Windows** the R executable file is normally in a folder (`R-<version>`) under `C:\Program Files\R\`. Specify the folder and **NOT** the binary!
>
> On **Linux** you just have to make sure that the R folder is in the PATH environment variable. If `R` in a terminal window starts R, then you are ready to go.

After installing the **Processing R Provider** plugin, you will find some example scripts in the *Processing Toolbox*:

- *Scatterplot* runs an R function that produces a scatter plot from two numerical fields of the provided vector layer.
- *test_sf* does some operations that depend on the `sf` package and can be used to check if the R package `sf` is installed. If the package is not installed, R will try to install it (and all the packages it depends on) for you, using the *Package repository* specified in *Provider* ► *R* in the Processing options. The default is https://cran.r-project.org/. Installing may take some time…
- *test_sp* can be used to check if the R package `sp` is installed. If the package is not installed, R will try to install it for you.



If you have R configured correctly for QGIS, you should be able to run these scripts.

Adding R scripts from the QGIS collection

R integration in QGIS is different from that of SAGA in that there is not a predefined set of algorithms you can run (except for some example script that come with the *Processing R Provider* plugin).

A set of example R scripts is available in the QGIS Repository. Perform the following steps to load and enable them using the *QGIS Resource Sharing* plugin.

1. Add the *QGIS Resource Sharing* plugin (you may have to enable *Show also experimental plugins* in the Plugin Manager *Settings*)
2. Open it (*Plugins* ► *Resource Sharing* ► *Resource Sharing*)
3. Choose the *Settings* tab
4. Click *Reload repositories*
5. Choose the *All* tab
6. Select *QGIS R script collection* in the list and click on the *Install* button
7. The collection should now be listed in the *Installed* tab
8. Close the plugin
9. Open the *Processing Toolbox*, and if everything is OK, the example scripts will be present under R, in various groups (only some of the groups are expanded in the screenshot below).

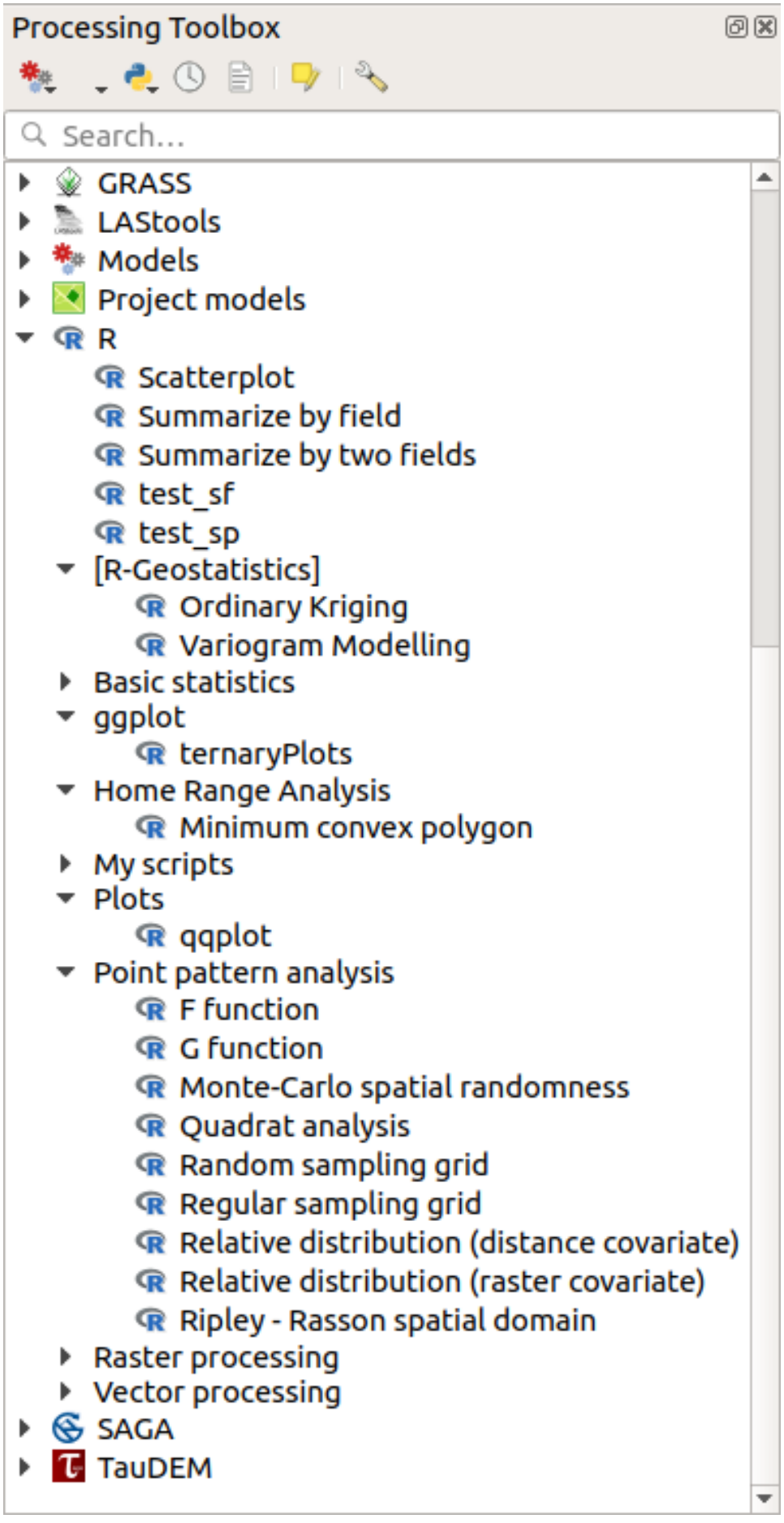
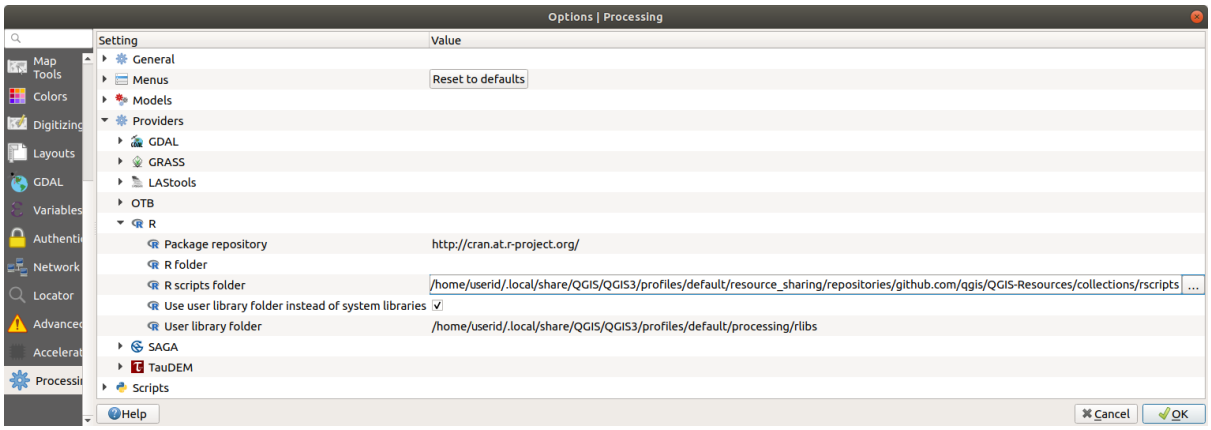


Fig. 23.35: The *Processing Toolbox* with some R scripts shown

The scripts at the top are the example scripts from the *Processing R Provider* plugin.

10. If, for some reason, the scripts are not available in the *Processing Toolbox*, you can try to:
    1. Open the Processing settings (*Settings* ► *Options* ► *Processing* tab)
    2. Go to *Providers* ► *R* ► *R scripts folder*
       - On Ubuntu, set the path to (or, better, include in the path):

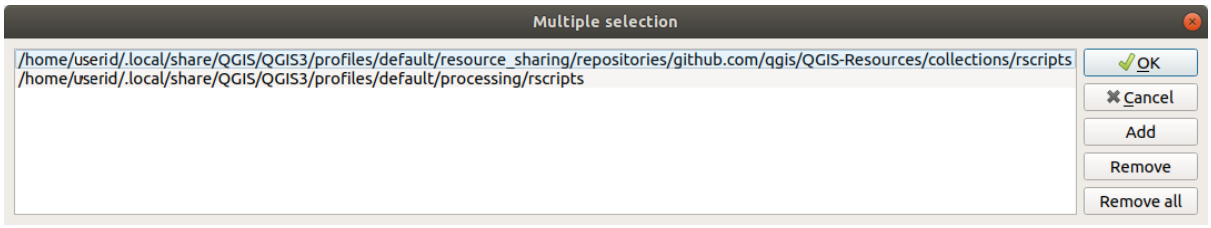
         /home/<user>/.local/share/QGIS/QGIS3/profiles/default/resource_sharing/repositories/github.com/qgis/QGIS-Resources/collections/rscripts



       - On Windows, set the path to (or, better, include in the path):

         C:\Users\<user>\AppData\Roaming\QGIS\QGIS3\profiles\default\resource_sharing\repositories\github.com\qgis\QGIS-Resources\collections\rscripts

    To edit, double-click. You can then choose to just paste / type the path, or you can navigate to the directory by using the … button and press the *Add* button in the dialog that opens. It is possible to provide several directories here. They will be separated by a semicolon (";").



If you would like to get all the R scrips from the QGIS 2 on-line collection, you can select *QGIS R script collection (from QGIS 2)* instead of *QGIS R script collection*. You will probably find that scripts that depend on vector data input or output will not work.

Creating R scripts

You can write scripts and call R commands, as you would do from R. This section shows you the syntax for using R commands in QGIS, and how to use QGIS objects (layers, tables) in them.

To add an algorithm that calls an R function (or a more complex R script that you have developed and you would like to have available from QGIS), you have to create a script file that performs the R commands.

R script files have the extension `.rsx`, and creating them is pretty easy if you just have a basic knowledge of R syntax and R scripting. They should be stored in the R scripts folder. You can specify the folder (*R scripts folder*) in the *R* settings group in Processing settings dialog).

Let's have a look at a very simple script file, which calls the R method `spsample` to create a random grid within the boundary of the polygons in a given polygon layer. This method belongs to the `maptools` package. Since almost all the algorithms that you might like to incorporate into QGIS will use or generate spatial data, knowledge of spatial packages like `maptools` and `sp`/`sf`, is very useful.

```
##Random points within layer extent=name
##Point pattern analysis=group
##Vector_layer=vector
##Number_of_points=number 10
##Output=output vector
library(sp)
spatpoly = as(Vector_layer, "Spatial")
pts=spsample(spatpoly,Number_of_points,type="random")
spdf=SpatialPointsDataFrame(pts, as.data.frame(pts))
Output=st_as_sf(spdf)
```

The first lines, which start with a double Python comment sign (`##`), define the display name and group of the script, and tell QGIS about its inputs and outputs.

Note

To find out more about how to write your own R scripts, have a look at the R Intro section in the training manual and consult the *QGIS R Syntax* section.

When you declare an input parameter, QGIS uses that information for two things: creating the user interface to ask the user for the value of that parameter, and creating a corresponding R variable that can be used as R function input.

In the above example, we have declared an input of type `vector`, named `Vector_layer`. When executing the algorithm, QGIS will open the layer selected by the user and store it in a variable named `Vector_layer`. So, the name of a parameter is the name of the variable that you use in R for accessing the value of that parameter (you should therefore avoid using reserved R words as parameter names).

Spatial parameters such as vector and raster layers are read using the `st_read()` (or `readOGR`) and `brick()` (or `readGDAL`) commands (you do not have to worry about adding those commands to your description file – QGIS will do it), and they are stored as `sf` (or `Spatial*DataFrame`) objects.

Table fields are stored as strings containing the name of the selected field.

Vector files can be read using the `readOGR()` command instead of `st_read()` by specifying `##load_vector_using_rgdal`. This will produce a `Spatial*DataFrame` object instead of an `sf` object.

Raster files can be read using the `readGDAL()` command instead of `brick()` by specifying `##load_raster_using_rgdal`.

If you are an advanced user and do not want QGIS to create the object for the layer, you can use `##pass_filenames` to indicate that you prefer a string with the filename. In this case, it is up to you to open the file before performing any operation on the data it contains.

With the above information, it is possible to understand the first lines of the R script (the first line not starting with a Python comment character).

```
library(sp)
spatpoly = as(Vector_layer, "Spatial")
pts=spsample(polyg,numpoints,type="random")
```

The `spsample` function is provided by the *sp* library, so the first thing we do is to load that library. The variable `Vector_layer` contains an `sf` object. Since we are going to use a function (`spsample`) from the *sp* library, we must convert the `sf` object to a `SpatialPolygonsDataFrame` object using the `as` function.

Then we call the `spsample` function with this object and the `numpoints` input parameter (which specifies the number of points to generate).

Since we have declared a vector output named `Output`, we have to create a variable named `Output` containing an `sf` object.

We do this in two steps. First we create a `SpatialPolygonsDataFrame` object from the result of the function, using the *SpatialPointsDataFrame* function, and then we convert that object to an `sf` object using the `st_as_sf` function (of the *sf* library).

You can use whatever names you like for your intermediate variables. Just make sure that the variable storing your final result has the defined name (in this case `Output`), and that it contains a suitable value (an `sf` object for vector layer output).

In this case, the result obtained from the `spsample` method had to be converted explicitly into an `sf` object via a `SpatialPointsDataFrame` object, since it is itself an object of class `ppp`, which can not be returned to QGIS.

If your algorithm generates raster layers, the way they are saved will depend on whether or not you have used the `##dontuserasterpackage` option. If you have used it, layers are saved using the `writeGDAL()` method. If not, the `writeRaster()` method from the `raster` package will be used.

If you have used the `##pass_filenames` option, outputs are generated using the `raster` package (with `writeRaster()`).

If your algorithm does not generate a layer, but a text result in the console instead, you have to indicate that you want the console to be shown once the execution is finished. To do so, just start the command lines that produce the results you want to print with the > ('greater than') sign. Only output from lines prefixed with > are shown. For instance, here is the description file of an algorithm that performs a normality test on a given field (column) of the attributes of a vector layer:

```
##layer=vector
##field=field layer
##nortest=group
library(nortest)
>lillie.test(layer[[field]])
```

The output of the last line is printed, but the output of the first is not (and neither are the outputs from other command lines added automatically by QGIS).

If your algorithm creates any kind of graphics (using the `plot()` method), add the following line (`output_plots_to_html` used to be `showplots`):

```
##output_plots_to_html
```

This will cause QGIS to redirect all R graphical outputs to a temporary file, which will be opened once R execution has finished.

Both graphics and console results will be available through the processing results manager.

For more information, please check the R scripts in the official QGIS collection (you download and install them using the *QGIS Resource Sharing* plugin, as explained in *Adding R scripts from the QGIS collection*). Most of them are rather simple and will greatly help you understand how to create your own scripts.

Note

The `sf`, `rgdal` and `raster` libraries are loaded by default, so you do not have to add the corresponding `library()` commands. However, other libraries that you might need have to be explicitly loaded by typing: `library(ggplot2)` (to load the `ggplot2` library). If the package is not already installed on your machine, Processing will try to download and install it. In this way the package will also become available in R Standalone. **Be aware** that if the package has to be downloaded, the script may take a long time to run the first time.

R libraries installed when running sf_test

The R script `sp_test` tries to load the R packages `sp` and `raster`.

The R script *sf_test* tries to load `sf` and `raster`. If these two packages are not installed, R may try to load and install them (and all the libraries that they depend on).

The following R libraries end up in `~/.local/share/QGIS/QGIS3/profiles/default/processing/rscripts` after `sf_test` has been run from the Processing Toolbox on Ubuntu with version 2.0 of the *Processing R Provider* plugin and a fresh install of *R* 3.4.4 (*apt* package `r-base-core` only):

```
abind, askpass, assertthat, backports, base64enc, BH, bit, bit64, blob, brew,␣
↪callr, classInt, cli, colorspace, covr, crayon, crosstalk, curl, DBI, deldir,
desc, dichromat, digest, dplyr, e1071, ellipsis, evaluate, fansi, farver, fastmap,␣
↪gdtools, ggplot2, glue, goftest, gridExtra, gtable, highr, hms,
htmltools, htmlwidgets, httpuv, httr, jsonlite, knitr, labeling, later, lazyeval,␣
↪leafem, leaflet, leaflet.providers, leafpop, leafsync, lifecycle, lwgeom,
magrittr, maps, mapview, markdown, memoise, microbenchmark, mime, munsell, odbc,␣
↪openssl, pillar, pkgbuild, pkgconfig, pkgload, plogr, plyr, png, polyclip,
praise, prettyunits, processx, promises, ps, purrr, R6, raster, RColorBrewer, Rcpp,
↪ reshape2, rex, rgeos, rlang, rmarkdown, RPostgres, RPostgreSQL,
rprojroot, RSQLite, rstudioapi, satellite, scales, sf, shiny, sourcetools, sp,␣
↪spatstat, spatstat.data, spatstat.utils, stars, stringi, stringr, svglite,
sys, systemfonts, tensor, testthat, tibble, tidyselect, tinytex, units, utf8, uuid,
↪ vctrs, viridis, viridisLite, webshot, withr, xfun, XML, xtable
```

GRASS

Configuring GRASS is not much different from configuring SAGA. First, the path to the GRASS folder has to be defined, but only if you are running Windows.

By default, the Processing framework tries to configure its GRASS connector to use the GRASS distribution that ships along with QGIS. This should work without problems for most systems, but if you experience problems, you might have to configure the GRASS connector manually. Also, if you want to use a different GRASS installation, you can change the setting to point to the folder where the other version is installed. GRASS 7 is needed for algorithms to work correctly.

If you are running Linux, you just have to make sure that GRASS is correctly installed, and that it can be run without problem from a terminal window.

GRASS algorithms use a region for calculations. This region can be defined manually using values similar to the ones found in the SAGA configuration, or automatically, taking the minimum extent that covers all the input layers used to execute the algorithm each time. If the latter approach is the behavior you prefer, just check the *Use min covering region* option in the GRASS configuration parameters.

LAStools

To use LAStools in QGIS, you need to download and install LAStools on your computer and install the LAStools plugin (available from the official repository) in QGIS.

On Linux platforms, you will need Wine to be able to run some of the tools.

LAStools is activated and configured in the Processing options (*Settings* ► *Options*, *Processing* tab, *Providers* ► *LAStools*), where you can specify the location of LAStools (*LAStools folder*) and Wine (*Wine folder*). On Ubuntu, the default Wine folder is `/usr/bin`.

OTB Applications

OTB applications are fully supported within the QGIS Processing framework.

OTB (Orfeo ToolBox) is an image processing library for remote sensing data. It also provides applications that provide image processing functionalities. The list of applications and their documentation are available in OTB CookBook

> **Note**
>
> Note that OTB is not distributed with QGIS and needs to be installed separately. Binary packages for OTB can be found on the download page.

To configure QGIS processing to find the OTB library:

1. Open the processing settings: *Settings* ► *Options* ► *Processing*
2. You can see `OTB` under *Providers* menu:
    1. Expand the *OTB* entry
    2. Set the *OTB folder*. This is the location of your OTB installation.
    3. Set the *OTB application folder*. This is the location of your OTB applications (`<PATH_TO_OTB_INSTALLATION>/lib/otb/applications`)
    4. Click *OK* to save the settings and close the dialog.

If settings are correct, OTB algorithms will be available in the *Processing Toolbox*.

Documentation of OTB settings available in QGIS Processing

- **OTB folder**: This is the directory where OTB is available.
- **OTB application folder**: This is the location(s) of OTB applications.

  Multiple paths are allowed.
- **Logger level** (optional): Level of logger to use by OTB applications.

  The level of logging controls the amount of detail printed during algorithm execution. Possible values for logger level are `INFO`, `WARNING`, `CRITICAL`, `DEBUG`. This value is `INFO` by default. This is an advanced user configuration.
- **Maximum RAM to use** (optional): by default, OTB applications use all available system RAM.

  You can, however, instruct OTB to use a specific amount of RAM (in MB) using this option. A value of 256 is ignored by the OTB processing provider. This is an advanced user configuration.
- **Geoid file** (optional): Path to the geoid file.

  This option sets the value of the elev.dem.geoid and elev.geoid parameters in OTB applications. Setting this value globally enables users to share it across multiple processing algorithms. Empty by default.
- **SRTM tiles folder** (optional): Directory where SRTM tiles are available.

  SRTM data can be stored locally to avoid downloading of files during processing. This option sets the value of elev.dem.path and elev.dem parameters in OTB applications. Setting this value globally enables users to share it across multiple processing algorithms. Empty by default.

Compatibility and Troubleshoot

Starting from OTB 6.6.1, new releases of OTB are made compatible with at least the latest QGIS version available at that time.

If you have issues with OTB applications in QGIS Processing, please open an issue on the OTB bug tracker, using the `qgis` label.

Additional information about OTB and QGIS can be found in OTB Cookbook.

CHAPTER

TWENTYFOUR

PROCESSING PROVIDERS AND ALGORITHMS

Processing algorithms and their parameters (as presented in the user interface) are documented here.

24.1 QGIS algorithm provider

QGIS algorithm provider implements various analysis and geoprocessing operations using mostly only QGIS API. So almost all algorithms from this provider will work "out of the box" without any additional configuration.

This provider incorporates some algorithms from plugins and also adds its own algorithms.

24.1.1 3D Tiles

Convert B3DM to GLTF

Converts files from the legacy `.B3DM` format to `.GLTF` or `.GLB`.

Parameters

Label	Name	Type	Description
Input B3DM	`INPUT`	[3D Tile]	Input file to convert.
Output file	`OUTPUT`	[3D Tile] Default: `[Save to temporary file]`	Specify the output 3D tile file. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Output file	`OUTPUT`	[3D Tile]	Output layer in `.GLTF` format.

Python code

Algorithm ID: `native:b3dmtogltf`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Convert GLTF to vector features

Converts `.GLTF` or `.GLB` file contents to standard vector layer formats.

Parameters

Label	Name	Type	Description
Input GLTF	`INPUT`	[3D Tile]	Input file to convert.
Output polygons Optional	`OUTPUT_POLYGONS`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output vector layer to store polygon features. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Output lines Optional	`OUTPUT_LINES`	[vector: line] Default: `[Create temporary layer]`	Specify the output vector layer to store line features. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Output polygons	`OUTPUT_POLYGONS`	[vector: polygon]	A vector layer storing the polygon features in input layer.
Output lines	`OUTPUT_LINES`	[vector: line]	A vector layer storing the line features in input layer.

Python code

Algorithm ID: `native:gltftovector`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.2 Cartography

Align points to features

Calculates the rotation required to align point features with their nearest feature from another reference layer. A new field is added to the output layer which is filled with the angle (in degrees, clockwise) to the nearest reference feature.

Optionally, the output layer's symbology can be set to automatically use the calculated rotation field to rotate marker symbols. If desired, a maximum distance to use when aligning points can be set, to avoid aligning isolated points to distant features.

Hint

This algorithm is designed for use cases like aligning building point symbols to follow the nearest road direction.

Allows *features in-place modification* of point features

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: point]	Point features to calculate the rotation for
Reference layer	`REFERENCE_LAYER`	[vector: geometry]	Layer to find the closest feature from for rotation calculation
Maximum distance to consider Optional	`MAX_DISTANCE`	[numeric: double] Default: Not set	If no reference feature is found within this distance, no rotation is assigned to the point feature.
Angle field name	`FIELD_NAME`	[string] Default: 'rotation'	Field in which to store the rotation value.
Automatically apply symbology	`APPLY_SYMBOLOGY`	[boolean] Default: True	Rotates the symbol marker of the features using the angle field value
Aligned layer	`OUTPUT`	[vector: point] Default: `[Save to temporary file]`	Specify the rotated output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Aligned layer	`OUTPUT`	[vector: point]	The point layer appended with a rotation field. If loaded to QGIS, it is applied by default the input layer symbology, with a data-defined rotation of its marker symbol.

Python code

Algorithm ID: `native:angletonearest`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Combine style databases

Combines multiple QGIS style databases into a single style database. If items of the same type with the same name exist in different source databases these will be renamed to have unique names in the output combined database.

See also

Create style database from project

Parameters

Label	Name	Type	Description
Input databases	`INPUT`	[file] [list]	Files containing QGIS style items
Objects to combine	`OBJECTS`	[enumeration] [list]	Types of style items in the input databases you would like to put in the new database. These can be: • 0 — *Symbols* • 1 — *Color ramps* • 2 — *Text formats* • 3 — *Label settings*
Output style database	`OUTPUT`	[file] Default: `[Save to temporary file]`	Output `.XML` file combining the selected style items. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Color ramp count	`COLORRAMPS`	[numeric: integer]	
Label settings count	`LABELSETTINGS`	[numeric: integer]	
Output style database	`OUTPUT`	[file]	Output `.XML` file combining the selected style items
Symbol count	`SYMBOLS`	[numeric: integer]	
Text format count	`TEXTFORMATS`	[numeric: integer]	

Python code

Algorithm ID: `native:combinestyles`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Create categorized renderer from styles

Sets a vector layer's renderer to a categorized renderer using matching symbols from a style database. If no style file is specified, symbols from the user's current *symbol library* are used instead.

A specified expression or field is used to create categories for the renderer. Each category is individually matched to the symbols which exist within the specified QGIS XML style database. Whenever a matching symbol name is found, the category's symbol will be set to this matched symbol.

If desired, outputs can also be tables containing lists of the categories which could not be matched to symbols, and symbols which were not matched to categories.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	Vector layer to apply a categorized style to
Categorize using expression	`FIELD`	[expression]	Field or expression to categorize the features
Style database (leave blank to use saved symbols)	`STYLE`	[file]	File (`.XML`) containing the symbols to apply to the input layer categories. The file can be obtained from the Style Manager *Share symbols* tool. If no file is specified, QGIS local symbols library is used.
Use case-sensitive match to symbol names	`CASE_SENSITIVE`	[boolean] Default: False	If True (checked), applies a case sensitive comparison between the categories and symbols names
Ignore non-alphanumeric characters while matching	`TOLERANT`	[boolean] Default: False	If True (checked), non-alphanumeric characters in the categories and symbols names will be ignored, allowing greater tolerance during the match.
Non-matching categories Optional	`NON_MATCHING_C`	[vector: table] Default: `[Skip output]`	Output table for categories which do not match any symbol in the database. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Non-matching symbol names Optional	`NON_MATCHING_S`	[vector: table] Default: `[Skip output]`	Output table for symbols from the provided style database which do not match any category. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Non-matching categories	NON_MATCHING_C	[vector: table]	Lists categories which could not be matched to any symbol in the provided style database
Non-matching symbol names	NON_MATCHING_S	[vector: table]	Lists symbols from the provided style database which could not match any category
Categorized layer	OUTPUT	[same as input]	The input vector layer with the categorized style applied. No new layer is output.

Python code

Algorithm ID: `native:categorizeusingstyle`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Create style database from project

Extracts all style objects (symbols, color ramps, text formats and label settings) from a QGIS project.

The extracted symbols are saved to a QGIS style database (`XML` format), which can be managed and imported via the *Style Manager* dialog.

See also

Combine style databases

Parameters

Label	Name	Type	Description
Input project (leave blank to use current) Optional	`INPUT`	[file]	A QGIS project file to extract the style items from
Objects to extract	`OBJECTS`	[enumeration] [list]	Types of style items in the input project you would like to put in the new database. These can be: • 0 — *Symbols* • 1 — *Color ramps* • 2 — *Text formats* • 3 — *Label settings*
Output style database	`OUTPUT`	[file] Default: `[Save to temporary file]`	Specify the output `.XML` file for the selected style items. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Color ramp count	`COLORRAMPS`	[numeric: integer]	Number of color ramps
Label settings count	`LABELSETTINGS`	[numeric: integer]	Number of label settings
Output style database	`OUTPUT`	[file]	Output `.XML` file for the selected style items
Symbol count	`SYMBOLS`	[numeric: integer]	Number of symbols
Text format count	`TEXTFORMATS`	[numeric: integer]	Number of text formats

Python code

Algorithm ID: `native:stylefromproject`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Export atlas layout as image

Exports the atlas of a print layout as image files (e.g. PNG or JPEG images).

If a coverage layer is set, the selected layout's atlas settings exposed in this algorithm will be overwritten. In this case, an empty filter or sort by expression will turn those settings off.

Parameters

Basic parameters

Label	Name	Type	Description
Atlas layout	`LAYOUT`	[layout]	Layout to export
Coverage layer Optional	`COVERAGE_LAYER`	[vector: geometry]	Layer to use to generate the atlas
Filter expression	`FILTER_EXPRESSION`	[expression]	Expression to use to filter out atlas features
Sort expression Optional	`SORTBY_EXPRESS`	[expression]	Expression to use to sort the atlas features
Reverse sort order	`SORTBY_REVERSE`	[boolean] Default: False	Determines if sorting should be inverted. Used when a sort expression is provided.
Output filename expression	`FILENAME_EXPRESSIO`	[expression] Default: 'output_'||@atlas_featurer	Expression for use to generate filenames
Output folder	`FOLDER`	[folder]	Destination folder where the images will be generated

Advanced parameters

Label	Name	Type	Description
Map layers to assign to unlocked map item(s) Optional	`LAYERS`	[enumeration] [layer]	Layers to display in the map item(s) whose contents are not locked
Image format	`EXTENSION`	[enumeration] Default: png	File format of the generated output(s). The list of available formats varies depending on OS and installed drivers.
DPI Optional	`DPI` Default: Not set	[numeric: double]	DPI of the output file(s). If not set, the value in the print layout settings will be used.
Generate world file	`GEOREFERENCE`	[boolean] Default: True	Determines if a world file should be generated
Export RDF metadata	`INCLUDE_METADATA`	[boolean] Default: True	Determines if RDF metadata (title, author, …) should be generated
Enable antialiasing	`ANTIALIAS`	[boolean] Default: True	Determines if antialiasing should be enabled

Outputs

Label	Name	Type	Description
Image file	OUTPUT	[file]	Image files generated by the atlas layout

Python code

Algorithm ID: native:atlaslayouttoimage

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Export atlas layout as PDF (multiple files)

Exports the atlas of a print layout to multiple PDF files.

If a coverage layer is set, the selected layout's atlas settings exposed in this algorithm will be overwritten. In this case, an empty filter or sort by expression will turn those settings off.

Parameters

Basic parameters

Label	Name	Type	Description
Atlas layout	LAYOUT	[layout]	Layout to export
Coverage layer Optional	COVERAGE_LAYER	[vector: geometry]	Layer to use to generate the atlas
Filter expression	FILTER_EXPRESSION	[expression]	Expression to use to filter out atlas features
Sort expression Optional	SORTBY_EXPRESS	[expression]	Expression to use to sort the atlas features
Reverse sort order	SORTBY_REVERSE	[boolean] Default: False	Determines if sorting should be inverted. Used when a sort expression is provided.
Output filename Optional	OUTPUT_FILENAME	[expression]	Name pattern of the PDF output files.
Output folder	OUTPUT_FOLDER	[folder]	Destination folder for the output PDF files.

Advanced parameters

Label	Name	Type	Description
Map layers to assign to unlocked map item(s) Optional	LAYERS	[enumeration] [layer]	Layers to display in the map item(s) whose contents are not locked
DPI Optional	DPI Default: Not set	[numeric: double]	DPI of the output file(s). If not set, the value in the print layout settings will be used.
Always export as vectors	FORCE_VECTOR	[boolean] Default: False	Determines if vectorial data should be left as vectors
Always export as raster	FORCE_RASTER	[boolean] Default: False	Forces all the items in the map to be rasterized. This parameter takes precedence over the FORCE_VECTOR parameter.
Append georeference information	GEOREFERENCE	[boolean] Default: True	Determines if a world file should be generated
Export RDF metadata	INCLUDE_METADATA	[boolean] Default: True	Determines if RDF metadata (title, author, …) should be generated
Disable tiled raster layer exports	DISABLE_TILED	[boolean] Default: False	Determines if raster should be tiled
Simplify geometries to reduce output file size	SIMPLIFY	[boolean] Default: True	Determines if geometries should be simplified to reduce output file size
Text export	TEXT_FORMAT	[enumeration] Default: 0	Determines if text should be exported as path or text objects. *Possible options* are: • 0 - Always export text as paths (recommended) • 1 - Always export texts as text objects • 2 - Prefer exporting texts as text objects
Image compression	IMAGE_COMPRESSIO	[enumeration] Default: 0	Determines compression level of the image and how suitable the file could be for printing outputs or post-production in external applications. Possible options are: • 0 - Lossy (JPEG) • 1 - Lossless

Outputs

Label	Name	Type	Description
PDF file	OUTPUT	[file]	PDF file corresponding to the exported atlas layout

Python code

Algorithm ID: `native:atlaslayouttomultiplepdf`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Export atlas layout as PDF (single file)

Exports the atlas of a print layout as a single PDF file.

If a coverage layer is set, the selected layout's atlas settings exposed in this algorithm will be overwritten. In this case, an empty filter or sort by expression will turn those settings off.

Parameters

Basic parameters

Label	Name	Type	Description
Atlas layout	`LAYOUT`	[layout]	Layout to export
Coverage layer Optional	`COVERAGE_LAYER`	[vector: geometry]	Layer to use to generate the atlas
Filter expression	`FILTER_EXPRESSION`	[expression]	Expression to use to filter out atlas features
Sort expression Optional	`SORTBY_EXPRESS`	[expression]	Expression to use to sort the atlas features
Reverse sort order	`SORTBY_REVERSE`	[boolean] Default: False	Determines if sorting should be inverted. Used when a sort expression is provided.
PDF file	`OUTPUT`	[file] Default: [Save to temporary file]	Name (including path) of the output file. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Map layers to assign to unlocked map item(s) Optional	LAYERS	[enumeration] [layer]	Layers to display in the map item(s) whose contents are not locked
DPI Optional	DPI Default: Not set	[numeric: double]	DPI of the output file(s). If not set, the value in the print layout settings will be used.
Always export as vectors	FORCE_VECTOR	[boolean] Default: False	Determines if vectorial data should be left as vectors
Always export as raster	FORCE_RASTER	[boolean] Default: False	Forces all the items in the map to be rasterized. This parameter takes precedence over the FORCE_VECTOR parameter.
Append georeference information	GEOREFERENCE	[boolean] Default: True	Determines if a world file should be generated
Export RDF metadata	INCLUDE_METADATA	[boolean] Default: True	Determines if RDF metadata (title, author, …) should be generated
Disable tiled raster layer exports	DISABLE_TILED	[boolean] Default: False	Determines if raster should be tiled
Simplify geometries to reduce output file size	SIMPLIFY	[boolean] Default: True	Determines if geometries should be simplified to reduce output file size
Text export	TEXT_FORMAT	[enumeration] Default: 0	Determines if text should be exported as path or text objects. *Possible options* are: • 0 - Always export text as paths (recommended) • 1 - Always export texts as text objects • 2 - Prefer exporting texts as text objects
Image compression	IMAGE_COMPRESSIO	[enumeration] Default: 0	Determines compression level of the image and how suitable the file could be for printing outputs or post-production in external applications. Possible options are: • 0 - Lossy (JPEG) • 1 - Lossless

Outputs

Label	Name	Type	Description
PDF file	OUTPUT	[file]	PDF file corresponding to the exported atlas layout

Python code

Algorithm ID: `native:atlaslayouttopdf`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Export print layout as image

Exports a print layout as an image file (e.g. PNG or JPEG images)

Parameters

Basic parameters

Label	Name	Type	Description
Print layout	`LAYOUT`	[layout]	Layout to export
Image file	`OUTPUT`	[file] Default: [Save to temporary file]	Name (including path) of the output file. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Map layers to assign to unlocked map item(s) Optional	`LAYERS`	[enumeration] [layer]	Layers to display in the map item(s) whose contents are not locked
DPI Optional	`DPI` Default: Not set	[numeric: double]	DPI of the output file(s). If not set, the value in the print layout settings will be used.
Generate world file	`GEOREFERENCE`	[boolean] Default: True	Determines if a world file should be generated
Export RDF metadata	`INCLUDE_METADATA`	[boolean] Default: True	Determines if RDF metadata (title, author, …) should be generated
Enable antialiasing	`ANTIALIAS`	[boolean] Default: True	Determines if antialiasing should be enabled

Outputs

Label	Name	Type	Description
Image file	`OUTPUT`	[file]	Image file corresponding to the exported print layout

Python code

Algorithm ID: `native:printlayouttoimage`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Export print layout as PDF

Exports a print layout as a PDF file.

Parameters

Basic parameters

Label	Name	Type	Description
Print Layout	`LAYOUT`	[layout]	Layout to export
PDF file	`OUTPUT`	[file] Default: [Save to temporary file]	Name (including path) of the output file. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Map layers to assign to unlocked map item(s) Optional	LAYERS	[enumeration] [layer]	Layers to display in the map item(s) whose contents are not locked
DPI Optional	DPI Default: Not set	[numeric: double]	DPI of the output file(s). If not set, the value in the print layout settings will be used.
Always export as vectors	FORCE_VECTOR	[boolean] Default: False	Determines if vectorial data should be left as vectors
Always export as raster	FORCE_RASTER	[boolean] Default: False	Forces all the items in the map to be rasterized. This parameter takes precedence over the FORCE_VECTOR parameter.
Append georeference information	GEOREFERENCE	[boolean] Default: True	Determines if a world file should be generated
Export RDF metadata	INCLUDE_METADATA	[boolean] Default: True	Determines if RDF metadata (title, author, …) should be generated
Disable tiled raster layer exports	DISABLE_TILED	[boolean] Default: False	Determines if raster should be tiled
Simplify geometries to reduce output file size	SIMPLIFY	[boolean] Default: True	Determines if geometries should be simplified to reduce output file size
Text export	TEXT_FORMAT	[enumeration] Default: 0	Determines if text should be exported as path or text objects. *Possible options* are: • 0 - Always export text as paths (recommended) • 1 - Always export texts as text objects • 2 - Prefer exporting texts as text objects
Image compression	IMAGE_COMPRESSIO	[enumeration] Default: 0	Determines compression level of the image and how suitable the file could be for printing outputs or post-production in external applications. Possible options are: • 0 - Lossy (JPEG) • 1 - Lossless
Export layers as separate PDF files	SEPARATE_LAYERS	[boolean] Default: False	If True, then a separate PDF file will be created per layer per map item in the layout. Additionally, separate PDF files may be created for other complex layout items, resulting in a set of PDF files which contain logical atomic components of the layout.

Outputs

Label	Name	Type	Description
PDF file	`OUTPUT`	[file]	PDF file(s) corresponding to the exported print layout

Python code

Algorithm ID: `native:printlayouttopdf`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Extract labels

Extracts label information from a rendered map at a given extent and scale.

If a map theme is provided, the rendered map will match the visibility and symbology of that theme. If left blank, all visible layers from the project will be used. Extracted label information include: position (served as point geometries), the associated layer name and feature ID, label text, rotation (in degree, clockwise), multiline alignment, and font details.

Warning

This algorithm drops existing primary keys or FID values and regenerates them in output layers.

Parameters

Basic parameters

Label	Name	Type	Description
Map extent	`EXTENT`	[extent]	Extent of the map to extract the labels from Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Map scale	`SCALE`	[scale]	Extracted labels will be rendered using their properties set at this scale.
Map theme Optional	`MAP_THEME`	[maptheme]	A map theme displaying the layers to extract the labels from. If unset, labels of the currently visible layers are extracted.
Include unplaced labels	`INCLUDE_UNPLACED`	[boolean] Default: True	Specify whether all overlapping labels should be extracted, including the conflicting (thus unplaced) ones.
Extracted labels	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specify the output vector layer for the extent(s). *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Map resolution (in DPI) Optional	`DPI` Default: 96.0	[numeric: double]	

Outputs

Label	Name	Type	Description
Extracted labels	`OUTPUT`	[vector: point]	Point vector layer representing the fetched labels. Each feature has attributes identifying its source (layer, feature ID) and the assigned labeling properties (text, font, size, rotation, …). A default style with labeling and null symbol is also applied to the layer. **Warning** Because some of the generated fields have name with more than 10 characters, using the ESRI shapefile format (`.SHP`) to store the output may lead to unexpected rendering while loading the layer in QGIS.

Python code

Algorithm ID: `native:extractlabels`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Print layout map extent to layer

Creates a polygon layer containing the extent of a print layout map item (or items), with attributes specifying the map size (in layout units, i.e. the *reference map* units), scale and rotation.

If the map item parameter is specified, then only the matching map extent will be exported. If it is not specified, all map extents from the layout will be exported.

Optionally, a specific output CRS can be specified. If it is not specified, the original map item CRS will be used.

Parameters

Basic parameters

Label	Name	Type	Description
Print layout	`LAYOUT`	[enumeration]	A print layout in the current project
Map item Optional	`MAP`	[enumeration] Default: *All the map items*	The map item(s) whose information you want to extract. If none is provided then all the map items are processed.
Extent	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output vector layer for the extent(s). *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Overrride CRS Optional	`CRS`	[crs] Default: *The layout CRS*	Select the CRS for the layer in which the information will be reported.

Outputs

Label	Name	Type	Description
Map height	`HEIGHT`	[numeric: double]	
Extent	`OUTPUT`	[vector: polygon]	Output polygon vector layer containing extents of all the input layout map item(s)
Map rotation	`ROTATION`	[numeric: double]	
Map scale	`SCALE`	[numeric: double]	
Map width	`WIDTH`	[numeric: double]	

Python code

Algorithm ID: `native:printlayoutmapextenttolayer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Set layer style

Applies a provided style to a layer. The style must be defined in a `QML` file.

No new output are created: the style is immediately assigned to the layer.

Parameters

Label	Name	Type	Description
Input Layer	`INPUT`	[layer]	Input layer you want to apply the style to
Style file	`STYLE`	[file]	Path to the `.qml` file of the style

Outputs

Label	Name	Type	Description
	`OUTPUT`	[same as input]	The input layer with the new style assigned. No new layer is created.

Python code

Algorithm ID: `native:setlayerstyle`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Topological coloring

Assigns a color index to polygon features in such a way that no adjacent polygons share the same color index, whilst minimizing the number of colors required.

The algorithm allows choice of method to use when assigning colors.

A minimum number of colors can be specified if desired. The color index is saved to a new attribute named **color_id**.

The following example shows the algorithm with four different colors chosen; as you can see each color class has the same amount of features.

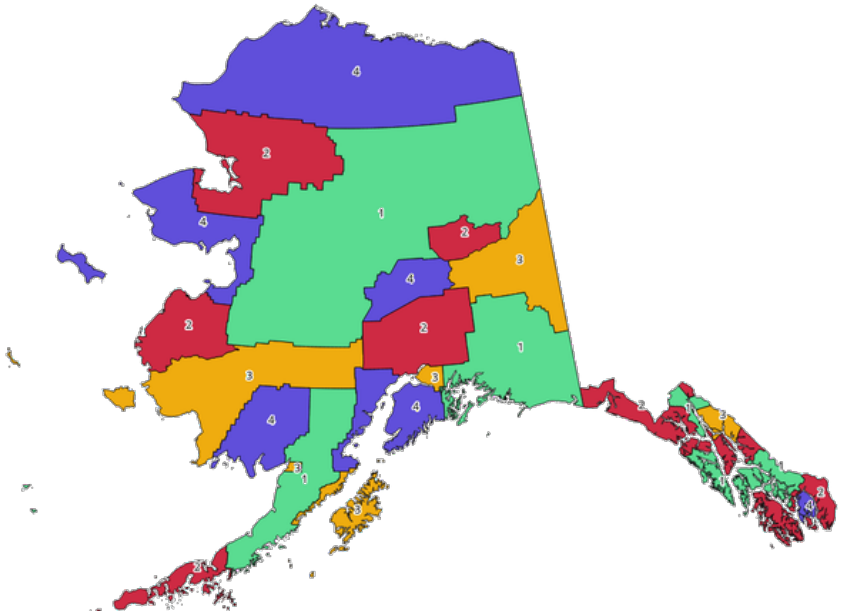


Fig. 24.1: Topological colors example

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: polygon]	The input polygon layer
Minimum number of colors	`MIN_COLORS`	[numeric: integer] Default: 4	The minimum number of colors to assign. Minimum 1, maximum 1000.
Minimum distance between features	`MIN_DISTANCE`	[numeric: double] Default: 0.0	Prevent nearby (but non-touching) features from being assigned equal colors. Minimum 0.0.
Balance color assignment	`BALANCE`	[enumeration] Default: 0	Options are: • 0 — By feature count Attempts to assign colors so that the count of features assigned to each individual color index is balanced. • 1 — By assigned area Assigns colors so that the total area of features assigned to each color is balanced. This mode can be useful to help avoid large features resulting in one of the colors appearing more dominant on a colored map. • 2 — By distance between colors Assigns colors in order to maximize the distance between features of the same color. This mode helps to create a more uniform distribution of colors across a map.

continues on next page

Table 24.4 – continued from previous page

Label	Name	Type	Description
Colored	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Colored	`OUTPUT`	[vector: polygon]	Polygon vector layer with an added `color_id` column

Python code

Algorithm ID: `qgis:topologicalcoloring`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Transfer annotations from main layer

Transfers all *annotations* from the main annotation layer in a project to a new annotation layer. Items placement can then be adjusted within the layer stack.

Parameters

Label	Name	Type	Description
New layer name	`LAYER_NAME`	[string] Default: 'Annotations'	Name of the annotations layer to create

Outputs

Label	Name	Type	Description
New layer name	`OUTPUT`	[layer]	A layer with items from the main annotation layer

Python code

Algorithm ID: `native:transferannotationsfrommain`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.3 Check Geometry

Dangle-end lines

Detects dangle-end lines in line geometries and reports them as errors. A dangle-end line is a line feature that terminates at a vertex connected to only one segment, resulting in an endpoint without a proper connection.

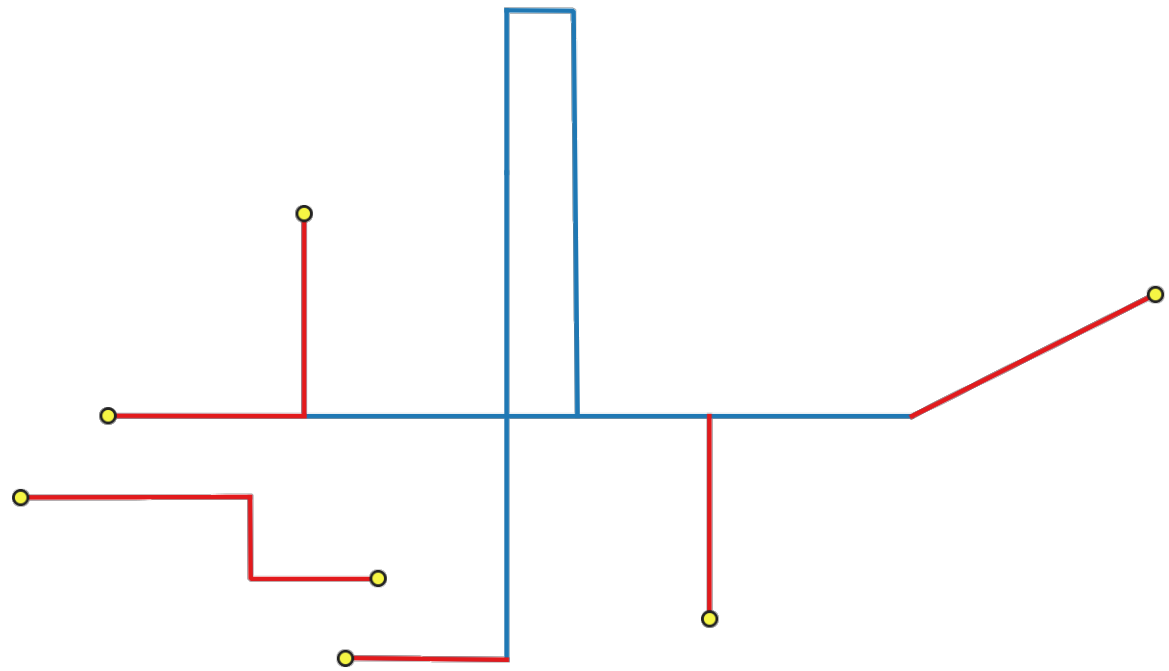


Fig. 24.2: Errors for dangle-end lines are reported on line features, and entire features with dangling ends are highlighted in red for clarity.

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line]	Layer with the geometries to check.
Unique feature identifier	`UNIQUE_ID`	[tablefield: any]	Field storing unique values for feature identification.
Dangle-end errors	`ERRORS`	[vector: point] Default: `[Create temporary layer]`	Specification of the output layer containing the errors location. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Dangle-end features Optional	`OUTPUT`	[vector: line] Default: `[Skip output]`	Line layer containing the identified dangle-end line features. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Tolerance	`TOLERANCE`	[numeric: integer] Default: 8	Numerical precision of geometric operations, given as an integer n, meaning that two vertices less than 10^{-n} apart (in map units) are considered to be merged.

Outputs

Label	Name	Type	Description
Dangle-end errors	ERRORS	[vector: point]	Output point layer representing the error locations and information. The output layer contains the following fields: • gc_layerid: the ID of the input layer. • gc_layername: the name of the input layer. • gc_partidx: the index of the feature's geometry part containing the dangle-end. • gc_ringidx: the index of the feature's geometry ring containing the dangle-end. • gc_vertidx: the vertex index of the dangle-end in the feature's geometry ring. • gc_errorx: the x coordinate of the dangle-end. • gc_errory: the y coordinate of the dangle-end. • gc_error • UNIQUE_ID field: the unique ID of the input feature with the dangle-end.
Dangle-end features	OUTPUT	[vector: line]	Output line layer containing the input dangle-end features. If no dangle-end features are found, the output layer will be empty. Available fields are the same as in the ERRORS output.

Python code

Algorithm ID: native:checkgeometrydangle

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Degenerate polygons

Checks the polygons with less than 3 points, which are degenerate polygons. Degenerate polygons are errors.

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: polygon]	Layer with the geometries to check.
Unique feature identifier	`UNIQUE_ID`	[tablefield: any]	Field storing unique values for feature identification.
Degenerate polygons errors	`ERRORS`	[vector: point] Default: `[Create temporary layer]`	Specification of the output layer containing the centroid of the degenerate polygons. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Degenerate polygons features Optional	`OUTPUT`	[vector: polygon] Default: `[Skip output]`	Specification of the output layer containing the degenerate polygons. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Tolerance	`TOLERANCE`	[numeric: integer] Default: 8	Numerical precision of geometric operations, given as an integer n, meaning that two vertices less than 10^{-n} apart (in map units) are considered to be merged.

Outputs

Label	Name	Type	Description
Degenerate polygons errors	ERRORS	[vector: point]	Output point layer representing the error locations and information. The output layer contains the following fields: • gc_layerid: the ID of the input layer. • gc_layername: the name of the input layer. • gc_partidx: the index of the geometry part of the degenerate polygon. • gc_ringidx: the index of the ring of the degenerate polygon. • gc_vertidx • gc_errorx: the x coordinate of the centroid of the degenerate polygon • gc_errory: the y coordinate of the centroid of the degenerate polygon • gc_error • UNIQUE_ID field: the unique ID of the input feature that is degenerate.
Degenerate polygons features	OUTPUT	[vector: polygon]	Output polygon layer with features containing the degenerate polygons. If no degenerate polygon features are found, the output layer will be empty. Available fields are the same as in the ERRORS output.

Python code

Algorithm ID: native:checkgeometrydegeneratepolygon

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Duplicated geometries

Checks for duplicated geometries in a vector layer, and reports them as errors.

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Layer with the geometries to check
Unique feature identifier	`UNIQUE_ID`	[tablefield: any]	Field storing unique values for feature identification
Duplicate geometries errors	`ERRORS`	[vector: point] Default: `[Create temporary layer]`	Specification of the output layer containing the errors location. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Duplicate geometries Optional	`OUTPUT`	[vector: same as input] Default: `[Skip output]`	Specification of the output layer containing the geometries that were found to be duplicated. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Tolerance	`TOLERANCE`	[numeric: integer] Default: 8	Numerical precision of geometric operations, given as an integer n, meaning that two vertices less than 10^{-n} apart (in map units) are considered to be merged.

Outputs

Label	Name	Type	Description
Duplicate geometries errors	ERRORS	[vector: point]	Output point layer representing the error locations and information. The output layer contains the following fields: • gc_layerid: the ID of the input layer. • gc_layername: the name of the input layer. • gc_partidx • gc_ringidx • gc_vertidx • gc_errorx: the x coordinate of the centroid of the duplicate geometry. • gc_errory: the y coordinate of the centroid of the duplicate geometry. • gc_error: the indices of the duplicate geometry (all except the highest index). • UNIQUE_ID field: the unique ID of the input feature that is a duplicate.
Duplicate geometries	OUTPUT	[vector: same as input]	Output layer containing the geometries that were found to be duplicated. Available fields are the same as in the ERRORS output.

Python code

Algorithm ID: native:checkgeometryduplicate

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Duplicated vertices

Checks for duplicated vertices in line or polygon geometries, and reports them as errors.

See also

Delete duplicated vertices, *Remove duplicate vertices*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line, polygon]	Layer with the geometries to check
Unique feature identifier	`UNIQUE_ID`	[tablefield: any]	Field storing unique values for feature identification
Duplicated vertices errors	`ERRORS`	[vector: point] Default: `[Create temporary layer]`	Specification of the output layer containing the errors location. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Duplicated vertices features Optional	`OUTPUT`	[vector: same as input] Default: `[Skip output]`	Specification of the output layer containing the features with duplicated vertices. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Tolerance	`TOLERANCE`	[numeric: integer] Default: 8	Numerical precision of geometric operations, given as an integer n, meaning that two vertices less than 10^{-n} apart (in map units) are considered to be merged.

Outputs

Label	Name	Type	Description
Duplicated vertices errors	ERRORS	[vector: point]	Output point layer representing the error locations and information. The output layer contains the following fields: • gc_layerid: the ID of the input layer. • gc_layername: the name of the input layer. • gc_partidx: the index of the feature's geometry part containing the duplicate vertex. • gc_ringidx: the index of the feature's geometry ring containing the duplicate vertex. • gc_vertidx: the index of the duplicate vertex in the feature's geometry ring. • gc_errorx: the x coordinate of the duplicate vertex. • gc_errory: the y coordinate of the duplicate vertex. • gc_error • UNIQUE_ID field: the unique ID of the input feature that has duplicate vertices.
Duplicated vertices features	OUTPUT	[vector: same as input]	Output layer containing the features with duplicated vertices. Available fields are the same as in the ERRORS output.

Python code

Algorithm ID: native:checkgeometryduplicatenodes

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Features inside polygon

Checks the input geometries contained in the polygons from the polygon layers list. A polygon layer can be checked against itself. Any contained features are reported as errors.

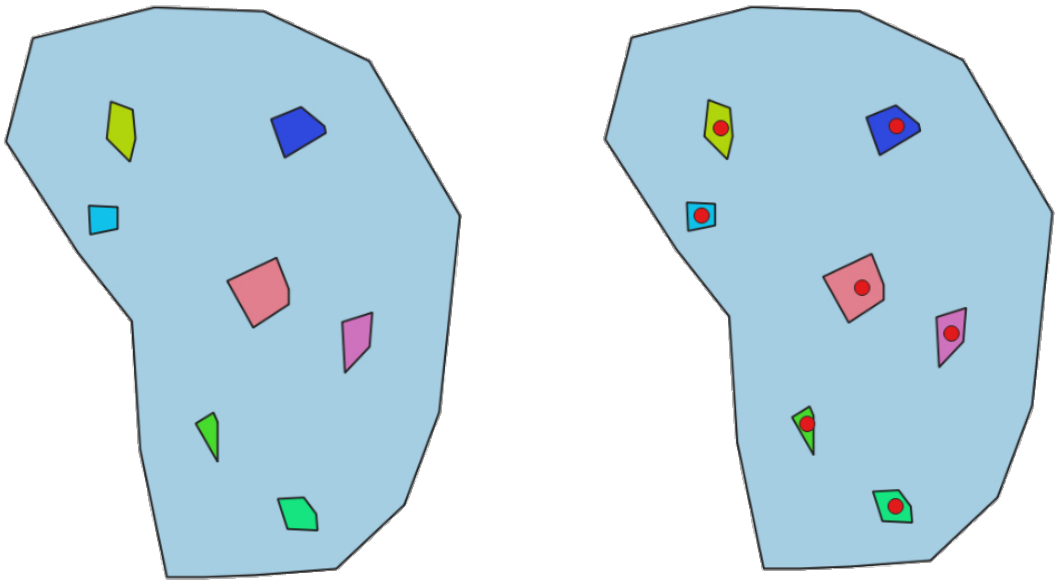


Fig. 24.3: Reporting errors on features inside a polygon.

See also

Delete features

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Layer with the geometries to check.
Unique feature identifier	`UNIQUE_ID`	[tablefield: any]	Field storing unique values for feature identification.
Polygon layers	`POLYGONS`	[vector: polygon][list]	List of polygon layers to check against.
Errors from contained features	`ERRORS`	[vector: point] Default: `[Create temporary layer]`	Point layer representing the contained features. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

continues on next page

Table 24.18 – continued from previous page

Label	Name	Type	Description
Contained features Optional	OUTPUT	[vector: same as input] Default: [Skip output]	Specification of the output layer containing features within the checked polygons. *One of*: • Skip Output • Create Temporary Layer (TEMPORARY_OUTPUT) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Tolerance	TOLERANCE	[numeric: integer] Default: 8	Numerical precision of geometric operations, given as an integer n, meaning that two vertices less than 10^{-n} apart (in map units) are considered to be merged.

Outputs

Label	Name	Type	Description
Errors from contained features	ERRORS	[vector: point]	Output point layer representing the error locations and information. The output layer contains the following fields: • gc_layerid: the ID of the input layer. • gc_layername: the name of the input layer. • gc_partidx • gc_ringidx • gc_vertidx • gc_errorx: the x coordinate of the centroid of the contained feature. • gc_errory: the y coordinate of the centroid of the contained feature. • gc_error: the layer name and feature ID of a polygon that contains the feature. • UNIQUE_ID field: the unique ID of the input feature that is contained within another.
Contained features	OUTPUT	[vector: same as input]	Output layer containing features within the checked polygons. Available fields are the same as in the ERRORS output.

Python code

Algorithm ID: `native:checkgeometrycontained`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Holes

Detects holes in polygon geometries and reports them as errors.

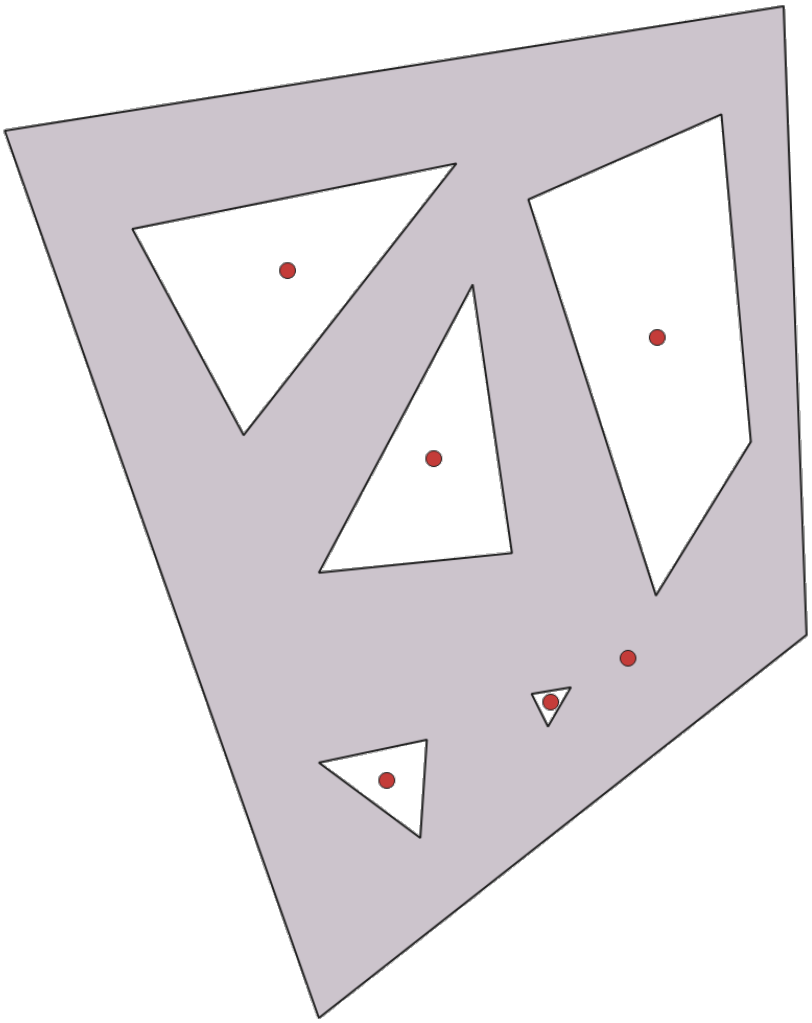


Fig. 24.4: Reporting errors for holes on a polygon feature.

See also

Fill holes

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: polygon]	Layer with the geometries to check
Unique feature identifier	`UNIQUE_ID`	[tablefield: any]	Field storing unique values for feature identification
Holes errors	`ERRORS`	[vector: point] Default: `[Create temporary layer]`	Specification of the output layer containing the errors location. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Polygons with holes Optional	`OUTPUT`	[vector: polygon] Default: `[Skip output]`	Polygon layer with the features containing holes. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Tolerance	`TOLERANCE`	[numeric: integer] Default: 8	Numerical precision of geometric operations, given as an integer n, meaning that two vertices less than 10^{-n} apart (in map units) are considered to be merged.

Outputs

Label	Name	Type	Description
Holes errors	`ERRORS`	[vector: point]	Output point layer representing the error locations and information. The output layer contains the following fields: • `gc_layerid`: the ID of the input layer. • `gc_layername`: the name of the input layer. • `gc_partidx`: the index of the feature's geometry part containing the hole. • `gc_ringidx`: the index of the feature's geometry ring containing the hole. • `gc_vertidx` • `gc_errorx`: the x coordinate of the centroid of the hole. • `gc_errory`: the y coordinate of the centroid of the hole. • `gc_error` • `UNIQUE_ID` field: the unique ID of the input feature that has a hole.
Polygon with holes	`OUTPUT`	[vector: polygon]	Output polygon layer with features containing holes. If no holes are found, the output layer will be empty. Available fields are the same as in the `ERRORS` output.

Python code

Algorithm ID: `native:checkgeometryhole`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Lines intersecting each other

Checks intersections between line geometries within a layer. Intersections between two different lines are errors.

See also

Line intersections, *Intersection*, *Self-intersections*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line]	Layer with the geometries to check.
Unique feature identifier	`UNIQUE_ID`	[tablefield: any]	Field storing unique values for feature identification.
Intersection errors	`ERRORS`	[vector: point] Default: `[Create temporary layer]`	Specification of the output layer containing the intersections location. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Intersecting feature Optional	`OUTPUT`	[vector: line] Default: `[Skip output]`	Specification of the output layer for features intersecting each other. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Tolerance	`TOLERANCE`	[numeric: integer] Default: 8	Numerical precision of geometric operations, given as an integer n, meaning that two vertices less than 10^{-n} apart (in map units) are considered to be merged.

Outputs

Label	Name	Type	Description
Intersection errors	`ERRORS`	[vector: point]	Output point layer representing the error locations and information. The output layer contains the following fields: • `gc_layerid`: the ID of the input layer. • `gc_layername`: the name of the input layer. • `gc_partidx`: the index of the feature's geometry part where the intersection occurs. • `gc_ringidx` • `gc_vertidx` • `gc_errorx`: the x coordinate of the intersection point. • `gc_errory`: the y coordinate of the intersection point. • `UNIQUE_ID` field: the unique ID of an intersecting input feature. • `gc_error`: the layer name and index of the other intersecting feature.
Intersecting feature	`OUTPUT`	[vector: line]	Output layer containing, for each identified intersection, the feature (with same `UNIQUE_ID` value) it belongs to. If no intersections are found, the output layer will be empty. Available fields are the same as in the `ERRORS` output.

Python code

Algorithm ID: `native:checkgeometrylineintersection`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Lines intersecting other layer

Checks if the input line layer features intersect with the check layer features. An input feature that intersects with a check layer feature is an error.

See also

Line intersections, *Intersection*, *Self-intersections*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line]	Layer with the geometries to check.
Unique feature identifier	`UNIQUE_ID`	[tablefield: any]	Field storing unique values for feature identification.
Check layer	`CHECK_LAYER`	[vector: line, polygon]	Layer with the geometries to check against.
Line intersecting other layer errors	`ERRORS`	[vector: point] Default: `[Create temporary layer]`	Specification of the output point layer containing the locations of intersection errors. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Line intersecting other layer features Optional	`OUTPUT`	[line] Default: `[Skip output]`	Line layer with the input features that intersect features in the check layer. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Tolerance	`TOLERANCE`	[numeric: integer] Default: 8	Numerical precision of geometric operations, given as an integer n, meaning that two vertices less than 10^{-n} apart (in map units) are considered to be merged.

Outputs

Label	Name	Type	Description
Line intersecting other layer errors	`ERRORS`	[vector: point]	Output point layer representing the error location and information. The output layer contains the following fields: • `gc_layerid`: the ID of the input layer. • `gc_layername`: the name of the input layer. • `gc_partidx`: the index of the feature's geometry part where the intersection occurs. • `gc_ringidx` • `gc_vertidx` • `gc_errorx`: the x coordinate of the intersection point. • `gc_errory`: the y coordinate of the intersection point. • `UNIQUE_ID` field: the unique ID of the intersecting input feature. • `gc_error`: the index of the check layer where the intersection occurs.
Line intersecting other layer features	`OUTPUT`	[vector: line]	Output layer containing, for each identified intersection, the input feature it belongs to. If no intersections are found, the output layer will be empty. Available fields are the same as in the `ERRORS` output.

Python code

Algorithm ID: `native:checkgeometrylinelayerintersection`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Overlaps

Calculates overlapping areas in polygon geometries, and reports areas smaller than a given minimum as errors.

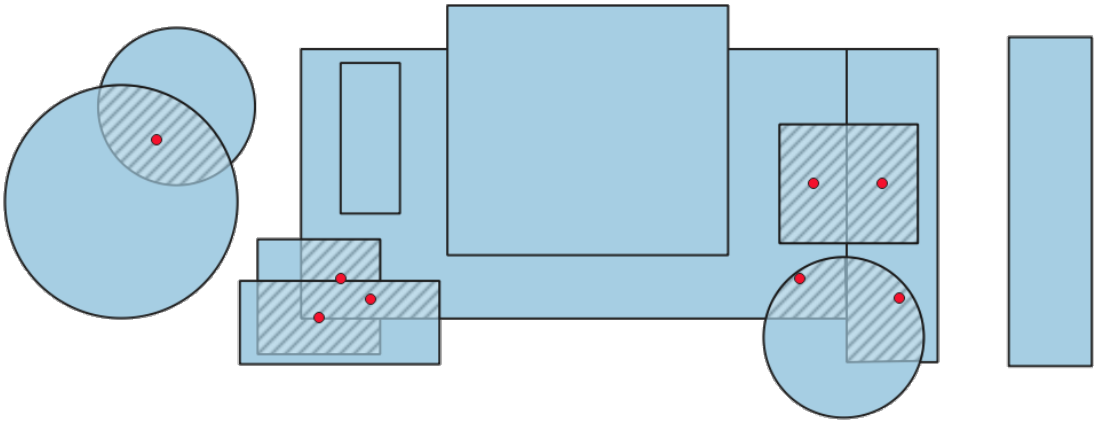


Fig. 24.5: Hashed polygons indicate overlapping areas smaller than the specified minimum.

See also

Delete overlaps

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: polygon]	Layer with the geometries to check.
Unique feature identifier	`UNIQUE_ID`	[tablefield: any]	Field storing unique values for feature identification.
Minimum overlap area	`MIN_OVERLAP_AR`	[numeric: double]	Minimum area of the overlap to be reported as an error. If set to 0, all overlaps are reported.
Overlap errors	`ERRORS`	[vector: point] Default: `[Create temporary layer]`	Specification of the output layer containing centroid points of the overlapping areas. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

continues on next page

Table 24.30 – continued from previous page

Label	Name	Type	Description
Overlap features Optional	`OUTPUT`	[vector: polygon] Default: `[Skip output]`	Specification of the output layer for features containing overlaps. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Tolerance	`TOLERANCE`	[numeric: integer] Default: 8	Numerical precision of geometric operations, given as an integer n, meaning that two vertices less than 10^{-n} apart (in map units) are considered to be merged.

Outputs

Label	Name	Type	Description
Overlap errors	`ERRORS`	[vector: point]	Output point layer representing the error locations and information. The output layer contains the following fields: • `gc_layerid`: the ID of the input layer. • `gc_layername`: the name of the input layer. • `gc_errorx`: the x coordinate of the centroid of the overlapping area. • `gc_errory`: the y coordinate of the centroid of the overlapping area. • `gc_error`: the area of the overlapping geometry. • `UNIQUE_ID` field: the unique ID of an overlapped input feature. • `gc_overlap_feature_{unique_id}`: the `UNIQUE_ID` field value for the other overlapping feature.
Overlap features	`OUTPUT`	[vector: polygon]	Output layer containing the overlapping areas. Available fields are the same as in the `ERRORS` output.

Python code

Algorithm ID: `native:checkgeometryoverlap`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Points outside lines

Checks if the points in the input layer are covered by a line in the selected line layers. A point not covered by a line is an error.

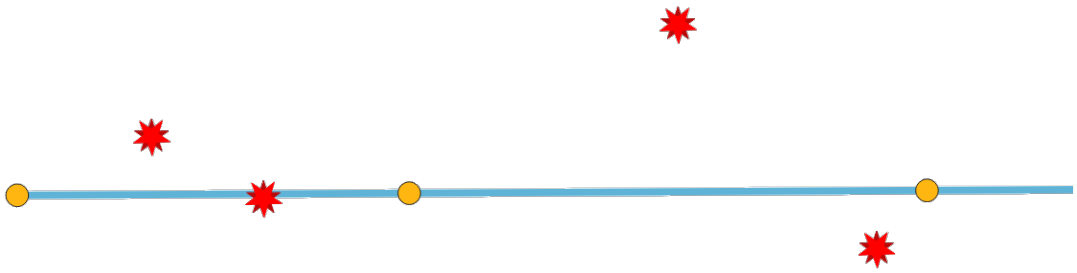


Fig. 24.6: Reporting errors on points not covered by a line.

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: point]	Layer with the geometries to check.
Unique feature identifier	`UNIQUE_ID`	[tablefield: any]	Field storing unique values for feature identification.
Line layers	`LINES`	[vector: line][list]	Layer(s) with the lines to check against.
Points not covered by a line	`ERRORS`	[vector: point] Default: `[Create temporary layer]`	Specification of the output layer containing points not covered by a line. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Tolerance	`TOLERANCE`	[numeric: integer] Default: 8	Numerical precision of geometric operations, given as an integer n, meaning that two vertices less than 10^{-n} apart (in map units) are considered to be merged.

Outputs

Label	Name	Type	Description
Points not covered by a line	`ERRORS`	[vector: point]	Output point layer representing the error locations and information. The output layer contains the following fields: • `gc_layerid`: the ID of the input layer. • `gc_layername`: the name of the input layer. • `gc_partidx` • `gc_ringidx` • `gc_vertidx` • `gc_errorx`: the x coordinate of point outside the lines. • `gc_errory`: the y coordinate of point outside the lines. • `gc_error` • `UNIQUE_ID` field: the unique ID of the input feature that is outside the lines.

Python code

Algorithm ID: `native:checkgeometrypointcoveredbyline`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Points outside polygons

Checks if points from the input layer are in polygons from the selected polygon layers. Points that are not fully inside polygons are errors.

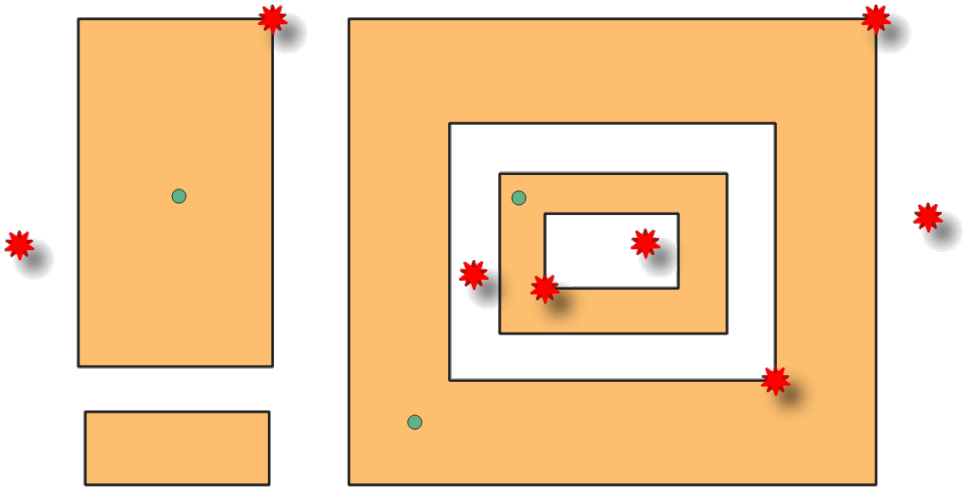


Fig. 24.7: Reporting errors on points outside polygons.

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: point]	Layer with the geometries to check.
Unique feature identifier	`UNIQUE_ID`	[tablefield: any]	Field storing unique values for feature identification.
Polygon layers	`POLYGONS`	[vector: polygon][list]	Layer(s) with the polygons to check against.
Points outside polygons errors	`ERRORS`	[vector: point] Default: `[Create temporary layer]`	Specification of the output layer containing points outside the polygons. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Tolerance	`TOLERANCE`	[numeric: integer] Default: 8	Numerical precision of geometric operations, given as an integer n, meaning that two vertices less than 10^{-n} apart (in map units) are considered to be merged.

Outputs

Label	Name	Type	Description
Points outside polygons errors	`ERRORS`	[vector: point]	Output point layer representing the error locations and information. The output layer contains the following fields: • `gc_layerid`: the ID of the input layer. • `gc_layername`: the name of the input layer. • `gc_partidx` • `gc_ringidx` • `gc_vertidx` • `gc_errorx`: the x coordinate of point outside the polygons. • `gc_errory`: the y coordinate of point outside the polygons. • `gc_error` • `UNIQUE_ID` field: the unique ID of the input feature that is outside the polygons.

Python code

Algorithm ID: `native:checkgeometrypointinpolygon`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Self-intersections

Detects self-intersections in line or polygon geometries, and reports them as errors.

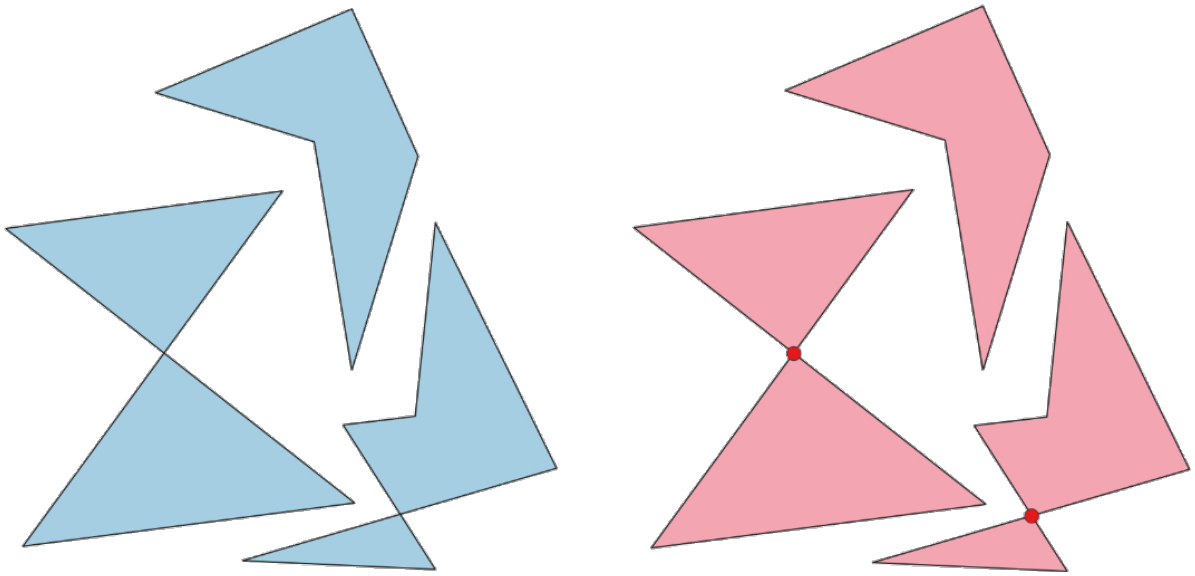


Fig. 24.8: Reporting errors on features with self-intersections.

See also

Split self-intersecting geometries

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line, polygon]	Layer with the geometries to check
Unique feature identifier	`UNIQUE_ID`	[tablefield: any]	Field storing unique values for feature identification
Self-intersecting errors	`ERRORS`	[vector: point] Default: `[Create temporary layer]`	Specification of the output layer containing the errors location. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

continues on next page

Table 24.39 – continued from previous page

Label	Name	Type	Description
Self-intersecting features Optional	`OUTPUT`	[same as input] Default: `[Skip output]`	Line or polygon layer containing self-intersected features. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Tolerance	`TOLERANCE`	[numeric: integer] Default: 8	Numerical precision of geometric operations, given as an integer n, meaning that two vertices less than 10^{-n} apart (in map units) are considered to be merged.

Outputs

Label	Name	Type	Description
Self-intersecting errors	ERRORS	[vector: point]	Output point layer representing the error locations and information. The output layer contains the following fields: • gc_layerid: the ID of the input layer. • gc_layername: the name of the input layer. • gc_partidx: the index of the feature's geometry part where the self-intersection occurs. • gc_ringidx: the index of the feature's geometry ring where the self-intersection occurs. • gc_vertidx • gc_errorx: the x coordinate of the self-intersection. • gc_errory: the y coordinate of the self-intersection. • gc_error • gc_segment_1: the index of the first segment involved in the intersection. • gc_segment_2: the index of the second segment involved in the intersection. • UNIQUE_ID field: the unique ID of the self-intersecting input feature.
Self-intersecting features	OUTPUT	[vector: line, polygon]	Output polygon or line layer with features containing the self-intersecting features. If no self-intersecting features are found, the output layer will be empty. Available fields are the same as in the ERRORS output.

Python code

Algorithm ID: native:checkgeometryselfintersection

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Small angles

Compares the angles within line or polygon geometries to a specified threshold, and reports as error any angle below that value.

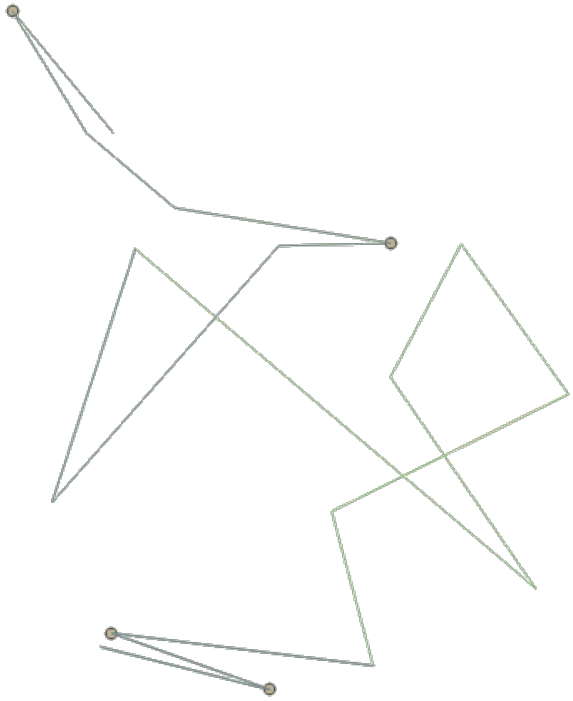


Fig. 24.9: Reporting errors on a line feature for angles lower than 15°.

See also

Delete small angles

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line, polygon]	Layer with the geometries to check.
Unique feature identifier	`UNIQUE_ID`	[tablefield: any]	Field storing unique values for feature identification.
Minimum angle	`MIN_ANGLE`	[numeric: double]	Allowed minimum angle between adjacent segments in a feature's geometry. Any angle below this threshold is reported as error.

continues on next page

Table 24.42 – continued from previous page

Label	Name	Type	Description
Small angle errors	ERRORS	[vector: point] Default: [Create temporary layer]	Specification of the output layer containing the errors location. *One of*: • Create Temporary Layer (TEMPORARY_OUTPUT) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Tolerance	TOLERANCE	[numeric: integer] Default: 8	Numerical precision of geometric operations, given as an integer n, meaning that two vertices less than 10^{-n} apart (in map units) are considered to be merged.

Outputs

Label	Name	Type	Description
Small angle errors	ERRORS	[vector: point]	Output point layer representing the error locations and information. The output layer contains the following fields: • gc_layerid: the ID of the input layer. • gc_layername: the name of the input layer. • gc_partidx: the index of the feature's geometry part containing the small angle. • gc_ringidx: the index of the feature's geometry ring containing the small angle. • gc_vertidx: the index of the vertex with the small angle in the feature's geometry ring. • gc_errorx: the x coordinate of the vertex with the small angle. • gc_errory: the y coordinate of the vertex with the small angle. • gc_error: the error angle value. • UNIQUE_ID field: the unique ID of the input feature with the small angle.

Python code

Algorithm ID: `native:checkgeometryangle`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Small segments

Calculates length of individual segments in line or polygon geometries, and reports segments shorter than a minimum length as errors.

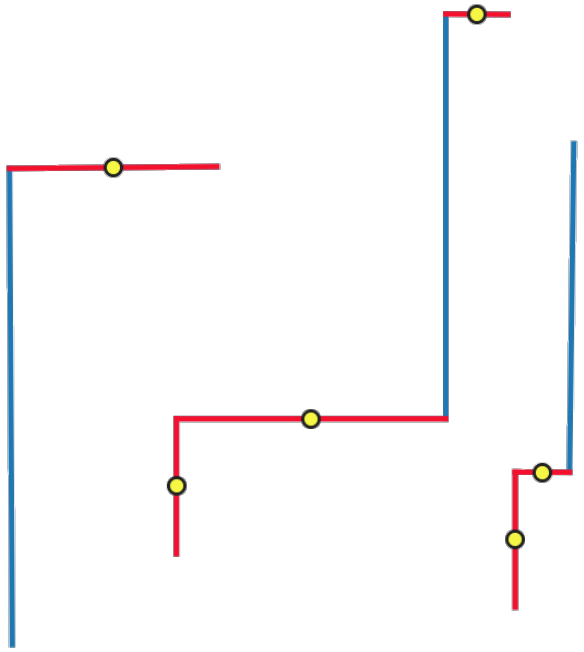


Fig. 24.10: Reporting errors (in red) on single line segments shorter than the given threshold.

See also

Delete features

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line, polygon]	Layer with the geometries to check.
Unique feature identifier	`UNIQUE_ID`	[tablefield: any]	Field storing unique values for feature identification.
Minimum segment length	`MIN_SEGMENT_LE`	[numeric: double] Default: 0.0	Minimum length of segments to be considered valid, in map units.
Short segments errors	`ERRORS`	[vector: point] Default: `[Create temporary layer]`	Specification of the output layer containing centroid of short segments. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Short segments features Optional	`OUTPUT`	[vector: same as input] Default: `[Skip output]`	Specification of the output layer for features containing short segments. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Tolerance	`TOLERANCE`	[numeric: integer] Default: 8	Numerical precision of geometric operations, given as an integer n, meaning that two vertices less than 10^{-n} apart (in map units) are considered to be merged.

Outputs

Label	Name	Type	Description
Short segments errors	ERRORS	[vector: point]	Output point layer representing the error locations and information. The output layer contains the following fields: • gc_layerid: the ID of the input layer. • gc_layername: the name of the input layer. • gc_partidx: the index of the feature's geometry part containing the small segment. • gc_ringidx: the index of the feature's geometry ring containing the small segment. • gc_vertidx: the index of the end vertex of the small segment in the feature's geometry ring. • gc_errorx: the x coordinate of the centroid of the small segment. • gc_errory: the y coordinate of the centroid of the small segment. • gc_error: the error segment length. • UNIQUE_ID field: the unique ID of the input feature with the small segment.
Short segments features	OUTPUT	[vector: same as input]	Output layer containing, for each identified short segment, the feature it belongs to. Available fields are the same as in the ERRORS output.

Python code

Algorithm ID: native:checkgeometrysegmentlength

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.4 Database

Export to PostgreSQL

Exports a vector layer to a PostgreSQL database, creating a new relation. If a relation with the same name exists, it can be removed before the new relation is created. Prior to this a connection between QGIS and the PostgreSQL database has to be created (see eg *Creating a stored Connection*).

Parameters

Label	Name	Type	Description
Layer to import	`INPUT`	[vector: any]	Vector layer to add to the database
Database (connection name)	`DATABASE`	[string]	Name of the database connection (not the database name). Existing connections will be shown in the combobox.
Schema (schema name) Optional	`SCHEMA`	[string] Default: 'public'	Name of the schema to store the data. It can be a new one or already exist.
Table to import to (leave blank to use layer name) Optional	`TABLENAME`	[string] Default: ''	Defines a table name for the imported vector file. If nothing is added, the layer name will be used.
Primary key field Optional	`PRIMARY_KEY`	[tablefield: any]	Sets the primary key field from an existing field in the vector layer. A column with **unique** values can be used as Primary key for the database.
Geometry column	`GEOMETRY_COLUMN`	[string] Default: 'geom'	Defines the name of the geometry column in the new PostgreSQL table. Geometry information for the features is stored in this column.
Encoding Optional	`ENCODING`	[string] Default: 'UTF-8'	Defines the encoding of the output layer
Overwrite	`OVERWRITE`	[boolean] Default: True	If the specified table exists, setting this option to `True` will make sure that it is deleted and a new table will be created before the features are added. If this option is `False` and the table exists, the algorithm will throw an exception ("relation already exists").
Create spatial index	`CREATEINDEX`	[boolean] Default: True	Specifies whether to create a spatial index or not
Convert field names to lowercase	`LOWERCASE_NAMES`	[boolean] Default: True	Converts the field names of the input vector layer to lowercase
Drop length constraint on character fields	`DROP_STRING_LE`	[boolean] Default: False	Should length constraints on character fields be dropped or not
Create single-part geometries instead of multi-part	`FORCE_SINGLEPA`	[boolean] Default: False	Should the features of the output layer be single-part instead of multi-part. By default the existing geometries information are preserved.

Outputs

The algorithm has no output.

Python code

Algorithm ID: `qgis:importintopostgis`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Export to SpatiaLite

Exports a vector layer to a SpatiaLite database. Prior to this a connection between QGIS and the SpatiaLite database has to be created (see eg *SpatiaLite Layers*).

Parameters

Label	Name	Type	Description
Layer to import	`INPUT`	[vector: any]	Vector layer to add to the database
File database	`DATABASE`	[vector: any]	The SQLite/SpatiaLite database file to connect to
Table to import to (leave blank to use layer name) Optional	`TABLENAME`	[string] Default: ''	Defines the table name for the imported vector file. If nothing is specified, the layer name will be used.
Primary key field Optional	`PRIMARY_KEY`	[tablefield: any]	Use a field in the input vector layer as the primary key
Geometry column	`GEOMETRY_COLUMN`	[string] Default: 'geom'	Defines the name of the geometry column in the new SpatiaLite table. Geometry information for the features is stored in this column.
Encoding Optional	`ENCODING`	[string] Default: 'UTF-8'	Defines the encoding of the output layer
Overwrite	`OVERWRITE`	[boolean] Default: True	If the specified table exists, setting this option to `True` will make sure that it is deleted and a new table will be created before the features of the layer is added. If this option is `False` and the table exists, the algorithm will throw an exception ("table already exists").
Create spatial index	`CREATEINDEX`	[boolean] Default: True	Specifies whether to create a spatial index or not
Convert field names to lowercase	`LOWERCASE_NAMES`	[boolean] Default: True	Convert the field names of the input vector layer to lowercase
Drop length constraint on character fields	`DROP_STRING_LE`	[boolean] Default: False	Should length constraints on character fields be dropped or not

continues on next page

Table 24.49 – continued from previous page

Label	Name	Type	Description
Create single-part geometries instead of multi-part	`FORCE_SINGLEPA`	[boolean] Default: False	Should the features of the output layer be single-part instead of multi-part. By default the existing geometries information are preserved.

Outputs

The algorithm has no output.

Python code

Algorithm ID: `native:importintospatialite`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Package layers

Adds layers to a GeoPackage.

If the GeoPackage exists and `Overwrite existing GeoPackage` is checked, it will be overwritten (removed and recreated). If the GeoPackage exists and `Overwrite existing GeoPackage` is not checked, the layer will be appended.

Parameters

Label	Name	Type	Description
Input layers	`LAYERS`	[vector: any] [list]	The (vector) layers to import into the GeoPackage. Raster layers are not supported. If a raster layer is added, a `QgsProcessingException` will be thrown.
Overwrite existing GeoPackage	`OVERWRITE`	[boolean] Default: False	If the specified GeoPackage exists, setting this option to `True` will make sure that it is deleted and a new one will be created before the layers are added. If set to `False`, layers will be appended.
Save layer styles into GeoPackage	`SAVE_STYLES`	[boolean] Default: True	Save the layer styles.
Save only selected features	`SELECTED_FEATURE`	[boolean] Default: False	If a layer has a selection, setting this option to `True` will result in only selected features being saved. For layers without a selection all features will be saved.
Export related layers following relations defined in the project	`EXPORT_RELATED_L`	[boolean] Default: False	If an input layer has *relations* set in the project, setting this option to `True` will result in exporting also its related layer(s). If the layer has features selected, then only their related features will be exported unless the related layer was also an input layer.
Destination GeoPackage	`OUTPUT`	[file] Default: `[Save to temporary file]`	Specify where to store the GeoPackage file. One of • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Layers within new package	`OUTPUT_LAYERS`	[string] [list]	The list of layers added to the GeoPackage.

Python code

Algorithm ID: `native:package`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

PostgreSQL execute and load SQL

Allows a SQL database query to be performed on a PostgreSQL database connected to QGIS and loads the result. The algorithm **won't** create a new layer: it is designed to run queries on the layer itself.

Example

1. Set all the values of an existing field to a fixed value. The SQL query string will be:

   ```
   UPDATE your_table SET field_to_update=20;
   ```

   In the example above, the values of the field `field_to_update` of the table `your_table` will be all set to `20`.

2. Create a new `area` column and calculate the area of each feature with the `ST_AREA` PostGIS function.

   ```
   -- Create the new column "area" on the table your_table"
   ALTER TABLE your_table ADD COLUMN area double precision;
   -- Update the "area" column and calculate the area of each feature:
   UPDATE your_table SET area=ST_AREA(geom);
   ```

See also

PostgreSQL execute SQL, *Execute SQL*, *SpatiaLite execute SQL*

Parameters

Label	Name	Type	Description
Database (connection name)	`DATABASE`	[string]	The database connection (not the database name). Existing connections will be shown in the combobox.
SQL query	`SQL`	[string]	Defines the SQL query, for example `'UPDATE my_table SET field=10'`.
Unique ID field name	`ID_FIELD`	[string] Default: id	Sets the primary key field (a column in the result table)
Geometry field name Optional	`GEOMETRY_FIELD`	[string] Default: 'geom'	Name of the geometry column (a column in the result table)

Outputs

Label	Name	Type	Description
SQL layer	`OUTPUT`	[vector: any]	The resulting vector layer to be loaded into QGIS.

Python code

Algorithm ID: `qgis:postgisexecuteandloadsql`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

PostgreSQL execute SQL

Allows a SQL database query to be performed on a PostgreSQL database connected to QGIS. The algorithm **won't** create a new layer: it is designed to run queries on the layer itself.

Example

1. Set all the values of an existing field to a fixed value. The SQL query string will be:

   ```
   UPDATE your_table SET field_to_update=20;
   ```

   In the example above, the values of the field `field_to_update` of the table `your_table` will be all set to `20`.

2. Create a new `area` column and calculate the area of each feature with the `ST_AREA` PostGIS function.

   ```
   -- Create the new column "area" on the table your_table"
   ALTER TABLE your_table ADD COLUMN area double precision;
   -- Update the "area" column and calculate the area of each feature:
   UPDATE your_table SET area=ST_AREA(geom);
   ```

See also

PostgreSQL execute and load SQL, *Execute SQL*, *SpatiaLite execute SQL*

Parameters

Label	Name	Type	Description
Database (connection name)	`DATABASE`	[string]	The database connection (not the database name). Existing connections will be shown in the combobox.
SQL query	`SQL`	[string]	Defines the SQL query, for example `'UPDATE my_table SET field=10'`.

Outputs

No output is created. The SQL query is executed in place.

Python code

Algorithm ID: `native:postgisexecutesql`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

SpatiaLite execute SQL

Allows a SQL database query to be performed on a SpatiaLite database. The database is determined by an input layer or file. The algorithm **won't** create a new layer: it is designed to run queries on the layer itself.

See also

PostgreSQL execute SQL, *Execute SQL*

For some SQL query examples see *PostgreSQL Query Examples*.

Parameters

Label	Name	Type	Description
Database layer (or file)	`DATABASE`	[file]	The SQLite/SpatiaLite database file to connect to
SQL query	`SQL`	[string] Default: ''	Defines the SQL query, for example `'UPDATE my_table SET field=10'`.

Outputs

No output is created. The SQL query is executed in place.

Python code

Algorithm ID: `native:spatialiteexecutesql`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

SpatiaLite execute SQL (registered DB)

Allows a SQL database query to be performed on a SpatiaLite database connected to QGIS. The algorithm **won't** create a new layer: it is designed to run queries on the layer itself.

See also

PostgreSQL execute SQL, *Execute SQL*

For some SQL query examples see *PostgreSQL Query Examples*.

Parameters

Label	Name	Type	Description
Database	`DATABASE`	[enumeration] Default: not set	Select a SQLite/SpatiaLite database connected to the current session
SQL query	`SQL`	[string] Default: ''	Defines the SQL query, for example `'UPDATE my_table SET field=10'`.

Outputs

No output is created. The SQL query is executed in place.

Python code

Algorithm ID: `native:spatialiteexecutesqlregistered`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.5 File tools

Download file via HTTP(S)

Downloads a URL to the file system with an HTTP(S) GET or POST request.

Parameters

Basic parameters

Label	Name	Type	Description
URL	`URL`	[string]	The URL of the file to download.
File destination	`OUTPUT`	[string] Default: `[Save to temporary file]`	Specification of the file destination. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Method	`METHOD`	[enumeration] Default: 0	The HTTP method to use for the request. Options are: • 0 — GET • 1 — POST
Data Optional	`DATA`	[string]	The data to add in the body if the request is a POST.

Outputs

Label	Name	Type	Description
File destination	`OUTPUT`	[string]	The location of the downloaded file

Python code

Algorithm ID: `qgis:filedownloader`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

HTTP(S) POST/GET request

Performs a HTTP(S) POST/GET request and returns the HTTP status code and the reply data. If an error occurs then the error code and the message will be returned. Optionally, the result can be written to a file on the disk. By default the algorithm will warn on errors. Optionally, the algorithm can be set to treat HTTP errors as failures.

Parameters

Label	Name	Type	Description
URL or file path	`URL`	[string]	The URL or path of the file to open.
Method	`METHOD`	[enumeration] Default: 0	The HTTP method to use for the request. Options are: • 0 — GET • 1 — POST
POST Data Optional	`DATA`	[string]	The data to add in the body if the request is a POST.
Authentication Optional	`AUTH_CONFIG`	[authconfig] Default: No authentication	An authentication configuration to pass
Consider HTTP errors as failures	`FAIL_ON_ERROR`	[boolean] Default: False	If set, the algorithm will fail on encountering an HTTP error.
File destination Optional	`OUTPUT`	[string] Default: `[Skip Output]`	The result can be written to a file instead of being returned as a string. Specification of the file destination. *One of*: • Skip Output • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
File destination	`OUTPUT`	[file]	The downloaded file with the returned result
HTTP Status	`STATUS_CODE`	[string]	The HTTP Status
Network error code	`ERROR_CODE`	[string]	The error code when it fails (like as well on 404 HTTP Status etc.)
Network error message	`ERROR_MESSAGE`	[string]	A string containing the error message in case of failure
Result data	`RESULT_DATA`	[string]	A string containing the result data in case of success

Python code

Algorithm ID: `native:httprequest`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Open file or URL

Opens files in their default associated application, or URLs in the user's default web browser.

Parameters

Label	Name	Type	Description
URL or file path	`URL`	[string]	The URL or path of the file to open.

Outputs

The algorithm has no output.

Python code

Algorithm ID: `native:openurl`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.6 Fix Geometry

Delete duplicated vertices

Deletes duplicated vertices from the input geometries, based on errors reported by the *Duplicated vertices* algorithm.

See also

Duplicated vertices, *Remove duplicate vertices*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line, polygon]	Layer with the geometries to fix, same as input in the check algorithm.
Error layer	`ERRORS`	[vector: point]	Point layer with the errors location, generated by the check algorithm.
Field of original feature unique identifier	`UNIQUE_ID`	[tablefield: any]	Field storing unique values for feature identification in input layer, as set in the check algorithm.
Field of part index	`PART_IDX`	[tablefield: integer] Default: `gc_partidx`	Field storing the erroneous feature's geometry part number.
Field of ring index	`RING_IDX`	[tablefield: integer] Default: `gc_ringidx`	Field storing the erroneous feature's geometry ring number.
Field of vertex index	`VERTEX_IDX`	[tablefield: integer] Default: `gc_vertidx`	Field storing the erroneous feature's geometry vertex number.
Fixed duplicate vertices layer	`OUTPUT`	[vector: same as input] Default: `[Create temporary layer]`	Specification of the output layer containing fixed features. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Report layer from fixing duplicate vertices	`REPORT`	[vector: point] Default: `[Create temporary layer]`	Specification of the output layer containing the fixes location. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Tolerance	`TOLERANCE`	[numeric: integer] Default: 8	Numerical precision of geometric operations, given as an integer n, meaning that two vertices less than 10^{-n} apart (in map units) are considered to be merged.

Outputs

Label	Name	Type	Description
Report layer from fixing duplicate vertices	`REPORT`	[vector: point]	Output point layer representing the error locations and fix applied. The output layer contains the following fields: • `gc_layerid`: the ID of the input layer. • `gc_layername`: the name of the input layer. • `gc_partidx`: the index of the feature's geometry part containing the duplicate vertex. • `gc_ringidx`: the index of the feature's geometry ring containing the duplicate vertex. • `gc_vertidx`: the index of the duplicate vertex in the feature's geometry ring. • `gc_errorx`: the x coordinate of the duplicate vertex. • `gc_errory`: the y coordinate of the duplicate vertex. • `gc_error` • `report`: a text field describing the fix applied. • `error_fixed`: a boolean field indicating whether the error was fixed. • `UNIQUE_ID` field: the unique ID of the input feature that has duplicate vertices.
Fixed duplicate vertices layer	`OUTPUT`	[vector: same as input]	Output layer with the geometry fix applied to the input features. The layer contains the same fields as in the input layer.

Python code

Algorithm ID: `native:fixgeometryduplicatenodes`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Delete features

Deletes error features based on an error layer from some check geometry algorithms, such as:

- *Features inside polygon*
- *Degenerate polygons*
- *Small segments*
- *Duplicated geometries*

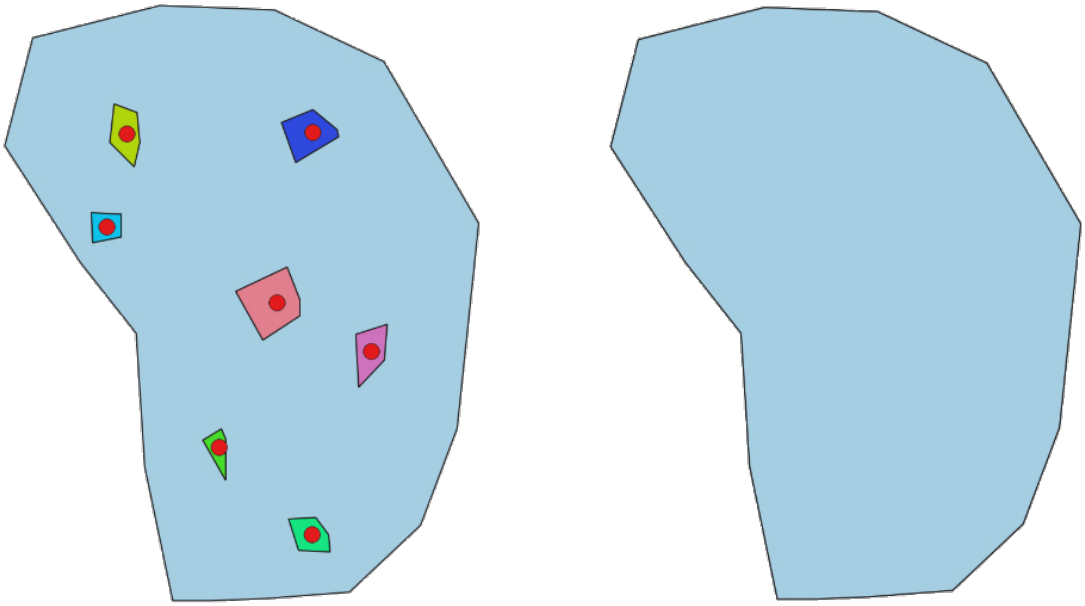


Fig. 24.11: Before and after deleting features inside a polygon.

See also

Delete duplicate geometries, *Drop geometries*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Layer with the geometries to fix, same as input in the check algorithm.
Error layer	`ERRORS`	[vector: point]	Point layer with the errors location, generated by the check algorithm.
Field of original feature unique identifier	`UNIQUE_ID`	[tablefield: any]	Field storing unique values for feature identification in input layer, as set in the check algorithm.
Cleaned layer	`OUTPUT`	[vector: same as input] Default: `[Create temporary layer]`	Specification of the output layer containing fixed features. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Report layer from deleting features	`REPORT`	[vector: point] Default: `[Create temporary layer]`	Specification of the output layer containing the fixes location. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Report layer from deleting features	`REPORT`	[vector: point]	Output point layer representing the error locations and fix applied. The output layer contains the following fields: • `gc_layerid`: the ID of the input layer. • `gc_layername`: the name of the input layer. • `gc_partidx` • `gc_ringidx` • `gc_vertidx` • `gc_errorx`: the x coordinate of the centroid of the erroneous geometry. • `gc_errory`: the y coordinate of the centroid of the erroneous geometry. • `gc_error`: the index of the feature where the error belongs. • `report`: a text field describing the fix applied. • `error_fixed`: a boolean field indicating whether the error was fixed. • `UNIQUE_ID` field: the unique ID of the input feature with the error.
Cleaned layer	`OUTPUT`	[vector: same as input]	Output layer with features removed based on detected errors. The layer contains the same fields as in the input layer.

Python code

Algorithm ID: `native:fixgeometrydeletefeatures`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Delete overlaps

Deletes overlapping areas based on an error layer from the *Overlaps* algorithm.

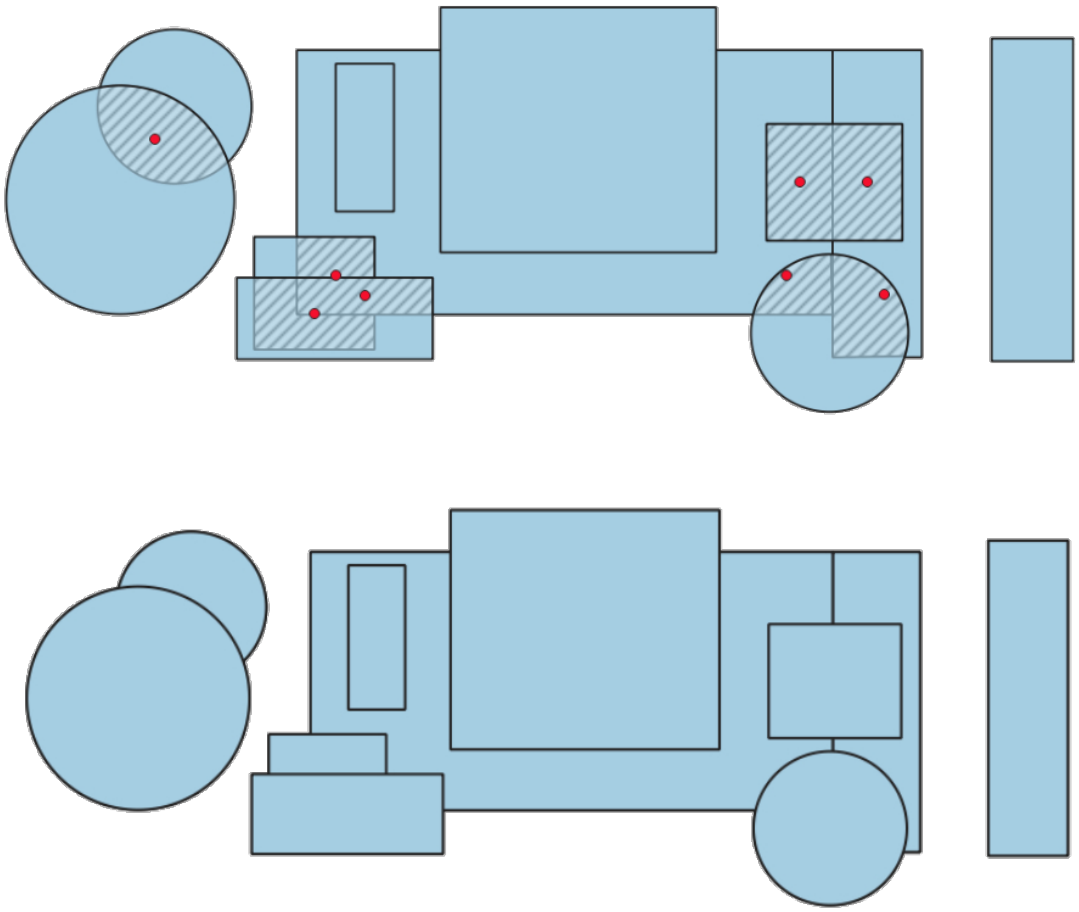


Fig. 24.12: Before and after deleting overlapping areas.

See also

Overlaps

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: polygon]	Layer with the geometries to fix, same as input in the check algorithm.
Error layer	`ERRORS`	[vector: point]	Point layer with the errors location, generated by the check algorithm.

continues on next page

Table 24.55 – continued from previous page

Label	Name	Type	Description
Field of original feature unique identifier	`UNIQUE_ID`	[tablefield: any]	Field storing unique values for feature identification in input layer, as set in the check algorithm.
Field of overlap feature unique identifier	`OVERLAP_FEATURE_UN`	[tablefield: any]	Field storing unique values for the overlapping feature identification in input layer, as set in the check algorithm.
Field of error value	`ERROR_VALUE_ID`	[tablefield: any]	Field storing the error value for each feature, as set in the check algorithm.
No-overlap layer	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specification of the output layer containing fixed features. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Report layer from fixing overlaps	`REPORT`	[vector: point] Default: `[Create temporary layer]`	Specification of the output layer containing the fixes location. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Tolerance	`TOLERANCE`	[numeric: integer] Default: 8	Numerical precision of geometric operations, given as an integer n, meaning that two vertices less than 10^{-n} apart (in map units) are considered to be merged.

Outputs

Label	Name	Type	Description
Report layer from fixing overlaps	`REPORT`	[vector: point]	Output point layer representing the error locations and fix applied. The output layer contains the following fields: • `gc_layerid`: the ID of the input layer. • `gc_layername`: the name of the input layer. • `gc_errorx`: the x coordinate of the centroid of the overlapping area. • `gc_errory`: the y coordinate of the centroid of the overlapping area. • `gc_error`: the area of the overlapping geometry. • `UNIQUE_ID` field: the unique ID of the unique ID of an overlapped input feature. • `gc_overlap_feature_{unique_id}`: the `UNIQUE_ID` field value for the other overlapping feature. • `report`: a text field describing the fix applied or justifying the failure. Possible values are: – Remove overlapping area from neighboring polygon with shortest shared edge – Error is obsolete • `error_fixed`: a boolean field indicating whether the error was fixed.
No-overlap layer	`OUTPUT`	[vector: polygon]	Output layer with input features edited. Overlapping areas reported as errors are removed. The layer contains the same fields as in the input layer.

Python code

Algorithm ID: `native:fixgeometryoverlap`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Delete small angles

Deletes vertices based on an error layer from the *Small angles* algorithm. When deletion of a vertex results in a duplicate vertex (when a spike vertex is deleted), the duplicate vertex is deleted to keep a single vertex and preserve topology.

Attention

This algorithm removes the vertex at the reported small angles, generating new segments that may form a new small angle.

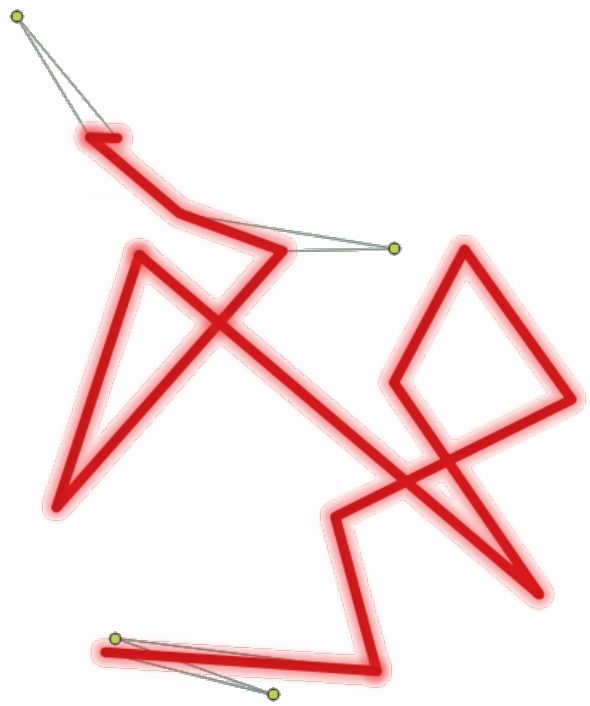


Fig. 24.13: Fixing reported errors on a line feature for angles lower than 15°.

See also

Small angles

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line, polygon]	Layer with the geometries to fix, same as input in the check algorithm
Error layer	`ERRORS`	[vector: point]	Point layer with the errors location, generated by the check algorithm

continues on next page

Table 24.58 – continued from previous page

Label	Name	Type	Description
Field of original feature unique identifier	`UNIQUE_ID`	[tablefield: any]	Field storing unique values for feature identification in input layer, as set in the check algorithm
Field of part index	`PART_IDX`	[tablefield: integer] Default: `gc_partidx`	Field storing the erroneous feature's geometry part number.
Field of ring index	`RING_IDX`	[tablefield: integer] Default: `gc_ringidx`	Field storing the erroneous feature's geometry ring number.
Field of vertex index	`VERTEX_IDX`	[tablefield: integer] Default: `gc_vertidx`	Field storing the erroneous feature's geometry vertex number.
Small angle fixed layer	`OUTPUT`	[vector: same as input] Default: `[Create temporary layer]`	Specification of the output layer containing fixed features. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Report layer from fixing small angles	`REPORT`	[vector: point] Default: `[Create temporary layer]`	Specification of the output layer containing the fixes location. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Tolerance	`TOLERANCE`	[numeric: integer] Default: 8	Numerical precision of geometric operations, given as an integer n, meaning that two vertices less than 10^{-n} apart (in map units) are considered to be merged.

Outputs

Label	Name	Type	Description
Report layer from fixing small angles	`REPORT`	[vector: point]	Output point layer representing the error locations and fix applied. The output layer contains the following fields: • `gc_layerid`: the ID of the input layer. • `gc_layername`: the name of the input layer. • `gc_partidx`: the index of the feature's geometry part containing the small angle. • `gc_ringidx`: the index of the feature's geometry ring containing the small angle. • `gc_vertidx`: the index of the vertex with the small angle in the feature's geometry ring. • `gc_errorx`: the x coordinate of the vertex with the small angle. • `gc_errory`: the y coordinate of the vertex with the small angle. • `gc_error`: the error angle value. • `UNIQUE_ID` field: the unique ID of the input feature with the small angle. • `report`: a text field describing the fix applied or justifying the failure. • `error_fixed`: a boolean field indicating whether the error was fixed.
Small angle fixed layer	`OUTPUT`	[same as input]	Output layer with the geometry fix applied to the input features The layer contains the same fields as in the input layer.

Python code

Algorithm ID: `native:fixgeometryangle`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Fill holes

Deletes holes in polygon geometries based on an error layer from the *Holes* algorithm.

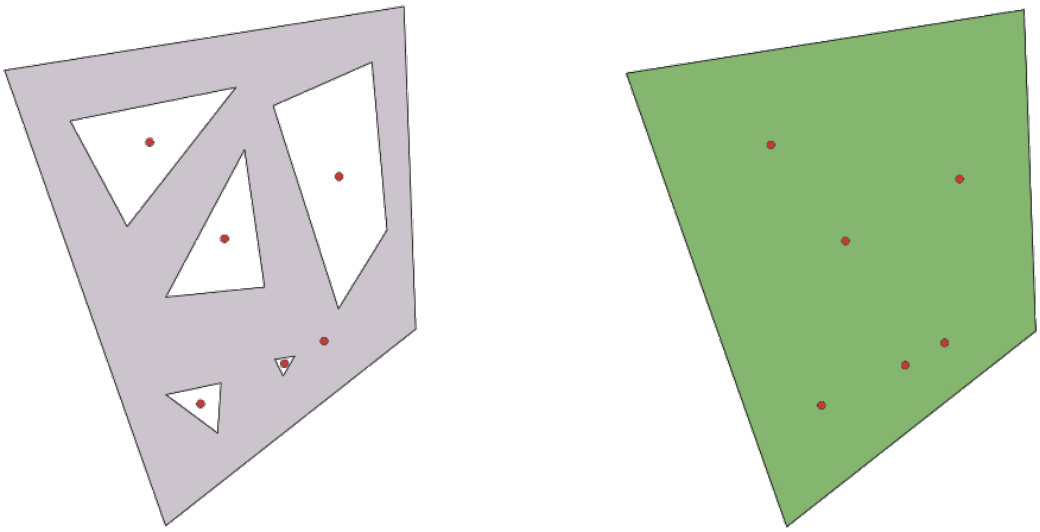


Fig. 24.14: Before and after fixing holes in a polygon feature.

See also

Holes, *Delete holes*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line, polygon]	Layer with the geometries to fix, same as input in the check algorithm
Error layer	`ERRORS`	[vector: point]	Point layer with the errors location, generated by the check algorithm
Field of original feature unique identifier	`UNIQUE_ID`	[tablefield: any]	Field storing unique values for feature identification in input layer, as set in the check algorithm
Field of part index	`PART_IDX`	[tablefield: integer] Default: `gc_partidx`	Field storing the erroneous feature's geometry part number.
Field of ring index	`RING_IDX`	[tablefield: integer] Default: `gc_ringidx`	Field storing the erroneous feature's geometry ring number.

continues on next page

Table 24.61 – continued from previous page

Label	Name	Type	Description
Field of vertex index	`VERTEX_IDX`	[tablefield: integer] Default: `gc_vertidx`	Field storing the erroneous feature's geometry vertex number.
Holes-filled layer	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specification of the output layer containing fixed features. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Report layer from fixing holes	`REPORT`	[vector: point] Default: `[Create temporary layer]`	Specification of the output layer containing the fixes location. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Tolerance	`TOLERANCE`	[numeric: integer] Default: 8	Numerical precision of geometric operations, given as an integer n, meaning that two vertices less than 10^{-n} apart (in map units) are considered to be merged.

Outputs

Label	Name	Type	Description
Report layer from fixing holes	`REPORT`	[vector: point]	Output point layer representing the error locations and fix applied. The output layer contains the following fields: • `gc_layerid`: the ID of the input layer. • `gc_layername`: the name of the input layer. • `gc_partidx`: the index of the feature's geometry part containing the hole. • `gc_ringidx`: the index of the feature's geometry ring containing the hole. • `gc_vertidx` • `gc_errorx`: the x coordinate of the centroid of the hole. • `gc_errory`: the y coordinate of the centroid of the hole. • `gc_error` • `UNIQUE_ID` field: the unique ID of the input feature that has a hole. • `report`: a text field describing the fix applied or justifying the failure. • `error_fixed`: a boolean field indicating whether the error was fixed.
Holes-filled layer	`OUTPUT`	[vector: polygon]	Output layer of polygons without holes. The layer contains the same fields as in the input layer.

Python code

Algorithm ID: `native:fixgeometryhole`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Split self-intersecting geometries

Splits self-intersecting geometries based on an error layer from the *Self-intersections* algorithm.

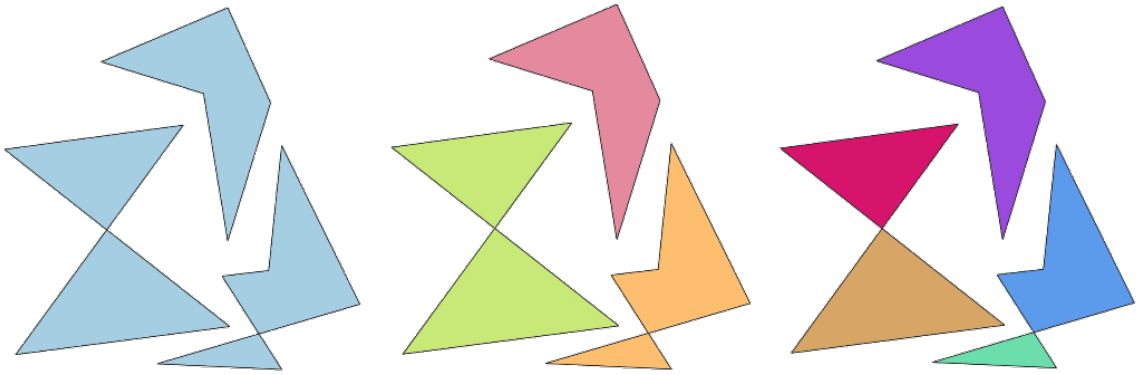


Fig. 24.15: Comparison between input layer, multi-object fix, and single-object fix geometries.

See also

Self-intersections

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line, polygon]	Layer with the geometries to fix, same as input in the check algorithm
Error layer	`ERRORS`	[vector: point]	Point layer with the errors location, generated by the check algorithm
Method	`METHOD`	[enumeration]	Method to apply to the self-intersecting geometries: • 0 – `Split feature into a multi-object feature`: splits the self-intersecting geometry into multiple parts stored in a single multipart feature. • 1 – `Split feature into multiple single-object features`: splits the self-intersecting geometry into multiple parts, each part stored in a separate feature.

continues on next page

Table 24.64 – continued from previous page

Label	Name	Type	Description
Field of original feature unique identifier	`UNIQUE_ID`	[tablefield: any]	Field storing unique values for feature identification in input layer, as set in the check algorithm
Field of part index	`PART_IDX`	[tablefield: integer] Default: `gc_partidx`	Field storing the erroneous feature's geometry part number.
Field of ring index	`RING_IDX`	[tablefield: integer] Default: `gc_ringidx`	Field storing the erroneous feature's geometry ring number.
Field of vertex index	`VERTEX_IDX`	[tablefield: integer] Default: `gc_vertidx`	Field storing the erroneous feature's geometry vertex number.
Field of segment 1	`SEGMENT_1`	[tablefield: integer] Default: `gc_segment_1`	Field storing the number of the first segment involved in the erroneous intersection.
Field of segment 2	`SEGMENT_2`	[tablefield: integer] Default: `gc_segment_2`	Field storing the number of the second segment involved in the erroneous intersection.
Self-intersections fixed layer	`OUTPUT`	[vector: same as input] Default: `[Create temporary layer]`	Specification of the output layer containing fixed features. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Report layer from fixing self-intersections	`REPORT`	[vector: point] Default: `[Create temporary layer]`	Specification of the output layer containing the fixes location. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Tolerance	`TOLERANCE`	[numeric: integer] Default: 8	Numerical precision of geometric operations, given as an integer n, meaning that two vertices less than 10^{-n} apart (in map units) are considered to be merged.

Outputs

Label	Name	Type	Description
Report layer from fixing self-intersections	`REPORT`	[vector: point]	Output point layer representing the error locations and fix applied. The output layer contains the following fields: • `gc_layerid`: the ID of the input layer. • `gc_layername`: the name of the input layer. • `gc_partidx`: the index of the feature's geometry part where the self-intersection occurs. • `gc_ringidx`: the index of the feature's geometry ring where the self-intersection occurs. • `gc_vertidx` • `gc_errorx`: the x coordinate of the self-intersection. • `gc_errory`: the y coordinate of the self-intersection. • `gc_error` • `gc_segment_1`: the index of the first segment involved in the intersection. • `gc_segment_2`: the index of the second segment involved in the intersection. • `UNIQUE_ID` field: the unique ID of the self-intersecting input feature. • `report`: a text field describing the fix applied. • `error_fixed`: a boolean field indicating whether the error was fixed.
Self-intersections fixed layer	`OUTPUT`	[vector: same as input]	Output layer with the geometry fix applied to the input features. The layer contains the same fields as in the input layer.

Python code

Algorithm ID: `native:fixgeometryselfintersection`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.7 GPS

Convert GPS data

Uses the GPSBabel tool to convert a GPS data file from a range of formats to the GPX standard format.

Parameters

Label	Name	Type	Description
Input file	`INPUT`	[file]	File containing the data to convert
Format	`FORMAT`	[enumeration]	Format of the file to convert, from this list.
Feature type	`FEATURE_TYPE`	[enumeration] Default: 0	The type of data to convert • 0 — Waypoints • 1 — Routes • 2 — Tracks
Output	`OUTPUT`	[vector: any] Default: `[Save to temporary file]`	Specification of the output GPX file. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Output layer	`OUTPUT_LAYER`	[vector: any]	Output layer with data in GPX standard format

Python code

Algorithm ID: `native:convertgpsdata`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Convert GPX feature type

Uses the GPSBabel tool to convert GPX features from one type to another (e.g. converting all waypoint features to a route).

Parameters

Label	Name	Type	Description
Input file	`INPUT`	[file]	File containing the data to convert
Conversion	`CONVERSION`	[enumeration] Default: 0	The type of conversion to apply • 0 — Waypoints from a route • 1 — Waypoints from a track • 2 — Routes from waypoints • 3 — Tracks from waypoints
Output	`OUTPUT`	[vector: point or line] Default: `[Save to temporary file]`	Specification of the output file. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Output	`OUTPUT`	[vector: any]	Output layer with converted GPX features

Python code

Algorithm ID: `native:convertgpxfeaturetype`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Download GPS data from device

Uses the GPSBabel tool to download data from a GPS device into the GPX standard format.

Parameters

Label	Name	Type	Description
Device	`DEVICE`	[enumeration] Default:*Garmin serial*	The GPS device used to create the data. Must be declared in *GPS Settings* dialog.
Port	`PORT`	[enumeration]	The port the device is connected to. Available ports depend on the OS.
Feature type	`FEATURE_TYPE`	[enumeration] Default: 0	The type of data to convert • 0 — Waypoints • 1 — Routes • 2 — Tracks
Output	`OUTPUT`	[vector: any] Default: `[Save to temporary file]`	Specification of the output file. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Output layer	`OUTPUT_LAYER`	[vector: any]	Output layer with data in GPX standard format

Python code

Algorithm ID: `native:downloadgpsdata`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Upload GPS data to device

Uses the GPSBabel tool to upload data to a GPS device from the GPX standard format.

Parameters

Label	Name	Type	Description
Input file	`INPUT`	[file]	`.GPX` file containing the data to upload
Device	`DEVICE`	[enumeration] Default:*Garmin serial*	The GPS device you would like to upload the data to. Must be declared in *GPS Settings* dialog.
Port	`PORT`	[enumeration]	The port the device is connected to. Available ports depend on the OS.
Feature type	`FEATURE_TYPE`	[enumeration] Default: 0	The type of data to upload • 0 — Waypoints • 1 — Routes • 2 — Tracks

Outputs

No output is provided. If successful, data are loaded to the device.

Python code

Algorithm ID: `native:uploadgpsdata`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.8 Interpolation

Heatmap (kernel density estimation)

Creates a density (heatmap) raster of an input point vector layer using kernel density estimation.

The density is calculated based on the number of points in a location, with larger numbers of clustered points resulting in larger values. Heatmaps allow easy identification of *hotspots* and clustering of points.

Parameters

Label	Name	Type	Description
Point layer	`INPUT`	[vector: point]	Point vector layer to use for the heatmap
Radius	`RADIUS`	[numeric: double] Default: 100.0	Heatmap search radius (or kernel bandwidth) in map units. The radius specifies the distance around a point at which the influence of the point will be felt. Larger values result in greater smoothing, but smaller values may show finer details and variation in point density.
Output raster size	`PIXEL_SIZE`	[numeric: double] Default: 0.1	Pixel size of the output raster layer in layer units. In the GUI, the size can be specified by the number of rows (`Number of rows`) / columns (`Number of columns`) **or** the pixel size(`Pixel Size X` / `Pixel Size Y`). Increasing the number of rows or columns will decrease the cell size and increase the file size of the output raster. The values in `Rows`, `Columns`, `Pixel Size X` and `Pixel Size Y` will be updated simultaneously - doubling the number of rows will double the number of columns, and the cell size will be halved. The extent of the output raster will remain the same (approximately).
Radius from field Optional	`RADIUS_FIELD`	[tablefield: numeric]	Sets the search radius for each feature from an attribute field in the input layer.
Weight from field Optional	`WEIGHT_FIELD`	[tablefield: numeric]	Allows input features to be weighted by an attribute field. This can be used to increase the influence certain features have on the resultant heatmap.
Kernel shape	`KERNEL`	[enumeration] Default: *0*	Controls the rate at which the influence of a point decreases as the distance from the point increases. Different kernels decay at different rates, so a triweight kernel gives features greater weight for distances closer to the point then the Epanechnikov kernel does. Consequently, triweight results in "sharper" hotspots and Epanechnikov results in "smoother" hotspots. There are many shapes available (please see the Wikipedia page for further information): • 0 — Quartic • 1 — Triangular • 2 — Uniform • 3 — Triweight • 4 — Epanechnikov

continues on next page

Table 24.74 – continued from previous page

Label	Name	Type	Description
Decay ratio (Triangular kernels only) Optional	`DECAY`	[numeric: double] Default: *0.0*	Can be used with Triangular kernels to further control how heat from a feature decreases with distance from the feature. • A value of 0 (=minimum) indicates that the heat will be concentrated in the center of the given radius and completely extinguished at the edge. • A value of 0.5 indicates that pixels at the edge of the radius will be given half the heat as pixels at the center of the search radius. • A value of 1 means the heat is spread evenly over the whole search radius circle. (This is equivalent to the 'Uniform' kernel.) • A value greater than 1 indicates that the heat is higher towards the edge of the search radius than at the center.
Output value scaling	`OUTPUT_VALUE`	[enumeration] Default: *0*	Allow to change the values of the output heatmap raster. One of: • 0 — Raw • 1 — Scaled
Heatmap	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specify the output raster layer with kernel density values. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Heatmap	`OUTPUT`	[raster]	Raster layer with kernel density values

Example: Creating a Heatmap

For the following example, we will use the `airports` vector point layer from the QGIS sample dataset (see *Downloading sample data*). Another excellent QGIS tutorial on making heatmaps can be found at http://qgistutorials.com.

In Fig. 24.16, the airports of Alaska are shown.

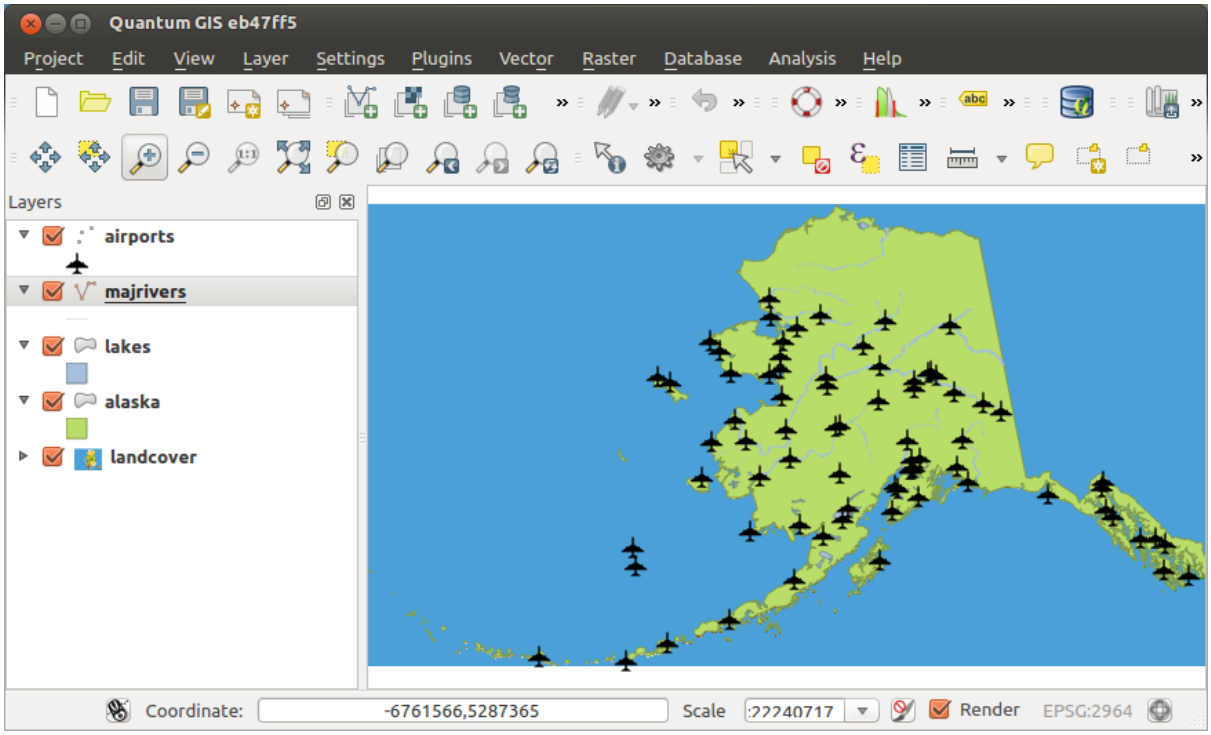


Fig. 24.16: Airports of Alaska

1. Open the *Heatmap (Kernel Density Estimation)* algorithm from the QGIS *Interpolation* group
2. In the *Point layer* field, select `airports` from the list of point layers loaded in the current project.
3. Change the *Radius* to `1000000` meters.
4. Change the *Pixel size X* to `1000`. The *Pixel size Y*, *Rows* and *Columns* will be automatically updated.
5. Click on *Run* to create and load the airports heatmap (see Fig. 24.18).

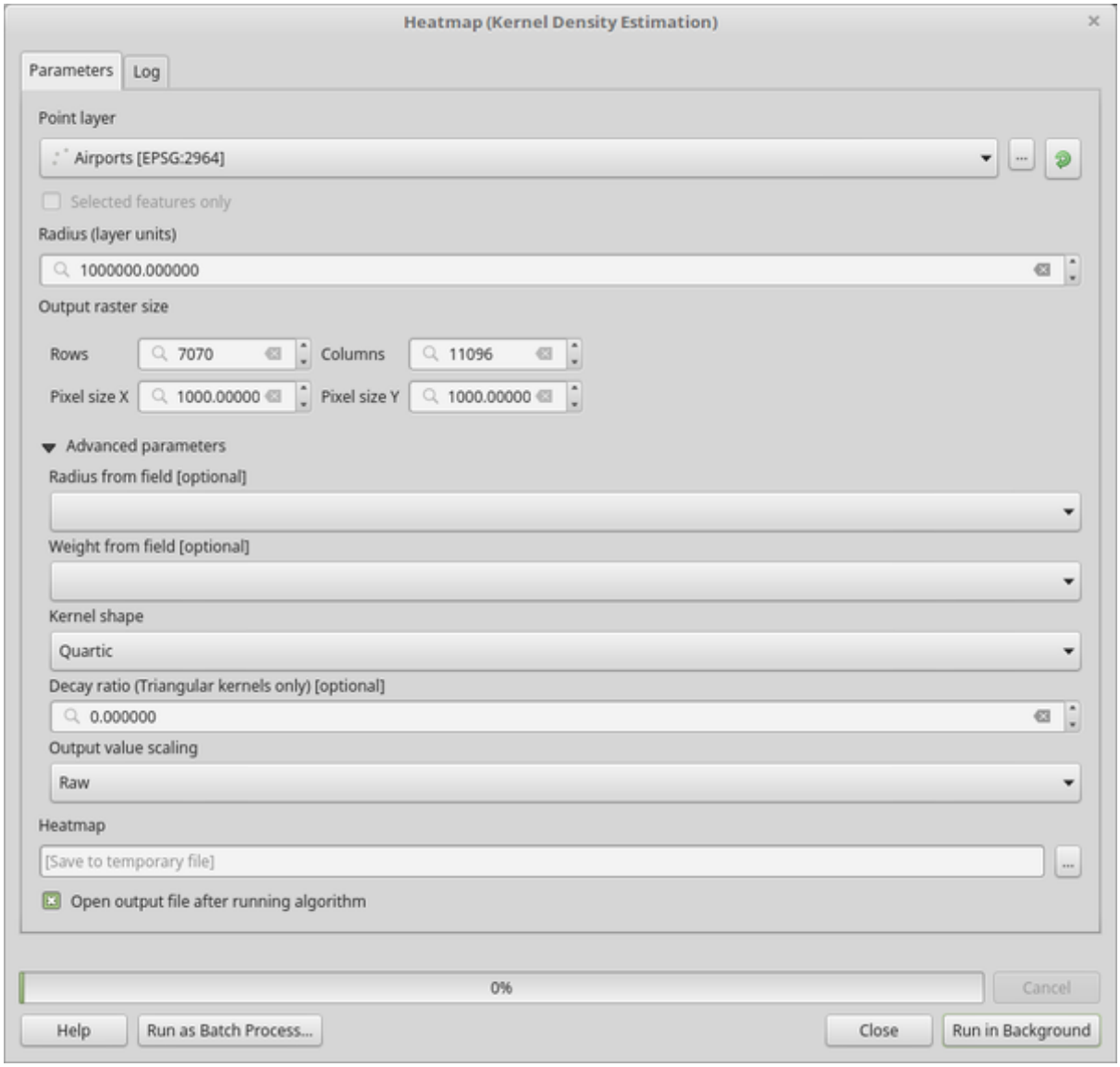


Fig. 24.17: The Heatmap Dialog

QGIS will generate the heatmap and add it to your map window. By default, the heatmap is shaded in greyscale, with lighter areas showing higher concentrations of airports. The heatmap can now be styled in QGIS to improve its appearance.

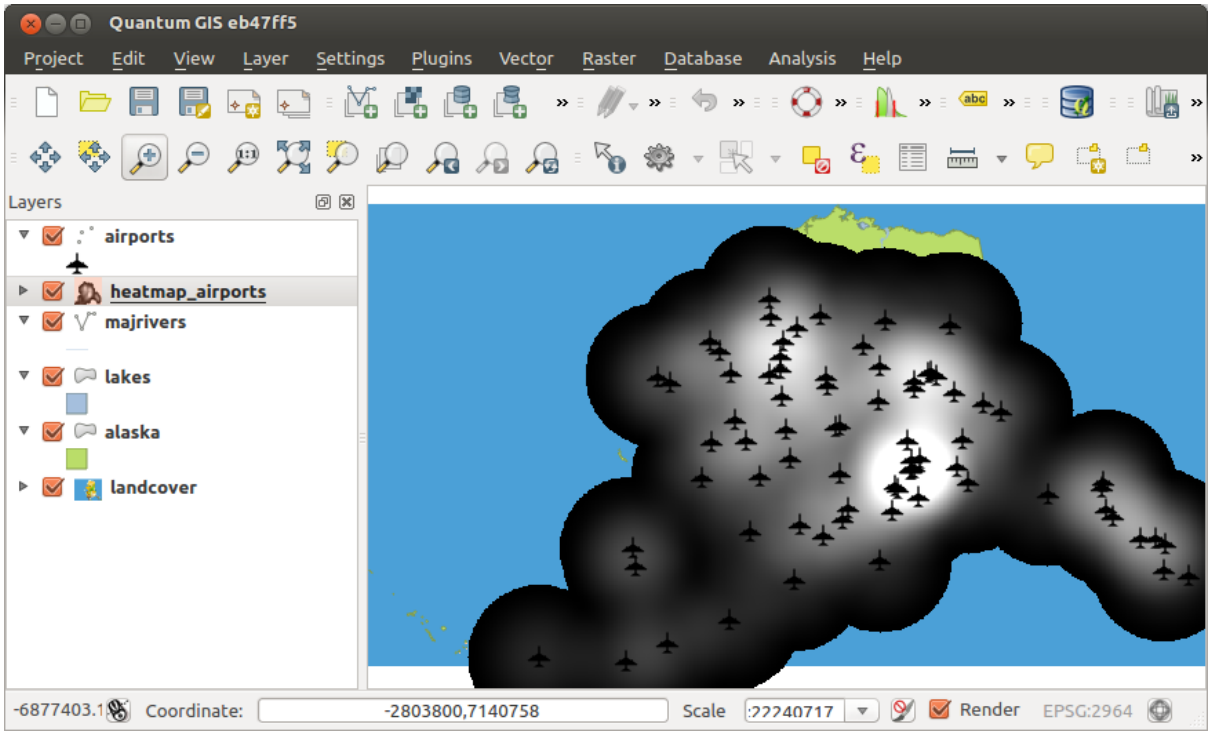


Fig. 24.18: The heatmap after loading looks like a grey surface

1. Open the properties dialog of the `heatmap_airports` layer (select the layer `heatmap_airports`, open the context menu with the right mouse button and select *Properties*).
2. Select the *Symbology* tab.
3. Change the *Render type*  to 'Singleband pseudocolor'.
4. Select a suitable *Color ramp*  , for instance `YlOrRd`.
5. Click the *Classify* button.
6. Press *OK* to update the layer.

The final result is shown in Fig. 24.19.

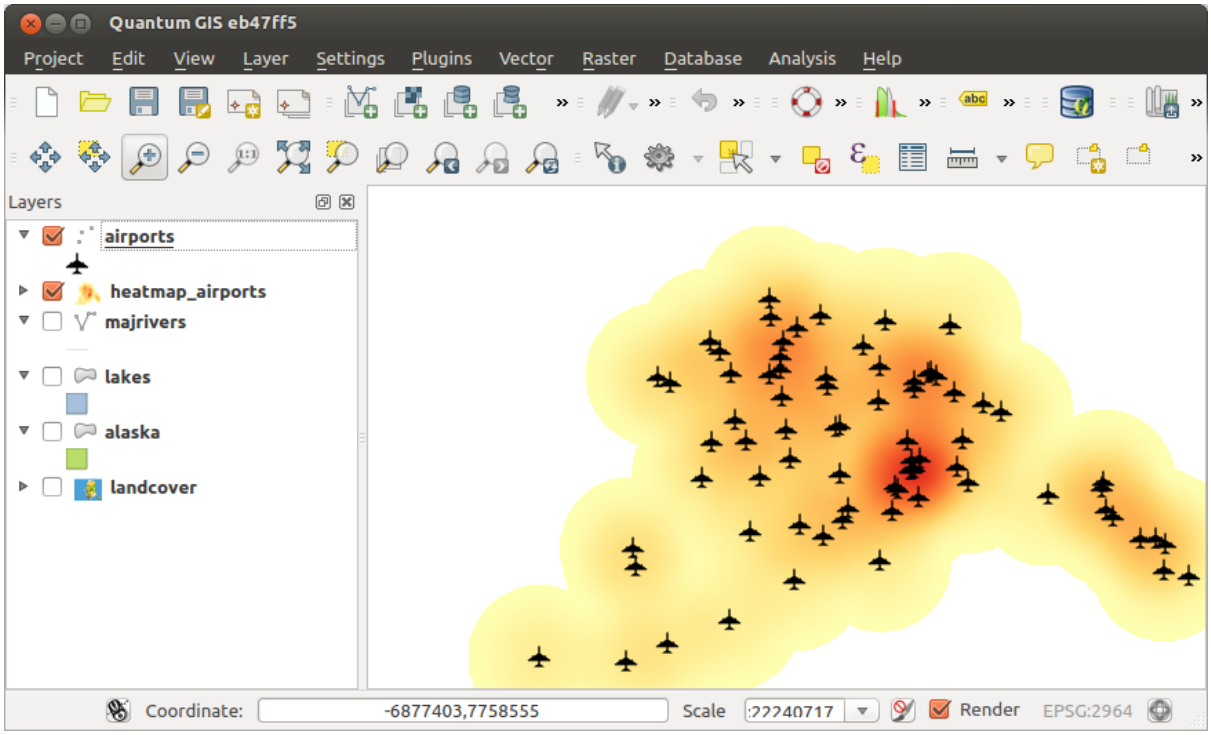


Fig. 24.19: Styled heatmap of airports of Alaska

Python code

Algorithm ID: `qgis:heatmapkerneldensityestimation`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

IDW Interpolation

Generates an Inverse Distance Weighted (IDW) interpolation of a point vector layer.

Sample points are weighted during interpolation such that the influence of one point relative to another declines with distance from the unknown point you want to create.

The IDW interpolation method also has some disadvantages: the quality of the interpolation result can decrease, if the distribution of sample data points is uneven.

Furthermore, maximum and minimum values in the interpolated surface can only occur at sample data points.

Parameters

Label	Name	Type	Description
Input layer(s)	`INTERPOLATION_DATA`	[string]	Vector layer(s) and field(s) to use for the interpolation, coded in a string (see the `ParameterInterpolationData` class in InterpolationWidgets for more details). The following GUI elements are provided to compose the interpolation data string: • **Vector layer** [vector: any] • **Interpolation attribute** [tablefield: numeric]: Attribute to use in the interpolation • **Use Z-coordinate for interpolation** [boolean]: Uses the layer's stored Z values (Default: False) For each of the added layer-field combinations, a type can be chosen: • *Points* • *Structured lines* • *Break lines* In the string, the layer-field elements are separated by `'::|::'`. The sub-elements of the layer-field elements are separated by `'::~::'`.
Distance coefficient P	`DISTANCE_COEFFICI`	[numeric: double] Default: 2.0	Sets the distance coefficient for the interpolation. Minimum: 0.0, maximum: 100.0.
Extent (xmin, xmax, ymin, ymax)	`EXTENT`	[extent]	Extent of the output raster layer. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`

continues on next page

Table 24.76 – continued from previous page

Label	Name	Type	Description
Output raster size	`PIXEL_SIZE`	[numeric: double] Default: 0.1	Pixel size of the output raster layer in layer units. In the GUI, the size can be specified by the number of rows (`Number of rows`) / columns (`Number of columns`) **or** the pixel size(`Pixel Size X` / `Pixel Size Y`). Increasing the number of rows or columns will decrease the cell size and increase the file size of the output raster. The values in `Rows`, `Columns`, `Pixel Size X` and `Pixel Size Y` will be updated simultaneously - doubling the number of rows will double the number of columns, and the cell size will be halved. The extent of the output raster will remain the same (approximately).
Interpolated	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Raster layer of interpolated values. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Interpolated	`OUTPUT`	[raster]	Raster layer of interpolated values

Python code

Algorithm ID: `qgis:idwinterpolation`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Line Density

Calculates for each raster cell, the density measure of linear features within a circular neighbourhood. This measure is obtained by summing all the line segments intersecting the circular neighbourhood and dividing this sum by the area of such neighbourhood. A weighting factor can be applied to the line segments.

Note

This algorithm uses ellipsoid based measurements and respects the current *ellipsoid settings*.

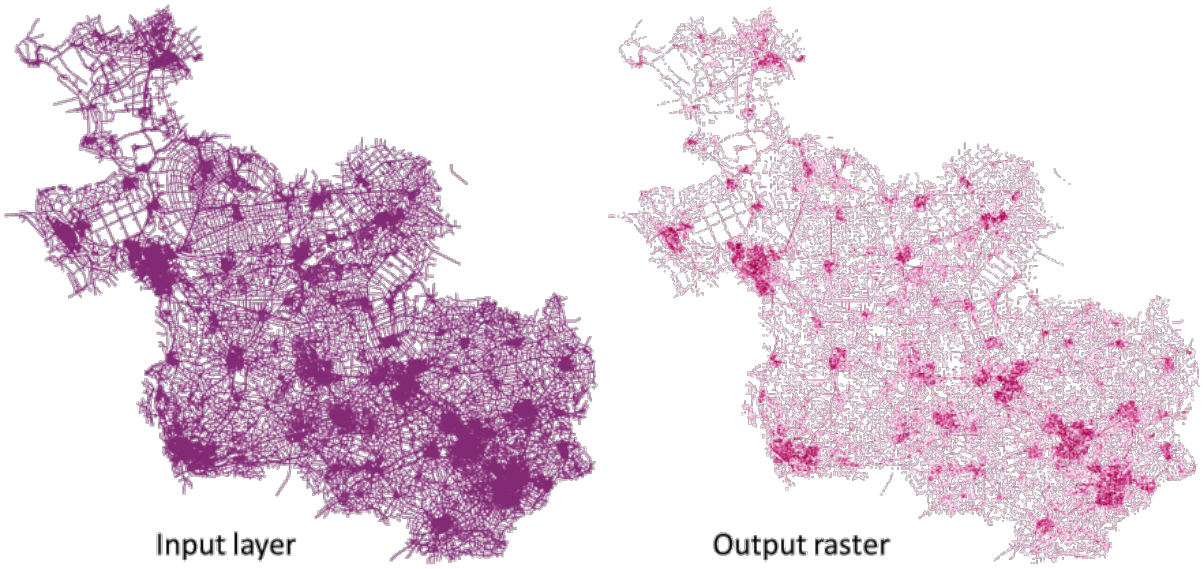


Fig. 24.20: Line density example. Input layer source: Roads Overijssel - The Netherlands (OSM).

Parameters

Basic parameters

Label	Name	Type	Description
Input line layer	`INPUT`	[vector: line]	Input vector layer containing line features
Weight field	`WEIGHT`	[tablefield: numeric]	Field of the layer containing the weight factor to use during the calculation
Search Radius	`RADIUS`	[numeric: double] Default: 10.0	Radius of the circular neighbourhood. Units can be specified here.
Pixel size	`PIXEL_SIZE`	[numeric: double] Default: 10.0	Pixel size of the output raster layer in layer units. The raster has square pixels.
Line density raster	`OUTPUT`	[raster] Default: `[Save to temporary file]`	The output as a raster layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `CREATE_OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Line density raster	`OUTPUT`	[raster]	The output line density raster layer.

Python code

Algorithm ID: `native:linedensity`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

TIN Interpolation

Generates a Triangulated Irregular Network (TIN) interpolation of a point vector layer.

With the TIN method you can create a surface formed by triangles of nearest neighbor points. To do this, circumcircles around selected sample points are created and their intersections are connected to a network of non overlapping and as compact as possible triangles. The resulting surfaces are not smooth.

The algorithm creates both the raster layer of the interpolated values and the vector line layer with the triangulation boundaries.

Parameters

Label	Name	Type	Description
Input layer(s)	`INTERPOLATION_DATA`	[string]	Vector layer(s) and field(s) to use for the interpolation, coded in a string (see the `ParameterInterpolationData` class in InterpolationWidgets for more details). The following GUI elements are provided to compose the interpolation data string: • **Vector layer** [vector: any] • **Interpolation attribute** [tablefield: numeric]: Attribute to use in the interpolation • **Use Z-coordinate for interpolation** [boolean]: Uses the layer's stored Z values (Default: False) For each of the added layer-field combinations, a type can be chosen: • *Points* • *Structured lines* • *Break lines* In the string, the layer-field elements are separated by `'::|::'`. The sub-elements of the layer-field elements are separated by `'::~::'`.

continues on next page

Table 24.81 – continued from previous page

Label	Name	Type	Description
Interpolation method	METHOD	[enumeration] Default: 0	Set the interpolation method to be used. One of: • *Linear* • *Clough-Toucher (cubic)*
Extent (xmin, xmax, ymin, ymax)	EXTENT	[extent]	Extent of the output raster layer. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Output raster size	PIXEL_SIZE	[numeric: double] Default: 0.1	Pixel size of the output raster layer in layer units. In the GUI, the size can be specified by the number of rows (`Number of rows`) / columns (`Number of columns`) **or** the pixel size(`Pixel Size X` / `Pixel Size Y`). Increasing the number of rows or columns will decrease the cell size and increase the file size of the output raster. The values in `Rows`, `Columns`, `Pixel Size X` and `Pixel Size Y` will be updated simultaneously - doubling the number of rows will double the number of columns, and the cell size will be halved. The extent of the output raster will remain the same (approximately).
Interpolated	OUTPUT	[raster] Default: `[Save to temporary file]`	The output TIN interpolation as a raster layer. *One of*: • Save to a Temporary File • Save to File…
Triangulation	TRIANGULATION	[vector: line] Default: `[Skip output]`	The output TIN as a vector layer. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Interpolated	`OUTPUT`	[raster]	The output TIN interpolation as a raster layer
Triangulation	`TRIANGULATION`	[vector: line]	The output TIN as a vector layer.

Python code

Algorithm ID: `qgis:tininterpolation`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.9 Layer tools

Export layer(s) information

Creates a polygon layer with features corresponding to the extent of selected layer(s).

Additional layer details (CRS, provider name, file path, layer name, subset filter, abstract and attribution) are attached as attributes to each feature.

Parameters

Label	Name	Type	Description
Input layers	`LAYERS`	[layer] [list]	Input layers to get information on.
Output	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specification of the output layer with information. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Output	`OUTPUT`	[vector: polygon]	Polygon vector layer showing extent of input layers and associated information in attributes.

Python code

Algorithm ID: `native:exportlayersinformation`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Export to spreadsheet

Exports the attributes of a selection of vector layers into a spreadsheet document or optionally appends them to an existing spreadsheet as additional sheets.

Parameters

Label	Name	Type	Description
Input layers	`LAYERS`	[vector: any] [list]	Input vector layers. The output spreadsheet will consist of a sheet, for each layer, that contains the attributes of this layer.
Use field aliases as column headings	`USE_ALIAS`	[boolean] Default: False	Use the field aliases from the attribute table for the spreadsheet.
Export formatted values instead of raw values	`FORMATTED_VALUES`	[boolean] Default: False	If `True`, exports the formatted, human readable values (e.g., from a *value map or value relation*) to the spreadsheet.
Overwrite existing spreadsheet	`OVERWRITE`	[boolean] Default: True	If the specified spreadsheet exists, setting this option to `True` will overwrite the existing spreadsheet. If this option is `False` and the spreadsheet exists, the layers will be appended as additional sheets.
Destination spreadsheet	`OUTPUT`	[file] Default: `[Save to temporary file]`	Output spreadsheet with a sheet for every layer. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Destination spreadsheet	`OUTPUT`	[file]	Spreadsheet with a sheet for every layer.
Layers within spreadsheet	`OUTPUT_LAYERS`	[list]	The list of sheets added to the spreadsheet.

Python code

Algorithm ID: `native:exporttospreadsheet`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Extract layer extent

Generates a vector layer with the minimum bounding box (rectangle with N-S orientation) that covers all the input features.

The output layer contains a single bounding box for the whole input layer.

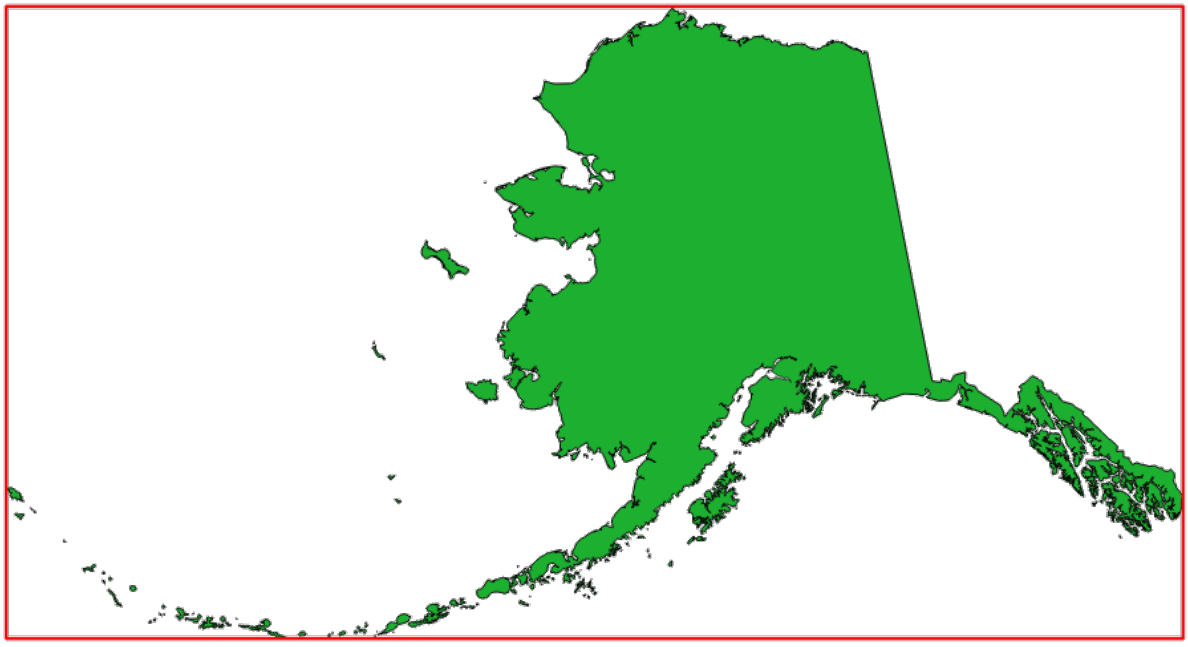


Fig. 24.21: In red the bounding box of the source layer

Default menu: *Vector ► Research Tools*

Parameters

Label	Name	Type	Description
Layer	`INPUT`	[layer]	Input layer
Extent	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the polygon vector layer for the output extent. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Extent	`OUTPUT`	[vector: polygon]	Output (polygon) vector layer with the extent (minimum bounding box)

Python code

Algorithm ID: `qgis:polygonfromlayerextent`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.10 Mesh

Export contours

Creates contours as a vector layer from a mesh scalar dataset.

Parameters

Label	Name	Type	Description
Input mesh layer	`INPUT`	[mesh]	The mesh layer to export data from
Dataset groups	`DATASET_GROUPS`	[layer] [list]	The dataset groups
Dataset time	`DATASET_TIME`	[datetime]	The time range to take into account • 0 — Current canvas time • 1 — Defined date/time • 2 — Dataset group time step

continues on next page

Table 24.83 – continued from previous page

Label	Name	Type	Description
Increment between contour levels Optional	INCREMENT	[numeric: double] Default: Not set	Interval between generated levels.
Minimum contour level Optional	MINIMUM	[numeric: double] Default: Not set	Starting level values of contours. No generated levels will be lower than this value.
Maximum contour level Optional	MAXIMUM	[numeric: double] Default: Not set	Maximum values of contours, i.e. no generated levels will be greater than this value.
List of contours level Optional	CONTOUR_LEVEL_LIS	[string] Default: Not set	List of desired levels of contours (separated by commas). If filled, the increment, minimum, and maximum fields will not be considered.
Output coordinate system Optional	CRS_OUTPUT	[crs]	Coordinate Reference System to assign to the output
Exported contour lines	OUTPUT_LINES	[vector: line] Default: [Create temporary layer]	Specify the output line layer representing the contours of the mesh layer. *One of*: • Create Temporary Layer (TEMPORARY_OUTPUT) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Exported contour polygons	OUTPUT_POLYGONS	[vector: polygon] Default: [Create temporary layer]	Specify the output polygon layer representing the contours of the mesh layer. *One of*: • Create Temporary Layer (TEMPORARY_OUTPUT) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Exported contour lines	OUTPUT_LINES	[vector: line]	Line layer representing the contours of the mesh layer.
Exported contour polygons	OUTPUT_POLYGONS	[vector: polygon]	Polygon layer representing the contours of the mesh layer.

Python code

Algorithm ID: `native:meshcontours`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Export cross section dataset values on lines from mesh

Extracts a mesh dataset's values from lines contained in a vector layer.

Each line is discretized with a resolution distance parameter for extraction of values on its vertices.

Parameters

Label	Name	Type	Description
Input mesh layer	`INPUT`	[mesh]	The mesh layer to export data from
Dataset groups	`DATASET_GROUPS`	[layer] [list]	The dataset groups
Dataset time	`DATASET_TIME`	[datetime]	The time range to take into account • 0 — Current canvas time • 1 — Defined date/time • 2 — Dataset group time step
Lines for data export	`INPUT_LINES`	[vector: line]	Lines where the data will be extracted from the dataset mesh
Line segmentation resolution	`RESOLUTION`	[numeric: double] Default: 10.0	The distance between points on the lines where the data will be extracted from the dataset mesh. Units can be selected.
Digits count for coordinates	`COORDINATES_DIGITS`	[numeric: integer] Default: 2	Number of digits for rounding the coordinate values
Digits count for dataset value	`DATASET_DIGITS`	[numeric: integer] Default: 2	Number of digits for rounding the dataset values
Exported data CSV file	`OUTPUT`	[file] Default: `[Save to temporary file]`	Specification of the output file. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Exported data CSV file	`OUTPUT`	[file]	

Python code

Algorithm ID: `native:meshexportcrosssection`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Export mesh edges

Exports a mesh layer's edges to a line vector layer, with the dataset values on edges as attribute values.

Parameters

Label	Name	Type	Description
Input mesh layer	`INPUT`	[mesh]	The mesh layer to export data from
Dataset groups	`DATASET_GROUPS`	[layer] [list]	The dataset groups
Dataset time	`DATASET_TIME`	[datetime]	The time range to take into account • 0 — Current canvas time • 1 — Defined date/time • 2 — Dataset group time step
Output coordinate system Optional	`CRS_OUTPUT`	[crs]	Coordinate Reference System to assign to the output
Export vector option	`VECTOR_OPTION`	[enumeration]	Coordinate type of vector value exportation. • 0 — Cartesian (x,y) • 1 — Polar (magnitude, degree) • 2 — Cartesian and polar
Output vector layer	`OUTPUT`	[vector: line] Default: `[Create temporary layer]`	Specification of the output file. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Output vector layer	OUTPUT	[vector: line]	Output vector line layer containing the edges of the input mesh layer with associated dataset values

Python code

Algorithm ID: native:exportmeshedges

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Export mesh faces

Exports a mesh layer's faces to a polygon vector layer, with the dataset values on faces as attribute values.

Parameters

Label	Name	Type	Description
Input mesh layer	INPUT	[mesh]	The mesh layer to export data from
Dataset groups	DATASET_GROUPS	[layer] [list]	The dataset groups
Dataset time	DATASET_TIME	[datetime]	The time range to take into account • 0 — Current canvas time • 1 — Defined date/time • 2 — Dataset group time step
Output coordinate system Optional	CRS_OUTPUT	[crs]	Coordinate Reference System to assign to the output
Export vector option	VECTOR_OPTION	[enumeration]	Coordinate type of vector value exportation. • 0 — Cartesian (x,y) • 1 — Polar (magnitude, degree) • 2 — Cartesian and polar
Output vector layer	OUTPUT	[vector: polygon] Default: [Create temporary layer]	Specification of the output file. *One of*: • Create Temporary Layer (TEMPORARY_OUTPUT) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Output vector layer	OUTPUT	[vector: polygon]	Output vector polygon layer containing the faces of the input mesh layer with associated dataset values

Python code

Algorithm ID: `native:exportmeshfaces`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Export mesh on grid

Exports a mesh layer's dataset values to a gridded point vector layer, with the dataset values on this point as attribute values.

For data on volume (3D stacked dataset values), the exported dataset values are averaged on faces using the method defined in *the mesh layer properties* (default is Multi level averaging method). 1D meshes are not supported.

Parameters

Label	Name	Type	Description
Input mesh layer	INPUT	[mesh]	The mesh layer to export data from
Dataset groups	DATASET_GROUPS	[layer] [list]	The dataset groups
Dataset time	DATASET_TIME	[datetime]	The time range to take into account • 0 — Current canvas time • 1 — Defined date/time • 2 — Dataset group time step

continues on next page

Table 24.87 – continued from previous page

Label	Name	Type	Description
Extent Optional	`EXTENT`	[extent]	Specify the spatial extent on which to process the data. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Grid spacing Optional	`GRID_SPACING`	[numeric: double] Default: 10.0	Spacing between the sample points to use
Output coordinate system Optional	`CRS_OUTPUT`	[crs]	Coordinate Reference System to assign to the output
Export vector option	`VECTOR_OPTION`	[enumeration]	Coordinate type of vector value exportation. • 0 — Cartesian (x,y) • 1 — Polar (magnitude, degree) • 2 — Cartesian and polar
Output vector layer	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specification of the output file. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Output vector layer	`OUTPUT`	[vector: point]	Output vector point layer with dataset values computed from the overlaid face.

Python code

Algorithm ID: `native:exportmeshongrid`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Export mesh vertices

Exports a mesh layer's vertices to a point vector layer, with the dataset values on vertices as attribute values.

Parameters

Label	Name	Type	Description
Input mesh layer	`INPUT`	[mesh]	The mesh layer to export data from
Dataset groups	`DATASET_GROUPS`	[layer] [list]	The dataset groups
Dataset time	`DATASET_TIME`	[datetime]	The time range to take into account • 0 — Current canvas time • 1 — Defined date/time • 2 — Dataset group time step
Output coordinate system Optional	`CRS_OUTPUT`	[crs]	Coordinate Reference System to assign to the output
Export vector option	`VECTOR_OPTION`	[enumeration]	Coordinate type of vector value exportation. • 0 — Cartesian (x,y) • 1 — Polar (magnitude, degree) • 2 — Cartesian and polar
Output vector layer	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specification of the output file. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Output vector layer	OUTPUT	[vector: point]	Output vector point layer containing the vertices of the input mesh layer with associated dataset values

Python code

Algorithm ID: native:exportmeshvertices

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Export time series values from points of a mesh dataset

Extracts a mesh dataset's time series values from points contained in a vector layer.

If the time step is kept to its default value (0 hours), the time step used is the one of the two first datasets of the first selected dataset group.

Parameters

Label	Name	Type	Description
Input mesh layer	INPUT	[mesh]	The mesh layer to extract data from
Dataset groups	DATASET_GROUPS	[layer] [list]	The dataset groups
Starting time	STARTING_TIME	[datetime]	The start of the time range to take into account • 0 — Current canvas time • 1 — Defined date/time • 2 — Dataset group time step
Finishing time	FINISHING_TIME	[datetime]	The end of the time range to take into account • 0 — Current canvas time • 1 — Defined date/time • 2 — Dataset group time step
Time step (hours) Optional	TIME_STEP	[numeric: double] Default: 0	Time between two consecutive steps to extract. Keep 0 to use time step of the first selected dataset group.
Points for data export	INPUT_POINTS	[vector: point]	Vector layer containing points where the data will be extracted from the dataset mesh
Digits count for coordinates	COORDINATES_DIGITS	[numeric: integer]	Number of digits for rounding coordinate values Default: 2
Digits count for dataset value	DATASET_DIGITS	[numeric: integer] Default: 2	Number of digits for rounding dataset values

continues on next page

Table 24.89 – continued from previous page

Label	Name	Type	Description
Exported data CSV file	`OUTPUT`	[file] Default: `[Save to temporary file]`	Specification of the output file. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Exported data CSV file	`OUTPUT`	[file]	`.CSV` file containing the mesh dataset time series values at the overlaying point features

Python code

Algorithm ID: `native:meshexporttimeseries`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Rasterize mesh dataset

Creates a raster layer from a mesh dataset.

For data on volume (3D stacked dataset values), the exported dataset values are averaged on faces using the method defined in *the mesh layer properties* (default is Multi level averaging method). 1D meshes are not supported.

Parameters

Basic parameters

Label	Name	Type	Description
Input mesh layer	`INPUT`	[mesh]	The mesh layer to export data from
Dataset groups	`DATASET_GROUPS`	[layer] [list]	The dataset groups
Dataset time	`DATASET_TIME`	[datetime]	The time range to take into account • 0 — Current canvas time • 1 — Defined date/time • 2 — Dataset group time step

continues on next page

Table 24.90 – continued from previous page

Label	Name	Type	Description
Extent Optional	`EXTENT`	[extent]	Specify the spatial extent on which to process the data. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Pixel size	`PIXEL_SIZE`	[numeric: double] Default: 1.0	Pixel size of the output raster layer.
Output coordinate system Optional	`CRS_OUTPUT`	[crs]	Coordinate Reference System to assign to the output
Output raster layer	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output file. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `CREATE_OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (`|`).

Outputs

Label	Name	Type	Description
Output raster layer	`OUTPUT`	[raster]	Output raster layer with dataset values computed from the mesh layer.

Python code

Algorithm ID: `native:meshrasterize`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Surface to Polygon

Exports a polygon file containing the boundary of a mesh layer. The resulting polygon may contain holes and may be a multi-part polygon.

Parameters

Label	Name	Type	Description
Input mesh layer	`INPUT`	[mesh]	The input mesh layer whose boundary will be exported as a polygon
Output coordinate system Optional	`OUTPUT_CRS`	[crs]	The coordinate reference system (CRS) for the output polygon layer. If not specified, the CRS of the input mesh layer will be used.
Output vector layer	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specification of the output file. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Output vector layer	`OUTPUT`	[vector: polygon]	The resulting polygon layer containing the boundary of the mesh layer. The polygon may contain holes and may be multi-part.

Python code

Algorithm ID: `native:surfacetopolygon`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

TIN mesh creation

Creates a TIN mesh layer from vector layers. The TIN mesh is created using a Delaunay triangulation.

Parameters

Label	Name	Type	Description
Input layers	`SOURCE_DATA`	[vector: any] [list]	Vector layers to combine to generate the mesh layer
Vector layer	GUI ONLY	[vector: any] [list]	A selector for the vector layers to combine to generate the mesh layer
Value on vertex	GUI ONLY	[tablefield: any]	A selector of the field to use from the selected layer. Each vertex is assigned the corresponding value of its original feature.
Use Z-coordinate for value on vertex	GUI ONLY	[boolean] Default: False	If checked, the Z value of vector layer points or polygons/lines vertices will be used to assign the Z value of the vertex mesh layer. Only available if the input layers are 3D.
Output format	`MESH_FORMAT`	[enumeration] Default: 2DM	Output format of the generated layer • 0 — 2DM • 1 — SELAFIN • 2 — PLY • 3 — Ugrid • 4 — Mike21
Output coordinate system Optional	`CRS_OUTPUT`	[crs]	Coordinate Reference System to assign to the output
Output file	`OUTPUT_MESH`	[mesh] Default: `[Save to temporary file]`	Specification of the output file. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Output file	`OUTPUT_MESH`	[mesh]	Output mesh layer with dataset values computed from the vector layers.

Python code

Algorithm ID: `native:tinmeshcreation`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.11 Metadata tools

Add history metadata

Adds a new history entry to the layer's metadata.

Parameters

Label	Name	Type	Description
Layer	`INPUT`	[layer]	The input layer (vector, raster, etc.) to which the history entry will be added.
History entry	`HISTORY`	[string]	The text to be appended as a new entry in the layer's history metadata. This will be added to any existing history entries.

Outputs

Label	Name	Type	Description
Output	`OUTPUT`	[same as input]	The resulting layer with the updated history in its **Metadata properties**.

Python code

Algorithm ID: `native:addhistorymetadata`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Copy Layer Metadata

Copies metadata from a source layer to a target layer. Any existing metadata in the target layer will be replaced with the metadata from the source layer. This includes all metadata fields, such as history, abstract, and other properties.

Parameters

Label	Name	Type	Description
Source layer	`SOURCE`	[layer]	The layer from which metadata will be copied.
Target layer	`TARGET`	[layer]	The layer to which metadata will be pasted. Any existing metadata in this layer will be replaced.
Save metadata as default	`DEFAULT`	[boolean] Default: False	If checked, the metadata information will be saved with the layer, hence available by default in subsequent projects.

Outputs

Label	Name	Type	Description
Output layer	`OUTPUT`	[layer]	The target layer with the metadata replaced by the metadata from the source layer. This includes all metadata fields, such as history, abstract, and other properties.

Python code

Algorithm ID: `native:copylayermetadata`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Export Layer Metadata

Exports the metadata of a layer to a QMD file.

Parameters

Label	Name	Type	Description
Layer	`INPUT`	[layer]	The layer whose metadata will be exported.
Output	`OUTPUT`	[file] Default: `[Save to temporary file]`	Specification of the output file. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Output file	`OUTPUT`	[file]	The `.qmd` file containing the exported metadata. This file can be used to import metadata into another layer.

Python code

Algorithm ID: `native:exportlayermetadata`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Set Layer Metadata

Applies metadata to a layer from a `.qmd` file.

Parameters

Label	Name	Type	Description
Layer	`INPUT`	[layer]	The layer to which the metadata will be applied.
Metadata file	`METADATA`	[file]	The `.qmd` file containing the metadata to be applied.
Save metadata as default	`DEFAULT`	[boolean] Default: False	If checked, the metadata information will be saved with the layer, hence available by default in subsequent projects.

Outputs

Label	Name	Type	Description
Output layer	OUTPUT	[same as input]	The input layer with the metadata replaced by the metadata from the .qmd file.

Python code

Algorithm ID: native:setlayermetadata

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Set Metadata Fields

Sets various metadata fields for a layer.

Parameters

Label	Name	Type	Description
Layer	INPUT	[layer]	The layer whose metadata fields will be updated
Identifier Optional	IDENTIFIER	[string]	A unique identifier for the layer
Parent identifier Optional	PARENT_IDENTIFIER	[string]	The identifier of the parent resource, if applicable
Title Optional	TITLE	[string]	The title of the layer
Type Optional	TYPE	[string]	The type of data stored in the layer
Language Optional	LANGUAGE	[string]	The language of the metadata
Encoding Optional	ENCODING	[string]	The character encoding used in the metadata
Abstract Optional	ABSTRACT	[string]	A brief description or abstract of the layer
Coordinate reference system Optional	CRS	[crs]	The coordinate reference system of the layer
Fees Optional	FEES	[string]	Information about any fees associated with accessing the layer
Ignore empty fields	IGNORE_EMPTY	[boolean] Default: False	If checked, no update will be done to metadata fields that are not filled

Outputs

Label	Name	Type	Description
Output layer	`OUTPUT`	[same as input]	The input layer with the specified metadata fields updated.

Python code

Algorithm ID: `native:setmetadatafields`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Update Layer Metadata

Copies all non-empty metadata fields from a source layer to a target layer. Leaves empty input fields unchanged in the target.

Parameters

Label	Name	Type	Description
Source layer	`SOURCE`	[layer]	The layer from which non-empty metadata fields will be copied
Target layer	`TARGET`	[layer]	The layer to which non-empty metadata fields will be pasted. Empty fields in the source layer will not overwrite existing metadata in the target layer.

Outputs

Label	Name	Type	Description
Target layer	`OUTPUT`	[layer]	The target layer with updated metadata.

Python code

Algorithm ID: `native:updatelayermetadata`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.12 Modeler tools

> **Warning**
>
> These tools are only available in the model designer. They are not available in the Processing Toolbox.

Calculate expression

It calculates the result of a QGIS expression and eliminates the need to use the same expression multiple times throughout a model when the same result is required more than once. Additionally, it enables use cases that would otherwise not be possible. For instance, you can generate a timestamp value once and use it multiple times within the model, if the timestamp were recalculated every time, the values would vary during the model's runtime.

Parameters

Label	Name	Type	Description
Input	`INPUT`	[expression]	Expression to calculate

Outputs

Label	Name	Type	Description
Value	`OUTPUT`	[Result Value]	Calculated result value, the data type of the output will vary based on the specific expression used in the algorithm.

Python code

Algorithm ID: `native:calculateexpression`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Conditional branch

Adds a conditional branch into a model, allowing parts of the model to be executed based on the result of an expression evaluation. Mostly by using tool dependencies to control the flow of a model.

Parameters

Label	Name	Type	Description
Field	`BRANCH`	[string]	Name of the condition
Field	`CONDITION`	[expression]	Expression to evaluate

Outputs

None.

Python code

Algorithm ID: `native:condition`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Create directory

Creates a new directory on a file system. Directories will be created recursively, creating all required parent directories in order to construct the full specified directory path. No errors will be raised if the directory already exists.

Parameters

Label	Name	Type	Description
Directory path	`PATH`	[string]	Folder path to create

Outputs

Label	Name	Type	Description
Output	`OUTPUT`	[folder]	Created folder

Python code

Algorithm ID: `native:createdirectory`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Feature filter

Filters features from the input layer and redirects them to one or several outputs. If you do not know about any attribute names that are common to all possible input layers, filtering is only possible on the feature geometry and general record mechanisms, such as `$id` and `uuid`.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	The input layer.
Outputs and filters (one or more)	`OUTPUT_<name of the filter>`	[same as input]	The output layers with filters (as many as there are filters).

Outputs

Label	Name	Type	Description
Output (one or more)	`native:filter_1: of filter>`	[same as input]	The output layers with filtered features (as many as there are filters).

Python code

Algorithm ID: `native:filter`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Filter layers by type

Filters layers by their type. Incoming layers will be directed to different outputs based on whether they are a vector or raster layer.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[layer]	Generic Map Layer

Outputs

Label	Name	Type	Description
Vector features Optional	VECTOR	[vector]	A Vector Layer of the input, if compatible
Raster layer Optional	RASTER	[raster]	A Raster Layer of the input, if compatible

Python code

Algorithm ID: `native:filterlayersbytype`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Load layer into project

Loads a layer to the current project.

Parameters

Label	Name	Type	Description
Layer	INPUT	[layer]	Layer to load in the legend
Loaded layer name	NAME	[string]	Name of the loaded layer

Outputs

Label	Name	Type	Description
Layer	OUTPUT	[same as input]	The (renamed) loaded layer

Python code

Algorithm ID: `native:loadlayer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Raise exception

Raises an exception and cancels a model's execution. The exception message can be customized, and optionally an expression based condition can be specified. If an expression condition is used, then the exception will only be raised if the expression result is true. A false result indicates that no exception will be raised, and the model execution can continue uninterrupted.

Parameters

Label	Name	Type	Description
Message	`MESSAGE`	[string]	Message to display
Condition Optional	`CONDITION`	[expression]	Expression to evaluate if true

Outputs

A message in the log panel.

Python code

Algorithm ID: `native:raiseexception`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Raise message

Raises an information message in the log. The message can be customized, and optionally an expression based condition can be specified. If an expression condition is used, then the message will only be logged if the expression result is true. A false result indicates that no message will be logged.

Parameters

Label	Name	Type	Description
Information message	`MESSAGE`	[string]	Message to display
Condition Optional	`CONDITION`	[expression]	Expression to evaluate if true

Outputs

A message in the log panel.

Python code

Algorithm ID: `native:raisemessage`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Raise warning

Raises a warning message in the log. The warning message can be customized, and optionally an expression based condition can be specified. If an expression condition is used, then the warning will only be logged if the expression result is true. A false result indicates that no warning will be logged.

Parameters

Label	Name	Type	Description
Message	`MESSAGE`	[string]	Message to display
Condition Optional	`CONDITION`	[expression]	Expression to evaluate if true

Outputs

A message in the log panel.

Python code

Algorithm ID: `native:raisewarning`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Rename layer

Renames a layer.

Parameters

Label	Name	Type	Description
Layer	`INPUT`	[layer]	Layer to rename
New name	`NAME`	[string]	The new name of the layer

Outputs

Label	Name	Type	Description
Layer	`OUTPUT`	[same as input]	The (renamed) output layer

Python code

Algorithm ID: `native:renamelayer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Save log to file

Saves the model's execution log to a file. Optionally, the log can be saved in a HTML formatted version.

Parameters

Label	Name	Type	Description
Use HTML	`USE_HTML`	[Boolean] Default: False	Use HTML formatting

Outputs

Label	Name	Type	Description
File	`OUTPUT`	[string]	Destination of the log

Python code

Algorithm ID: `native:savelog`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Set project variable

Sets an expression variable for the current project.

Parameters

Label	Name	Type	Description
Variable name	`NAME`	[string]	Name of the variable
Variable value	`VALUE`	[string]	Value to be stored

Outputs

None.

Python code

Algorithm ID: `native:setprojectvariable`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

String concatenation

Concatenates two strings into a single one in the Processing Modeler.

Parameters

Label	Name	Type	Description
Input 1	`INPUT_1`	[string]	First string
Input 2	`INPUT_2`	[string]	Second string

Outputs

Label	Name	Type	Description
Concatenation	`CONCATENATION`	[string]	The concatenated string

Python code

Algorithm ID: `native:stringconcatenation`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Variable distance buffer

Warning

This algorithm is deprecated and can be removed anytime. Prefer using *Buffer* algorithm instead.

Computes a buffer area for all the features in an input layer.

The size of the buffer for a given feature is defined by an attribute, so it allows different features to have different buffer sizes.

See also

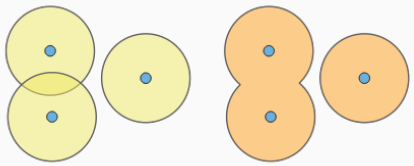
Buffer

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	Input vector layer
Distance field	`DISTANCE`	[tablefield: numeric]	Attribute for the distance radius of the buffer
Segments	`SEGMENTS`	[numeric: integer] Default: *5*	Controls the number of line segments to use to approximate a quarter circle when creating rounded offsets.

continues on next page

Table 24.93 – continued from previous page

Label	Name	Type	Description
Dissolve result	DISSOLVE	[boolean] Default: *False*	Choose to dissolve the final buffer, resulting in a single feature covering all input features.  Fig. 24.22: Normal and dissolved buffer
End cap style	END_CAP_STYLE	[enumeration] Default: Round	Controls how line endings are handled in the buffer.  Fig. 24.23: Round, flat and square cap styles
Join style	JOIN_STYLE	[enumeration] Default: Round	Specifies whether round, miter or beveled joins should be used when offsetting corners in a line.
Miter limit	MITER_LIMIT	[numeric: double] Default: 2.0	Only applicable for mitered join styles, and controls the maximum distance from the offset curve to use when creating a mitered join.

Outputs

Label	Name	Type	Description
Buffer	OUTPUT	[vector: polygon]	Buffer polygon vector layer.

Python code

Algorithm ID: `qgis:variabledistancebuffer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.13 Network analysis

Service area (from layer)

Returns all the edges or parts of edges of a network that can be reached within a distance or a time, starting from a point layer. This allows evaluation of accessibility within a network, e.g., what are the places I can navigate to on a road network without spending cost greater than a given value (the cost can be distance or time).

Note

This algorithm uses ellipsoid based measurements and respects the current *ellipsoid settings*.

Parameters

Basic parameters

Label	Name	Type	Description
Vector layer representing network	`INPUT`	[vector: line]	Line vector layer representing the network to be covered
Vector layer with start points	`START_POINTS`	[vector: point]	Point vector layer whose features are used as start points to generate the service areas
Path type to calculate	`STRATEGY`	[enumeration] Default: 0	The type of path to calculate. One of: • 0 — Shortest • 1 — Fastest
Travel cost (distance for "Shortest", time for "Fastest")	`TRAVEL_COST`	[numeric: double] Default: 0.0	The value is estimated as a distance (in the network layer units) when looking for the *Shortest* path and as time (in hours) for the *Fastest* path.
Service area (lines)	`OUTPUT_LINES`	[vector: line] Default: `[Create temporary layer]`	Specify the output line layer for the service area. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Service area (boundary nodes)	`OUTPUT`	[vector: point] Default: `[Skip output]`	Specify the output point layer for the service area boundary nodes. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

continues on next page

Table 24.94 – continued from previous page

Label	Name	Type	Description
Non-routable features	`OUTPUT_NON_ROUTAB`	[vector: point] Default: `[Skip output]`	Specify the output which will be used to store any input features which could not be routed (e.g., those which are too far from the network layer). *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Direction field Optional	`DIRECTION_FIELD`	[tablefield: any]	The field used to specify directions for the network edges. The values used in this field are specified with the three parameters `Value for forward direction`, `Value for backward direction` and `Value for both directions`. Forward and reverse directions correspond to a one-way edge, "both directions" indicates a two-way edge. If a feature does not have a value in this field, or no field is set then the default direction setting (provided with the `Default direction` parameter) is used.
Value for forward direction Optional	`VALUE_FORWARD`	[string] Default: '' (empty string)	Value set in the direction field to identify edges with a forward direction
Value for backward direction Optional	`VALUE_BACKWARD`	[string] Default: '' (empty string)	Value set in the direction field to identify edges with a backward direction
Value for both directions Optional	`VALUE_BOTH`	[string] Default: '' (empty string)	Value set in the direction field to identify bidirectional edges
Default direction	`DEFAULT_DIRECTIO`	[enumeration] Default: 2	If a feature has no value set in the direction field or if no direction field is set, then this direction value is used. One of: • 0 — Forward direction • 1 — Backward direction • 2 — Both directions
Speed field Optional	`SPEED_FIELD`	[tablefield: numeric]	Field providing the speed value (in `km/h`) for the edges of the network when looking for the fastest path. If a feature does not have a value in this field, or no field is set then the default speed value (provided with the `Default speed` parameter) is used.

continues on next page

Table 24.95 – continued from previous page

Label	Name	Type	Description
Default speed (km/h)	DEFAULT_SPEED	[numeric: double] Default: 50.0	Value to use to calculate the travel time if no speed value is provided for an edge in the specified field
Topology tolerance	TOLERANCE	[numeric: double] Default: 0.0	Two lines with nodes closer than the specified tolerance are considered connected
Include upper/lower bound points	INCLUDE_BOUNDS	[boolean] Default: False	Creates a point layer output with two points for each edge at the boundaries of the service area. One point is the start of that edge, the other is the end.
Maximum point distance from network Optional	POINT_TOLERANC	[numeric: double] Default: Not set	Specifies an optional limit on the distance from the points to the network layer. If a point is further from the network than this distance it will be treated as non-routable. If not set, endpoints will be snapped to the nearest point on the network layer, regardless of how far away from the network they actually are.

Outputs

Label	Name	Type	Description
Service area (boundary nodes)	OUTPUT	[vector: point]	The output point layer with the service area boundary nodes.
Service area (lines)	OUTPUT_LINES	[vector: line]	Line layer representing the parts of the network that can be serviced by the start points, for the given cost.
Non routable features	OUTPUT_NON_ROUTAB	[vector: point]	An optional output which will be used to store any input features which could not be routed (e.g., those which are too far from the network layer).

Python code

Algorithm ID: `qgis:serviceareafromlayer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Service area (from point)

Returns all the edges or parts of edges of a network that can be reached within a given distance or time, starting from a point feature. This allows the evaluation of accessibility within a network, e.g., what are the places I can navigate to on a road network without spending a cost greater than a given value (the cost can be distance or time).

> **Note**
>
> This algorithm uses ellipsoid based measurements and respects the current *ellipsoid settings*.

Parameters

Basic parameters

Label	Name	Type	Description
Vector layer representing the network	`INPUT`	[vector: line]	Line vector layer representing the network to be covered
Path type to calculate	`STRATEGY`	[enumeration] Default: 0	The type of path to calculate. One of: • 0 — Shortest • 1 — Fastest
Start point	`START_POINT`	[coordinates]	Coordinate of the point to calculate the service area around. Press the … button next to the option and click on the canvas to fill the parameter with the clicked point coordinate.
Travel cost (distance for "Shortest", time for "Fastest")	`TRAVEL_COST`	[numeric: double] Default: 0.0	The value is estimated as a distance (in the network layer units) when looking for the *Shortest* path and as time (in hours) for the *Fastest* path.
Service area (lines) Optional	`OUTPUT_LINES`	[vector: line] Default: `[Create temporary layer]`	Specify the output line layer for the service area. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Service area (boundary nodes) Optional	`OUTPUT`	[vector: point] Default: `[Skip output]`	Specify the output point layer for the service area boundary nodes. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Direction field Optional	DIRECTION_FIELD	[tablefield: any]	The field used to specify directions for the network edges. The values used in this field are specified with the three parameters Value for forward direction, Value for backward direction and Value for both directions. Forward and reverse directions correspond to a one-way edge, "both directions" indicates a two-way edge. If a feature does not have a value in this field, or no field is set then the default direction setting (provided with the Default direction parameter) is used.
Value for forward direction Optional	VALUE_FORWARD	[string] Default: '' (empty string)	Value set in the direction field to identify edges with a forward direction
Value for backward direction Optional	VALUE_BACKWARD	[string] Default: '' (empty string)	Value set in the direction field to identify edges with a backward direction
Value for both directions Optional	VALUE_BOTH	[string] Default: '' (empty string)	Value set in the direction field to identify bidirectional edges
Default direction	DEFAULT_DIRECTIO	[enumeration] Default: 2	If a feature has no value set in the direction field or if no direction field is set, then this direction value is used. One of: • 0 — Forward direction • 1 — Backward direction • 2 — Both directions
Speed field Optional	SPEED_FIELD	[tablefield: numeric]	Field providing the speed value (in km/h) for the edges of the network when looking for the fastest path. If a feature does not have a value in this field, or no field is set then the default speed value (provided with the Default speed parameter) is used.
Default speed (km/h)	DEFAULT_SPEED	[numeric: double] Default: 50.0	Value to use to calculate the travel time if no speed value is provided for an edge in the specified field
Topology tolerance	TOLERANCE	[numeric: double] Default: 0.0	Two lines with nodes closer than the specified tolerance are considered connected

Maximum point distance from network Optional	POINT_TOLERANC	[numeric: double] Default: 0.0	Specifies an optional limit on the distance from the start point to the network layer. If the point is further from the network than this distance an error will be raised. If not set, the point will be snapped to the nearest point on the network layer, regardless of how far away from the network it actually is.
Include upper/lower bound points	IN-CLUDE_BOUNDS	[boolean] Default: False	Creates a point layer output with two points for each edge at the boundaries of the service area. One point is the start of that edge, the other is the end.

Outputs

Label	Name	Type	Description
Service area (boundary nodes)	OUTPUT	[vector: point]	The output point layer with the service area boundary nodes.
Service area (lines)	OUTPUT_LINES	[vector: line]	Line layer representing the parts of the network that can be serviced by the start point, for the given cost.

Python code

Algorithm ID: `native:serviceareafrompoint`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Shortest path (layer to point)

Computes the optimal (shortest or fastest) routes from multiple start points defined by a vector layer and a given end point.

> **Note**
>
> This algorithm uses ellipsoid based measurements and respects the current *ellipsoid settings*.

Parameters

Basic parameters

Label	Name	Type	Description
Vector layer representing network	`INPUT`	[vector: line]	Line vector layer representing the network to be covered
Path type to calculate	`STRATEGY`	[enumeration] Default: 0	The type of path to calculate. One of: • 0 — Shortest • 1 — Fastest
Vector layer with start points	`START_POINTS`	[vector: point]	Point vector layer whose features are used as start points of the routes
End point (x, y)	`END_POINT`	[coordinates]	Point feature representing the end point of the routes Press the … button next to the option and click on the canvas to fill the parameter with the clicked point coordinate.
Shortest path	`OUTPUT`	[vector: line] Default: `[Create temporary layer]`	Specify the output line layer for the shortest paths. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Non-routable features Optional	`OUTPUT_NON_ROUTAB`	[vector: point] Default: `[Skip output]`	Specify the output which will be used to store any input features which could not be routed (e.g., those which are too far from the network layer). *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Direction field Optional	DIREC- TION_FIELD	[tablefield: any]	The field used to specify directions for the network edges. The values used in this field are specified with the three parameters `Value for forward direction`, `Value for backward direction` and `Value for both directions`. Forward and reverse directions correspond to a one-way edge, "both directions" indicates a two-way edge. If a feature does not have a value in this field, or no field is set then the default direction setting (provided with the `Default direction` parameter) is used.
Value for forward direction Optional	VALUE_FORWARD	[string] Default: '' (empty string)	Value set in the direction field to identify edges with a forward direction
Value for backward direction Optional	VALUE_BACKWARD	[string] Default: '' (empty string)	Value set in the direction field to identify edges with a backward direction
Value for both directions Optional	VALUE_BOTH	[string] Default: '' (empty string)	Value set in the direction field to identify bidirectional edges
Default direction	DE- FAULT_DIRECTIO	[enumeration] Default: 2	If a feature has no value set in the direction field or if no direction field is set, then this direction value is used. One of: • 0 — Forward direction • 1 — Backward direction • 2 — Both directions
Speed field Optional	SPEED_FIELD	[tablefield: numeric]	Field providing the speed value (in `km/h`) for the edges of the network when looking for the fastest path. If a feature does not have a value in this field, or no field is set then the default speed value (provided with the `Default speed` parameter) is used.
Default speed (km/h)	DEFAULT_SPEED	[numeric: double] Default: 50.0	Value to use to calculate the travel time if no speed value is provided for an edge in the specified field
Topology tolerance	TOLERANCE	[numeric: double] Default: 0.0	Two lines with nodes closer than the specified tolerance are considered connected

Maximum point distance from network Optional	POINT_TOLERANC	[numeric: double] Default: Not set	Specifies an optional limit on the distance from the start and end points to the network layer. If a start feature is further from the network than this distance it will be treated as non-routable. If the end point is further from the network than this distance an error will be raised. If not set, points will be snapped to the nearest point on the network layer, regardless of how far away from the network they actually are.

Outputs

Label	Name	Type	Description
Shortest path	`OUTPUT`	[vector: line]	Line layer of the shortest or fastest path from each of the start points to the end point
Non routable features	`OUTPUT_NON_ROUTAB`	[vector: point]	An optional output layer which will be used to store any input features which could not be routed (e.g., those which are too far from the network layer).

Python code

Algorithm ID: `native:shortestpathlayertopoint`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Shortest path (point to layer)

Computes the optimal (shortest or fastest) routes between a given start point and multiple end points defined by a point vector layer.

Note

This algorithm uses ellipsoid based measurements and respects the current *ellipsoid settings*.

Warning

This algorithm drops existing primary keys or FID values and regenerates them in output layers.

Parameters

Basic parameters

Label	Name	Type	Description
Vector layer representing network	`INPUT`	[vector: line]	Line vector layer representing the network to be covered
Path type to calculate	`STRATEGY`	[enumeration] Default: 0	The type of path to calculate. One of: • 0 — Shortest • 1 — Fastest

continues on next page

Table 24.100 – continued from previous page

Label	Name	Type	Description
Start point	START_POINT	[coordinates]	Point feature representing the start point of the routes Press the … button next to the option and click on the canvas to fill the parameter with the clicked point coordinate.
Vector layer with end points	END_POINTS	[vector: point]	Point vector layer whose features are used as end points of the routes
Shortest path	OUTPUT	[vector: line] Default: [Create temporary layer]	Specify the output line layer for the shortest paths. *One of*: • Create Temporary Layer (TEMPORARY_OUTPUT) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Non-routable features Optional	OUTPUT_NON_ROUTAB	[vector: point] Default: [Skip output]	Specify the output which will be used to store any input features which could not be routed (e.g., those which are too far from the network layer). *One of*: • Skip Output • Create Temporary Layer (TEMPORARY_OUTPUT) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Direction field Optional	DIRECTION_FIELD	[tablefield: any]	The field used to specify directions for the network edges. The values used in this field are specified with the three parameters Value for forward direction, Value for backward direction and Value for both directions. Forward and reverse directions correspond to a one-way edge, "both directions" indicates a two-way edge. If a feature does not have a value in this field, or no field is set then the default direction setting (provided with the Default direction parameter) is used.
Value for forward direction Optional	VALUE_FORWARD	[string] Default: '' (empty string)	Value set in the direction field to identify edges with a forward direction
Value for backward direction Optional	VALUE_BACKWARD	[string] Default: '' (empty string)	Value set in the direction field to identify edges with a backward direction
Value for both directions Optional	VALUE_BOTH	[string] Default: '' (empty string)	Value set in the direction field to identify bidirectional edges

continues on next page

Table 24.101 – continued from previous page

Label	Name	Type	Description
Default direction	DEFAULT_DIRECTIO	[enumeration] Default: 2	If a feature has no value set in the direction field or if no direction field is set, then this direction value is used. One of: • 0 — Forward direction • 1 — Backward direction • 2 — Both directions
Speed field Optional	SPEED_FIELD	[tablefield: numeric]	Field providing the speed value (in km/h) for the edges of the network when looking for the fastest path. If a feature does not have a value in this field, or no field is set then the default speed value (provided with the Default speed parameter) is used.
Default speed (km/h)	DEFAULT_SPEED	[numeric: double] Default: 50.0	Value to use to calculate the travel time if no speed value is provided for an edge in the specified field
Topology tolerance	TOLERANCE	[numeric: double] Default: 0.0	Two lines with nodes closer than the specified tolerance are considered connected
Maximum point distance from network Optional	POINT_TOLERANC	[numeric: double] Default: Not set	Specifies an optional limit on the distance from the start and end points to the network layer. If the start point is further from the network than this distance an error will be raised. If an end feature is further from the network than this distance it will be treated as non-routable. If not set, points will be snapped to the nearest point on the network layer, regardless of how far away from the network they actually are.

Outputs

Label	Name	Type	Description
Shortest path	OUTPUT	[vector: line]	Line layer of the shortest or fastest path from each of the start points to the end point
Non routable features	OUTPUT_NON_ROUTAB	[vector: point]	An optional output layer which will be used to store any input features which could not be routed (e.g., those which are too far from the network layer).

Python code

Algorithm ID: `native:shortestpathpointtolayer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Shortest path (point to point)

Computes the optimal (shortest or fastest) route between a given start point and a given end point.

Note

This algorithm uses ellipsoid based measurements and respects the current *ellipsoid settings*.

Parameters

Basic parameters

Label	Name	Type	Description
Vector layer representing network	`INPUT`	[vector: line]	Line vector layer representing the network to be covered
Path type to calculate	`STRATEGY`	[enumeration] Default: 0	The type of path to calculate. One of: • 0 — Shortest • 1 — Fastest
Start point (x, y)	`START_POINT`	[coordinates]	Point feature representing the start point of the routes. Press the … button next to the option and click on the canvas to fill the parameter with the clicked point coordinate.
End point (x, y)	`END_POINT`	[coordinates]	Point feature representing the end point of the routes. Press the … button next to the option and click on the canvas to fill the parameter with the clicked point coordinate.
Shortest path	`OUTPUT`	[vector: line]	Specify the output line layer for the shortest paths. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Direction field Optional	DIRECTION_FIELD	[tablefield: any]	The field used to specify directions for the network edges. The values used in this field are specified with the three parameters Value for forward direction, Value for backward direction and Value for both directions. Forward and reverse directions correspond to a one-way edge, "both directions" indicates a two-way edge. If a feature does not have a value in this field, or no field is set then the default direction setting (provided with the Default direction parameter) is used.
Value for forward direction Optional	VALUE_FORWARD	[string] Default: '' (empty string)	Value set in the direction field to identify edges with a forward direction
Value for backward direction Optional	VALUE_BACKWARD	[string] Default: '' (empty string)	Value set in the direction field to identify edges with a backward direction
Value for both directions Optional	VALUE_BOTH	[string] Default: '' (empty string)	Value set in the direction field to identify bidirectional edges
Default direction	DEFAULT_DIRECTIO	[enumeration] Default: 2	If a feature has no value set in the direction field or if no direction field is set, then this direction value is used. One of: • 0 — Forward direction • 1 — Backward direction • 2 — Both directions
Speed field Optional	SPEED_FIELD	[tablefield: numeric]	Field providing the speed value (in km/h) for the edges of the network when looking for the fastest path. If a feature does not have a value in this field, or no field is set then the default speed value (provided with the Default speed parameter) is used.
Default speed (km/h)	DEFAULT_SPEED	[numeric: double] Default: 50.0	Value to use to calculate the travel time if no speed value is provided for an edge in the specified field
Topology tolerance	TOLERANCE	[numeric: double] Default: 0.0	Two lines with nodes closer than the specified tolerance are considered connected

Maximum point distance from network Optional	POINT_TOLERANC	[numeric: double] Default: Not set	Specifies an optional limit on the distance from the start and end points to the network layer. If either point is further from the network than this distance an error will be raised. If not set, points will be snapped to the nearest point on the network layer, regardless of how far away from the network they actually are.

Outputs

Label	Name	Type	Description
Shortest path	`OUTPUT`	[vector: line]	Line layer of the shortest or fastest path from each of the start point to the end point

Python code

Algorithm ID: `native:shortestpathpointtopoint`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.14 Plots

Bar plot

Creates a bar plot from a category and a layer field.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Category field name	`NAME_FIELD`	[tablefield: any]	Categorical field to use for grouping the bars (X axis)
Value field	`VALUE_FIELD`	[tablefield: numeric]	Value to use for the plot (Y axis).
Title Optional	`TITLE`	[string] Default: “”	Title of the plot
X-axis Title Optional	`XAXIS_TITLE`	[string] Default: “”	If empty, the name of the category field is used. With a single space, the axis title is hidden.
Y-axis Title Optional	`YAXIS_TITLE`	[string] Default: “”	If empty, the name of the value field is used. With a single space, the axis title is hidden.
Bar plot	`OUTPUT`	[html] Default: `[Save to temporary file]`	Specify the HTML file for the plot. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Bar plot	`OUTPUT`	[html]	HTML file with the plot. Available in the *Processing* ► *Result Viewer*.

Python code

Algorithm ID: `qgis:barplot`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Box plot

Creates a box plot from a category field and a numerical layer field.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Category name field	`NAME_FIELD`	[tablefield: any]	Categorical field to use for grouping the boxes (X axis)
Value field	`VALUE_FIELD`	[tablefield: numeric]	Value to use for the plot (Y axis).
Additional statistic lines	`MSD`	[enumeration] Default: 0	Additional statistics information to add to the plot. One of: • 0 — Show Mean • 1 — Show Standard Deviation • 2 — Don't show mean and standard deviation
Title Optional	`TITLE`	[string] Default: ""	Title of the plot
X-axis Title Optional	`XAXIS_TITLE`	[string] Default: ""	If empty, the name of the category field is used. With a single space, the axis title is not shown.
Y-axis Title Optional	`YAXIS_TITLE`	[string] Default: ""	If empty, the name of the value field is used. With a single space, the axis title is not shown.
Box plot	`OUTPUT`	[html] Default: `[Save to temporary file]`	Specify the HTML file for the plot. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Box plot	`OUTPUT`	[html]	HTML file with the plot. Available in the *Processing* ► *Result Viewer*.

Python code

Algorithm ID: `qgis:boxplot`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Generate elevation profile image

Creates an elevation profile image from a list of map layers and an optional terrain layer.

Parameters

Basic parameters

Label	Name	Type	Description
Profile curve	`CURVE`	[vector: line]	The line layer representing the profile curve along which the elevation profile will be generated
Map layers	`MAP_LAYERS`	[layer] [list]	The list of map layers to include in the elevation profile
Chart width (in pixels)	`WIDTH`	[numeric: integer] Default: 400 Minimum value: 0	The width of the output chart in pixels.
Chart height (in pixels)	`HEIGHT`	[numeric: integer] Default: 300 Minimum value: 0	The height of the output chart in pixels.
Terrain layer Optional	`TERRAIN_LAYER`	[raster]	A terrain layer (e.g., DEM) to use for elevation data. If not provided, elevation data will be derived from the map layers.

Advanced parameters

Label	Name	Type	Description
Chart minimum distance (X axis) Optional	`MINIMUM_DISTANCE`	[numeric: double]	The minimum distance (X axis) to display on the chart. If not specified, the chart will auto-scale.
Chart maximum distance (X axis) Optional	`MAXIMUM_DISTANCE`	[numeric: double]	The maximum distance (X axis) to display on the chart. If not specified, the chart will auto-scale.
Chart minimum elevation (Y axis) Optional	`MINIMUM_ELEVATION`	[numeric: double]	The minimum elevation (Y axis) to display on the chart. If not specified, the chart will auto-scale.
Chart maximum elevation (Y axis) Optional	`MAXIMUM_ELEVATION`	[numeric: double]	The maximum elevation (Y axis) to display on the chart. If not specified, the chart will auto-scale.
Chart text color Optional	`TEXT_COLOR`	[color]	The color of the text in the chart (e.g., axis labels, titles).
Chart background color Optional	`BACKGROUND_COLOR`	[color]	The background color of the chart.
Chart border color Optional	`BORDER_COLOR`	[color]	The color of the chart border.
Profile tolerance	`TOLERANCE`	[numeric: double] Default: 5.0 Minimum value: 0	Defines how far a feature (vector point, line, polygon, or point cloud) can be from the profile line to be included in the results. It uses map units and does not affect other layer types.
Chart DPI	`DPI`	[numeric: integer] Default: 96 Minimum value: 0	The resolution of the output image in dots per inch (DPI).
Output image	`OUTPUT`	[file] Default: `[Save to temporary file]`	Specify the image file for the plot. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Output image	`OUTPUT`	[file]	The generated elevation profile image in the specified format.

Python code

Algorithm ID: `qgis:generateelevationprofileimage`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Mean and standard deviation plot

Creates a box plot with mean and standard deviation values.

Parameters

Label	Name	Type	Description
Input table	`INPUT`	[vector: geometry]	Input vector layer
Category name field	`NAME_FIELD`	[tablefield: any]	Categorical field to use for grouping the boxes (X axis)
Value field	`VALUE_FIELD`	[tablefield: numeric]	Value to use for the plot (Y axis).
Plot	`OUTPUT`	[html] Default: `[Save to temporary file]`	Specify the HTML file for the plot. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Plot	`OUTPUT`	[html]	HTML file with the plot. Available in the *Processing* ► *Result Viewer*.

Python code

Algorithm ID: `qgis:meanandstandarddeviationplot`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Polar plot

Generates a polar plot based on the value of an input vector layer.

Two fields must be entered as parameters: one that defines the category each feature (to group features) and another one with the variable to plot (this has to be a numeric one).

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Category name field	`NAME_FIELD`	[tablefield: any]	Categorical field to use for grouping the features (X axis)
Value field	`VALUE_FIELD`	[tablefield: numeric]	Value to use for the plot (Y axis).
Polar plot	`OUTPUT`	[html] Default: `[Save to temporary file]`	Specify the HTML file for the plot. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Polar plot	`OUTPUT`	[html]	HTML file with the plot. Available in the *Processing* ► *Result Viewer*.

Python code

Algorithm ID: `qgis:polarplot`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Raster layer histogram

Generates a histogram with the values of a raster layer.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Input raster layer
Band number	`BAND`	[raster band]	Raster band to use for the histogram
number of bins	`BINS`	[numeric: integer] Default: 10	The number of bins to use in the histogram (X axis). Minimum 2.
Histogram	`OUTPUT`	[html] Default: `[Save to temporary file]`	Specify the HTML file for the plot. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Histogram	`OUTPUT`	[html]	HTML file with the plot. Available in the *Processing* ► *Result Viewer*.

Python code

Algorithm ID: `qgis:rasterlayerhistogram`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Vector layer histogram

Generates a histogram with the values of the attribute of a vector layer.

The attribute to use for computing the histogram must be numeric.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	Input vector layer
Attribute	`FIELD`	[tablefield: numeric]	Value to use for the plot (Y axis).
number of bins	`BINS`	[numeric: integer] Default: 10	The number of bins to use in the histogram (X axis). Minimum 2.
Histogram	`OUTPUT`	[html] Default: `[Save to temporary file]`	Specify the HTML file for the plot. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Histogram	`OUTPUT`	[html]	HTML file with the plot. Available in the *Processing* ► *Result Viewer*.

Python code

Algorithm ID: `qgis:vectorlayerhistogram`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Vector layer scatterplot

Creates a simple `X` - `Y` scatter plot for a vector layer.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
X attribute	`XFIELD`	[tablefield: numeric]	Field to use for the X axis
Y attribute	`YFIELD`	[tablefield: numeric]	Field to use for the Y axis
Hover text Optional	`HOVERTEXT`	[expression] Default: “”	Text to be shown when hovering with the mouse over data points. It can be picked from a field or based on an expression.
Title Optional	`TITLE`	[string] Default: “”	Title of the plot
X-axis Title Optional	`XAXIS_TITLE`	[string] Default: “”	If empty, the field name of the x attribute is used. With a single space, the axis title is not shown.
Y-axis Title Optional	`YAXIS_TITLE`	[string] Default: “”	If empty, the field name of the y attribute is used. With a single space, the axis title is not shown.
Use logarithmic scale for x-axis	`XAXIS_LOG`	[boolean] Default: False	When enabled, uses logarithmic scale for the x-axis
Use logarithmic scale for y-axis	`YAXIS_LOG`	[boolean] Default: False	When enabled, uses logarithmic scale for the y-axis
Scatterplot	`OUTPUT`	[html] Default: `[Save to temporary file]`	Specify the HTML file for the plot. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Scatterplot	`OUTPUT`	[html]	HTML file with the plot. Available in the *Processing* ► *Result Viewer*.

Python code

Algorithm ID: `qgis:vectorlayerscatterplot`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Vector layer scatterplot 3D

Creates a 3D scatter plot for a vector layer.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
X attribute	`XFIELD`	[tablefield: numeric]	Field to use for the X axis
Y attribute	`YFIELD`	[tablefield: numeric]	Field to use for the Y axis
Z attribute	`ZFIELD`	[tablefield: numeric]	Field to use for the Z axis
Title Optional	`TITLE`	[string] Default: “”	Title of the plot
X-axis Title Optional	`XAXIS_TITLE`	[string] Default: “”	If empty, the field name of the X attribute is used.
Y-axis Title Optional	`YAXIS_TITLE`	[string] Default: “”	If empty, the field name of the Y attribute is used.
Z-axis Title Optional	`ZAXIS_TITLE`	[string] Default: “”	If empty, the field name of the Z attribute is used.
Scatterplot 3D	`OUTPUT`	[html] Default: `[Save to temporary file]`	Specify the HTML file for the plot. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Scatterplot 3D	`OUTPUT`	[html]	HTML file with the plot. Available in the *Processing* ► *Result Viewer*.

Python code

Algorithm ID: `qgis:scatter3dplot`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.15 Point Cloud Conversion

Note

These algorithms are only available if QGIS uses the PDAL library version 2.5.0 or newer.

Convert format

Converts a point cloud to a different file format, e.g. creates a compressed `.LAZ`.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[point cloud]	Input point cloud layer to convert
Converted	`OUTPUT`	[point cloud] Default: `[Save to temporary file]`	Specify the point cloud file to use as output. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Converted	`OUTPUT`	[point cloud]	Output point cloud layer in a modified file format. Currently supported formats are `.LAS`, `.LAZ`, `.COPC.LAZ` and `.VPC`.

Python code

Algorithm ID: `pdal:convertformat`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Export to raster

Exports point cloud data to a 2D raster grid having cell size of given resolution, writing values from the specified attribute.

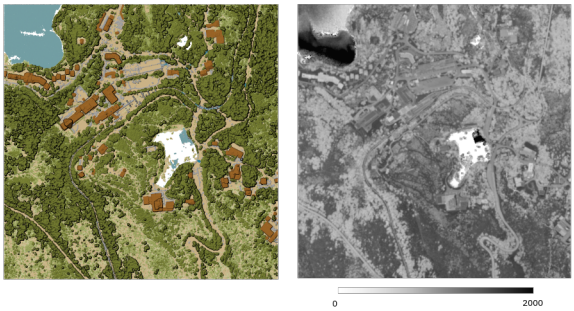


Fig. 24.24: Raster output using Intensity attribute of points

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[point cloud]	Input point cloud layer to export
Attribute	`ATTRIBUTE`	[field] [enumeration]	A Field of the point cloud layer to extract the values from
Resolution of the density raster	`RESOLUTION`	[numeric: double] Default: 1.0	Cell size of the output raster
Tile size for parallel runs	`TILE_SIZE`	[numeric: integer] Default: 1000	
Exported	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specify the raster file to export the data to. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Filter expression Optional	`FILTER_EXPRESSION`	[expression]	A *PDAL expression* for selecting a subset of features in the point cloud data
Cropping extent Optional	`FILTER_EXTENT`	[extent]	A map extent for selecting a subset of features in the point cloud data Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
X origin of a tile for parallel runs Optional	`ORIGIN_X`	[numeric: double]	
Y origin of a tile for parallel runs Optional	`ORIGIN_Y`	[numeric: double]	

Outputs

Label	Name	Type	Description
Exported	`OUTPUT`	[raster]	Output raster layer features of the point cloud layer are exported to. Currently supported format is `.TIF`.

Python code

Algorithm ID: `pdal:exportraster`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Export raster (using triangulation)

Exports point cloud data to a 2D raster grid using a triangulation of points and then interpolating cell values from triangles.

> **Note**
>
> Using this algorithm can be slower if you are dealing with a large dataset. If your point cloud is dense, you can export your ground points as a raster using the *Export to raster* algorithm.

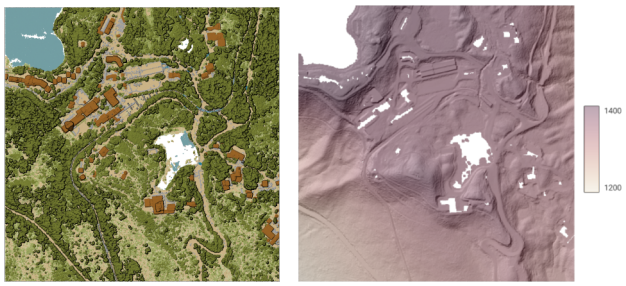


Fig. 24.25: Terrain raster output generated by point cloud triangulation

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[point cloud]	Input point cloud layer to export
Resolution of the density raster	`RESOLUTION`	[numeric: double] Default: 1.0	Cell size of the output raster
Tile size for parallel runs	`TILE_SIZE`	[numeric: integer] Default: 1000	
Exported	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specify the raster file to export the data to. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Filter expression Optional	`FILTER_EXPRESSION`	[expression]	A *PDAL expression* for selecting a subset of features in the point cloud data
Cropping extent Optional	`FILTER_EXTENT`	[extent]	A map extent for selecting a subset of features in the point cloud data Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
X origin of a tile for parallel runs Optional	`ORIGIN_X`	[numeric: double]	
Y origin of a tile for parallel runs Optional	`ORIGIN_Y`	[numeric: double]	

Outputs

Label	Name	Type	Description
Exported (using triangulation)	`OUTPUT`	[raster]	Output raster layer features of the point cloud layer are exported to. Currently supported format is `.TIF`.

Python code

Algorithm ID: `pdal:exportrastertin`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Export to vector

Exports point cloud data to a vector layer with 3D points (a GeoPackage), optionally with extra attributes.

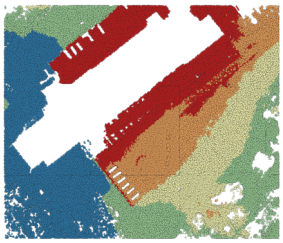


Fig. 24.26: Exporting point cloud (ground points) to a vector layer styled based on the elevation

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[point cloud]	Input point cloud layer to export
Attribute Optional	`ATTRIBUTE`	[field] [list]	One or more fields of the point cloud layer to export with the points.
Exported	`OUTPUT`	[vector] Default: `[Save to temporary file]`	Specify the vector file to export the data to. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Filter expression Optional	`FILTER_EXPRESSION`	[expression]	A *PDAL expression* for selecting a subset of features in the point cloud data
Cropping extent Optional	`FILTER_EXTENT`	[extent]	A map extent for selecting a subset of features in the point cloud data Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`

Outputs

Label	Name	Type	Description
Exported	`OUTPUT`	[vector]	Output vector layer features of the point cloud layer are exported to. Currently supported format is `.GPKG`.

Python code

Algorithm ID: `pdal:exportvector`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.16 Point Cloud Data Management

Note

These algorithms are only available if QGIS uses the PDAL library version 2.5.0 or newer.

Assign projection

Assigns a Coordinate Reference System to a point cloud layer, if it is missing or wrong. A new layer is created.

See also

Reproject

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[point cloud]	Input point cloud layer to assign a CRS to
Desired CRS	`CRS`	[crs]	The CRS to apply to the layer
Output layer	`OUTPUT`	[point cloud] Default: `[Save to temporary file]`	Specify the point cloud file to use as output. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Converted	`OUTPUT`	[point cloud]	Output point cloud layer with a new CRS. Currently supported formats are `.LAS`, `.LAZ`, `.COPC.LAZ` and `.VPC`.

Python code

Algorithm ID: `pdal:assignprojection`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Build virtual point cloud (VPC)

Creates a *virtual point cloud (VPC)* from input point cloud data.

If you leave the optional parameters unchecked, the VPC file will be built very quickly as the algorithm will only read metadata of input files. With any of the optional parameters set, the algorithm will read all points which can take some time.



Fig. 24.27: Generating Virtual Point Cloud with overview from a set of point cloud tiles

See also

Merge

Parameters

Label	Name	Type	Description
Input layers	`LAYERS`	[point cloud] [list]	Input point cloud layers to combine inside a virtual point cloud layer
Calculate boundary polygon	`BOUNDARY`	[boolean] Default: False	Set to True to show the exact boundaries of data (rather than just rectangular extent)
Calculate statistics	`STATISTICS`	[boolean] Default: False	Set to True to understand ranges of values of the various attributes
Build overview point cloud	`OVERVIEW`	[boolean] Default: False	Generates a single "thinned" point cloud of all the input data (using only every 1000th point from original data). The overview point cloud will be created next to the VPC file - for example, for `mydata.vpc`, the overview point cloud would be named `mydata-overview.copc.laz`.
Virtual point cloud	`OUTPUT`	[point cloud] Default: `[Save to temporary file]`	Specify the point cloud file to build the data into. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Virtual point cloud	`OUTPUT`	[raster]	Output point cloud layer combining all the input data, as a virtual file.

Python code

Algorithm ID: `pdal:virtualpointcloud`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Clip

Clips a point cloud layer by a polygon layer so that the resulting point cloud contains only points within the polygons.

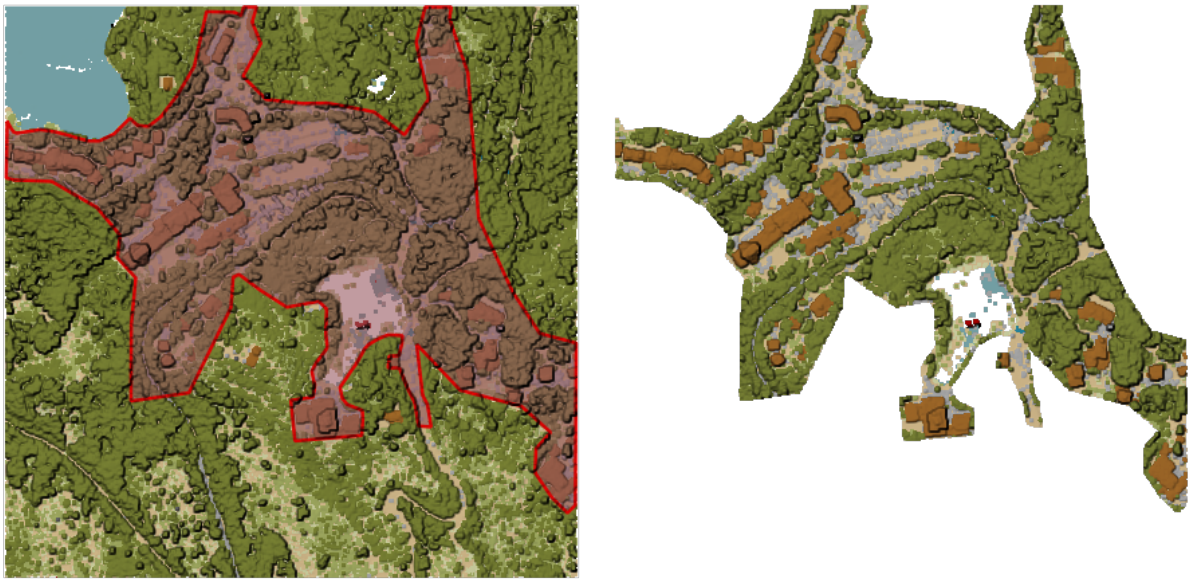


Fig. 24.28: Clipping an input point cloud layer with a polygon coverage

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[point cloud]	Input point cloud layer to clip
Clipping polygons	`OVERLAY`	[vector: polygon]	Polygon vector layer to use as coverage for clipping the points
Clipped	`OUTPUT`	[point cloud] Default: `[Save to temporary file]`	Specify the point cloud file to export the clipped points to. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Filter expression Optional	`FILTER_EXPRESSION`	[expression]	A *PDAL expression* for selecting a subset of features in the point cloud data
Cropping extent Optional	`FILTER_EXTENT`	[extent]	A map extent for selecting a subset of features in the point cloud data Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`

Outputs

Label	Name	Type	Description
Clipped	`OUTPUT`	[raster]	Output point cloud whose features are the points within the coverage polygon layer.

Python code

Algorithm ID: `pdal:clip`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Create COPC

Creates the index for all the input point cloud files in a batch mode.

Parameters

Label	Name	Type	Description
Input layers	`LAYERS`	[point cloud] [list]	Input point cloud layers to create an index for
Output directory Optional	`OUTPUT`	[folder] Default: `[Skip output]`	Specify the folder to create the new files in. *One of*: • Skip Output • Save to a Temporary Directory • Save to Directory

Outputs

Label	Name	Type	Description
Output directory	`OUTPUT`	[folder]	Output folder containing point cloud layers with accompanying COPC index files.

Python code

Algorithm ID: `pdal:createcopc`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Information

Outputs basic metadata from an input point cloud file.

Example of output information:

```
LAS           1.4
point format  6
count         56736130
scale         0.001 0.001 0.001
offset        431749.999 5440919.999 968.898
extent        431250 5440420 424.266
              432249.999 5441419.999 1513.531
crs           ETRS89 / UTM zone 34N (N-E) (EPSG:3046)  (vertical CRS missing!)
units         horizontal=metre  vertical=unknown

Attributes:
- X floating 8
- Y floating 8
```

(continues on next page)

(continued from previous page)

```
- Z floating 8
- Intensity unsigned 2
- ReturnNumber unsigned 1
- NumberOfReturns unsigned 1
- ScanDirectionFlag unsigned 1
- EdgeOfFlightLine unsigned 1
- Classification unsigned 1
- ScanAngleRank floating 4
- UserData unsigned 1
- PointSourceId unsigned 2
- GpsTime floating 8
- ScanChannel unsigned 1
- ClassFlags unsigned 1
```

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[point cloud]	Input point cloud layer to extract metadata information from
Layer information	`OUTPUT`	[file] Default: `[Save to temporary file]`	Specify the file to store the metadata information. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Layer information	`OUTPUT`	[vector]	`HTML` file to store the metadata information.

Python code

Algorithm ID: `pdal:info`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Merge

Merges multiple point cloud files into a single one.

See also

Build virtual point cloud (VPC)

Parameters

Basic parameters

Label	Name	Type	Description
Input layers	`LAYERS`	[point cloud] [list]	Input point cloud layers to merge into a single one
Merged	`OUTPUT`	[point cloud] Default: `[Save to temporary file]`	Specify the output point cloud merging input files. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Filter expression Optional	`FILTER_EXPRESSION`	[expression]	A *PDAL expression* for selecting a subset of features in the point cloud data
Cropping extent Optional	`FILTER_EXTENT`	[extent]	A map extent for selecting a subset of features in the point cloud data Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`

Outputs

Label	Name	Type	Description
Merged	`OUTPUT`	[point cloud]	Output point cloud layer merging all the input files.

Python code

Algorithm ID: `pdal:merge`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Reproject

Reprojects a point cloud to a different Coordinate Reference System (CRS).

See also

Assign projection

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[point cloud]	Input point cloud layer to reproject to a different CRS
Target CRS	`CRS`	[crs]	The CRS to apply to the layer
Reprojected	`OUTPUT`	[point cloud] Default: `[Save to temporary file]`	Specify the reprojected point cloud file. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Coordinate operation Optional	`OPERATION`	[datum]	The *datum transformation* to use to reproject the data between the origin and target systems.

Outputs

Label	Name	Type	Description
REPROJECTED	`OUTPUT`	[point cloud]	Output point cloud layer in the target CRS.

Python code

Algorithm ID: `pdal:reproject`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Thin (by sampling radius)

Creates a thinned version of the point cloud by performing sampling by distance point (reduces the number of points within a certain radius).

See also

Thin (by skipping points)

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[point cloud]	Input point cloud layer to create a thinned version from
Sampling radius (in map units)	`SAMPLING_RADIUS`	[numeric: double] Default: 1.0	Distance within which points are sampled to a unique point
Thinned (by radius)	`OUTPUT`	[point cloud] Default: `[Save to temporary file]`	Specify the output point cloud with reduced points. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Filter expression Optional	`FILTER_EXPRESSION`	[expression]	A *PDAL expression* for selecting a subset of features in the point cloud data
Cropping extent Optional	`FILTER_EXTENT`	[extent]	A map extent for selecting a subset of features in the point cloud data Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`

Outputs

Label	Name	Type	Description
Thinned (by radius)	`OUTPUT`	[point cloud]	Output point cloud layer with reduced points.

Python code

Algorithm ID: `pdal:thinbyradius`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Thin (by skipping points)

Creates a thinned version of the point cloud by keeping only every N-th point (reduces the number of points by skipping nearby points).

See also

Thin (by sampling radius)

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[point cloud]	Input point cloud layer to create a thinned version from
Number of points to skip	`POINTS_NUMBER`	[numeric: integer] Default: 1	Keep only every N-th point in the input layer
Thinned (by decimation)	`OUTPUT`	[point cloud] Default: `[Save to temporary file]`	Specify the output point cloud with reduced points. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Filter expression Optional	`FILTER_EXPRESSION`	[expression]	A *PDAL expression* for selecting a subset of features in the point cloud data
Cropping extent Optional	`FILTER_EXTENT`	[extent]	A map extent for selecting a subset of features in the point cloud data Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`

Outputs

Label	Name	Type	Description
Thinned (by decimation)	`OUTPUT`	[point cloud]	Output point cloud layer with reduced points.

Python code

Algorithm ID: `pdal:thinbydecimate`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Tile

Creates tiles from input point cloud files, recommended for best performance (in display or analysis) with such datasets in QGIS.

See also

Build virtual point cloud (VPC), *Create COPC*

Parameters

Basic parameters

Label	Name	Type	Description
Input layers	`LAYERS`	[point cloud] [list]	Input point cloud layers to create tiles from
Tile length	`LENGTH`	[numeric: double] Default: 1000.0	Size of the edge of each generated tile
Output directory	`OUTPUT`	[folder] Default: `[Save to temporary folder]`	Specify the folder to store the generated tiles. *One of*: • Save to a Temporary Directory • Save to Directory

Advanced parameters

Label	Name	Type	Description
Assign CRS Optional	`CRS`	[crs]	The CRS to apply to the layer

Outputs

Label	Name	Type	Description
Output directory	`OUTPUT`	[folder]	Output folder containing the tiles generated from input files.

Python code

Algorithm ID: `pdal:tile`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.17 Point Cloud Extraction

> **Note**
>
> These algorithms are only available if QGIS uses the PDAL library version 2.5.0 or newer.

Boundary

Exports a polygon file containing point cloud layer boundary. It may contain holes and it may be a multi-part polygon.

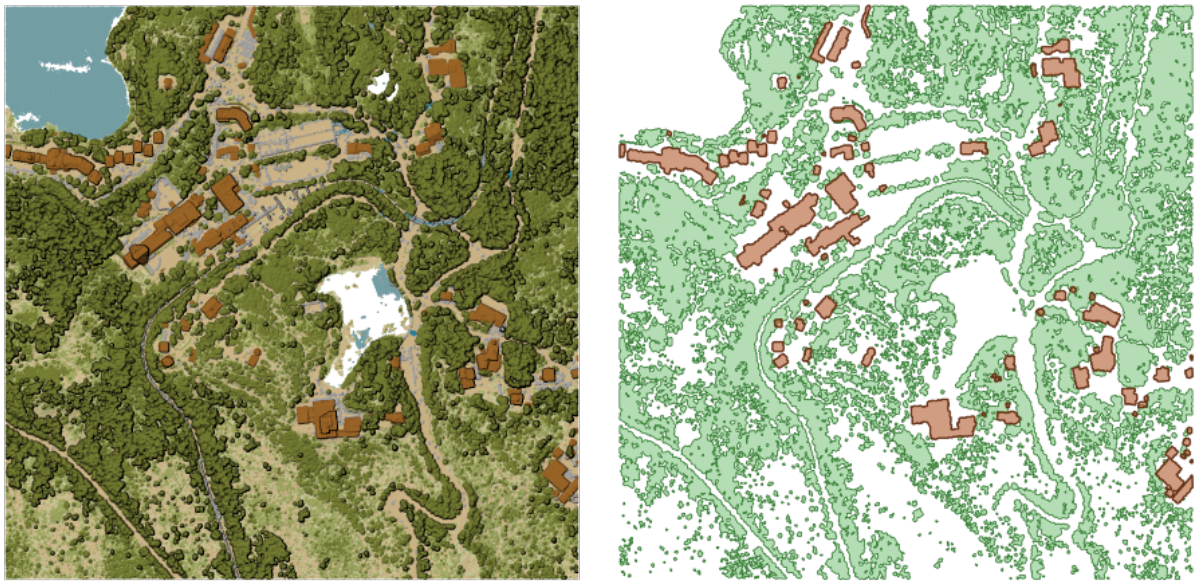


Fig. 24.29: Extracting high vegetation and building polygons from an input point cloud layer

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[point cloud]	Input point cloud layer to calculate boundary for
Resolution of cells used to calculate boundary Optional	`RESOLUTION`	[numeric: double]	Resolution of cells used to calculate boundary
Minimal number of points in a cell to consider cell occupied Optional	`THRESHOLD`	[numeric: integer]	Minimal number of points in a cell to consider cell occupied
Boundary	`OUTPUT`	[vector: polygon] Default: `[Save to temporary file]`	Specify the point cloud file to use as output. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Filter expression Optional	`FILTER_EXPRESSION`	[expression]	A *PDAL expression* for selecting a subset of features in the point cloud data
Cropping extent Optional	`FILTER_EXTENT`	[extent]	A map extent for selecting a subset of features in the point cloud data Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`

Outputs

Label	Name	Type	Description
Boundary	`OUTPUT`	[vector: polygon]	Output polygon vector layer with the point cloud boundary. Currently supported format is `.GPKG`.

Python code

Algorithm ID: `pdal:boundary`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Density

Exports a raster file based on the number of points within each raster cell - useful for quality checking of point cloud datasets.

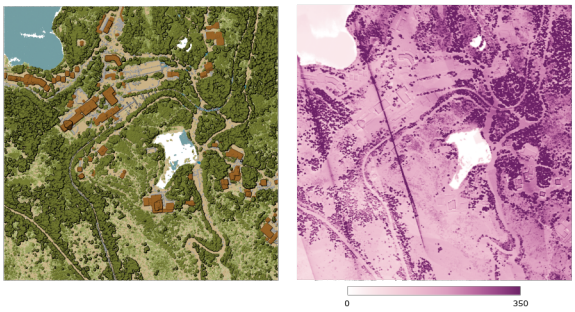


Fig. 24.30: Point density (number of points per 2x2 m) as a raster

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[point cloud]	Input point cloud layer to export
Resolution of the density raster	`RESOLUTION`	[numeric: double] Default: 1.0	Cell size of the output raster
Tile size for parallel runs	`TILE_SIZE`	[numeric: integer] Default: 1000	Size of the tiles to split the data into for parallel runs
Density	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specify the raster file to export the data to. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Filter expression Optional	`FILTER_EXPRESSION`	[expression]	A *PDAL expression* for selecting a subset of features in the point cloud data
Cropping extent Optional	`FILTER_EXTENT`	[extent]	A map extent for selecting a subset of features in the point cloud data Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
X origin of a tile for parallel runs Optional	`ORIGIN_X`	[numeric: double]	
Y origin of a tile for parallel runs Optional	`ORIGIN_Y`	[numeric: double]	

Outputs

Label	Name	Type	Description
Density	`OUTPUT`	[raster]	Output raster layer with number of points within each cell. Currently supported format is `.TIF`.

Python code

Algorithm ID: `pdal:density`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Filter

Extracts point from the input point cloud which match PDAL expression and/or are inside of a cropping rectangle.

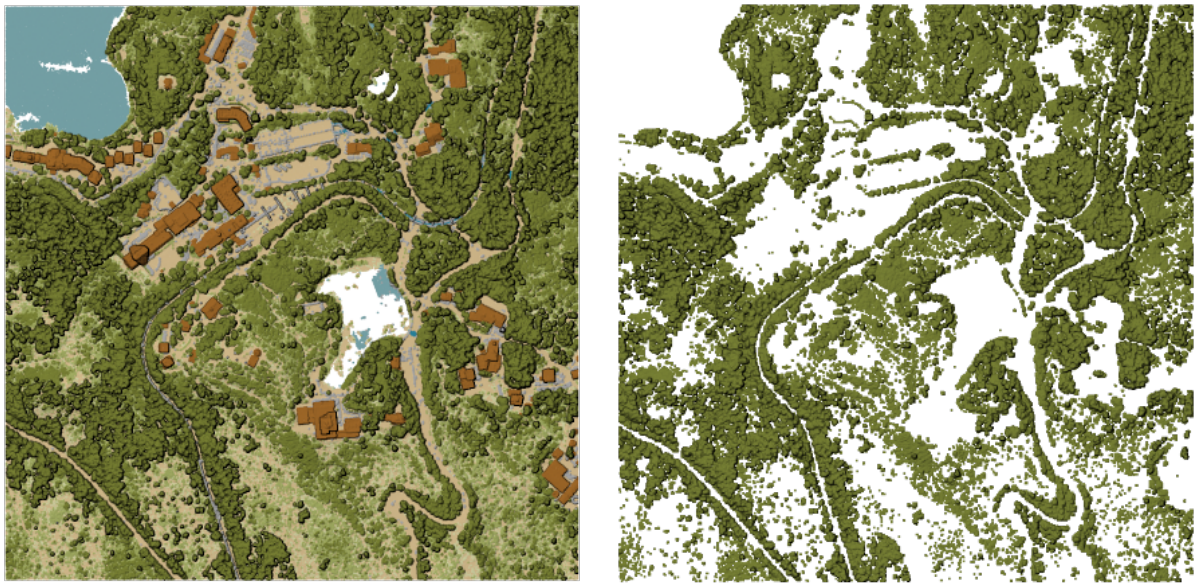


Fig. 24.31: Filtering of high vegetation class from an input point cloud layer

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[point cloud]	Input point cloud layer to export
Filter expression Optional	`FILTER_EXPRESSION`	[expression]	A *PDAL expression* for selecting a subset of features in the point cloud data
Cropping extent Optional	`FILTER_EXTENT`	[extent]	A map extent for selecting a subset of features in the point cloud data Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Filtered	`OUTPUT`	[point cloud] Default: `[Save to temporary file]`	Specify the point cloud file to export the data to. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Filtered	`OUTPUT`	[point cloud]	Output point cloud layer with the filtered features of the input point cloud layer. Currently supported formats are `.LAS`, `.LAZ`, `.COPC.LAZ` and `.VPC`.

Python code

Algorithm ID: `pdal:filter`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.18 Raster analysis

Cell stack percent rank from value

Calculates the cell-wise percentrank value of a stack of rasters based on a single input value and writes them to an output raster.

At each cell location, the specified value is ranked among the respective values in the stack of all overlaid and sorted cell values from the input rasters. For values outside of the stack value distribution, the algorithm returns NoData because the value cannot be ranked among the cell values.

There are two methods for percentile calculation:

- Inclusive linear interpolation (PERCENTRANK.INC)
- Exclusive linear interpolation (PERCENTRANK.EXC)

The linear interpolation methods return the unique percent rank for different values. Both interpolation methods follow their counterpart methods implemented by LibreOffice or Microsoft Excel.

The output raster's extent and resolution is defined by a reference raster. Input raster layers that do not match the cell size of the reference raster layer will be resampled using nearest neighbor resampling. NoData values in any of the input layers will result in a NoData cell output if the "Ignore NoData values" parameter is not set. The output raster data type will always be `Float32`.

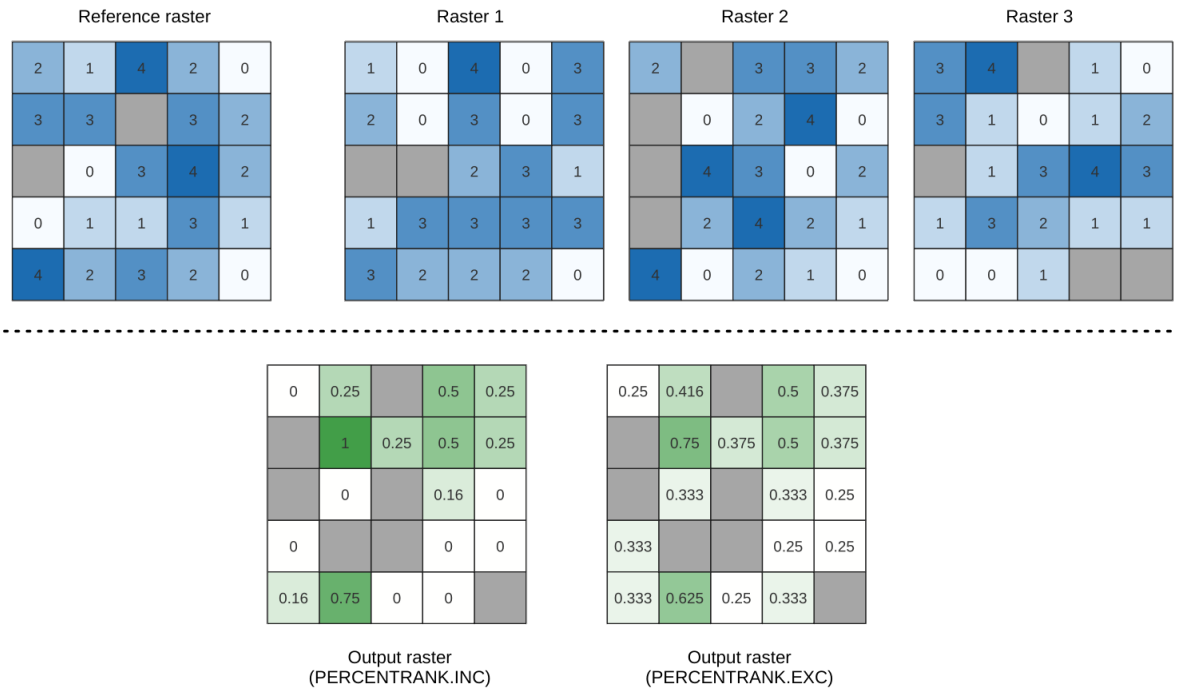


Fig. 24.32: Percent ranking Value = 1. `NoData` cells (grey) are ignored.

See also

Cell stack percentile, *Cell stack percentrank from raster layer*

Parameters

Basic parameters

Label	Name	Type	Description
Input layers	`INPUT`	[raster] [list]	Raster layers to evaluate. If multiband rasters are used in the data raster stack, the algorithm will always perform the analysis on the first band of the rasters
Method	`METHOD`	[enumeration] Default: 0	Method for percentile calculation: • 0 — Inclusive linear interpolation (PERCENTRANK.INC) • 1 — Exclusive linear interpolation (PERCENTRANK.EXC)
Value	`VALUE`	[numeric: double] Default: 10.0	Value to rank among the respective values in the stack of all overlaid and sorted cell values from the input rasters
Ignore NoData values	`IGNORE_NODATA`	[boolean] Default: True	If unchecked, any NoData cells in the input layers will result in a NoData cell in the output raster
Reference layer	`REFERENCE_LAYER`	[raster]	The reference layer for the output layer creation (extent, CRS, pixel dimensions)

continues on next page

Table 24.105 – continued from previous page

Label	Name	Type	Description
Output layer	`OUTPUT`	[same as input] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Output NoData value	`OUTPUT_NODATA_VAL`	[numeric: double] Default: -9999.0	Value to use for NoData in the output layer
Creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `CREATE_OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Output layer	`OUTPUT`	[raster]	Output raster layer containing the result
CRS authority identifier	`CRS_AUTHID`	[string]	The coordinate reference system of the output raster layer
Extent	`EXTENT`	[string]	The spatial extent of the output raster layer
Width in pixels	`WIDTH_IN_PIXEL`	[numeric: integer]	The number of columns in the output raster layer
Height in pixels	`HEIGHT_IN_PIXE`	[numeric: integer]	The number of rows in the output raster layer
Total pixel count	`TOTAL_PIXEL_COUN`	[numeric: integer]	The count of pixels in the output raster layer

Python code

Algorithm ID: `native:cellstackpercentrankfromvalue`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Cell stack percentile

Calculates the cell-wise percentile value of a stack of rasters and writes the results to an output raster. The percentile to return is determined by the percentile input value (ranges between 0 and 1). At each cell location, the specified percentile is obtained using the respective value from the stack of all overlaid and sorted cell values of the input rasters.

There are three methods for percentile calculation:

- Nearest rank: returns the value that is nearest to the specified percentile
- Inclusive linear interpolation (PERCENTRANK.INC)
- Exclusive linear interpolation (PERCENTRANK.EXC)

The linear interpolation methods return the unique values for different percentiles. Both interpolation methods follow their counterpart methods implemented by LibreOffice or Microsoft Excel.

The output raster's extent and resolution is defined by a reference raster. Input raster layers that do not match the cell size of the reference raster layer will be resampled using nearest neighbor resampling. NoData values in any of the input layers will result in a NoData cell output if the "Ignore NoData values" parameter is not set. The output raster data type will always be `Float32`.

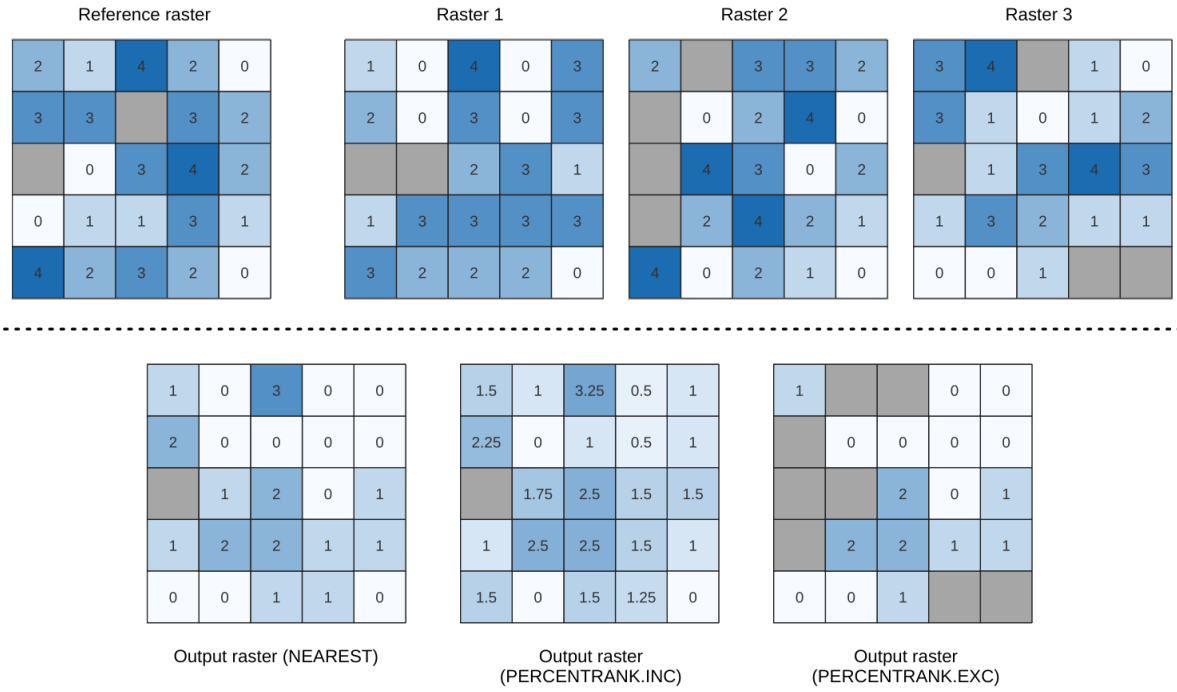


Fig. 24.33: Percentile = 0.25. `NoData` cells (grey) are ignored.

See also

Cell stack percentile, *Cell stack percentrank from raster layer*

Parameters

Basic parameters

Label	Name	Type	Description
Input layers	`INPUT`	[raster] [list]	Raster layers to evaluate. If multiband rasters are used in the data raster stack, the algorithm will always perform the analysis on the first band of the rasters
Method	`METHOD`	[enumeration] Default: 0	Method for percentile calculation: • 0 — Nearest rank: returns the value that is nearest to the specified percentile • 1 — Inclusive linear interpolation (PERCENTILE.INC) • 2 — Exclusive linear interpolation (PERCENTILE.EXC)
Percentile	`VALUE`	[numeric: double] Default: 0.25	Value to rank among the respective values in the stack of all overlaid and sorted cell values from the input rasters. Between 0 and 1.
Ignore NoData values	`IGNORE_NODATA`	[boolean] Default: True	If unchecked, any NoData cells in the input layers will result in a NoData cell in the output raster
Reference layer	`REFERENCE_LAYER`	[raster]	The reference layer for the output layer creation (extent, CRS, pixel dimensions)
Output layer	`OUTPUT`	[same as input] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Output NoData value	`OUTPUT_NODATA_VAL`	[numeric: double] Default: -9999.0	Value to use for NoData in the output layer
Creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `CREATE_OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (`|`).

Outputs

Label	Name	Type	Description
Output layer	`OUTPUT`	[raster]	Output raster layer containing the result
CRS authority identifier	`CRS_AUTHID`	[string]	The coordinate reference system of the output raster layer
Extent	`EXTENT`	[string]	The spatial extent of the output raster layer
Width in pixels	`WIDTH_IN_PIXEL`	[numeric: integer]	The number of columns in the output raster layer
Height in pixels	`HEIGHT_IN_PIXE`	[numeric: integer]	The number of rows in the output raster layer
Total pixel count	`TOTAL_PIXEL_COUN`	[numeric: integer]	The count of pixels in the output raster layer

Python code

Algorithm ID: `native:cellstackpercentile`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Cell stack percentrank from raster layer

Calculates the cell-wise percentrank value of a stack of rasters based on an input value raster and writes them to an output raster.

At each cell location, the current value of the value raster is ranked among the respective values in the stack of all overlaid and sorted cell values of the input rasters. For values outside of the the stack value distribution, the algorithm returns NoData because the value cannot be ranked among the cell values.

There are two methods for percentile calculation:

- Inclusive linear interpolation (PERCENTRANK.INC)
- Exclusive linear interpolation (PERCENTRANK.EXC)

The linear interpolation methods return the unique values for different percentiles. Both interpolation methods follow their counterpart methods implemented by LibreOffice or Microsoft Excel.

The output raster's extent and resolution is defined by a reference raster. Input raster layers that do not match the cell size of the reference raster layer will be resampled using nearest neighbor resampling. NoData values in any of the input layers will result in a NoData cell output if the "Ignore NoData values" parameter is not set. The output raster data type will always be `Float32`.

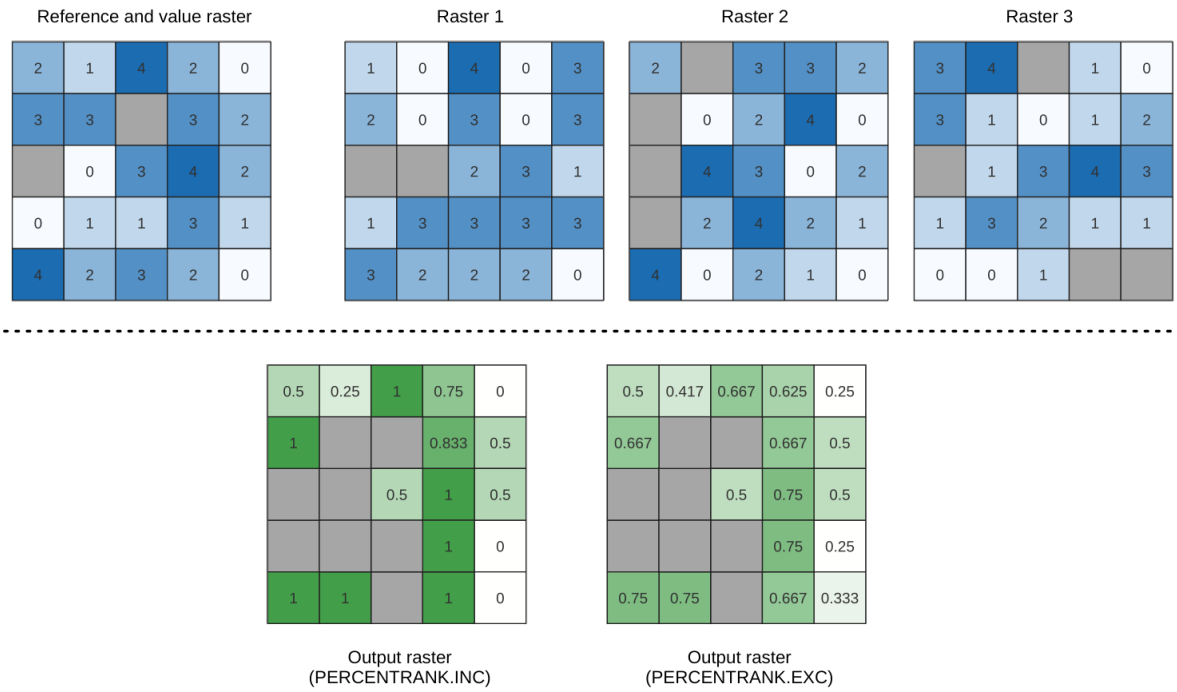


Fig. 24.34: Ranking the value raster layer cells. `NoData` cells (grey) are ignored.

See also

Cell stack percentile, *Cell stack percent rank from value*

Parameters

Basic parameters

Label	Name	Type	Description
Input layers	`INPUT`	[raster] [list]	Raster layers to evaluate. If multiband rasters are used in the data raster stack, the algorithm will always perform the analysis on the first band of the rasters
Value raster layer	`INPUT_VALUE_RAST`	[raster]	The layer to rank the values among the stack of all overlaid layers
Value raster band	`VALUE_RASTER_B`	[numeric: integer] Default: 1	Band of the "value raster layer" to compare to
Method	`METHOD`	[enumeration] Default: 0	Method for percentile calculation: • 0 — Inclusive linear interpolation (PERCENTRANK.INC) • 1 — Exclusive linear interpolation (PERCENTRANK.EXC)
Ignore NoData values	`IGNORE_NODATA`	[boolean] Default: True	If unchecked, any NoData cells in the input layers will result in a NoData cell in the output raster
Reference layer	`REFERENCE_LAYER`	[raster]	The reference layer for the output layer creation (extent, CRS, pixel dimensions)

continues on next page

Table 24.109 – continued from previous page

Label	Name	Type	Description
Output layer	`OUTPUT`	[same as input] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Output NoData value	`OUTPUT_NODATA_VAL`	[numeric: double] Default: -9999.0	Value to use for NoData in the output layer
Creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `CREATE_OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Output layer	`OUTPUT`	[raster]	Output raster layer containing the result
CRS authority identifier	`CRS_AUTHID`	[string]	The coordinate reference system of the output raster layer
Extent	`EXTENT`	[string]	The spatial extent of the output raster layer
Width in pixels	`WIDTH_IN_PIXEL`	[numeric: integer]	The number of columns in the output raster layer
Height in pixels	`HEIGHT_IN_PIXE`	[numeric: integer]	The number of rows in the output raster layer
Total pixel count	`TOTAL_PIXEL_COUN`	[numeric: integer]	The count of pixels in the output raster layer

Python code

Algorithm ID: `native:cellstackpercentrankfromrasterlayer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Cell statistics

Computes per-cell statistics based on input raster layers and for each cell writes the resulting statistics to an output raster. At each cell location, the output value is defined as a function of all overlaid cell values of the input rasters.

By default, a NoData cell in ANY of the input layers will result in a NoData cell in the output raster. If the *Ignore NoData values* option is checked, then NoData inputs will be ignored in the statistic calculation. This may result in NoData output for locations where all cells are NoData.

The *Reference layer* parameter specifies an existing raster layer to use as a reference when creating the output raster. The output raster will have the same extent, CRS, and pixel dimensions as this layer.

Calculation details: Input raster layers that do not match the cell size of the reference raster layer will be resampled using `nearest neighbor resampling`. The output raster data type will be set to the most complex data type present in the input datasets except when using the functions `Mean`, `Standard deviation` and `Variance` (data type is always `Float32` or `Float64` depending on input float type) or `Count` and `Variety` (data type is always `Int32`).

- `Count`: The count statistic will always result in the number of cells without NoData values at the current cell location.
- `Median`: If the number of input layers is even, the median will be calculated as the arithmetic mean of the two middle values of the ordered cell input values.
- `Minority/Majority`: If no unique minority or majority could be found, the result is NoData, except all input cell values are equal.

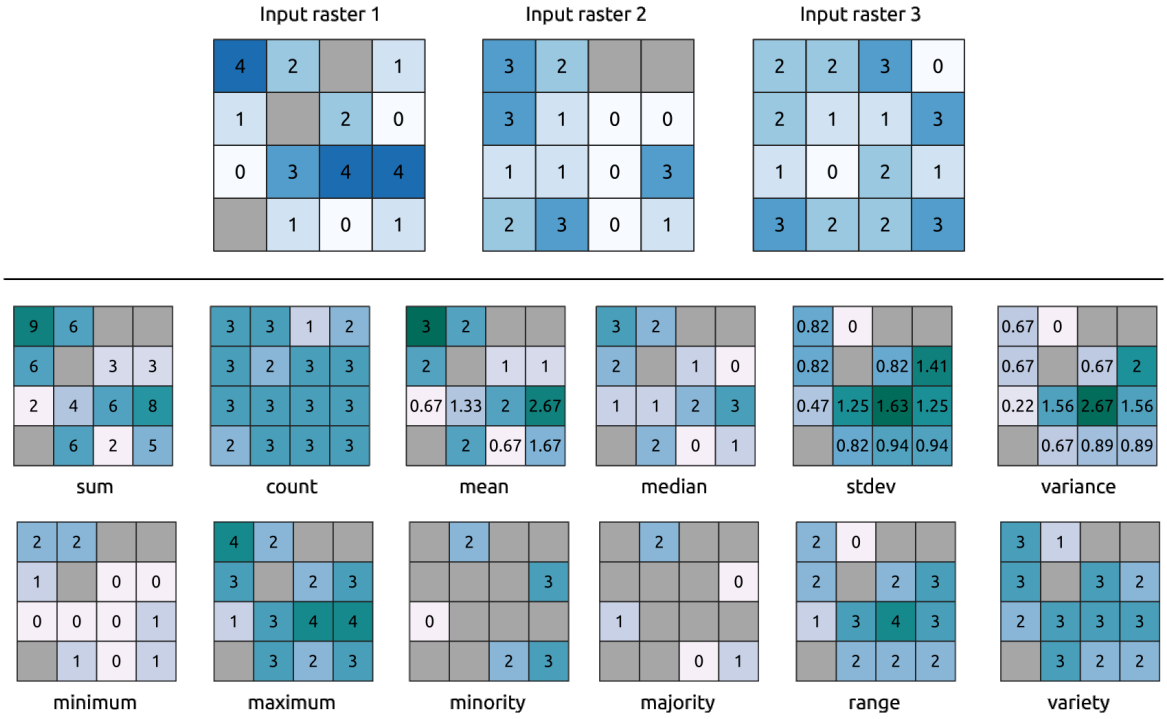


Fig. 24.35: Example with all the statistic functions. `NoData` cells (grey) are taken into account.

Parameters

Basic parameters

Label	Name	Type	Description
Input layers	`INPUT`	[raster] [list]	Input raster layers
Statistic	`STATISTIC`	[enumeration] Default: 0	Available statistics. Options: • 0 — Sum • 1 — Count • 2 — Mean • 3 — Median • 4 — Standard deviation • 5 — Variance • 6 — Minimum • 7 — Maximum • 8 — Minority (least common value) • 9 — Majority (most common value) • 10 — Range (max - min) • 11 — Variety (unique value count)
Ignore NoData values	`IGNORE_NODATA`	[boolean] Default: True	Calculate statistics also for all cells stacks, ignoring NoData occurrence.
Reference layer	`REF_LAYER`	[raster]	The reference layer to create the output layer from (extent, CRS, pixel dimensions)
Output layer	`OUTPUT`	[same as input] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Output NoData value Optional	`OUTPUT_NO_DATA_VA`	[numeric: double] Default: -9999.0	Value to use for NoData in the output layer
Creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `CREATE_OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
CRS authority identifier	`CRS_AUTHID`	[crs]	The coordinate reference system of the output raster layer
Extent	`EXTENT`	[string]	The spatial extent of the output raster layer
Height in pixels	`HEIGHT_IN_PIXE`	[numeric: integer]	The number of rows in the output raster layer
Output raster	`OUTPUT`	[raster]	Output raster layer containing the result
Total pixel count	`TOTAL_PIXEL_COUN`	[numeric: integer]	The count of pixels in the output raster layer
Width in pixels	`WIDTH_IN_PIXEL`	[numeric: integer]	The number of columns in the output raster layer

Python code

Algorithm ID: `native:cellstatistics`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Equal to frequency

Evaluates on a cell-by-cell basis the frequency (number of times) the values of an input stack of rasters are equal to the value of a value layer. The output raster extent and resolution are defined by the input raster layer and is always of `Int32` type.

If multiband rasters are used in the data raster stack, the algorithm will always perform the analysis on the first band of the rasters - use GDAL to use other bands in the analysis. The output NoData value can be set manually.

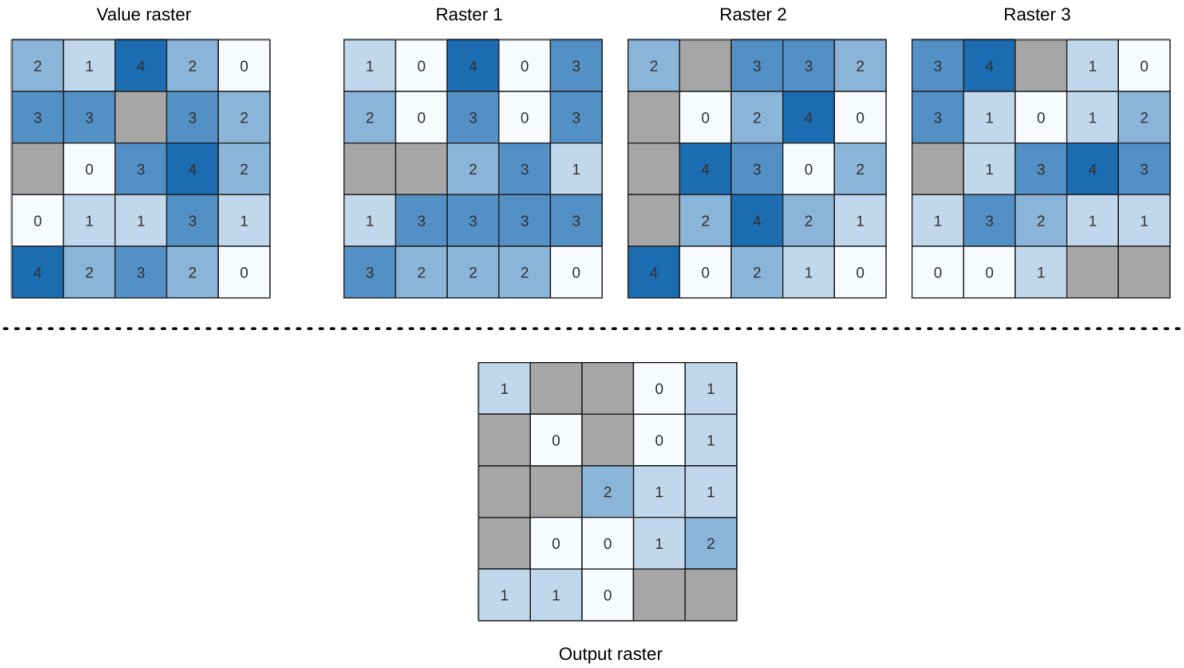


Fig. 24.36: For each cell in the output raster, the value represents the number of times that the corresponding cells in the list of rasters are the same as the value raster. `NoData` cells (grey) are taken into account.

See also

Greater than frequency, *Less than frequency*

Parameters

Basic parameters

Label	Name	Type	Description
Input value raster	`INPUT_VALUE_RAST`	[raster]	The input value layer serves as reference layer for the sample layers
Value raster band	`INPUT_VALUE_RAST`	[raster band] Default: The first band of the raster layer	Select the band you want to use as sample
Input raster layers	`INPUT_RASTERS`	[raster] [list]	Raster layers to evaluate. If multiband rasters are used in the data raster stack, the algorithm will always perform the analysis on the first band of the rasters
Ignore NoData values	`IGNORE_NODATA`	[boolean] Default: False	If unchecked, any NoData cells in the value raster or the data layer stack will result in a NoData cell in the output raster
Output layer	`OUTPUT`	[same as input] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Output NoData value Optional	OUTPUT_NO_DATA_VA	[numeric: double] Default: -9999.0	Value to use for NoData in the output layer
Creation options Optional	CREATION_OPTIONS (for QGIS <= 3.42, this was CREATE_OPTIONS)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Output layer	OUTPUT	[raster]	Output raster layer containing the result
CRS authority identifier	CRS_AUTHID	[string]	The coordinate reference system of the output raster layer
Extent	EXTENT	[string]	The spatial extent of the output raster layer
Count of cells with equal value occurrences	FOUND_LOCATION	[numeric: integer]	
Height in pixels	HEIGHT_IN_PIXE	[numeric: integer]	The number of rows in the output raster layer
Total pixel count	TOTAL_PIXEL_COUN	[numeric: integer]	The count of pixels in the output raster layer
Mean frequency at valid cell locations	MEAN_FREQUENCY	[numeric: double]	
Count of value occurrences	OCCURRENCE_COUNT	[numeric: integer]	
Width in pixels	WIDTH_IN_PIXEL	[numeric: integer]	The number of columns in the output raster layer

Python code

Algorithm ID: `native:equaltofrequency`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Fuzzify raster (gaussian membership)

Transforms an input raster to a fuzzified raster by assigning a membership value to each pixel, using a Gaussian membership function. Membership values range from 0 to 1. In the fuzzified raster, a value of 0 implies no membership of the defined fuzzy set, whereas a value of 1 means full membership. The gaussian membership function is defined as $\mu(x) = e^{-f1*(x-f2)^2}$, where *f1* is the spread and *f2* the midpoint.

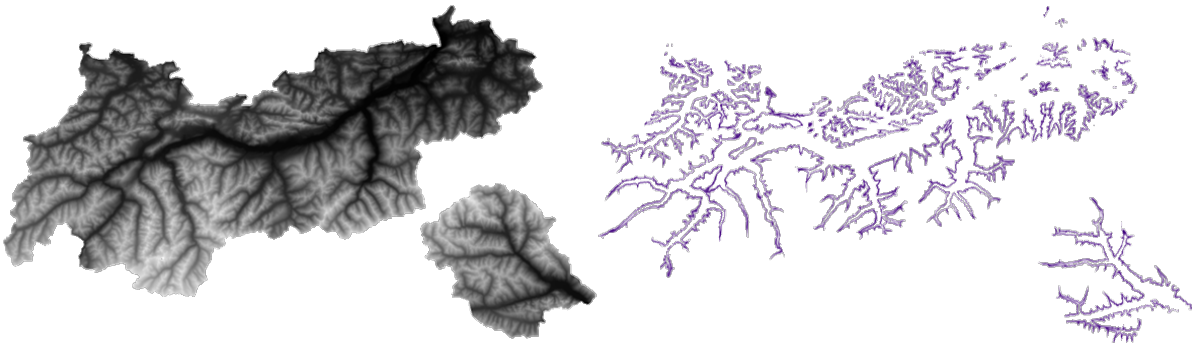


Fig. 24.37: Fuzzify raster example. Input raster source: Land Tirol - data.tirol.gv.at.

See also

Fuzzify raster (large membership), *Fuzzify raster (linear membership)*, *Fuzzify raster (near membership)*, *Fuzzify raster (power membership)*, *Fuzzify raster (small membership)*

Parameters

Basic parameters

Label	Name	Type	Description
Input Raster	`INPUT`	[raster]	Input raster layer
Band Number	`BAND`	[raster band] Default: The first band of the raster layer	If the raster is multiband, choose the band that you want to fuzzify.
Function midpoint	`FUZZYMIDPOINT`	[numeric: double] Default: 10.0	Midpoint of the gaussian function
Function spread	`FUZZYSPREAD`	[numeric: double] Default: 0.01	Spread of the gaussian function
Fuzzified raster	`OUTPUT`	[same as input] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Creation options Optional	CREATION_OPTIONS (for QGIS <= 3.42, this was CREATE_OPTIONS)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Fuzzified raster	OUTPUT	[same as input]	Output raster layer containing the result
CRS authority identifier	CRS_AUTHID	[crs]	The coordinate reference system of the output raster layer
Extent	EXTENT	[string]	The spatial extent of the output raster layer
Width in pixels	WIDTH_IN_PIXEL	[numeric: integer]	The number of columns in the output raster layer
Height in pixels	HEIGHT_IN_PIXE	[numeric: integer]	The number of rows in the output raster layer
Total pixel count	TOTAL_PIXEL_COUN	[numeric: integer]	The count of pixels in the output raster layer

Python code

Algorithm ID: `native:fuzzifyrastergaussianmembership`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Fuzzify raster (large membership)

Transforms an input raster to a fuzzified raster by assigning a membership value to each pixel, using a Large membership function. Membership values range from 0 to 1. In the fuzzified raster, a value of 0 implies no membership of the defined fuzzy set, whereas a value of 1 means full membership. The large membership function is defined as $\mu(x) = \frac{1}{1+\left(\frac{x}{f2}\right)^{-f1}}$, where $f1$ is the spread and $f2$ the midpoint.

See also

Fuzzify raster (gaussian membership), *Fuzzify raster (linear membership)*, *Fuzzify raster (near membership)*, *Fuzzify raster (power membership)*, *Fuzzify raster (small membership)*

Parameters

Basic parameters

Label	Name	Type	Description
Input Raster	`INPUT`	[raster]	Input raster layer
Band Number	`BAND`	[raster band] Default: The first band of the raster layer	If the raster is multiband, choose the band that you want to fuzzify.
Function midpoint	`FUZZYMIDPOINT`	[numeric: double] Default: 50.0	Midpoint of the large function
Function spread	`FUZZYSPREAD`	[numeric: double] Default: 5.0	Spread of the large function
Fuzzified raster	`OUTPUT`	[same as input] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `CREATE_OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (`|`).

Outputs

Label	Name	Type	Description
Fuzzified raster	`OUTPUT`	[same as input]	Output raster layer containing the result
CRS authority identifier	`CRS_AUTHID`	[crs]	The coordinate reference system of the output raster layer
Extent	`EXTENT`	[string]	The spatial extent of the output raster layer
Width in pixels	`WIDTH_IN_PIXEL`	[numeric: integer]	The number of columns in the output raster layer
Height in pixels	`HEIGHT_IN_PIXE`	[numeric: integer]	The number of rows in the output raster layer
Total pixel count	`TOTAL_PIXEL_COUN`	[numeric: integer]	The count of pixels in the output raster layer

Python code

Algorithm ID: `native:fuzzifyrasterlargemembership`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Fuzzify raster (linear membership)

Transforms an input raster to a fuzzified raster by assigning a membership value to each pixel, using a Linear membership function. Membership values range from 0 to 1. In the fuzzified raster, a value of 0 implies no membership of the defined fuzzy set, whereas a value of 1 means full membership. The linear function is defined as $\mu(X)\begin{cases}0 & x \leq a \\ \frac{x-a}{b-a} & a < x < b \\ 1 & x \geq b\end{cases}$, where a is the low bound and b the high bound. This equation assigns membership values using a linear transformation for pixel values between the low and high bounds. Pixels values smaller than the low bound are given 0 membership whereas pixel values greater than the high bound are given 1 membership.

See also

Fuzzify raster (gaussian membership), *Fuzzify raster (large membership)*, *Fuzzify raster (near membership)*, *Fuzzify raster (power membership)*, *Fuzzify raster (small membership)*

Parameters

Basic parameters

Label	Name	Type	Description
Input Raster	`INPUT`	[raster]	Input raster layer
Band Number	`BAND`	[raster band] Default: The first band of the raster layer	If the raster is multiband, choose the band that you want to fuzzify.
Low fuzzy membership bound	`FUZZYLOWBOUND`	[numeric: double] Default: 0.0	Low bound of the linear function
High fuzzy membership bound	`FUZZYHIGHBOUND`	[numeric: double] Default: 1.0	High bound of the linear function
Fuzzified raster	`OUTPUT`	[same as input] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Creation options Optional	CREATION_OPTIONS (for QGIS <= 3.42, this was CREATE_OPTIONS)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Fuzzified raster	OUTPUT	[same as input]	Output raster layer containing the result
CRS authority identifier	CRS_AUTHID	[crs]	The coordinate reference system of the output raster layer
Extent	EXTENT	[string]	The spatial extent of the output raster layer
Width in pixels	WIDTH_IN_PIXEL	[numeric: integer]	The number of columns in the output raster layer
Height in pixels	HEIGHT_IN_PIXE	[numeric: integer]	The number of rows in the output raster layer
Total pixel count	TOTAL_PIXEL_COUN	[numeric: integer]	The count of pixels in the output raster layer

Python code

Algorithm ID: `native:fuzzifyrasterlinearmembership`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Fuzzify raster (near membership)

Transforms an input raster to a fuzzified raster by assigning a membership value to each pixel, using a Near membership function. Membership values range from 0 to 1. In the fuzzified raster, a value of 0 implies no membership of the defined fuzzy set, whereas a value of 1 means full membership. The near membership function is defined as $\mu(x) = \frac{1}{1 + f1 * (x - f2)^2}$, where *f1* is the spread and *f2* the midpoint.

See also

Fuzzify raster (gaussian membership), *Fuzzify raster (large membership)*, *Fuzzify raster (linear membership)*, *Fuzzify raster (power membership)*, *Fuzzify raster (small membership)*

Parameters

Basic parameters

Label	Name	Type	Description
Input Raster	`INPUT`	[raster]	Input raster layer
Band Number	`BAND`	[raster band] Default: The first band of the raster layer	If the raster is multiband, choose the band that you want to fuzzify.
Function midpoint	`FUZZYMIDPOINT`	[numeric: double] Default: 50.0	Midpoint of the near function
Function spread	`FUZZYSPREAD`	[numeric: double] Default: 0.01	Spread of the near function
Fuzzified raster	`OUTPUT`	[same as input] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `CREATE_OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Fuzzified raster	`OUTPUT`	[same as input]	Output raster layer containing the result
CRS authority identifier	`CRS_AUTHID`	[crs]	The coordinate reference system of the output raster layer
Extent	`EXTENT`	[string]	The spatial extent of the output raster layer
Width in pixels	`WIDTH_IN_PIXEL`	[numeric: integer]	The number of columns in the output raster layer
Height in pixels	`HEIGHT_IN_PIXE`	[numeric: integer]	The number of rows in the output raster layer
Total pixel count	`TOTAL_PIXEL_COUN`	[numeric: integer]	The count of pixels in the output raster layer

Python code

Algorithm ID: `native:fuzzifyrasternearmembership`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Fuzzify raster (power membership)

Transforms an input raster to a fuzzified raster by assigning a membership value to each pixel, using a Power membership function. Membership values range from 0 to 1. In the fuzzified raster, a value of 0 implies no membership of the defined fuzzy set, whereas a value of 1 means full membership. The power function is defined as $\mu(x)\begin{cases}0 & x \leq a \\ \left(\frac{x-a}{b-a}\right)^{f1} & a < x < b \\ 1 & x \geq b\end{cases}$, where a is the low bound, b is the high bound, and $f1$ the exponent. This equation assigns membership values using the power transformation for pixel values between the low and high bounds. Pixels values smaller than the low bound are given 0 membership whereas pixel values greater than the high bound are given 1 membership.

See also

Fuzzify raster (gaussian membership), *Fuzzify raster (large membership)*, *Fuzzify raster (linear membership)*, *Fuzzify raster (near membership)*, *Fuzzify raster (small membership)*

Parameters

Basic parameters

Label	Name	Type	Description
Input Raster	`INPUT`	[raster]	Input raster layer
Band Number	`BAND`	[raster band] Default: The first band of the raster layer	If the raster is multiband, choose the band that you want to fuzzify.
Low fuzzy membership bound	`FUZZYLOWBOUND`	[numeric: double] Default: 0.0	Low bound of the power function
High fuzzy membership bound	`FUZZYHIGHBOUND`	[numeric: double] Default: 1.0	High bound of the power function
High fuzzy membership bound	`FUZZYEXPONENT`	[numeric: double] Default: 2.0	Exponent of the power function
Fuzzified raster	`OUTPUT`	[same as input] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Creation options Optional	CREATION_OPTIONS (for QGIS <= 3.42, this was CREATE_OPTIONS)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Fuzzified raster	OUTPUT	[same as input]	Output raster layer containing the result
CRS authority identifier	CRS_AUTHID	[crs]	The coordinate reference system of the output raster layer
Extent	EXTENT	[string]	The spatial extent of the output raster layer
Width in pixels	WIDTH_IN_PIXEL	[numeric: integer]	The number of columns in the output raster layer
Height in pixels	HEIGHT_IN_PIXE	[numeric: integer]	The number of rows in the output raster layer
Total pixel count	TOTAL_PIXEL_COUN	[numeric: integer]	The count of pixels in the output raster layer

Python code

Algorithm ID: `native:fuzzifyrasterpowermembership`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Fuzzify raster (small membership)

Transforms an input raster to a fuzzified raster by assigning a membership value to each pixel, using a Small membership function. Membership values range from 0 to 1. In the fuzzified raster, a value of 0 implies no membership of the defined fuzzy set, whereas a value of 1 means full membership. The small membership function is defined as $\mu(x) = \frac{1}{1 + \left(\frac{x}{f2}\right)^{f1}}$, where *f1* is the spread and *f2* the midpoint.

See also

Fuzzify raster (gaussian membership), *Fuzzify raster (large membership)* *Fuzzify raster (linear membership)*, *Fuzzify raster (near membership)*, *Fuzzify raster (power membership)*

Parameters

Basic parameters

Label	Name	Type	Description
Input Raster	`INPUT`	[raster]	Input raster layer
Band Number	`BAND`	[raster band] Default: The first band of the raster layer	If the raster is multiband, choose the band that you want to fuzzify.
Function midpoint	`FUZZYMIDPOINT`	[numeric: double] Default: 50.0	Midpoint of the small function
Function spread	`FUZZYSPREAD`	[numeric: double] Default: 5.0	Spread of the small function
Fuzzified raster	`OUTPUT`	[same as input] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `CREATE_OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (`|`).

Outputs

Label	Name	Type	Description
Fuzzified raster	`OUTPUT`	[same as input]	Output raster layer containing the result
CRS authority identifier	`CRS_AUTHID`	[crs]	The coordinate reference system of the output raster layer
Extent	`EXTENT`	[string]	The spatial extent of the output raster layer
Width in pixels	`WIDTH_IN_PIXEL`	[numeric: integer]	The number of columns in the output raster layer
Height in pixels	`HEIGHT_IN_PIXE`	[numeric: integer]	The number of rows in the output raster layer
Total pixel count	`TOTAL_PIXEL_COUN`	[numeric: integer]	The count of pixels in the output raster layer

Python code

Algorithm ID: `native:fuzzifyrastersmallmembership`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Greater than frequency

Evaluates on a cell-by-cell basis the frequency (number of times) the values of an input stack of rasters are equal to the value of a value raster. The output raster extent and resolution is defined by the input raster layer and is always of `Int32` type.

If multiband rasters are used in the data raster stack, the algorithm will always perform the analysis on the first band of the rasters - use GDAL to use other bands in the analysis. The output NoData value can be set manually.

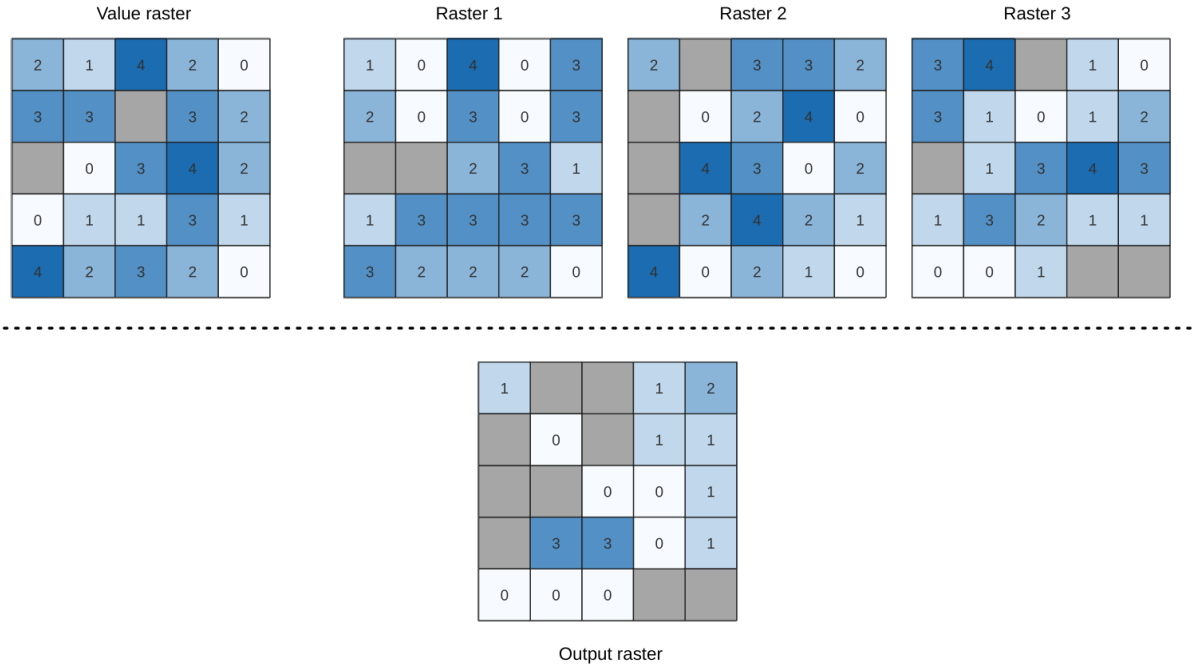


Fig. 24.38: For each cell in the output raster, the value represents the number of times that the corresponding cells in the list of rasters are greater than the value raster. `NoData` cells (grey) are taken into account.

See also

Equal to frequency, *Less than frequency*

Parameters

Basic parameters

Label	Name	Type	Description
Input value raster	`INPUT_VALUE_RAST`	[raster]	The input value layer serves as reference layer for the sample layers
Value raster band	`INPUT_VALUE_RAST`	[raster band] Default: The first band of the raster layer	Select the band you want to use as sample
Input raster layers	`INPUT_RASTERS`	[raster] [list]	Raster layers to evaluate. If multiband rasters are used in the data raster stack, the algorithm will always perform the analysis on the first band of the rasters
Ignore NoData values	`IGNORE_NODATA`	[boolean] Default: False	If unchecked, any NoData cells in the value raster or the data layer stack will result in a NoData cell in the output raster
Output layer	`OUTPUT`	[same as input] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Output NoData value Optional	`OUTPUT_NO_DATA_VA`	[numeric: double] Default: -9999.0	Value to use for NoData in the output layer
Creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `CREATE_OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Output layer	`OUTPUT`	[raster]	Output raster layer containing the result
CRS authority identifier	`CRS_AUTHID`	[string]	The coordinate reference system of the output raster layer
Extent	`EXTENT`	[string]	The spatial extent of the output raster layer
Count of cells with equal value occurrences	`FOUND_LOCATION`	[numeric: integer]	
Height in pixels	`HEIGHT_IN_PIXE`	[numeric: integer]	The number of rows in the output raster layer
Total pixel count	`TOTAL_PIXEL_COUN`	[numeric: integer]	The count of pixels in the output raster layer
Mean frequency at valid cell locations	`MEAN_FREQUENCY`	[numeric: double]	
Count of value occurrences	`OCCURRENCE_COUNT`	[numeric: integer]	
Width in pixels	`WIDTH_IN_PIXEL`	[numeric: integer]	The number of columns in the output raster layer

Python code

Algorithm ID: `native:greaterthanfrequency`

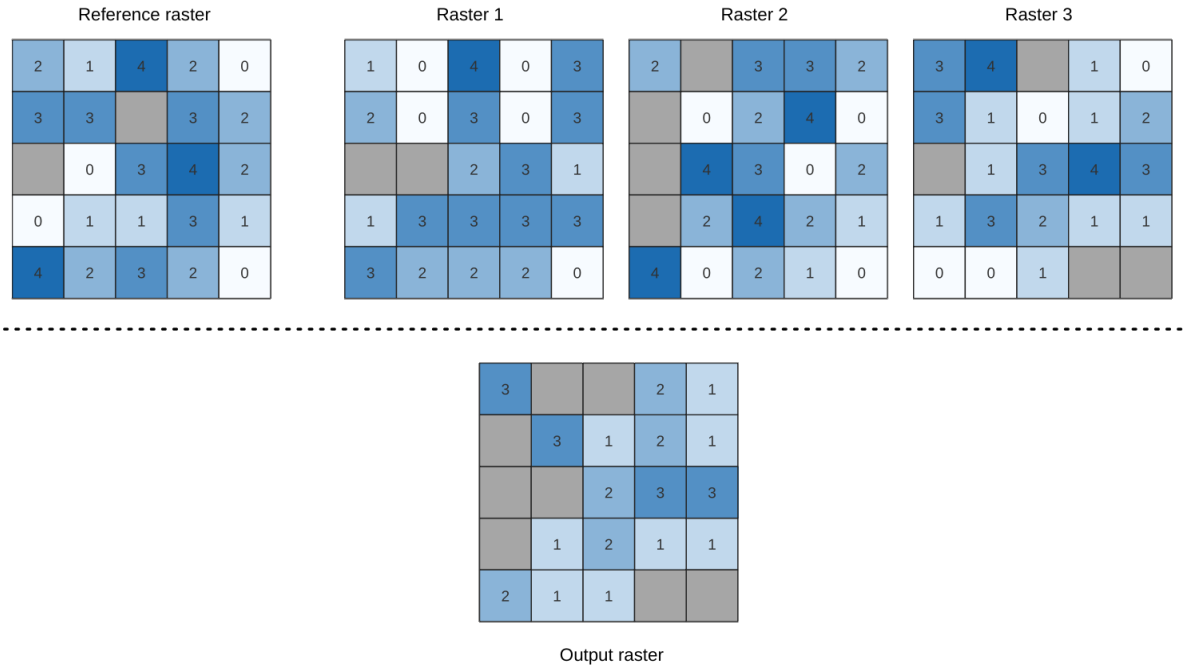
```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Highest position in raster stack

Evaluates on a cell-by-cell basis the position of the raster with the highest value in a stack of rasters. Position counts start with 1 and range to the total number of input rasters. The order of the input rasters is relevant for the algorithm. If multiple rasters feature the highest value, the first raster will be used for the position value.

If multiband rasters are used in the data raster stack, the algorithm will always perform the analysis on the first band of the rasters - use GDAL to use other bands in the analysis. Any NoData cells in the raster layer stack will result in a NoData cell in the output raster unless the “ignore NoData” parameter is checked. The output NoData value can be set manually. The output rasters extent and resolution is defined by a reference raster layer and is always of `Int32` type.



See also

Lowest position in raster stack

Parameters

Basic parameters

Label	Name	Type	Description
Input raster layers	`INPUT_RASTERS`	[raster] [list]	List of raster layers to compare with
Reference layer	`REFERENCE_LAYER`	[raster]	The reference layer for the output layer creation (extent, CRS, pixel dimensions)
Ignore NoData values	`IGNORE_NODATA`	[boolean] Default: False	If unchecked, any NoData cells in the data layer stack will result in a NoData cell in the output raster
Output layer	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output raster containing the result. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Output NoData value	OUTPUT_NODATA_VAL	[numeric: double] Default: -9999.0	Value to use for NoData in the output layer
Creation options Optional	CREATION_OPTIONS (for QGIS <= 3.42, this was CREATE_OPTIONS)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Output layer	OUTPUT	[raster]	Output raster layer containing the result
CRS authority identifier	CRS_AUTHID	[string]	The coordinate reference system of the output raster layer
Extent	EXTENT	[string]	The spatial extent of the output raster layer
Width in pixels	WIDTH_IN_PIXEL	[numeric: integer]	The number of columns in the output raster layer
Height in pixels	HEIGHT_IN_PIXE	[numeric: integer]	The number of rows in the output raster layer
Total pixel count	TOTAL_PIXEL_COUN	[numeric: integer]	The count of pixels in the output raster layer

Python code

Algorithm ID: `native:highestpositioninrasterstack`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Less than frequency

Evaluates on a cell-by-cell basis the frequency (number of times) the values of an input stack of rasters are less than the value of a value raster. The output raster extent and resolution is defined by the input raster layer and is always of `Int32` type.

If multiband rasters are used in the data raster stack, the algorithm will always perform the analysis on the first band of the rasters - use GDAL to use other bands in the analysis. The output NoData value can be set manually.

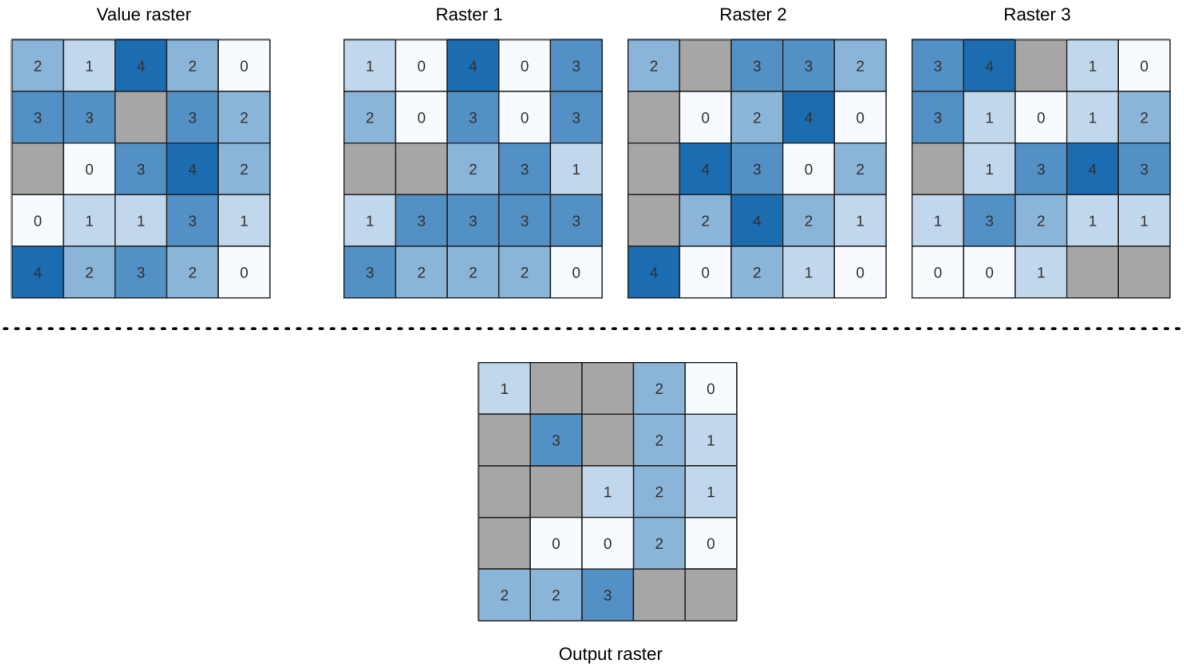


Fig. 24.39: For each cell in the output raster, the value represents the number of times that the corresponding cells in the list of rasters are less than the value raster. `NoData` cells (grey) are taken into account.

See also

Equal to frequency, *Greater than frequency*

Parameters

Basic parameters

Label	Name	Type	Description
Input value raster	`INPUT_VALUE_RAST`	[raster]	The input value layer serves as reference layer for the sample layers
Value raster band	`INPUT_VALUE_RAST`	[raster band] Default: The first band of the raster layer	Select the band you want to use as sample
Input raster layers	`INPUT_RASTERS`	[raster] [list]	Raster layers to evaluate. If multiband rasters are used in the data raster stack, the algorithm will always perform the analysis on the first band of the rasters
Ignore NoData values	`IGNORE_NODATA`	[boolean] Default: False	If unchecked, any NoData cells in the value raster or the data layer stack will result in a NoData cell in the output raster
Output layer	`OUTPUT`	[same as input] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Output NoData value Optional	OUTPUT_NO_DATA_VA	[numeric: double] Default: -9999.0	Value to use for NoData in the output layer
Creation options Optional	CREATION_OPTIONS (for QGIS <= 3.42, this was CREATE_OPTIONS)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Output layer	OUTPUT	[raster]	Output raster layer containing the result
CRS authority identifier	CRS_AUTHID	[string]	The coordinate reference system of the output raster layer
Extent	EXTENT	[string]	The spatial extent of the output raster layer
Count of cells with equal value occurrences	FOUND_LOCATION	[numeric: integer]	
Height in pixels	HEIGHT_IN_PIXE	[numeric: integer]	The number of rows in the output raster layer
Total pixel count	TOTAL_PIXEL_COUN	[numeric: integer]	The count of pixels in the output raster layer
Mean frequency at valid cell locations	MEAN_FREQUENCY	[numeric: double]	
Count of value occurrences	OCCURRENCE_COUNT	[numeric: integer]	
Width in pixels	WIDTH_IN_PIXEL	[numeric: integer]	The number of columns in the output raster layer

Python code

Algorithm ID: `native:lessthanfrequency`

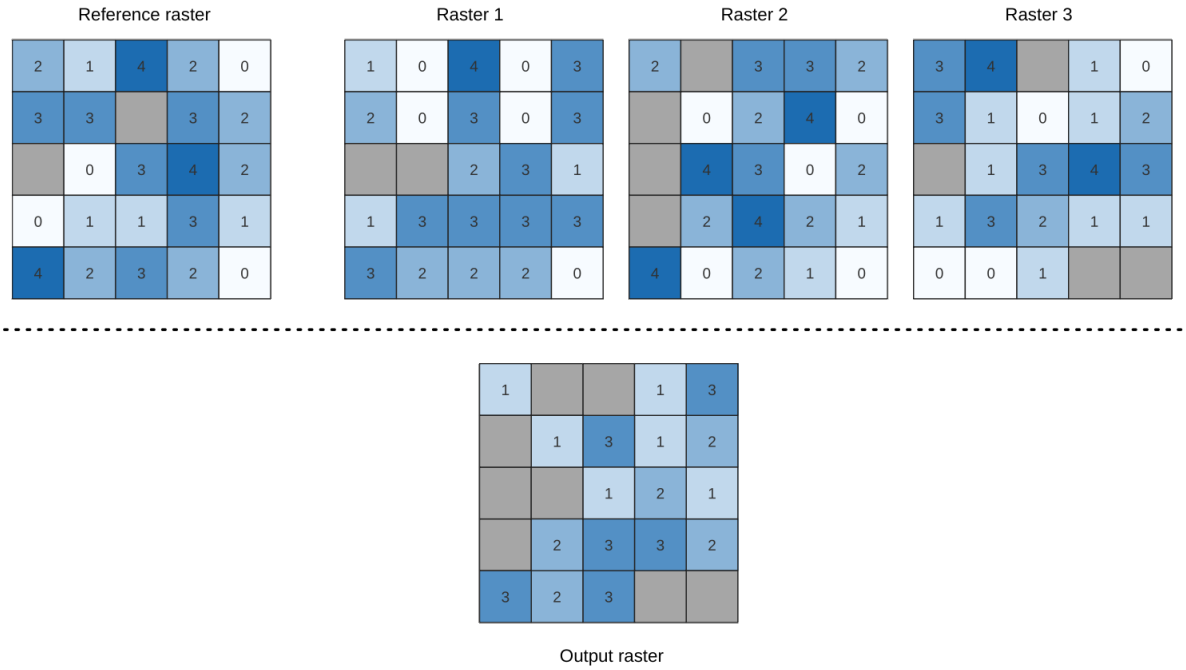
```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Lowest position in raster stack

Evaluates on a cell-by-cell basis the position of the raster with the lowest value in a stack of rasters. Position counts start with 1 and range to the total number of input rasters. The order of the input rasters is relevant for the algorithm. If multiple rasters feature the lowest value, the first raster will be used for the position value.

If multiband rasters are used in the data raster stack, the algorithm will always perform the analysis on the first band of the rasters - use GDAL to use other bands in the analysis. Any NoData cells in the raster layer stack will result in a NoData cell in the output raster unless the "ignore NoData" parameter is checked. The output NoData value can be set manually. The output rasters extent and resolution is defined by a reference raster layer and is always of `Int32` type.



See also

Highest position in raster stack

Parameters

Basic parameters

Label	Name	Type	Description
Input raster layers	`INPUT_RASTERS`	[raster] [list]	List of raster layers to compare with
Reference layer	`REFERENCE_LAYER`	[raster]	The reference layer for the output layer creation (extent, CRS, pixel dimensions)
Ignore NoData values	`IGNORE_NODATA`	[boolean] Default: False	If unchecked, any NoData cells in the data layer stack will result in a NoData cell in the output raster

continues on next page

Table 24.127 – continued from previous page

Label	Name	Type	Description
Output layer	OUTPUT	[raster] Default: [Save to temporary file]	Specification of the output raster containing the result. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Output NoData value	OUTPUT_NODATA_VAL	[numeric: double] Default: -9999.0	Value to use for NoData in the output layer
Creation options Optional	CREATION_OPTIONS (for QGIS <= 3.42, this was CREATE_OPTIONS)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Output layer	OUTPUT	[raster]	Output raster layer containing the result
CRS authority identifier	CRS_AUTHID	[string]	The coordinate reference system of the output raster layer
Extent	EXTENT	[string]	The spatial extent of the output raster layer
Width in pixels	WIDTH_IN_PIXEL	[numeric: integer]	The number of columns in the output raster layer
Height in pixels	HEIGHT_IN_PIXE	[numeric: integer]	The number of rows in the output raster layer
Total pixel count	TOTAL_PIXEL_COUN	[numeric: integer]	The count of pixels in the output raster layer

Python code

Algorithm ID: `native:lowestpositioninrasterstack`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Raster boolean AND

Calculates the boolean `AND` for a set of input rasters. If all of the input rasters have a non-zero value for a pixel, that pixel will be set to `1` in the output raster. If any of the input rasters have `0` values for the pixel it will be set to `0` in the output raster.

The reference layer parameter specifies an existing raster layer to use as a reference when creating the output raster. The output raster will have the same extent, CRS, and pixel dimensions as this layer.

By default, a NoData pixel in ANY of the input layers will result in a NoData pixel in the output raster. If the *Treat NoData values as false* option is checked, then NoData inputs will be treated the same as a `0` input value.

See also

Raster boolean OR

Parameters

Basic parameters

Label	Name	Type	Description
Input layers	`INPUT`	[raster] [list]	List of input raster layers
Reference layer	`REF_LAYER`	[raster]	The reference layer to create the output layer from (extent, CRS, pixel dimensions)
Treat NoData values as false	`NO-DATA_AS_FALSE`	[boolean] Default: False	Treat NoData values in the input files as 0 when performing the operation
Output layer	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output raster containing the result. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Output NoData value	`NO_DATA`	[numeric: double] Default: -9999.0	Value to use for NoData in the output layer

continues on next page

Table 24.130 – continued from previous page

Label	Name	Type	Description
Output data type	DATA_TYPE	[enumeration] Default: 5	Output raster data type. Options: • 0 — Byte (Eight bit unsigned integer (quint8)) • 1 — Int16 (Sixteen bit signed integer (qint16)) • 2 — UInt16 (Sixteen bit unsigned integer (quint16)) • 3 — Int32 (Thirty two bit signed integer (qint32)) • 4 — UInt32 (Thirty two bit unsigned integer (quint32)) • 5 — Float32 (Thirty two bit floating point (float)) • 6 — Float64 (Sixty four bit floating point (double)) • 7 — CInt16 (Complex Int16) • 8 — CInt32 (Complex Int32) • 9 — CFloat32 (Complex Float32) • 10 — CFloat64 (Complex Float64) • 11 — Int8 (Eight bit signed integer (qint8)) Available options depend on the GDAL version built with QGIS (see *Help* ► *About* menu)
Creation options Optional	CREATION_OPTIONS (for QGIS <= 3.42, this was CREATE_OPTIONS)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Extent	EXTENT	[string]	The spatial extent of the output raster layer
CRS authority identifier	CRS_AUTHID	[crs]	The coordinate reference system of the output raster layer
Width in pixels	WIDTH_IN_PIXEL	[numeric: integer]	The number of columns in the output raster layer
Height in pixels	HEIGHT_IN_PIXE	[numeric: integer]	The number of rows in the output raster layer
Total pixel count	TOTAL_PIXEL_COUN	[numeric: integer]	The count of pixels in the output raster layer
NoData pixel count	NODATA_PIXEL_COU	[numeric: integer]	The count of NoData pixels in the output raster layer
True pixel count	TRUE_PIXEL_COU	[numeric: integer]	The count of True pixels (value = 1) in the output raster layer
False pixel count	FALSE_PIXEL_CO	[numeric: integer]	The count of False pixels (value = 0) in the output raster layer

continues on next page

Table 24.131 – continued from previous page

Label	Name	Type	Description
Output layer	OUTPUT	[raster]	Output raster layer containing the result

Python code

Algorithm ID: native:rasterbooleanand

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Raster boolean OR

Calculates the boolean OR for a set of input rasters. If all of the input rasters have a zero value for a pixel, that pixel will be set to 0 in the output raster. If any of the input rasters have 1 values for the pixel it will be set to 1 in the output raster.

The reference layer parameter specifies an existing raster layer to use as a reference when creating the output raster. The output raster will have the same extent, CRS, and pixel dimensions as this layer.

By default, a NoData pixel in ANY of the input layers will result in a NoData pixel in the output raster. If the *Treat NoData values as false* option is checked, then NoData inputs will be treated the same as a 0 input value.

See also

Raster boolean AND

Parameters

Basic parameters

Label	Name	Type	Description
Input layers	INPUT	[raster] [list]	List of input raster layers
Reference layer	REF_LAYER	[raster]	The reference layer to create the output layer from (extent, CRS, pixel dimensions)
Treat NoData values as false	NODATA_AS_FALSE	[boolean] Default: False	Treat NoData values in the input files as 0 when performing the operation
Output layer	OUTPUT	[raster] Default: [Save to temporary file]	Specification of the output raster containing the result. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Output NoData value	NO_DATA	[numeric: double] Default: -9999.0	Value to use for NoData in the output layer
Output data type	DATA_TYPE	[enumeration] Default: 5	Output raster data type. Options: • 0 — Byte (Eight bit unsigned integer (quint8)) • 1 — Int16 (Sixteen bit signed integer (qint16)) • 2 — UInt16 (Sixteen bit unsigned integer (quint16)) • 3 — Int32 (Thirty two bit signed integer (qint32)) • 4 — UInt32 (Thirty two bit unsigned integer (quint32)) • 5 — Float32 (Thirty two bit floating point (float)) • 6 — Float64 (Sixty four bit floating point (double)) • 7 — CInt16 (Complex Int16) • 8 — CInt32 (Complex Int32) • 9 — CFloat32 (Complex Float32) • 10 — CFloat64 (Complex Float64) • 11 — Int8 (Eight bit signed integer (qint8)) Available options depend on the GDAL version built with QGIS (see *Help* ► *About* menu)
Creation options Optional	CREATION_OPTIONS (for QGIS <= 3.42, this was CREATE_OPTIONS)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Extent	EXTENT	[string]	The spatial extent of the output raster layer
CRS authority identifier	CRS_AUTHID	[crs]	The coordinate reference system of the output raster layer
Width in pixels	WIDTH_IN_PIXEL	[numeric: integer]	The number of columns in the output raster layer
Height in pixels	HEIGHT_IN_PIXE	[numeric: integer]	The number of rows in the output raster layer
Total pixel count	TOTAL_PIXEL_COUN	[numeric: integer]	The count of pixels in the output raster layer
NoData pixel count	NODATA_PIXEL_COU	[numeric: integer]	The count of NoData pixels in the output raster layer

continues on next page

Table 24.134 – continued from previous page

Label	Name	Type	Description
True pixel count	TRUE_PIXEL_COU	[numeric: integer]	The count of True pixels (value = 1) in the output raster layer
False pixel count	FALSE_PIXEL_CO	[numeric: integer]	The count of False pixels (value = 0) in the output raster layer
Output layer	OUTPUT	[raster]	Output raster layer containing the result

Python code

Algorithm ID: `native:rasterbooleanor`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Raster calculator

Performs algebraic operations using raster layers.

The resulting layer will have its values computed according to an expression. The expression can contain numerical values, operators and references to any of the layers in the current project.

See also

Raster calculator (virtual), *Raster calculator*, *Raster Calculator*

Parameters

Basic parameters

Label	Name	Type	Description
Input layers	INPUT	[raster] [list]	List of input raster layers
Expression	EXPRESSION	[expression]	*Raster-based expression* that will be used to calculate the output raster layer.

continues on next page

Table 24.135 – continued from previous page

Label	Name	Type	Description
Output extent Optional	`EXTENT`	[extent]	Specify the spatial extent of the output raster layer. If the extent is not specified, the minimum extent that covers all the selected reference layers will be used. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Output cell size (leave empty to set automatically) Optional	`CELLSIZE`	[numeric: double]	Cell size of the output raster layer. If the cell size is not specified, the minimum cell size of the selected reference layer(s) will be used. The cell size will be the same for the X and Y axes.
Output CRS Optional	`CRS`	[crs]	CRS of the output raster layer. If the output CRS is not specified, the CRS of the first reference layer will be used.
Calculated	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Creation options Optional	`CRE-ATION_OPTIONS`	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (`|`).

Outputs

Label	Name	Type	Description
Calculated	`OUTPUT`	[raster]	Output raster file with the calculated values.

Python code

Algorithm ID: `native:rastercalc`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Raster calculator (virtual)

Performs algebraic operations using raster layers and generates in-memory result.

The resulting layer will have its values computed according to an expression. The expression can contain numerical values, operators and references to any of the layers in the current project.

A virtual raster layer is a raster layer defined by its URI and whose pixels are calculated on-the-fly. It's not a new file on disk; the virtual layer is still connected to the rasters used in the calculation meaning that deleting or moving these rasters would break it. A *Layer name* can be provided, otherwise the calculation expression is used as such. Removing the virtual layer from the project deletes it, and it can be made persistent in file using the layer *Export* ► *Save as…* contextual menu.

See also

Raster calculator, *Raster calculator*, *Raster Calculator*

Parameters

Label	Name	Type	Description
Input layers	`INPUT`	[raster] [list]	List of input raster layers
Expression	`EXPRESSION`	[expression]	*Raster-based expression* that will be used to calculate the output raster layer.

continues on next page

Table 24.137 – continued from previous page

Label	Name	Type	Description
Output extent Optional	`EXTENT`	[extent]	Specify the spatial extent of the output raster layer. If the extent is not specified, the minimum extent that covers all the selected reference layers will be used. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Output cell size (leave empty to set automatically) Optional	`CELLSIZE`	[numeric: double]	Cell size of the output raster layer. If the cell size is not specified, the minimum cell size of the selected reference layer(s) will be used. The cell size will be the same for the X and Y axes.
Output CRS Optional	`CRS`	[crs]	CRS of the output raster layer. If the output CRS is not specified, the CRS of the first reference layer will be used.
Output layer name Optional	`LAYER_NAME`	[string]	The name to assign to the generated layer. If not set, the text of the calculation expression is used.

Outputs

Label	Name	Type	Description
Calculated	`OUTPUT`	[raster]	Output virtual raster layer with the calculated values.

Python code

Algorithm ID: `native:virtualrastercalc`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Raster layer properties

Returns basic properties of the given raster layer, including the extent, size in pixels and dimensions of pixels (in map units), number of bands, and NoData value.

This algorithm is intended for use as a means of extracting these useful properties to use as the input values to other algorithms in a model - e.g. to allow to pass an existing raster's pixel sizes over to a GDAL raster algorithm.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Input raster layer
Band number Optional	`BAND`	[raster band] Default: Not set	Whether to also return properties of a specific band. If a band is specified, the noData value for the selected band is also returned.

Outputs

Label	Name	Type	Description
Number of bands in raster	`BAND_COUNT`	[numeric: integer]	The number of bands in the raster
CRS authority identifier	`CRS_AUTHID`	[string]	The coordinate reference system of the output raster layer
Extent	`EXTENT`	[string]	The raster layer extent in the CRS
Band has a NoData value set	`HAS_NODATA_VAL`	[Boolean]	Indicates whether the raster layer has a value set for NoData pixels in the selected band
Height in pixels	`HEIGHT_IN_PIXE`	[numeric: integer]	The number of columns in the raster layer
Band NoData value	`NODATA_VALUE`	[numeric: double]	The value (if set) of the NoData pixels in the selected band
Pixel size (height) in map units	`PIXEL_HEIGHT`	[numeric: integer]	Vertical size in map units of the pixel
Pixel size (width) in map units	`PIXEL_WIDTH`	[numeric: integer]	Horizontal size in map units of the pixel
Width in pixels	`WIDTH_IN_PIXEL`	[numeric: integer]	The number of rows in the raster layer
Maximum x-coordinate	`X_MAX`	[numeric: double]	
Minimum x-coordinate	`X_MIN`	[numeric: double]	
Maximum y-coordinate	`Y_MAX`	[numeric: double]	
Minimum y-coordinate	`Y_MIN`	[numeric: double]	

Python code

Algorithm ID: `native:rasterlayerproperties`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Raster layer statistics

Calculates basic statistics from the values in a given band of the raster layer. The output is loaded in the *Processing* ► *Results viewer* menu.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Input raster layer
Band number	`BAND`	[raster band] Default: The first band of the input layer	If the raster is multiband, choose the band you want to get statistics for.
Statistics	`OUTPUT_HTML_FILE`	[html] Default: `[Save to temporary file]`	Specification of the output file: • Skip Output • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Maximum value	`MAX`	[numeric: double]	
Mean value	`MEAN`	[numeric: double]	
Minimum value	`MIN`	[numeric: double]	

continues on next page

Table 24.138 – continued from previous page

Label	Name	Type	Description
Statistics	`OUTPUT_HTML_FILE`	[html]	The output file contains the following information: • Analyzed file: path of the raster layer • Minimum value: minimum value of the raster • Maximum value: maximum value of the raster • Range: difference between the maximum and minimum values • Sum: total sum of the values • Mean value: mean of the values • Standard deviation: standard deviation of the values • Sum of the squares: sum of the squared differences of each observation from the overall mean
Range	`RANGE`	[numeric: double]	
Standard deviation	`STD_DEV`	[numeric: double]	
Sum	`SUM`	[numeric: double]	
Sum of the squares	`SUM_OF_SQUARES`	[numeric: double]	

Python code

Algorithm ID: `native:rasterlayerstatistics`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Raster layer unique values report

Returns the count and area of each unique value in a given raster layer. The calculation of the area is done in the area unit of the layer's CRS.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Input raster layer
Band number	`BAND`	[raster band] Default: The first band of the input layer	If the raster is multiband, choose the band you want to get statistics for.

continues on next page

Table 24.139 – continued from previous page

Label	Name	Type	Description
Unique values report	OUTPUT_HTML_FILE	[file] Default: [Save to temporary file]	Specification of the output file: • Skip Output • Save to a Temporary File • Save to File…
Unique values table	OUTPUT_TABLE	[vector: table] Default: [Skip output]	Specification of the table for unique values: • Skip Output • Create Temporary Layer (TEMPORARY_OUTPUT) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
CRS authority identifier	CRS_AUTHID	[string]	The coordinate reference system of the output raster layer
Extent	EXTENT	[string]	The spatial extent of the output raster layer
Height in pixels	HEIGHT_IN_PIXE	[numeric: integer]	The number of rows in the output raster layer
NoData pixel count	NODATA_PIXEL_COU	[numeric: integer]	The number of NoData pixels in the output raster layer
Total pixel count	TOTAL_PIXEL_COUN	[numeric: integer]	The count of pixels in the output raster layer
Unique values report	OUTPUT_HTML_FILE	[html]	The output HTML file contains the following information: • Analyzed file: the path of the raster layer • Extent: xmin, ymin, xmax, ymax coordinates of the extent • Projection: projection of the layer • Width in pixels: number of columns and pixel width size • Height in pixels: number of rows and pixel width size • Total pixel count: count of all the pixels • NoData pixel count: count of pixels with NoData value
Unique values table	OUTPUT_TABLE	[vector: table]	A table with three columns: • *value*: pixel value • *count*: count of pixels with this value • *m2* or *deg2* or *ft2* or … : total area of pixels with this value. The column name depends on the area unit of the layer's CRS and the calculation is done in that unit.

continues on next page

Table 24.140 – continued from previous page

Label	Name	Type	Description
Width in pixels	`WIDTH_IN_PIXEL`	[numeric: integer]	The number of columns in the output raster layer

Python code

Algorithm ID: `native:rasterlayeruniquevaluesreport`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Raster layer zonal statistics

Calculates statistics for a raster layer's values, categorized by zones defined in another raster layer.

See also

Zonal statistics

Parameters

Basic parameters

Label	Name	Type	Description
Input Layer	`INPUT`	[raster]	Input raster layer
Band number	`BAND`	[raster band] Default: The first band of the raster layer	If the raster is multiband choose the band for which you want to calculate the statistics.
Zones layer	`ZONES`	[raster]	Raster layer defining zones. Zones are given by contiguous pixels having the same pixel value.
Zones band number	`ZONES_BAND`	[raster band] Default: The first band of the raster layer	If the raster is multiband, choose the band that defines the zones
Statistics	`OUTPUT_TABLE`	[vector: table] Default: `[Create temporary layer]`	Specification of the output report. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Reference layer Optional	REF_LAYER	[enumeration] Default: 0	Raster layer used to calculate the centroids that will be used as reference when determining the zones in the output layer. One of: • 0 — Input layer: zones are determined by sampling the zone raster layer value at the centroid of each pixel from the source raster layer • 1 — Zones layer: the input raster layer will be sampled at the centroid of each pixel from the zones raster layer

Outputs

Label	Name	Type	Description
CRS authority identifier	CRS_AUTHID	[string]	The coordinate reference system of the output raster layer
Extent	EXTENT	[string]	The spatial extent of the output raster layer
Height in pixels	HEIGHT_IN_PIXE	[numeric: integer]	The number of rows in the output raster layer
NoData pixel count	NO-DATA_PIXEL_COU	[numeric: integer]	The number of NoData pixels in the output raster layer
Statistics	OUTPUT_TABLE	[vector: table]	The output layer contains the following information **for each zone**: • Area: the area in square raster units in the zone; • Sum: the total sum of the pixel values in the zone; • Count: the number of pixels in the zone; • Min: the minimum pixel value in the zone; • Max: the maximum pixel value in the zone; • Mean: the mean of the pixel values in the zone;
Total pixel count	TO-TAL_PIXEL_COUN	[numeric: integer]	The count of pixels in the output raster layer
Width in pixels	WIDTH_IN_PIXEL	[numeric: integer]	The number of columns in the output raster layer

Python code

Algorithm ID: `native:rasterlayerzonalstats`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Raster minimum/maximum

Extracts the minimum and maximum pixel values from a specified band of a raster layer. If multiple pixels share the same minimum or maximum value, only one of them will be included in the output.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Raster layer from which to extract the minimum and maximum pixel values.
Band number	`BAND`	[raster band] Default: 1	The band of the raster to analyze. If the raster is multiband, specify the band number (starting from 1).
Extract	`EXTREMA`	[enumeration] Default: 0 (Minimum and Maximum)	Choose which extrema to extract: • 0: Minimum and Maximum • 1: Minimum only • 2: Maximum only
Output layer	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specification of the output layer. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Output layer	`OUTPUT`	[vector: point]	Vector layer with point features at the locations of the minimum and/or maximum pixel values.

Python code

Algorithm ID: `native:rasterminmax`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Raster rank

`NEW in 3.44`

Performs a cell-by-cell analysis in which output values match the rank of a sorted list of overlapping cell values from input layers. The output raster will be multi-band if multiple ranks are provided. If multiband rasters are used in the data raster stack, the algorithm will always perform the analysis on the first band of the rasters.

See also

Cell stack percent rank from value, *Cell stack percentile*, *Cell stack percentrank from raster layer*, *Cell statistics*, *Raster calculator*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT_RASTERS`	[raster] [list]	Input raster layer(s) to calculate the rank from.
Rank	`RANKS`	[numeric: integer] [list] Default: [1]	List of ranks to calculate for each pixel in the sorted list of cells values from the input rasters. Cells whose values match the given rank(s) are kept in the output layer. The ranks are 1-based, meaning that the first rank is 1. Negative values are also supported.
NoData value handling	`NODATA_HANDLING`	[enumeration] Default: 0	Choose how to handle NoData values in the input raster: • 0 — Exclude NoData from values lists: will not include NoData values in the list of pixel values when calculating ranks. • 1 — Presence of NoData in a values list results in NoData output cells: the output raster will have NoData values in the corresponding locations.

continues on next page

Table 24.145 – continued from previous page

Label	Name	Type	Description
Ranked	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output raster containing the rank values. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Output extent Optional	`EXTENT`	[extent]	Specify the spatial extent of the output raster layer. If the extent is not specified, the minimum extent that covers all the selected reference layers will be used. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Output cell size Optional	`CELL_SIZE`	[numeric: double]	Cell size of the output raster layer. If the cell size is not specified, the minimum cell size of the selected reference layer(s) will be used. The cell size will be the same for the X and Y axes.
Output CRS Optional	`CRS`	[crs]	CRS of the output raster layer. If the output CRS is not specified, the CRS of the first reference layer will be used.

Outputs

Label	Name	Type	Description
Ranked	`OUTPUT`	[raster]	Output raster layer containing the result.

Python code

Algorithm ID: `native:rasterrank`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Raster surface volume

Calculates the volume under a raster surface relative to a given base level. This is mainly useful for Digital Elevation Models (DEM).

Parameters

Label	Name	Type	Description
INPUT layer	`INPUT`	[raster]	Input raster, representing a surface
Band number	`BAND`	[raster band] Default: The first band of the raster layer	If the raster is multiband, choose the band that shall define the surface.
Base level	`LEVEL`	[numeric: double] Default: 0.0	Define a base or reference value. This base is used in the volume calculation according to the `Method` parameter (see below).
Method	`METHOD`	[enumeration] Default: 0	Define the method for the volume calculation given by the difference between the raster pixel value and the `Base level`. Options: • 0 — Count Only Above Base Level: only pixels above the base level will add to the volume. • 1 — Count Only Below Base Level: only pixels below the base level will add to the volume. • 2 — Subtract Volumes Below Base level: pixels above the base level will add to the volume, pixels below the base level will subtract from the volume. • 3 — Add Volumes Below Base level: Add the volume regardless whether the pixel is above or below the base level. This is equivalent to sum the absolute values of the difference between the pixel value and the base level.

continues on next page

Table 24.147 – continued from previous page

Label	Name	Type	Description
Surface volume report	`OUTPUT_HTML_FILE`	[html] Default: `[Save to temporary file]`	Specification of the output HTML report. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Surface volume table	`OUTPUT_TABLE`	[vector: table] Default: `[Skip output]`	Specification of the output table. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Volume	`VOLUME`	[numeric: double]	The calculated volume
Area	`AREA`	[numeric: double]	The area in square map units
Pixel_count	`PIXEL_COUNT`	[numeric: integer]	The total number of pixels that have been analyzed
Surface volume report	`OUTPUT_HTML_FILE`	[html]	The output report (containing volume, area and pixel count) in HTML format
Surface volume table	`OUTPUT_TABLE`	[vector: table]	The output table (containing volume, area and pixel count)

Python code

Algorithm ID: `native:rastersurfacevolume`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Reclassify by layer

Reclassifies a raster band by assigning new class values based on the ranges specified in a vector table.

Parameters

Basic parameters

Label	Name	Type	Description
Raster layer	`INPUT_RASTER`	[raster]	Raster layer to reclassify
Band number	`RASTER_BAND`	[raster band] Default: The first band of the raster layer	If the raster is multiband, choose the band you want to reclassify.
Layer containing class breaks	`INPUT_TABLE`	[vector: any]	Vector layer containing the values to use for classification.
Minimum class value field	`MIN_FIELD`	[tablefield: numeric]	Field with the minimum value of the range for the class. Use `-inf` to capture the lowest possible value.
Maximum class value field	`MAX_FIELD`	[tablefield: numeric]	Field with the maximum value of the range for the class. Use `inf` to capture the highest possible value.
Output value field	`VALUE_FIELD`	[tablefield: numeric]	Field with the value that will be assigned to the pixels that fall in the class (between the corresponding min and max values). Use `nan` to set the value of the range to NoData.
Reclassified raster	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Output NoData value	`NO_DATA`	[numeric: double] Default: -9999.0	Value to apply to NoData values.
Range boundaries	`RANGE_BOUNDARI`	[enumeration] Default: 0	Defines comparison rules for the classification. Options: • 0 — min < value <= max • 1 — min <= value < max • 2 — min <= value <= max • 3 — min < value < max
Use NoData when no range matches value	`NO-DATA_FOR_MISSI`	[boolean] Default: False	Applies the NoData value to band values that do not fall in any class. If False, the original value is kept.

continues on next page

Table 24.149 – continued from previous page

Label	Name	Type	Description
Output data type	DATA_TYPE	[enumeration] Default: 5	Defines the format of the output raster file. Options: • 0 — Byte (Eight bit unsigned integer (quint8)) • 1 — Int16 (Sixteen bit signed integer (qint16)) • 2 — UInt16 (Sixteen bit unsigned integer (quint16)) • 3 — Int32 (Thirty two bit signed integer (qint32)) • 4 — UInt32 (Thirty two bit unsigned integer (quint32)) • 5 — Float32 (Thirty two bit floating point (float)) • 6 — Float64 (Sixty four bit floating point (double)) • 7 — CInt16 (Complex Int16) • 8 — CInt32 (Complex Int32) • 9 — CFloat32 (Complex Float32) • 10 — CFloat64 (Complex Float64) • 11 — Int8 (Eight bit signed integer (qint8)) Available options depend on the GDAL version built with QGIS (see *Help* ► *About* menu)
Creation options Optional	CREATION_OPTIONS (for QGIS <= 3.42, this was CREATE_OPTIONS)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Reclassified raster	OUTPUT	[raster]	Output raster layer with reclassified band values

Python code

Algorithm ID: `native:reclassifybylayer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Reclassify by table

Reclassifies a raster band by assigning new class values based on the ranges specified in a fixed table.

Parameters

Basic parameters

Label	Name	Type	Description
Raster layer	`INPUT_RASTER`	[raster]	Raster layer to reclassify
Band number	`RASTER_BAND`	[raster band] Default: 1	Raster band for which you want to recalculate values.
Reclassification table	`TABLE`	[vector: table]	A 3-columns table to fill with the values to set the boundaries of each class (`Minimum` and `Maximum`) and the new `Value` to assign to the band values that fall in the class. The value `-inf` can be used as minimum, `inf` as maximum, and `nan` can be used to set the output value to NoData.
Reclassified raster	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output raster layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Output NoData value	`NO_DATA`	[numeric: double] Default: -9999.0	Value to apply to NoData values.
Range boundaries	`RANGE_BOUNDARI`	[enumeration] Default: 0	Defines comparison rules for the classification. Options: • 0 — min < value <= max • 1 — min <= value < max • 2 — min <= value <= max • 3 — min < value < max
Use NoData when no range matches value	`NO-DATA_FOR_MISSI`	[boolean] Default: False	Applies the NoData value to band values that do not fall in any class. If False, the original value is kept.

continues on next page

Table 24.151 – continued from previous page

Label	Name	Type	Description
Output data type	DATA_TYPE	[enumeration] Default: 5	Defines the format of the output raster file. Options: • 0 — Byte (Eight bit unsigned integer (quint8)) • 1 — Int16 (Sixteen bit signed integer (qint16)) • 2 — UInt16 (Sixteen bit unsigned integer (quint16)) • 3 — Int32 (Thirty two bit signed integer (qint32)) • 4 — UInt32 (Thirty two bit unsigned integer (quint32)) • 5 — Float32 (Thirty two bit floating point (float)) • 6 — Float64 (Sixty four bit floating point (double)) • 7 — CInt16 (Complex Int16) • 8 — CInt32 (Complex Int32) • 9 — CFloat32 (Complex Float32) • 10 — CFloat64 (Complex Float64) • 11 — Int8 (Eight bit signed integer (qint8)) Available options depend on the GDAL version built with QGIS (see *Help* ► *About* menu)
Creation options Optional	CREATION_OPTIONS (for QGIS <= 3.42, this was CREATE_OPTIONS)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Reclassified raster	OUTPUT	[raster]	Output raster layer with reclassified band values

Python code

Algorithm ID: `native:reclassifybytable`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Rescale raster

Rescales raster layer to a new value range, while preserving the shape (distribution) of the raster's histogram (pixel values). Input values are mapped using a linear interpolation from the source raster's minimum and maximum pixel values to the destination minimum and miximum pixel range.

By default the algorithm preserves the original NoData value, but there is an option to override it.

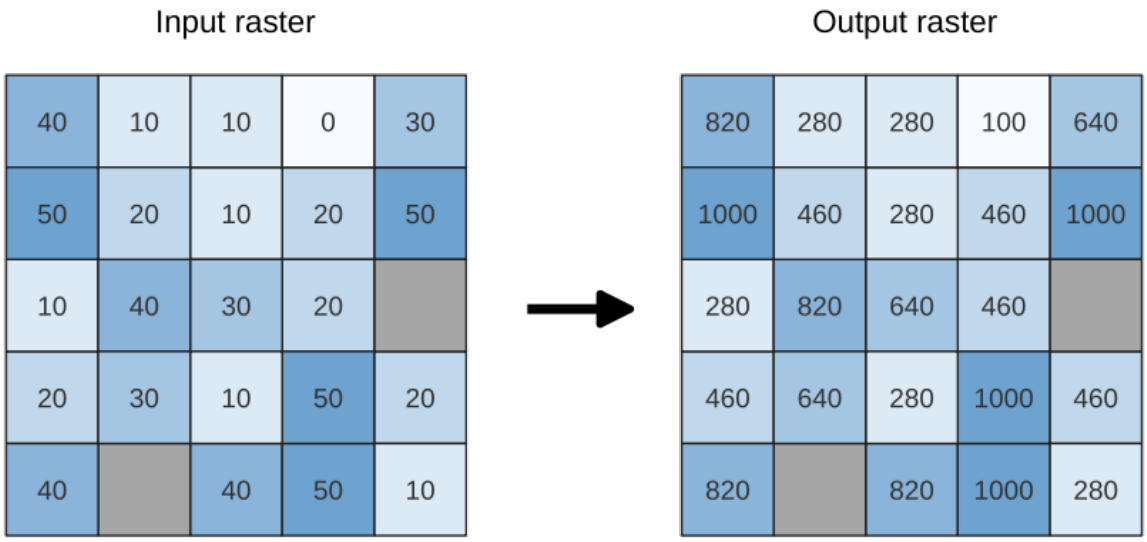


Fig. 24.40: Rescaling values of a raster layer from [0 - 50] to [100 - 1000]

Parameters

Basic parameters

Label	Name	Type	Description
Input Raster	`INPUT`	[raster]	Raster layer to use for rescaling
Band number	`BAND`	[raster band] Default: The first band of the input layer	If the raster is multiband, choose a band.
New minimum value	`MINIMUM`	[numeric: double] Default value: 0.0	Minimum pixel value to use in the rescaled layer
New maximum value	`MAXIMUM`	[numeric: double] Default value: 255.0	Maximum pixel value to use in the rescaled layer
New NoData value Optional	`NODATA`	[numeric: double] Default value: Not set	Value to assign to the NoData pixels. If unset, original NoData values are preserved.
Rescaled	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output raster layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Creation options Optional	CREATION_OPTIONS (for QGIS <= 3.42, this was CREATE_OPTIONS)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Rescaled	OUTPUT	[raster]	Output raster layer with rescaled band values

Python code

Algorithm ID: `native:rescaleraster`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Round raster

Rounds the cell values of a raster dataset according to the specified number of decimals.

Alternatively, a negative number of decimal places may be used to round values to powers of a base n. For example, with a Base value n of 10 and Decimal places of -1, the algorithm rounds cell values to multiples of 10, -2 rounds to multiples of 100, and so on. Arbitrary base values may be chosen, the algorithm applies the same multiplicative principle. Rounding cell values to multiples of a base n may be used to generalize raster layers.

The algorithm preserves the data type of the input raster. Therefore byte/integer rasters can only be rounded to multiples of a base n, otherwise a warning is raised and the raster gets copied as byte/integer raster.

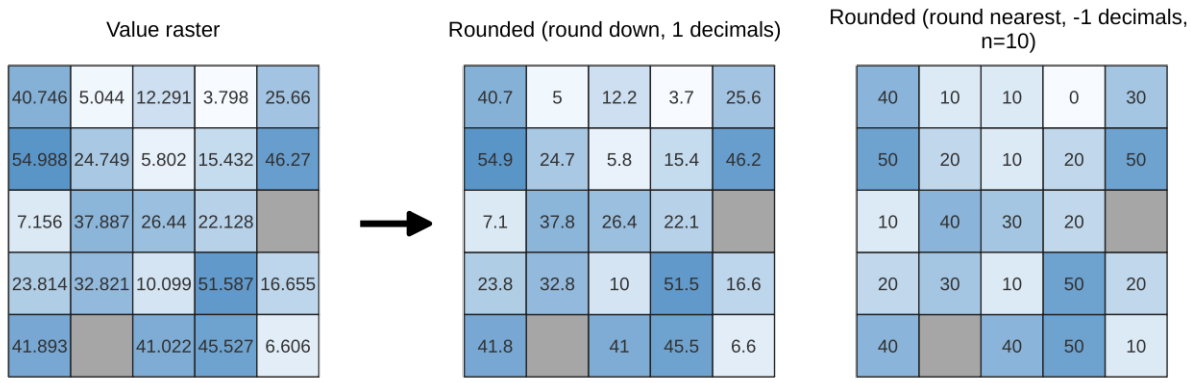


Fig. 24.41: Rounding values of a raster

Parameters

Basic parameters

Label	Name	Type	Description
Input raster	`INPUT`	[raster]	The raster to process.
Band number	`BAND`	[raster band] Default: 1	The band of the raster
Rounding direction	`ROUNDING_DIRECTION`	[list] Default: 1	How to choose the target rounded value. Options are: • 0 — Round up • 1 — Round to nearest • 2 — Round down
Number of decimals places	`DECIMAL_PLACES`	[numeric: integer] Default: 2	Number of decimals places to round to. Use negative values to round cell values to a multiple of a base n
Output raster	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output file. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Base n for rounding to multiples of n	`BASE_N`	[numeric: integer] Default: 10	When the `DECIMAL_PLACES` parameter is negative, raster values are rounded to multiples of the base n value

continues on next page

Table 24.154 – continued from previous page

Label	Name	Type	Description
Creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `CREATE_OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Output raster	`OUTPUT`	[raster]	The output raster layer with values rounded for the selected band.

Python code

Algorithm ID: `native:roundrastervalues`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Sample raster values

Extracts raster values at the point locations. If the raster layer is multiband, each band is sampled.

The attribute table of the resulting layer will have as many new columns as the raster layer band count.

Parameters

Label	Name	Type	Description
Input Layer	`INPUT`	[vector: point]	Point vector layer to use for sampling
Raster Layer	`RASTERCOPY`	[raster]	Raster layer to sample at the given point locations.
Output column prefix	`COLUMN_PREFIX`	[string] Default: 'SAMPLE_'	Prefix for the names of the added columns.
Sampled Optional	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specify the output layer containing the sampled values. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Sampled	`OUTPUT`	[vector: point]	The output layer containing the sampled values.

Python code

Algorithm ID: `native:rastersampling`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Zonal histogram

Appends fields representing counts of each unique value from a raster layer contained within polygon features.

The output layer attribute table will have as many fields as the unique values of the raster layer that intersects the polygon(s).

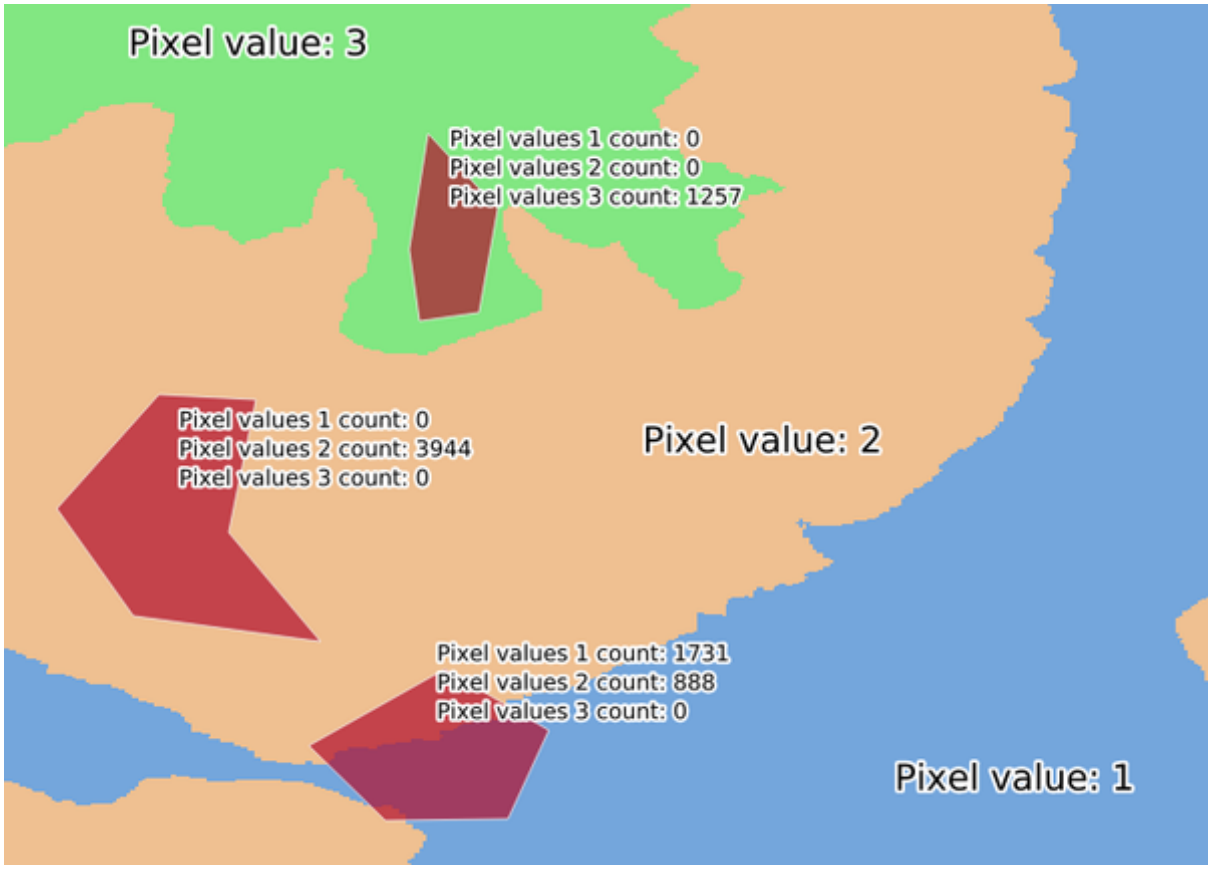


Fig. 24.42: Raster layer histogram example

Parameters

Label	Name	Type	Description
Raster layer	`INPUT_RASTER`	[raster]	Input raster layer.
Band number	`RASTER_BAND`	[raster band] Default: The first band of the input layer	If the raster is multiband, choose a band.
Vector layer containing zones	`INPUT_VECTOR`	[vector: polygon]	Vector polygon layer that defines the zones.
Output column prefix	`COLUMN_PREFIX` Optional	[string] Default: 'HISTO_'	Prefix for the output columns names.
Output zones	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output vector polygon layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Output zones	`OUTPUT`	[vector: polygon]	The output vector polygon layer.

Python code

Algorithm ID: `native:zonalhistogram`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Zonal Minimum/Maximum Point

Extracts point features corresponding to the minimum and maximum pixel values within polygon zones.

The output will contain one point feature for the minimum and one for the maximum raster value for every individual zonal feature from a polygon layer. The created point layer will be in the same spatial reference system as the selected raster layer.

Warning

This algorithm drops existing primary keys or FID values and regenerates them in output layers.

Parameters

Label	Name	Type	Description
Input Layer	`INPUT`	[vector: polygon]	Polygon layer defining the zones.
Raster Layer	`RASTER`	[raster]	Raster layer to extract the minimum and maximum values from.
Raster Band	`BAND`	[raster band] Default: The first band of the raster layer	If the raster has multiple bands, select the band to process.
Zonal extrema	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specification of the output layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Zonal extrema	`OUTPUT`	[vector: point]	Output layer containing the minimum and maximum points for each zone.

Python code

Algorithm ID: `native:zonalminmaxpoint`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Zonal statistics

Calculates statistics of a raster layer for each feature of an overlapping polygon vector layer.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: polygon]	Vector polygon layer that contains the zones.
Raster layer	`INPUT_RASTER`	[raster]	Input raster layer.
Raster band	`RASTER_BAND`	[raster band] Default: The first band of the input layer	If the raster is multiband, choose a band for the statistics.
Output column prefix	`COLUMN_PREFIX`	[string] Default: '_'	Prefix for the output columns names.
Statistics to calculate	`STATISTICS`	[enumeration] [list] Default: [0,1,2]	List of statistical operator for the output. Options: • 0 — Count • 1 — Sum • 2 — Mean • 3 — Median • 4 — St. dev. • 5 — Minimum • 6 — Maximum • 7 — Range • 8 — Minority • 9 — Majority • 10 — Variety • 11 — Variance

continues on next page

Table 24.155 – continued from previous page

Label	Name	Type	Description
Zonal Statistics	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output vector polygon layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Zonal Statistics	`OUTPUT`	[vector: polygon]	The zone vector layer with added statistics.

Python code

Algorithm ID: `native:zonalstatisticsfb`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.19 Raster Creation

Create constant raster layer

Generates raster layer for given extent and cell size filled with the specified value.

Additionally an output data type can be specified. The algorithm will abort if a value has been entered that cannot be represented by the selected output raster data type.

Parameters

Basic parameters

Label	Name	Type	Description
Desired extent	`EXTENT`	[extent]	Specify the spatial extent of the output raster layer. It will internally be extended to a multiple of the tile size. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Target CRS	`TARGET_CRS`	[crs] Default: Project CRS	CRS for the output raster layer
Pixel size	`PIXEL_SIZE`	[numeric: double] Default: 1.0	Pixel size (X=Y) in map units.
Constant value	`NUMBER`	[numeric: double] Default: 1.0	Constant pixel value for the output raster layer.
Constant	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Output raster data type	`OUTPUT_TYPE` Default: 5	[enumeration]	Defines the data type of the output raster file. Options: • 0 — Byte • 1 — Integer16 • 2 — Unsigned Integer16 • 3 — Integer32 • 4 — Unsigned Integer32 • 5 — Float32 • 6 — Float64
Creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `CREATE_OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Constant	`OUTPUT`	[raster]	Raster covering the desired extent with the specified pixel size and value.

Python code

Algorithm ID: `native:createconstantrasterlayer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Create random raster layer (binomial distribution)

Generates a raster layer for given extent and cell size filled with binomially distributed random values.

By default, the values will be chosen given an N of 10 and a probability of 0.5. This can be overridden by using the advanced parameter for N and probability. The raster data type is set to Integer types (Integer16 by default). The binomial distribution random values are defined as positive integer numbers. A floating point raster will represent a cast of integer values to floating point.

Parameters

Basic parameters

Label	Name	Type	Description
Desired extent	`EXTENT`	[extent]	Specify the spatial extent of the output raster layer. It will internally be extended to a multiple of the tile size. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Target CRS	`TARGET_CRS`	[crs] Default: Project CRS	CRS for the output raster layer
Pixel size	`PIXEL_SIZE`	[numeric: double] Default: 1.0	Pixel size (X=Y) in map units.
Output raster	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Output raster data type	`OUTPUT_TYPE` Default: 0	[enumeration]	Defines the data type of the output raster file. Options: • 0 — Integer16 • 1 — Unsigned Integer16 • 2 — Integer32 • 3 — Unsigned Integer32 • 4 — Float32 • 5 — Float64
N	`N`	[numeric: integer] Default: 10	
Probability	`PROBABILITY`	[numeric: double] Default: 0.5	
Creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `CREATE_OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Output raster	`OUTPUT`	[raster]	Raster covering the desired extent with the cell size filled with random values

Python code

Algorithm ID: `native:createrandombinomialrasterlayer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Create random raster layer (exponential distribution)

Generates a raster layer for given extent and cell size filled with exponentially distributed random values.

By default, the values will be chosen given a lambda of 1.0. This can be overridden by using the advanced parameter for lambda. The raster data type is set to Float32 by default as the exponential distribution random values are floating point numbers.

Parameters

Basic parameters

Label	Name	Type	Description
Desired extent	`EXTENT`	[extent]	Specify the spatial extent of the output raster layer. It will internally be extended to a multiple of the tile size. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Target CRS	`TARGET_CRS`	[crs] Default: Project CRS	CRS for the output raster layer
Pixel size	`PIXEL_SIZE`	[numeric: double] Default: 1.0	Pixel size (X=Y) in map units.
Output raster	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Output raster data type	`OUTPUT_TYPE` Default: 0	[enumeration]	Defines the data type of the output raster file. Options: • 0 — Float32 • 1 — Float64
Lambda	`LAMBDA`	[numeric: double] Default: 1.0	
Creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `CREATE_OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Output raster	`OUTPUT`	[raster]	Raster covering the desired extent with the cell size filled with random values

Python code

Algorithm ID: `native:createrandomexponentialrasterlayer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Create random raster layer (gamma distribution)

Generates a raster layer for given extent and cell size filled with gamma distributed random values.

By default, the values will be chosen given an alpha and beta value of 1.0. This can be overridden by using the advanced parameter for alpha and beta. The raster data type is set to Float32 by default as the gamma distribution random values are floating point numbers.

Parameters

Basic parameters

Label	Name	Type	Description
Desired extent	`EXTENT`	[extent]	Specify the spatial extent of the output raster layer. It will internally be extended to a multiple of the tile size. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Target CRS	`TARGET_CRS`	[crs] Default: Project CRS	CRS for the output raster layer
Pixel size	`PIXEL_SIZE`	[numeric: double] Default: 1.0	Pixel size (X=Y) in map units.
Output raster	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Output raster data type	`OUTPUT_TYPE` Default: 0	[enumeration]	Defines the data type of the output raster file. Options: • 0 — Float32 • 1 — Float64
Alpha	`ALPHA`	[numeric: double] Default: 1.0	
Beta	`BETA`	[numeric: double] Default: 1.0	
Creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `CREATE_OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Output raster	`OUTPUT`	[raster]	Raster covering the desired extent with the cell size filled with randomly distributed values

Python code

Algorithm ID: `native:createrandomgammarasterlayer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Create random raster layer (geometric distribution)

Generates a raster layer for given extent and cell size filled with geometrically distributed random values.

By default, the values will be chosen given a probability of 0.5. This can be overridden by using the advanced parameter for mean value. The raster data type is set to Integer types (Integer16 by default). The geometric distribution random values are defined as positive integer numbers. A floating point raster will represent a cast of integer values to floating point.

Parameters

Basic parameters

Label	Name	Type	Description
Desired extent	`EXTENT`	[extent]	Specify the spatial extent of the output raster layer. It will internally be extended to a multiple of the tile size. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Target CRS	`TARGET_CRS`	[crs] Default: Project CRS	CRS for the output raster layer
Pixel size	`PIXEL_SIZE`	[numeric: double] Default: 1.0	Pixel size (X=Y) in map units.
Output raster	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Output raster data type	`OUTPUT_TYPE` Default: 0	[enumeration]	Defines the data type of the output raster file. Options: • 0 — Integer16 • 1 — Unsigned Integer16 • 2 — Integer32 • 3 — Unsigned Integer32 • 4 — Float32 • 5 — Float64
Probability	`PROBABILITY`	[numeric: double] Default: 0.5	
Creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `CREATE_OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Output raster	`OUTPUT`	[raster]	Raster covering the desired extent with the cell size filled with randomly distributed values

Python code

Algorithm ID: `native:createrandomgeometricrasterlayer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Create random raster layer (negative binomial distribution)

Generates a raster layer for given extent and cell size filled with negative binomially distributed random values.

By default, the values will be chosen given a distribution parameter k of 10.0 and a probability of 0.5. This can be overridden by using the advanced parameters for k and probability. The raster data type is set to Integer types (Integer16 by default). The negative binomial distribution random values are defined as positive integer numbers. A floating point raster will represent a cast of integer values to floating point.

Parameters

Basic parameters

Label	Name	Type	Description
Desired extent	`EXTENT`	[extent]	Specify the spatial extent of the output raster layer. It will internally be extended to a multiple of the tile size. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Target CRS	`TARGET_CRS`	[crs] Default: Project CRS	CRS for the output raster layer
Pixel size	`PIXEL_SIZE`	[numeric: double] Default: 1.0	Pixel size (X=Y) in map units.
Output raster	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Output raster data type	`OUTPUT_TYPE` Default: 0	[enumeration]	Defines the data type of the output raster file. Options: • 0 — Integer16 • 1 — Unsigned Integer16 • 2 — Integer32 • 3 — Unsigned Integer32 • 4 — Float32 • 5 — Float64
Distribution parameter k	`K_PARAMETER`	[numeric: integer] Default: 10	
Probability	`PROBABILITY`	[numeric: double] Default: 0.5	
Creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `CREATE_OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Output raster	`OUTPUT`	[raster]	Raster covering the desired extent with the cell size filled with randomly distributed values

Python code

Algorithm ID: `native:createrandomnegativebinomialrasterlayer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Create random raster layer (normal distribution)

Generates a raster layer for given extent and cell size filled with normally distributed random values.

By default, the values will be chosen given a mean of 0.0 and a standard deviation of 1.0. This can be overridden by using the advanced parameters for mean and standard deviation value. The raster data type is set to Float32 by default as the normal distribution random values are floating point numbers.

Parameters

Basic parameters

Label	Name	Type	Description
Desired extent	`EXTENT`	[extent]	Specify the spatial extent of the output raster layer. It will internally be extended to a multiple of the tile size. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Target CRS	`TARGET_CRS`	[crs] Default: Project CRS	CRS for the output raster layer
Pixel size	`PIXEL_SIZE`	[numeric: double] Default: 1.0	Pixel size (X=Y) in map units.
Output raster	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Output raster data type	`OUTPUT_TYPE` Default: 0	[enumeration]	Defines the data type of the output raster file. Options: • 0 — Float32 • 1 — Float64
Mean of normal distribution	`MEAN`	[numeric: double] Default: 0.0	
Standard deviation of normal distribution	`STDDEV`	[numeric: double] Default: 1.0	
Creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `CREATE_OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Output raster	`OUTPUT`	[raster]	Raster covering the desired extent with the cell size filled with randomly distributed values

Python code

Algorithm ID: `native:createrandomnormalrasterlayer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Create random raster layer (poisson distribution)

Generates a raster layer for given extent and cell size filled with poisson distributed random values.

By default, the values will be chosen given a mean of 1.0. This can be overridden by using the advanced parameter for mean value. The raster data type is set to Integer types (Integer16 by default). The poisson distribution random values are positive integer numbers. A floating point raster will represent a cast of integer values to floating point.

Parameters

Basic parameters

Label	Name	Type	Description
Desired extent	`EXTENT`	[extent]	Specify the spatial extent of the output raster layer. It will internally be extended to a multiple of the tile size. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Target CRS	`TARGET_CRS`	[crs] Default: Project CRS	CRS for the output raster layer
Pixel size	`PIXEL_SIZE`	[numeric: double] Default: 1.0	Pixel size (X=Y) in map units.
Output raster	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Output raster data type	`OUTPUT_TYPE` Default: 0	[enumeration]	Defines the data type of the output raster file. Options: • 0 — Integer16 • 1 — Unsigned Integer16 • 2 — Integer32 • 3 — Unsigned Integer32 • 4 — Float32 • 5 — Float64
Mean	`MEAN`	[numeric: double] Default: 1.0	
Creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `CREATE_OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Output raster	`OUTPUT`	[raster]	Raster covering the desired extent with the cell size filled with randomly distributed values

Python code

Algorithm ID: `native:createrandompoissonrasterlayer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Create random raster layer (uniform distribution)

Generates a raster layer for given extent and cell size filled with random values.

By default, the values will range between the minimum and maximum value of the specified output raster type. This can be overridden by using the advanced parameters for lower and upper bound value. If the bounds have the same value or both are zero (default) the algorithm will create random values in the full value range of the chosen raster data type. Choosing bounds outside the acceptable range of the output raster type will abort the algorithm.

Parameters

Basic parameters

Label	Name	Type	Description
Desired extent	`EXTENT`	[extent]	Specify the spatial extent of the output raster layer. It will internally be extended to a multiple of the tile size. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Target CRS	`TARGET_CRS`	[crs] Default: Project CRS	CRS for the output raster layer
Pixel size	`PIXEL_SIZE`	[numeric: double] Default: 1.0	Pixel size (X=Y) in map units.
Output raster	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Output raster data type	`OUTPUT_TYPE` Default: 5	[enumeration]	Defines the data type of the output raster file. Options: • 0 — Byte • 1 — Integer16 • 2 — Unsigned Integer16 • 3 — Integer32 • 4 — Unsigned Integer32 • 5 — Float32 • 6 — Float64
Lower bound for random number range	`LOWER_BOUND`	[numeric: double] Default: 0.0	
Upper bound for random number range	`UPPER_BOUND`	[numeric: double] Default: 0.0	
Creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `CREATE_OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Output raster	`OUTPUT`	[raster]	Raster covering the desired extent with the cell size filled with randomly distributed values

Python code

Algorithm ID: `native:createrandomuniformrasterlayer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.20 Raster terrain analysis

Aspect

Calculates the aspect of the Digital Terrain Model in input. The final aspect raster layer contains values from 0 to 360 that express the slope direction, starting from north (0°) and continuing clockwise.

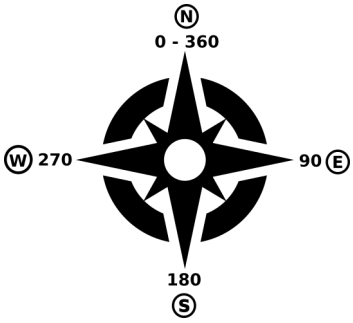


Fig. 24.43: Aspect values

The following picture shows the aspect layer reclassified with a color ramp:

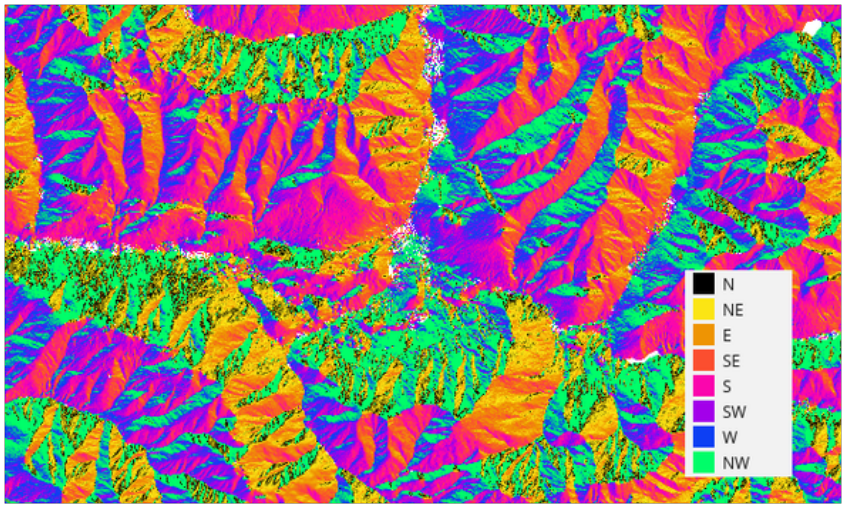


Fig. 24.44: Aspect layer reclassified

Parameters

Label	Name	Type	Description
Elevation layer	`INPUT`	[raster]	Digital Terrain Model raster layer
Z factor	`Z_FACTOR`	[numeric: double] Default: 1.0	Vertical exaggeration. This parameter is useful when the Z units differ from the X and Y units, for example feet and meters. You can use this parameter to adjust for this. The default is 1 (no exaggeration).
Aspect	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specify the output aspect raster layer. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Aspect	`OUTPUT`	[raster]	The output aspect raster layer

Python code

Algorithm ID: `qgis:aspect`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

DTM filter (slope-based)

Can be used to filter a digital elevation model in order to classify its cells into ground and object (non-ground) cells.

The tool uses concepts as described by Vosselman (2000) and is based on the assumption that a large height difference between two nearby cells is unlikely to be caused by a steep slope in the terrain. The probability that the higher cell might be non-ground increases when the distance between the two cells decreases. Therefore the filter defines a maximum height difference (`dz_max`) between two cells as a function of the distance (`d`) between the cells (`dz_max( d ) = d`). A cell is classified as terrain if there is no cell within the kernel radius to which the height difference is larger than the allowed maximum height difference at the distance between these two cells.

The approximate terrain slope (`s`) parameter is used to modify the filter function to match the overall slope in the study area (`dz_max( d ) = d * s`). A 5 % confidence interval (`ci = 1.65 * sqrt( 2 * stddev )`) may be used to modify the filter function even further by either relaxing (`dz_max( d ) = d * s + ci`) or amplifying (`dz_max( d ) = d * s - ci`) the filter criterium.

References: Vosselman, G. (2000): Slope based filtering of laser altimetry data. IAPRS, Vol. XXXIII, Part B3, Amsterdam, The Netherlands, 935-942

See also

This tool is a port of the SAGA DTM Filter (slope-based)

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Digital Terrain Model raster layer
Band number	`BAND`	[raster band]	The band of the DEM to consider
Kernel radius (pixels)	`RADIUS`	[numeric: integer] Default: 5	The radius of the filter kernel (in pixels). Must be large enough to reach ground cells next to non-ground objects.

continues on next page

Table 24.156 – continued from previous page

Label	Name	Type	Description
Terrain slope (%, pixel size/vertical units)	TERRAIN_SLOPE	[numeric: double] Default: 30.0	The approximate terrain slope in %. The terrain slope must be adjusted to account for the ratio of height units vs raster pixel dimensions. Used to relax the filter criterium in steeper terrain.
Filter modification	FILTER_MODIFICATI	[list] Default: 0	Choose whether to apply the filter kernel without modification or to use a confidence interval to relax or amplify the height criterium. • 0 - None • 1 - Relax filter • 2 - Amplify
Standard deviation	STANDARD_DEVIATION	[numeric: double] Default: 0.1	The standard deviation used to calculate a 5% confidence interval applied to the height threshold.
Output layer (ground) Optional	OUTPUT_GROUND	[raster] Default: [Save to temporary file]	Specify the filtered DEM containing only cells classified as ground. *One of*: • Skip Output • Save to a Temporary File • Save to File…
Output layer (non-ground objects) Optional	OUTPUT_NONGROUND	[raster] Default: [Skip output]	Specify the non-ground objects removed by the filter. *One of*: • Skip Output • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Creation options Optional	CREATION_OPTIONS (for QGIS <= 3.42, this was CREATE_OPTIONS)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Output layer (ground)	`OUTPUT_GROUND`	[raster]	The filtered DEM containing only cells classified as ground.
Output layer (non-ground objects)	`OUTPUT_NONGROUND`	[raster]	The non-ground objects removed by the filter.

Python code

Algorithm ID: `native:dtmslopebasedfilter`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Fill sinks (Wang & Liu)

`NEW in 3.44`

Uses a method proposed by Wang & Liu to identify and fill surface depressions in digital elevation models.

The method was enhanced to allow the creation of hydrologically sound elevation models, i.e. not only to fill the depression(s) but also to preserve a downward slope along the flow path. If desired, this is accomplished by preserving a minimum slope gradient (and thus elevation difference) between cells.

References: Wang, L. & H. Liu (2006): An efficient method for identifying and filling surface depressions in digital elevation models for hydrologic analysis and modelling. International Journal of Geographical Information Science, Vol. 20, No. 2: 193-213.

See also

This tool is a port of the SAGA Fill Sinks (Wang & Liu) tool.

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Digital Terrain Model raster layer
Band number	`BAND`	[raster band] Default: 1	The band of the DEM to consider
Minimum slope (degrees)	`MIN_SLOPE`	[numeric: double] Default: 0.10	Minimum slope gradient to preserve from cell to cell; with a value of zero, sinks are filled up to the spill elevation (which results in flat areas).

continues on next page

Table 24.158 – continued from previous page

Label	Name	Type	Description
Output layer (filled DEM) Optional	OUTPUT_FILLED_DEM	[raster] Default: [Save to temporary file]	Specify the ouput raster corresponding to the depression-free DEM. *One of*: • Skip Output • Save to a Temporary File • Save to File…
Output layer (flow directions) Optional	OUTPUT_FLOW_DIREC	[raster] Default: [Skip output]	Specify the output raster with computed flow directions. *One of*: • Skip Output • Save to a Temporary File • Save to File…
Output layer (watershed basins) Optional	OUTPUT_WATERSHED_	[raster] Default: [Skip output]	Specify the output raster corresponding to the delineated watershed basins. *One of*: • Skip Output • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Creation options Optional	CREATION_OPTIONS	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Output layer (filled DEM)	OUTPUT_FILLED_DEM	[raster]	Ouput raster corresponding to the depression-free digital elevation model.
Output layer (flow directions)	OUTPUT_FLOW_DIREC	[raster]	Output raster with computed flow directions; 0=N, 1=NE, 2=E, … 7=NW.
Output layer (watershed basins)	OUTPUT_WATERSHED_	[raster]	Output raster corresponding to the delineated watershed basins.

Python code

Algorithm ID: `native:fillsinkswangliu`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Hillshade

Calculates the hillshade raster layer given an input Digital Terrain Model.

The shading of the layer is calculated according to the sun position: you have the options to change both the horizontal angle (azimuth) and the vertical angle (sun elevation) of the sun.

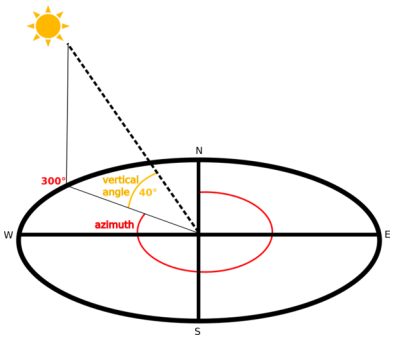


Fig. 24.45: Azimuth and vertical angle

The hillshade layer contains values from 0 (complete shadow) to 255 (complete sun). Hillshade is used usually to better understand the relief of the area.

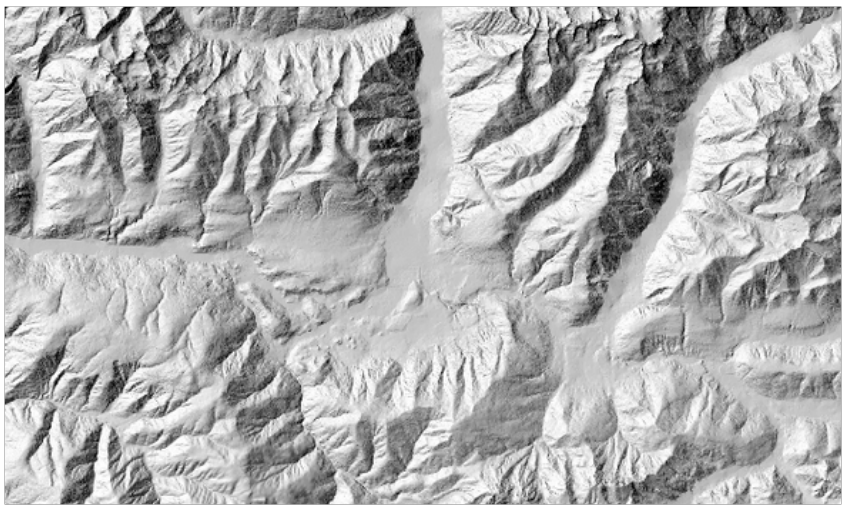


Fig. 24.46: Hillshade layer with azimuth 300 and vertical angle 45

Particularly interesting is to give the hillshade layer a transparency value and overlap it with the elevation raster:

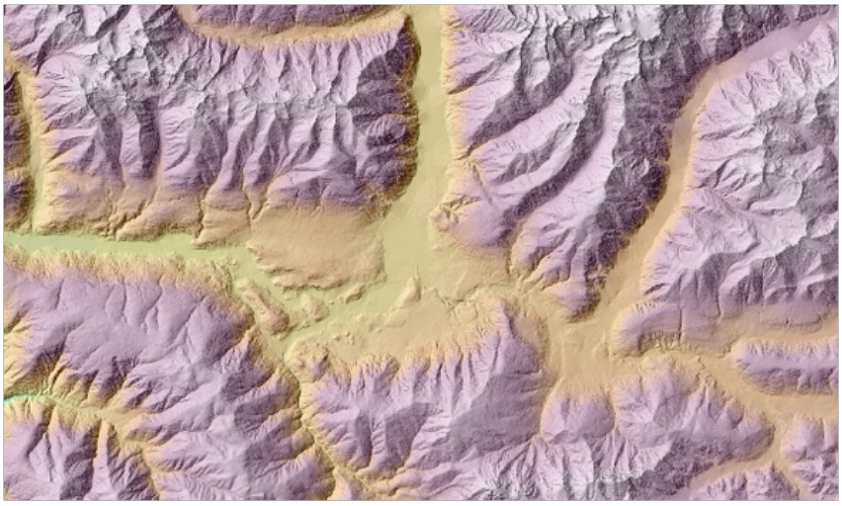


Fig. 24.47: Overlapping the hillshade with the elevation layer

Parameters

Label	Name	Type	Description
Elevation layer	`INPUT`	[raster]	Digital Terrain Model raster layer
Z factor	`Z_FACTOR`	[numeric: double] Default: 1.0	Vertical exaggeration. This parameter is useful when the Z units differ from the X and Y units, for example feet and meters. You can use this parameter to adjust for this. Increasing the value of this parameter will exaggerate the final result (making it look more "hilly"). The default is 1 (no exaggeration).
Azimuth (horizontal angle)	`AZIMUTH`	[numeric: double] Default: 300.0	Set the horizontal angle (in degrees) of the sun (clockwise direction). Range: 0 to 360. 0 is north.
Vertical angle	`V_ANGLE`	[numeric: double] Default: 40.0	Set the vertical angle (in degrees) of the sun, that is the height of the sun. Values can go from 0 (minimum elevation) to 90 (maximum elevation).
Hillshade	`OUTPUT`	[raster] Default: `Save to temporary file`	Specify the output hillshade raster layer. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Hillshade	`OUTPUT`	[raster]	The output hillshade raster layer

Python code

Algorithm ID: `qgis:hillshade`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Hypsometric curves

Calculates hypsometric curves for an input Digital Elevation Model. Curves are produced as CSV files in an output folder specified by the user.

A hypsometric curve is a cumulative histogram of elevation values in a geographical area.

You can use hypsometric curves to detect differences in the landscape due to the geomorphology of the territory.

Parameters

Label	Name	Type	Description
DEM to analyze	`INPUT_DEM`	[raster]	Digital Terrain Model raster layer to use for calculating altitudes
Boundary layer	`BOUNDARY_LAYER`	[vector: polygon]	Polygon vector layer with boundaries of areas used to calculate hypsometric curves
Step	`STEP`	[numeric: double] Default: 100.0	Vertical distance between curves
Use % of area instead of absolute value	`USE_PERCENTAGE`	[boolean] Default: False	Write area percentage to "Area" field of the CSV file instead of the absolute area
Hypsometric curves	`OUTPUT_DIRECTORY`	[folder]	Specify the output folder for the hypsometric curves. *One of*: • Save to a Temporary Directory • Save to Directory

Outputs

Label	Name	Type	Description
Hypsometric curves	`OUTPUT_DIRECTORY`	[folder]	Directory containing the files with the hypsometric curves. For each feature from the input vector layer, a CSV file with area and altitude values will be created. The file names start with `histogram_`, followed by layer name and feature ID.

	A	B
1	Area	Elevation
2	177475194.383	307
3	233206029.24	407
4	295553735.793	507
5	394718815.615	607
6	501801102.615	707
7	624399019.792	807
8	828877274.39	907
9	1042693465.68	1007
10	1277373021.81	1107
11	1556443975.41	1207
12	1888617494.27	1307
13	2248520437.31	1407
14	2627916813.17	1507
15	3010880212.04	1607
16	3411087555.34	1707

Python code

Algorithm ID: `qgis:hypsometriccurves`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Relief

Creates a shaded relief layer from digital elevation data. You can specify the relief color manually, or you can let the algorithm choose automatically all the relief classes.



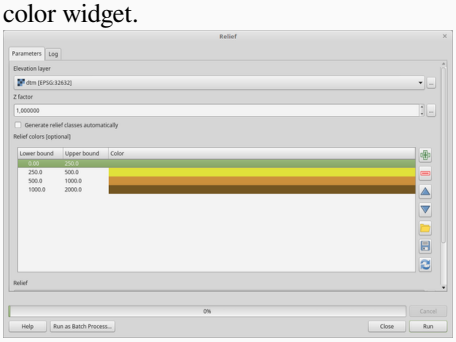
Fig. 24.48: Relief layer

Parameters

Label	Name	Type	Description
Elevation layer	`INPUT`	[raster]	Digital Terrain Model raster layer
Z factor	`Z_FACTOR`	[numeric: double] Default: 1.0	Vertical exaggeration. This parameter is useful when the Z units differ from the X and Y units, for example feet and meters. You can use this parameter to adjust for this. Increasing the value of this parameter will exaggerate the final result (making it look more "hilly"). The default is 1 (no exaggeration).
Generate relief classes automatically	`AUTO_COLORS`	[boolean] Default: False	If you check this option the algorithm will create all the relief color classes automatically

continues on next page

Table 24.162 – continued from previous page

Label	Name	Type	Description
Relief colors Optional	COLORS	[table widget]	Use the table widget if you want to choose the relief colors manually. You can add as many color classes as you want: for each class you can choose the lower and upper bound and finally by clicking on the color row you can choose the color thanks to the color widget.  Fig. 24.49: Manually setting of relief color classes The buttons in the right side panel give you the chance to: add or remove color classes, change the order of the color classes already defined, open an existing file with color classes and save the current classes as file.
Relief	OUTPUT	[raster] Default: [Save to temporary file]	Specify the output relief raster layer. *One of*: • Save to a Temporary File • Save to File…
Frequency distribution Optional	FREQUENCY_DISTRIB	[vector: table] Default: [Skip output]	Specify the CSV table for the output frequency distribution. *One of*: • Skip Output • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Relief	OUTPUT	[raster]	The output relief raster layer
Frequency distribution	OUTPUT	[vector: table]	The output frequency distribution

Python code

Algorithm ID: `qgis:relief`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Ruggedness index

Calculates the quantitative measurement of terrain heterogeneity described by Riley et al. (1999). It is calculated for every location, by summarizing the change in elevation within the 3x3 pixel grid.

Each pixel contains the difference in elevation from a center cell and the 8 cells surrounding it.

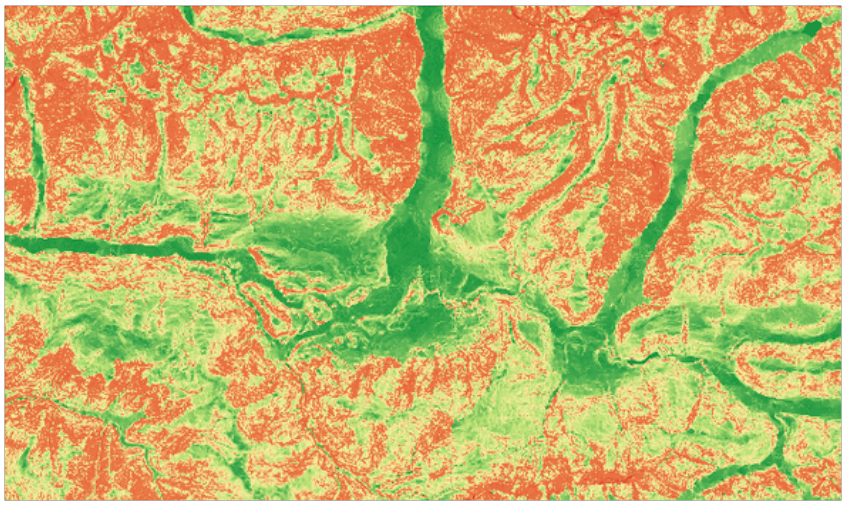


Fig. 24.50: Ruggedness layer from low (red) to high values (green)

Parameters

Label	Name	Type	Description
Elevation layer	`INPUT`	[raster]	Digital Terrain Model raster layer
Z factor	`Z_FACTOR`	[numeric: double] Default: 1.0	Vertical exaggeration. This parameter is useful when the Z units differ from the X and Y units, for example feet and meters. You can use this parameter to adjust for this. Increasing the value of this parameter will exaggerate the final result (making it look more rugged). The default is 1 (no exaggeration).
Ruggedness	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specify the output ruggedness raster layer. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Ruggedness	`OUTPUT`	[raster]	The output ruggedness raster layer

Python code

Algorithm ID: `qgis:ruggednessindex`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Slope

Calculates the slope from an input raster layer. The slope is the angle of inclination of the terrain and is expressed in **degrees**.

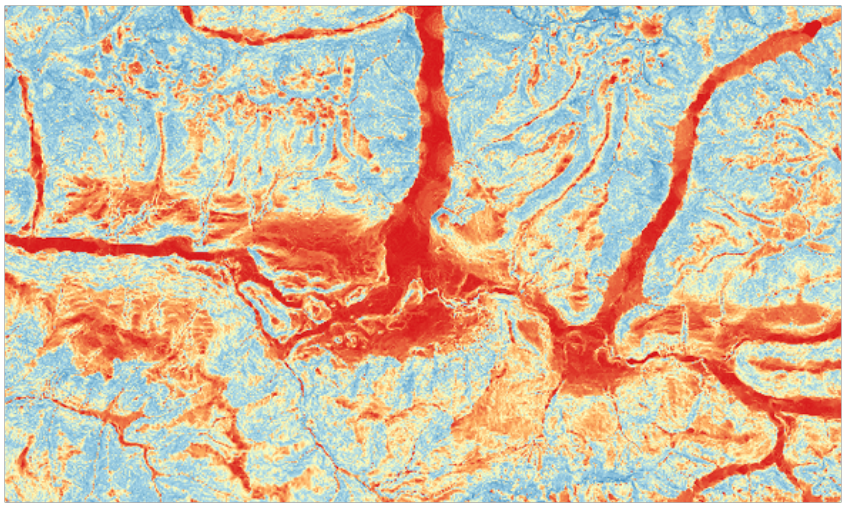


Fig. 24.51: Flat areas in red, steep areas in blue

Parameters

Label	Name	Type	Description
Elevation layer	`INPUT`	[raster]	Digital Terrain Model raster layer
Z factor	`Z_FACTOR`	[numeric: double] Default: 1.0	Vertical exaggeration. This parameter is useful when the Z units differ from the X and Y units, for example feet and meters. You can use this parameter to adjust for this. Increasing the value of this parameter will exaggerate the final result (making it steeper). The default is 1 (no exaggeration).
Slope	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specify the output slope raster layer. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Slope	`OUTPUT`	[raster]	The output slope raster layer

Python code

Algorithm ID: `qgis:slope`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.21 Raster tools

Align raster

Aligns raster by resampling it to the same cell size and reprojecting to the same CRS as a reference raster.

> **Warning**
>
> **This algorithm is ONLY available in the** *Model Designer* context. For other contexts, use instead *Align rasters*.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Input raster layer to align
Resampling method	`RESAMPLING_METHOD`	[enumeration] Default: 0	Method to use for input layer resampling. `Available options` are: • 0 — Nearest Neighbour • 1 — Bilinear (2x2 kernel) • 2 — Cubic (4x4 kernel) • 3 — Cubic B-Spline (4x4 kernel) • 4 — Lanczos (6x6 kernel) • 5 — Average • 6 — Mode • 7 — Maximum • 8 — Minimum • 9 — Median • 10 — First quartile (Q1) • 11 — Third quartile (Q3)
Rescale values according to the cell size	`RESCALE`	[boolean] Default: No	
Reference layer	`REFERENCE_LAYER`	[raster]	A raster layer that will be used to fetch extent, cell size and CRS that will be applied to input layers.
Override reference CRS Optional	`CRS`	[crs]	CRS to be used instead of the reference layer's
Override reference cell size X Optional	`CELL_SIZE_X`	[numeric: double]	Cell size in X direction to be used instead of the reference layer's
Override reference cell size Y Optional	`CELL_SIZE_Y`	[numeric: double]	Cell size in Y direction to be used instead of the reference layer's
Override reference grid offset X Optional	`GRID_OFFSET_X`	[numeric: double]	Offset in X direction to apply to cells grid
Override reference grid offset Y Optional	`GRID_OFFSET_Y`	[numeric: double]	Offset in Y direction to apply to cells grid

continues on next page

Table 24.163 – continued from previous page

Label	Name	Type	Description
Clip to extent Optional	`EXTENT`	[extent]	Specify the extent of the output raster layer. It will internally be extended to a multiple of the tile size. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Aligned raster	`OUTPUT`	[raster]	output raster with pixels resampled

Outputs

Label	Name	Type	Description
Aligned raster	`OUTPUT`	[raster]	output raster with pixels resampled

Python code

Algorithm ID: `native:alignsingleraster`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Align rasters

Aligns rasters by resampling them to the same cell size and reprojecting to the same CRS as a reference raster.

Warning

This algorithm is NOT available in the *Model Designer* context. Use instead *Align raster*.

Parameters

Label	Name	Type	Description
Input layers	`INPUT`	[raster] [list]	List of input raster layers with resampling options associated (filled as a `QgsProcessingParameterAlignRasterLayers` item — done in GUI by pressing *Configure Raster…* button for each selected layer): **Input layer [string] (`inputFile`)** Full path of the input layer to align **Output file [string] (`outputFile`)** Full path of the corresponding aligned output layer **Resampling method [enumeration] (`resampleMethod`)** Method to use for input layer resampling. `Available options` are: • 0 — Nearest Neighbour • 1 — Bilinear (2x2 kernel) • 2 — Cubic (4x4 kernel) • 3 — Cubic B-Spline (4x4 kernel) • 4 — Lanczos (6x6 kernel) • 5 — Average • 6 — Mode • 7 — Maximum • 8 — Minimum • 9 — Median • 10 — First quartile (Q1) • 11 — Third quartile (Q3) **Rescale values according to the cell size** [boolean] (`rescale`)
Reference layer	`REFERENCE_LAYER`	[raster]	A raster layer that will be used to fetch extent, cell size and CRS that will be applied to input layers.
Override reference CRS Optional	`CRS`	[crs]	CRS to be used instead of the reference layer's
Override reference cell size X Optional	`CELL_SIZE_X`	[numeric: double]	Cell size in X direction to be used instead of the reference layer's
Override reference cell size Y Optional	`CELL_SIZE_Y`	[numeric: double]	Cell size in Y direction to be used instead of the reference layer's
Override reference grid offset X Optional	`GRID_OFFSET_X`	[numeric: double]	Offset in X direction to apply to cells grid
Override reference grid offset Y Optional	`GRID_OFFSET_Y`	[numeric: double]	Offset in Y direction to apply to cells grid

continues on next page

Table 24.164 – continued from previous page

Label	Name	Type	Description
Clip to extent Optional	`EXTENT`	[extent]	Specify the extent of the output raster layer. It will internally be extended to a multiple of the tile size. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`

Outputs

Label	Name	Type	Description
Aligned rasters	`OUTPUT_LAYERS`	[raster] [list]	output rasters with pixels resampled

Python code

Algorithm ID: `native:alignrasters`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Convert map to raster

Creates a raster image of map canvas content.

A *map theme* can be selected to render a predetermined set of layers with a defined style for each layer.

Alternatively, a single layer can be selected if no map theme is set.

If neither map theme nor layer is set, the current map content will be rendered. The minimum extent entered will internally be extended to be a multiple of the tile size.

Parameters

Label	Name	Type	Description
Minimum extent to render (xmin, xmax, ymin, ymax)	`EXTENT`	[extent]	Specify the extent of the output raster layer. It will internally be extended to a multiple of the tile size. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Tile size	`TILE_SIZE`	[numeric: integer] Default: 1024	Size of the tile of the output raster layer. Minimum value: 64.
Map units per pixel	`MAP_UNITS_PER_`	[numeric: double] Default: 100.0	Pixel size (in map units). Minimum value: 0.0
Make background transparent	`MAKE_BACKGROUN`	[boolean] Default: False	Allows exporting the map with a transparent background. Outputs an RGBA (instead of RGB) image if set to `True`.
Map theme to render Optional	`MAP_THEME`	[enumeration]	Use an existing *map theme* for the rendering.
Single layer to render Optional	`LAYER`	[enumeration]	Choose a single layer for the rendering
Output layer	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Output layer	`OUTPUT`	[raster]	Output raster layer

Python code

Algorithm ID: `native:rasterize`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Fill NoData cells

Resets the NoData values in the input raster to a chosen value, resulting in raster dataset with no NoData pixels.

The algorithm respects the input raster data type, e.g. a floating point fill value will be truncated when applied to an integer raster.

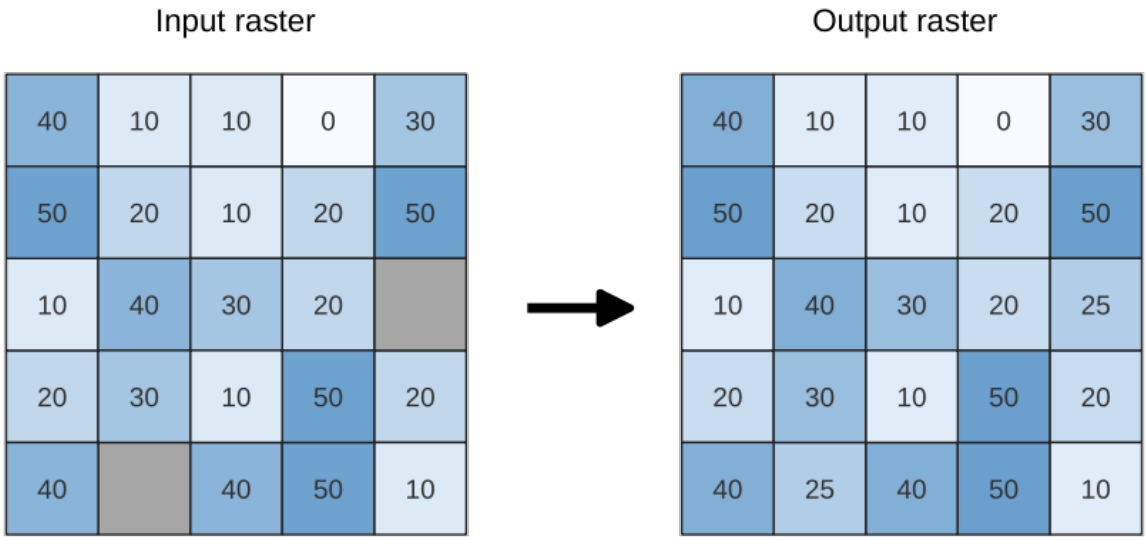


Fig. 24.52: Filling NoData values (in grey) of a raster

Parameters

Basic parameters

Label	Name	Type	Description
Input raster	`INPUT`	[raster]	The raster to process.
Band number	`BAND`	[raster band] Default: 1	The band of the raster
Fill value	`FILL_VALUE`	[numeric: double] Default: 1.0	Set the value to use for the NoData pixels
Output raster	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `CREATE_OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

Outputs

Label	Name	Type	Description
Output raster	`OUTPUT`	[raster]	The output raster layer with filled data cells.

Python code

Algorithm ID: `native:fillnodata`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Generate XYZ tiles (Directory)

Generates raster "XYZ" tiles using the current QGIS project as individual images to a directory structure.

Optionally, a Leaflet HTML output file using the generated tiles as a map layer could be created.

Parameters

Basic parameters

Label	Name	Type	Description
Extent (xmin, xmax, ymin, ymax)	`EXTENT`	[extent]	Specify the extent of the tiles. It will internally be extended to a multiple of the tile size. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Minimum zoom	`ZOOM_MIN`	[numeric: integer] Default: 12	Minimum 0, maximum 25.
Maximum zoom	`ZOOM_MAX`	[numeric: integer] Default: 12	Minimum 0, maximum 25.
DPI	`DPI`	[numeric: integer] Default: 96	Minimum 48, maximum 600.
Background color Optional	`BACKGROUND_COLOR`	[color] Default: QColor(0, 0, 0, 0)	Choose the background color for the tiles
Enable antialiasing	`ANTIALIAS`	[boolean] Default: True	Determines if antialiasing should be enabled
Tile format	`TILE_FORMAT`	[enumeration] Default: 0	One of: • 0 — PNG • 1 — JPG
Quality (JPG only) Optional	`QUALITY`	[numeric: integer] Default: 75	Minimum 1, maximum 100.
Metatile size Optional	`METATILESIZE`	[numeric: integer] Default: 4	Specify a custom metatile size when generating XYZ tiles. Larger values may speed up the rendering of tiles and provide better labelling (fewer gaps without labels) at the expense of using more memory. Minimum 1, maximum 20.
Tile width Optional	`TILE_WIDTH`	[numeric: integer] Default: 256	Minimum 1, maximum 4096.
Tile height Optional	`TILE_HEIGHT`	[numeric: integer] Default: 256	Minimum 1, maximum 4096.
Use inverted tile Y axis (TMS conventions)	`TMS_CONVENTION`	[boolean] Default: False	

continues on next page

Table 24.167 – continued from previous page

Label	Name	Type	Description
Output directory	`OUTPUT_DIRECTORY`	[folder] Default: `[Save to temporary folder]`	Specification of the output directory (for the tiles). *One of*: • Skip Output • Save to a Temporary Directory • Save to Directory
Output html (Leaflet) Optional	`OUTPUT_HTML`	[html] Default: `[Save to temporary file]`	Specification of the output HTML file. *One of*: • Skip Output • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Leaflet HTML output title Optional	`HTML_TITLE`	[string] Default: Not set	HTML <title>-tag used for the Leaflet HTML output file.
Leaflet HTML output attribution Optional	`HTML_ATTRIBUTI`	[string] Default: Not set	Custom map attribution used for the Leaflet HTML output file. HTML links are possible.
Include OpenStreetMap basemap in Leaflet HTML output	`HTML_OSM`	[boolean] Default: False	An OpenStreetMap basemap layer (source: https://tile.openstreetmap.org) is included in the Leaflet HTML output file. Proper map attribution is added automatically.

Outputs

Label	Name	Type	Description
Output directory	`OUTPUT_DIRECTORY`	[folder]	Output directory (for the tiles)
Output html (Leaflet)	`OUTPUT_HTML`	[html]	The output HTML (Leaflet) file

Python code

Algorithm ID: `native:tilesxyzdirectory`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Generate XYZ tiles (MBTiles)

Generates raster "XYZ" tiles using the current QGIS project as a single file in the "MBTiles" format.

Parameters

Label	Name	Type	Description
Extent (xmin, xmax, ymin, ymax)	`EXTENT`	[extent]	Specify the extent of the tiles. It will internally be extended to a multiple of the tile size. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Minimum zoom	`ZOOM_MIN`	[numeric: integer] Default: 12	Minimum 0, maximum 25.
Maximum zoom	`ZOOM_MAX`	[numeric: integer] Default: 12	Minimum 0, maximum 25.
DPI	`DPI`	[numeric: integer] Default: 96	Minimum 48, maximum 600.
Background color Optional	`BACKGROUND_COLOR`	[color] Default: QColor(0, 0, 0, 0)	Choose the background color for the tiles
Enable antialiasing	`ANTIALIAS`	[boolean] Default: True	Determines if antialiasing should be enabled
Tile format	`TILE_FORMAT`	[enumeration] Default: 0	One of: • 0 — PNG • 1 — JPG
Quality (JPG only) Optional	`QUALITY`	[numeric: integer] Default: 75	Minimum 1, maximum 100.
Metatile size Optional	`METATILESIZE`	[numeric: integer] Default: 4	Specify a custom metatile size when generating XYZ tiles. Larger values may speed up the rendering of tiles and provide better labelling (fewer gaps without labels) at the expense of using more memory. Minimum 1, maximum 20.

continues on next page

Table 24.169 – continued from previous page

Label	Name	Type	Description
Output file (for MBTiles)	`OUTPUT_FILE`	[file] Default: `[Save to temporary file]`	Specification of the output file. *One of*: • Skip Output • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Output file (for MBTiles)	`OUTPUT_FILE`	[file]	The output file.

Python code

Algorithm ID: `native:tilesxyzmbtiles`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.22 Vector analysis

Basic statistics for fields

Generates basic statistics for a field of the attribute table of a vector layer. Numeric, date, time and string fields are supported. The statistics returned will depend on the field type.

Statistics can be generated as a table or an HTML file and are available in the *Processing* ► *Results viewer*.

Default menu: *Vector* ► *Analysis Tools*

Parameters

Label	Name	Type	Description
Input vector	`INPUT_LAYER`	[vector: any]	Vector layer to calculate the statistics on
Field to calculate statistics on	`FIELD_NAME`	[tablefield: any]	Any supported table field to calculate the statistics
Statistics Optional	`OUTPUT`	[vector: table] Default: `[Create temporary layer]`	Specify the output table for the generated statistics. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Statistics report Optional	`OUTPUT_HTML_FILE`	[html] Default: `[Save to temporary file]`	Specification of the file for the calculated statistics. *One of*: • Skip Output • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Statistics	`OUTPUT`	[vector: table]	Table containing the calculated statistics
Statistics report	`OUTPUT_HTML_FILE`	[html]	HTML file with the calculated statistics
Count	`COUNT`	[numeric: integer]	
Number of unique values	`UNIQUE`	[numeric: integer]	
Number of empty (null) values	`EMPTY`	[numeric: integer]	
Number of non-empty values	`FILLED`	[numeric: integer]	
Minimum value	`MIN`	[same as input]	
Maximum value	`MAX`	[same as input]	
Minimum length	`MIN_LENGTH`	[numeric: integer]	
Maximum length	`MAX_LENGTH`	[numeric: integer]	
Mean length	`MEAN_LENGTH`	[numeric: double]	
Coefficient of Variation	`CV`	[numeric: double]	
Sum	`SUM`	[numeric: double]	
Mean value	`MEAN`	[numeric: double]	
Standard deviation	`STD_DEV`	[numeric: double]	
Range	`RANGE`	[numeric: double]	
Median	`MEDIAN`	[numeric: double]	
Minority (rarest occurring value)	`MINORITY`	[same as input]	
Majority (most frequently occurring value)	`MAJORITY`	[same as input]	
First quartile	`FIRSTQUARTILE`	[numeric: double]	
Third quartile	`THIRDQUARTILE`	[numeric: double]	
Interquartile Range (IQR)	`IQR`	[numeric: double]	

Python code

Algorithm ID: `qgis:basicstatisticsforfields`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Climb along line

Calculates the total climb and descent along line geometries. The input layer must have Z values present. If Z values are not available, the *Drape (set Z value from raster)* algorithm may be used to add Z values from a DEM layer.

The output layer is a copy of the input layer with additional fields that contain the total climb (`climb`), total descent (`descent`), the minimum elevation (`minelev`) and the maximum elevation (`maxelev`) for each line geometry. If the input layer contains fields with the same names as these added fields, they will be renamed (field names will be altered to "name_2", "name_3", etc, finding the first non-duplicate name).

Parameters

Label	Name	Type	Description
Line layer	`INPUT`	[vector: line]	Line layer to calculate the climb for. Must have Z values
Climb layer	`OUTPUT`	[vector: line] Default: `[Create temporary layer]`	Specification of the output (line) layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Climb layer	`OUTPUT`	[vector: line]	Line layer containing new attributes with the results from climb calculations.
Total climb	`TOTALCLIMB`	[numeric: double]	The sum of the climb for all the line geometries in the input layer
Total descent	`TOTALDESCENT`	[numeric: double]	The sum of the descent for all the line geometries in the input layer
Minimum elevation	`MINELEVATION`	[numeric: double]	The minimum elevation for the geometries in the layer
Maximum elevation	`MAXELEVATION`	[numeric: double]	The maximum elevation for the geometries in the layer

Python code

Algorithm ID: `native:climbalongline`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Count points in polygon

Takes a point and a polygon layer and counts the number of points from the point layer in each of the polygons of the polygon layer.

A new polygon layer is generated, with the exact same content as the input polygon layer, but containing an additional field with the points count corresponding to each polygon.

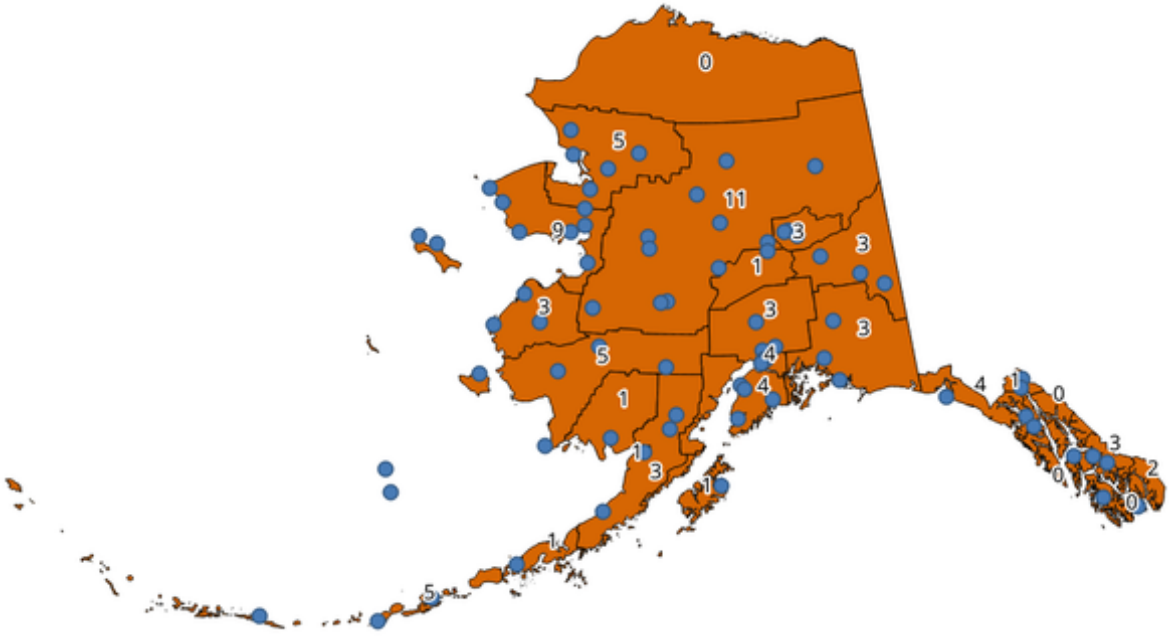


Fig. 24.53: The labels in the polygons show the point count

An optional weight field can be used to assign weights to each point. Alternatively, a unique class field can be specified. If both options are used, the weight field will take precedence and the unique class field will be ignored.

Allows *features in-place modification* of polygon features

`Default menu`: *Vector* ► *Analysis Tools*

Parameters

Label	Name	Type	Description
Polygons	`POLYGONS`	[vector: polygon]	Polygon layer whose features are associated with the count of points they contain
Points	`POINTS`	[vector: point]	Point layer with features to count
Weight field Optional	`WEIGHT`	[tablefield: any]	A field from the point layer. The count generated will be the sum of the weight field of the points contained by the polygon. If the weight field is not numeric, the count will be `0`.
Class field Optional	`CLASSFIELD`	[tablefield: any]	Points are classified based on the selected attribute and if several points with the same attribute value are within the polygon, only one of them is counted. The final count of the points in a polygon is, therefore, the count of different classes that are found in it.
Count field name	`FIELD`	[string] Default: 'NUMPOINTS'	The name of the field to store the count of points
Count	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specification of the output layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Count	`OUTPUT`	[vector: polygon]	Resulting layer with the attribute table containing the new column with the points count

Python code

Algorithm ID: `native:countpointsinpolygon`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

DBSCAN clustering

Clusters point features based on a 2D implementation of Density-based spatial clustering of applications with noise (DBSCAN) algorithm.

The algorithm requires two parameters, a minimum cluster size, and the maximum distance allowed between clustered points.

See also

ST-DBSCAN clustering, *K-means clustering*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: point]	Layer to analyze
Minimum cluster size	`MIN_SIZE`	[numeric: integer] Default: 5	Minimum number of features to generate a cluster
Maximum distance between clustered points	`EPS`	[numeric: double] Default: 1.0	Distance beyond which two features can not belong to the same cluster (eps)
Clusters	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specify the vector layer for the result of the clustering. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Treat border points as noise (DBSCAN*)	`DBSCAN*`	[boolean] Default: False	If checked, points on the border of a cluster are themselves treated as unclustered points, and only points in the interior of a cluster are tagged as clustered.
Cluster field name	`FIELD_NAME`	[string] Default: 'CLUSTER_ID'	Name of the field where the associated cluster number shall be stored
Cluster size field name	`SIZE_FIELD_NAM`	[string] Default: 'CLUSTER_SIZE'	Name of the field with the count of features in the same cluster

Outputs

Label	Name	Type	Description
Clusters	`OUTPUT`	[vector: point]	Vector layer containing the original features with a field setting the cluster they belong to
Number of clusters	`NUM_CLUSTERS`	[numeric: integer]	The number of clusters discovered

Python code

Algorithm ID: `native:dbscanclustering`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Distance matrix

Calculates for point features distances to their nearest features in the same layer or in another layer.

Default menu: *Vector* ► *Analysis Tools*

See also

Join attributes by nearest

Parameters

Label	Name	Type	Description
Input point layer	`INPUT`	[vector: point]	Point layer for which the distance matrix is calculated (**from** points)
Input unique ID field	`INPUT_FIELD`	[tablefield: any]	Field to use to uniquely identify features of the input layer. Used in the output attribute table.
Target point layer	`TARGET`	[vector: point]	Point layer containing the nearest point(s) to search (**to** points)
Target unique ID field	`TARGET_FIELD`	[tablefield: any]	Field to use to uniquely identify features of the target layer. Used in the output attribute table.

continues on next page

Table 24.174 – continued from previous page

Label	Name	Type	Description
Output matrix type	`MATRIX_TYPE`	[enumeration] Default: 0	Different types of calculation are available: • 0 — Linear (N * *k* x 3) distance matrix: for each input point, reports the distance to each of the *k* nearest target points. The output matrix consists of up to *k* rows per input point, and each row has three columns: *InputID*, *TargetID* and *Distance*. • 1 — Standard (N x T) distance matrix • 2 — Summary distance matrix (mean, std. dev., min, max): for each input point, reports statistics on the distances to its target points.
Use only the nearest (k) target points	`NEAREST_POINTS`	[numeric: integer] Default: 0	You can choose to calculate the distance to all the points in the target layer (*0*) or limit to a number (*k*) of closest features.
Distance matrix	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specification of the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Distance matrix	`OUTPUT`	[vector: point]	Point (or MultiPoint for the "Linear (N * *k* x 3)" case) vector layer containing the distance calculation for each input feature. Its features and attribute table depend on the selected output matrix type.

Python code

Algorithm ID: `qgis:distancematrix`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Distance to nearest hub (line to hub)

Creates lines that join each feature of an input vector to the nearest feature in a destination layer. Distances are calculated based on the *center* of each feature.

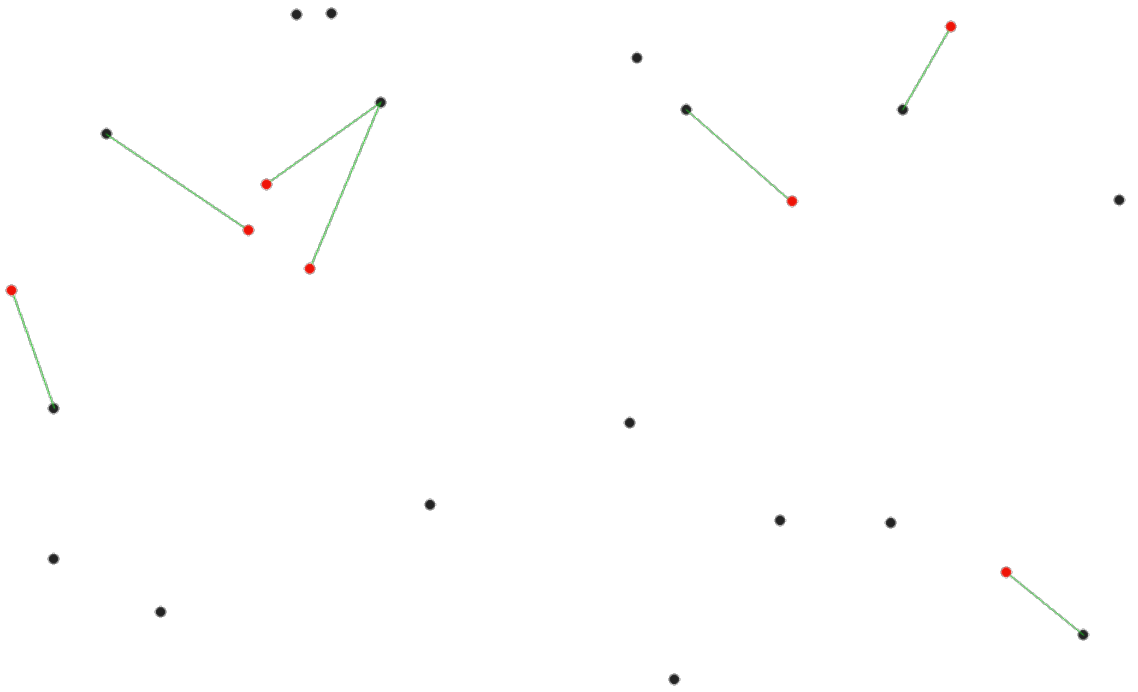


Fig. 24.54: Display the nearest hub for the red input features

See also

Distance to nearest hub (points), *Join attributes by nearest*

Parameters

Label	Name	Type	Description
Source points layer	`INPUT`	[vector: geometry]	Vector layer for which the nearest feature is searched
Destination hubs layer	`HUBS`	[vector: geometry]	Vector layer containing the features to search for
Hub layer name attribute	`FIELD`	[tablefield: any]	Field to use to uniquely identify features of the destination layer. Used in the output attribute table
Measurement unit	`UNIT`	[enumeration] Default: 0	Units in which to report the distance to the closest feature: • 0 — Meters • 1 — Feet • 2 — Miles • 3 — Kilometers • 4 — Layer units
Hub distance	`OUTPUT`	[vector: line] Default: `[Create temporary layer]`	Specify the output line vector layer connecting the matching points. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Hub distance	`OUTPUT`	[vector: line]	Line vector layer with the attributes of the input features, the identifier of their closest feature and the calculated distance.

Python code

Algorithm ID: `qgis:distancetonearesthublinetohub`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Distance to nearest hub (points)

Creates a point layer representing the *center* of the input features with the addition of two fields containing the identifier of the nearest feature (based on its center point) and the distance between the points.

See also

Distance to nearest hub (line to hub), *Join attributes by nearest*

Parameters

Label	Name	Type	Description
Source points layer	`INPUT`	[vector: any]	Vector layer for which the nearest feature is searched
Destination hubs layer	`HUBS`	[vector: any]	Vector layer containing the features to search for
Hub layer name attribute	`FIELD`	[tablefield: any]	Field to use to uniquely identify features of the destination layer. Used in the output attribute table
Measurement unit	`UNIT`	[enumeration] Default: 0	Units in which to report the distance to the closest feature: • 0 — Meters • 1 — Feet • 2 — Miles • 3 — Kilometers • 4 — Layer units
Hub distance	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specify the output point vector layer with the nearest hub. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Hub distance	`OUTPUT`	[vector: point]	Point vector layer representing the center of the source features with their attributes, the identifier of their closest feature and the calculated distance.

Python code

Algorithm ID: `qgis:distancetonearesthubpoints`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Join by lines (hub lines)

Creates hub and spoke diagrams by connecting lines from points on the Spoke layer to matching points in the Hub layer.

Determination of which hub goes with each point is based on a match between the Hub ID field on the hub points and the Spoke ID field on the spoke points.

If input layers are not point layers, a point on the surface of the geometries will be taken as the connecting location.

Optionally, geodesic lines can be created, which represent the shortest path on the surface of an ellipsoid. When geodesic mode is used, it is possible to split the created lines at the antimeridian (±180 degrees longitude), which can improve rendering of the lines. Additionally, the distance between vertices can be specified. A smaller distance results in a denser, more accurate line.

> **Warning**
>
> This algorithm drops existing primary keys or FID values and regenerates them in output layers.

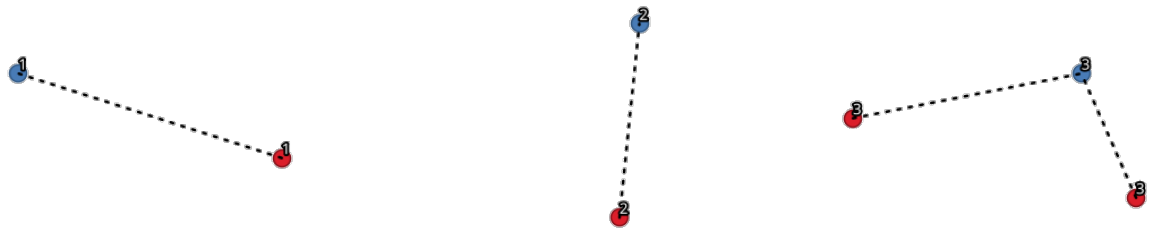


Fig. 24.55: Join points based on a common field / attribute

Parameters

Basic parameters

Label	Name	Type	Description
Hub layer	`HUBS`	[vector: geometry]	Input layer
Hub ID field	`HUB_FIELD`	[tablefield: any]	Field of the hub layer with ID to join

continues on next page

Table 24.176 – continued from previous page

Label	Name	Type	Description
Hub layer fields to copy (leave empty to copy all fields) Optional	HUB_FIELDS	[tablefield: any] [list]	The field(s) of the hub layer to be copied. If no field(s) are chosen all fields are taken.
Spoke layer	SPOKES	[vector: geometry]	Additional spoke point layer
Spoke ID field	SPOKE_FIELD	[tablefield: any]	Field of the spoke layer with ID to join
Spoke layer fields to copy (leave empty to copy all fields) Optional	SPOKE_FIELDS	[tablefield: any] [list]	Field(s) of the spoke layer to be copied. If no fields are chosen all fields are taken.
Create geodesic lines	GEODESIC	[boolean] Default: False	Create geodesic lines (the shortest path on the surface of an ellipsoid)
Hub lines	OUTPUT	[vector: line] Default: [Create temporary layer]	Specify the output hub line vector layer. *One of*: • Create Temporary Layer (TEMPORARY_OUTPUT) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Distance between vertices (geodesic lines only)	GEODESIC_DISTA	[numeric: double] Default: 1000.0 (kilometers)	Distance between consecutive vertices (in kilometers). A smaller distance results in a denser, more accurate line
Split lines at antimeridian (±180 degrees longitude)	ANTIMERIDIAN_SPLIT	[boolean] Default: False	Split lines at ±180 degrees longitude (to improve rendering of the lines)

Outputs

Label	Name	Type	Description
Hub lines	OUTPUT	[vector: line]	The resulting line layer connecting matching points in input layers

Python code

Algorithm ID: `native:hublines`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

K-means clustering

Calculates the 2D distance based k-means cluster number for each input feature.

K-means clustering aims to partition the features into k clusters in which each feature belongs to the cluster with the nearest mean. The mean point is represented by the barycenter of the clustered features.

If input geometries are lines or polygons, the clustering is based on the centroid of the feature.

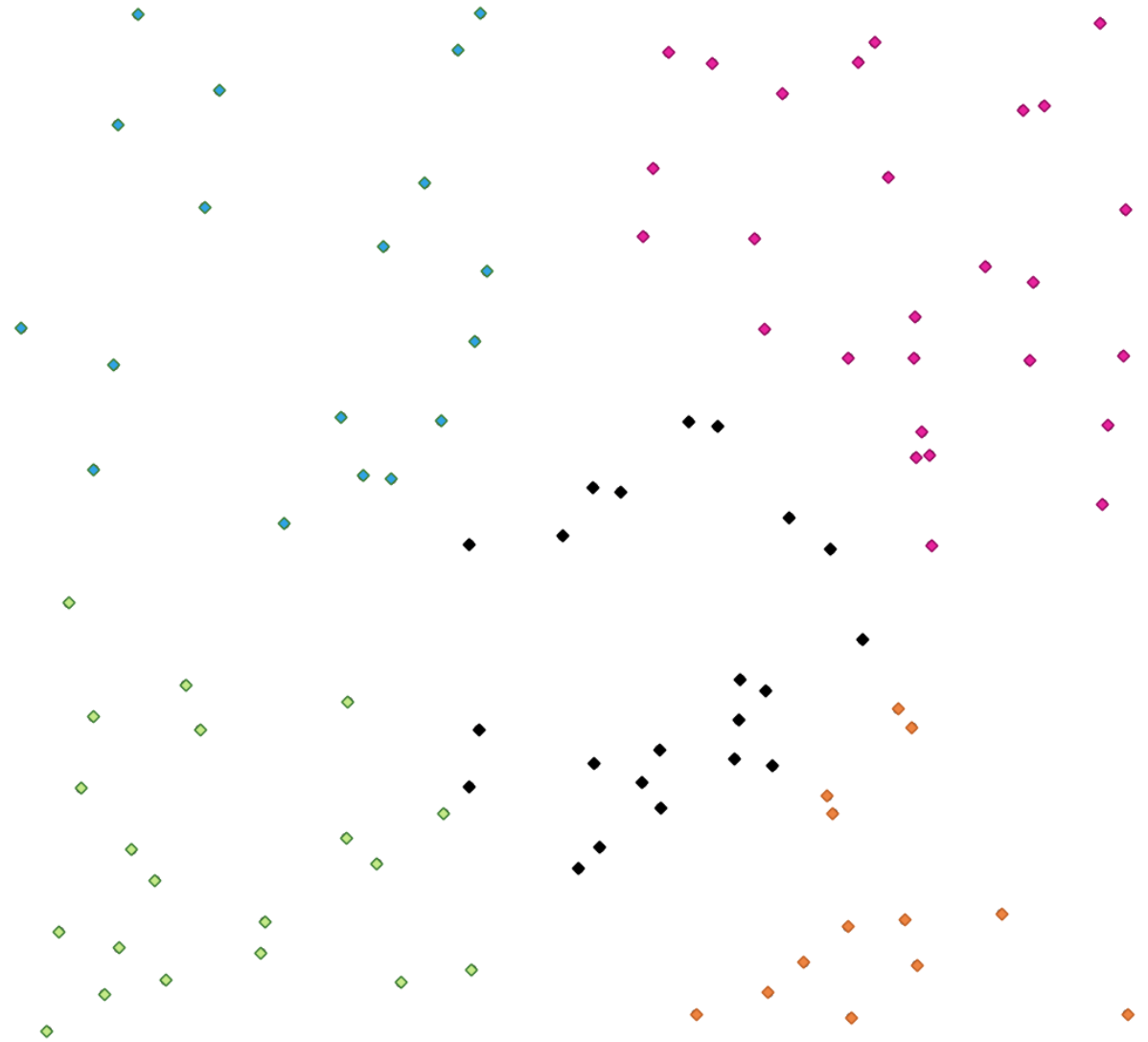


Fig. 24.56: A five class point clusters

See also

DBSCAN clustering, *ST-DBSCAN clustering*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Layer to analyze
Number of clusters	`CLUSTERS`	[numeric: integer] Default: 5	Number of clusters to create with the features
Method	`METHOD`	[enumeration] Default: 0	Initial centers selection method. Possible values are: • 0 — Farthest points: the initial centers are selected as the farthest points from each other. • 1 — K-means++: The first center is selected uniformly at random from the set of input features. Subsequent centers are chosen based on a weighted probability distribution that favors features farther from existing centers. This ensures that the initial centers are well-dispersed, reducing the likelihood of poor clustering and improving the algorithm's performance. *References: Arthur, David & Vassilvitskii, Sergei. (2007). K-Means++: The Advantages of Careful Seeding. Proc. of the Annu. ACM-SIAM Symp. on Discrete Algorithms. 8.*
Clusters	`OUTPUT`	[vector: same as input] Default: `[Create temporary layer]`	Specify the output vector layer for the generated clusters. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Cluster field name	`FIELD_NAME`	[string] Default: 'CLUSTER_ID'	Name of the field where the associated cluster number shall be stored
Cluster size field name	`SIZE_FIELD_NAM`	[string] Default: 'CLUSTER_SIZE'	Name of the field with the count of features in the same cluster

Outputs

Label	Name	Type	Description
Clusters	`OUTPUT`	[vector: same as input]	Vector layer containing the original features with fields specifying the cluster they belong to and their number in it

Python code

Algorithm ID: `native:kmeansclustering`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

List unique values

Lists unique values of an attribute table field and counts their number.

Default menu: *Vector* ► *Analysis Tools*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	Layer to analyze
Target field(s)	`FIELDS`	[tablefield: any] [list]	Field(s) to analyze
Unique values Optional	`OUTPUT`	[vector: table] Default: `[Create temporary layer]`	Specify the summary table layer with unique values. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
HTML report Optional	`OUTPUT_HTML_FILE`	[html] Default: `[Save to temporary file]`	HTML report of unique values in the *Processing* ► *Results viewer*. *One of*: • Skip Output • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Unique values	`OUTPUT`	[vector: table]	Summary table layer with unique values
HTML report	`OUTPUT_HTML_FILE`	[html]	HTML report of unique values. Can be opened from the *Processing* ► *Results viewer*
Total unique values	`TOTAL_VALUES`	[numeric: integer]	The number of unique values in the input field
Unique values concatenated	`UNIQUE_VALUES`	[string]	A string with the comma separated list of unique values found in the input field

Python code

Algorithm ID: `native:listuniquevalues`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Mean coordinate(s)

Computes a point layer with the center of mass of geometries in an input layer.

An attribute can be specified as containing weights to be applied to each feature when computing the center of mass.

If an attribute is selected in the parameter, features will be grouped according to values in this field. Instead of a single point with the center of mass of the whole layer, the output layer will contain a center of mass for the features in each category.

Default menu: *Vector* ► *Analysis Tools*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Weight field Optional	`WEIGHT`	[tablefield: numeric]	Field to use if you want to perform a weighted mean
Unique ID field	`UID`	[tablefield: any]	Unique field on which the calculation of the mean will be made
Mean coordinates	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specify the (point vector) layer for the result. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Mean coordinates	`OUTPUT`	[vector: point]	Resulting point(s) layer

Python code

Algorithm ID: `native:meancoordinates`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Nearest neighbour analysis

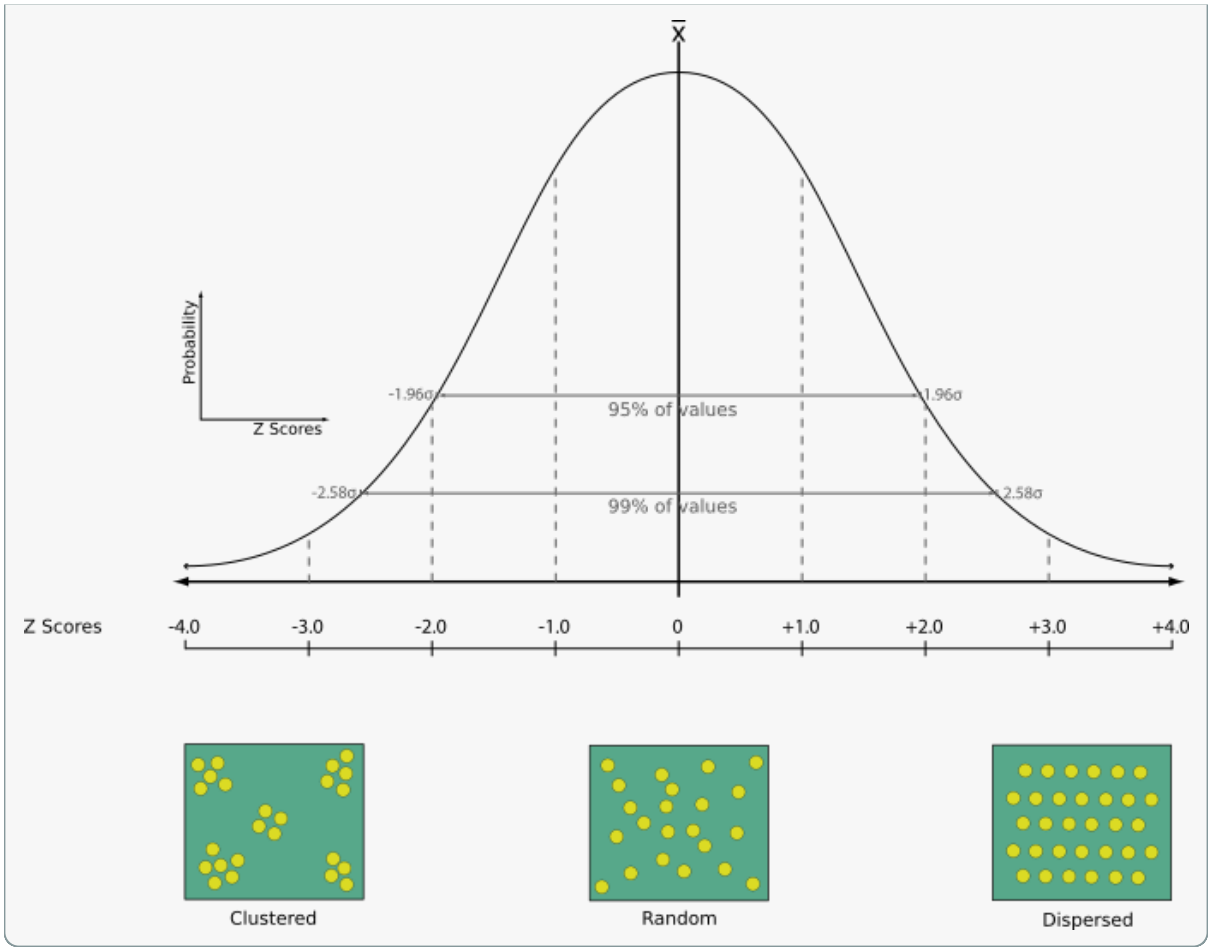
Performs nearest neighbor analysis for a point layer. The output tells you how your data are distributed (clustered, randomly or distributed).

Output is generated as an HTML file with the computed statistical values:

- Observed mean distance
- Expected mean distance
- Nearest neighbour index
- Number of points
- Z-Score: Comparing the Z-Score with the normal distribution tells you how your data are distributed. A low Z-Score means that the data are unlikely to be the result of a spatially random process, while a high Z-Score means that your data are likely to be a result of a spatially random process.

Note

This algorithm uses ellipsoid based measurements and respects the current *ellipsoid settings*.



Default menu: *Vector* ► *Analysis Tools*

See also

Join attributes by nearest

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: point]	Point vector layer to calculate the statistics on
Nearest neighbour Optional	`OUTPUT_HTML_FILE`	[html] Default: `[Save to temporary file]`	Specification of the HTML file for the computed statistics. *One of*: • Skip Output • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Nearest neighbour	`OUTPUT_HTML_FILE`	[html]	HTML file with the computed statistics
Observed mean distance	`OBSERVED_MD`	[numeric: double]	Observed mean distance
Expected mean distance	`EXPECTED_MD`	[numeric: double]	Expected mean distance
Nearest neighbour index	`NN_INDEX`	[numeric: integer]	Nearest neighbour index
Number of points	`POINT_COUNT`	[numeric: integer]	Number of points
Z-Score	`Z_SCORE`	[numeric: double]	Z-Score

Python code

Algorithm ID: `native:nearestneighbouranalysis`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Overlap analysis

Calculates the area and percentage cover by which features from an input layer are overlapped by features from a selection of overlay layers.

New attributes are added to the output layer reporting the total area of overlap and percentage of the input feature overlapped by each of the selected overlay layers.

Note

This algorithm uses ellipsoid based measurements and respects the current *ellipsoid settings*.

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: polygon]	The input layer.
Overlay layers	`LAYERS`	[vector: polygon] [list]	The overlay layers.
Overlap	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Grid size Optional	`GRID_SIZE`	[numeric: double] Default: Not set	If provided, the input geometries are snapped to a grid of the given size, and the result vertices are computed on that same grid. Requires GEOS 3.9.0 or higher.

Outputs

Label	Name	Type	Description
Overlap	`OUTPUT`	[same as input]	The output layer with additional fields reporting the overlap (in map units and percentage) of the input feature overlapped by each of the selected layers.

Python code

Algorithm ID: `native:calculatevectoroverlaps`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Shortest line between features

Creates a line layer as the shortest line between the source and the destination layer. By default only the first nearest feature of the destination layer is taken into account. The n-nearest neighboring features number can be specified. If a maximum distance is specified, then only features which are closer than this distance will be considered.

The output features will contain all the source layer attributes, all the attributes from the n-nearest feature and the additional field of the distance.

Important

This algorithm uses purely Cartesian calculations for distance, and does not consider geodetic or ellipsoid properties when determining feature proximity. The measurement and output coordinate system is based on the coordinate system of the source layer.

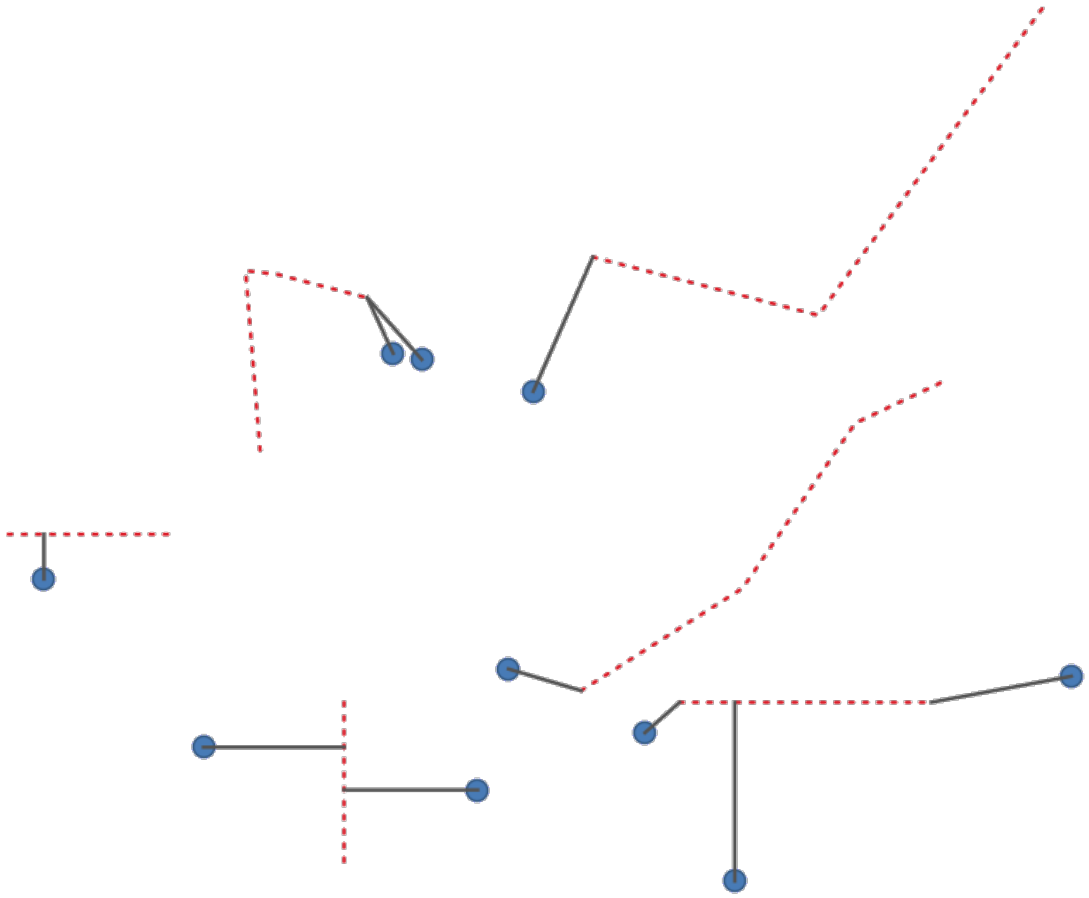


Fig. 24.57: Shortest line from point features to lines

Parameters

Label	Name	Type	Description
Source layer	`SOURCE`	[vector: geometry]	Origin layer for which to search for nearest neighbors
Destination layer	`DESTINATION`	[vector: geometry]	Target Layer in which to search for nearest neighbors
Method	`METHOD`	[enumeration] Default: 0	Shortest distance calculation method Possible values are: • 0 — Distance to nearest point on feature • 1 — Distance to feature centroid
Maximum number of neighbors	`NEIGHBORS`	[numeric: integer] Default: 1	Maximum number of neighbors to look for
Maximum distance Optional	`DISTANCE`	[numeric: double]	Only destination features which are closer than this distance will be considered.
Shortest lines	`OUTPUT`	[vector: line] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Output layer	`OUTPUT`	[vector: line]	Line vector layer joining source features to their nearest neighbor(s) in the destination layer. Contains all attributes for both source and destination features, and the computed distance.

Python code

Algorithm ID: `native:shortestline`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

ST-DBSCAN clustering

Clusters point features based on a 2D implementation of spatiotemporal Density-based clustering of applications with noise (ST-DBSCAN) algorithm.

See also

DBSCAN clustering, *K-means clustering*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: point]	Layer to analyze
Date/time field	`DATE-TIME_FIELD`	[tablefield: date]	Field containing the temporal information
Minimum cluster size	`MIN_SIZE`	[numeric: integer] Default: 5	Minimum number of features to generate a cluster
Maximum distance between clustered points	`EPS`	[numeric: double] Default: 1.0	Distance beyond which two features can not belong to the same cluster (eps)
Maximum time duration between clustered points	`EPS2`	[numeric: double] Default: 0.0 (days)	Time duration beyond which two features can not belong to the same cluster (eps2). Available time units are milliseconds, seconds, minutes, hours, days and weeks.
Clusters	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specify the vector layer for the result of the clustering. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Treat border points as noise (DBSCAN*)	`DBSCAN*`	[boolean] Default: False	If checked, points on the border of a cluster are themselves treated as unclustered points, and only points in the interior of a cluster are tagged as clustered.
Cluster field name	`FIELD_NAME`	[string] Default: 'CLUSTER_ID'	Name of the field where the associated cluster number shall be stored
Cluster size field name	`SIZE_FIELD_NAM`	[string] Default: 'CLUSTER_SIZE'	Name of the field with the count of features in the same cluster

Outputs

Label	Name	Type	Description
Clusters	`OUTPUT`	[vector: point]	Vector layer containing the original features with a field setting the cluster they belong to
Number of clusters	`NUM_CLUSTERS`	[numeric: integer]	The number of clusters discovered

Python code

Algorithm ID: `native:stdbscanclustering`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Statistics by categories

Calculates statistics of a field depending on a parent class. The parent class is a combination of values from other fields.

Parameters

Label	Name	Type	Description
Input vector layer	`INPUT`	[vector: any]	Input vector layer with unique classes and values
Field to calculate statistics on (if empty, only count is calculated) Optional	`VALUES_FIELD_NAME`	[tablefield: any]	If empty only the count will be calculated
Field(s) with categories	`CATEGORIES_FIELD_N`	[tablefield: any] [list]	The fields that (combined) define the categories
Statistics by category	`OUTPUT`	[vector: table] Default: `[Create temporary layer]`	Specify the output table for the generated statistics. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Statistics by category	`OUTPUT`	[vector: table]	Table containing the statistics

Depending on the type of the field being analyzed, the following statistics are returned for each grouped value:

Statistics	String	Numeric	Date
Count (`COUNT`)	✓	✓	✓
Unique values (`UNIQUE`)	✓		✓
Empty (null) values (`EMPTY`)	✓		✓
Non-empty values (`FILLED`)	✓		✓
Minimal value (`MIN`)	✓	✓	✓
Maximal value (`MAX`)	✓	✓	✓
Range (`RANGE`)		✓	
Sum (`SUM`)		✓	
Mean value (`MEAN`)		✓	
Median value (`MEDIAN`)		✓	
Standard Deviation (`STD_DEV`)		✓	
Coefficient of variation (`CV`)		✓	
Minority (rarest occurring value - `MINORITY`)		✓	
Majority (most frequently occurring value - `MAJORITY`)		✓	
First Quartile (`FIRSTQUARTILE`)		✓	
Third Quartile (`THIRDQUARTILE`)		✓	
Inter Quartile Range (`IQR`)		✓	
Minimum Length (`MIN_LENGTH`)	✓		
Mean Length (`MEAN_LENGTH`)	✓		
Maximum Length (`MAX_LENGTH`)	✓		

Python code

Algorithm ID: `qgis:statisticsbycategories`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Sum line lengths

Takes a polygon layer and a line layer and measures the total length of lines and the total number of them that cross each polygon.

The resulting layer has the same features as the input polygon layer, but with two additional attributes containing the length and count of the lines across each polygon.

> **Note**
>
> This algorithm uses ellipsoid based measurements and respects the current *ellipsoid settings*.

Allows *features in-place modification* of polygon features

Default menu: *Vector* ► *Analysis Tools*

Parameters

Label	Name	Type	Description
Lines	`LINES`	[vector: line]	Input vector line layer
Polygons	`POLYGONS`	[vector: polygon]	Polygon vector layer
Lines length field name	`LEN_FIELD`	[string] Default: 'LENGTH'	Name of the field for the lines length
Lines count field name	`COUNT_FIELD`	[string] Default: 'COUNT'	Name of the field for the lines count
Line length	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output polygon layer with generated statistics. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Line length	`OUTPUT`	[vector: polygon]	Polygon output layer with fields of lines length and line count

Python code

Algorithm ID: `native:sumlinelengths`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.23 Vector coverage

Dissolve coverage

Operates on a coverage (represented as a set of polygon features with exactly matching edge geometry) to dissolve (union) the geometries. It provides a heavily optimized approach for unioning these features compared with the standard Dissolve tools.

See also

Dissolve

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: polygon]	Input polygon vector layer
Dissolved	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Dissolved	`OUTPUT`	[vector: polygon]	The output polygon vector layer with dissolved geometries.

Python code

Algorithm ID: `native:coverageunion`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Simplify coverage

Operates on a coverage (represented as a set of polygon features with exactly matching edge geometry) to apply a Visvalingam–Whyatt simplification to the edges, reducing complexity in proportion with the provided tolerance, while retaining a valid coverage (i.e. no edges will cross or touch after the simplification). Geometries will never be removed, but they may be simplified down to just a triangle. Also, some geometries (such as polygons which have too few non-repeated points) will be returned unchanged.

If the input dataset is not a valid coverage due to overlaps, it will still be simplified, but invalid topology such as crossing edges will still be invalid.

Requires version of GEOS >= 3.12

See also

Simplify, *Smooth*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: polygon]	Input polygon vector layer
Tolerance	`TOLERANCE`	[numeric: double] Default: 0.0	The maximum distance (in unit of choice) between two consecutive vertices to get merged.
Preserve boundary	`PRESERVE_BOUNDARY`	[boolean] Default: False	When enabled, the outside edges of the coverage will be preserved without simplification
Simplified	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Simplified	`OUTPUT`	[vector: polygon]	The output polygon vector layer, with a lower number of vertices.

Python code

Algorithm ID: `native:coveragesimplify`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Validate coverage

Analyzes a coverage (represented as a set of polygon features with exactly matching edge geometry) to find places where the assumption of exactly matching edges is not met. Invalidity includes polygons that overlap or that have gaps smaller than the specified gap width.

Requires version of GEOS >= 3.12

See also

Check validity

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: polygon]	Input polygon vector layer
Gap width	`GAP_WIDTH`	[numeric: double] Default: 0.0	The maximum width of gap to detect
Invalid edges	`INVALID_EDGES`	[vector: line] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Invalid edges	`OUTPUT`	[vector: line]	The output line vector layer showing error edges for each invalid polygon.
Validity result	`IS_VALID`	[boolean]	Returns whether the coverage is valid or not.

Python code

Algorithm ID: `native:coveragevalidate`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.24 Vector creation

Array of offset (parallel) lines

Creates copies of line features in a layer, by creating multiple offset versions of each feature. Each new version is incrementally offset by a specified distance.

Positive distance will offset lines to the left, and negative distances will offset them to the right.

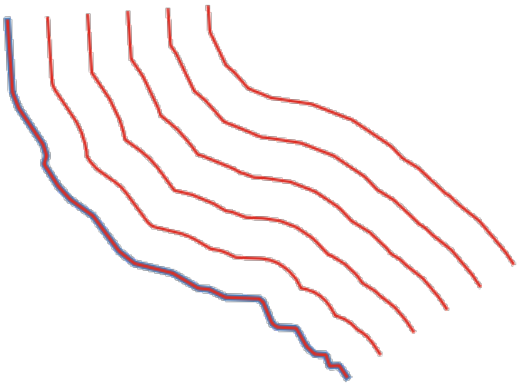


Fig. 24.58: In blue the source layer, in red the offset one

Allows *features in-place modification* of line features

Warning

This algorithm drops existing primary keys or FID values and regenerates them in output layers.

See also

Offset lines, *Array of translated features*

Parameters

Basic parameters

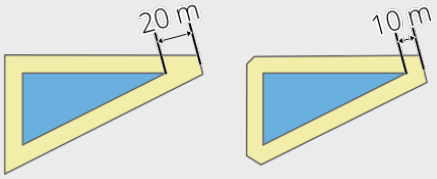
Label	Name	Type	Description
Input layer	`INPUT`	[vector: line]	Input line vector layer to use for the offsets.
Number of features to create	`COUNT`	[numeric: integer] Default: 10	Number of offset copies to generate for each feature
Offset step distance	`OFFSET`	[numeric: double] Default: 1.0	Distance between two consecutive offset copies
Offset lines	`OUTPUT`	[vector: line] Default: `[Create temporary layer]`	Specify the output line layer with offset features. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Segments	`SEGMENTS`	[numeric: double] Default: 8	Number of line segments to use to approximate a quarter circle when creating rounded offsets
Join style	`JOIN_STYLE`	[enumeration] Default: 0	Specify whether round, miter or beveled joins should be used when offsetting corners in a line. One of: • 0 — Round • 1 — Miter • 2 — Bevel  Fig. 24.59: Round, miter, and bevel join styles

continues on next page

Table 24.185 – continued from previous page

Label	Name	Type	Description
Miter limit	`MITER_LIMIT`	[numeric: double] Default: 2.0	Sets the maximum distance from the offset geometry to use when creating a mitered join as a factor of the offset distance (only applicable for miter join styles). Minimum: 1.0  Fig. 24.60: A 10m buffer with a limit of 2 and a 10m buffer with a limit of 1

Outputs

Label	Name	Type	Description
Offset lines	`OUTPUT`	[vector: line]	Output line layer with the original and offset features. Attributes will be copied to the corresponding outputs. The following attributes related to the input parameters will be added: `instance` indexing the `COUNT` offset lines and `offset` designating the offset distance from the original line.

Python code

Algorithm ID: `native:arrayoffsetlines`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Array of translated features

Creates copies of features in a layer by creating multiple translated versions of each. Each copy is incrementally displaced by a preset amount in the X, Y and/or Z axis.

M values present in the geometry can also be translated.

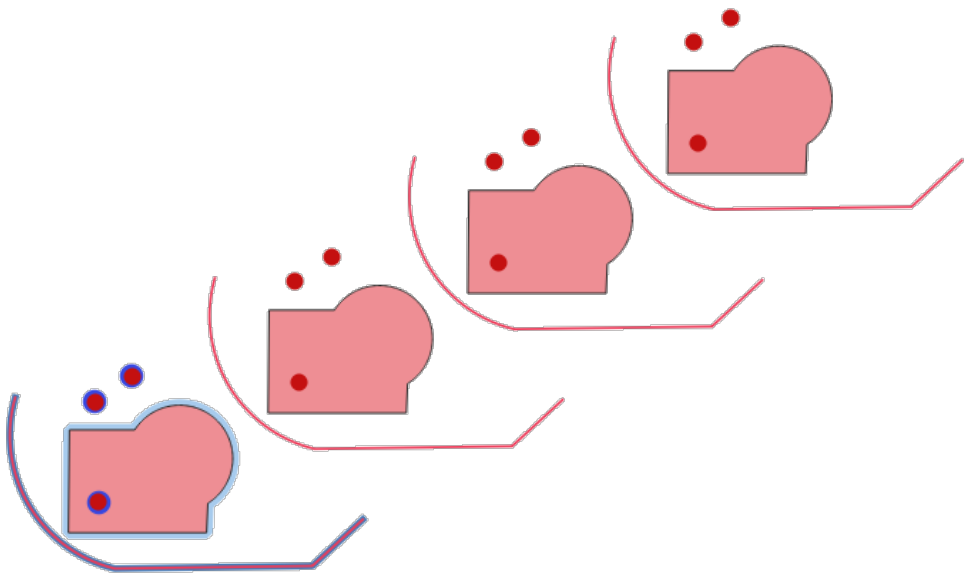


Fig. 24.61: Input layers in blue tones, output layers with translated features in red tones

Allows *features in-place modification* of point, line, and polygon features

Warning

This algorithm drops existing primary keys or FID values and regenerates them in output layers.

See also

Translate, *Array of offset (parallel) lines*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer to translate
Number of features to create	`COUNT`	[numeric: integer] Default: 10	Number of copies to generate for each feature
Step distance (x-axis)	`DELTA_X`	[numeric: double] Default: 0.0	Displacement to apply on the X axis
Step distance (y-axis)	`DELTA_Y`	[numeric: double] Default: 0.0	Displacement to apply on the Y axis
Step distance (z-axis)	`DELTA_Z`	[numeric: double] Default: 0.0	Displacement to apply on the Z axis

continues on next page

Table 24.186 – continued from previous page

Label	Name	Type	Description
Step distance (m values)	DELTA_M	[numeric: double] Default: 0.0	Displacement to apply on M
Translated	OUTPUT	[same as input] Default: [Create temporary layer]	Output vector layer with translated (moved) copies of the features. The original features are also copied. *One of*: • Create Temporary Layer (TEMPORARY_OUTPUT) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Translated	OUTPUT	[same as input]	Output vector layer with translated (moved) copies of the features. The original features are also copied.

Python code

Algorithm ID: native:arraytranslatedfeatures

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Create grid

Creates a vector layer with a grid covering a given extent. Grid cells can have different shapes:

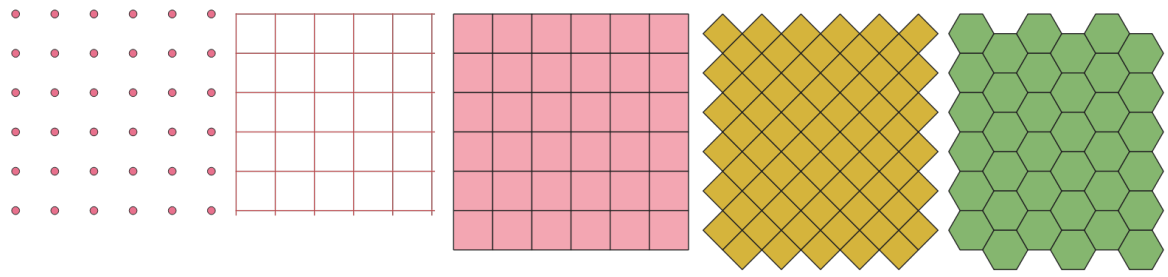


Fig. 24.62: Different grid cell shapes applied to the same extent, without overlaps

The size and/or placement of each element in the grid is defined using a horizontal and vertical spacing. The CRS of the output layer must be defined. The grid extent and the spacing values must be expressed in the coordinates and

units of this CRS.

Default menu: *Vector ► Research Tools*

Parameters

Label	Name	Type	Description
Grid type	`TYPE`	[enumeration] Default: 0	Shape of the grid. One of: • 0 — Point • 1 — Line • 2 — Rectangle (polygon) • 3 — Diamond (polygon) • 4 — Hexagon (polygon)
Grid extent	`EXTENT`	[extent]	Extent of the grid Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Horizontal spacing	`HSPACING`	[numeric: double] Default: 1.0	Size of a grid cell on the X-axis
Vertical spacing	`VSPACING`	[numeric: double] Default: 1.0	Size of a grid cell on the Y-axis
Horizontal overlay	`HOVERLAY`	[numeric: double] Default: 0.0	Overlay distance between two consecutive grid cells on the X-axis
Vertical overlay	`VOVERLAY`	[numeric: double] Default: 0.0	Overlay distance between two consecutive grid cells on the Y-axis
Grid CRS	`CRS`	[crs] Default: *Project CRS*	Coordinate reference system to apply to the grid
Grid	`OUTPUT`	[vector: geometry] Default: `[Create temporary layer]`	Resulting vector grid layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Grid	`OUTPUT`	[vector: geometry]	Resulting vector grid layer. The output geometry type (point, line or polygon) depends on the *Grid type*. Features are created from top to bottom, left to right. The attribute table is filled with: • an `id` • coordinates on the `left`, `right`, `top` and `bottom` sides • and their placement in the grid: `row_index` and `column_index` (available for point, rectangle and hexagon grid types)

Python code

Algorithm ID: `native:creategrid`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Create layer from extent

Creates a new vector layer that contains a single feature with geometry matching the extent of the input layer.

It can be used in models to convert a literal extent (`xmin`, `xmax`, `ymin`, `ymax` format) into a layer which can be used for other algorithms which require a layer based input.

See also

Create layer from point

Parameters

Label	Name	Type	Description
Extent (xmin, xmax, ymin, ymax)	`INPUT`	[extent]	Input extent Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Extent	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Extent	`OUTPUT`	[vector: polygon]	The output (extent) vector layer

Python code

Algorithm ID: `native:extenttolayer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Create layer from point

Creates a new vector layer that contains a single feature with geometry matching a point parameter. It can be used in models to convert a point into a point layer for algorithms which require a layer based input.

See also

Create layer from extent

Parameters

Label	Name	Type	Description
Point	`INPUT`	[coordinates]	Input point, including CRS info (example: `397254,6214446 [EPSG:32632]`). If the CRS is not provided, the Project CRS will be used. The point can be specified by clicking on the map canvas.
Point	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specify the output layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Point	`OUTPUT`	[vector: point]	The output point vector layer containing the input point.

Python code

Algorithm ID: `native:pointtolayer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Create points layer from table

Creates points layer from a table with columns that contain coordinates fields.

Besides X and Y coordinates you can also specify Z and M fields.

> **Warning**
>
> This algorithm drops existing primary keys or FID values and regenerates them in output layers.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	Input vector layer or a table.
X field	`XFIELD`	[tablefield: any]	Field containing the X coordinate
Y field	`YFIELD`	[tablefield: any]	Field containing the Y coordinate
Z field Optional	`ZFIELD`	[tablefield: any]	Field containing the Z coordinate
M field Optional	`MFIELD`	[tablefield: any]	Field containing the M value
Target CRS	`TARGET_CRS`	[crs] Default: `EPSG:4326`	Coordinate reference system to use for layer. The provided coordinates are assumed to be compliant.
Points from table	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specify the resulting point layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Points from table	`OUTPUT`	[vector: point]	The resulting point layer

Python code

Algorithm ID: `native:createpointslayerfromtable`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Generate points (pixel centroids) along line

Generates a point vector layer from an input raster and line layer.

The points correspond to the pixel centroids that intersect the line layer.

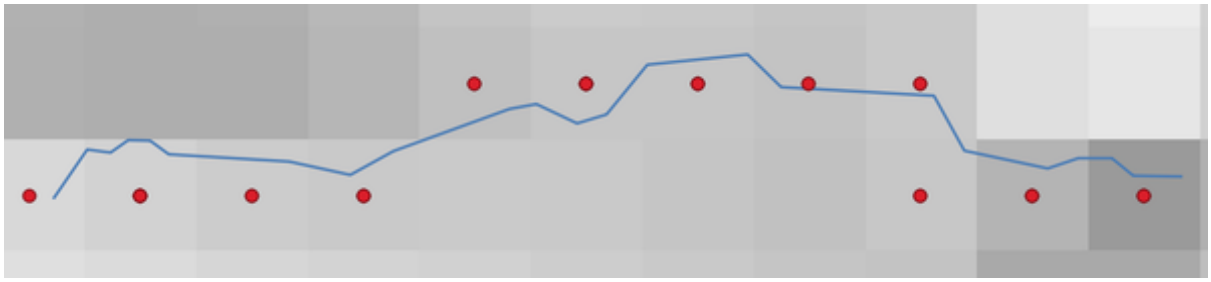


Fig. 24.63: Points of the pixel centroids

Parameters

Label	Name	Type	Description
Raster layer	`INPUT_RASTER`	[raster]	Input raster layer
Vector layer	`INPUT_VECTOR`	[vector: line]	Input line vector layer
Points along line	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Resulting point layer with pixel centroids. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Points along line	`OUTPUT`	[vector: point]	Resulting point layer with pixel centroids

Python code

Algorithm ID: `qgis:generatepointspixelcentroidsalongline`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Generate points (pixel centroids) inside polygon

Generates a point vector layer from an input raster and polygon layer.

The points correspond to the pixel centroids that intersect the polygon layer.

> **Warning**
>
> This algorithm drops existing primary keys or FID values and regenerates them in output layers.

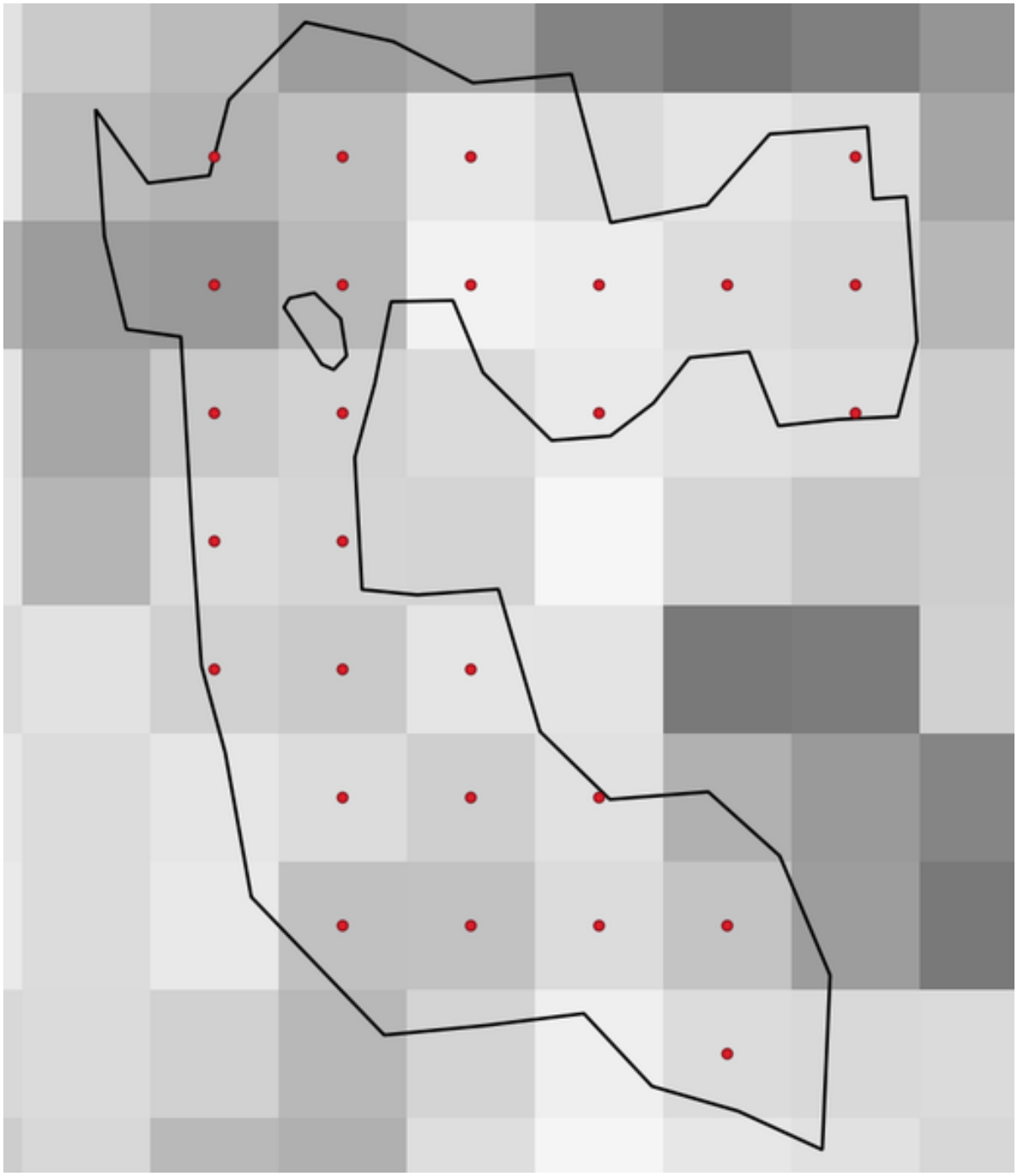


Fig. 24.64: Points of the pixel centroids

Parameters

Label	Name	Type	Description
Raster layer	`INPUT_RASTER`	[raster]	Input raster layer
Vector layer	`INPUT_VECTOR`	[vector: polygon]	Input polygon vector layer
Points inside polygons	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Resulting point layer of pixel centroids. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Points inside polygons	`OUTPUT`	[vector: point]	Resulting point layer of pixel centroids

Python code

Algorithm ID: `native:generatepointspixelcentroidsinsidepolygons`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Import geotagged photos

Creates a point layer corresponding to the geotagged locations from JPEG or HEIC/HEIF images from a source folder.

The point layer will contain a single PointZ feature per input file from which the geotags could be read. Any altitude information from the geotags will be used to set the point's Z value.

Besides longitude and latitude also altitude, direction and timestamp information, if present in the photo, will be added to the point as attributes.

Parameters

Label	Name	Type	Description
Input folder	`FOLDER`	[folder]	Path to the source folder containing the geotagged photos
Scan recursively	`RECURSIVE`	[boolean] Default: False	If checked, the folder and its subfolders will be scanned
Photos Optional	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specify the point vector layer for the geotagged photos. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Invalid photos table Optional	`INVALID`	[vector: table] Default: `[Skip output]`	Specify the table of unreadable or non-geotagged photos. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Photos	`OUTPUT`	[vector: point]	Point vector layer with geotagged photos. The form of the layer is automatically filled with paths and photo previews settings.
Invalid photos table Optional	`INVALID`	[vector: table]	Table of unreadable or non-geotagged photos can also be created.

Python code

Algorithm ID: `native:importphotos`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Points to path

Converts a point layer to a line layer, by joining points in an order defined by an expression or a field in the input point layer.

Points can be grouped by a field or an expression to distinguish line features.

In addition to the line vector layer, a text file is output that describes the resulting line as a start point and a sequence of bearings / directions (relative to azimuth) and distances.

Note

This algorithm uses ellipsoid based measurements and respects the current *ellipsoid settings*.

Parameters

Label	Name	Type	Description
Input point layer	`INPUT`	[vector: point]	Input point vector layer
Create closed paths	`CLOSE_PATH`	[boolean] Default: False	If checked, the first and last points of the line will be connected and close the generated path
Order expression Optional	`ORDER_EXPRESSION`	[expression]	Field or expression providing the order to connect the points in the path. If not set, the feature ID (`$id`) is used.
Sort text containing numbers naturally	`NATURAL_SORT`	[boolean] Default: False	If checked, naturally sorts the features based on the provided expression (i.e., 'a9' < 'a10').
Path group expression Optional	`GROUP_EXPRESSI`	[expression]	Point features of the same value in the field or expression will be grouped in the same line. If not set, a single path is drawn with all the input points.
Paths	`OUTPUT`	[vector: line] Default: `[Create temporary layer]`	Specify the line vector layer of the path. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Directory for text output Optional	`OUTPUT_TEXT_DIR`	[folder] Default: `[Skip output]`	Specify the directory that will contain the description files of points and paths. *One of*: • Skip Output • Save to a Temporary Directory • Save to Directory

Outputs

Label	Name	Type	Description
Paths	`OUTPUT`	[vector: line]	Line vector layer of the path
Directory for text output	`OUTPUT_TEXT_DIR`	[folder]	Directory containing description files of points and paths

Python code

Algorithm ID: `native:pointstopath`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Random points along line

Creates a new point layer, with points placed on the lines of another layer.

For each line in the input layer, a given number of points is added to the resulting layer. The procedure for adding a point is to:

1. randomly select a line feature from the input layer
2. if the feature is multi-part, randomly select a part of it
3. randomly select a segment of that line
4. randomly select a position on that segment.

The procedure means that curved parts of the lines (with relatively short segments) will get more points than straight parts (with relatively long segments), as demonstrated in the illustration below, where the output of the *Random points along lines* algorithm can be compared with the output of the *Random points on lines* algorithm (that produces points with an, on average, even distribution along the lines).

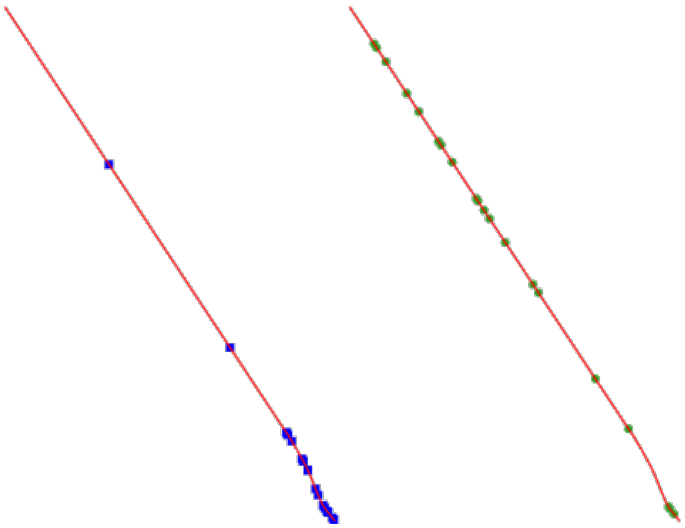


Fig. 24.65: Example algorithm output. Left: *Random points along line*, right: *Random points on lines*

A minimum distance can be specified, to avoid points being too close to each other.

Warning

This algorithm drops existing primary keys or FID values and regenerates them in output layers.

See also

Random points on lines

Parameters

Label	Name	Type	Description
Input point layer	`INPUT`	[vector: line]	Input line vector layer
Number of points	`POINTS_NUMBER`	[numeric: integer] Default: 1	Number of points to create
Minimum distance between points	`MIN_DISTANCE`	[numeric: double] Default: 0.0	The minimum distance between points
Random points	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	The output random points. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Random points	`OUTPUT`	[vector: point]	The output random points layer.

Python code

Algorithm ID: `qgis:qgisrandompointsalongline`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Random points in extent

Creates a new point layer with a given number of random points, all of them within a given extent.

A distance factor can be specified, to avoid points being too close to each other. If the minimum distance between points makes it impossible to create new points, either distance can be decreased or the maximum number of attempts may be increased.

Default menu: *Vector ► Research Tools*

Parameters

Basic parameters

Label	Name	Type	Description
Input extent	`EXTENT`	[extent]	Map extent for the random points Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Number of points	`POINTS_NUMBER`	[numeric: integer] Default: 1	Number of point to create
Minimum distance between points	`MIN_DISTANCE`	[numeric: double] Default: 0.0	The minimum distance between points
Target CRS	`TARGET_CRS`	[crs] Default: *Project CRS*	CRS of the random points layer
Random points	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	The output random points. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Maximum number of search attempts given the minimum distance	`MAX_ATTEMPTS`	[numeric: integer] Default: 200	Maximum number of attempts to place the points

Outputs

Label	Name	Type	Description
Random points	`OUTPUT`	[vector: point]	The output random points layer.

Python code

Algorithm ID: `native:randompointsinextent`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Random points in layer bounds

Creates a new point layer with a given number of random points, all of them within the extent of a given layer.

A minimum distance can be specified, to avoid points being too close to each other.

Default menu: *Vector* ► *Research Tools*

> **Warning**
>
> This algorithm drops existing primary keys or FID values and regenerates them in output layers.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: polygon]	Input polygon layer defining the area
Number of points	`POINTS_NUMBER`	[numeric: integer] Default: 1	Number of points to create
Minimum distance between points	`MIN_DISTANCE`	[numeric: double] Default: 0.0	The minimum distance between points
Random points	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	The output random points. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Random points	`OUTPUT`	[vector: point]	The output random points layer.

Python code

Algorithm ID: `qgis:randompointsinlayerbounds`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Random points in polygons

Creates a point layer with points placed inside the polygons of another layer.

For each feature (polygon / multi-polygon) geometry in the input layer, the given number of points is added to the result layer.

Per feature and global minimum distances can be specified in order to avoid points being too close in the output point layer. If a minimum distance is specified, it may not be possible to generate the specified number of points for each feature. The total number of generated points and missed points are available as output from the algorithm.

The illustration below shows the effect of per feature and global minimum distances and zero/non-zero minimum distances (generated with the same seed, so at least the first point generated will be the same).

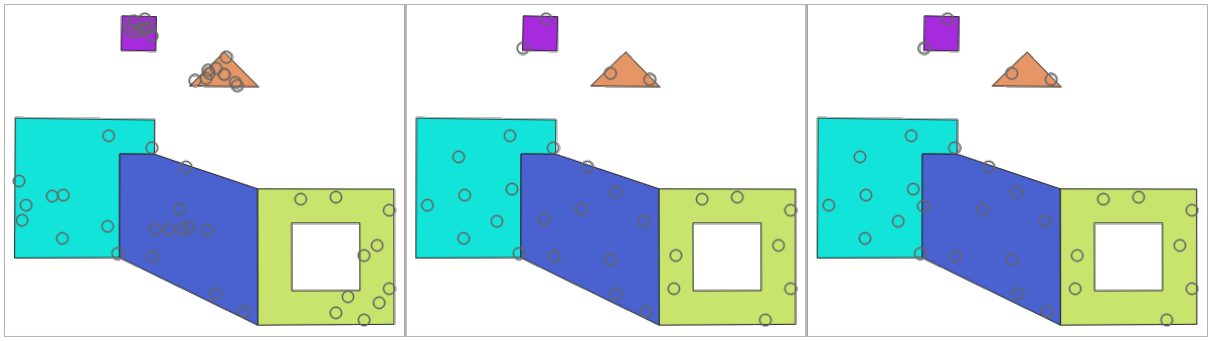


Fig. 24.66: Ten points per polygon feature, *left*: min. distances = 0, *middle*: min.distances = 1, *right*: min. distance = 1, global min. distance = 0

The maximum number of tries per point can be specified. This is only relevant for non-zero minimum distance.

A seed for the random number generator can be provided, making it possible to get identical random number sequences for different runs of the algorithm.

The attributes of the polygon feature on which a point was generated can be included (*Include polygon attributes*).

If you want approximately the same point density for all the features, you can data-define the number of points using the area of the polygon feature geometry.

See also

Random points inside polygons

Parameters

Basic parameters

Label	Name	Type	Description
Input polygon layer	`INPUT`	[vector: line]	Input polygon vector layer
Number of points for each feature	`POINTS_NUMBER`	[numeric: integer] Default: 1	Number of points to create
Minimum distance between points Optional	`MIN_DISTANCE`	[numeric: double] Default: 0.0	The minimum distance between points within one polygon feature
Random points in polygons	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	The output random points. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Global minimum distance between points Optional	MIN_DISTANCE_G	[numeric: double] Default: 0.0	The global minimum distance between points. Should be smaller than the *Minimum distance between points (per feature)* for that parameter to have an effect.
Maximum number of search attempts (for Min. dist. > 0) Optional	MAX_TRIES_PER_	[numeric: integer] Default: 10	The maximum number of tries per point. Only relevant if the minimum distance between points is set (and greater than 0).
Random seed Optional	SEED	[numeric: integer] Default: Not set	The seed to use for the random number generator.
Include polygon attributes	INCLUDE_POLYGON_	[boolean] Default: True	If set, a point will get the attributes from the line on which it is placed.

Outputs

Label	Name	Type	Description
Random points in polygons	OUTPUT	[vector: point]	The output random points layer.
Number of features with empty or no geometry	FEATURES_WITH_EMP	[numeric: integer]	
Total number of points generated	OUTPUT_POINTS	[numeric: integer]	
Number of missed points	POINTS_MISSED	[numeric: integer]	The number of points that could not be generated due to the minimum distance constraint.
Number of features with missed points	POLYGONS_WITH_MISS	[numeric: integer]	Not including features with empty or no geometry

Python code

Algorithm ID: `native:randompointsinpolygons`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Random points inside polygons

Creates a new point layer with a given number of random points inside each polygon of the input polygon layer.

Two sampling strategies are available:

- Points count: number of points for each feature
- Points density: density of points for each feature

A minimum distance can be specified, to avoid points being too close to each other.

Default menu: *Vector* ► *Research Tools*

Warning

This algorithm drops existing primary keys or FID values and regenerates them in output layers.

See also

Random points in polygons

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: polygon]	Input polygon vector layer
Sampling strategy	`STRATEGY`	[enumeration] Default: 0	Sampling strategy to use. One of: • 0 — Points count: number of points for each feature • 1 — Points density: density of points for each feature
Point count or density	`VALUE`	[numeric: double] Default: 1.0	The number or density of points, depending on the chosen *Sampling strategy*.
Minimum distance between points	`MIN_DISTANCE`	[numeric: double] Default: 0.0	The minimum distance between points
Random points	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	The output random points. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Random points	`OUTPUT`	[vector: point]	The output random points layer.

Python code

Algorithm ID: `qgis:randompointsinsidepolygons`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Random points on lines

Creates a point layer with points placed on the lines of another layer.

For each feature (line / multi-line) geometry in the input layer, the given number of points is added to the result layer.

Per feature and global minimum distances can be specified in order to avoid points being too close in the output point layer. If a minimum distance is specified, it may not be possible to generate the specified number of points for each feature. The total number of generated points and missed points are available as output from the algorithm.

The illustration below shows the effect of per feature and global minimum distances and zero/non-zero minimum distances (generated with the same seed, so at least the first point generated will be the same).



Fig. 24.67: Five points per line feature, *left*: min. distances = 0, *middle*: min.distances != 0, *right*: min. distance != 0, global min. distance = 0

The maximum number of tries per point can be specified. This is only relevant for non-zero minimum distance.

A seed for the random number generator can be provided, making it possible to get identical random number sequences for different runs of the algorithm.

The attributes of the line feature on which a point was generated can be included (*Include line attributes*).

If you want approximately the same point density for all the line features, you can data-define the number of points using the length of the line feature geometry.

See also

Random points along line

Parameters

Basic parameters

Label	Name	Type	Description
Input line layer	`INPUT`	[vector: line]	Input line vector layer
Number of points for each feature	`POINTS_NUMBER`	[numeric: integer] Default: 1	Number of points to create

continues on next page

Table 24.196 – continued from previous page

Label	Name	Type	Description
Minimum distance between points (per feature) Optional	MIN_DISTANCE	[numeric: double] Default: 0.0	The minimum distance between points within one line feature
Random points on lines	OUTPUT	[vector: point] Default: [Create temporary layer]	The output random points. *One of*: • Create Temporary Layer (TEMPORARY_OUTPUT) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Global minimum distance between points Optional	MIN_DISTANCE_G	[numeric: double] Default: 0.0	The global minimum distance between points. Should be smaller than the *Minimum distance between points (per feature)* for that parameter to have an effect.
Maximum number of search attempts (for Min. dist. > 0) Optional	MAX_TRIES_PER_	[numeric: integer] Default: 10	The maximum number of tries per point. Only relevant if the minimum distance between points is set (and greater than 0).
Random seed Optional	SEED	[numeric: integer] Default: Not set	The seed to use for the random number generator.
Include line attributes	INCLUDE_LINE_ATT	[boolean] Default: True	If set, a point will get the attributes from the line on which it is placed.

Outputs

Label	Name	Type	Description
Random points on lines	OUTPUT	[vector: point]	The output random points layer.
Number of features with empty or no geometry	FEATURES_WITH_EMP	[numeric: integer]	
Number of features with missed points	LINES_WITH_MIS	[numeric: integer]	Not including features with empty or no geometry
Total number of points generated	POINTS_GENERAT	[numeric: integer]	
Number of missed points	POINTS_MISSED	[numeric: integer]	The number of points that could not be generated due to the minimum distance constraint.

Python code

Algorithm ID: `native:randompointsonlines`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Raster pixels to points

Creates a vector layer of points corresponding to each pixel in a raster layer.

Converts a raster layer to a vector layer, by creating point features for each individual pixel's center in the raster layer. Any NoData pixels are skipped in the output.

Parameters

Label	Name	Type	Description
Raster layer	`INPUT_RASTER`	[raster]	Input raster layer
Band number	`RASTER_BAND`	[raster band]	Raster band to extract data from
Field name	`FIELD_NAME`	[string] Default: 'VALUE'	Name of the field to store the raster band value
Vector points	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specify the resulting point layer of pixels centroids. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Vector points	`OUTPUT`	[vector: point]	Resulting point layer with pixels centroids

Python code

Algorithm ID: `native:pixelstopoints`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Raster pixels to polygons

Creates a vector layer of polygons corresponding to each pixel in a raster layer.

Converts a raster layer to a vector layer, by creating polygon features for each individual pixel's extent in the raster layer. Any NoData pixels are skipped in the output.

Parameters

Label	Name	Type	Description
Raster layer	`INPUT_RASTER`	[raster]	Input raster layer
Band number	`RASTER_BAND`	[raster band]	Raster band to extract data from
Field name	`FIELD_NAME`	[string] Default: 'VALUE'	Name of the field to store the raster band value
Vector polygons	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the resulting polygon layer of pixel extents. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Vector polygons	`OUTPUT`	[vector: polygon]	Resulting polygon layer of pixel extents

Python code

Algorithm ID: `native:pixelstopolygons`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Regular points

Creates a new point layer with its points placed in a regular grid within a given extent.

The grid is specified either by the spacing between the points (same spacing for all dimensions) or by the number of points to generate. In the latter case, the spacing will be determined from the extent. In order to generate a full rectangular grid, at least the number of points specified by the user is generated for the latter case.

Random offsets to the point spacing can be applied, resulting in a non-regular point pattern.

Default menu: *Vector ► Research Tools*

Parameters

Label	Name	Type	Description
Input extent (xmin, xmax, ymin, ymax)	`EXTENT`	[extent]	Map extent for the random points Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Point spacing/count	`SPACING`	[numeric: integer] Default: 100	Spacing between the points, or the number of points, depending on whether `Use point spacing` is checked or not.
Initial inset from corner (LH side)	`INSET`	[numeric: double] Default: 0.0	Offsets the points relative to the upper left corner. The value is used for both the X and Y axis.
Apply random offset to point spacing	`RANDOMIZE`	[boolean] Default: False	If checked the points will have a random spacing
Use point spacing	`IS_SPACING`	[boolean] Default: True	If unchecked the point spacing is not taken into account
Output layer CRS	`CRS`	[crs] Default: *Project CRS*	CRS of the random points layer
Regular points	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specify the output regular point layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Regular points	`OUTPUT`	[vector: point]	The output regular point layer.

Python code

Algorithm ID: `qgis:regularpoints`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.25 Vector general

Assign projection

Assigns a new projection to a vector layer.

It creates a new layer with the exact same features and geometries as the input one, but assigned to a new CRS. The geometries are **not** reprojected, they are just assigned to a different CRS.

This algorithm can be used to repair layers which have been assigned an incorrect projection.

Attributes are not modified by this algorithm.

See also

Define projection, *Find projection*, *Reproject layer*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Vector layer with wrong or missing CRS
Assigned CRS	`CRS`	[crs] Default: `EPSG:4326 - WGS84`	Select the new CRS to assign to the vector layer
Assigned CRS Optional	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Assigned CRS	`OUTPUT`	[same as input]	Vector layer with assigned projection

Python code

Algorithm ID: `native:assignprojection`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Batch Nominatim geocoder

Performs batch geocoding using the Nominatim service against an input layer string field. The output layer will have a point geometry reflecting the geocoded location as well as a number of attributes associated to the geocoded location.

Allows *features in-place modification* of point features

Note

This algorithm is compliant with the usage policy of the Nominatim geocoding service provided by the OpenStreetMap Foundation.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	Vector layer to geocode the features
Address field	`FIELD`	[tablefield: string]	Field containing the addresses to geocode
Geocoded	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specify the output layer containing only the geocoded addresses. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Geocoded	`OUTPUT`	[vector: point]	Vector layer with point features corresponding to the geocoded addresses

Python code

Algorithm ID: `native:batchnominatimgeocoder`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Convert layer to spatial bookmarks

Creates spatial bookmarks corresponding to the extent of features contained in a layer.

Parameters

Label	Name	Type	Description
Input Layer	`INPUT`	[vector: line, polygon]	The input vector layer
Bookmark destination	`DESTINATION`	[enumeration] Default: 0	Select the destination for the bookmarks. One of: • 0 — Project bookmarks • 1 — User bookmarks
Name field	`NAME_EXPRESSIO`	[expression]	Field or expression that will give names to the generated bookmarks
Group field	`GROUP_EXPRESSI`	[expression]	Field or expression that will provide groups for the generated bookmarks

Outputs

Label	Name	Type	Description
Count of bookmarks added	`COUNT`	[numeric: integer]	

Python code

Algorithm ID: `native:layertobookmarks`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Convert spatial bookmarks to layer

Creates a new layer containing polygon features for stored spatial bookmarks. The export can be filtered to only bookmarks belonging to the current project, to all user bookmarks, or a combination of both.

Parameters

Label	Name	Type	Description
Bookmark source	`SOURCE`	[enumeration] [list] Default: [0,1]	Select the source(s) of the bookmarks. One or more of: • 0 — Project bookmarks • 1 — User bookmarks
Output CRS	`CRS`	[crs] Default: `EPSG:4326 - WGS 84`	The CRS of the output layer
Output	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Output	`OUTPUT`	[vector: polygon]	The output (bookmarks) vector layer

Python code

Algorithm ID: `native:bookmarkstolayer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Create attribute index

Creates an index against a field of the attribute table to speed up queries. The support for index creation depends on both the layer's data provider and the field type.

No outputs are created: the index is stored on the layer itself.

Parameters

Label	Name	Type	Description
Input Layer	`INPUT`	[vector: any]	Select the vector layer you want to create an attribute index for
Attribute to index	`FIELD`	[tablefield: any]	Field of the vector layer

Outputs

Label	Name	Type	Description
Indexed layer	`OUTPUT`	[same as input]	A copy of the input vector layer with an index for the specified field

Python code

Algorithm ID: `native:createattributeindex`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Create spatial index

Creates an index to speed up access to the features in a layer based on their spatial location. Support for spatial index creation is dependent on the layer's data provider.

No new output layers are created.

Default menu: *Vector ► Data Management Tools*

Parameters

Label	Name	Type	Description
Input Layer	`INPUT`	[vector: geometry]	Input vector layer

Outputs

Label	Name	Type	Description
Indexed layer	`OUTPUT`	[same as input]	A copy of the input vector layer with a spatial index

Python code

Algorithm ID: `native:createspatialindex`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Define projection

Sets an existing layer's projection to the provided CRS without reprojecting features.

Contrary to the *Assign projection* algorithm, it modifies the current layer and will not output a new layer.

Note

For Shapefile datasets, the `.prj` and `.qpj` files will be overwritten - or created if missing - to match the provided CRS.

Default menu: *Vector ► Data Management Tools*

See also

Assign projection, *Find projection*, *Reproject layer*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Vector layer with missing projection information
CRS	`CRS`	[crs]	Select the CRS to assign to the vector layer

Outputs

Label	Name	Type	Description
	`INPUT`	[same as input]	The input vector layer with the defined projection

Python code

Algorithm ID: `qgis:definecurrentprojection`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Delete duplicate geometries

Finds and removes duplicated geometries.

Attributes are not checked, so in case two features have identical geometries but different attributes, only one of them will be added to the result layer.

Note

This algorithm does not require valid geometries as input.

See also

Drop geometries, *Remove null geometries*, *Delete duplicates by attribute*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	The layer with duplicate geometries you want to clean
Cleaned	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output layer containing the unique features. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Count of discarded duplicate records	`DUPLICATE_COUNT`	[numeric: integer]	Count of discarded duplicate records
Cleaned	`OUTPUT`	[same as input]	The output layer without any duplicated geometries
Count of retained records	`RETAINED_COUNT`	[numeric: integer]	Count of unique records

Python code

Algorithm ID: `native:deleteduplicategeometries`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Delete duplicates by attribute

Deletes duplicate rows by only considering the specified field(s). The first matching row will be retained, and duplicates will be discarded.

Optionally, these duplicate records can be saved to a separate output for analysis.

See also

Delete duplicate geometries

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	The input layer
Fields to match duplicates by	`FIELDS`	[tablefield: any] [list]	Fields defining duplicates. Features with identical values for all these fields are considered duplicates.
Filtered (no duplicates)	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output layer containing the unique features. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Filtered (duplicates) Optional	`DUPLICATES`	[same as input] Default: `[Skip output]`	Specify the output layer containing only the duplicates. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Filtered (duplicates)	`DUPLICATES`	[same as input]	Vector layer containing the duplicate features.
Count of discarded duplicate records	`DUPLICATE_COUNT`	[numeric: integer]	Count of discarded duplicate records
Filtered (no duplicates)	`OUTPUT`	[same as input]	Vector layer containing the unique features.
Count of retained records	`RETAINED_COUNT`	[numeric: integer]	Count of unique records

Python code

Algorithm ID: `native:removeduplicatesbyattribute`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Detect dataset changes

Compares two vector layers, and determines which features are unchanged, added or deleted between the two. It is designed for comparing two different versions of the same dataset.

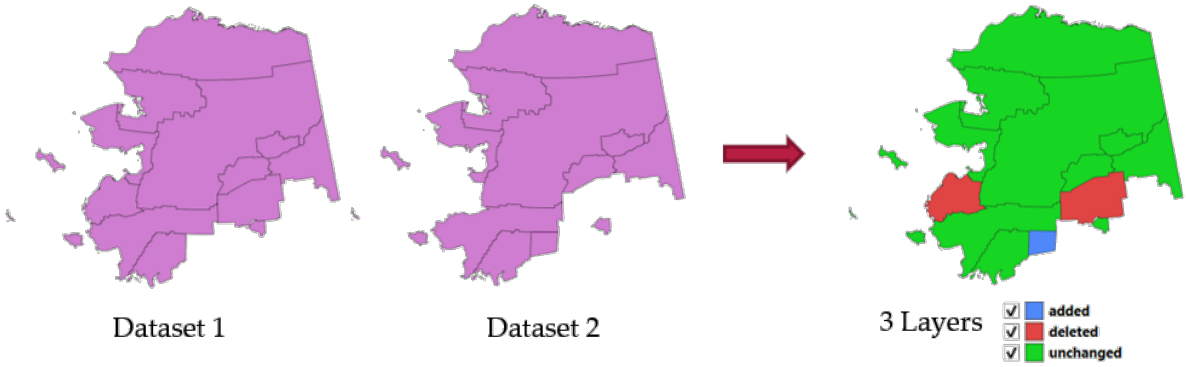


Fig. 24.68: Detect dataset change example

Parameters

Label	Name	Type	Description
Original layer	`ORIGINAL`	[vector: geometry]	The vector layer considered as the original version
Revised layer	`REVISED`	[vector: geometry]	The revised or modified vector layer
Attributes to consider for match Optional	`COMPARE_ATTRIBUTE`	[tablefield: any] [list]	Attributes to consider for match. By default, all attributes are compared.
Geometry comparison behavior Optional	`MATCH_TYPE`	[enumeration] Default: 1	Defines the criteria for comparison. Options: • 0 — Exact Match: includes the order and vertices count of geometries • 1 — Tolerant Match (Topological Equality): geometries are considered equal
Unchanged features Optional	`UNCHANGED`	[vector: same as Original layer]	Specify the output vector layer containing the unchanged features. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

continues on next page

Table 24.203 – continued from previous page

Label	Name	Type	Description
Added features Optional	ADDED	[vector: same as Original layer]	Specify the output vector layer containing the added features. *One of*: • Skip Output • Create Temporary Layer (TEMPORARY_OUTPUT) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Deleted features Optional	DELETED	[vector: same as Original layer]	Specify the output vector layer containing the deleted features. *One of*: • Skip Output • Create Temporary Layer (TEMPORARY_OUTPUT) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Unchanged features	UNCHANGED	[vector: same as Original layer]	Vector layer containing the unchanged features.
Added features	ADDED	[vector: same as Original layer]	Vector layer containing the added features.
Deleted features	DELETED	[vector: same as Original layer]	Vector layer containing the deleted features.
Count of unchanged features	UNCHANGED_COUNT	[numeric: integer]	Count of unchanged features.
Count of features added in revised layer	ADDED_COUNT	[numeric: integer]	Count of features added in revised layer.
Count of features deleted from original layer	DELETED_COUNT	[numeric: integer]	Count of features deleted from original layer.

Python code

Algorithm ID: `native:detectvectorchanges`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Drop geometries

Creates a simple *geometryless* copy of the input layer attribute table. It keeps the attribute table of the source layer.

If the file is saved in a local folder, you can choose between many file formats.

Allows *features in-place modification* of point, line, and polygon features

See also

Delete duplicate geometries, *Remove null geometries*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	The input vector layer
Dropped geometries	`OUTPUT`	[vector: table]	Specify the output geometryless layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Dropped geometries	`OUTPUT`	[vector: table]	The output geometryless layer. A copy of the original attribute table.

Python code

Algorithm ID: `native:dropgeometries`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Execute SQL

Runs a simple or complex query based only on SELECT with `SQL` syntax on the source layer.

Input datasources are identified with `input1`, `input2`... `inputN` and a simple query will look like `SELECT * FROM input1`.

Beside a simple query, you can add expressions or variables within the `SQL query` parameter itself. This is particulary useful if this algorithm is executed within a Processing model and you want to use a model input as a parameter of the query. An example of a query will then be `SELECT * FROM [% @table %]` where `@table` is the variable that identifies the model input.

The result of the query will be added as a new layer.

See also

SpatiaLite execute SQL, *PostgreSQL execute SQL*

Parameters

Label	Name	Type	Description
Additional input datasources (called input1, .., inputN in the query)	`INPUT_DATASOURCE`	[vector: any] [list]	List of layers to query. In the SQL editor you can refer these layers with their **real** name or also with **input1**, **input2**, **inputN** depending on how many layers have been chosen.
SQL query	`INPUT_QUERY`	[string]	Type the string of your SQL query, e.g. `SELECT * FROM input1`.
Unique identifier field Optional	`INPUT_UID_FIELD`	[string]	Specify the column with unique ID
Geometry field Optional	`INPUT_GEOMETRY_F`	[string]	Specify the geometry field
Geometry type Optional	`INPUT_GEOMETRY_T`	[enumeration] Default: 0	Choose the geometry of the result. By default the algorithm will autodetect it. One of: • 0 — Autodetect • 1 — No geometry • 2 — Point • 3 — LineString • 4 — Polygon • 5 — MultiPoint • 6 — MultiLineString • 7 — MultiPolygon
CRS Optional	`INPUT_GEOMETRY_C`	[crs]	The CRS to assign to the output layer
SQL Output	`OUTPUT`	[vector: geometry] Default: `[Create temporary layer]`	Specify the output layer created by the query. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
SQL Output	`OUTPUT`	[vector: geometry]	Vector layer created by the query

Python code

Algorithm ID: `qgis:executesql`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Export layers to DXF

Exports layers to DXF file. For each layer, you can choose a field whose values are used to split features in generated destination layers in DXF output.

See also

Creating new DXF files

Parameters

Label	Name	Type	Description
Input layers	LAYERS	[vector: geometry] [list]	List of input vector layers with options associated (filled as a QgsProcessingParameterDxfLayers item — done in GUI by pressing *Configure Layer…* button for each selected layer): **Layer [string] (layer)** Full path of the input layer to export **Output layer attribute [tablefield: any] (attributeIndex)** Attribute index to split the input layer using unique values **Output layer name [string] (overriddenLayerName)** Overridden layer name to be used in the exported DXF file (the original project layer remains untouched) **Allow data defined symbol blocks** [boolean] (buildDataDefinedBlocks) **Maximum number of symbol blocks [numeric: integer] (dataDefinedBlocksMaximumNumberOfClasses)** −1 means no limitation.
Symbology mode	SYMBOLOGY_MODE	[enumeration] Default: 0	Type of symbology to apply to output layers. You can choose between: • 0 — No Symbology • 1 — Feature Symbology • 2 — Symbol Layer Symbology
Symbology scale	SYMBOLOGY_SCALE	[scale] Default: 1:1 000 000	Default scale of data export.
Map theme Optional	MAP_THEME	[map theme]	Match layer styling to the provided map theme.
Encoding	ENCODING	[enumeration]	Encoding to apply to layers.
CRS	CRS	[crs]	Choose the CRS for the output layer.
Extent Optional	EXTENT	[extent]	Limit exported features to those with geometries intersecting the provided extent.
Use layer title as name	USE_LAYER_TITL	[boolean] Default: False	Name the output layer with the layer title (as set in layer metadata or QGIS Server properties) instead of the layer name.
Force 2D	FORCE_2D	[boolean] Default: False	
Export labels as MTEXT elements	MTEXT	[boolean] Default: True	Exports labels as MTEXT or TEXT elements
Use only selected features	SELECTED_FEATURE	[boolean] Default: False	Exports only the selected features.
DXF	OUTPUT	[file] Default: [Save to temporary file]	Specification of the output DXF file. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
DXF	`OUTPUT`	[file]	`.DXF` file containing the input layers

Python code

Algorithm ID: `native:dxfexport`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Extract selected features

Saves the selected features as a new layer.

> **Note**
>
> If the selected layer has no selected features, the newly created layer will be empty.

Parameters

Label	Name	Type	Description
Input Layer	`INPUT`	[vector: any]	Layer to save the selection from
Selected features	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the vector layer for the selected features. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Selected features	`OUTPUT`	[same as input]	Vector layer with only the selected features, or no feature if none was selected.

Python code

Algorithm ID: `native:saveselectedfeatures`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Extract Shapefile encoding

Extracts the attribute encoding information embedded in a Shapefile. Both the encoding specified by an optional `.cpg` file and any encoding details present in the `.dbf` LDID header block are considered.

Parameters

Label	Name	Type	Description
Input Layer	`INPUT`	[vector: geometry]	ESRI Shapefile (`.SHP`) Layer to extract the encoding information.

Outputs

Label	Name	Type	Description
Shapefile encoding	`ENCODING`	[string]	Encoding information specified in the input file
CPG encoding	`CPG_ENCODING`	[string]	Encoding information specified in any optional `.CPG` file
LDID encoding	`LDID_ENCODING`	[string]	Encoding information specified in `.dbf` LDID header block

Python code

Algorithm ID: `native:shpencodinginfo`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Find projection

Creates a shortlist of candidate coordinate reference systems, for instance for a layer with an unknown projection.

The area that the layer is expected to cover must be specified via the target area parameter. The coordinate reference system for this target area must be known to QGIS.

The algorithm operates by testing the layer's extent in every known reference system and then listing any for which the bounds would be near the target area if the layer was in this projection.

See also

Assign projection, *Define projection*, *Reproject layer*

Parameters

Label	Name	Type	Description
Input Layer	`INPUT`	[vector: geometry]	Layer with unknown projection
Target area for layer (xmin, xmax, ymin, ymax)	`TARGET_AREA`	[extent]	The area that the layer covers. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
CRS candidates	`OUTPUT`	[vector: table] Default: `[Create temporary layer]`	Specify the table (geometryless layer) for the CRS suggestions (EPSG codes). *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
CRS candidates	`OUTPUT`	[vector: table]	A table with all the CRS (EPSG codes) of the matching criteria.

Python code

Algorithm ID: `qgis:findprojection`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Flatten relationship

Flattens a *relationship* for a vector layer, exporting a single layer containing one parent feature per related child feature. This master feature contains all the attributes for the related features. This allows to have the relation as a plain table that can be e.g. exported to CSV.

> **Warning**
>
> This algorithm drops existing primary keys or FID values and regenerates them in output layers.

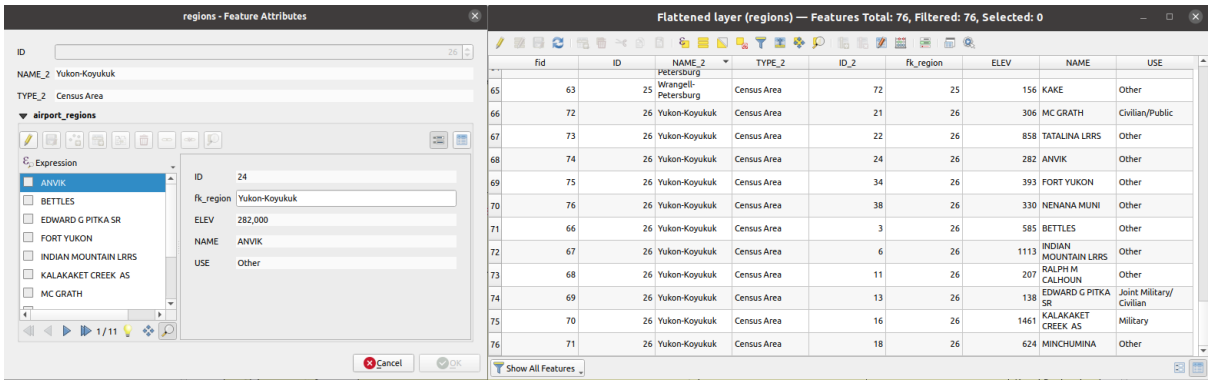


Fig. 24.69: Form of a region with related children (left) - A duplicate region feature for each related child, with joined attributes (right)

Parameters

Label	Name	Type	Description
Input Layer	`INPUT`	[vector: any]	Layer with the relationship that should be de-normalized
Flattened Layer Optional	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output (flattened) layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Flattened layer	`OUTPUT`	[same as input]	A layer containing master features with all the attributes for the related features

Python code

Algorithm ID: `native:flattenrelationships`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Join attributes by field value

Takes an input vector layer and creates a new vector layer that is an extended version of the input one, with additional attributes in its attribute table.

The additional attributes and their values are taken from a second vector layer. An attribute is selected in each of them to define the join criteria.

Warning

This algorithm drops existing primary keys or FID values and regenerates them in output layers.

See also

Join attributes by nearest, *Join attributes by location*

Parameters

Label	Name	Type	Description
Input Layer	`INPUT`	[vector: any]	Input vector layer. The output layer will consist of the features of this layer with attributes from matching features in the second layer.
Table field	`FIELD`	[tablefield: any]	Field of the source layer to use for the join
Input layer 2	`INPUT_2`	[vector: any]	Layer with the attribute table to join
Table field 2	`FIELD_2`	[tablefield: any]	Field of the second (join) layer to use for the join The type of the field must be equal to (or compatible with) the input table field type.
Layer 2 fields to copy Optional	`FIELDS_TO_COPY`	[tablefield: any] [list]	Select the specific fields you want to add. By default all the fields are added.
Join type	`METHOD`	[enumeration] Default: 1	The type of the final joined layer. One of: • 0 — Create separate feature for each matching feature (one-to-many) • 1 — Take attributes of the first matching feature only (one-to-one)
Discard records which could not be joined	`DISCARD_NONMATCHI`	[boolean] Default: True	Check if you don't want to keep the features that could not be joined
Joined field prefix Optional	`PREFIX`	[string]	Add a prefix to joined fields in order to easily identify them and avoid field name collision
Joined layer Optional	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer for the join. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Unjoinable features from first layer Optional	`NON_MATCHING`	[same as input] Default: `[Skip output]`	Specify the output vector layer for unjoinable features from first layer. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Number of joined features from input table	`JOINED_COUNT`	[numeric: integer]	
Unjoinable features from first layer Optional	`NON_MATCHING`	[same as input]	Vector layer with the non-matched features
Joined layer Optional	`OUTPUT`	[same as input]	Output vector layer with added attributes from the join
Number of unjoinable features from input table Optional	`UNJOINABLE_COUNT`	[numeric: integer]	

Python code

Algorithm ID: `native:joinattributestable`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Join attributes by location

Takes an input vector layer and creates a new vector layer that is an extended version of the input one, with additional attributes in its attribute table.

The additional attributes and their values are taken from a second vector layer. A spatial criteria is applied to select the values from the second layer that are added to each feature from the first layer.

Default menu: *Vector ► Data Management Tools*

Warning

This algorithm drops existing primary keys or FID values and regenerates them in output layers.

See also

Join attributes by nearest, *Join attributes by field value*, *Join attributes by location (summary)*

Exploring spatial relations

Geometric predicates are boolean functions used to determine the spatial relation a feature has with another by comparing whether and how their geometries share a portion of space.

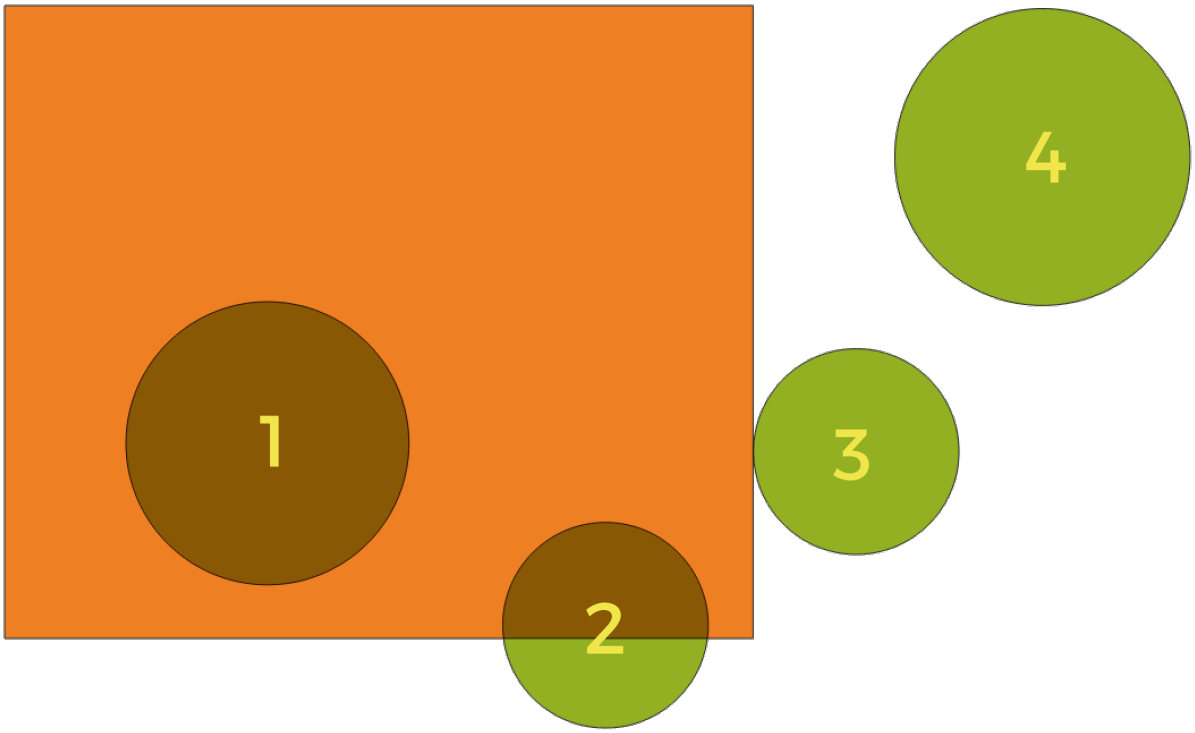


Fig. 24.70: Looking for spatial relations between layers

Using the figure above, we are looking for the green circles by spatially comparing them to the orange rectangle feature. Available geometric predicates are:

Intersect
: Tests whether a geometry intersects another. Returns 1 (true) if the geometries spatially intersect (share any portion of space - overlap or touch) and 0 if they don't. In the picture above, this will return circles 1, 2 and 3.

Contain
: Returns 1 (true) if and only if no points of b lie in the exterior of a, and at least one point of the interior of b lies in the interior of a. In the picture, no circle is returned, but the rectangle would be if you would look for it the other way around, as it contains circle 1 completely. This is the opposite of *are within*.

Disjoint
: Returns 1 (true) if the geometries do not share any portion of space (no overlap, not touching). Only circle 4 is returned.

Equal
: Returns 1 (true) if and only if geometries are exactly the same. No circles will be returned.

Touch
: Tests whether a geometry touches another. Returns 1 (true) if the geometries have at least one point in common, but their interiors do not intersect. Only circle 3 is returned.

Overlap
: Tests whether a geometry overlaps another. Returns 1 (true) if the geometries share space, are of the same dimension, but are not completely contained by each other. Only circle 2 is returned.

Are within
: Tests whether a geometry is within another. Returns 1 (true) if geometry a is completely inside geometry b. Only circle 1 is returned.

Cross

Returns 1 (true) if the supplied geometries have some, but not all, interior points in common and the actual crossing is of a lower dimension than the highest supplied geometry. For example, a line crossing a polygon will cross as a line (true). Two lines crossing will cross as a point (true). Two polygons cross as a polygon (false). In the picture, no circles will be returned.

Parameters

Label	Name	Type	Description
Join to features in	`INPUT`	[vector: geometry]	Input vector layer. The output layer will consist of the features of this layer with attributes from matching features in the second layer.
Where the features	`PREDICATE`	[enumeration] [list] Default: [0]	Type of spatial relation the source feature should have with the target feature so that they could be joined. One or more of: • 0 — intersect • 1 — contain • 2 — equal • 3 — touch • 4 — overlap • 5 — are within • 6 — cross If more than one condition is chosen, at least one of them (OR operation) has to be met for a feature to be extracted.
By comparing to	`JOIN`	[vector: geometry]	The join layer. Features of this vector layer will **add** their attributes to the source layer attribute table if they satisfy the spatial relationship.
Fields to add (leave empty to use all fields) Optional	`JOIN_FIELDS`	[tablefield: any] [list]	Select the specific fields you want to add from the join layer. By default all the fields are added.
Join type	`METHOD`	[enumeration]	The type of the final joined layer. One of: • 0 — Create separate feature for each matching feature (one-to-many) • 1 — Take attributes of the first matching feature only (one-to-one) • 2 — Take attributes of the feature with largest overlap only (one-to-one)
Discard records which could not be joined	`DISCARD_NONMATCHI`	[boolean] Default: False	Remove from the output the input layer's features which could not be joined
Joined field prefix Optional	`PREFIX`	[string]	Add a prefix to joined fields in order to easily identify them and avoid field name collision

continues on next page

Table 24.210 – continued from previous page

Label	Name	Type	Description
Joined layer Optional	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer for the join. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Unjoinable features from first layer Optional	`NON_MATCHING`	[same as input] Default: `[Skip output]`	Specify the output vector layer for unjoinable features from first layer. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Number of joined features from input table	`JOINED_COUNT`	[numeric: integer]	
Unjoinable features from first layer Optional	`NON_MATCHING`	[same as input]	Vector layer of the non-matched features
Joined layer	`OUTPUT`	[same as input]	Output vector layer with added attributes from the join

Python code

Algorithm ID: `native:joinattributesbylocation`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Join attributes by location (summary)

Takes an input vector layer and creates a new vector layer that is an extended version of the input one, with additional attributes in its attribute table.

The additional attributes and their values are taken from a second vector layer. A spatial criteria is applied to select the values from the second layer that are added to each feature from the first layer.

The algorithm calculates a statistical summary for the values from matching features in the second layer (e.g. maximum value, mean value, etc).

See also

Join attributes by location

Exploring spatial relations

Geometric predicates are boolean functions used to determine the spatial relation a feature has with another by comparing whether and how their geometries share a portion of space.

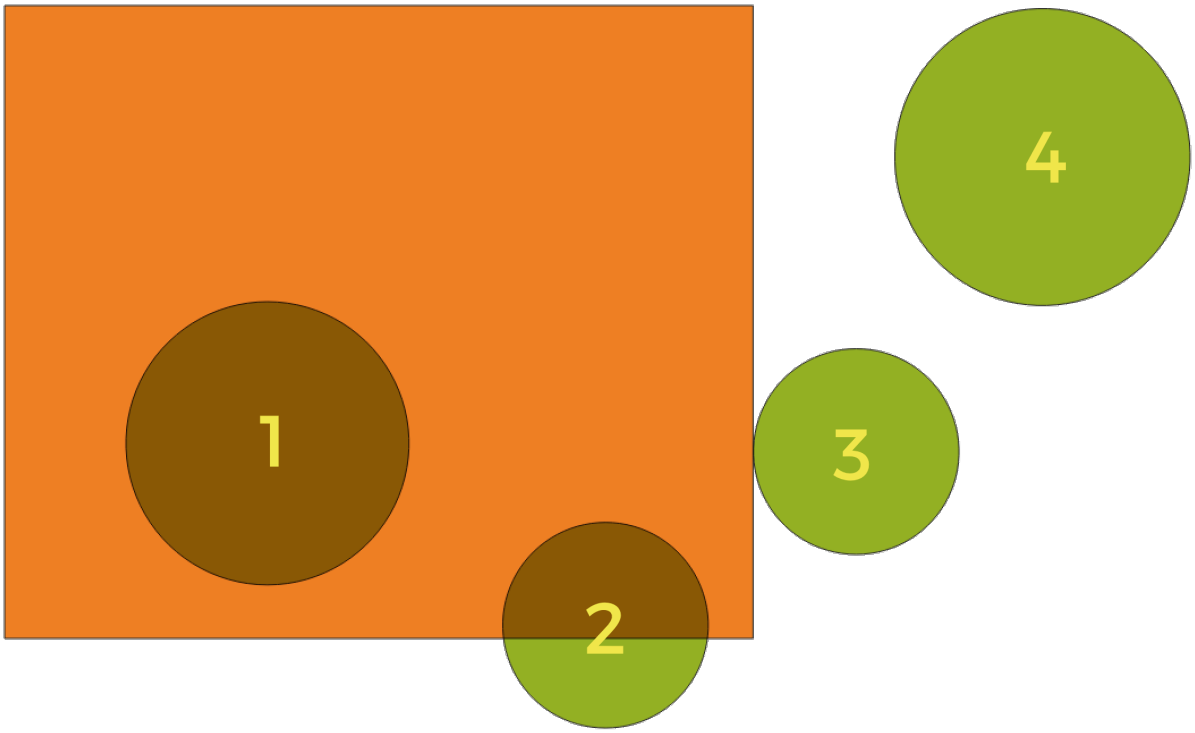


Fig. 24.71: Looking for spatial relations between layers

Using the figure above, we are looking for the green circles by spatially comparing them to the orange rectangle feature. Available geometric predicates are:

Intersect
: Tests whether a geometry intersects another. Returns 1 (true) if the geometries spatially intersect (share any portion of space - overlap or touch) and 0 if they don't. In the picture above, this will return circles 1, 2 and 3.

Contain
: Returns 1 (true) if and only if no points of b lie in the exterior of a, and at least one point of the interior of b lies in the interior of a. In the picture, no circle is returned, but the rectangle would be if you would look for it the other way around, as it contains circle 1 completely. This is the opposite of *are within*.

Disjoint
Returns 1 (true) if the geometries do not share any portion of space (no overlap, not touching). Only circle 4 is returned.

Equal
Returns 1 (true) if and only if geometries are exactly the same. No circles will be returned.

Touch
Tests whether a geometry touches another. Returns 1 (true) if the geometries have at least one point in common, but their interiors do not intersect. Only circle 3 is returned.

Overlap
Tests whether a geometry overlaps another. Returns 1 (true) if the geometries share space, are of the same dimension, but are not completely contained by each other. Only circle 2 is returned.

Are within
Tests whether a geometry is within another. Returns 1 (true) if geometry a is completely inside geometry b. Only circle 1 is returned.

Cross
Returns 1 (true) if the supplied geometries have some, but not all, interior points in common and the actual crossing is of a lower dimension than the highest supplied geometry. For example, a line crossing a polygon will cross as a line (true). Two lines crossing will cross as a point (true). Two polygons cross as a polygon (false). In the picture, no circles will be returned.

Parameters

Label	Name	Type	Description
Join to features in	`INPUT`	[vector: geometry]	Input vector layer. The output layer will consist of the features of this layer with attributes from matching features in the second layer.
Where the features	`PREDICATE`	[enumeration] [list] Default: [0]	Type of spatial relation the source feature should have with the target feature so that they could be joined. One or more of: • 0 — intersect • 1 — contain • 2 — equal • 3 — touch • 4 — overlap • 5 — are within • 6 — cross If more than one condition is chosen, at least one of them (OR operation) has to be met for a feature to be extracted.
By comparing to	`JOIN`	[vector: geometry]	The join layer. Features of this vector layer will **add** summaries of their attributes to the source layer attribute table if they satisfy the spatial relationship.
Fields to summarize (leave empty to use all fields) Optional	`JOIN_FIELDS`	[tablefield: any] [list]	Select the specific fields you want to add from the join layer. By default all the fields are added.

continues on next page

Table 24.211 – continued from previous page

Label	Name	Type	Description
Summaries to calculate (leave empty to use all fields) Optional	SUMMARIES	[enumeration] [list] Default: []	For each input feature, statistics are calculated on joined fields of their matching features. One or more of: • 0 — count • 1 — unique • 2 — min • 3 — max • 4 — range • 5 — sum • 6 — mean • 7 — median • 8 — stddev • 9 — minority • 10 — majority • 11 — q1 • 12 — q3 • 13 — iqr • 14 — empty • 15 — filled • 16 — min_length • 17 — max_length • 18 — mean_length
Discard records which could not be joined	DISCARD_NONMATCHI	[boolean] Default: False	Remove from the output the input layer's features which could not be joined
Joined layer	OUTPUT	[same as input] Default: [Create temporary layer]	Specify the output vector layer for the join. *One of*: • Create Temporary Layer (TEMPORARY_OUTPUT) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Joined layer	OUTPUT	[same as input]	Output vector layer with summarized attributes from the join

Python code

Algorithm ID: `qgis:joinbylocationsummary`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Join attributes by nearest

Takes an input vector layer and creates a new vector layer with additional fields in its attribute table. The additional attributes and their values are taken from a second vector layer. Features are joined by finding the closest features from each layer.

By default only the nearest feature is joined, but the join can also join to the k-nearest neighboring features.

If a maximum distance is specified, only features which are closer than this distance will be matched.

Warning

This algorithm drops existing primary keys or FID values and regenerates them in output layers.

See also

Nearest neighbour analysis, *Join attributes by field value*, *Join attributes by location*, *Distance matrix*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	The input layer.
Input layer 2	`INPUT_2`	[vector: geometry]	The join layer.
Layer 2 fields to copy (leave empty to copy all fields)	`FIELDS_TO_COPY`	[fields]	Join layer fields to copy (if empty, all fields will be copied).
Discard records which could not be joined	`DISCARD_NONMATCHI`	[boolean] Default: False	Remove from the output the input layer records which could not be joined
Joined field prefix	`PREFIX`	[string]	Joined field prefix
Maximum nearest neighbors	`NEIGHBORS`	[numeric: integer] Default: 1	Maximum number of nearest neighbors
Maximum distance	`MAX_DISTANCE`	[numeric: double]	Maximum search distance

continues on next page

Table 24.212 – continued from previous page

Label	Name	Type	Description
Joined layer Optional	OUTPUT	[same as input] Default: [Create temporary layer]	Specify the vector layer containing the joined features. *One of*: • Skip Output • Create Temporary Layer (TEMPORARY_OUTPUT) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Unjoinable features from first layer	NON_MATCHING	[same as input] Default: [Skip output]	Specify the vector layer containing the features that could not be joined. *One of*: • Skip Output • Create Temporary Layer (TEMPORARY_OUTPUT) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Joined layer	OUTPUT	[same as input]	The output joined layer.
Unjoinable features from first layer	NON_MATCHING	[same as input]	Layer containing the features from first layer that could not be joined to any features in the join layer.
Number of joined features from input table	JOINED_COUNT	[numeric: integer]	Number of features from the input table that have been joined.
Number of unjoinable features from input table	UNJOINABLE_COUNT	[numeric: integer]	Number of features from the input table that could not be joined.

Python code

Algorithm ID: native:joinbynearest

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

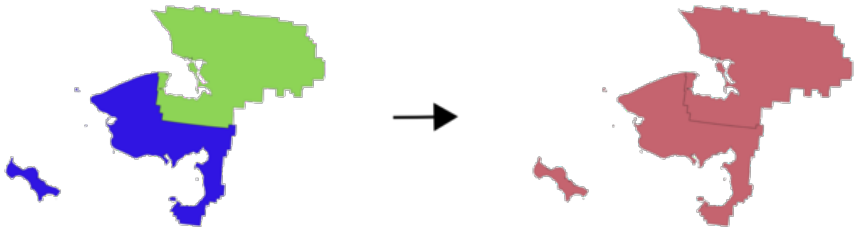
Merge vector layers

Combines multiple vector layers of the **same geometry** type into a single one.

The attribute table of the resulting layer will contain the fields from all input layers. If fields with the same name but different types are found then the exported field will be automatically converted into a string type field. Optionally, new fields storing the original layer name and source can be added.

If any input layers contain Z or M values, then the output layer will also contain these values. Similarly, if any of the input layers are multi-part, the output layer will also be a multi-part layer.

Optionally, the destination coordinate reference system (CRS) for the merged layer can be set. If it is not set, the CRS will be taken from the first input layer. All layers will be reprojected to match this CRS.



Default menu: *Vector ► Data Management Tools*

Warning

This algorithm drops existing primary keys or FID values and regenerates them in output layers.

See also

Split vector layer

Parameters

Label	Name	Type	Description
Input Layers	`LAYERS`	[vector: any] [list]	The layers that are to be merged into a single layer. Layers should be of the same geometry type.
Destination CRS Optional	`CRS`	[crs]	Choose the CRS for the output layer. If not specified, the CRS of the first input layer is used.
Add source layer information (layer name and path)	`ADD_SOURCE_FIE`	[boolean] Default: True	Add fields storing the original layer name and path
Merged	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Merged	`OUTPUT`	[same as input]	Output vector layer containing all the features and attributes from the input layers.

Python code

Algorithm ID: `native:mergevectorlayers`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Order by expression

Sorts a vector layer according to an expression: changes the feature index according to an expression.

Be careful, it might not work as expected with some providers, the order might not be kept every time.

Warning

This algorithm drops existing primary keys or FID values and regenerates them in output layers.

Parameters

Label	Name	Type	Description
Input Layer	`INPUT`	[vector: any]	Input vector layer to sort
Expression	`EXPRESSION`	[expression]	Expression to use for the sorting
Sort ascending	`ASCENDING`	[boolean] Default: True	If checked the vector layer will be sorted from small to large values.
Sort nulls first	`NULLS_FIRST`	[boolean] Default: False	If checked, Null values are placed first
Ordered	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Ordered	`OUTPUT`	[same as input]	Output (sorted) vector layer

Python code

Algorithm ID: `native:orderbyexpression`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Repair Shapefile

Repairs a broken ESRI Shapefile dataset by (re)creating the SHX file.

Parameters

Label	Name	Type	Description
Input Shapefile	`INPUT`	[file]	Full path to the ESRI Shapefile dataset with a missing or broken SHX file

Outputs

Label	Name	Type	Description
Repaired layer	`OUTPUT`	[vector: geometry]	The input vector layer with the SHX file repaired

Python code

Algorithm ID: `native:repairshapefile`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Reproject layer

Reprojects a vector layer in a different CRS. The reprojected layer will have the same features and attributes of the input layer.

Allows *features in-place modification* of point, line, and polygon features

See also

Assign projection, *Define projection*, *Find projection*

Parameters

Basic parameters

Label	Name	Type	Description
Input Layer	`INPUT`	[vector: geometry]	Input vector layer to reproject
Target CRS	`TARGET_CRS`	[crs] Default: `EPSG:4326 - WGS 84`	Destination coordinate reference system
Convert curved geometries to straight segments	`CONVERT_CURVED_GE`	[boolean] Default: False	If checked, curved geometries will be converted to straight segments in the process, avoiding potential distortion issues.
Reprojected	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Coordinate Operation Optional	`OPERATION`	[string]	Specific operation to use for a particular reprojection task, instead of always forcing use of the current project's transformation settings. Useful when reprojecting a particular layer and control over the exact transformation pipeline is required. Requires proj version >= 6. Read more at *Datum Transformations*.

Outputs

Label	Name	Type	Description
Reprojected	`OUTPUT`	[same as input]	Output (reprojected) vector layer

Python code

Algorithm ID: `native:reprojectlayer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Save vector features to file

Saves vector features to a specified file dataset.

For dataset formats supporting layers, an optional layer name parameter can be used to specify a custom string. Optional GDAL-defined dataset and layer options can be specified. For more information on this, read the online GDAL documentation on the format.

Parameters

Basic parameters

Label	Name	Type	Description
Vector features	`INPUT`	[vector: any]	Input vector layer.
Saved features	`OUTPUT`	[same as input] Default: `[Save to temporary file]`	Specify the file to save the features to. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Layer name Optional	LAYER_NAME	[string]	Name to use for the output layer
GDAL dataset options Optional	DATASOURCE_OPTIONS	[string]	GDAL dataset creation options of the output format. Separate individual options with semicolons.
GDAL layer options Optional	LAYER_OPTIONS	[string]	GDAL layer creation options of the output format. Separate individual options with semicolons.
Action to take on pre-existing file	ACTION_ON_EXISTI	[enumeration] Default: 0	How to manage existing features. Valid methods are: • 0 — Create or overwrite file • 1 — Create or overwrite layer • 2 — Append features to existing layer, but do not create new fields • 3 — Append features to existing layer, and create new fields if needed

Outputs

Label	Name	Type	Description
Saved features	OUTPUT	[same as input]	Vector layer with the saved features.
File name and path	FILE_PATH	[string]	Output file name and path.
Layer name	LAYER_NAME	[string]	Name of the layer, if any.

Python code

Algorithm ID: `native:savefeatures`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Set layer encoding

Sets the encoding used for reading a layer's attributes. No permanent changes are made to the layer, rather it affects only how the layer is read during the current session.

Note

Changing the encoding is only supported for some vector layer data sources.

Parameters

Label	Name	Type	Description
Saved features	`INPUT`	[vector: geometry]	Vector layer to set the encoding.
Encoding	`ENCODING`	[string]	Text encoding to assign to the layer in the current QGIS session.

Outputs

Label	Name	Type	Description
Output layer	`OUTPUT`	[same as input]	Input vector layer with the set encoding.

Python code

Algorithm ID: `native:setlayerencoding`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Split features by character

Features are split into multiple output features by splitting a field's value at a specified character. For instance, if a layer contains features with multiple comma separated values contained in a single field, this algorithm can be used to split these values up across multiple output features. Geometries and other attributes remain unchanged in the output. Optionally, the separator string can be a regular expression for added flexibility.

Allows *features in-place modification* of point, line, and polygon features

Warning

This algorithm drops existing primary keys or FID values and regenerates them in output layers.

Parameters

Label	Name	Type	Description
Input Layer	`INPUT`	[vector: any]	Input vector layer
Split using values in the field	`FIELD`	[tablefield: any]	Field to use for splitting
Split value using character	`CHAR`	[string]	Character to use for splitting
Use regular expression separator	`REGEX`	[boolean] Default: False	

continues on next page

Table 24.214 – continued from previous page

Label	Name	Type	Description
Split	`OUTPUT`	[same as input] Default: `Create temporary layer`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Split	`OUTPUT`	[same as input]	The output vector layer.

Python code

Algorithm ID: `native:splitfeaturesbycharacter`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Split vector layer

Creates a set of vectors in an output folder based on an input layer and an attribute. The output folder will contain as many layers as the unique values found in the desired field.

The number of files generated is equal to the number of different values found for the specified attribute.

It is the opposite operation of *merging*.

Default menu: *Vector* ► *Data Management Tools*

See also

Merge vector layers

Parameters

Basic parameters

Label	Name	Type	Description
Input Layer	`INPUT`	[vector: any]	Input vector layer
Unique ID field	`FIELD`	[tablefield: any]	Field to use for splitting
Output directory	`OUTPUT`	[folder] Default: `[Save to temporary folder]`	Specify the directory for the output layers. *One of*: • Save to a Temporary Directory • Save to Directory

Advanced parameters

Label	Name	Type	Description
Output file type Optional	`FILE_TYPE`	[enumeration] Default: `gpkg` in the dialog window	Select the extension of the output files. If not specified or invalid, the output files format will be the one set in the "Default output vector layer extension" Processing setting.

Outputs

Label	Name	Type	Description
Output directory	`OUTPUT`	[folder]	The directory for the output layers
Output layers	`OUTPUT_LAYERS`	[same as input] [list]	The output vector layers resulting from the split.

Python code

Algorithm ID: `native:splitvectorlayer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Truncate table

Truncates a layer, by deleting all features from within the layer.

> **Warning**
>
> This algorithm modifies the layer in place, and deleted features cannot be restored!

Parameters

Label	Name	Type	Description
Input Layer	`INPUT`	[vector: any]	Input vector layer

Outputs

Label	Name	Type	Description
Truncated layer	`OUTPUT`	[folder]	The input layer, all features deleted

Python code

Algorithm ID: `native:truncatetable`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.26 Vector geometry

Add geometry attributes

Computes geometric properties of the features in a vector layer and includes them in the output layer.

It generates a new vector layer with the same content as the input one, but with additional attributes, containing geometric measurements based on a selected CRS.

The attributes added to the table depend on the geometry type and dimension of the input layer:

- for **point** layers: X (`xcoord`), Y (`ycoord`), Z (`zcoord`) coordinates and/or M value (`mvalue`)
- for **line** layers: `length` and, for the LineString and CompoundCurve geometry types, the feature `sinuosity` and straight distance (`straightdis`)
- for **polygon** layers: `perimeter` and `area`

> **Note**
>
> This algorithm uses ellipsoid based measurements and respects the current *ellipsoid settings*.

Default menu: *Vector ► Geometry Tools*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Calculate using	`CALC_METHOD`	[enumeration] Default: 0	Calculation parameters to use for the geometric properties. One of: • 0 — Layer CRS • 1 — Project CRS • 2 — Ellipsoidal
Added geom info	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output (input copy with geometry) layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Added geom info	`OUTPUT`	[same as input]	Copy of the input vector layer with the addition of the geometry fields

Python code

Algorithm ID: `qgis:exportaddgeometrycolumns`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Affine transform

Applies an affine transformation to the layer geometries. Affine transformations can include translation, scaling and rotation. The operations are performed in the following order: scale, rotation, and translation.

Z and M values (if present) can be translated and scaled.

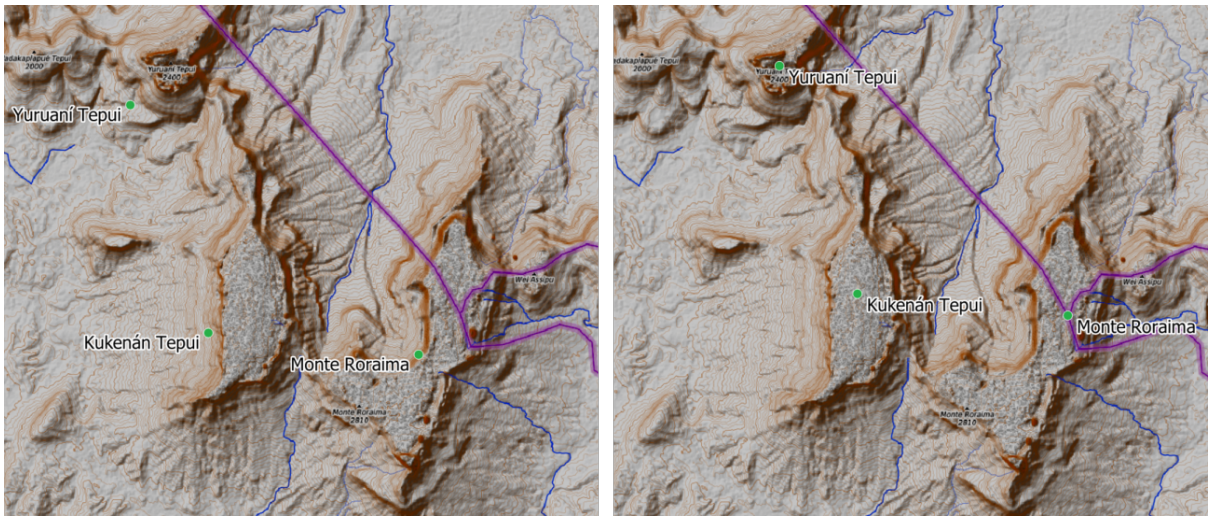


Fig. 24.72: Vector point layer (green dots) before (left), and after (right) an affine transformation (translation).

Allows *features in-place modification* of point, line, and polygon features

See also

Translate

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Translation (x-axis)	`DELTA_X`	[numeric: double] Default: 0.0	Displacement to apply on the X axis.
Translation (y-axis)	`DELTA_Y`	[numeric: double] Default: 0.0	Displacement to apply on the Y axis.
Translation (z-axis)	`DELTA_Z`	[numeric: double] Default: 0.0	Displacement to apply on the Z axis.
Translation (m-values)	`DELTA_M`	[numeric: double] Default: 0.0	Offset to apply on m values.
Scale factor (x-axis)	`SCALE_X`	[numeric: double] Default: 1.0	Scaling value (expansion or contraction) to apply on the X axis.

continues on next page

Table 24.217 – continued from previous page

Label	Name	Type	Description
Scale factor (y-axis)	`SCALE_Y`	[numeric: double] Default: 1.0	Scaling value (expansion or contraction) to apply on the Y axis.
Scale factor (z-axis)	`SCALE_Z`	[numeric: double] Default: 1.0	Scaling value (expansion or contraction) to apply on the Z axis.
Scale factor (m-values)	`SCALE_M`	[numeric: double] Default: 1.0	Scaling value (expansion or contraction) to apply on m values.
Rotation around z-axis (degrees counter-clockwise)	`ROTATION_Z`	[numeric: double] Default: 0.0	Angle of the rotation in degrees.
Transformed	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Transformed	`OUTPUT`	[same as input]	Output (transformed) vector layer.

Python code

Algorithm ID: `native:affinetransform`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Aggregate

Takes a vector or table layer and creates a new layer by aggregating features based on a `group by` expression.

Features for which `group by` expression returns the same value are grouped together.

It is possible to group all source features together using constant value in `group by` parameter, example: NULL.

It is also possible to group features by multiple fields using Array function, example: Array("Field1", "Field2").

Geometries (if present) are combined into one multipart geometry for each group. Output attributes are computed depending on each given aggregate definition.

This algorithm allows to use the default *aggregates functions* of the QGIS Expression engine.

Note

This algorithm uses ellipsoid based measurements and respects the current *ellipsoid settings*.

See also

Collect geometries, *Dissolve*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	Input vector layer
Group by expression	`GROUP_BY`	[tablefield: any] Default: 'NULL'	Choose the grouping field. If *NULL* all features will be grouped.

continues on next page

Table 24.219 – continued from previous page

Label	Name	Type	Description
Aggregates	AGGREGATES	[list]	List of output layer field definitions. Example of a field definition: *{'aggregate': 'sum', 'delimiter': ',', 'input': ' $area', 'length': 10, 'name': 'totarea', 'precision': 0, 'type': 6}* By default, the list contains all the fields of the input layer. In the GUI, you can edit these fields and their definitions, and you can also: • Click the  button to add a new field. • Click  to delete the selected field. • Use  and  to change order of the fields. • Click  to reset to the default (the fields of the input layer). For each of the fields you'd like to retrieve information from, you need to define the following: ***Input expression*** **(`input`) [expression]** Field or expression from the input layer. ***Aggregate function*** **(`aggregate`) [enumeration]** Function to use on the input expression to return the aggregated value. These are mainly the *Aggregates Functions*, plus dedicated "first_value" and "last_value" functions for ordered list items. Default: *concatenate* (for string data type), *sum* (for numeric data type) ***Delimiter*** **(`delimiter`) [string]** Text string to separate aggregated values, for example in case of concatenation. Default: , ***Output field name*** **(`name`) [string]** Name of the aggregated field in the output layer. By default input field name is kept. ***Type*** **(`type`) [enumeration]** Data type of the output field. Available types may not be compatible with the output layer provider. One of: **Attention** For certain field types, e.g. lists, an extra `sub_type` parameter helps refine the specific type of the data. It is automatically set in the GUI but may be needed if you're running the algorithm in Python or from the command line. • 1 — Boolean • 2 — Integer (32bit) • 4 — Integer (64bit) • 6 — Decimal (double)

Table 24.219 – continued from previous page

Label	Name	Type	Description
Load fields from layer	GUI only	[vector: any]	You can load fields from another layer and use them for the aggregation
Aggregated	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output (aggregate) layer *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Aggregated	`OUTPUT`	[same as input]	Multigeometry vector layer with the aggregated values

Python code

Algorithm ID: `native:aggregate`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Boundary

Returns the closure of the combinatorial boundary of the input geometries (i.e. the topological boundary of the geometry).

Only for polygon and line layers.

For **polygon geometries** , the boundary consists of all the lines making up the rings of the polygon.

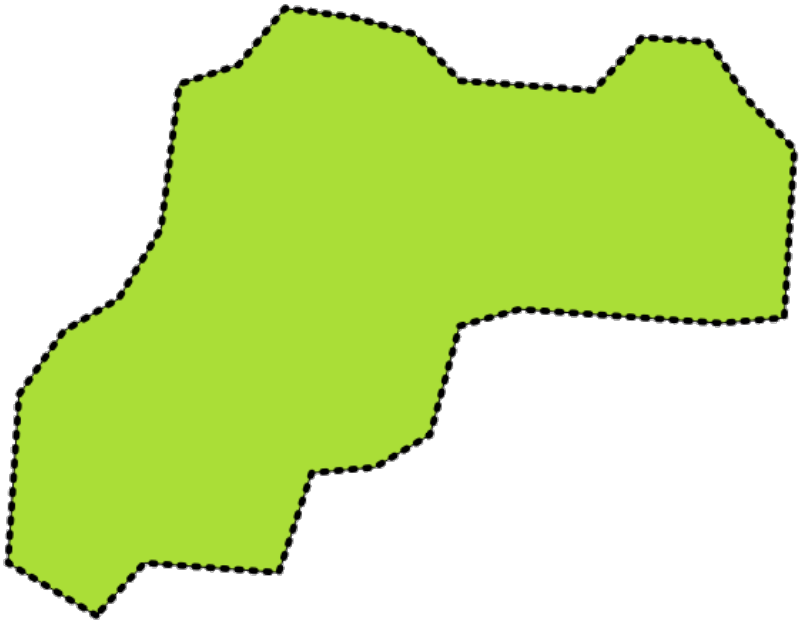


Fig. 24.73: Boundaries (black dashed line) of the source polygon layer

For **lines geometries**, the boundaries are their end points.

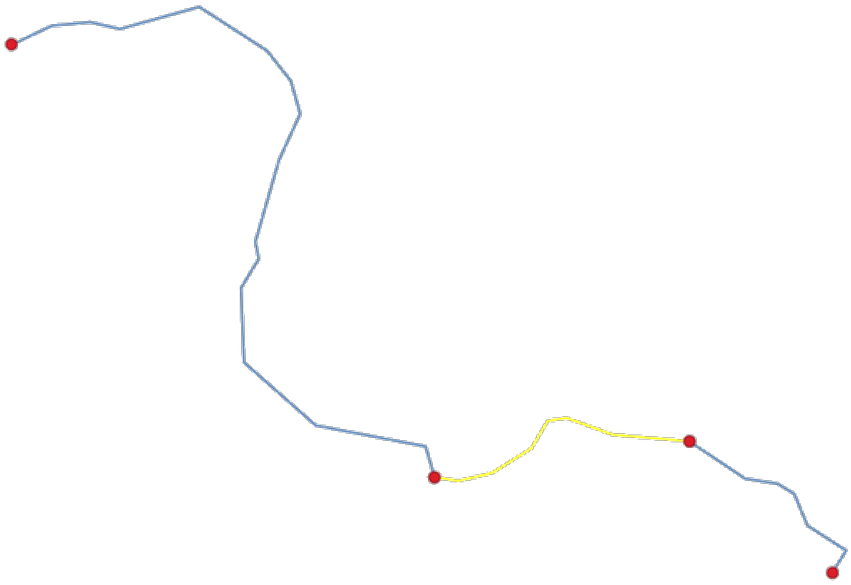


Fig. 24.74: Boundary layer (red points) for lines. In yellow a selected feature.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line, polygon]	Input line or polygon vector layer
Boundary	`OUTPUT`	[vector: point, line] Default: `[Create temporary layer]`	Specify the output (boundary) layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Boundary	`OUTPUT`	[vector: point, line]	Boundaries from the input layer (point for line, and line for polygon)

Python code

Algorithm ID: `native:boundary`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Bounding boxes

Calculates the bounding box (envelope) of each feature in an input layer. Polygon and line geometries are supported.

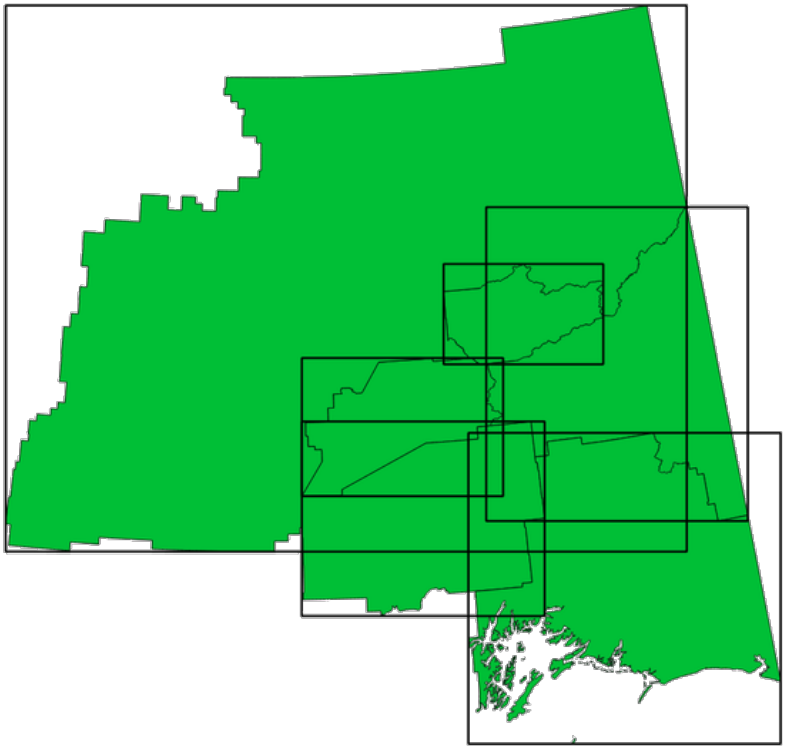


Fig. 24.75: Black lines represent the bounding boxes of each polygon feature

Allows *features in-place modification* of polygon features

See also

Minimum bounding geometry

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line, polygon]	Input line or polygon vector layer
Bounds	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output (bounding box) layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Bounds	`OUTPUT`	[vector: polygon]	Bounding boxes of input layer. Other than the input attributes, the output layer also contains following fields: `width`, `height`, `area` and `perimeter` of the generated polygon.

Python code

Algorithm ID: `native:boundingboxes`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Buffer

Computes a buffer area for all the features in an input layer, using a fixed or data defined distance.

It is possible to use a negative distance for polygon input layers. In this case the buffer will result in a smaller polygon (setback).

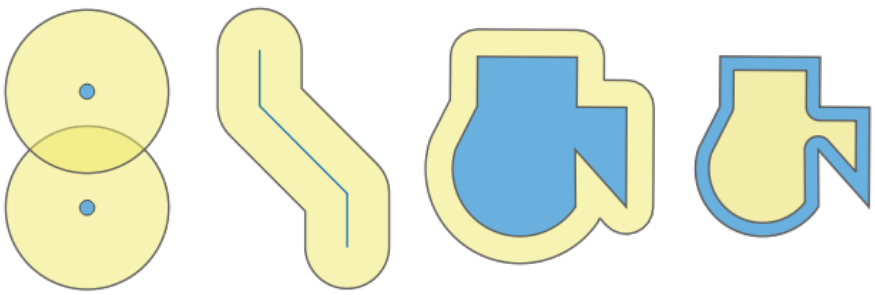


Fig. 24.76: Buffer (in yellow) of points, line, polygon with positive buffer, and polygon with negative buffer

Allows *features in-place modification* of polygon features

Default menu: *Vector* ► *Geoprocessing Tools*

Warning

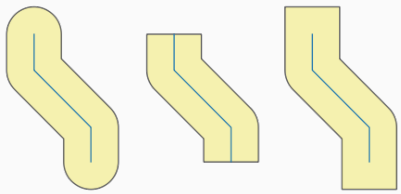
This algorithm may drop existing primary keys or FID values and regenerate them in output layers, depending on the input parameters.

See also

Variable distance buffer, *Multi-ring buffer (constant distance)*, *Variable width buffer (by M value)*

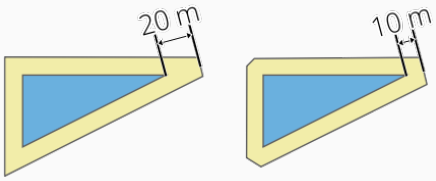
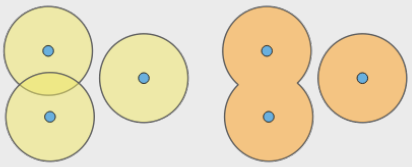
Parameters

Basic parameters

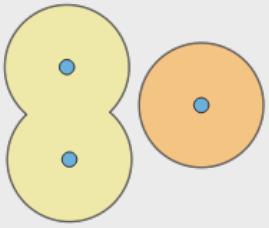
Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Distance	`DISTANCE`	[numeric: double] Default: 10.0	Buffer distance (from the boundary of each feature). You can use the Data Defined button on the right to choose a field from which the radius will be calculated. This way you can have different radius for each feature.
Segments	`SEGMENTS`	[numeric: integer] Default: 5	Controls the number of line segments to use to approximate a quarter circle when creating rounded offsets.
End cap style	`END_CAP_STYLE`	[enumeration] Default: 0	Controls how line endings are handled in the buffer. One of: • 0 — Round • 1 — Flat • 2 — Square  Fig. 24.77: Round, flat and square cap styles
Join style	`JOIN_STYLE`	[enumeration] Default: 0	Specifies whether round, miter or beveled joins should be used when offsetting corners in a line. Options are: • 0 — Round • 1 — Miter • 2 — Bevel  Fig. 24.78: Round, miter, and bevel join styles

continues on next page

Table 24.220 – continued from previous page

Label	Name	Type	Description
Miter limit	`MITER_LIMIT`	[numeric: double] Default: 2.0	Sets the maximum distance from the offset geometry to use when creating a mitered join as a factor of the offset distance (only applicable for miter join styles). Minimum: 1.0  Fig. 24.79: A 10m buffer with a limit of 2 and a 10m buffer with a limit of 1
Dissolve result	`DISSOLVE`	[boolean] Default: False	Dissolve the final buffer. If `True` (checked), overlapping buffers will be dissolved (combined) into a single multipart feature.  Fig. 24.80: Standard (three single part features - left), dissolved (1 multipart feature with 2 parts - right)
Buffered	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output (buffer) layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Keep disjoint features separate	`SEPARATE_DISJOINT`	[boolean] Default: False	If `True` (checked) and dissolved is checked, features that do not overlap or touch will be exported as separate features (instead of parts of a multipart feature).  Fig. 24.81: Results in 2 single part features

Outputs

Label	Name	Type	Description
Buffered	`OUTPUT`	[vector: polygon]	Output (buffer) polygon layer

Python code

Algorithm ID: `native:buffer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Centroids

Creates a new point layer, with points representing the centroids of the geometries of the input layer.

The centroid is a single point representing the barycenter (of all parts) of the feature, so it can be outside the feature borders. But can also be a point on each part of the feature.

The attributes of the points in the output layer are the same as for the original features.

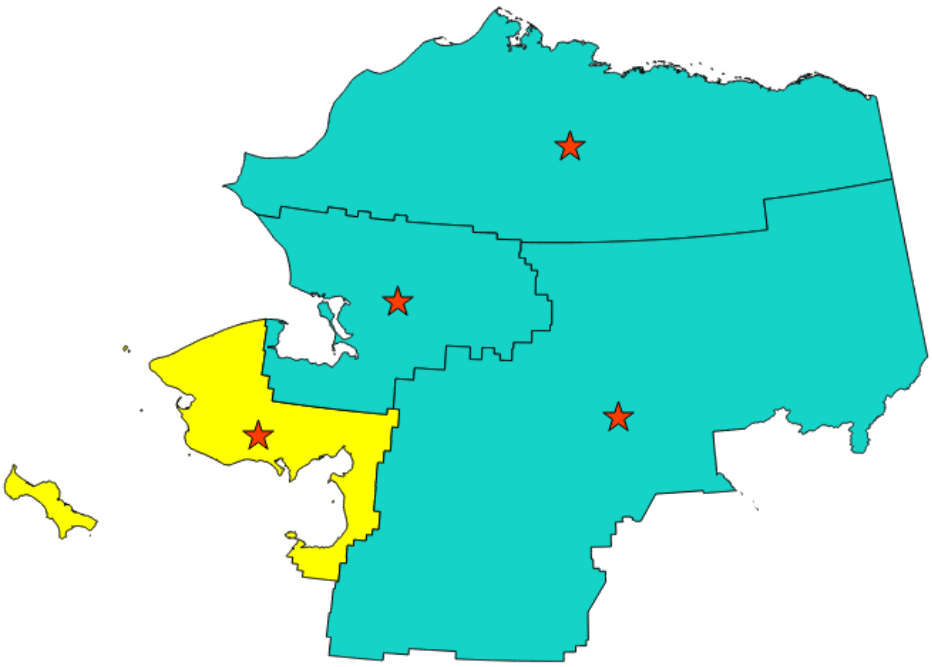


Fig. 24.82: The red stars represent the centroids of the features of the input layer.

Allows *features in-place modification* of point features

Default menu: *Vector* ► *Geometry Tools*

Warning

This algorithm may drop existing primary keys or FID values and regenerate them in output layers, depending on the input parameters.

See also

Point on Surface

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Create centroid for each part	`ALL_PARTS`	[boolean] Default: False	If True (checked), a centroid will be created for each part of the geometry
Centroids	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specify the output (centroid) layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Centroids	`OUTPUT`	[vector: point]	Output point vector layer (centroids)

Python code

Algorithm ID: `native:centroids`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Check validity

Performs a validity check on the geometries of a vector layer.

The geometries are classified in three groups (valid, invalid and error) and for each group, a vector layer with its features is generated:

- The **Valid output** layer contains only the valid features (without topological errors).
- The **Invalid output** layer contains all the invalid features found by the algorithm.
- The **Error output** layer is a point layer that points to where the invalid features were found.

The attribute tables of the generated layers will contain some additional information ("message" for the **error** layer, "FID" and "_errors" for the **invalid** layer and only "FID" for the **valid** layer):

The attribute table of each generated vector layer will contain some additional information (number of errors found and types of error):

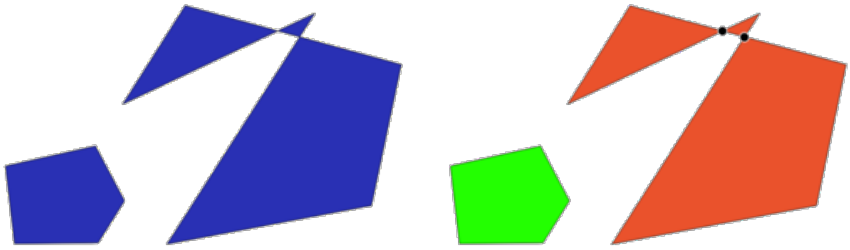


Fig. 24.83: Left: the input layer. Right: the valid layer (green), the invalid layer (orange)

Default menu: *Vector ► Geometry Tools*

See also

Fix geometries and the core plugin *Geometry Checker Plugin*, *Validate coverage*

Parameters

Label	Name	Type	Description
Input layer	`INPUT_LAYER`	[vector: geometry]	Input vector layer
Method	`METHOD`	[enumeration] Default: 2	Method to use to check validity. Options: • 0: The one selected in digitizing settings • 1: QGIS • 2: GEOS
Ignore ring self intersection	`IGNORE_RING_SELF`	[boolean] Default: False	Ignore self intersecting rings when checking for validity.

continues on next page

Table 24.221 – continued from previous page

Label	Name	Type	Description
Valid output	VALID_OUTPUT	[same as input] Default: [Create temporary layer]	Specify the vector layer to contain a copy of the valid features of the source layer. *One of*: • Skip Output • Create Temporary Layer (TEMPORARY_OUTPUT) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Invalid output	INVALID_OUTPUT	[same as input] Default: [Create temporary layer]	Vector layer containing copy of the invalid features of the source layer with the field _errors listing the summary of the error(s) found. *One of*: • Skip Output • Create Temporary Layer (TEMPORARY_OUTPUT) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Error output	ERROR_OUTPUT	[vector: point] Default: [Create temporary layer]	Point layer of the exact position of the validity problems detected with the message field describing the error(s) found. *One of*: • Skip Output • Create Temporary Layer (TEMPORARY_OUTPUT) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Count of errors	ERROR_COUNT	[numeric: integer]	The number of geometries that caused errors.
Error output	ERROR_OUTPUT	[vector: point]	Point layer of the exact position of the validity problems detected with the message field describing the error(s) found.
Count of invalid features	INVALID_COUNT	[numeric: integer]	The number of invalid geometries.
Invalid output	INVALID_OUTPUT	[same as input]	Vector layer containing copy of the invalid features of the source layer with the field _errors listing the summary of the error(s) found.
Count of valid features	VALID_COUNT	[numeric: integer]	The number of valid geometries.
Valid output	VALID_OUTPUT	[same as input]	Vector layer containing a copy of the valid features of the source layer.

Python code

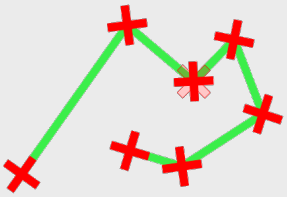
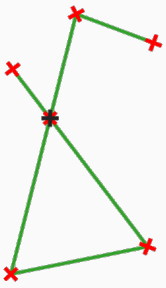
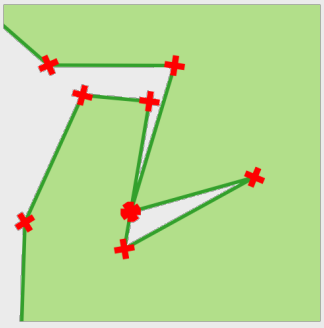
Algorithm ID: `native:checkvalidity`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

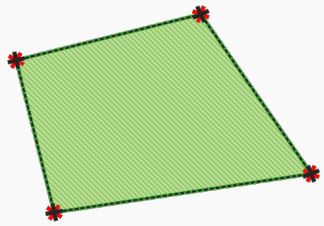
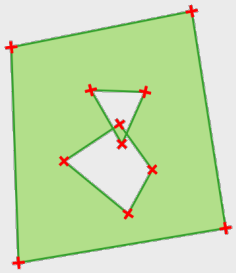
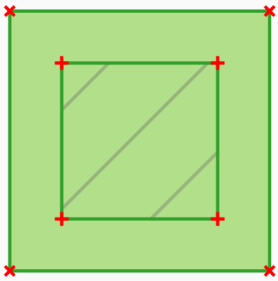
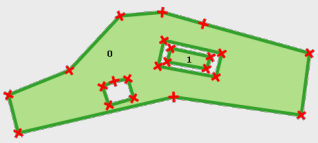
Types of error messages and their meanings

Table 24.223: Some examples of geometry check failures in QGIS

Geometry context	GEOS validation and error message	QGIS validation and error message
Consecutive points on a line have the same coordinates 	✗ *Repeated point*	✗ *Line a contains x duplicate node(s) at b*
Segments of a line intersect each other 		✗ *Segments a and b of line c intersect at d*
Polygon geometry touches itself and generates a ring 	✗ *Ring self-intersection*	✗ *Ring self-intersection*

continues on next page

Table 24.223 – continued from previous page

Geometry context	GEOS validation and error message	QGIS validation and error message
Two rings (exterior or interior) of a polygon geometry are identical 	✗ *Duplicate rings*	
Geometry touches itself 	✗ *Self-intersection*	
A polygon geometry is on top of another polygon geometry 	✗ *Nested shell*	
Part of a MultiPolygon geometry is within a hole of a MultiPolygon geometry 		✗ *Polygon a lies inside polygon b*

continues on next page

Table 24.223 – continued from previous page

Geometry context	GEOS validation and error message	QGIS validation and error message
Point geometry does not have a proper coordinate pair. The coordinate pair does not contain a latitude value and a longitude value in that order.	*Invalid coordinate*	

Collect geometries

Takes a vector layer and collects its geometries into new multipart geometries.

One or more attributes can be specified to collect only geometries belonging to the same class (having the same value for the specified attributes), alternatively all geometries can be collected.

All output geometries will be converted to multi geometries, even those with just a single part. This algorithm does not dissolve overlapping geometries - they will be collected together without modifying the shape of each geometry part.

See the 'Promote to multipart' or 'Aggregate' algorithms for alternative options.

Default menu: *Vector ► Geometry Tools*

Warning

This algorithm drops existing primary keys or FID values and regenerates them in output layers.

See also

Aggregate, *Promote to multipart*, *Dissolve*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Unique ID fields	`FIELD`	[tablefield: any] [list]	Choose one or more attributes to collect the geometries
Collected	`OUTPUT`	[same as input]	Vector layer with collected geometries

Outputs

Label	Name	Type	Description
Collected	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer for the collected geometries. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Python code

Algorithm ID: `native:collect`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Concave hull

Computes the concave hull of the features from an input point layer.

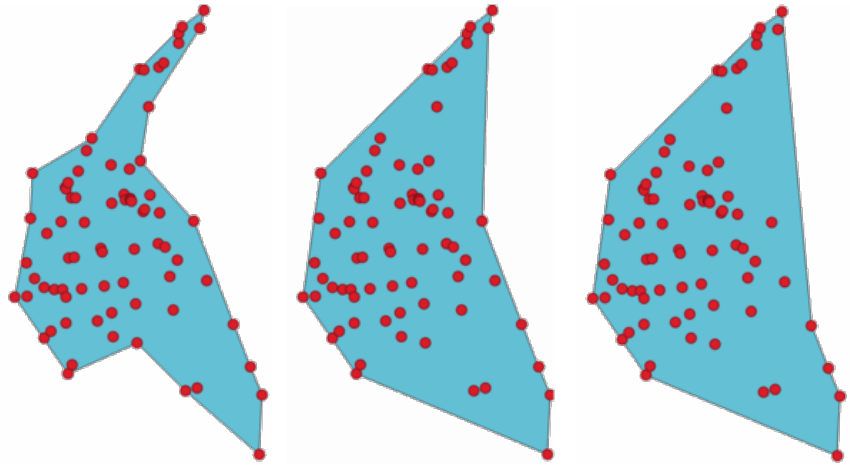


Fig. 24.84: Concave hulls with different thresholds (0.3, 0.6, 0.9)

See also

Convex hull

Parameters

Label	Name	Type	Description
Input point layer	`INPUT`	[vector: point]	Input point vector layer
Threshold	`ALPHA`	[numeric: double] Default: 0.3	Number from 0 (maximum concave hull) to 1 (convex hull).
Allow holes	`HOLES`	[boolean] Default: True	Choose whether to allow holes in the final concave hull
Split multipart geometry into singlepart geometries	`NO_MULTIGEOMET`	[boolean] Default: False	Check if you want to have singlepart geometries instead of multipart ones.
Concave hull	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Concave hull	`OUTPUT`	[vector: polygon]	The output vector layer

Python code

Algorithm ID: `native:concavehull`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Convert geometry type

Generates a new layer based on an existing one, with a different type of geometry.

The attribute table of the output layer is the same as the one of the input layer.

Not all conversions are possible. For instance, a line layer can be converted to a point layer, but a point layer cannot be converted to a line layer.

See also

Polygonize, *Lines to polygons*, *Polygons to lines*, *Points to path*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
New geometry type	`TYPE`	[enumeration] Default: 0	Geometry type to apply to the output features. One of: • 0 — Centroids • 1 — Nodes • 2 — Linestrings • 3 — Multilinestrings • 4 — Polygons
Converted	`OUTPUT`	[vector: geometry] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Converted	`OUTPUT`	[vector: geometry]	Output vector layer - the type depends on the parameters

Python code

Algorithm ID: `qgis:convertgeometrytype`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Convert to curved geometries

Converts a geometry into its curved geometry equivalent.

Already curved geometries will be retained without change.

Allows *features in-place modification* of line and polygon features

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line or polygon]	Input vector layer
Maximum distance tolerance	`DISTANCE`	[numeric: double] Default: 0.000001	The maximum distance allowed between the original location of vertices and where they would fall on the converted curved geometries
Maximum angle tolerance	`ANGLE`	[numeric: double] Default: 0.000001	Segments are considered as suitable for replacing with an arc if the points are all regularly spaced on the candidate arc. This parameter specifies the maximum angular deviation (in degrees) allowed when testing for regular point spacing. Between 0 and 45°.
Curves	`OUTPUT`	[vector: compoundcurve or curvepolygon] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Curves	`OUTPUT`	[vector: compoundcurve or curvepolygon]	Output vector layer with curved geometries

Python code

Algorithm ID: `native:converttocurves`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Convex hull

Calculates the convex hull for each feature in an input layer.

See the 'Minimum bounding geometry' algorithm for a convex hull calculation which covers the whole layer or grouped subsets of features.

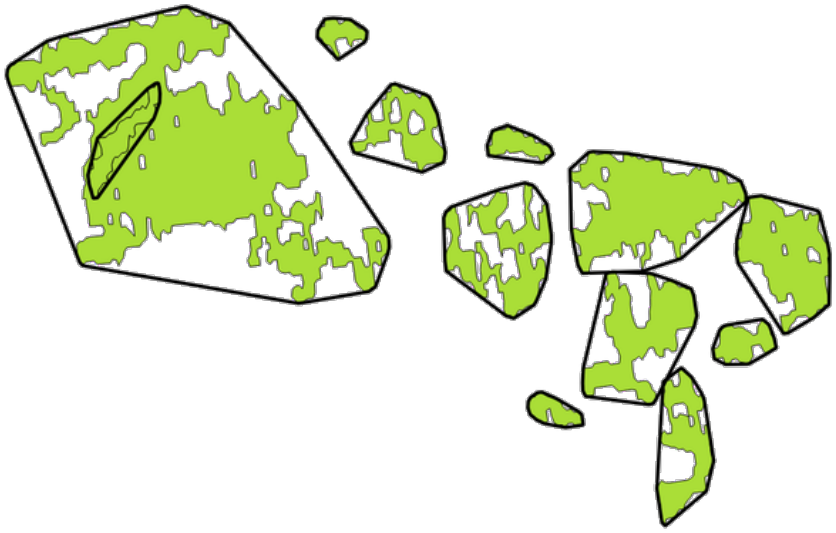


Fig. 24.85: Black lines identify the convex hull for each layer feature

Allows *features in-place modification* of polygon features

Default menu: *Vector* ► *Geoprocessing Tools*

See also

Minimum bounding geometry, *Concave hull*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Convex hull	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Convex hull	`OUTPUT`	[vector: polygon]	The output (convex hull) vector layer. Other than the input attributes, the output layer also contains the following fields: `area` and `perimeter` of the generated polygon.

Python code

Algorithm ID: `native:convexhull`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Create wedge buffers

Creates wedge shaped buffers from input points.

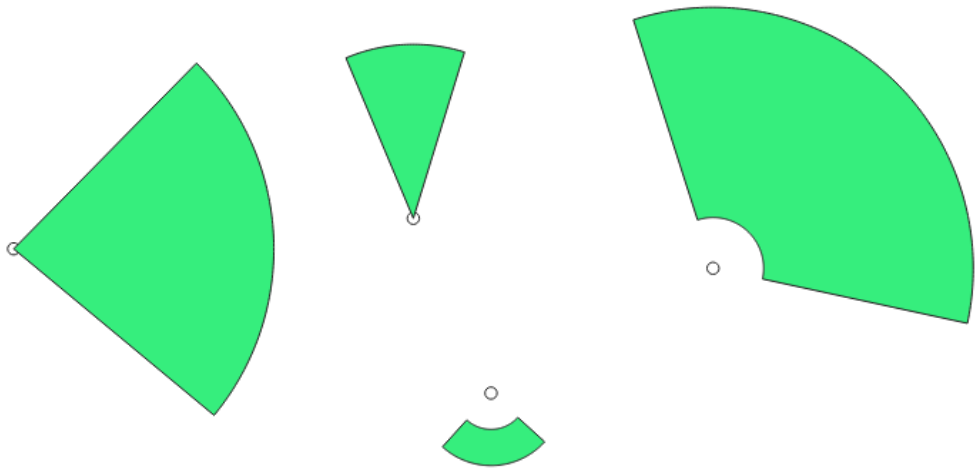


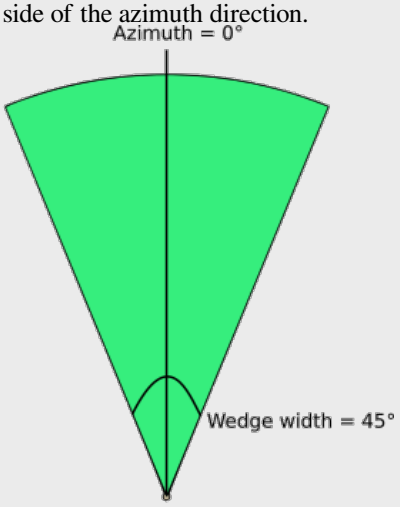
Fig. 24.86: Wedge buffers

The native output from this algorithm are CurvePolygon geometries, but these may be automatically segmentized to Polygons depending on the output format.

See also

Buffer, *Variable width buffer (by M value)*, *Tapered buffers*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: point]	Input point vector layer
Azimuth (degrees from North)	`AZIMUTH`	[numeric: double] Default: 0.0	Angle (in degrees) as the middle value of the wedge
Wedge width (in degrees)	`WIDTH`	[numeric: double] Default: 45.0	Width (in degrees) of the buffer. The wedge will extend to half of the angular width either side of the azimuth direction.  Fig. 24.87: Azimuth and width values of the wedge buffer
Outer radius	`OUTER_RADIUS`	[numeric: double] Default: 1.0	The outer *size* (length) of the wedge: the size is meant from the source point to the edge of the wedge shape.
Inner radius Optional	`INNER_RADIUS`	[numeric: double] Default: 0.0	Inner radius value. If 0 the wedge will begin from the source point.
Buffers	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Buffers	`OUTPUT`	[vector: polygon]	The output (wedge buffer) vector layer

Python code

Algorithm ID: `native:wedgebuffers`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Delaunay triangulation

Creates a polygon layer with the Delaunay triangulation corresponding to the input point layer.

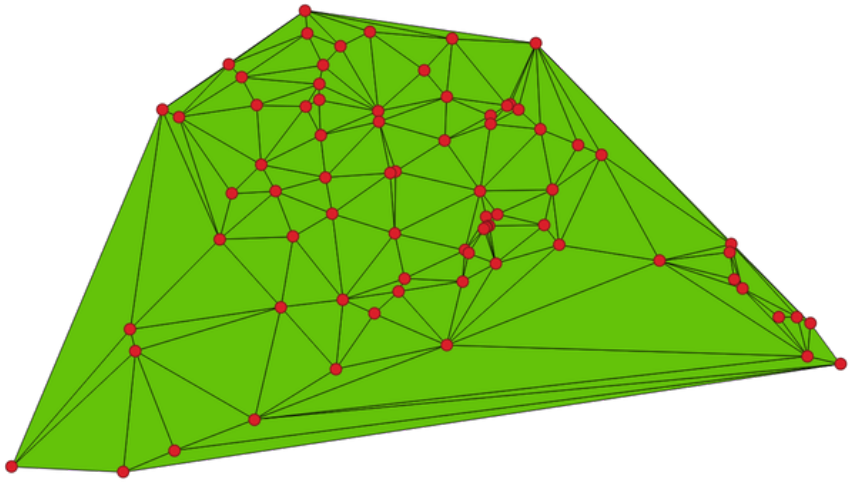


Fig. 24.88: Delaunay triangulation on points

Default menu: *Vector ► Geometry Tools*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: point]	Input point vector layer
Tolerance Optional	`TOLERANCE`	[numeric: double] Default: 0.0	Specifies an optional snapping tolerance which can be used to improve the robustness of the triangulation.
Add point IDs to output	`ADD_ATTRIBUTES`	[boolean] Default: True	Specifies whether fields storing involved point features ID should be added to the output. If False, an `id` field is used to identify the polygons.
Delaunay triangulation	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Delaunay triangulation	`OUTPUT`	[vector: polygon]	The output (Delaunay triangulation) vector layer

Python code

Algorithm ID: `native:delaunaytriangulation`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Delete holes

Takes a polygon layer and removes holes in polygons. It creates a new vector layer in which polygons with holes have been replaced by polygons with only their external ring. Attributes are not modified.

An optional minimum area parameter allows removing only holes which are smaller than a specified area threshold. Leaving this parameter at `0.0` results in all holes being removed.

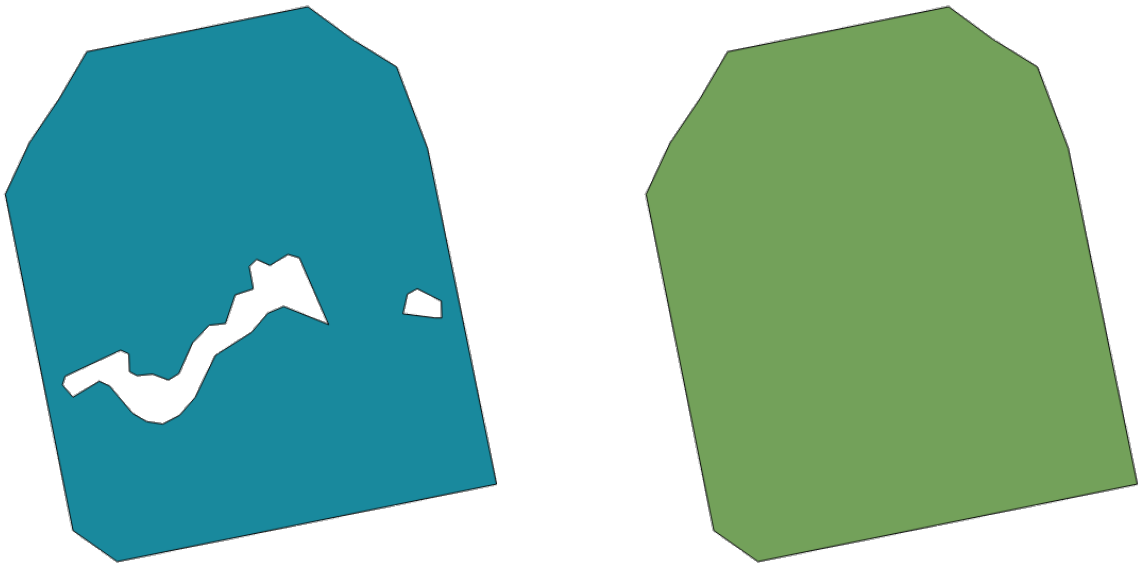


Fig. 24.89: Before and after the cleaning

Allows *features in-place modification* of polygon features

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: polygon]	Input polygon vector layer
Remove holes with area less than Optional	`MIN_AREA`	[numeric: double] Default: 0.0	Only holes with an area less than this threshold will be deleted. With a value of `0.0`, **all** the holes will be deleted.
Cleaned	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Cleaned	`OUTPUT`	[same as input]	The output (cleaned) vector layer

Python code

Algorithm ID: `native:deleteholes`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Densify by count

Takes a polygon or line layer and generates a new one in which the geometries have a larger number of vertices than the original one.

If the geometries have Z or M values present then these will be linearly interpolated at the added vertices.

The number of new vertices to add to each segment is specified as an input parameter.

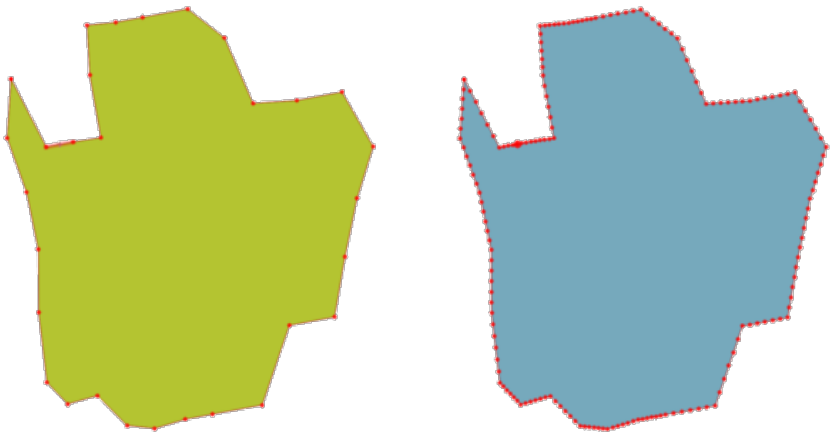


Fig. 24.90: Red points show the vertices before and after the densify

Allows *features in-place modification* of line and polygon features

Default menu: *Vector* ► *Geometry Tools*

See also

Densify by interval

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line, polygon]	Input line or polygon vector layer
Vertices to add	`VERTICES`	[numeric: integer] Default: 1	Number of vertices to add to each segment
Densified	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Densified	`OUTPUT`	[same as input]	The output (densified) vector layer

Python code

Algorithm ID: `native:densifygeometries`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Densify by interval

Takes a polygon or line layer and generates a new one in which the geometries have a larger number of vertices than the original one.

The geometries are densified by adding regularly placed extra vertices inside each segment so that the maximum distance between any two vertices does not exceed the specified distance.

If the geometries have Z or M values present then these will be linearly interpolated at the added vertices.

Example

Specifying a distance of 3 would cause the segment `[0 0] -> [10 0]` to be converted to `[0 0] -> [2.5 0] -> [5 0] -> [7.5 0] -> [10 0]`, since 3 extra vertices are required on the segment and spacing these at 2.5 increments allows them to be evenly spaced over the segment.

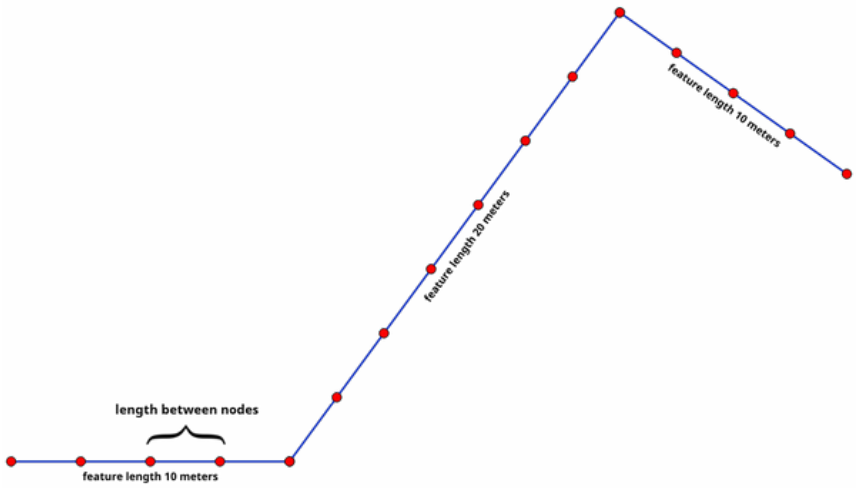


Fig. 24.91: Densify geometry at a given interval

Allows *features in-place modification* of line and polygon features

See also

Densify by count

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line, polygon]	Input line or polygon vector layer
Interval between vertices to add	`INTERVAL`	[numeric: double] Default: 1.0	Maximum distance between two consecutive vertices
Densified	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Densified	`OUTPUT`	[same as input]	The output (densified) vector layer

Python code

Algorithm ID: `native:densifygeometriesgivenaninterval`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Dissolve

Takes a vector layer and combines its features into new features. One or more attributes can be specified to dissolve features belonging to the same class (having the same value for the specified attributes), alternatively all features can be dissolved to a single feature.

All output geometries will be converted to multi geometries. In case the input is a polygon layer, common boundaries of adjacent polygons being dissolved will get erased. If enabled, the optional "Keep disjoint features separate" setting will cause features and parts that do not overlap or touch to be exported as separate features (instead of parts of a single multipart feature).

The resulting attribute table will have the same fields as the input layer. The values in the output layer's fields are the ones of the first input feature that happens to be processed.

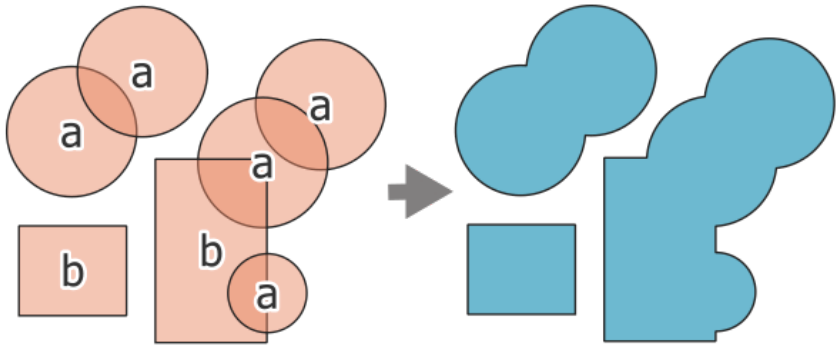


Fig. 24.92: Dissolving a layer into a single multipart feature

Default menu: *Vector* ► *Geoprocessing Tools*

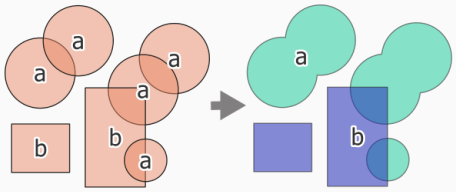
> **Warning**
>
> This algorithm drops existing primary keys or FID values and regenerates them in output layers.

See also

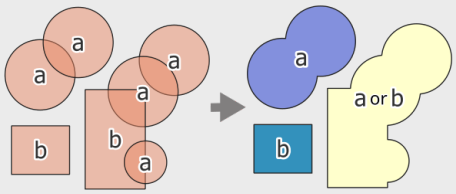
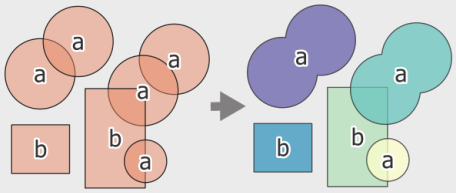
Dissolve coverage, *Aggregate*, *Collect geometries*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Dissolve field(s) Optional	`FIELD`	[tablefield: any] [list] Default: []	Features having the same value for the selected field(s) will be replaced with a single one and their geometries are merged. If no field is provided then all the features are dissolved, resulting in a single (multipart) feature.  Fig. 24.93: Dissolve the polygon layer on a common attribute (2 multipart features)
Dissolved	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Keep disjoint features separate	`SEPARATE_DISJOINT`	[boolean] Default: False	Parts of dissolved features are exported as separate features (instead of parts of a multipart feature).  Fig. 24.94: source (left), dissolve all (3 distinct features - right)  Fig. 24.95: source (left), dissolve on field (5 distinct features - right)

Outputs

Label	Name	Type	Description
Dissolved	`OUTPUT`	[same as input]	The output vector layer with dissolved geometries

Python code

Algorithm ID: `native:dissolve`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Drape (set Z value from raster)

Uses values sampled from a band within a raster layer to set the Z value for every overlapping vertex in the feature geometry. The raster values can optionally be scaled by a preset amount.

If Z values already exist in the layer, they will be overwritten with the new value. If no Z values exist, the geometry will be upgraded to include the Z dimension.

Allows *features in-place modification* of point, line, and polygon features with Z enabled

See also

Set M value from raster, *Set Z value*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Raster layer	`RASTER`	[raster]	Raster layer with Z values
Band number	`BAND`	[raster band] Default: 1	The raster band to take the Z values from
Value for No-Data or non-intersecting vertices	`NODATA`	[numeric: double] Default: 0.0	Value to use in case the vertex does not intersect (a valid pixel of) the raster
Scale factor	`SCALE`	[numeric: double] Default: 1.0	Scaling value: the band values are multiplied by this value.
Offset	`OFFSET`	[numeric: double] Default: 0.0	Offset value: it is algebraically added to the band values after applying the "Scale factor".
Updated	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer (with Z values from the raster layer). *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Updated	`OUTPUT`	[same as input]	The output vector layer with Z values from the raster layer

Python code

Algorithm ID: `native:setzfromraster`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Drop M/Z values

Removes M (measure) or Z (altitude) values from input geometries.

See also

Set M value, *Set Z value*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer with M or Z values
Drop M Values	`DROP_M_VALUES`	[boolean] Default: False	Removes the M values from the geometries
Drop Z Values	`DROP_Z_VALUES`	[boolean] Default: False	Removes the Z values from the geometries
Z/M Dropped	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Z/M Dropped	`OUTPUT`	[same as input]	The output vector layer (identical to the input layer, except that the M and/or Z dimensions have been removed from the geometries).

Python code

Algorithm ID: `native:dropmzvalues`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Eliminate selected polygons

Combines selected polygons of the input layer with certain adjacent polygons by erasing their common boundary. The adjacent polygon can be either the one with the largest or smallest area or the one sharing the largest common boundary with the polygon to be eliminated.

Eliminate is normally used to get rid of sliver polygons, i.e. tiny polygons that are a result of polygon intersection processes where boundaries of the inputs are similar but not identical.

Default menu: *Vector ► Geoprocessing Tools*

See also

Fix geometries

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: polygon]	Input polygon vector layer
Merge selection with the neighboring polygon with the	`MODE`	[enumeration] Default: Not set	Choose the parameter to use in order to get rid of the selected polygons: • 0 — Largest Area • 1 — Smallest Area • 2 — Largest Common Boundary
Eliminated	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Eliminated	`OUTPUT`	[vector: polygon]	The output polygon vector layer.

Python code

Algorithm ID: `qgis:eliminateselectedpolygons`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Explode lines

Takes a lines layer and creates a new one in which each line layer is replaced by a set of lines representing the segments in the original line.

Each line in the resulting layer contains only a start and an end point, with no intermediate vertices between them.

If the input layer consists of CircularStrings or CompoundCurves, the output layer will be of the same type and contain only single curve segments.

Note

- This algorithm drops existing primary keys or FID values and regenerates them in output layers.
- This algorithm does not require valid geometries as input.



Fig. 24.96: The original line layer and the exploded one

Allows *features in-place modification* of line features

See also

Subdivide, *Line substring*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line]	Input line vector layer
Exploded	`OUTPUT`	[vector: line] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Exploded	`OUTPUT`	[vector: line]	The output line vector layer with features representing each segment of the input layer.

Python code

Algorithm ID: `native:explodelines`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Extend lines

Extends line geometry by a specified amount at the start and end of the line.

Lines are extended using the bearing of the first and last segment in the line.

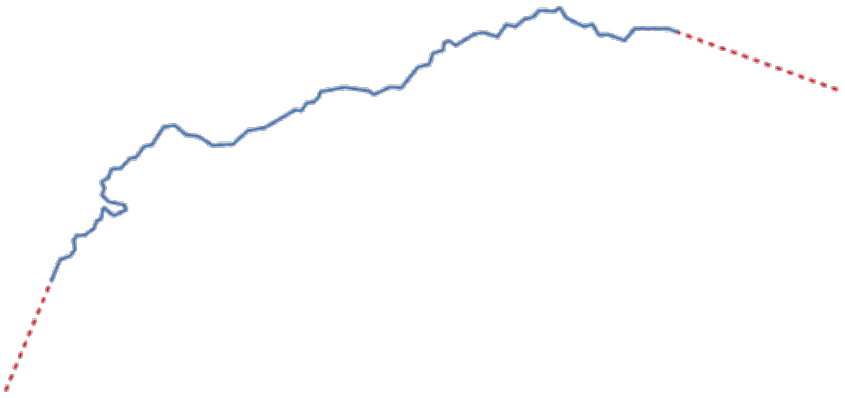


Fig. 24.97: The red dashes represent the initial and final extension of the original layer

Allows *features in-place modification* of line features

See also

Line substring

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line]	Input line vector layer
Start distance	`START_DISTANCE`	[numeric: double] Default: 0.0	Distance by which to extend the first segment of the line (starting point)
End distance	`END_DISTANCE`	[numeric: double] Default: 0.0	Distance by which to extend the last segment of the line (ending point)
Extended	`OUTPUT`	[vector: line] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Extended	`OUTPUT`	[vector: line]	The output (extended) line vector layer.

Python code

Algorithm ID: `native:extendlines`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Extract M values

Extracts M values from geometries into feature attributes.

By default only the M value from the first vertex of each feature is extracted, however the algorithm can optionally calculate statistics on all of the geometry's M values, including sum, mean, minimum and maximum.

See also

Extract Z values, *Set M value*, *Drop M/Z values*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Summaries to calculate	`SUMMARIES`	[enumeration] Default: [0]	Statistics on the M values of a geometry. One or more of: • 0 — First • 1 — Last • 2 — Count • 3 — Sum • 4 — Mean • 5 — Median • 6 — St.dev (pop) • 7 — Minimum • 8 — Maximum • 9 — Range • 10 — Minority • 11 — Majority • 12 — Variety • 13 — Q1 • 14 — Q3 • 15 — IQR
Output column prefix	`COLUMN_PREFIX`	[string] Default: 'm_'	The prefix for the output (M) column
Extracted	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Extracted	`OUTPUT`	[same as input]	The output vector layer (with M values)

Python code

Algorithm ID: `native:extractmvalues`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Extract specific vertices

Takes a vector layer and generates a point layer with points representing specific vertices in the input geometries.

For instance, this algorithm can be used to extract the first or last vertices in the geometry. The attributes associated to each point are the same ones associated to the feature that the vertex belongs to.

The vertex indices parameter accepts a comma separated string specifying the indices of the vertices to extract. The first vertex corresponds to an index of 0, the second vertex has an index of 1, etc. Negative indices can be used to find vertices at the end of the geometry, e.g., an index of -1 corresponds to the last vertex, -2 corresponds to the second last vertex, etc.

Additional fields are added to the vertices indicating the specific vertex position (e.g., 0, -1, etc), the original vertex index, the vertex's part and its index within the part (as well as its ring for polygons), distance along the original geometry and bisector angle of vertex for the original geometry.

Allows *features in-place modification* of point features

Warning

This algorithm drops existing primary keys or FID values and regenerates them in output layers.

See also

Extract vertices, *Filter vertices by M value*, *Filter vertices by Z value*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Vertex indices	`VERTICES`	[string] Default: '0'	Comma-separated string of the indices of the vertices to extract.
Vertices	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Vertices	`OUTPUT`	[vector: point]	The output (point) vector layer containing the specified vertices from the input layer geometries. Other than the input attributes, the output layer also contains the following fields: • `vertex_pos`: the index of the vertex in the geometry, as requested by the user. • `vertex_index`: the index of the vertex within the geometry. The first vertex is indexed as 0. • `vertex_part`: the part number of the geometry that the vertex belongs to. For single-part geometries, this is always 0. For multi-part geometries (e.g. MultiPolygon or MultiLine), this indicates which part the vertex comes from. • `vertex_part_ring`: for polygon geometries, this indicates whether the vertex belongs to the outer ring (0) or one of the inner rings (1, 2, …). • `vertex_part_index`: the index of the vertex in its specific ring (outer or inner). • `distance`: the cumulative length of line segments from the first vertex of the geometry. • `angle`: the angle (in degrees) between the previous and the current vertex.

Python code

Algorithm ID: `native:extractspecificvertices`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Extract vertices

Takes a vector layer and generates a point layer with points representing the vertices in the input geometries.

The attributes associated to each point are the same ones associated to the feature that the vertex belongs to.

Additional fields are added to the vertices indicating the vertex index (beginning at 0), the feature's part and its index within the part (as well as its ring for polygons), distance along original geometry and bisector angle of vertex for original geometry.

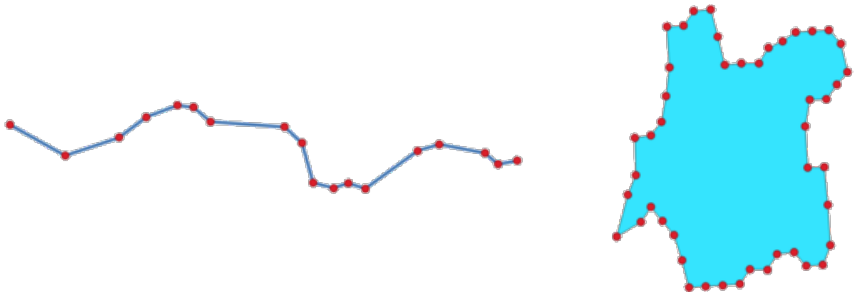


Fig. 24.98: Vertices extracted for line and polygon layer

Allows *features in-place modification* of point features

Default menu: *Vector* ► *Geometry Tools*

Warning

This algorithm drops existing primary keys or FID values and regenerates them in output layers.

See also

Extract specific vertices, *Filter vertices by M value*, *Filter vertices by Z value*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Vertices	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Vertices	`OUTPUT`	[vector: point]	The output (point) vector layer containing the vertices from the input layer geometries. Other than the input attributes, the output layer also contains the following fields: • `vertex_index`: the index of the vertex within its part. The first vertex is indexed as 0. • `vertex_part`: the part number of the geometry that the vertex belongs to. For single-part geometries, this is always 0. For multi-part geometries (e.g. MultiPolygon or MultiLine), this indicates which part the vertex comes from. • `vertex_part_ring`: for polygon geometries, this indicates whether the vertex belongs to the outer ring (0) or one of the inner rings (1, 2, …). • `vertex_part_index`: is the index of a vertex in a specific ring. • `distance`: the distance from the previous vertex. • `angle`: returns approximate angle at a vertex. Usually the average angle between adjacent segments, and can be pictured as the orientation of a line following the curvature of the geometry at the specified vertex.

Python code

Algorithm ID: `native:extractvertices`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Extract Z values

Extracts Z values from geometries into feature attributes.

By default only the Z value from the first vertex of each feature is extracted, however the algorithm can optionally calculate statistics on all of the geometry's Z values, including sum, mean, minimum and maximum.

See also

Extract M values, *Set Z value*, *Drop M/Z values*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Summaries to calculate	`SUMMARIES`	[enumeration] Default: [0]	Statistics on the Z values of a geometry. One or more of: • 0 — First • 1 — Last • 2 — Count • 3 — Sum • 4 — Mean • 5 — Median • 6 — St.dev (pop) • 7 — Minimum • 8 — Maximum • 9 — Range • 10 — Minority • 11 — Majority • 12 — Variety • 13 — Q1 • 14 — Q3 • 15 — IQR
Output column prefix	`COLUMN_PREFIX`	[string] Default: 'z_'	The prefix for the output (Z) column
Extracted	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Extracted	`OUTPUT`	[same as input]	The output vector layer (with Z values)

Python code

Algorithm ID: `native:extractzvalues`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Filter vertices by M value

Filters away vertices based on their M value, returning geometries with only vertex points that have a M value greater than or equal to the specified minimum value and/or less than or equal to the maximum value.

If the minimum value is not specified then only the maximum value is tested, and similarly if the maximum value is not specified then only the minimum value is tested.

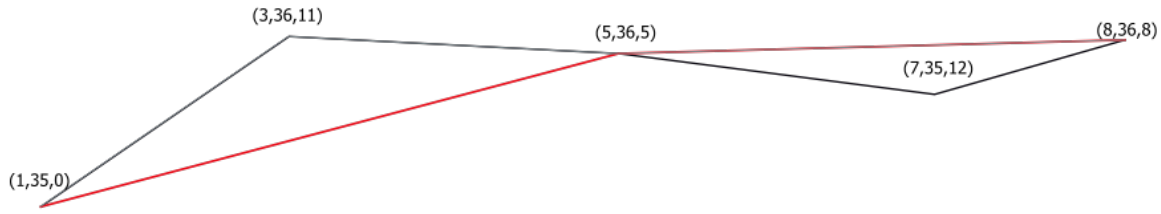


Fig. 24.99: The red line represents the black line with only vertices whose M value is <=10.

Allows *features in-place modification* of point, line and polygon features with M enabled

Warning

Depending on the input geometry attributes and the filters used, the geometries created by this algorithm may no longer be valid.

See also

Filter vertices by Z value, *Extract vertices*, *Extract specific vertices*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer to remove vertices from
Minimum Optional	`MIN`	[numeric: double]  Default: Not set	Minimum of M values allowed
Maximum Optional	`MAX`	[numeric: double]  Default: Not set	Maximum of M values allowed
Filtered	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Filtered	`OUTPUT`	[same as input]	The output vector layer of features with only the filtered vertices.

Python code

Algorithm ID: `native:filterverticesbym`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Filter vertices by Z value

Filters away vertices based on their Z value, returning geometries with only vertex points that have a Z value greater than or equal to the specified minimum value and/or less than or equal to the maximum value.

If the minimum value is not specified then only the maximum value is tested, and similarly if the maximum value is not specified then only the minimum value is tested.

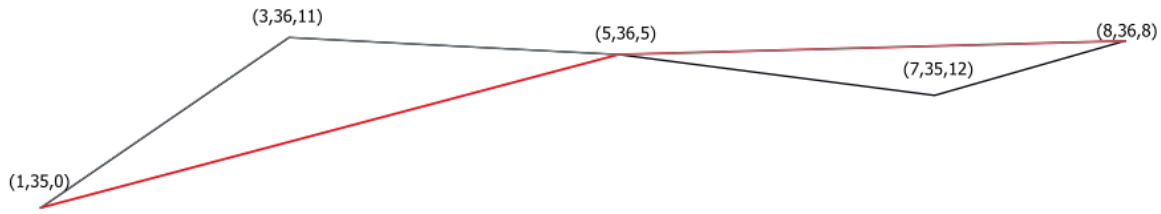


Fig. 24.100: The red line represents the black line with only vertices whose Z value is <=10.

Allows *features in-place modification* of point, line and polygon features with Z enabled

Warning

Depending on the input geometry attributes and the filters used, the geometries created by this algorithm may no longer be valid. You may need to run the *Fix geometries* algorithm to ensure their validity.

See also

Filter vertices by M value, *Extract vertices*, *Extract specific vertices*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer to remove vertices from
Minimum Optional	`MIN`	[numeric: double] Default: Not set	Minimum of Z values allowed
Maximum Optional	`MAX`	[numeric: double] Default: Not set	Maximum of Z values allowed
Filtered	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Filtered	`OUTPUT`	[same as input]	The output vector layer of features with only the filtered vertices.

Python code

Algorithm ID: `native:filterverticesbyz`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Fix geometries

Attempts to create a valid representation of a given invalid geometry without losing any of the input vertices. Already valid geometries are returned without further intervention. Always outputs multi-geometry layer.

Allows *features in-place modification* of point, line, and polygon features without M enabled

Warning

M values will be dropped from the output.

See also

Check validity

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Repair method	`METHOD`	[enumeration] Default: 1	Method used to repair the geometries. One of: • 0 — `Linework`: combines all rings into a set of noded lines and then extracts valid polygons from that linework • 1 — `Structure`: first makes all rings valid and then merges shells and subtracts holes from shells to generate valid result. Assumes that holes and shells are correctly categorized. Requires QGIS version built with GEOS 3.10 or later (check *Help* ► *About* menu).
Fixed geometries	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Fixed geometries	`OUTPUT`	[same as input]	The output vector layer with fixed geometries.

Python code

Algorithm ID: `native:fixgeometries`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Force right-hand-rule

Forces polygon geometries to respect the Right-Hand-Rule, in which the area that is bounded by a polygon is to the right of the boundary. In particular, the exterior ring is oriented in a clockwise direction and any interior rings in a counter-clockwise direction.

Allows *features in-place modification* of polygon features

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: polygon]	Input vector layer
Reoriented	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Reoriented	`OUTPUT`	[vector: polygon]	The output vector layer with reoriented geometries.

Python code

Algorithm ID: `native:forcerhr`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Geodesic line split at antimeridian

Splits a line into multiple geodesic segments, whenever the line crosses the antimeridian (±180 degrees longitude).

Splitting at the antimeridian helps the visual display of the lines in some projections. The returned geometry will always be a multi-part geometry.

Whenever line segments in the input geometry cross the antimeridian, they will be split into two segments, with the latitude of the breakpoint being determined using a geodesic line connecting the points either side of this segment. The current project ellipsoid setting will be used when calculating this breakpoint.

If the input geometry contains M or Z values, these will be linearly interpolated for the new vertices created at the antimeridian.

> **Note**
>
> This algorithm uses ellipsoid based measurements and respects the current *ellipsoid settings*.

Allows *features in-place modification* of line features

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line]	Input line vector layer
Split	`OUTPUT`	[vector: line] Default: `[Create temporary layer]`	Specify the output line vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Split	`OUTPUT`	[vector: line]	The output line vector layer split at the antimeridian.

Python code

Algorithm ID: `native:antimeridiansplit`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Geometry by expression

Updates existing geometries (or creates new geometries) for input features by use of a QGIS expression.

This allows complex geometry modifications which can utilize all the flexibility of the QGIS expression engine to manipulate and create geometries for output features.

For help with QGIS expression functions, see the inbuilt help available in the *expression builder*.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	Input vector layer
Output geometry type	`OUTPUT_GEOMETRY`	[enumeration] Default: 0	The output geometry strongly depends on the expression: for instance, if you create a buffer the geometry type has to be polygon. One of: • 0 — Polygon • 1 — Line • 2 — Point
Output geometry has z values	`WITH_Z`	[boolean] Default: False	Choose if the output geometry should include the Z dimension
Output geometry has m values	`WITH_M`	[boolean] Default: False	Choose if the output geometry should include the M dimension
Geometry expression	`EXPRESSION`	[expression] Default: '$geometry'	Add the geometry expression you want to use. You can use the button to open the Expression Dialog. The dialog lists all the relevant expressions, together with their help and guide.
Modified geometry	`OUTPUT`	[vector: geometry] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Modified geometry	`OUTPUT`	[vector: geometry]	The output vector layer

Python code

Algorithm ID: `native:geometrybyexpression`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Interpolate point on line

Creates a point geometry interpolated at a set distance along line or curve geometries.

Z and M values are linearly interpolated from existing values.

If a multipart geometry is encountered, only the first part is considered when calculating the substring.

If the specified distance is greater than the input feature's length, the resultant feature will have a null geometry.

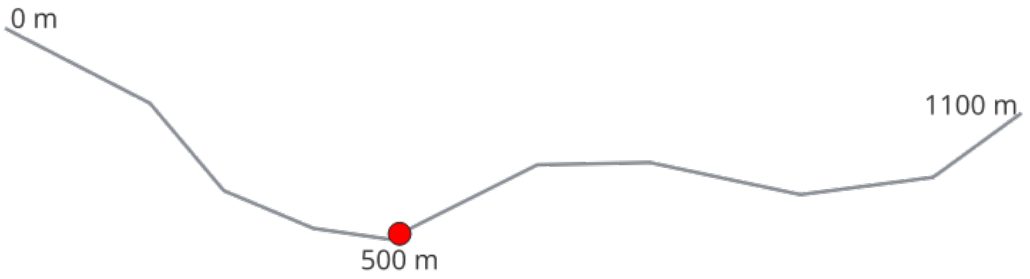


Fig. 24.101: Interpolated point at 500m of the beginning of the line

See also

Points along geometry

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line, polygon]	Input line or polygon vector layer
Distance	`DISTANCE`	[numeric: double] Default: 0.0	Distance from the beginning of the line
Interpolated points	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Interpolated points	`OUTPUT`	[vector: point]	The output point vector layer with features at a set distance along the line or polygon boundary

Python code

Algorithm ID: `native:interpolatepoint`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Keep N biggest parts

Takes a layer with polygons or multipolygons and returns a new layer in which only the *n* largest polygons of each multipolygon feature are kept. If a feature has *n* or fewer parts, the feature will just be copied.

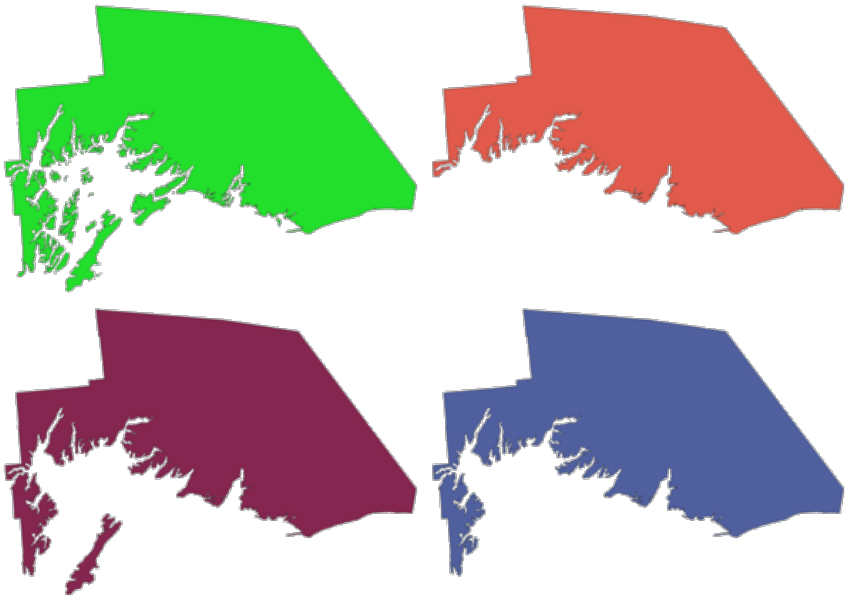


Fig. 24.102: Clockwise from top left: original multipart feature, one, two and three biggest parts kept

Parameters

Label	Name	Type	Description
Polygons	`INPUT`	[vector: polygon]	Input polygon vector layer
Parts to keep	`PARTS`	[numeric: integer]  Default: 1	Number of parts to keep. If 1, only the biggest part of the feature will be kept.
Parts	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output polygon vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Parts	`OUTPUT`	[vector: polygon]	The output polygon vector layer with the N biggest parts of each feature

Python code

Algorithm ID: `qgis:keepnbiggestparts`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Line substring

Returns the portion of a line (or curve) which falls between the specified start and end distances (measured from the beginning of the line).

Z and M values are linearly interpolated from existing values.

If a multipart geometry is encountered, only the first part is considered when calculating the substring.



Fig. 24.103: Substring line with starting distance set at 0 meters and the ending distance at 250 meters.

Allows *features in-place modification* of line features

See also

Extend lines

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line]	Input line vector layer
Start distance	`START_DISTANCE`	[numeric: double] Default: 0.0	Distance along the input line to the start point of the output feature
End distance	`END_DISTANCE`	[numeric: double] Default: 1.0	Distance along the input line to the end point of the output feature
Substring	`OUTPUT`	[vector: line] Default: `[Create temporary layer]`	Specify the output line vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Substring	`OUTPUT`	[vector: line]	The output line vector layer.

Python code

Algorithm ID: `native:linesubstring`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Lines to polygons

Generates a polygon layer using as polygon rings the lines from an input line layer.

The attribute table of the output layer is the same as the one of the input layer.

Default menu: *Vector ► Geometry Tools*

See also

Polygons to lines, *Polygonize*, *Convert geometry type*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line]	Input line vector layer
Polygons	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output polygon vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Polygons	`OUTPUT`	[vector: polygon]	The output polygon vector layer.

Python code

Algorithm ID: `qgis:linestopolygons`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Merge lines

Joins all connected parts of MultiLineString geometries into single LineString geometries.

If any parts of the input MultiLineString geometries are not connected, the resultant geometry will be a MultiLineString containing any lines which could be merged and any non-connected line parts.

Allows *features in-place modification* of line features

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line]	Input line vector layer
Merged	`OUTPUT`	[vector: line] Default: `[Create temporary layer]`	Specify the output line vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Merged	`OUTPUT`	[vector: line]	The output (merged) line vector layer.

Python code

Algorithm ID: `native:mergelines`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Minimum bounding geometry

Creates geometries which enclose the features from an input layer. The features can be grouped by a field. The output layer will then contain one feature per group value with a geometry (MBB) that covers the geometries of the features with matching value.

The following enclosing geometry types are supported:

- bounding box (envelope)
- oriented rectangle
- circle
- convex hull

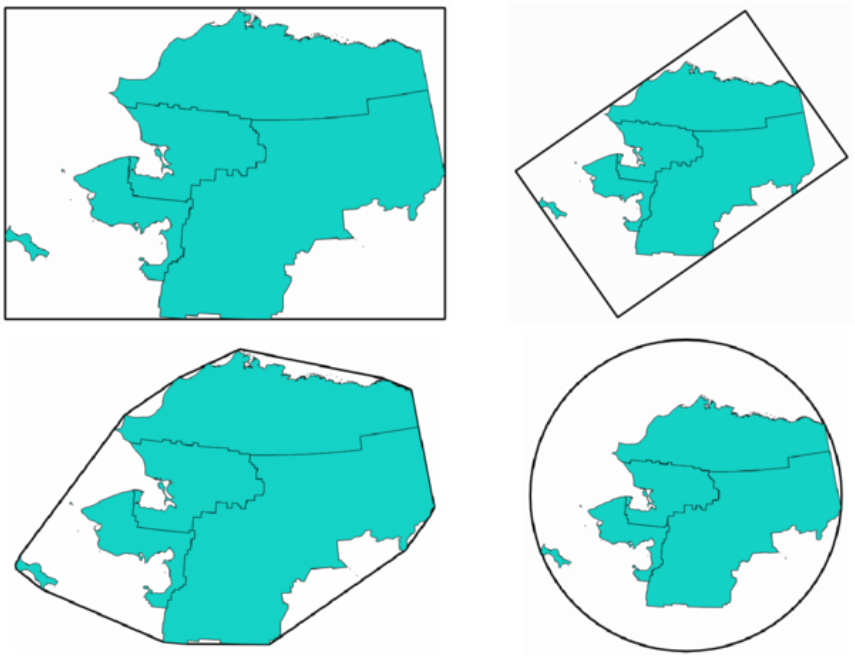


Fig. 24.104: Clockwise from top left: envelope, oriented rectangle, circle, convex hull

See also

Minimum enclosing circles

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Field Optional	`FIELD`	[tablefield: any]	Features can be grouped by a field. If set, this causes the output layer to contain one feature per grouped value with a minimal geometry covering only the features with matching values.
Geometry type	`TYPE`	[enumeration] Default: 0	Enclosing geometry types. One of: • 0 — Envelope (Bounding Box) • 1 — Minimum Oriented Rectangle • 2 — Minimum Enclosing Circle • 3 — Convex Hull
Bounding geometry	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output polygon vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Bounding geometry	`OUTPUT`	[vector: polygon]	The output (bounding) polygon vector layer. It has the fields from the input layer and, depending on the selected parameter `TYPE`, the output also has the attributes: • `width`, `height`, `area` and `perimeter` of the Envelope (Bounding Box) • `width`, `height`, `area`, `angle` and `perimeter` of the Minimum Oriented Rectangle • `radius` and `area` of the Minimum Enclosing Circle • `area` and `perimeter` of the Convex Hull

Python code

Algorithm ID: `qgis:minimumboundinggeometry`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Minimum enclosing circles

Calculates the minimum enclosing circles of the features in the input layer.

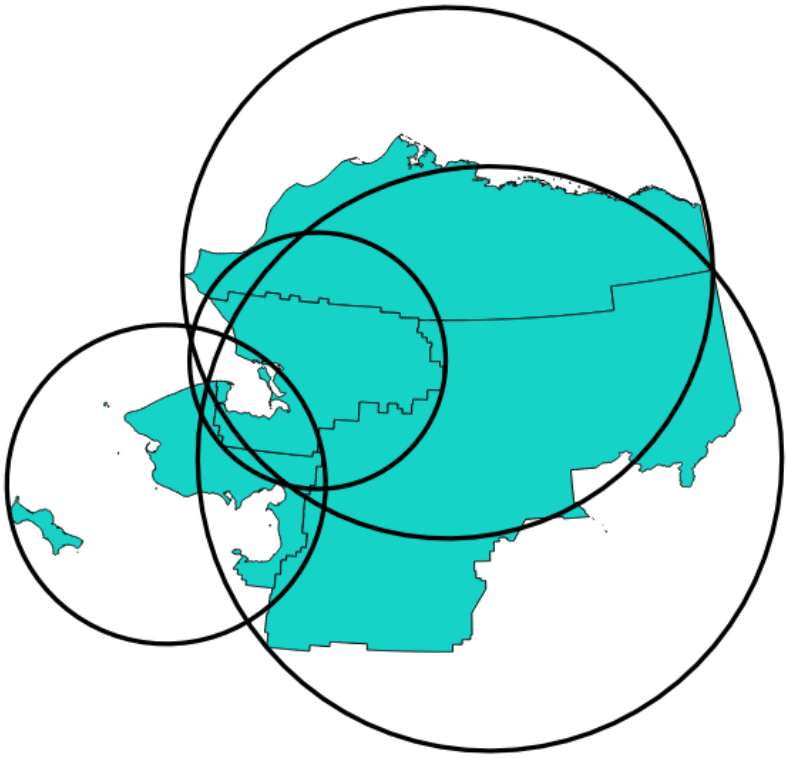


Fig. 24.105: Enclosing circles for each feature

Allows *features in-place modification* of polygon features

See also

Minimum bounding geometry

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Number of segments in circles	`SEGMENTS`	[numeric: integer] Default: 72	The number of segments used to approximate a circle. Minimum 8, maximum 100000.
Minimum enclosing circles	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output polygon vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Minimum enclosing circles	`OUTPUT`	[vector: polygon]	The output polygon vector layer. Other than the input attributes, the output layer also contains the following fields: `area` and `radius` of the generated polygon.

Python code

Algorithm ID: `native:minimumenclosingcircle`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Multi-ring buffer (constant distance)

Computes multi-ring (*donut*) buffer for the features of the input layer, using a fixed or dynamic distance and number of rings.

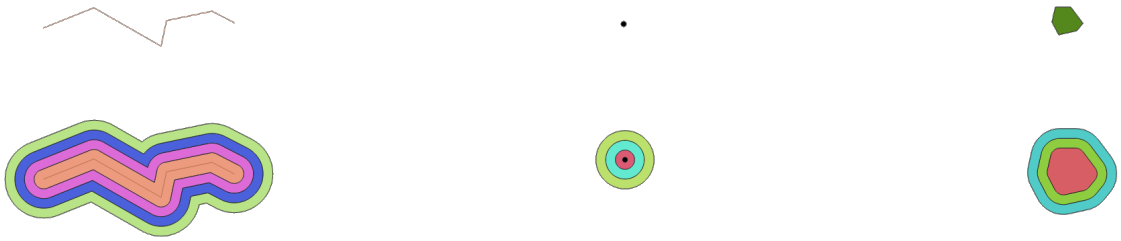


Fig. 24.106: Multi-ring buffer for a line, point and polygon layer

Allows *features in-place modification* of polygon features

Warning

This algorithm drops existing primary keys or FID values and regenerates them in output layers.

See also

Buffer, *Variable distance buffer*, *Rectangles, ovals, diamonds*, *Single sided buffer*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Number of rings	`RINGS`	[numeric: integer] Default: 1	The number of rings. It can be a unique value (same number of rings for all the features) or it can be taken from features data (the number of rings depends on feature values).
Distance between rings	`DISTANCE`	[numeric: double] Default: 1.0	Distance between the rings. It can be a unique value (same distance for all the features) or it can be taken from features data (the distance depends on feature values).
Multi-ring buffer (constant distance)	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output polygon vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Multi-ring buffer (constant distance)	`OUTPUT`	[vector: polygon]	The output polygon vector layer. Other than the input attributes, the output layer also contains the following fields: `ringId` and `distance` of the generated buffer polygon.

Python code

Algorithm ID: `native:multiringconstantbuffer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Multipart to singleparts

Splits multipart features in the input layer into singlepart features.

The attributes of the output layer are the same as the original ones but divided into single features.



Fig. 24.107: Left the multipart source layer and right the single part output result

Allows *features in-place modification* of point, line, and polygon features

Default menu: *Vector* ► *Geometry Tools*

Warning

This algorithm drops existing primary keys or FID values and regenerates them in output layers.

See also

Collect geometries, *Promote to multipart*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Single parts	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output polygon vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Single parts	`OUTPUT`	[same as input]	The output vector layer.

Python code

Algorithm ID: `native:multiparttosingleparts`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Offset lines

Offsets lines by a specified distance. Positive distances will offset lines to the left, and negative distances will offset them to the right.

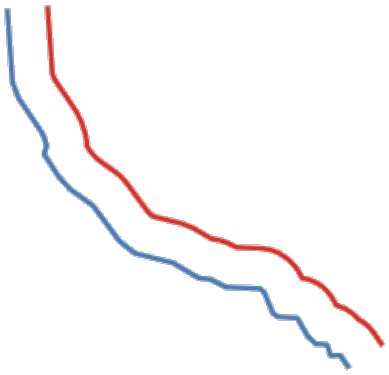


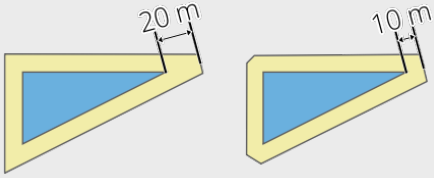
Fig. 24.108: In blue the source layer, in red the offset one

Allows *features in-place modification* of line features

See also

Array of offset (parallel) lines, *Translate*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line]	Input line vector layer
Distance	`DISTANCE`	[numeric: double] Default: 10.0	Offset distance. You can use the Data Defined button on the right to choose a field from which the radius will be calculated. This way you can have different radius for each feature (see *Variable distance buffer*).
Segments	`SEGMENTS`	[numeric: integer] Default: 8	Controls the number of line segments to use to approximate a quarter circle when creating rounded offsets.
Join style	`JOIN_STYLE`	[enumeration] Default: 0	Specifies whether round, miter or beveled joins should be used when offsetting corners in a line. Options are: • 0 — Round • 1 — Miter • 2 — Bevel  Fig. 24.109: Round, miter, and bevel join styles
Miter limit	`MITER_LIMIT`	[numeric: double] Default: 2.0	Sets the maximum distance from the offset geometry to use when creating a mitered join as a factor of the offset distance (only applicable for miter join styles). Minimum: 1.0  Fig. 24.110: A 10m buffer with a limit of 2 and a 10m buffer with a limit of 1
Offset	`OUTPUT`	[vector: line] Default: `[Create temporary layer]`	Specify the output (offset) layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Offset	`OUTPUT`	[vector: line]	Output (offset) line layer

Python code

Algorithm ID: `native:offsetline`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Oriented minimum bounding box

Calculates the minimum area rotated rectangle for each feature in the input layer.

> **Note**
>
> For singlepart point features, the output corresponds to the bounding box of the geometry.

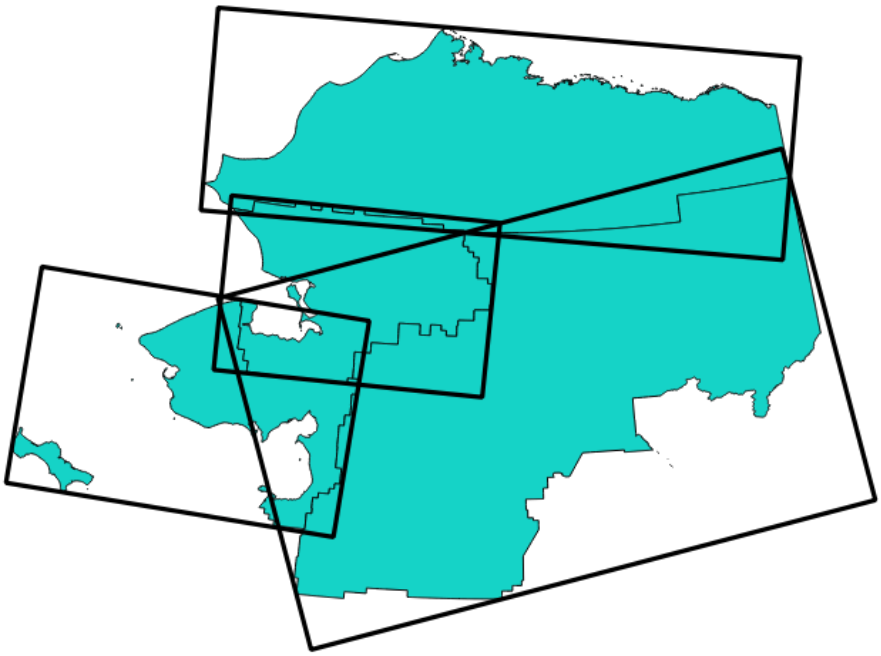


Fig. 24.111: Oriented minimum bounding box

Allows *features in-place modification* of polygon features

See also

Minimum bounding geometry

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Bounding boxes	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output polygon vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Bounding boxes	`OUTPUT`	[vector: polygon]	The output polygon vector layer. Other than the input attributes, the output layer also contains the following fields: `width`, `height`, `angle`, `area` and `perimeter` of the generated polygon.

Python code

Algorithm ID: `native:orientedminimumboundingbox`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Orthogonalize

Attempts to orthogonalize the geometries of the input line or polygon layer. This process shifts the vertices in the geometries to try to make every angle in the geometry either a right angle or a straight line.

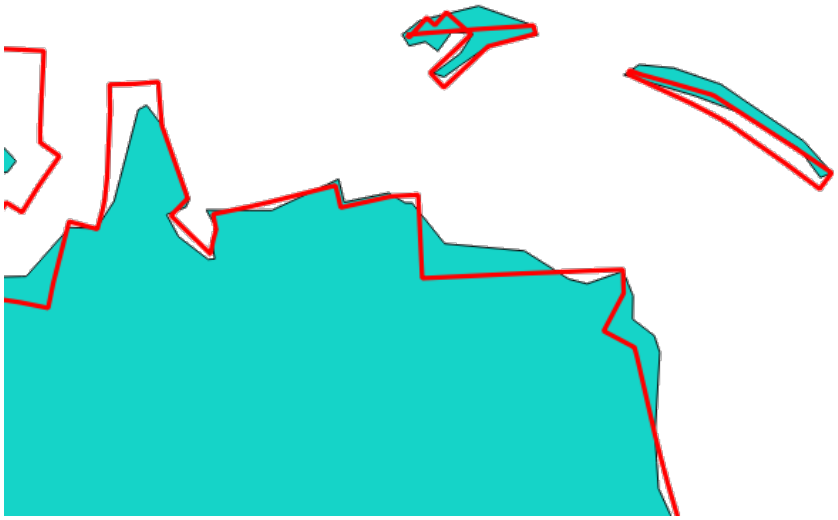


Fig. 24.112: In blue the source layer and in the red orthogonalized result

Allows *features in-place modification* of line and polygon features

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line, polygon]	Input line or polygon vector layer
Maximum angle tolerance (degrees)	`ANGLE_TOLERANCE`	[numeric: double] Default: 15.0	Specify the maximum deviation from a right angle or straight line a vertex can have for it to be adjusted. Smaller tolerances mean that only vertices which are already closer to right angles will be adjusted, and larger tolerances mean that vertices which deviate further from right angles will also be adjusted.
Maximum algorithm iterations	`MAX_ITERATIONS`	[numeric: integer] Default: 1000	Setting a larger number for the maximum number of iterations will result in a more orthogonal geometry at the cost of extra processing time.
Orthogonalized	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output polygon vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Orthogonalized	`OUTPUT`	[same as input]	The output polygon vector layer with adjusted angles.

Python code

Algorithm ID: `native:orthogonalize`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Point on Surface

For each feature of the input layer, returns a point that is guaranteed to lie on the surface of the feature geometry.

Allows *features in-place modification* of point features

Warning

This algorithm may drop existing primary keys or FID values and regenerate them in output layers, depending on the input parameters.

See also

Centroids

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Create point on surface for each part	`ANGLE_TOLERANCE`	[boolean]	If checked, a point will be created for each part of the geometry.
Point	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specify the output point vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Point	`OUTPUT`	[vector: point]	The output point vector layer.

Python code

Algorithm ID: `native:pointonsurface`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Points along geometry

Creates points at regular intervals along line or polygon geometries. Created points will have new attributes added for the distance along the geometry and the angle of the line at the point.

An optional start and end offset can be specified, which controls how far from the start and end of the geometry the points should be created.

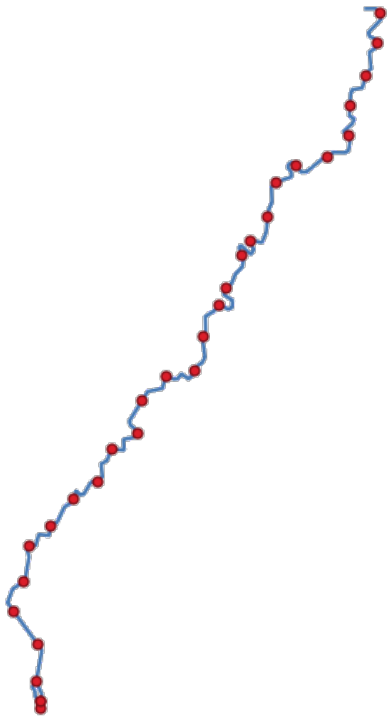


Fig. 24.113: Points created along the source line layer

Warning

This algorithm drops existing primary keys or FID values and regenerates them in output layers.

See also

Interpolate point on line

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line, polygon]	Input line or polygon vector layer
Distance	`DISTANCE`	[numeric: double] Default: 1.0	Distance between two consecutive points along the line
Start offset	`START_OFFSET`	[numeric: double] Default: 0.0	Distance from the beginning of the input line, representing the position of the first point.
End offset	`END_OFFSET`	[numeric: double] Default: 0.0	Distance from the end of the input line, representing the position beyond which no point feature shoud be created.
Interpolated points	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Interpolated points	`OUTPUT`	[vector: point]	Point vector layer with features placed along lines or polygon boundaries of the input layer. Other than the input attributes on every related point, the output layer also contains the following fields: `distance` and `angle` of the generated point.

Python code

Algorithm ID: `native:pointsalonglines`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Points displacement

Given a distance of proximity, identifies nearby point features and radially distributes them over a circle whose center represents their barycenter. A convenient tool to scatter overlaid features.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: point]	Input point vector layer
Minimum distance to other points	`PROXIMITY`	[numeric: double] Default: 1.0	Distance below which point features are considered close. Close features are distributed altogether.
Displacement distance	`DISTANCE`	[numeric: double] Default: 1.0	Radius of the circle on which close features are placed
Horizontal distribution for two point case	`HORIZONTAL`	[boolean] Default: False	When only two points are identified as close, aligns them horizontally on the circle instead of vertically.
Displaced	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Displaced	`OUTPUT`	[vector: point]	Output point vector layer

Python code

Algorithm ID: `qgis:pointsdisplacement`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Pole of inaccessibility

Calculates the pole of inaccessibility for a polygon layer, which is the most distant internal point from the boundary of the surface.

This algorithm uses the 'polylabel' algorithm (Vladimir Agafonkin, 2016), which is an iterative approach guaranteed to find the true pole of inaccessibility within a specified tolerance. A more precise tolerance (lower value) requires more iterations and will take longer to calculate.



Fig. 24.114: Pole of inaccessibility

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: polygon]	Input vector layer
Tolerance	`TOLERANCE`	[numeric: double] Default: 1.0	Set the tolerance for the calculation
Point	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specify the output polygon vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Point	`OUTPUT`	[vector: point]	The output point vector layer. It will contain a new attribute with the distance `dist_pole` from the calculated pole to the polygon boundary.

Python code

Algorithm ID: `native:poleofinaccessibility`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Polygonize

Creates a polygon layer whose features boundaries are generated from a line layer of **closed** features.

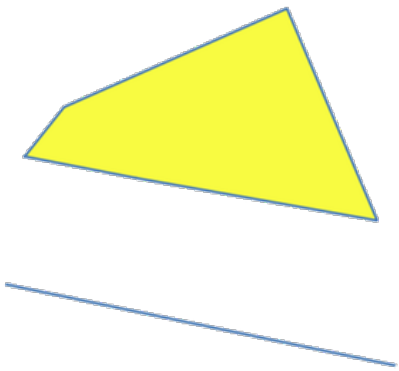


Fig. 24.115: The yellow polygons generated from the closed lines

Note

The line layer must have closed shapes in order to be transformed into a polygon.

See also

Polygons to lines, *Lines to polygons*, *Convert geometry type*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line]	Input line vector layer
Keep fields from the input layer	`KEEP_FIELDS`	[boolean] Default: False	Check to keep the fields (only the table structure, not the values) of the input layer
Polygons from lines	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output polygon vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Polygons from lines	`OUTPUT`	[vector: polygon]	The output polygon vector layer from lines

Python code

Algorithm ID: `native:polygonize`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Polygons to lines

Takes a polygon layer and creates a line layer, with lines representing the boundaries of the polygons in the input layer.

The attribute table of the output layer is the same as the one of the input layer.



Fig. 24.116: Black lines as the result of the algorithm

Default menu: *Vector ► Geometry Tools*

See also

Lines to polygons, *Polygonize*, *Convert geometry type*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: polygon]	Input polygon vector layer
Lines	`OUTPUT`	[vector: line] Default: `[Create temporary layer]`	Specify the output line vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Lines	`OUTPUT`	[vector: line]	The output line vector layer from polygons

Python code

Algorithm ID: `native:polygonstolines`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Project points (Cartesian)

Projects point geometries by a specified distance and bearing (azimuth).

Allows *features in-place modification* of point features

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: point]	Input point vector layer
Bearing (degrees from North)	`BEARING`	[numeric: double] Default: 0.0	Clockwise angle starting from North, in degree (°) unit
Distance	`DISTANCE`	[numeric: double] Default: 1.0	Distance to offset geometries, in layer units
Projected	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specify the output point vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Projected	`OUTPUT`	[vector: point]	The output (projected) point vector layer

Python code

Algorithm ID: `native:projectpointcartesian`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Promote to multipart

Takes a vector layer with singlepart geometries and generates a new one in which all geometries are multipart.

Input features which are already multipart features will remain unchanged.

This algorithm can be used to force geometries to multipart types in order to be compatible with data providers that require multipart features.

Allows *features in-place modification* of point, line, and polygon features

See also

Aggregate, *Collect geometries*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Multiparts	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output multipart vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Multiparts	`OUTPUT`	[same as input]	The output multipart vector layer

Python code

Algorithm ID: `native:promotetomulti`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Rectangles, ovals, diamonds

Creates a buffer area with a rectangle, oval or diamond shape for each feature of the input point layer.

The shape parameters can be fixed for all features or dynamic using a field or an expression.

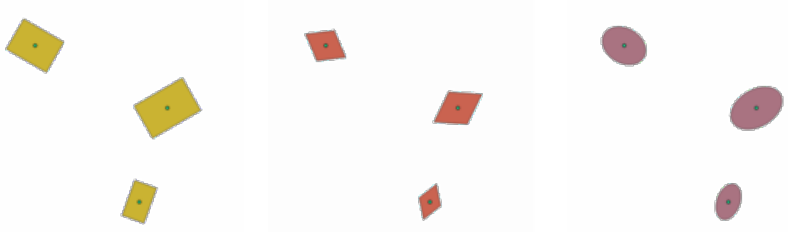


Fig. 24.117: Different buffer shapes with dynamic parameters

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: point]	Input point vector layer
Buffer shape	`SHAPE`	[enumeration]	The shape to use. One of: • 0 — Rectangles • 1 — Ovals • 2 — Diamonds
Width	`WIDTH`	[numeric: double] Default: 1.0	Width of the buffer shape
Height	`HEIGHT`	[numeric: double] Default: 1.0	Height of the buffer shape
Rotation Optional	`ROTATION`	[numeric: double] Default: 0.0	Rotation of the buffer shape
Number of segments	`SEGMENTS`	[numeric: integer] Default: 36	Number of segments for a full circle (*Ovals* shape)
Output	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Output	`OUTPUT`	[vector: polygon]	The output vector layer (with the buffer shapes)

Python code

Algorithm ID: `native:rectanglesovalsdiamonds`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Remove duplicate vertices

Removes duplicate vertices from features, wherever removing the vertices does not result in a degenerate geometry.

The tolerance parameter specifies the tolerance for coordinates when determining whether vertices are identical.

By default, Z values are not considered when detecting duplicate vertices. E.g. two vertices with the same X and Y coordinate but different Z values will still be considered duplicate and one will be removed. If the *Use Z Value* parameter is true, then the Z values are also tested and vertices with the same X and Y but different Z will be maintained.

 Allows *features in-place modification* of point, line, and polygon features

Note

Duplicate vertices are not tested between different parts of a multipart geometry, e.g. a multipoint geometry with overlapping points will not be changed by this method.

See also

Extract vertices, *Extract specific vertices*, *Delete duplicate geometries*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Tolerance	`TOLERANCE`	[numeric: double]  Default: 0.000001	Vertices closer than the specified distance are considered duplicates
Use Z value	`USE_Z_VALUE`	[boolean]  Default: False	If the *Use Z Value* parameter is true, then the Z values are also tested and vertices with the same X and Y but different Z will be maintained.
Cleaned	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Cleaned	`OUTPUT`	[same as input]	The output vector layer (without duplicate vertices)

Python code

Algorithm ID: `native:removeduplicatevertices`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Remove null geometries

Removes any features which do not have a geometry from a vector layer. All other features will be copied unchanged.

The features with null geometries can be saved to a separate layer.

If *Also remove empty geometries* is checked, the algorithm removes features whose geometries have no coordinates, i.e., geometries that are empty. In that case, also the null output will reflect this option, containing both null and empty geometries.

See also

Delete duplicate geometries

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer (with non-NULL geometries)
Also remove empty geometries	`REMOVE_EMPTY`	[boolean]	
Non null geometries	`OUTPUT` Optional	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer for the non-NULL (and non-empty) geometries. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

continues on next page

Table 24.241 – continued from previous page

Label	Name	Type	Description
Null geometries Optional	NULL_OUTPUT	[same as input] Default: [Skip output]	Specify the output vector layer for the NULL (and empty) geometries. *One of*: • Skip Output • Create Temporary Layer (TEMPORARY_OUTPUT) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Null geometries	NULL_OUTPUT	[same as input]	Output vector layer (for NULL and, if chosen, empty geometries)
Non null geometries	OUTPUT	[same as input]	The output vector layer (without NULL and, if chosen, empty geometries)

Python code

Algorithm ID: native:removenullgeometries

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Reverse line direction

Inverts the direction of a line layer.

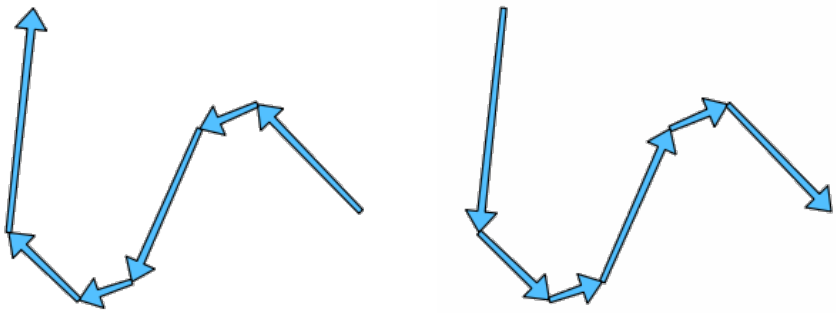


Fig. 24.118: Before and after the direction inversion

Allows *features in-place modification* of line features

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line]	Input line vector layer
Reversed	`OUTPUT`	[vector: line] Default: `[Create temporary layer]`	Specify the output line vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Reversed	`OUTPUT`	[vector: line]	The output line vector layer (with reversed lines)

Python code

Algorithm ID: `native:reverselinedirection`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Rotate

Rotates feature geometries by the specified angle clockwise. The rotation occurs around each feature's centroid, or optionally around a unique preset point.

Allows *features in-place modification* of point, line, and polygon features

See also

Translate, *Swap X and Y coordinates*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Rotation (degrees clockwise)	`ANGLE`	[numeric: double] Default: 0.0	Angle of the rotation in degrees
Rotation anchor point (x, y) Optional	`ANCHOR`	[point] Default: Not set	X,Y coordinates of the point to rotate the features around. If not set the rotation occurs around each feature's centroid.
Rotated	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer (with rotated geometries). *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Rotated	`OUTPUT`	[same as input]	The output vector layer with rotated geometries

Python code

Algorithm ID: `native:rotatefeatures`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Roundness

Calculates the roundness of each feature and stores it as a new field `roundness`. The input vector layer must contain polygons.

The roundness of a polygon is defined as $4\pi \times$ polygon area / perimeter^2. The roundness value varies between 0 and 1. A perfect circle has a roundness of 1, while a completely flat polygon has a roundness of 0.

Note

The algorithm returns NULL for multipart polygon features.

Allows *features in-place modification* of polygon features

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: polygon]	Input vector layer
Roundness	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output vector layer (with roundness field). *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Roundness	`OUTPUT`	[same as input]	The output vector layer with `roundness` value in a field

Python code

Algorithm ID: `native:roundness`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Segmentize by maximum angle

Segmentizes a geometry by converting curved sections to linear sections.

The segmentization is performed by specifying the maximum allowed radius angle between vertices on the straightened geometry (e.g the angle of the arc created from the original arc center to consecutive output vertices on the linearized geometry). Non-curved geometries will be retained without change.

See also

Segmentize by maximum distance, *Simplify*, *Smooth*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line, polygon]	Input line or polygon vector layer
Maximum angle between vertices (degrees)	`ANGLE`	[numeric: double] Default: 5.0	Maximum allowed radius angle between vertices on the straightened geometry
Segmentized	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer (with segmentized geometries). *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Segmentized	`OUTPUT`	[same as input]	The output vector layer with segmentized geometries

Python code

Algorithm ID: `native:segmentizebymaxangle`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Segmentize by maximum distance

Segmentizes a geometry by converting curved sections to linear sections.

The segmentization is performed by specifying the maximum allowed offset distance between the original curve and the segmentized representation. Non-curved geometries will be retained without change.

See also

Segmentize by maximum angle, *Simplify*, *Smooth*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line, polygon]	Input line or polygon vector layer
Maximum offset distance	`DISTANCE`	[numeric: double] Default: 1.0	Maximum allowed offset distance between the original curve and the segmentized representation, in the layer units.
Segmentized	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer (with segmentized geometries). *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Segmentized	`OUTPUT`	[same as input]	The output vector layer with segmentized geometries

Python code

Algorithm ID: `native:segmentizebymaxdistance`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Set M value

Sets the M value for geometries in a layer.

If M values already exist in the layer, they will be overwritten with the new value. If no M values exist, the geometry will be upgraded to include M values and the specified value used as the initial M value for all geometries.

Allows *features in-place modification* of point, line, and polygon features with M enabled

Tip

Use the Identify Features button to check the added M value: the results are available in the *Identify Results* dialog.

See also

Set M value from raster, *Set Z value*, *Drop M/Z values*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
M Value	`M_VALUE`	[numeric: double] Default: 0.0	M value to assign to the feature geometries
M Added	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
M Added	`OUTPUT`	[same as input]	The output vector layer (with M values assigned to the geometries)

Python code

Algorithm ID: `native:setmvalue`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Set M value from raster

Uses values sampled from a band within a raster layer to set the M value for every overlapping vertex in the feature geometry. The raster values can optionally be scaled by a preset amount.

If M values already exist in the layer, they will be overwritten with the new value. If no M values exist, the geometry will be upgraded to include M values.

Allows *features in-place modification* of point, line, and polygon features with M enabled

See also

Drape (set Z value from raster), *Set M value*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Raster layer	`RASTER`	[raster]	Raster layer with M values
Band number	`BAND`	[raster band] Default: 1	The raster band from which the M values are taken
Value for No-Data or non-intersecting vertices	`NODATA`	[numeric: double] Default: 0.0	Value to use in case the vertex does not intersect (a valid pixel of) the raster
Scale factor	`SCALE`	[numeric: double] Default: 1.0	Scaling value: the band values are multiplied by this value.
Offset	`OFFSET`	[numeric: double] Default: 0.0	Offset value: it is algebraically added to the band values after applying the "Scale factor".
Updated	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer (with updated M values). *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Updated	`OUTPUT`	[same as input]	The output vector layer (with updated M values)

Python code

Algorithm ID: `native:setmfromraster`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Set Z value

Sets the Z value for geometries in a layer.

If Z values already exist in the layer, they will be overwritten with the new value. If no Z values exist, the geometry will be upgraded to include Z values and the specified value used as the initial Z value for all geometries.

Allows *features in-place modification* of point, line, and polygon features with Z enabled

Tip

Use the 

button to check the added Z value: the results are available in the *Identify Results* dialog.

See also

Drape (set Z value from raster), *Set M value*, *Drop M/Z values*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Z Value	`Z_VALUE`	[numeric: double] Default: 0.0	Z value to assign to the feature geometries
Z Added	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Z Added	`OUTPUT`	[same as input]	The output vector layer (with Z values assigned)

Python code

Algorithm ID: `native:setzvalue`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Simplify

Simplifies the geometries in a line or polygon layer. It creates a new layer with the same features as the ones in the input layer, but with geometries containing a lower number of vertices.

The algorithm gives a choice of simplification methods, including distance based (the "Douglas-Peucker" algorithm), area based ("Visvalingam" algorithm) and snapping geometries to grid.



Fig. 24.119: From left to right, source layer and increasing simplification tolerances

Allows *features in-place modification* of line and polygon features

Default menu: *Vector* ► *Geometry Tools*

See also

Simplify coverage, *Smooth*, *Densify by count*, *Densify by interval*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line, polygon]	Input line or polygon vector layer
Simplification method	`METHOD`	[enumeration] Default: 0	Simplification method. One of: • 0 — Distance (Douglas-Peucker) • 1 — Snap to grid • 2 — Area (Visvalingam)
Tolerance	`TOLERANCE`	[numeric: double]  Default: 1.0	Threshold tolerance (in units of the layer): if the distance between two nodes is smaller than the tolerance value, the segment will be simplified and vertices will be removed.
Simplified	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output (simplified) vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Simplified	`OUTPUT`	[same as input]	The output (simplified) vector layer

Python code

Algorithm ID: `native:simplifygeometries`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Single sided buffer

Computes a buffer on lines by a specified distance on one side of the line only.

Buffer always results in a polygon layer.



Fig. 24.120: Left versus right side buffer on the same vector line layer

See also

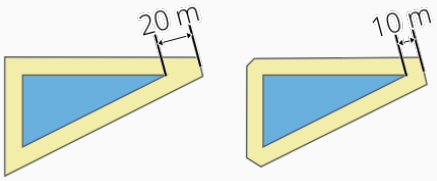
Buffer

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line]	Input line vector layer
Distance	`DISTANCE`	[numeric: double]  Default: 10.0	Buffer distance.
Side	`SIDE`	[enumeration] Default: 0	Which side to create the buffer on. One of: • 0 – Left • 1 – Right
Segments	`SEGMENTS`	[numeric: integer] Default: 8	Controls the number of line segments to use to approximate a quarter circle when creating rounded offsets.
Join style	`JOIN_STYLE`	[enumeration] Default: 0	Specifies whether round, miter or beveled joins should be used when offsetting corners in a line. Options are: • 0 — Round • 1 — Miter • 2 — Bevel  Fig. 24.121: Round, miter, and bevel join styles

continues on next page

Table 24.244 – continued from previous page

Label	Name	Type	Description
Miter limit	`MITER_LIMIT`	[numeric: double] Default: 2.0	Sets the maximum distance from the offset geometry to use when creating a mitered join as a factor of the offset distance (only applicable for miter join styles). Minimum: 1.0  Fig. 24.122: A 10m buffer with a limit of 2 and a 10m buffer with a limit of 1
Buffer	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output (buffer) layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Buffer	`OUTPUT`	[vector: polygon]	Output (buffer) polygon layer

Python code

Algorithm ID: `native:singlesidedbuffer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Smooth

Smooths the geometries in a line or polygon layer by adding more **vertices and corners** to the feature geometries.

The iterations parameter dictates how many smoothing iterations will be applied to each geometry. A higher number of iterations results in smoother geometries with the cost of greater number of nodes in the geometries.

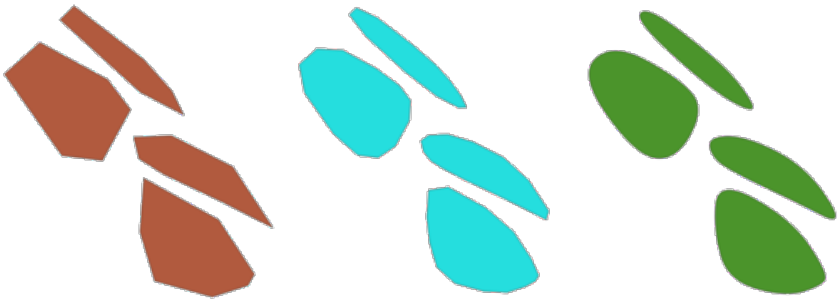


Fig. 24.123: Increasing number of iterations causes smoother geometries

The offset parameter controls how "tightly" the smoothed geometries follow the original geometries. Smaller values results in a tighter fit, and larger values will create a looser fit.

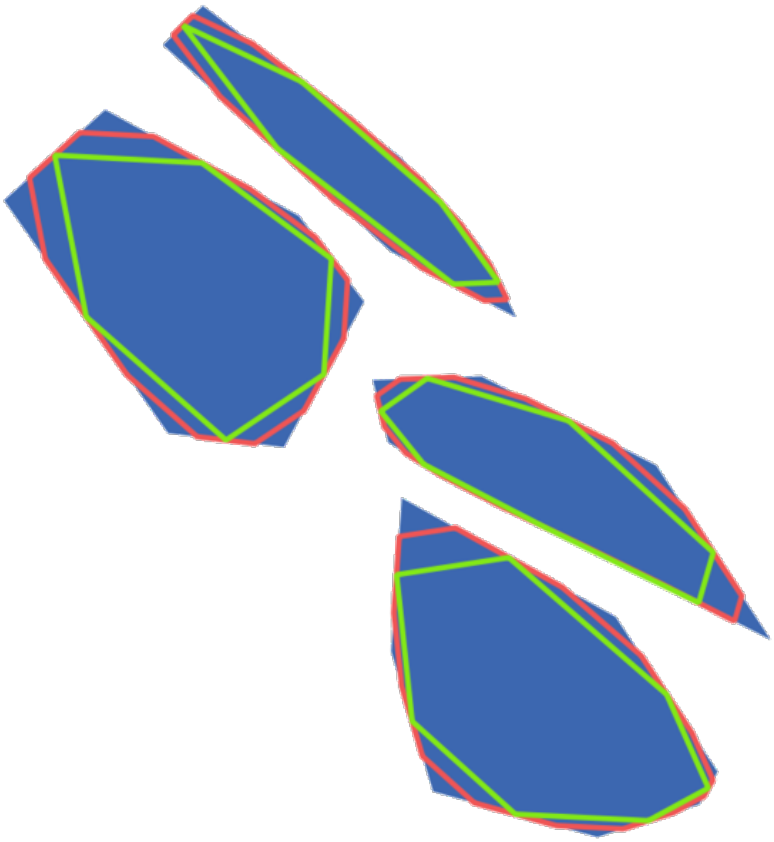


Fig. 24.124: Blue: the input layer. Offset 0.25 gives the red line, while offset 0.50 gives the green line.

The maximum angle parameter can be used to prevent smoothing of nodes with large angles. Any node where the angle of the segments to either side is larger than this will not be smoothed. For example, setting the maximum angle to 90 degrees or lower would preserve right angles in the geometry.

Allows *features in-place modification* of line and polygon features

See also

Simplify, *Simplify coverage*, *Densify by count*, *Densify by interval*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line, polygon]	Input line or polygon vector layer
Iterations	`ITERATIONS`	[numeric: integer] Default: 1	Increasing the number of iterations will give smoother geometries (and more vertices).
Offset	`OFFSET`	[numeric: double] Default: 0.25	Increasing values will *move* the smoothed lines / boundaries further away from the input lines / boundaries.
Maximum node angle to smooth	`MAX_ANGLE`	[numeric: double] Default: 180.0	Every node below this value will be smoothed
Smoothed	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output (smoothed) layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Smoothed	`OUTPUT`	[same as input]	Output (smoothed) vector layer

Python code

Algorithm ID: `native:smoothgeometry`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Snap geometries to layer

Snaps the geometries in a layer either to the geometries from another layer, or to geometries within the same layer.

Matching is done based on a tolerance distance, and vertices will be inserted or removed as required to make the geometries match the reference geometries.

Allows *features in-place modification* of point, line, and polygon features

See also

Snap points to grid

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Reference layer	`REFERENCE_LAYER`	[vector: geometry]	Vector layer to snap to
Tolerance	`TOLERANCE`	[numeric: double] Default: 10.0	Control how close input vertices need to be to the reference layer geometries before they are snapped.

continues on next page

Table 24.246 – continued from previous page

Label	Name	Type	Description
Behavior	`BEHAVIOR`	[enumeration] Default: 0	Snapping can be done to an existing node or a segment (its closest point to the vertex to move). Available snapping options: • 0 — Prefer aligning nodes, insert extra vertices where required Prefer to snap to nodes, even when a segment may be closer than a node. New nodes will be inserted to make geometries follow each other exactly when inside allowable tolerance. • 1 — Prefer closest point, insert extra vertices where required Snap to closest point, regardless of it is a node or a segment. New nodes will be inserted to make geometries follow each other exactly when inside allowable tolerance. • 2 — Prefer aligning nodes, don't insert new vertices Prefer to snap to nodes, even when a segment may be closer than a node. No new nodes will be inserted. • 3 — Prefer closest point, don't insert new vertices Snap to closest point, regardless of it is a node or a segment. No new nodes will be inserted. • 4 — Move end points only, prefer aligning nodes Only snap start/end points of lines (point features will also be snapped, polygon features will not be modified), prefer to snap to nodes. • 5 — Move end points only, prefer closest point Only snap start/end points of lines (point features will also be snapped, polygon features will not be modified), snap to closest point • 6 — Snap end points to end points only Only snap the start/end points of lines to other start/end points of lines • 7 — Snap to anchor nodes (single layer only)
Snapped geometry	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output (snapped) layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Snapped geometry	`OUTPUT`	[same as input]	Output (snapped) vector layer

Python code

Algorithm ID: `native:snapgeometries`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Snap points to grid

Modifies the coordinates of geometries in a vector layer, so that all points or vertices are snapped to the closest point of a grid.

If the snapped geometry cannot be calculated (or is totally collapsed) the feature's geometry will be cleared.

Snapping can be performed on the X, Y, Z or M axis. A grid spacing of 0 for any axis will disable snapping for that axis.

Allows *features in-place modification* of point, line, and polygon features

Warning

Snapping to grid may generate an invalid geometry in some corner cases.

See also

Snap geometries to layer

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
X Grid Spacing	`HSPACING`	[numeric: double] Default: 1.0	Grid spacing on the X axis
Y Grid Spacing	`VSPACING`	[numeric: double] Default: 1.0	Grid spacing on the Y axis
Z Grid Spacing	`ZSPACING`	[numeric: double] Default: 0.0	Grid spacing on the Z axis

continues on next page

Table 24.247 – continued from previous page

Label	Name	Type	Description
M Grid Spacing	`MSPACING`	[numeric: double] Default: 0.0	Grid spacing on the M axis
Snapped	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output (snapped) layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Snapped	`OUTPUT`	[same as input]	Output (snapped) vector layer

Python code

Algorithm ID: `native:snappointstogrid`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Split lines by maximum length

Takes a line (or curve) layer and splits each feature into multiple parts, where each part is of a specified maximum length. Z and M values at the start and end of the new line substrings are linearly interpolated from existing values.

Allows *features in-place modification* of line features

Warning

This algorithm drops existing primary keys or FID values and regenerates them in output layers.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line]	The input line vector layer
Maximum line length	`LENGTH`	[numeric: double]  Default: 10.0	The maximum length of a line in the output.
Split	`OUTPUT`	[vector: line] Default: `[Create temporary layer]`	Specify the output line vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Split	`OUTPUT`	[vector: line]	The new line vector layer. The length of the feature geometries is less than or equal to the length specified in the LENGTH parameter. An additional field containing the `order` will be added.

Python code

Algorithm ID: `native:splitlinesbylength`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Subdivide

Subdivides the geometry. The returned geometry will be a collection containing subdivided parts from the original geometry, where no part has more than the specified maximum number of nodes.

This is useful for dividing a complex geometry into less complex parts, easier to spatially index and faster to perform spatial operations. Curved geometries will be segmentized before subdivision.

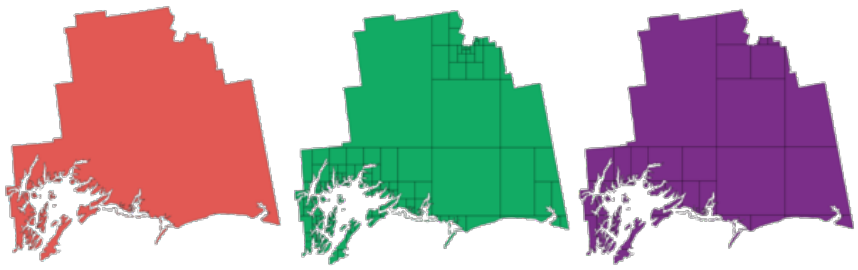


Fig. 24.125: Left the input layer, middle maximum nodes value is 100 and right maximum value is 200

Allows *features in-place modification* of point, line, and polygon features

Note

Subdividing a geometry can generate geometry parts that may not be valid and may contain self-intersections.

See also

Explode lines, *Line substring*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	The input vector layer
Maximum nodes in parts	`MAX_NODES`	[numeric: integer] Default: 256	Maximum number of vertices each new geometry part is allowed to have. Fewer *subparts* for higher values.
Subdivided	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output (subdivided) vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Subdivided	`OUTPUT`	[same as input]	Output vector layer

Python code

Algorithm ID: `native:subdivide`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Swap X and Y coordinates

Switches the X and Y coordinate values in input geometries.

It can be used to repair geometries which have accidentally had their latitude and longitude values reversed.

Allows *features in-place modification* of point, line, and polygon features

See also

Translate, *Rotate*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	The input vector layer
Swapped	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Swapped	`OUTPUT`	[same as input]	Output (swapped) vector layer

Python code

Algorithm ID: `native:swapxy`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Tapered buffers

Creates tapered buffer along line geometries, using a specified start and end buffer diameter.

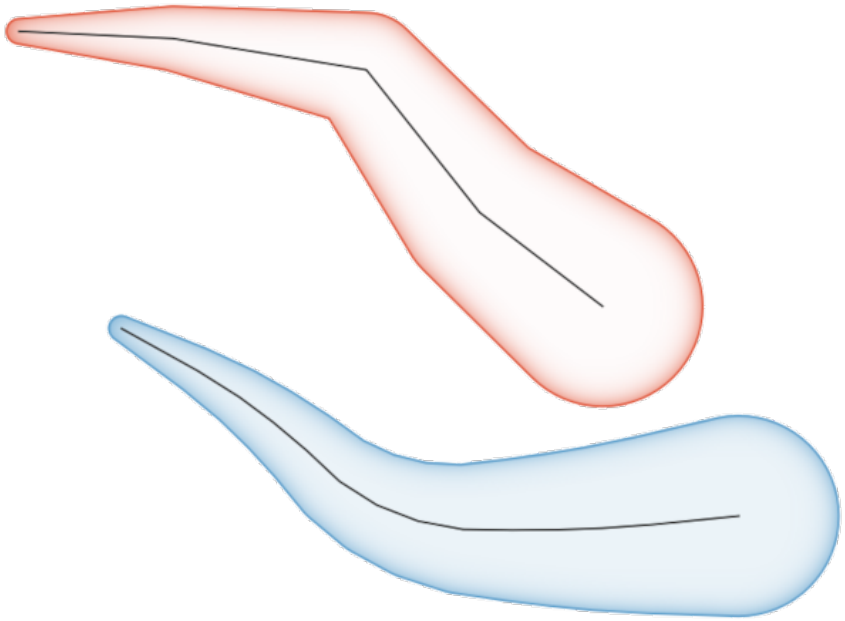


Fig. 24.126: Tapered buffer example

See also

Variable width buffer (by M value), *Buffer*, *Create wedge buffers*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line]	Input line vector layer
Start width	`START_WIDTH`	[numeric: double] Default: 0.0	Represents the radius of the buffer applied at the start point of the line feature

continues on next page

Table 24.248 – continued from previous page

Label	Name	Type	Description
End width	`END_WIDTH`	[numeric: double] Default: 0.0	Represents the radius of the buffer applied at the end point of the line feature.
Segments	`SEGMENTS`	[numeric: integer] Default: 16	Controls the number of line segments to use to approximate a quarter circle when creating rounded offsets.
Buffered	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output (buffer) layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Buffered	`OUTPUT`	[vector: polygon]	Output (buffer) polygon layer

Python code

Algorithm ID: `native:taperedbuffer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Tessellate

Tessellates a polygon geometry layer, dividing the geometries into triangular components.

The output layer consists of multipolygon geometries for each input feature, with each multipolygon consisting of multiple triangle component polygons.

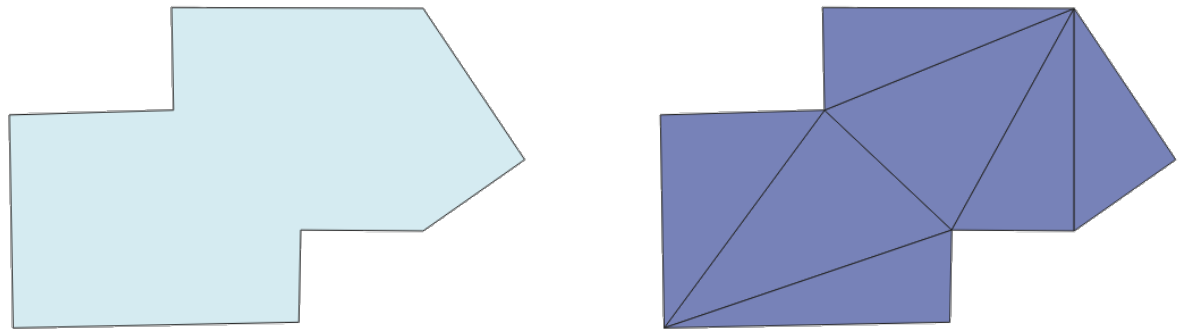


Fig. 24.127: Tessellated polygon (right)

Allows *features in-place modification* of polygon features

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: polygon]	Input polygon vector layer
Tesselated	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Tesselated	`OUTPUT`	[vector: polygon]	Output multipolygonZ layer

Python code

Algorithm ID: `3d:tessellate`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Transect

Creates transects on vertices for (multi)linestring.

A transect is a line oriented from an angle (by default perpendicular) to the input polylines (at vertices).

Field(s) from feature(s) are returned in the transect with these new fields:

- TR_FID: ID of the original feature
- TR_ID: ID of the transect. Each transect have an unique ID
- TR_SEGMENT: ID of the segment of the linestring
- TR_ANGLE: Angle in degrees from the original line at the vertex
- TR_LENGTH: Total length of the transect returned
- TR_ORIENT: Side of the transect (only on the left or right of the line, or both side)

Warning

This algorithm drops existing primary keys or FID values and regenerates them in output layers.

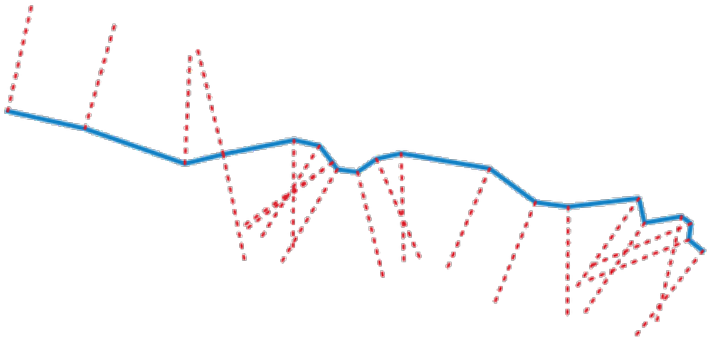


Fig. 24.128: Dashed red lines represent the transect of the input line layer

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line]	Input line vector layer
Length of the transect	`LENGTH`	[numeric: double]  Default: 5.0	Length in map unit of the transect
Angle in degrees from the original line at the vertices	`ANGLE`	[numeric: double]  Default: 90.0	Change the angle of the transect
Side to create the transect	`SIDE`	[enumeration]	Choose the side of the transect. Available options are: • 0 — Left • 1 — Right • 2 — Both
Transect	`OUTPUT`	[vector: line] Default: `[Create temporary layer]`	Specify the output line layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Transect	`OUTPUT`	[vector: line]	Output line layer

Python code

Algorithm ID: `native:transect`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Translate

Moves the geometries within a layer, by offsetting with a predefined X and Y displacement.

Z and M values present in the geometry can also be translated.

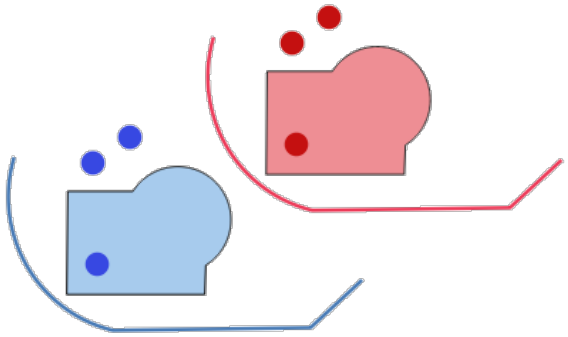


Fig. 24.129: Features in red represent the translated geometry of the input (blue) layer

Allows *features in-place modification* of point, line, and polygon features

See also

Array of translated features, *Offset lines*, *Rotate*, *Swap X and Y coordinates*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Offset distance (x-axis)	`DELTA_X`	[numeric: double] Default: 0.0	Displacement to apply on the X axis
Offset distance (y-axis)	`DELTA_Y`	[numeric: double] Default: 0.0	Displacement to apply on the Y axis
Offset distance (z-axis)	`DELTA_Z`	[numeric: double] Default: 0.0	Displacement to apply on the Z axis
Offset distance (m values)	`DELTA_M`	[numeric: double] Default: 0.0	Displacement to apply on the M axis
Translated	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Translated	`OUTPUT`	[same as input]	Output vector layer

Python code

Algorithm ID: `native:translategeometry`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Variable width buffer (by M value)

Creates variable width buffers along lines, using the M value of the line geometries as the diameter of the buffer at each vertex.

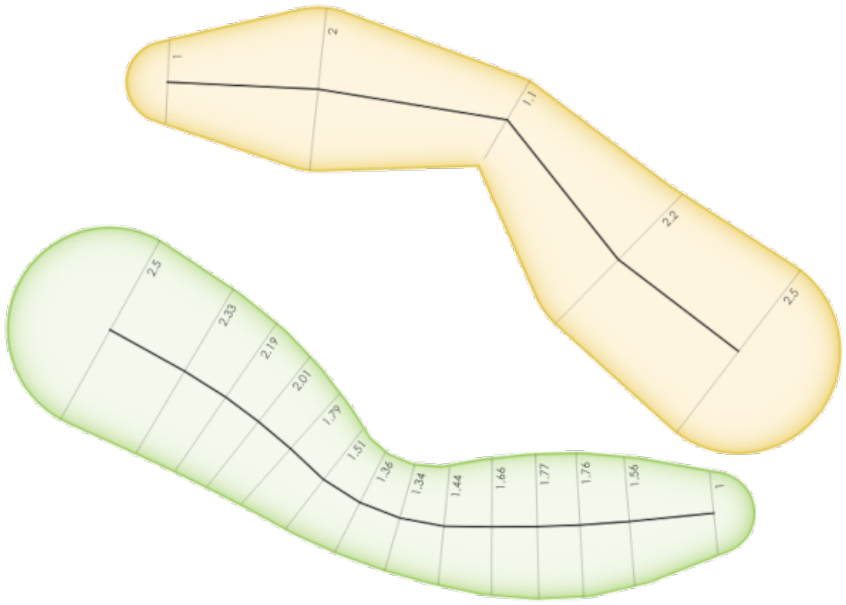


Fig. 24.130: Variable buffer example

See also

Tapered buffers, *Buffer*, *Set M value*, *Variable distance buffer*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line]	Input line vector layer
Segments	`SEGMENTS`	[numeric: integer]  Default: 16	Number of the buffer segments per quarter circle. It can be a unique value (same value for all the features), or it can be taken from features data (the value can depend on feature attributes).
Buffered	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output (buffer) layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Buffered	`OUTPUT`	[vector: polygon]	Variable buffer polygon layer

Python code

Algorithm ID: `native:bufferbym`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Voronoi polygons

Takes a point layer and generates a polygon layer containing the Voronoi polygons (known also as Thiessen polygons) corresponding to those input points.

Any location within a Voronoi polygon is closer to the associated point than to any other point.

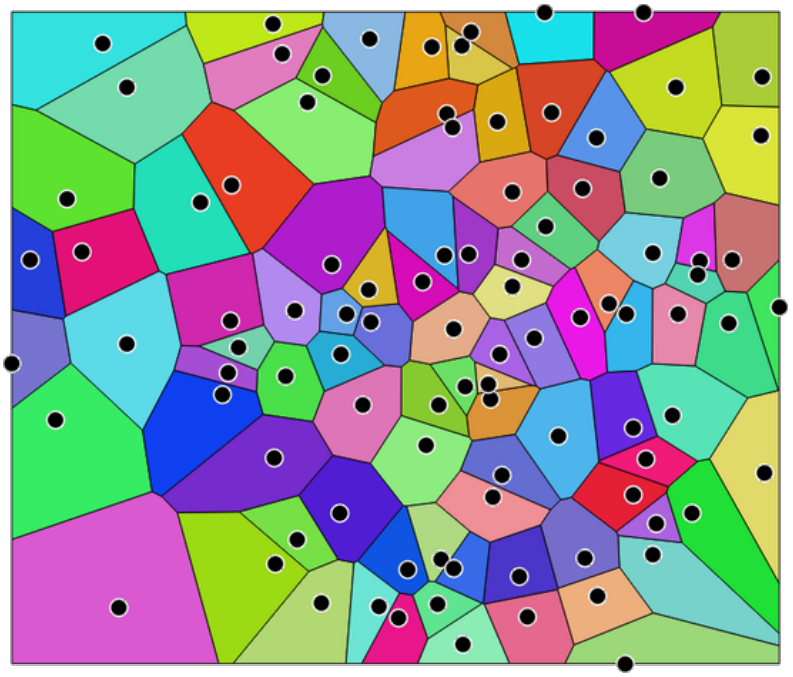


Fig. 24.131: Voronoi polygons

Default menu: *Vector* ► *Geometry Tools*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: point]	Input point vector layer
Buffer region (% of extent)	`BUFFER`	[numeric: double] Default: 0.0	The extent of the output layer will be this much bigger than the extent of the input layer
Tolerance Optional	`TOLERANCE`	[numeric: double] Default: 0.0	Specifies an optional snapping tolerance which can be used to improve the robustness of the voronoi.
Copy attributes from input features	`COPY_ATTRIBUTE`	[boolean] Default: True	Specifies whether fields storing involved point features ID should be added to the output. If False, an `id` field is created to identify the polygons.
Voronoi polygons	`OUTPUT`	[vector: polygon] Default: `[Create temporary layer]`	Specify the output layer (with the Voronoi polygons). *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Voronoi polygons	`OUTPUT`	[vector: polygon]	Voronoi polygons of the input point vector layer

Python code

Algorithm ID: `native:voronoipolygons`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.27 Vector overlay

Clip

Clips a vector layer using the features of an additional polygon layer.

Only the parts of the features in the input layer that fall within the polygons of the overlay layer will be added to the resulting layer.

Warning

Geometry modification only

This operation modifies only the features geometry. The attribute values of the features are **not modified**, although properties such as area or length of the features will be modified by the overlay operation. If such properties are stored as attributes, those attributes will have to be manually updated.

This algorithm uses spatial indexes on the providers, prepared geometries and apply a clipping operation if the geometry isn't wholly contained by the mask geometry.

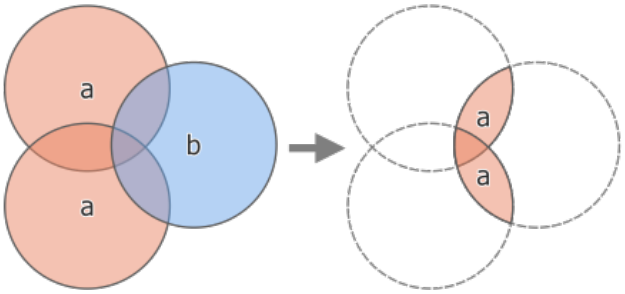


Fig. 24.132: Clipping operation between a two-feature input layer 'a' and a single feature overlay layer 'b' (left) - resulting in a new layer with the modified 'a' features (right)

Allows *features in-place modification* of point, line, and polygon features

Default menu: *Vector* ► *Geoprocessing Tools*

See also

Intersection, *Difference*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Layer containing the features to be clipped
Overlay layer	`OVERLAY`	[vector: polygon]	Layer containing the clipping features
Clipped	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the layer to contain the features from the input layer that are inside the overlay (clipping) layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Clipped	`OUTPUT`	[same as input]	Layer containing features from the input layer split by the overlay layer.

Python code

Algorithm ID: `qgis:clip`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Difference

Extracts features from the input layer that don't fall within the boundaries of the overlay layer.

Input layer features that partially overlap the overlay layer feature(s) are split along the boundary of those feature(s) and only the portions outside the overlay layer features are retained.

Warning

Geometry modification only

This operation modifies only the features geometry. The attribute values of the features are **not modified**, although properties such as area or length of the features will be modified by the overlay operation. If such properties are stored as attributes, those attributes will have to be manually updated.

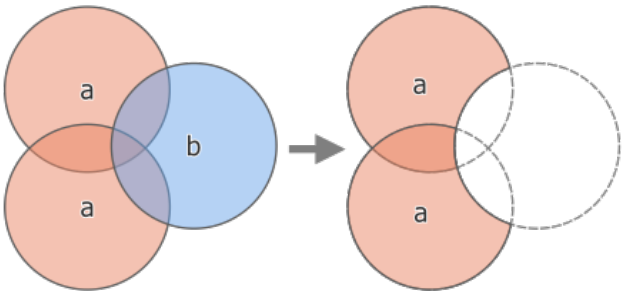


Fig. 24.133: Difference operation between a two-features input layer 'a' and a single feature overlay layer 'b' (left) - resulting in a new layer with the modified 'a' features (right)

Allows *features in-place modification* of point, line, and polygon features

Default menu: *Vector* ► *Geoprocessing Tools*

See also

Difference (multiple), *Symmetrical difference*, *Clip*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Layer to extract (parts of) features from.
Overlay layer	`OVERLAY`	[vector: geometry]	Layer containing the geometries that will be subtracted from the input layer geometries. It is expected to have at least as many dimensions (point: 0D, line: 1D, polygon: 2D, volume: 3D) as the input layer geometries.
Difference	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the layer to contain the (parts of) features from the input layer that are not inside the overlay layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Grid size Optional	`GRID_SIZE`	[numeric: double] Default: Not set	If provided, the input geometries are snapped to a grid of the given size, and the result vertices are computed on that same grid. Requires GEOS 3.9.0 or higher.

Outputs

Label	Name	Type	Description
Difference	`OUTPUT`	[same as input]	Layer containing (parts of) features from the input layer not overlapping the overlay layer.

Python code

Algorithm ID: `qgis:difference`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Difference (multiple)

Extracts features from the input layer that fall completely outside or only partially overlap the features from any of the overlay layer(s).

For each overlay layer the difference is calculated between the result of all previous difference operations and this overlay layer. Input layer features that partially overlap feature(s) in the overlay layer are split along those features' boundary and only the portions outside the overlay layer features are retained.

Warning

Geometry modification only

This operation modifies only the features geometry. The attribute values of the features are **not modified**, although properties such as area or length of the features will be modified by the overlay operation. If such properties are stored as attributes, those attributes will have to be manually updated.

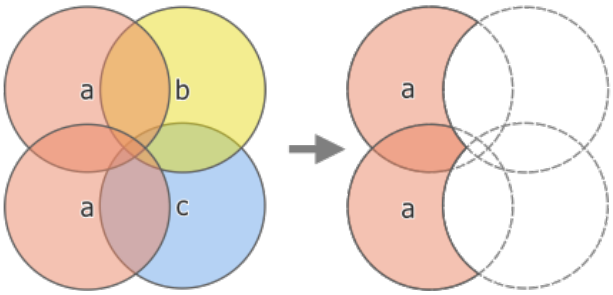


Fig. 24.134: Difference operation between a two-feature input layer 'a' and single feature overlay layers 'b' and 'c' (left) - resulting in a new layer with the modified 'a' features (right)

See also

Difference, *Symmetrical difference*, *Clip*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Layer to extract (parts of) features from.
Overlay layers	`OVERLAYS`	[vector: geometry] [list]	List of layers containing the geometries that will be subtracted from the input layer geometries. They are expected to have at least as many dimensions (point: 0D, line: 1D, polygon: 2D, volume: 3D) as the input layer geometries.
Difference	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the layer to contain the (parts of) features from the input layer that do not overlap features of the overlay layers. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Difference	`OUTPUT`	[same as input]	Layer containing (parts of) features from the input layer not overlapping features from the overlay layers.

Python code

Algorithm ID: `qgis:multidifference`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Extract/clip by extent

Creates a new vector layer that only contains features which fall within a specified extent. Any features which intersect the extent will be included by default.

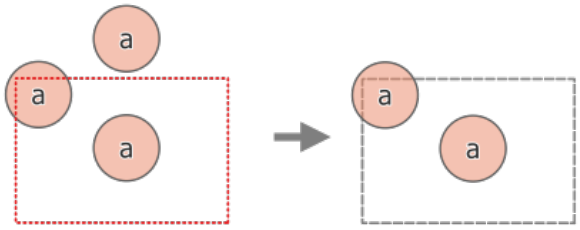


Fig. 24.135: Extract operation between a three-feature input layer 'a' and a dashed extent (left) - resulting features with dashed extent for reference (right)

Optionally, feature geometries can also be clipped to the extent.

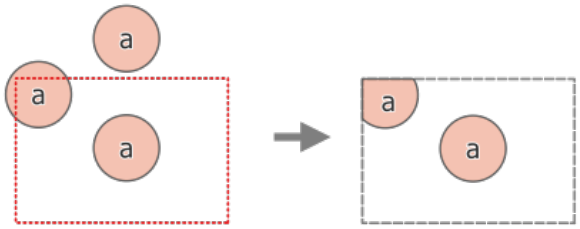


Fig. 24.136: Extract operation between a three-feature input layer 'a' and a dashed extent (left) - resulting features with dashed extent for reference (right)

See also

Clip

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Layer to extract (parts of) features from.
Extent (xmin, xmax, ymin, ymax)	`EXTENT`	[extent]	Extent for clipping. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Clip features to extent	`CLIP`	[boolean] Default: False	If checked, the geometries will be clipped to the extent chosen instead of taking the whole geometry as output. Moreover, output geometries will be automatically converted to multi geometries to ensure uniform output types.
Extracted	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the layer to contain the features from the input layer that are inside the clip extent. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Extracted	`OUTPUT`	[same as input]	Layer containing the clipped features.

Python code

Algorithm ID: `native:extractbyextent`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Intersection

Extracts the portions of features from the input layer that overlap features in the overlay layer.

Features in the intersection layer are assigned the attributes of the overlapping features from both the input and overlay layers.

Warning

Geometry modification only

This operation modifies only the features geometry. The attribute values of the features are **not modified**, although properties such as area or length of the features will be modified by the overlay operation. If such properties are stored as attributes, those attributes will have to be manually updated.

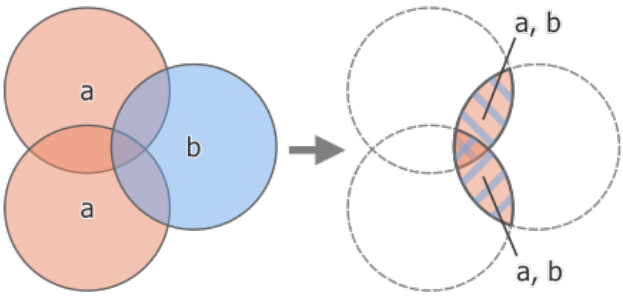


Fig. 24.137: Intersection operation between a two-feature input layer 'a' and a single feature overlay layer 'b' (left) - overlapping areas become a new two-feature layer with both layers' attributes (right)

Default menu: *Vector* ► *Geoprocessing Tools*

Warning

This algorithm drops existing primary keys or FID values and regenerates them in output layers.

See also

Intersection (multiple), *Clip*, *Difference*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Layer to extract (parts of) features from.
Overlay layer	`OVERLAY`	[vector: geometry]	Layer containing the features to check for overlap. Its features' geometry is expected to have at least as many dimensions (point: 0D, line: 1D, polygon: 2D, volume: 3D) as the input layer's.
Input fields to keep (leave empty to keep all fields) Optional	`INPUT_FIELDS`	[tablefield: any] [list] Default: Not set	Field(s) of the input layer to keep in the output. If no fields are chosen all fields are taken.
Overlay fields to keep (leave empty to keep all fields) Optional	`OVERLAY_FIELDS`	[tablefield: any] [list] Default: Not set	Field(s) of the overlay layer to keep in the output. If no fields are chosen all fields are taken. Duplicate field names will be appended a count suffix to avoid collision.
Intersection	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the layer to contain (the parts of) the features from the input layer that overlap one or more features from the overlay layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Overlay fields prefix Optional	`OVERLAY_FIELDS_PRE`	[string]	Add a prefix to identify fields of the overlay layer. Duplicate field names will be appended a count suffix to avoid collision.
Grid size Optional	`GRID_SIZE`	[numeric: double] Default: Not set	If provided, the input geometries are snapped to a grid of the given size, and the result vertices are computed on that same grid. Requires GEOS 3.9.0 or higher.

Outputs

Label	Name	Type	Description
Intersection	`OUTPUT`	[same as input]	Layer containing (parts of) features from the input layer that overlap the overlay layer.

Python code

Algorithm ID: `qgis:intersection`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Intersection (multiple)

Extracts the overlapping portions of features in the input and all overlay layers.

Features in the output layer are assigned the attributes of the overlapping features from both the input and overlay layers.

Warning

Geometry modification only

This operation modifies only the features geometry. The attribute values of the features are **not modified**, although properties such as area or length of the features will be modified by the overlay operation. If such properties are stored as attributes, those attributes will have to be manually updated.

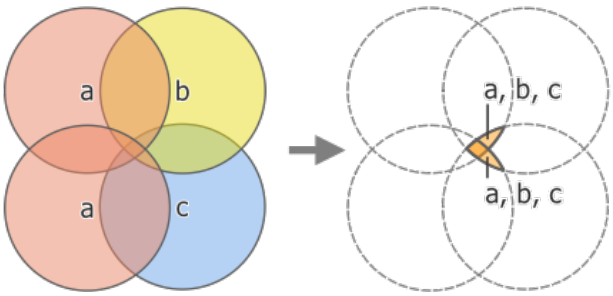


Fig. 24.138: Intersection operation between a two-feature input layer 'a' and single feature overlay layers 'b' and 'c' (left) - overlapping areas become a new two-feature layer with all layers' attributes (right)

Warning

This algorithm drops existing primary keys or FID values and regenerate them in output layers.

See also

Intersection, *Clip*, *Difference*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Layer to extract (parts of) features from.
Overlay layers	`OVERLAYS`	[vector: geometry] [list]	Layers containing the features to check for overlap. The features' geometry is expected to have at least as many dimensions (point: 0D, line: 1D, polygon: 2D, volume: 3D) as the input layer's.

Advanced parameters

Label	Name	Type	Description
Overlay fields prefix Optional	`OVERLAY_FIELDS_PRE`	[string]	Add a prefix to identify fields of the overlay layers. Duplicate field names will be appended a count suffix to avoid collision.

Outputs

Label	Name	Type	Description
Intersection	`OUTPUT`	[same as input]	Layer containing (parts of) features from the input layer that overlap all the overlay layers.

Python code

Algorithm ID: `qgis:multiintersection`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Line intersections

Creates point features where the lines from the two layers intersect.

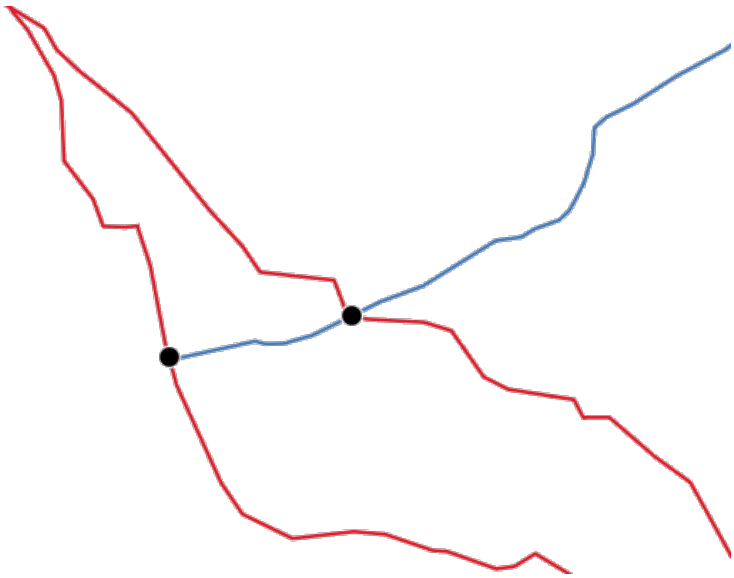


Fig. 24.139: Points of intersection

Default menu: *Vector* ► *Analysis Tools*

> **Warning**
>
> This algorithm drops existing primary keys or FID values and regenerate them in output layers.

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line]	Input line layer.
Intersect layer	`INTERSECT`	[vector: line]	Layer to use to find line intersections.
Input fields to keep (leave empty to keep all fields) Optional	`INPUT_FIELDS`	[tablefield: any] [list] Default: Not set	Field(s) of the input layer to keep in the output. If no fields are chosen all fields are taken.
Intersect fields to keep (leave empty to keep all fields) Optional	`INTERSECT_FIELDS`	[tablefield: any] [list] Default: Not set	Field(s) of the intersect layer to keep in the output. If no fields are chosen all fields are taken. Duplicate field names will be appended a count suffix to avoid collision.

continues on next page

Table 24.259 – continued from previous page

Label	Name	Type	Description
Intersection	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specify the layer to contain the intersection points of the lines from the input and overlay layers. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Intersect fields prefix Optional	`INTERSECT_FIELDS_PR`	[string]	Add a prefix to identify fields of the intersect layer.

Outputs

Label	Name	Type	Description
Intersections	`OUTPUT`	[vector: point]	Point vector layer of the lines intersections, with both layers' attributes.

Python code

Algorithm ID: `qgis:lineintersections`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Split with lines

Splits the lines or polygons in one layer using the lines or polygon rings in another layer to define the breaking points. Intersection between geometries in both layers are considered as split points.

Output will contain multi geometries for split features.

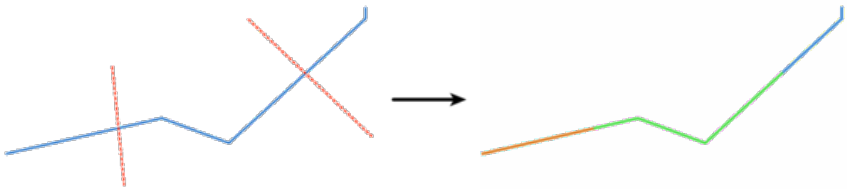


Fig. 24.140: Split lines

Allows *features in-place modification* of line and polygon features

Warning

This algorithm drops existing primary keys or FID values and regenerate them in output layers.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line, polygon]	Layer containing the lines or polygons to split.
Split layer	`LINES`	[vector: line, polygon]	Layer whose lines or rings are used to define the breaking points.
Split	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the layer to contain the splitted (in case they are intersected by a line in the split layer) line/polygon features from the input layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Split	`OUTPUT`	[same as input]	Output vector layer with split lines or polygons from input layer.

Python code

Algorithm ID: `qgis:splitwithlines`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Symmetrical difference

Creates a layer containing features from both the input and overlay layers but with the overlapping areas between the two layers removed.

The attribute table of the symmetrical difference layer contains attributes and fields from both the input and overlay layers.

Warning

Geometry modification only

This operation modifies only the features geometry. The attribute values of the features are **not modified**, although properties such as area or length of the features will be modified by the overlay operation. If such properties are stored as attributes, those attributes will have to be manually updated.

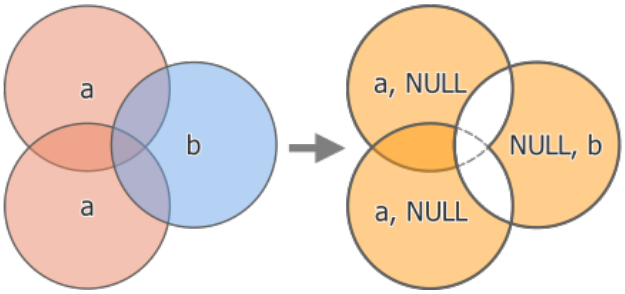


Fig. 24.141: Symmetrical difference operation between a two-features input layer 'a' and a single feature overlay layer 'b' (left) - resulting three-feature layer with both layers' attributes (right)

Default menu: *Vector ► Geoprocessing Tools*

Warning

This algorithm drops existing primary keys or FID values and regenerate them in output layers.

See also

Difference, *Clip*, *Intersection*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	First layer to extract (parts of) features from.
Overlay layer	`OVERLAY`	[vector: geometry]	Second layer to extract (parts of) features from. Ideally the geometry type should be the same as input layer.
Symmetrical difference	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the layer to contain (the parts of) the features from the input and overlay layers that do not overlap features from the other layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Overlay fields prefix Optional	`OVERLAY_FIELDS_PRE`	[string]	Add a prefix to identify fields of the overlay layer. Duplicate field names will be appended a count suffix to avoid collision.
Grid size Optional	`GRID_SIZE`	[numeric: double] Default: Not set	If provided, the input geometries are snapped to a grid of the given size, and the result vertices are computed on that same grid. Requires GEOS 3.9.0 or higher.

Outputs

Label	Name	Type	Description
Symmetrical difference	`OUTPUT`	[same as input]	Layer containing (parts of) features from each layer not overlapping the other layer, with both layers' attributes.

Python code

Algorithm ID: `qgis:symmetricaldifference`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Union

Checks overlaps between features within the input layer and creates separate features for overlapping and non-overlapping parts. The area of overlap will create as many identical overlapping features as there are features that participate in that overlap.

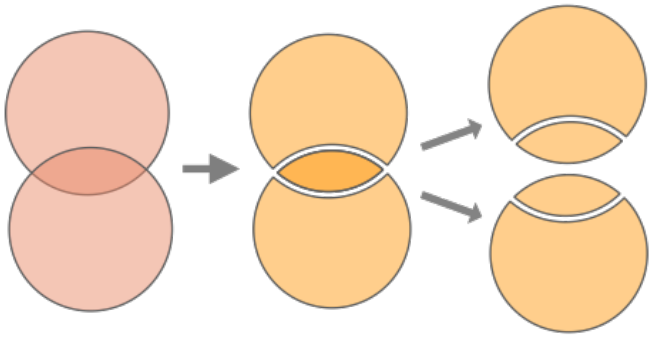


Fig. 24.142: Union operation with a single input layer with two overlapping features (left) - resulting in four features (middle) - features moved for clarity (right)

An overlay layer can also be used, in which case features from each layer are split at their overlap with features from the other one, creating a layer containing all the portions from both input and overlay layers. Features on the same layer will not split each other. The attribute table of the union layer is filled with attribute values from the respective original layer for non-overlapping features, and attribute values from both layers for overlapping features.

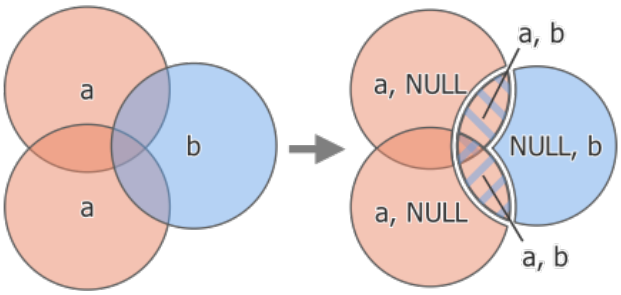


Fig. 24.143: Union operation between a two-feature input layer 'a' and a single feature overlay layer 'b' (left) - resulting five-feature layer with attributes from both layers (right)

Note

With an overlay layer, features on the same layer will not split each other. If you want to split overlaps on the same layer as well as other layers, first run the algorithm with multiple layers then run the algorithm again with only the previous output.

Default menu: *Vector ► Geoprocessing Tools*

Warning

This algorithm drops existing primary keys or FID values and regenerate them in output layers.

See also

Union (multiple), *Clip*, *Difference*, *Intersection*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer to split at any intersections.
Overlay layer Optional	`OVERLAY`	[vector: geometry]	Layer that will be combined to the first one. Ideally the geometry type should be the same as input layer.
Union	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the layer to contain the (split and duplicated) features from the input layer and the overlay layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Overlay fields prefix Optional	`OVERLAY_FIELDS_PRE`	[string]	Add a prefix to identify fields of the overlay layer. Duplicate field names will be appended a count suffix to avoid collision.
Grid size Optional	`GRID_SIZE`	[numeric: double] Default: Not set	If provided, the input geometries are snapped to a grid of the given size, and the result vertices are computed on that same grid. Requires GEOS 3.9.0 or higher.

Outputs

Label	Name	Type	Description
Union	`OUTPUT`	[same as input]	Layer containing all the overlapping and non-overlapping parts from the processed layer(s).

Python code

Algorithm ID: `qgis:union`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Union (multiple)

Checks overlaps between features within the input layer and creates separate features for overlapping and non-overlapping parts. The area of overlap will create as many identical overlapping features as there are features that participate in that overlap.

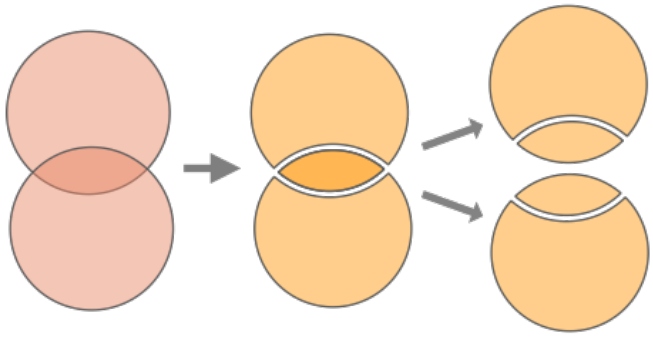


Fig. 24.144: Union operation with a single input layer with two overlapping features (left) - resulting in four features (middle) - features moved for clarity (right)

Multiple overlay layers can also be used, in which case features from each layer are split at their overlap with features from all other layers, creating a layer containing all the portions from both input and overlay layers. Features on the same layer will not split each other. The attribute table of the Union layer is filled with attribute values from the respective original layer for non-overlapping features, and attribute values from overlay layers for overlapping features.

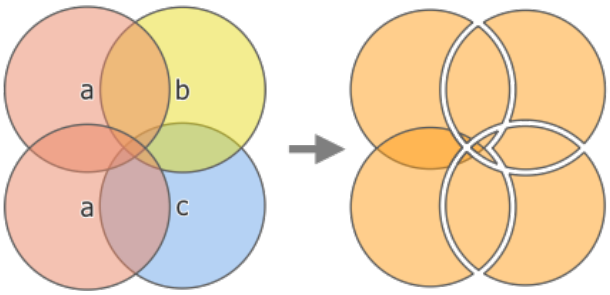


Fig. 24.145: Union operation between a two-feature input layer 'a' and single feature overlay layers 'b' and 'c' (left) - resulting eleven-feature layer with attributes from all layers (right)

Note

With an overlay layer, features on the same layer will not split each other. If you want to split overlaps on the same layer as well as other layers, first run the algorithm with multiple layers then run the algorithm again with only the previous output.

Warning

This algorithm drops existing primary keys or FID values and regenerate them in output layers.

See also

Union, *Clip*, *Difference*, *Intersection*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer to split at any intersections.
Overlay layers Optional	`OVERLAYS`	[vector: geometry] [list]	Layers that will be combined to the first one. Ideally the geometry type should be the same as input layer.
Union	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the layer to contain the (split and duplicated) features from the input layer and the overlay layers. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Advanced parameters

Label	Name	Type	Description
Overlay fields prefix Optional	`OVERLAY_FIELDS_PRE`	[string]	Add a prefix to identify fields of the overlay layers. Duplicate field names will be appended a count suffix to avoid collision.

Outputs

Label	Name	Type	Description
Union	`OUTPUT`	[same as input]	Layer containing all the overlapping and non-overlapping parts from the processed layer(s), with all layers' attributes.

Python code

Algorithm ID: `qgis:multiunion`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.28 Vector selection

Extract by attribute

Creates two vector layers from an input layer: one will contain only matching features while the second will contain all the non-matching features.

The criteria for adding features to the resulting layer is based on the values of an attribute from the input layer.

See also

Select by attribute

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	Layer to extract features from.
Selection attribute	`FIELD`	[tablefield: any]	Filtering field of the layer
Operator	`OPERATOR`	[enumeration] Default: 0	Many different operators are available: • 0 — = • 1 — ≠ • 2 — >>• 3 — >= • 4 — < • 5 — <= • 6 — begins with • 7 — contains • 8 — is null • 9 — is not null • 10 — does not contain
Value Optional	`VALUE`	[string]	Value to be evaluated
Extracted (attribute)	`OUTPUT`	[same as input] Default: `[Create Temporary Layer]`	Specify the output vector layer for matching features. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Extracted (non-matching)	`FAIL_OUTPUT`	[same as input] Default: `[Skip output]`	Specify the output vector layer for non-matching features. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Extracted (attribute)	`OUTPUT`	[same as input]	Vector layer with matching features from the input layer
Extracted (non-matching)	`FAIL_OUTPUT`	[same as input]	Vector layer with non-matching features from the input layer

Python code

Algorithm ID: `native:extractbyattribute`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Extract by expression

Creates two vector layers from an input layer: one will contain only matching features while the second will contain all the non-matching features.

The criteria for adding features to the resulting layer is based on a QGIS expression. For more information about expressions see the *Expressions*.

See also

Select by expression

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	Input vector layer
Expression	`EXPRESSION`	[expression]	Expression to filter the vector layer
Matching features	`OUTPUT`	[same as input] Default: `[Create Temporary Layer]`	Specify the output vector layer for matching features. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Non-matching	`FAIL_OUTPUT`	[same as input] Default: `[Skip output]`	Specify the output vector layer for non-matching features. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Matching features	`OUTPUT`	[same as input]	Vector layer with matching features from the input layer
Non-matching	`FAIL_OUTPUT`	[same as input]	Vector layer with non-matching features from the input layer

Python code

Algorithm ID: `native:extractbyexpression`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Extract by location

Creates a new vector layer that only contains matching features from an input layer.

The criteria for adding features to the resulting layer is based on the spatial relationship between each feature and the features in an additional layer.

See also

Select by location, *Extract within distance*

Exploring spatial relations

Geometric predicates are boolean functions used to determine the spatial relation a feature has with another by comparing whether and how their geometries share a portion of space.

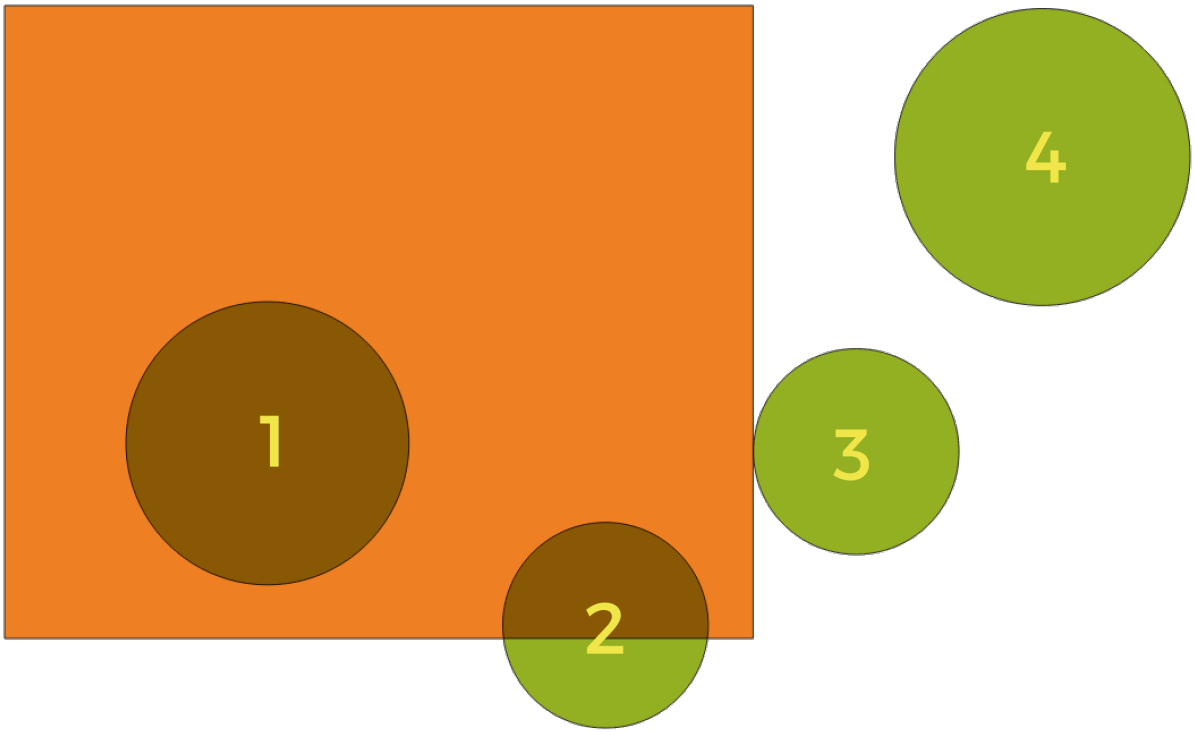


Fig. 24.146: Looking for spatial relations between layers

Using the figure above, we are looking for the green circles by spatially comparing them to the orange rectangle feature. Available geometric predicates are:

Intersect
Tests whether a geometry intersects another. Returns 1 (true) if the geometries spatially intersect (share any portion of space - overlap or touch) and 0 if they don't. In the picture above, this will return circles 1, 2 and 3.

Contain
Returns 1 (true) if and only if no points of b lie in the exterior of a, and at least one point of the interior of b lies in the interior of a. In the picture, no circle is returned, but the rectangle would be if you would look for it the other way around, as it contains circle 1 completely. This is the opposite of *are within*.

Disjoint
Returns 1 (true) if the geometries do not share any portion of space (no overlap, not touching). Only circle 4 is returned.

Equal
Returns 1 (true) if and only if geometries are exactly the same. No circles will be returned.

Touch
Tests whether a geometry touches another. Returns 1 (true) if the geometries have at least one point in common, but their interiors do not intersect. Only circle 3 is returned.

Overlap
Tests whether a geometry overlaps another. Returns 1 (true) if the geometries share space, are of the same dimension, but are not completely contained by each other. Only circle 2 is returned.

Are within
Tests whether a geometry is within another. Returns 1 (true) if geometry a is completely inside geometry b. Only circle 1 is returned.

Cross
Returns 1 (true) if the supplied geometries have some, but not all, interior points in common and the actual crossing is of a lower dimension than the highest supplied geometry. For example, a line crossing a polygon

will cross as a line (true). Two lines crossing will cross as a point (true). Two polygons cross as a polygon (false). In the picture, no circles will be returned.

Parameters

Label	Name	Type	Description
Extract features from	`INPUT`	[vector: geometry]	Input vector layer
Where the features (geometric predicate)	`PREDICATE`	[enumeration] [list] Default: [0]	Type of spatial relation the input feature should have with an intersect feature so that it could be selected. One or more of: • 0 — intersect • 1 — contain • 2 — disjoint • 3 — equal • 4 — touch • 5 — overlap • 6 — are within • 7 — cross If more than one condition is chosen, at least one of them (OR operation) has to be met for a feature to be extracted.
By comparing to the features from	`INTERSECT`	[vector: geometry]	Intersection vector layer
Extracted (location)	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer for the features that have the chosen spatial relationship(s) with one or more features in the comparison layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Extracted (location)	`OUTPUT`	[same as input]	Vector layer with features from the input layer that have the chosen spatial relationship(s) with one or more features in the comparison layer.

Python code

Algorithm ID: `native:extractbylocation`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Extract within distance

Creates a new vector layer that only contains matching features from an input layer. Features are copied wherever they are within the specified maximum distance from the features in an additional reference layer.

See also

Select within distance, *Extract by location*

Parameters

Label	Name	Type	Description
Extract features from	`INPUT`	[vector: geometry]	Input vector layer to copy features from
By comparing to the features from	`REFERENCE`	[vector: geometry]	Vector layer whose features closeness is used
Where the features are within	`DISTANCE`	[numeric: double] Default: 100.0	The maximum distance around reference features to select input features within
Modify current selection by	`METHOD`	[enumeration] Default: 0	How the selection of the algorithm should be managed. One of: • 0 — creating new selection • 1 — adding to current selection • 2 — selecting within current selection • 3 — removing from current selection
Extracted (location)	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer for the features that are within the set distance from reference features. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Extracted (location)	`OUTPUT`	[same as input]	Vector layer with features from the input layer matching the condition of distance from reference features

Python code

Algorithm ID: `native:extractwithindistance`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Filter by geometry type

Filters features by their geometry type. Incoming features will be directed to different outputs based on whether they have a point, line or polygon geometry.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	Layer to evaluate

Outputs

Label	Name	Type	Description
Point features Optional	`POINTS`	[vector: point]	Layer with points
Line features Optional	`LINES`	[vector: line]	Layer with lines
Polygon features Optional	`POLYGONS`	[vector: polygon]	Layer with polygons
Features with no geometry Optional	`NO_GEOMETRY`	[vector: table]	Geometry-less vector layer

Python code

Algorithm ID: `native:filterbygeometry`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Random extract

Takes a vector layer and generates a new one that contains only a subset of the features in the input layer.

The subset is defined randomly, based on feature IDs, using a percentage or count value to define the total number of features in the subset.

Warning

This algorithm drops existing primary keys or FID values and regenerate them in output layers.

See also

Random selection

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	Source vector layer to select the features from
Method	`METHOD`	[enumeration] Default: 0	Random selection methods. One of: • 0 — Number of selected features • 1 — Percentage of selected features
Number/percentage of selected features	`NUMBER`	[numeric: integer] Default: 10	Number or percentage of features to select
Extracted (random)	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer for the randomly selected features. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Extracted (random)	`OUTPUT`	[same as input]	Vector layer containing randomly selected features from the input layer

Python code

Algorithm ID: `native:randomextract`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Random extract within subsets

Takes a vector layer and generates a new one that contains only a subset of the features in the input layer.

The subset is defined randomly, based on feature IDs, using a percentage or count value to define the total number of features in the subset. The percentage/count value is not applied to the whole layer, but instead to each category. Categories are defined according to a given attribute.

See also

Random selection within subsets

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	Vector layer to select the features from
ID field	`FIELD`	[tablefield: any]	Category of the source vector layer to select the features from
Method	`METHOD`	[enumeration] Default: 0	Random selection method. One of: • 0 — Number of selected features • 1 — Percentage of selected features
Number/percentage of selected features	`NUMBER`	[numeric: integer] Default: 10	Number or percentage of features to select

continues on next page

Table 24.268 – continued from previous page

Label	Name	Type	Description
Extracted (random stratified)	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer for the randomly selected features. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Extracted (random stratified)	`OUTPUT`	[same as input]	Vector layer containing randomly selected features from the input layer

Python code

Algorithm ID: `qgis:randomextractwithinsubsets`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Random selection

Takes a vector layer and selects a subset of its features. No new layer is generated by this algorithm.

The subset is defined randomly, based on feature IDs, using a percentage or count value to define the total number of features in the subset.

Default menu: *Vector ► Research Tools*

See also

Random extract

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	Vector layer for the selection
Method	`METHOD`	[enumeration] Default: 0	Random selection method. One of: • 0 — Number of selected features • 1 — Percentage of selected features
Number/percentage of selected features	`NUMBER`	[numeric: integer] Default: 10	Number or percentage of features to select

Outputs

Label	Name	Type	Description
Input layer	`INPUT`	[same as input]	The input layer with features selected

Python code

Algorithm ID: `qgis:randomselection`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Random selection within subsets

Takes a vector layer and selects a subset of its features. No new layer is generated by this algorithm.

The subset is defined randomly, based on feature IDs, using a percentage or count value to define the total number of features in the subset.

The percentage/count value is not applied to the whole layer, but instead to each category.

Categories are defined according to a given attribute, which is also specified as an input parameter for the algorithm.

No new outputs are created.

Default menu: *Vector ► Research Tools*

See also

Random extract within subsets

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	Vector layer to select features in
ID field	`FIELD`	[tablefield: any]	Category of the input layer to select the features from
Method	`METHOD`	[enumeration] Default: 0	Random selection method. One of: • 0 — Number of selected features • 1 — Percentage of selected features
Number/percentage of selected features	`NUMBER`	[numeric: integer] Default: 10	Number or percentage of features to select

Outputs

Label	Name	Type	Description
Input layer	`INPUT`	[same as input]	The input layer with features selected

Python code

Algorithm ID: `qgis:randomselectionwithinsubsets`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Select by attribute

Creates a selection in a vector layer.

The criteria for selecting features is based on the values of an attribute from the input layer.

See also

Extract by attribute

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	Vector layer to select features in
Selection attribute	`FIELD`	[tablefield: any]	Filtering field of the layer
Operator	`OPERATOR`	[enumeration] Default: 0	Many different operators are available: • 0 — = • 1 — ≠ • 2 — >> • 3 — >= • 4 — < • 5 — <= • 6 — begins with • 7 — contains • 8 — is null • 9 — is not null • 10 — does not contain
Value Optional	`VALUE`	[string]	Value to be evaluated
Modify current selection by	`METHOD`	[enumeration] Default: 0	How the selection of the algorithm should be managed. One of: • 0 — creating new selection • 1 — adding to current selection • 2 — removing from current selection • 3 — selecting within current selection

Outputs

Label	Name	Type	Description
Input layer	`INPUT`	[same as input]	The input layer with features selected

Python code

Algorithm ID: `qgis:selectbyattribute`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Select by expression

Creates a selection in a vector layer.

The criteria for selecting features is based on a QGIS expression. For more information about expressions see the *Expressions*.

See also

Extract by expression

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	Input vector layer
Expression	`EXPRESSION`	[expression]	Expression to filter the input layer
Modify current selection by	`METHOD`	[enumeration] Default: 0	How the selection of the algorithm should be managed. One of: • 0 — creating new selection • 1 — adding to current selection • 2 — removing from current selection • 3 — selecting within current selection

Outputs

Label	Name	Type	Description
Input layer	`INPUT`	[same as input]	The input layer with features selected

Python code

Algorithm ID: `qgis:selectbyexpression`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Select by location

Creates a selection in a vector layer.

The criteria for selecting features is based on the spatial relationship between each feature and the features in an additional layer.

Default menu: *Vector ► Research Tools*

See also

Extract by location, *Select within distance*

Exploring spatial relations

Geometric predicates are boolean functions used to determine the spatial relation a feature has with another by comparing whether and how their geometries share a portion of space.

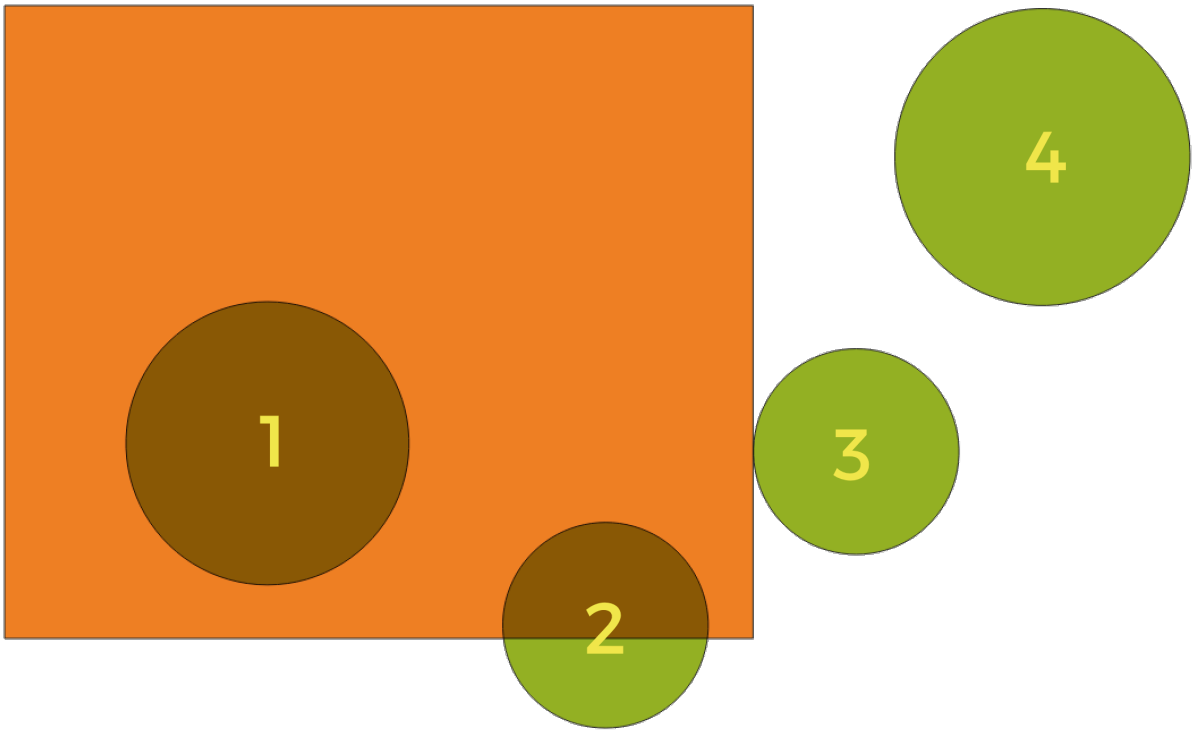


Fig. 24.147: Looking for spatial relations between layers

Using the figure above, we are looking for the green circles by spatially comparing them to the orange rectangle feature. Available geometric predicates are:

Intersect
: Tests whether a geometry intersects another. Returns 1 (true) if the geometries spatially intersect (share any portion of space - overlap or touch) and 0 if they don't. In the picture above, this will return circles 1, 2 and 3.

Contain
: Returns 1 (true) if and only if no points of b lie in the exterior of a, and at least one point of the interior of b lies in the interior of a. In the picture, no circle is returned, but the rectangle would be if you would look for it the other way around, as it contains circle 1 completely. This is the opposite of *are within*.

Disjoint
Returns 1 (true) if the geometries do not share any portion of space (no overlap, not touching). Only circle 4 is returned.

Equal
Returns 1 (true) if and only if geometries are exactly the same. No circles will be returned.

Touch
Tests whether a geometry touches another. Returns 1 (true) if the geometries have at least one point in common, but their interiors do not intersect. Only circle 3 is returned.

Overlap
Tests whether a geometry overlaps another. Returns 1 (true) if the geometries share space, are of the same dimension, but are not completely contained by each other. Only circle 2 is returned.

Are within
Tests whether a geometry is within another. Returns 1 (true) if geometry a is completely inside geometry b. Only circle 1 is returned.

Cross
Returns 1 (true) if the supplied geometries have some, but not all, interior points in common and the actual crossing is of a lower dimension than the highest supplied geometry. For example, a line crossing a polygon will cross as a line (true). Two lines crossing will cross as a point (true). Two polygons cross as a polygon (false). In the picture, no circles will be returned.

Parameters

Label	Name	Type	Description
Select features from	`INPUT`	[vector: geometry]	Input vector layer
Where the features (geometric predicate)	`PREDICATE`	[enumeration] [list] Default: [0]	Type of spatial relation the input feature should have with an intersect feature so that it could be selected. One or more of: • 0 — intersect • 1 — contain • 2 — disjoint • 3 — equal • 4 — touch • 5 — overlap • 6 — are within • 7 — cross If more than one condition is chosen, at least one of them (OR operation) has to be met for a feature to be extracted.
By comparing to the features from	`INTERSECT`	[vector: geometry]	Intersection vector layer
Modify current selection by	`METHOD`	[enumeration] Default: 0	How the selection of the algorithm should be managed. One of: • 0 — creating new selection • 1 — adding to current selection • 2 — selecting within current selection • 3 — removing from current selection

Outputs

Label	Name	Type	Description
Input layer	`INPUT`	[same as input]	The input layer with features selected

Python code

Algorithm ID: `native:selectbylocation`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Select within distance

creates a selection in a vector layer. Features are selected wherever they are within the specified maximum distance from the features in an additional reference layer.

See also

Extract within distance, *Select by location*

Parameters

Label	Name	Type	Description
Select features from	`INPUT`	[vector: geometry]	Input vector layer to select features from
By comparing to the features from	`REFERENCE`	[vector: geometry]	Vector layer whose features closeness is used
Where the features are within	`DISTANCE`	[numeric: double] Default: 100.0	The maximum distance around reference features to select input features
Modify current selection by	`METHOD`	[enumeration] Default: 0	How the selection of the algorithm should be managed. One of: • 0 — creating new selection • 1 — adding to current selection • 2 — selecting within current selection • 3 — removing from current selection

Outputs

Label	Name	Type	Description
Input layer	`INPUT`	[same as input]	The input layer with features selected

Python code

Algorithm ID: `native:selectwithindistance`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.29 Vector table

Add autoincremental field

Adds a new integer field to a vector layer, with a sequential value for each feature.

This field can be used as a unique ID for features in the layer. The new attribute is not added to the input layer but a new layer is generated instead.

The initial starting value for the incremental series can be specified. Optionally, the incremental series can be based on grouping fields and a sort order for features can also be specified.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	The input vector layer.
Field name	`FIELD_NAME`	[string] Default: 'AUTO'	Name of the field with autoincremental values
Start values at Optional	`START`	[numeric: integer] Default: 0	Choose the initial number of the incremental count
Modulus value Optional	`MODULUS`	[numeric: integer] Default: 0	Specifying an optional modulus value will restart the count to START whenever the field value reaches the modulus value. `0` means no restart.
Group values by Optional	`GROUP_FIELDS`	[tablefield: any] [list]	Select grouping field(s): instead of a single count run for the whole layer, a separate count is processed for each value returned by the combination of these fields.
Sort expression Optional	`SORT_EXPRESSIO`	[expression]	Use an expression to sort the features in the layer either globally or if set, based on group fields.
Sort ascending	`SORT_ASCENDING`	[boolean] Default: True	When a `sort expression` is set, use this option to control the order in which features are assigned values.

continues on next page

Table 24.272 – continued from previous page

Label	Name	Type	Description
Sort nulls first	SORT_NULLS_FIR	[boolean] Default: False	When a sort expression is set, use this option to set whether *Null* values are counted first or last.
Incremented	OUTPUT	[same as input] Default: [Create temporary layer]	Specify the output vector layer with the auto increment field. *One of*: • Create Temporary Layer (TEMPORARY_OUTPUT) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Incremented	OUTPUT	[same as input]	Vector layer with auto incremental field

Python code

Algorithm ID: native:addautoincrementalfield

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Add field to attributes table

Adds a new field to a vector layer.

The name and characteristics of the attribute are defined as parameters.

The new attribute is not added to the input layer but a new layer is generated instead.

Parameters

Label	Name	Type	Description
Input layer	INPUT	[vector: any]	The input layer
Field name	FIELD_NAME	[string]	Name of the new field

continues on next page

Table 24.273 – continued from previous page

Label	Name	Type	Description
Field type	`FIELD_TYPE`	[enumeration] Default: 0	Type of the new field. You can choose between: • 0 — Integer (32 bit) • 1 — Decimal (double) • 2 — Text (string) • 3 — Boolean • 4 — Date • 5 — Time • 6 — Date & Time • 7 — Binary Object (BLOB) • 8 — String List • 9 — Integer List • 10 — Decimal (double) List
Field length	`FIELD_LENGTH`	[numeric: integer] Default: 10	Length of the field
Field precision	`FIELD_PRECISIO`	[numeric: integer] Default: 0	Precision of the field. Useful with Float field type.
Field alias Optional	`FIELD_ALIAS`	[string]	Set a name to use as alias for the field. Not supported by all format types.
Field comment Optional	`FIELD_COMMENT`	[string]	Store a comment describing the field. Not supported by all format types.
Added	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Added	`OUTPUT`	[same as input]	Vector layer with new field added

Python code

Algorithm ID: `native:addfieldtoattributestable`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Add unique value index field

Takes a vector layer and an attribute and adds a new numeric field.

Values in this field correspond to values in the specified attribute, so features with the same value for the attribute will have the same value in the new numeric field.

This creates a numeric equivalent of the specified attribute, which defines the same classes.

The new attribute is not added to the input layer but a new layer is generated instead.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	The input layer.
Class field	`FIELD`	[tablefield: any]	Features that have the same value for this field will get the same index.
Output field name	`FIELD_NAME`	[string] Default: 'NUM_FIELD'	Name of the new field containing the indexes.
Layer with index field	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Vector layer with the numeric field containing indexes. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.
Class summary	`SUMMARY_OUTPUT`	[vector: table] Default: `[Skip output]`	Specify the table to contain the summary of the class field mapped to the corresponding unique value. *One of*: • Skip Output • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Layer with index field	`OUTPUT`	[same as input]	Vector layer with the numeric field containing indexes.
Class summary	`SUMMARY_OUTPUT`	[vector: table]	Table with summary of the class field mapped to the corresponding unique value.

Python code

Algorithm ID: `native:adduniquevalueindexfield`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Add X/Y fields to layer

Adds X and Y (or latitude/longitude) fields to a point layer. The X/Y fields can be calculated in a different CRS to the layer (e.g. creating latitude/longitude fields for a layer in a projected CRS).

Allows *features in-place modification* of point features

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: point]	The input layer.
Coordinate system	`CRS`	[crs] Default: "EPSG:4326"	Coordinate reference system to use for the generated x and y fields.
Field prefix Optional	`PREFIX`	[string]	Prefix to add to the new field names to avoid name collisions with fields in the input layer.
Added fields	`OUTPUT`	[vector: point] Default: `[Create temporary layer]`	Specify the output layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Added fields	`OUTPUT`	[vector: point]	The output layer - identical to the input layer but with two new double fields, `x` and `y`.

Python code

Algorithm ID: `native:addxyfields`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Advanced Python field calculator

Adds a new attribute to a vector layer, with values resulting from applying an expression to each feature.

The expression is defined as a Python function.

Warning

This algorithm is a potential security risk if executed with unchecked inputs, and may result in system damage or data leaks.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	Input vector layer
Result field name	`FIELD_NAME`	[string] Default: 'NewField'	Name of the new field
Field type	`FIELD_TYPE`	[enumeration] Default: 0	Type of the new field. One of: • 0 — Integer (32 bit) • 1 — Decimal (double) • 2 — Text (string) • 3 — Boolean • 4 — Date • 5 — Time • 6 — Date & Time • 7 — Binary Object (BLOB) • 8 — String List • 9 — Integer List • 10 — Decimal (double) List
Field length	`FIELD_LENGTH`	[numeric: integer] Default: 10	Length of the field
Field precision	`FIELD_PRECISIO`	[numeric: integer] Default: 3	Precision of the field. Useful with Float field type.
Global expression Optional	`GLOBAL`	[string]	The code in the global expression section will be executed only once before the calculator starts iterating through all the features of the input layer. Therefore, this is the correct place to import necessary modules or to calculate variables that will be used in subsequent calculations.

continues on next page

Table 24.276 – continued from previous page

Label	Name	Type	Description
Formula	FORMULA	[string]	The Python formula to evaluate. Example: To calculate the area of an input polygon layer you can add: `value = $geom.area()`
Calculated	OUTPUT	[same as input] Default: [Create temporary layer]	Specify the vector layer with the new calculated field. One of: • Create Temporary Layer (TEMPORARY_OUTPUT) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Calculated	OUTPUT	[same as input]	Vector layer with the new calculated field

Python code

Algorithm ID: `qgis:advancedpythonfieldcalculator`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Drop field(s)

Takes a vector layer and generates a new one that has the same features but without the selected columns.

See also

Retain fields

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	Input vector layer to drop field(s) from
Fields to drop	`COLUMN`	[tablefield: any] [list]	The field(s) to drop
Remaining fields	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer with the remaining fields. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Remaining fields	`OUTPUT`	[same as input]	Vector layer with the remaining fields

Python code

Algorithm ID: `native:deletecolumn`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Explode HStore Field

Creates a copy of the input layer and adds a new field for every unique key in the HStore field.

The expected field list is an optional comma separated list. If this list is specified, only these fields are added and the HStore field is updated. By default, all unique keys are added.

The PostgreSQL HStore is a simple key-value store used in PostgreSQL and GDAL (when reading an OSM file with the `other_tags` field.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
HStore field	`FIELD`	[tablefield: any]	The field(s) to drop
Expected list of fields separated by a comma Optional	`EXPECTED_FIELDS`	[string] Default: ''	Comma-separated list of fields to extract. The HStore field will be updated by removing these keys.
Exploded	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specify the output vector layer. One of: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Exploded	`OUTPUT`	[same as input]	Output vector layer

Python code

Algorithm ID: `native:explodehstorefield`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Extract binary field

Extracts contents from a binary field, saving them to individual files. Filenames can be generated using values taken from an attribute in the source table or based on a more complex expression.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	Input vector layer containing the binary data
Binary field	`FIELD`	[tablefield: any]	Field containing the binary data
File name	`FILENAME`	[expression]	Field or expression-based text to name each output file
Destination folder	`FOLDER`	[folder] Default: `[Save to temporary folder]`	Folder in which to store the output files. *One of*: • Save to a Temporary Directory • Save to Directory

Outputs

Label	Name	Type	Description
Folder	`FOLDER`	[folder]	The folder that contains the output files.

Python code

Algorithm ID: `native:extractbinary`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Field calculator

Opens the field calculator (see *Expressions*). You can use all the supported expressions and functions.

A new layer is created with the result of the expression.

The field calculator is very useful when used in *The model designer*.

Note

This algorithm uses ellipsoid based measurements and respects the current *ellipsoid settings*.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	The layer to calculate on
Output field name	`FIELD_NAME`	[string]	The name of the field for the results
Output field type	`FIELD_TYPE`	[enumeration] Default: 0	The type of the field. One of: • 0 — Decimal (double) • 1 — Integer (32 bit) • 2 — Text (string) • 3 — Date • 4 — Time • 5 — Date & Time • 6 — Boolean • 7 — Binary Object (BLOB) • 8 — String List • 9 — Integer List • 10 — Decimal (double) List
Output field width	`FIELD_LENGTH`	[numeric: integer] Default: 0	The length of the result field (minimum 0)
Field precision	`FIELD_PRECISIO`	[numeric: integer] Default: 0	The precision of the result field (minimum 0, maximum 15)
Create new field	`NEW_FIELD`	[boolean] Default: True	Should the result field be a new field
Formula	`FORMULA`	[expression]	The formula to use to calculate the result
Calculated	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specification of the output layer. • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Calculated	`OUTPUT`	[same as input]	Output layer with the calculated field values

Python code

Algorithm ID: `native:fieldcalculator`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Refactor fields

Allows editing the structure of the attribute table of a vector layer.

Fields can be modified in their type and name, using a fields mapping.

The original layer is not modified. A new layer is generated, which contains a modified attribute table, according to the provided fields mapping.

Note

When using a template layer with *constraints* on fields, the information is displayed in the widget with a coloured background and tooltip. Treat this information as a hint during configuration. No constraints will be added on an output layer nor will they be checked or enforced by the algorithm.

The Refactor fields algorithm allows to:

- Change field names and types
- Add and remove fields
- Reorder fields
- Calculate new fields based on expressions
- Load field list from another layer

Note

This algorithm uses ellipsoid based measurements and respects the current *ellipsoid settings*.

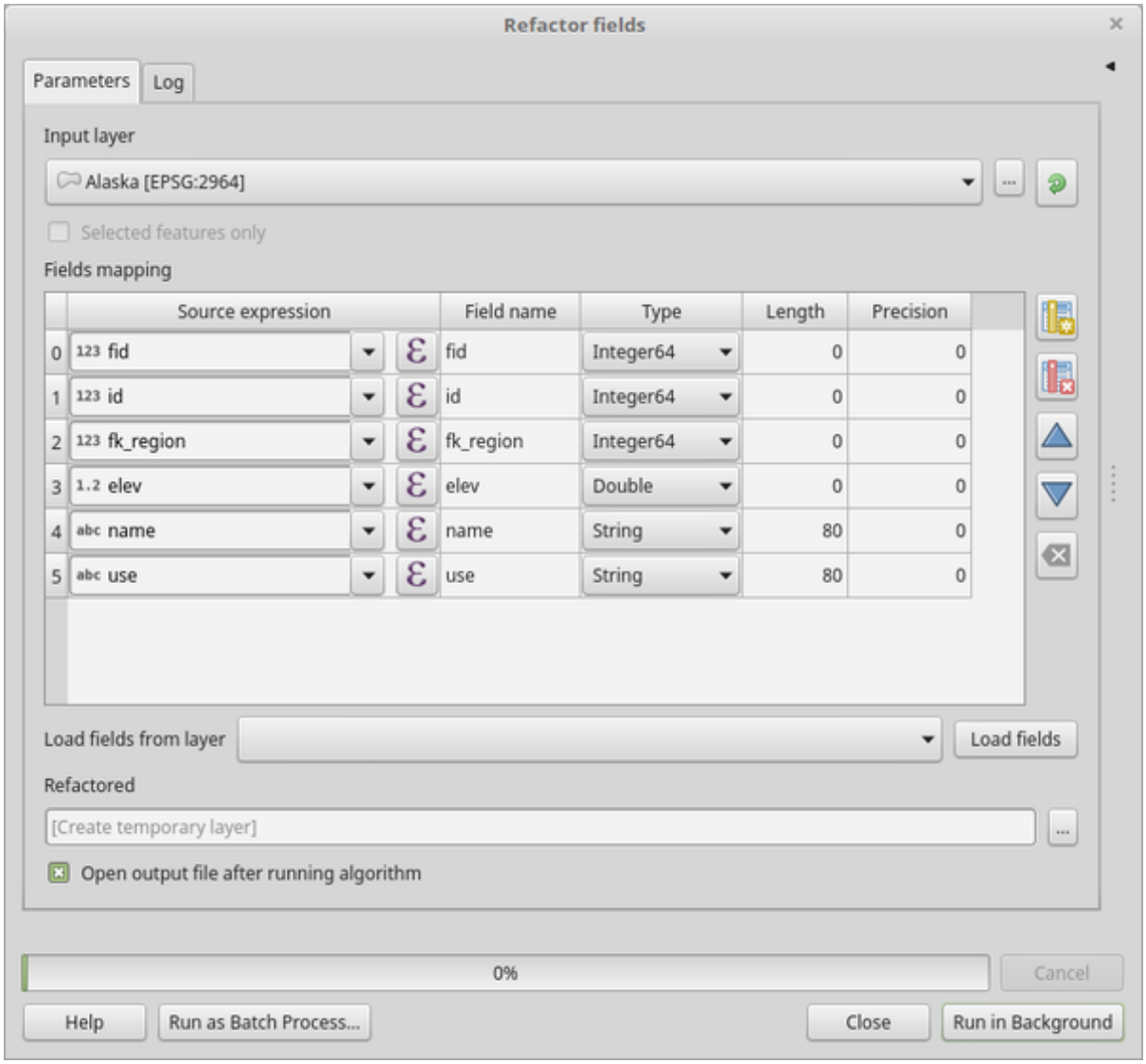


Fig. 24.148: Refactor fields dialog

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	The layer to modify

continues on next page

Table 24.279 – continued from previous page

Label	Name	Type	Description
Fields mapping	`FIELDS_MAPPING`	[list]	List of output fields with their definitions. The embedded table lists all the fields of the source layer and allows you to edit them: • Click  to create a new field. • Click  to remove a field. • Use  and  to change the selected field order. • Click  to reset to the default view. • Click  to invert the selection in the fields list. For each of the fields you'd like to reuse, you need to fill the following options: ***Source expression*** **(`expression`) [expression]** Field or expression from the input layer. ***Name*** **(`name`) [string]** Name of the field in the output layer. By default input field name is kept. ***Type*** **(`type`) [enumeration]** Data type of the output field. Available types may not be compatible with the output layer provider. One of: **Attention** For certain field types, e.g. lists, an extra `sub_type` parameter helps refine the specific type of the data. It is automatically set in the GUI but may be needed if you're running the algorithm in Python or from the command line. • 1 — Boolean • 2 — Integer (32bit) • 4 — Integer (64bit) • 6 — Decimal (double) • 9 — Integer list • 9 — Integer (64bit) list • 9 — Decimal (double) list • 10 — Text (string) • 11 — String list • 12 — Binary Object (BLOB) • 14 — Date • 15 — Time • 16 — Date & Time ***Sub-type*** **(`sub_type`) [enumeration]** For certain field types, e.g. lists, this parameter helps refine the specific `type` of the data. It is automatically set in the GUI but may be needed if you're running the algorithm in Python or from the command line. *One of*: • 2 — Integer list • 4 — Integer (64bit) list • 6 — Decimal (double) list • 10 — String list

Table 24.279 – continued from previous page

Label	Name	Type	Description
Refactored	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specification of the output layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Refactored	`OUTPUT`	[same as input]	Output layer with refactored fields

Python code

Algorithm ID: `native:refactorfields`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Rename field

Renames an existing field from a vector layer.

The original layer is not modified. A new layer is generated where the attribute table contains the renamed field.

See also

Refactor fields

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	The input vector layer
Field to rename	`FIELD`	[tablefield: any]	The field to be altered
New field name	`NEW_NAME`	[string]	The new field name

continues on next page

Table 24.280 – continued from previous page

Label	Name	Type	Description
Renamed	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specification of the output layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Renamed	`OUTPUT`	[same as input]	Output layer with the renamed field

Python code

Algorithm ID: `qgis:renametablefield`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Retain fields

Takes a vector layer and generates a new one that retains only the selected fields. All other fields will be dropped.

See also

Drop field(s)

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	The input vector layer
Fields to retain	`FIELDS`	[tablefield: any] [list]	List of fields to keep in the layer

continues on next page

Table 24.281 – continued from previous page

Label	Name	Type	Description
Retained fields	`OUTPUT`	[same as input] Default: `[Create temporary layer]`	Specification of the output layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Retained fields	`OUTPUT`	[same as input]	Output layer with the retained fields

Python code

Algorithm ID: `native:retainfields`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Text to float

Modifies the type of a given attribute in a vector layer, converting a text attribute containing numeric strings into a numeric attribute (e.g. '1' to `1.0`).

The algorithm creates a new vector layer so the source one is not modified.

If the conversion is not possible the selected column will have `NULL` values.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	The input vector layer.
Text attribute to convert to float	`FIELD`	[tablefield: string]	The string field for the input layer that is to be converted to a float field.
Float from text	`OUTPUT`	[same as input] Default: `[Create Temporary Layer]`	Specify the output layer. *One of*: • Create Temporary Layer (`TEMPORARY_OUTPUT`) • Save to File… • Save to Geopackage… • Save to Database Table… • Append to Layer… The file encoding can also be changed here.

Outputs

Label	Name	Type	Description
Float from text	`OUTPUT`	[same as input]	Output vector layer with the string field converted into a float field

Python code

Algorithm ID: `qgis:texttofloat`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.1.30 Vector Tiles

Download vector tiles

Downloads vector tiles of an input vector tile layer and saves them in a local vector tile file.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector tiles]	A vector tile layer to extract some tiles from
Extent	`EXTENT`	[extent]	Specify the spatial extent of the downloaded area. It will internally be extended to a multiple of the tile size. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Maximum zoom level to download	`MAX_ZOOM`	[numeric: integer] Default: 10	Defines how far to zoom in and fetch data from the tiles
Tile limit	`TILE_LIMIT`	[numeric: integer] Default: 100	Maximum number of tiles to download, considering the zoom levels and the extent.
Output	`OUTPUT`	[vector tiles] Default: [Save to temporary file]	Specification of the output vector tile file. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Output	`OUTPUT`	[vector tiles]	A local vector tile file storing the downloaded tiles.

Python code

Algorithm ID: `native:downloadvectortiles`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Write vector tiles (MBTiles)

Exports one or more vector layers to vector tiles, a data format optimized for fast map rendering and small data size.

MBTiles is a specification for storing tiled map data in SQLite databases for immediate usage and for transfer. MBTiles files are known as tilesets.

Parameters

Label	Name	Type	Description
Input layers	`INPUT`	[vector: any] [list]	A list of layers to combine to generate the vector tiles
Minimum zoom level	`MIN_ZOOM`	[numeric: integer] Default: 0	The lowest zoom level for which the tileset provides data. Set between 0 and 24.
Maximum zoom level	`MAX_ZOOM`	[numeric: integer] Default: 3	The highest zoom level for which the tileset provides data. Set between 0 and 24.
Extent Optional	`EXTENT`	[extent] Default: Not set	The maximum extent of the rendered map area. Bounds must define an area covered by all zoom levels.
Metadata: Name Optional	`META_NAME`	[string]	Name of the tileset
Metadata: Description Optional	`META_DESCRIPTI`	[string]	A description of the tileset's contents
Metadata: Attribution Optional	`META_ATTRIBUTI`	[string]	An attribution string, which explains the sources of data and/or style for the map.
Metadata: Version Optional	`META_VERSION`	[string]	The version of the tileset. This refers to a revision of the tileset itself, not of the MBTiles specification.
Metadata: Type Optional	`META_TYPE`	[string]	Type of tileset. Possible values are `overlay` or `baselayer`.
Metadata: Center Optional	`META_CENTER`	[string]	The center (string of comma-separated numbers: the longitude, latitude, and zoom level) of the default view of the map. Example: `-122.1906,37.7599,11`
Destination MBTiles	`OUTPUT`	[vector tiles] Default: [Save to temporary file]	Specification of the output MBTiles file. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Destination MBTiles	`OUTPUT`	[file]	Output vector tiles `.mbtiles` file.

Python code

Algorithm ID: `native:writevectortiles_mbtiles`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Write vector tiles (XYZ)

Exports one or more vector layers to vector tiles, a data format optimized for fast map rendering and small data size.

Parameters

Label	Name	Type	Description
File template	`XYZ_TEMPLATE`	[string] Default: '{z}/{x}/{y}.pbf'	Template to generate the vector tiles url
Input layers	`INPUT`	[vector: any] [list]	A list of layers to combine to generate the vector tiles
Minimum zoom level	`MIN_ZOOM`	[numeric: integer] Default: 0	The lowest zoom level for which the tileset provides data. Set between 0 and 24.
Maximum zoom level	`MAX_ZOOM`	[numeric: integer] Default: 3	The highest zoom level for which the tileset provides data. Set between 0 and 24.
Extent Optional	`EXTENT`	[extent] Default: Not set	The maximum extent of the rendered map area. Bounds must define an area covered by all zoom levels.
Output directory	`OUTPUT_DIRECTORY`	[folder] Default: [Save to temporary folder]	Specification of the output vector tiles folder. *One of*: • Save to a Temporary Directory • Save to Directory

Outputs

Label	Name	Type	Description
Output directory	`OUTPUT_DIRECTORY`	[folder]	A folder containing different subsets of the vector tiles files (`.pbf`) stored in subfolders corresponding to the zoom levels.

Python code

Algorithm ID: `native:writevectortiles_xyz`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.2 GDAL algorithm provider

GDAL (Geospatial Data Abstraction Library) is a translator library for raster and vector geospatial data formats. Algorithms in the Processing Framework are derived from the GDAL raster programs and GDAL vector programs.

24.2.1 Raster analysis

Aspect

Generates an aspect map from any GDAL-supported elevation raster. Aspect is the compass direction that a slope faces. The pixels will have a value from 0-360° measured in degrees from north indicating the azimuth. On the northern hemisphere, the north side of slopes is often shaded (small azimuth from 0°-90°), while the southern side receives more solar radiation (higher azimuth from 180°-270°).

This algorithm is derived from the GDAL DEM utility.

Default menu: *Raster* ► *Analysis*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Input elevation raster layer
Band number	`BAND`	[raster band] Default: 1	The number of the band to use as elevation
Return trigonometric angle instead of azimuth	`TRIG_ANGLE`	[boolean] Default: False	Activating the trigonometric angle results in different categories: 0° (East), 90° (North), 180° (West), 270° (South).
Return 0 for flat instead of -9999	`ZERO_FLAT`	[boolean] Default: False	Activating this option will insert a 0-value for the value -9999 on flat areas.
Compute edges	`COMPUTE_EDGES`	[boolean] Default: False	Generates edges from the elevation raster
Use Zevenbergen&Thorne formula instead of the Horn's one	`ZEVENBERGEN`	[boolean] Default: False	Activates Zevenbergen&Thorne formula for smooth landscapes

continues on next page

Table 24.282 – continued from previous page

Label	Name	Type	Description
Aspect	OUTPUT	[raster] Default: [Save to temporary file]	Output raster layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Additional creation options Optional	CREATION_OPTIONS (for QGIS <= 3.42, this was OPTIONS)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).
Additional command-line parameters Optional	EXTRA	[string] Default: Not set	Add extra GDAL command line options

Outputs

Label	Name	Type	Description
Aspect	OUTPUT	[raster]	Output raster with angle values in degrees

Python code

Algorithm ID: `gdal:aspect`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Color relief

Generates a color relief map from any GDAL-supported elevation raster. Color reliefs can particularly be used to depict elevations. The Algorithm outputs a 4-band raster with values computed from the elevation and a text-based color configuration file. By default, the colors between the given elevation values are blended smoothly and the result is a nice colorized elevation raster.

This algorithm is derived from the GDAL DEM utility.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Input elevation raster layer
Band number	`BAND`	[raster band] Default: 1	The number of the band to use as elevation
Compute edges	`COMPUTE_EDGES`	[boolean] Default: False	Generates edges from the elevation raster
Color configuration file	`COLOR_TABLE`	[file]	A text-based color configuration file
Matching mode	`MATCH_MODE`	[enumeration] Default: 2	One of: • 0 — Use strict color matching • 1 — Use closest RGBA quadruples • 2 — Use smoothly blended colours
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).
Additional command-line parameters Optional	`EXTRA`	[string] Default: Not set	Add extra GDAL command line options
Color relief	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Output raster layer. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Color relief	`OUTPUT`	[raster]	A 4-band output raster

Python code

Algorithm ID: `gdal:colorrelief`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Fill NoData

Fill raster regions with NoData values by interpolation from edges. The values for the NoData regions are calculated by the surrounding pixel values using inverse distance weighting. After the interpolation a smoothing of the results takes place. Input can be any GDAL-supported raster layer. This algorithm is generally suitable for interpolating missing regions of fairly continuously varying rasters (such as elevation models for instance). It is also suitable for filling small holes and cracks in more irregularly varying images (like airphotos). It is generally not so great for interpolating a raster from sparse point data.

This algorithm is derived from the GDAL fillnodata utility.

Default menu: *Raster* ► *Analysis*

Parameters

Basic parameters

Label	Name	Type	Description
Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Input raster layer
Band number	`BAND`	[raster band] Default: 1	The band to operate on. NoData values must be represented by the value 0.
Maximum distance (in pixels) to search out for values to interpolate	`DISTANCE`	[numeric: integer] Default: 10	The number of pixels to search in all directions to find values to interpolate from
Number of smoothing iterations to run after the interpolation	`ITERATIONS`	[numeric: integer] Default: 0	The number of 3x3 filter passes to run (0 or more) to smoothen the results of the interpolation.
Validity mask	`MASK_LAYER`	[raster]	A raster layer that defines the areas to fill.
Filled	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output raster layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Additional creation options Optional	CREATION_OPTIONS (for QGIS <= 3.42, this was OPTIONS)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).
Additional command-line parameters Optional	EXTRA	[string] Default: Not set	Add extra GDAL command line options

Outputs

Label	Name	Type	Description
Filled	OUTPUT	[raster]	Output raster

Python code

Algorithm ID: `gdal:fillnodata`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Grid (Data metrics)

Computes some data metrics using the specified window and output grid geometry.

This algorithm is derived from the GDAL grid utility.

Default menu: *Raster* ► *Analysis*

See also

GDAL grid tutorial

Parameters

Basic parameters

Label	Name	Type	Description
Point layer	`INPUT`	[vector: point]	Input point vector layer
Data metric to use	`METRIC`	[enumeration] Default: 0	One of: • 0 — Minimum, minimum value found in grid node search ellipse • 1 — Maximum, maximum value found in grid node search ellipse • 2 — Range, a difference between the minimum and maximum values found in grid node search ellipse • 3 — Count, a number of data points found in grid node search ellipse • 4 — Average distance, an average distance between the grid node (center of the search ellipse) and all of the data points found in grid node search ellipse • 5 — Average distance between points, an average distance between the data points found in grid node search ellipse. The distance between each pair of points within ellipse is calculated and average of all distances is set as a grid node value
The first radius of search ellipse	`RADIUS_1`	[numeric: double] Default: 0.0	The first radius (X axis if rotation angle is 0) of the search ellipse
The second radius of search ellipse	`RADIUS_2`	[numeric: double] Default: 0.0	The second radius (Y axis if rotation angle is 0) of the search ellipse
Angle of search ellipse rotation in degrees (counter clockwise)	`ANGLE`	[numeric: double] Default: 0.0	Angle of ellipse rotation in degrees. Ellipse rotated counter clockwise.
Minimum number of data points to use	`MIN_POINTS`	[numeric: double] Default: 0.0	Minimum number of data points to average. If less amount of points found the grid node considered empty and will be filled with NoData marker.
NoData	`NODATA`	[numeric: double] Default: 0.0	NoData marker to fill empty points
Interpolated (data metrics)	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specify the output raster layer with interpolated values. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Z value from field Optional	`Z_FIELD`	[tablefield: numeric]	Field for the interpolation
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).
Additional command-line parameters Optional	`EXTRA`	[string] Default: Not set	Add extra GDAL command line options
Output data type	`DATA_TYPE`	[enumeration] Default: 5	Defines the data type of the output raster file. Options: • 0 — Byte (Eight bit unsigned integer (quint8)) • 1 — Int16 (Sixteen bit signed integer (qint16)) • 2 — UInt16 (Sixteen bit unsigned integer (quint16)) • 3 — UInt32 (Thirty two bit unsigned integer (quint32)) • 4 — Int32 (Thirty two bit signed integer (qint32)) • 5 — Float32 (Thirty two bit floating point (float)) • 6 — Float64 (Sixty four bit floating point (double)) • 7 — CInt16 (Complex Int16) • 8 — CInt32 (Complex Int32) • 9 — CFloat32 (Complex Float32) • 10 — CFloat64 (Complex Float64) • 11 — Int8 (Eight bit signed integer (qint8)) Available options depend on the GDAL version built with QGIS (see *Help* ► *About* menu)

Outputs

Label	Name	Type	Description
Interpolated (data metrics)	`OUTPUT`	[raster]	Output raster with interpolated values

Python code

Algorithm ID: `gdal:griddatametrics`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Grid (IDW with nearest neighbor searching)

Computes the Inverse Distance to a Power gridding combined to the nearest neighbor method. Ideal when a maximum number of data points to use is required.

This algorithm is derived from the GDAL grid utility.

See also

GDAL grid tutorial

Parameters

Label	Name	Type	Description
Point layer	`INPUT`	[vector: point]	Input point vector layer
Weighting power	`POWER`	[numeric: double] Default: 2.0	Weighting power
Smoothing	`SMOOTHING`	[numeric: double] Default: 0.0	Smoothing parameter
The radius of the search circle	`RADIUS`	[numeric: double] Default: 1.0	The radius of the search circle
Maximum number of data points to use	`MAX_POINTS`	[numeric: integer] Default: 12	Do not search for more points than this number.
Minimum number of data points to use	`MIN_POINTS`	[numeric: integer] Default: 0	Minimum number of data points to average. If less amount of points found the grid node considered empty and will be filled with NoData marker.
NoData	`NODATA`	[numeric: double] Default: 0.0	NoData marker to fill empty points

continues on next page

Table 24.289 – continued from previous page

Label	Name	Type	Description
Interpolated (IDW with NN search)	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specify the output raster layer with interpolated values. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Z value from field Optional	`Z_FIELD`	[tablefield: numeric]	Field for the interpolation
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).
Additional command-line parameters Optional	`EXTRA`	[string] Default: Not set	Add extra GDAL command line options
Output data type	`DATA_TYPE`	[enumeration] Default: 5	Defines the data type of the output raster file. Options: • 0 — Byte (Eight bit unsigned integer (quint8)) • 1 — Int16 (Sixteen bit signed integer (qint16)) • 2 — UInt16 (Sixteen bit unsigned integer (quint16)) • 3 — UInt32 (Thirty two bit unsigned integer (quint32)) • 4 — Int32 (Thirty two bit signed integer (qint32)) • 5 — Float32 (Thirty two bit floating point (float)) • 6 — Float64 (Sixty four bit floating point (double)) • 7 — CInt16 (Complex Int16) • 8 — CInt32 (Complex Int32) • 9 — CFloat32 (Complex Float32) • 10 — CFloat64 (Complex Float64) • 11 — Int8 (Eight bit signed integer (qint8)) Available options depend on the GDAL version built with QGIS (see *Help* ► *About* menu)

Outputs

Label	Name	Type	Description
Interpolated (IDW with NN search)	`OUTPUT`	[raster]	Output raster with interpolated values

Python code

Algorithm ID: `gdal:gridinversedistancenearestneighbor`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Grid (Inverse distance to a power)

The Inverse Distance to a Power gridding method is a weighted average interpolator.

You should supply the input arrays with the scattered data values including coordinates of every data point and output grid geometry. The function will compute interpolated value for the given position in output grid.

This algorithm is derived from the GDAL grid utility.

Default menu: *Raster* ► *Analysis*

See also

GDAL grid tutorial

Parameters

Basic parameters

Label	Name	Type	Description
Point layer	`INPUT`	[vector: point]	Input point vector layer
Weighting power	`POWER`	[numeric: double] Default: 2.0	Weighting power
Smothing	`SMOOTHING`	[numeric: double] Default: 0.0	Smoothing parameter
The first radius of search ellipse	`RADIUS_1`	[numeric: double] Default: 0.0	The first radius (X axis if rotation angle is 0) of the search ellipse
The second radius of search ellipse	`RADIUS_2`	[numeric: double] Default: 0.0	The second radius (Y axis if rotation angle is 0) of the search ellipse
Angle of search ellipse rotation in degrees (counter clockwise)	`ANGLE`	[numeric: double] Default: 0.0	Angle of ellipse rotation in degrees. Ellipse rotated counter clockwise.

continues on next page

Table 24.291 – continued from previous page

Label	Name	Type	Description
Maximum number of data points to use	`MAX_POINTS`	[numeric: integer] Default: 0	Do not search for more points than this number.
Minimum number of data points to use	`MIN_POINTS`	[numeric: integer] Default: 0	Minimum number of data points to average. If less amount of points found the grid node considered empty and will be filled with NoData marker.
NoData	`NODATA`	[numeric: double] Default: 0.0	NoData marker to fill empty points
Interpolated (IDW)	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specify the output raster layer with interpolated values. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Z value from field Optional	`Z_FIELD`	[tablefield: numeric]	Field for the interpolation
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).
Additional command-line parameters Optional	`EXTRA`	[string] Default: Not set	Add extra GDAL command line options

continues on next page

Table 24.292 – continued from previous page

Label	Name	Type	Description
Output data type	`DATA_TYPE`	[enumeration] Default: 5	Defines the data type of the output raster file. Options: • 0 — Byte (Eight bit unsigned integer (quint8)) • 1 — Int16 (Sixteen bit signed integer (qint16)) • 2 — UInt16 (Sixteen bit unsigned integer (quint16)) • 3 — UInt32 (Thirty two bit unsigned integer (quint32)) • 4 — Int32 (Thirty two bit signed integer (qint32)) • 5 — Float32 (Thirty two bit floating point (float)) • 6 — Float64 (Sixty four bit floating point (double)) • 7 — CInt16 (Complex Int16) • 8 — CInt32 (Complex Int32) • 9 — CFloat32 (Complex Float32) • 10 — CFloat64 (Complex Float64) • 11 — Int8 (Eight bit signed integer (qint8)) Available options depend on the GDAL version built with QGIS (see *Help* ► *About* menu)

Outputs

Label	Name	Type	Description
Interpolated (IDW)	`OUTPUT`	[raster]	Output raster with interpolated values

Python code

Algorithm ID: `gdal:gridinversedistance`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Grid (Linear)

The Linear method perform linear interpolation by computing a Delaunay triangulation of the point cloud, finding in which triangle of the triangulation the point is, and by doing linear interpolation from its barycentric coordinates within the triangle. If the point is not in any triangle, depending on the radius, the algorithm will use the value of the nearest point or the NoData value.

This algorithm is derived from the GDAL grid utility.

Parameters

Basic parameters

Label	Name	Type	Description
Point layer	`INPUT`	[vector: point]	Input point vector layer
Search distance	`RADIUS`	[numeric: double] Default: -1.0	In case the point to be interpolated does not fit into a triangle of the Delaunay triangulation, use that maximum distance to search a nearest neighbour, or use NoData otherwise. If set to `-1`, the search distance is infinite. If set to `0`, NoData value will be used.
NoData	`NODATA`	[numeric: double] Default: 0.0	NoData marker to fill empty points
Interpolated (Linear)	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specify the output raster layer with interpolated values. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Z value from field Optional	`Z_FIELD`	[tablefield: numeric]	Field for the interpolation
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (`|`).
Additional command-line parameters Optional	`EXTRA`	[string] Default: Not set	Add extra GDAL command line options

continues on next page

Table 24.294 – continued from previous page

Label	Name	Type	Description
Output data type	`DATA_TYPE`	[enumeration] Default: 5	Defines the data type of the output raster file. Options: • 0 — Byte (Eight bit unsigned integer (quint8)) • 1 — Int16 (Sixteen bit signed integer (qint16)) • 2 — UInt16 (Sixteen bit unsigned integer (quint16)) • 3 — UInt32 (Thirty two bit unsigned integer (quint32)) • 4 — Int32 (Thirty two bit signed integer (qint32)) • 5 — Float32 (Thirty two bit floating point (float)) • 6 — Float64 (Sixty four bit floating point (double)) • 7 — CInt16 (Complex Int16) • 8 — CInt32 (Complex Int32) • 9 — CFloat32 (Complex Float32) • 10 — CFloat64 (Complex Float64) • 11 — Int8 (Eight bit signed integer (qint8)) Available options depend on the GDAL version built with QGIS (see *Help* ► *About* menu)

Outputs

Label	Name	Type	Description
Interpolated (Linear)	`OUTPUT`	[raster]	Output raster with interpolated values

Python code

Algorithm ID: `gdal:gridlinear`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Grid (Moving average)

The Moving Average is a simple data averaging algorithm. It uses a moving window of elliptic form to search values and averages all data points within the window. Search ellipse can be rotated by specified angle, the center of ellipse located at the grid node. Also the minimum number of data points to average can be set, if there are not enough points in window, the grid node considered empty and will be filled with specified NoData value.

This algorithm is derived from the GDAL grid utility.

Default menu: *Raster* ► *Analysis*

See also

GDAL grid tutorial

Parameters

Basic parameters

Label	Name	Type	Description
Point layer	`INPUT`	[vector: point]	Input point vector layer
The first radius of search ellipse	`RADIUS_1`	[numeric: double] Default: 0.0	The first radius (X axis if rotation angle is 0) of the search ellipse
The second radius of search ellipse	`RADIUS_2`	[numeric: double] Default: 0.0	The second radius (Y axis if rotation angle is 0) of the search ellipse
Angle of search ellipse rotation in degrees (counter clockwise)	`ANGLE`	[numeric: double] Default: 0.0	Angle of ellipse rotation in degrees. Ellipse rotated counter clockwise.
Minimum number of data points to use	`MIN_POINTS`	[numeric: integer] Default: 0	Minimum number of data points to average. If less amount of points found the grid node considered empty and will be filled with NoData marker.
NoData	`NODATA`	[numeric: double] Default: 0.0	NoData marker to fill empty points
Interpolated (moving average)	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specify the output raster layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Z value from field Optional	`Z_FIELD`	[tablefield: numeric]	Field for the interpolation

continues on next page

Table 24.296 – continued from previous page

Label	Name	Type	Description
Additional creation options Optional	CREATION_OPTIONS (for QGIS <= 3.42, this was OPTIONS)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).
Additional command-line parameters Optional	EXTRA	[string] Default: Not set	Add extra GDAL command line options
Output data type	DATA_TYPE	[enumeration] Default: 5	Defines the data type of the output raster file. Options: • 0 — Byte (Eight bit unsigned integer (quint8)) • 1 — Int16 (Sixteen bit signed integer (qint16)) • 2 — UInt16 (Sixteen bit unsigned integer (quint16)) • 3 — UInt32 (Thirty two bit unsigned integer (quint32)) • 4 — Int32 (Thirty two bit signed integer (qint32)) • 5 — Float32 (Thirty two bit floating point (float)) • 6 — Float64 (Sixty four bit floating point (double)) • 7 — CInt16 (Complex Int16) • 8 — CInt32 (Complex Int32) • 9 — CFloat32 (Complex Float32) • 10 — CFloat64 (Complex Float64) • 11 — Int8 (Eight bit signed integer (qint8)) Available options depend on the GDAL version built with QGIS (see *Help* ► *About* menu)

Outputs

Label	Name	Type	Description
Interpolated (moving average)	OUTPUT	[raster]	Output raster with interpolated values

Python code

Algorithm ID: `gdal:gridaverage`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Grid (Nearest neighbor)

The Nearest Neighbor method doesn't perform any interpolation or smoothing, it just takes the value of nearest point found in grid node search ellipse and returns it as a result. If there are no points found, the specified NoData value will be returned.

This algorithm is derived from the GDAL grid utility.

Default menu: *Raster* ► *Analysis*

See also

GDAL grid tutorial

Parameters

Basic parameters

Label	Name	Type	Description
Point layer	`INPUT`	[vector: point]	Input point vector layer
The first radius of search ellipse	`RADIUS_1`	[numeric: double] Default: 0.0	The first radius (X axis if rotation angle is 0) of the search ellipse
The second radius of search ellipse	`RADIUS_2`	[numeric: double] Default: 0.0	The second radius (Y axis if rotation angle is 0) of the search ellipse
Angle of search ellipse rotation in degrees (counter clockwise)	`ANGLE`	[numeric: double] Default: 0.0	Angle of ellipse rotation in degrees. Ellipse rotated counter clockwise.
NoData	`NODATA`	[numeric: double] Default: 0.0	NoData marker to fill empty points
Interpolated (Nearest neighbour)	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specify the output raster layer with interpolated values. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Z value from field Optional	`Z_FIELD`	[tablefield: numeric]	Field for the interpolation
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).
Additional command-line parameters Optional	`EXTRA`	[string] Default: Not set	Add extra GDAL command line options
Output data type	`DATA_TYPE`	[enumeration] Default: 5	Defines the data type of the output raster file. Options: • 0 — Byte (Eight bit unsigned integer (quint8)) • 1 — Int16 (Sixteen bit signed integer (qint16)) • 2 — UInt16 (Sixteen bit unsigned integer (quint16)) • 3 — UInt32 (Thirty two bit unsigned integer (quint32)) • 4 — Int32 (Thirty two bit signed integer (qint32)) • 5 — Float32 (Thirty two bit floating point (float)) • 6 — Float64 (Sixty four bit floating point (double)) • 7 — CInt16 (Complex Int16) • 8 — CInt32 (Complex Int32) • 9 — CFloat32 (Complex Float32) • 10 — CFloat64 (Complex Float64) • 11 — Int8 (Eight bit signed integer (qint8)) Available options depend on the GDAL version built with QGIS (see *Help* ► *About* menu)

Outputs

Label	Name	Type	Description
Interpolated (Nearest neighbour)	`OUTPUT`	[raster]	Output raster with interpolated values

Python code

Algorithm ID: `gdal:gridnearestneighbor`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Hillshade

Outputs a raster with a nice shaded relief effect. It's very useful for visualizing the terrain. You can optionally specify the azimuth and altitude of the light source, a vertical exaggeration factor and a scaling factor to account for differences between vertical and horizontal units.

This algorithm is derived from the GDAL DEM utility .

Default menu: *Raster* ► *Analysis*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Input Elevation raster layer
Band number	`BAND`	[raster band] Default: 1	Band containing the elevation information
Z factor (vertical exaggeration)	`Z_FACTOR`	[numeric: double] Default: 1.0	The factor exaggerates the height of the output elevation raster
Scale (ratio of vertical units to horizontal)	`SCALE`	[numeric: double] Default: 1.0	The ratio of vertical units to horizontal units
Azimuth of the light	`AZIMUTH`	[numeric: double] Default: 315.0	Defines the azimuth of the light shining on the elevation raster in degrees. If it comes from the top of the raster the value is 0, if it comes from the east it is 90 a.s.o.
Altitude of the light	`ALTITUDE`	[numeric: double] Default: 45.0	Defines the altitude of the light, in degrees. 90 if the light comes from above the elevation raster, 0 if it is raking light.
Compute edges	`COMPUTE_EDGES`	[boolean] Default: False	Generates edges from the elevation raster

continues on next page

Table 24.299 – continued from previous page

Label	Name	Type	Description
Use Zevenbergen&Thorne formula (instead of the Horn's one)	`ZEVENBERGEN`	[boolean] Default: False	Activates Zevenbergen&Thorne formula for smooth landscapes
Combined shading	`COMBINED`	[boolean] Default: False	
Multidirectional shading	`MULTIDIRECTIONAL`	[boolean] Default: False	
Hillshade	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specify the output raster layer with interpolated values. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (`|`).
Additional command-line parameters Optional	`EXTRA`	[string] Default: Not set	Add extra GDAL command line options

Outputs

Label	Name	Type	Description
Hillshade	`OUTPUT`	[raster]	Output raster with interpolated values

Python code

Algorithm ID: `gdal:hillshade`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Near black

Converts nearly black/white borders to black.

This algorithm will scan an image and try to set all pixels that are nearly or exactly black, white or one or more custom colors around the collar to black or white. This is often used to "fix up" lossy compressed airphotos so that color pixels can be treated as transparent when mosaicking.

This algorithm is derived from the GDAL nearblack utility.

Default menu: *Raster* ► *Analysis*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Input Elevation raster layer
How far from black (white)	`NEAR`	[numeric: integer] Default: 15	Select how far from black, white or custom colors the pixel values can be and still considered near black, white or custom color.
Search for nearly white pixels instead of nearly black	`WHITE`	[boolean] Default: False	Search for nearly white (255) pixels instead of nearly black pixels
Nearblack	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specify the output raster layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (`|`).
Additional command-line parameters Optional	`EXTRA`	[string] Default: Not set	Add extra GDAL command line options

Outputs

Label	Name	Type	Description
Nearblack	`OUTPUT`	[raster]	Output raster

Python code

Algorithm ID: `gdal:nearblack`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Proximity (raster distance)

Generates a raster proximity map indicating the distance from the center of each pixel to the center of the nearest pixel identified as a target pixel. Target pixels are those in the source raster for which the raster pixel value is in the set of target pixel values.

This algorithm is derived from the GDAL proximity utility.

Default menu: *Raster* ► *Analysis*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Input Elevation raster layer
Band number	`BAND`	[raster band] Default: 1	Band containing the elevation information
List of target pixels Optional	`VALUES`	[string] Default: ''	Comma-separated list of pixel values in the source image to consider as target pixels. If not specified, all non-zero pixels will be considered target pixels.
Distance units	`UNITS`	[enumeration] Default: 1	Indicate whether distances generated should be in pixel or georeferenced coordinates. One of: • 0 — Georeferenced coordinates • 1 — Pixel coordinates
The maximum distance to generate Optional	`MAX_DISTANCE`	[numeric: double] Default: 0.0	The maximum distance to be generated. The NoData value will be used for pixels beyond this distance. Distance is interpreted according to the value of the *Distance units* parameter.

continues on next page

Table 24.303 – continued from previous page

Label	Name	Type	Description
Value to apply to pixels within the maximum distance Optional	`REPLACE`	[numeric: double] Default: 0.0	Specify a value to apply to all pixels that are within the maximum distance from target pixels (including the target pixels) instead of a distance value.
NoData value to use for the destination proximity raster Optional	`NODATA`	[numeric: double] Default: 0.0	Specify the NoData value to use for the pixels beyond the maximum distance. If a NoData value is not provided, the output band will be queried for its NoData value. If the output band does not have a NoData value, then the value `65535` will be used.
Proximity map	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specify the output raster layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (`|`).
Additional command-line parameters Optional	`EXTRA`	[string] Default: Not set	Add extra GDAL command line options

continues on next page

Table 24.304 – continued from previous page

Label	Name	Type	Description
Output data type	DATA_TYPE	[enumeration] Default: 5	Defines the data type of the output raster file. Options: • 0 — Byte (Eight bit unsigned integer (quint8)) • 1 — Int16 (Sixteen bit signed integer (qint16)) • 2 — UInt16 (Sixteen bit unsigned integer (quint16)) • 3 — UInt32 (Thirty two bit unsigned integer (quint32)) • 4 — Int32 (Thirty two bit signed integer (qint32)) • 5 — Float32 (Thirty two bit floating point (float)) • 6 — Float64 (Sixty four bit floating point (double)) • 7 — CInt16 (Complex Int16) • 8 — CInt32 (Complex Int32) • 9 — CFloat32 (Complex Float32) • 10 — CFloat64 (Complex Float64) • 11 — Int8 (Eight bit signed integer (qint8)) Available options depend on the GDAL version built with QGIS (see *Help* ► *About* menu)

Outputs

Label	Name	Type	Description
Proximity map	OUTPUT	[raster]	Output raster

Python code

Algorithm ID: `gdal:proximity`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Roughness

Outputs a single-band raster with values computed from the elevation. Roughness is the degree of irregularity of the surface. It's calculated by the largest inter-cell difference of a central pixel and its surrounding cell. The determination of the roughness plays a role in the analysis of terrain elevation data, it's useful for calculations of the river morphology, in climatology and physical geography in general.

This algorithm is derived from the GDAL DEM utility.

Default menu: *Raster* ► *Analysis*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Input elevation raster layer
Band number	`BAND`	[raster band] Default: 1	The number of the band to use as elevation
Compute edges	`COMPUTE_EDGES`	[boolean] Default: False	Generates edges from the elevation raster
Roughness	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specify the output raster layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (`|`).

Outputs

Label	Name	Type	Description
Roughness	`OUTPUT`	[raster]	Single-band output roughness raster. The value -9999 is used as NoData value.

Python code

Algorithm ID: `gdal:roughness`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Sieve

Removes raster polygons smaller than a provided threshold size (in pixels) and replaces them with the pixel value of the largest neighbour polygon. It is useful if you have a large amount of small areas on your raster map.

This algorithm is derived from the GDAL sieve utility.

Default menu: *Raster* ► *Analysis*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Input elevation raster layer
Threshold	`THRESHOLD`	[numeric: integer] Default: 10	Only raster polygons smaller than this size will be removed
Use 8-connectedness	`EIGHT_CONNECTE`	[boolean] Default: False	If False, consider as neighbours only the (4) pixels that connect the edges horizontally or vertically. If True, every pixel that touches an edge or a corner is considered as neighbour.
Do not use the default validity mask for the input band	`NO_MASK`	[boolean] Default: False	
Validity mask Optional	`MASK_LAYER`	[raster]	Validity mask to use instead of the default
Sieved	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specify the output raster layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Additional command-line parameters Optional	`EXTRA`	[string] Default: Not set	Add extra GDAL command line options

Outputs

Label	Name	Type	Description
Sieved	`OUTPUT`	[raster]	Output raster layer.

Python code

Algorithm ID: `gdal:sieve`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Slope

Generates a slope map from any GDAL-supported elevation raster. Slope is the angle of inclination to the horizontal. You have the option of specifying the type of slope value you want: degrees or percent slope.

This algorithm is derived from the GDAL DEM utility.

Default menu: *Raster* ► *Analysis*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Input Elevation raster layer
Band number	`BAND`	[raster band] Default: 1	Band containing the elevation information
Ratio of vertical units to horizontal	`SCALE`	[numeric: double] Default: 1.0	The ratio of vertical units to horizontal units
Express slope as percent (instead of degrees)	`AS_PERCENT`	[boolean] Default: False	Express slope as percent instead of degrees
Compute edges	`COMPUTE_EDGES`	[boolean] Default: False	Generates edges from the elevation raster
Use Zevenbergen&Thorne formula (instead of the Horn's one)	`ZEVENBERGEN`	[boolean] Default: False	Activates Zevenbergen&Thorne formula for smooth landscapes
Slope	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specify the output raster layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Additional creation options Optional	CREATION_OPTIONS (for QGIS <= 3.42, this was OPTIONS)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).
Additional command-line parameters Optional	EXTRA	[string] Default: Not set	Add extra GDAL command line options

Outputs

Label	Name	Type	Description
Slope	OUTPUT	[raster]	Output raster

Python code

Algorithm ID: `gdal:slope`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Terrain Ruggedness Index (TRI)

Outputs a single-band raster with values computed from the elevation. TRI stands for Terrain Ruggedness Index, which is defined as the mean difference between a central pixel and its surrounding cells.

This algorithm is derived from the GDAL DEM utility.

Default menu: *Raster* ► *Analysis*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	INPUT	[raster]	Input elevation raster layer

continues on next page

Table 24.311 – continued from previous page

Label	Name	Type	Description
Band number	`BAND`	[raster band] Default: 1	The number of the band to use as elevation
Compute edges	`COMPUTE_EDGES`	[boolean] Default: False	Generates edges from the elevation raster
Terrain Ruggedness Index	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specify the output raster layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (`|`).

Outputs

Label	Name	Type	Description
Terrain Ruggedness Index	`OUTPUT`	[raster]	Output ruggedness raster. The value -9999 is used as NoData value.

Python code

Algorithm ID: `gdal:triterrainruggednessindex`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Topographic Position Index (TPI)

Outputs a single-band raster with values computed from the elevation. TPI stands for Topographic Position Index, which is defined as the difference between a central pixel and the mean of its surrounding cells.

This algorithm is derived from the GDAL DEM utility.

Default menu: *Raster* ► *Analysis*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Input elevation raster layer
Band number	`BAND`	[raster band] Default: 1	The number of the band to use for elevation values
Compute edges	`COMPUTE_EDGES`	[boolean] Default: False	Generates edges from the elevation raster
Terrain Ruggedness Index	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specify the output raster layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (`|`).

Outputs

Label	Name	Type	Description
Terrain Ruggedness Index	`OUTPUT`	[raster]	Output raster.

Python code

Algorithm ID: `gdal:tpitopographicpositionindex`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.2.2 Raster conversion

gdal2xyz

Converts raster data to XYZ ASCII file format.

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Raster layer to convert
Band number	`BAND`	[raster band] Default: The first band of the input layer	If the raster is multiband, choose the band you want to convert
Input pixel value to treat as NoData Optional	`NODATA_INPUT`	[numeric: integer] Default: Not set	Input pixel value to treat as "NoData" (GDAL >= 3.7).
Destination No-Data Optional	`NODATA_OUTPUT`	[numeric: integer] Default: Not set	Assign specified "NoData" value to output (GDAL >= 3.7).
Do not output No-Data values	`SKIP_NODATA`	[boolean] Default: False	Do not output "NoData" values (GDAL >= 3.3).
Output comma-separated values	`CSV`	[boolean] Default: False	Sets whether the output file should be of type comma-separated values (csv).
XYZ ASCII file	`OUTPUT`	[file] Default: `[Save to temporary file]`	Specification of the output file. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
XYZ ASCII file	`INPUT`	[table]	Table file containing the values exported from the raster band.

Python code

Algorithm ID: `gdal:gdal2xyz`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

PCT to RGB

Converts an 8 bit paletted image to a 24 bit RGB. It will convert a pseudocolor band from the input file to an RGB file of the desired format.

This algorithm is derived from the GDAL pct2rgb utility.

Default menu: *Raster* ► *Conversion*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Input 8 bit raster image
Band number	`BAND`	[raster band] Default: The first band of the input layer	If the raster is multiband, choose the band you want to convert
Generate a RGBA file	`RGBA`	[boolean] Default: False	Sets whether the output file should be of type RGBA.
PCT to RGB	`OUTPUT`	[file] Default: `[Save to temporary file]`	Specification of the output file. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
PCT to RGB	`OUTPUT`	[raster]	24 bit RGB raster image

Python code

Algorithm ID: `gdal:pcttorgb`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Polygonize (raster to vector)

Creates vector polygons for all connected regions of pixels in the raster sharing a common pixel value. Each polygon is created with an attribute indicating the pixel value of that polygon.

This algorithm is derived from the GDAL polygonize utility.

Default menu: *Raster* ► *Conversion*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Input raster layer
Band number	`BAND`	[raster band] Default: The first band of the input layer	If the raster is multiband, choose the band you want to use
Name of the field to create	`FIELD`	[string] Default: 'DN'	Specify the field name for the attributes of the connected regions.
Use 8-connectedness	`EIGHT_CONNECTE`	[boolean] Default: False	If not set, raster cells must have a common border to be considered connected (*4-connected*). If set, touching raster cells are also considered connected (*8-connected*).
Vectorized	`OUTPUT`	[vector: polygon] Default: `[Save to temporary file]`	Specification of the output (polygon) vector layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Additional command-line parameters Optional	`EXTRA`	[string] Default: Not set	Add extra GDAL command line options

Outputs

Label	Name	Type	Description
Vectorized	`OUTPUT`	[vector: polygon]	Output vector layer

Python code

Algorithm ID: `gdal:polygonize`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Rearrange bands

Creates a new raster using selected band(s) from a given raster layer. The algorithm also makes it possible to reorder the bands for the newly-created raster.

This algorithm is derived from the GDAL translate utility.

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Input raster layer
Selected band(s)	`BANDS`	[raster band] [list] Default: Not set	Ordered list of the bands to use to create the new raster
Converted	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).
Output data type	`DATA_TYPE`	[enumeration] Default: 0	Defines the data type of the output raster file. Options: • 0 — Use Input Layer Data Type • 1 — Byte (Eight bit unsigned integer (quint8)) • 2 — Int16 (Sixteen bit signed integer (qint16)) • 3 — UInt16 (Sixteen bit unsigned integer (quint16)) • 4 — UInt32 (Thirty two bit unsigned integer (quint32)) • 5 — Int32 (Thirty two bit signed integer (qint32)) • 6 — Float32 (Thirty two bit floating point (float)) • 7 — Float64 (Sixty four bit floating point (double)) • 8 — CInt16 (Complex Int16) • 9 — CInt32 (Complex Int32) • 10 — CFloat32 (Complex Float32) • 11 — CFloat64 (Complex Float64) • 12 — Int8 (Eight bit signed integer (qint8)) Available options depend on the GDAL version built with QGIS (see *Help* ► *About* menu)

Outputs

Label	Name	Type	Description
Converted	`OUTPUT`	[raster]	Output raster layer with rearranged bands.

Python code

Algorithm ID: `gdal:rearrange_bands`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

RGB to PCT

Converts a 24 bit RGB image into a 8 bit paletted. Computes an optimal pseudo-color table for the given RGB-image using a median cut algorithm on a downsampled RGB histogram. Then it converts the image into a pseudo-colored image using the color table. This conversion utilizes Floyd-Steinberg dithering (error diffusion) to maximize output image visual quality.

If you want to classify a raster map and want to reduce the number of classes it can be helpful to downsample your image with this algorithm before.

This algorithm is derived from the GDAL rgb2pct utility.

Default menu: *Raster* ► *Conversion*

Parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Input (RGB) raster layer
Number of colors	`NCOLORS`	[numeric: integer] Default: 2	The number of colors the resulting image will contain. A value from 2-256 is possible.
RGB to PCT	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output raster. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
RGB to PCT	`OUTPUT`	[raster]	Output raster layer.

Python code

Algorithm ID: `gdal:rgbtopct`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Translate (convert format)

Converts raster data between different formats.

This algorithm is derived from the GDAL translate utility.

Default menu: *Raster* ► *Conversion*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Input raster layer
Override the projection of the output file Optional	`TARGET_CRS`	[crs]	Specify a projection for the output file
Assign a specified NoData value to output bands Optional	`NODATA`	[numeric: double] Default: Not set	Defines the value to use for NoData in the output raster
Copy all subdatasets of this file to individual output files	`COPY_SUBDATASE`	[boolean] Default: False	Create individual files for subdatasets
Converted	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output (translated) raster layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Additional creation options Optional	CREATION_OPTIONS (for QGIS <= 3.42, this was OPTIONS)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).
Additional command-line parameters Optional	EXTRA	[string] Default: Not set	Add extra GDAL command line options
Output data type	DATA_TYPE	[enumeration] Default: 0	Defines the data type of the output raster file. Options: • 0 — Use Input Layer Data Type • 1 — Byte (Eight bit unsigned integer (quint8)) • 2 — Int16 (Sixteen bit signed integer (qint16)) • 3 — UInt16 (Sixteen bit unsigned integer (quint16)) • 4 — UInt32 (Thirty two bit unsigned integer (quint32)) • 5 — Int32 (Thirty two bit signed integer (qint32)) • 6 — Float32 (Thirty two bit floating point (float)) • 7 — Float64 (Sixty four bit floating point (double)) • 8 — CInt16 (Complex Int16) • 9 — CInt32 (Complex Int32) • 10 — CFloat32 (Complex Float32) • 11 — CFloat64 (Complex Float64) • 12 — Int8 (Eight bit signed integer (qint8)) Available options depend on the GDAL version built with QGIS (see *Help* ► *About* menu)

Outputs

Label	Name	Type	Description
Converted	`OUTPUT`	[raster]	Output (translated) raster layer.

Python code

Algorithm ID: `gdal:translate`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.2.3 Raster extraction

Clip raster by extent

Clips any GDAL-supported raster file to a given extent.

This algorithm is derived from the GDAL translate utility.

Default menu: *Raster* ► *Extraction*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	The input raster

continues on next page

Table 24.321 – continued from previous page

Label	Name	Type	Description
Clipping extent	`EXTENT`	[extent]	Extent that should be used for the output raster. Only pixels within the specified bounding box will be included in the output. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Override the projection for the output file	`OVERCRS`	[boolean] Default: False	If checked, the output file is assigned the input layer CRS.
Assign a specified NoData value to output bands Optional	`NODATA`	[numeric: double] Default: Not set	Defines a value that should be inserted for the NoData values in the output raster
Clipped (extent)	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output raster layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).
Output data type	`DATA_TYPE`	[enumeration] Default: 0	Defines the format of the output raster file. Options: • 0 — Use Input Layer Data Type • 1 — Byte (Eight bit unsigned integer (quint8)) • 2 — Int16 (Sixteen bit signed integer (qint16)) • 3 — UInt16 (Sixteen bit unsigned integer (quint16)) • 4 — UInt32 (Thirty two bit unsigned integer (quint32)) • 5 — Int32 (Thirty two bit signed integer (qint32)) • 6 — Float32 (Thirty two bit floating point (float)) • 7 — Float64 (Sixty four bit floating point (double)) • 8 — CInt16 (Complex Int16) • 9 — CInt32 (Complex Int32) • 10 — CFloat32 (Complex Float32) • 11 — CFloat64 (Complex Float64) • 12 — Int8 (Eight bit signed integer (qint8)) Available options depend on the GDAL version built with QGIS (see *Help* ► *About* menu)
Additional command-line parameters Optional	`EXTRA`	[string] Default: Not set	Add extra GDAL command line options

Outputs

Label	Name	Type	Description
Clipped (extent)	`OUTPUT`	[raster]	Output raster layer clipped by the given extent

Python code

Algorithm ID: `gdal:cliprasterbyextent`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Clip raster by mask layer

Clips any GDAL-supported raster by a vector mask layer.

This algorithm is derived from the GDAL warp utility.

Default menu: *Raster* ► *Extraction*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	The input raster
Mask layer	`MASK`	[vector: polygon]	Vector mask for clipping the raster
Source CRS	`SOURCE_CRS`	[crs]	Set the coordinate reference to use for the input raster
Target CRS	`TARGET_CRS`	[crs]	Set the coordinate reference to use for the mask layer

continues on next page

Table 24.322 – continued from previous page

Label	Name	Type	Description
Target extent Optional	TARGET_EXTENT	[extent]	Extent of the output file to be created Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Assign a specified NoData value to output bands Optional	NODATA	[numeric: double] Default: Not set	Defines a value that should be inserted for the NoData values in the output raster
Create an output alpha band	ALPHA_BAND	[boolean] Default: False	Creates an alpha band for the result. The alpha band then includes the transparency values of the pixels.
Match the extent of the clipped raster to the extent of the mask layer	CROP_TO_CUTLIN	[boolean] Default: True	Applies the vector layer extent to the output raster if checked.
Keep resolution of input raster	KEEP_RESOLUTIO	[boolean] Default: False	The resolution of the output raster will not be changed
Set output file resolution	SET_RESOLUTION	[boolean] Default: False	Shall the output resolution (cell size) be specified
X Resolution to output bands Optional	X_RESOLUTION	[numeric: double] Default: Not set	The width of the cells in the output raster
Y Resolution to output band Optional	Y_RESOLUTION	[numeric: double] Default: Not set	The height of the cells in the output raster
Use multithreaded warping implementation	MULTITHREAD-ING	[boolean] Default: False	Two threads will be used to process chunks of image and perform input/output operation simultaneously. Note that computation is not multithreaded itself.
Clipped (mask)	OUTPUT	[raster] Default: `[Save to temporary file]`	Specification of the output raster layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Additional creation options Optional	CREATION_OPTIONS (for QGIS <= 3.42, this was OPTIONS)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).
Output data type	DATA_TYPE	[enumeration] Default: 0	Defines the format of the output raster file. Options: • 0 — Use Input Layer Data Type • 1 — Byte (Eight bit unsigned integer (quint8)) • 2 — Int16 (Sixteen bit signed integer (qint16)) • 3 — UInt16 (Sixteen bit unsigned integer (quint16)) • 4 — UInt32 (Thirty two bit unsigned integer (quint32)) • 5 — Int32 (Thirty two bit signed integer (qint32)) • 6 — Float32 (Thirty two bit floating point (float)) • 7 — Float64 (Sixty four bit floating point (double)) • 8 — CInt16 (Complex Int16) • 9 — CInt32 (Complex Int32) • 10 — CFloat32 (Complex Float32) • 11 — CFloat64 (Complex Float64) • 12 — Int8 (Eight bit signed integer (qint8)) Available options depend on the GDAL version built with QGIS (see *Help* ► *About* menu)
Additional command-line parameters Optional	EXTRA	[string] Default: Not set	Add extra GDAL command line options

Outputs

Label	Name	Type	Description
Clipped (mask)	`OUTPUT`	[raster]	Output raster layer clipped by the vector layer

Python code

Algorithm ID: `gdal:cliprasterbymasklayer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Contour

Extracts contour lines from any GDAL-supported elevation raster.

This algorithm is derived from the GDAL contour utility.

Default menu: *Raster* ► *Extraction*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Input raster
Band number	`BAND`	[raster band] Default: 1	Raster band to create the contours from
Interval between contour lines	`INTERVAL`	[numeric: double] Default: 10.0	Defines the interval between the contour lines in the given units of the elevation raster (minimum value 0)
Attribute name (if not set, no elevation attribute is attached) Optional	`FIELD_NAME`	[string] Default: 'ELEV'	Provides a name for the attribute in which to put the elevation.
Offset from zero relative to which to interpret intervals Optional	`OFFSET`	[numeric: double] Default: 0.0	
Contours	`OUTPUT`	[vector: line] Default: `[Save to temporary file]`	Specification of the output vector layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Produce 3D vector	`CREATE_3D`	[boolean] Default: False	Forces production of 3D vectors instead of 2D. Includes elevation at every vertex.
Treat all raster values as valid	`IGNORE_NODATA`	[boolean] Default: False	Ignores any NoData values in the dataset.
Input pixel value to treat as "No-Data" Optional	`NODATA`	[numeric: double] Default: Not set	Defines a value that should be inserted for the NoData values in the output raster
Additional command-line parameters Optional	`EXTRA`	[string] Default: Not set	Add extra GDAL command line options. Refer to the corresponding GDAL utility documentation.

Outputs

Label	Name	Type	Description
Contours	`OUTPUT`	[vector: line]	Output vector layer with contour lines

Python code

Algorithm ID: `gdal:contour`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Contour Polygons

Extracts contour polygons from any GDAL-supported elevation raster.

This algorithm is derived from the GDAL contour utility.

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Input raster
Band number	`BAND`	[raster band] Default: 1	Raster band to create the contours from

continues on next page

Table 24.324 – continued from previous page

Label	Name	Type	Description
Interval between contour lines	`INTERVAL`	[numeric: double] Default: 10.0	Defines the interval between the contour lines in the given units of the elevation raster (minimum value 0)
Offset from zero relative to which to interpret intervals Optional	`OFFSET`	[numeric: double] Default: 0.0	
Attribute name for minimum elevation of contour polygon Optional	`FIELD_NAME_MIN`	[string] Default: 'ELEV_MIN'	Provides a name for the attribute in which to put the minimum elevation of contour polygon. If not provided no minimum elevation attribute is attached.
Attribute name for maximum elevation of contour polygon Optional	`FIELD_NAME_MAX`	[string] Default: 'ELEV_MAX'	Provides a name for the attribute in which to put the maximum elevation of contour polygon. If not provided no maximum elevation attribute is attached.
Contours	`OUTPUT`	[vector: polygon] Default: `[Save to temporary file]`	Specification of the output vector layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Produce 3D vector	`CREATE_3D`	[boolean] Default: False	Forces production of 3D vectors instead of 2D. Includes elevation at every vertex.
Treat all raster values as valid	`IGNORE_NODATA`	[boolean] Default: False	Ignores any NoData values in the dataset.
Input pixel value to treat as "No-Data" Optional	`NODATA`	[numeric: double] Default: Not set	Defines a value that should be inserted for the NoData values in the output raster
Additional command-line parameters Optional	`EXTRA`	[string] Default: Not set	Add extra GDAL command line options. Refer to the corresponding GDAL utility documentation.

Outputs

Label	Name	Type	Description
Contours	`OUTPUT`	[vector: polygon]	Output vector layer with contour polygons

Python code

Algorithm ID: `gdal:contour_polygon`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.2.4 Raster miscellaneous

Build overviews (pyramids)

To speed up rendering time of raster layers overviews (pyramids) can be created. Overviews are lower resolution copies of the data which QGIS uses depending of the level of zoom.

This algorithm is derived from the GDAL addo utility.

Default menu: *Raster* ► *Miscellaneous*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Input raster layer
Remove all existing overviews	`CLEAN`	[boolean] Default: False	Removes existing overviews from the raster. By default these are not removed.

Advanced parameters

Label	Name	Type	Description
Overview levels	`LEVELS`	[string] Default: '2 4 8 16'	Defines the number of overview levels calculated by the original resolution of the input raster layer. By default 4 levels will be taken into consideration.
Resampling method Optional	`RESAMPLING`	[enumeration] Default: 0	Calculates the overviews with a defined resampling method. Possible resampling methods are: • 0 – Nearest Neighbour (`nearest`) • 1 – Average (`average`) • 2 – Gaussian (`gauss`) • 3 – Cubic (4x4 kernel) (`cubic`) • 4 – Cubic B-Spline (4x4 kernel) (`cubicspline`) • 5 – Lanczos (6x6 kernel) (`lanczos`) • 6 – Average MP (`average_mp`) • 7 – Average in Mag/Phase Space (`average_magphase`) • 8 – Mode (`mode`)
Overviews format Optional	`FORMAT`	[enumeration] Default: 0	The overviews can be stored internally, or externally as GTiff or ERDAS Imagine file. By default the overviews are stored in the output raster. Possible formats methods are: • 0 – Internal (if possible) • 1 – External (GTiff .ovr) • 2 – External (ERDAS Imagine .aux)
Additional command-line parameters Optional	`EXTRA`	[string] Default: Not set	Add extra GDAL command line options

Outputs

Label	Name	Type	Description
Pyramidized	`OUTPUT`	[raster]	Output raster layer with overviews

Python code

Algorithm ID: `gdal:overviews`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Build virtual raster

Builds a VRT (Virtual Dataset) that is a mosaic of the list of input GDAL-supported rasters. With a mosaic you can merge several raster files.

This algorithm is derived from the GDAL buildvrt utility.

Default menu: *Raster* ► *Miscellaneous*

Parameters

Basic parameters

Label	Name	Type	Description
Input layers	`INPUT`	[raster] [list]	GDAL-supported raster layers.
Resolution	`RESOLUTION`	[enumeration] Default: 0	The output resolution of the mosaic. By default the average resolution of the raster files will be chosen. Options: • 0 — Average (`average`) • 1 — Highest (`highest`) • 2 — Lowest (`lowest`)
Place each input file into a separate band	`SEPARATE`	[boolean] Default: False	With 'True' you can define that each raster file goes into a separated stacked band in the VRT band.
Allow projection difference	`PROJ_DIFFERENC`	[boolean] Default: False	Allows that the output bands have different projections derived from the projection of the input raster layers.
Virtual	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output raster layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Add alpha mask band to VRT when source raster has none	`ADD_ALPHA`	[boolean] Default: False	Adds an alpha mask band to the VRT when the source raster has none.
Override projection for the output file Optional	`ASSIGN_CRS`	[crs] Default: Not set	Overrides the projection for the output file. No reprojection is done.
Resampling algorithm	`RESAMPLING`	[enumeration] Default: 0	The resampling algorithm to use. Options: • 0 — Nearest Neighbour (`nearest`) • 1 — Bilinear (2x2 kernel) (`bilinear`) • 2 — Cubic (4x4 kernel) (`cubic`) • 3 — Cubic B-Spline (4x4 kernel) (`cubicspline`) • 4 — Lanczos (6x6 kernel) (`lanczos`) • 5 — Average (`average`) • 6 — Mode (`mode`)
NoData value(s) for input bands (space separated) Optional	`SRC_NODATA`	[string] Default: Not set	Space separated NoData value(s) for input band(s)
Additional command-line parameters	`EXTRA`	[string] Default: Not set	Add extra GDAL command line options

Outputs

Label	Name	Type	Description
Virtual	`OUTPUT`	[raster]	Output raster layer

Python code

Algorithm ID: `gdal:buildvirtualraster`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

gdal2tiles

Generates a directory with small tiles and metadata, following the OSGeo Tile Map Service Specification. See also the OpenGIS Web Map Tile Service Implementation Standard. Simple web pages with viewers based on Google Maps, OpenLayers and Leaflet are generated as well. To explore your maps on-line in the web browser, you only need to upload the generated directory onto a web server.

This algorithm also creates the necessary metadata for Google Earth (KML SuperOverlay), in case the supplied map uses `EPSG:4326` projection.

ESRI world files and embedded georeferencing is used during tile generation, but you can publish a picture without proper georeferencing too.

This algorithm is derived from the GDAL gdal2tiles utility.

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	GDAL-supported raster layer.
Tile cutting profile	`PROFILE`	[enumeration] Default: 0	One of: • 0 — Mercator (`mercator`) • 1 — Geodetic (`geodetic`) • 2 — Raster (`raster`)
Zoom levels to render Optional	`ZOOM`	[string] Default: ''	
Web viewer to generate	`VIEWER`	[enumeration] Default: 0	One of: • 0 — All (`all`) • 1 — GoogleMaps (`google`) • 2 — OpenLayers (`openlayers`) • 3 — Leaflet (`leaflet`) • 4 — None (`none`)
Title of the map Optional	`TITLE`	[string] Default: ''	
Copyright of the map	`COPYRIGHT`	[string] Default: ''	
Output directory	`OUTPUT`	[folder] Default: `[Save to temporary folder]`	Specify the output folder for the tiles. *One of*: • Save to a Temporary Directory • Save to Directory

Advanced parameters

Label	Name	Type	Description
Resampling method	`RESAMPLING`	[enumeration] Default: 0	The resampling algorithm to use. Options: • 0 — Average (`average`) • 1 — Nearest Neighbour (`near`) • 2 — Bilinear (2x2 kernel) (`bilinear`) • 3 — Cubic (4x4 kernel) (`cubic`) • 4 — Cubic B-Spline (4x4 kernel) (`cubicspline`) • 5 — Lanczos (6x6 kernel) (`lanczos`) • 6 — Antialias (`antialias`)
The spatial reference system used for the source input data Optional	`SOURCE_CRS`	[crs] Default: Not set	
Transparency value to assign to the input data Optional	`NODATA`	[numeric: double] Default: 0.0	
URL address where the generated tiles are going to be published Optional	`URL`	[string] Default: ''	
Google Maps API key (http://code.google.c Optional	`GOOGLE_KEY`	[string] Default: ''	Your Google maps API key.
Bing Maps API key (https://www.bingm Optional	`BING_KEY`	[string] Default: ''	Your Bing maps API key.
Generate only missing files	`RESUME`	[boolean] Default: False	
Generate KML for Google Earth	`KML`	[boolean] Default: False	
Avoid automatic generation of KML files for EPSG:4326	`NO_KML`	[boolean] Default: False	

Outputs

Label	Name	Type	Description
Output directory	`OUTPUT`	[folder]	The output folder (for the tiles)

Python code

Algorithm ID: `gdal:gdal2tiles`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Merge

Merges raster files in a simple way. Here you can use a pseudocolor table from an input raster and define the output raster type. All the images must be in the same coordinate system.

This algorithm is derived from the GDAL merge utility.

Default menu: *Raster* ► *Miscellaneous*

Parameters

Basic parameters

Label	Name	Type	Description
Input layers	`INPUT`	[raster] [list]	Input raster layers
Grab pseudocolor table from first layer	`PCT`	[boolean] Default: False	The pseudocolor table from the first layer will be used for the coloring
Place each input file into a separate band	`SEPARATE`	[boolean] Default: False	Place each input file into a separate band

continues on next page

Table 24.328 – continued from previous page

Label	Name	Type	Description
Output data type	`DATA_TYPE`	[enumeration] Default: 5	Defines the format of the output raster file. Options: • 0 — Byte (Eight bit unsigned integer (quint8)) • 1 — Int16 (Sixteen bit signed integer (qint16)) • 2 — UInt16 (Sixteen bit unsigned integer (quint16)) • 3 — UInt32 (Thirty two bit unsigned integer (quint32)) • 4 — Int32 (Thirty two bit signed integer (qint32)) • 5 — Float32 (Thirty two bit floating point (float)) • 6 — Float64 (Sixty four bit floating point (double)) • 7 — CInt16 (Complex Int16) • 8 — CInt32 (Complex Int32) • 9 — CFloat32 (Complex Float32) • 10 — CFloat64 (Complex Float64) • 11 — Int8 (Eight bit signed integer (qint8)) Available options depend on the GDAL version built with QGIS (see *Help* ► *About* menu)
Merged	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output raster layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Input pixel value to treat as "No-Data" Optional	`NODATA_INPUT`	[numeric: double] Default: Not set	Ignores pixels from files being merged in with this pixel value
Assign specified "NoData" value to output Optional	`NODATA_OUTPUT`	[numeric: double] Default: Not set	Assigns the specified NoData value to output bands.
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).
Additional command-line parameters	`EXTRA`	[string] Default: Not set	Add extra GDAL command line options

Outputs

Label	Name	Type	Description
Merged	`OUTPUT`	[raster]	Output raster layer

Python code

Algorithm ID: `gdal:merge`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Pansharpening

Performs a pan-sharpening operation. It can create a "classic" output dataset (such as GeoTIFF), or a VRT dataset describing the pan-sharpening operation.

See GDAL Pansharpen.

Parameters

Basic parameters

Label	Name	Type	Description
Spectral dataset	`SPECTRAL`	[raster]	Input (spectral) raster layer
Panchromatic dataset	`PANCHROMATIC`	[raster]	Input (panchromatic) raster layer
Output	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specify the output (sharpened) raster layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Resampling algorithm	`RESAMPLING`	[enumeration] Default: 2	The resampling algorithm to use. Options: • 0 — Nearest Neighbour (`nearest`) • 1 — Bilinear (2x2 kernel) (`bilinear`) • 2 — Cubic (4x4 kernel) (`cubic`) • 3 — Cubic B-Spline (4x4 kernel) (`cubicspline`) • 4 — Lanczos (6x6 kernel) (`lanczos`) • 5 — Average (`average`)
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (`|`).
Additional command-line parameters Optional	`EXTRA`	[string] Default: Not set	Add extra GDAL command line options

Outputs

Label	Name	Type	Description
Output	`OUTPUT`	[raster]	Output (sharpened) raster layer

Python code

Algorithm ID: `gdal:pansharp`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Raster calculator

Command line raster calculator with numpy syntax. Use any basic arithmetic supported by numpy arrays, such as +, -, *, and / along with logical operators, such as >. Note that all input rasters must have the same dimensions, but no projection checking is performed.

See the GDAL Raster Calculator utility docs.

See also

Raster calculator, *Raster calculator (virtual)*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer A	`INPUT_A`	[raster]	First input raster layer (mandatory)
Number of raster band for A	`BAND_A`	[raster band]	Band for input layer A (mandatory)
Input layer B Optional	`INPUT_B`	[raster] Default: Not set	Second input raster layer
Number of raster band for B Optional	`BAND_B`	[raster band]	Band for input layer B
Input layer C Optional	`INPUT_C`	[raster] Default: Not set	Third input raster layer
Number of raster band for C Optional	`BAND_C`	[raster band]	Band for input layer C
Input layer D Optional	`INPUT_D`	[raster] Default: Not set	Fourth input raster layer
Number of raster band for D Optional	`BAND_D`	[raster band]	Band for input layer D
Input layer E Optional	`INPUT_E`	[raster] Default: Not set	Fifth input raster layer
Number of raster band for E Optional	`BAND_E`	[raster band]	Band for input layer E
Input layer F Optional	`INPUT_F`	[raster]	Sixth input raster layer

continues on next page

Table 24.330 – continued from previous page

Label	Name	Type	Description
Number of raster band for F Optional	`BAND_F`	[raster band] Default: Not set	Band for input layer F
Calculation in gdalnumeric syntax using +-/* or any numpy array functions (i.e. logical_and())	`FORMULA`	[string] Default: ''	The calculation formula. Examples: • `A*(A>0)` — outputs the value of the raster A if the value of A is greater than 0. If not, outputs 0. • `A*(A>0 and A>B)` — outputs the value of A if that value is bigger than 0 and bigger than the value of B. If not, outputs 0. • `A*logical_or(A<=177, A>=185)` — outputs the value of A if A <= 177 or A >= 185. If not, outputs 0. • `sqrt(A*A+B*B)` — Outputs the square root of the sum of the value of A squared and the value of B squared.
Set output NoData value Optional	`NO_DATA`	[numeric: double] Default: Not set	Value to use for NoData
Handling of extent differences	`EXTENT_OPT`	[enumeration] Default: 0	Determines how to handle rasters with different extents. Only available with GDAL 3.3+. Supported options are: • 0 — Ignore • 1 — Fail • 2 — Union • 3 — Intersect
Output extent Optional	`INPUT`	[extent]	Custom extent of the output raster. Only available with GDAL 3.3+. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`

continues on next page

Table 24.330 – continued from previous page

Label	Name	Type	Description
Output raster type	`RTYPE`	[enumeration] Default: 5	Defines the data type of the output raster file. Options: • 0 — Byte (Eight bit unsigned integer (quint8)) • 1 — Int16 (Sixteen bit signed integer (qint16)) • 2 — UInt16 (Sixteen bit unsigned integer (quint16)) • 3 — UInt32 (Thirty two bit unsigned integer (quint32)) • 4 — Int32 (Thirty two bit signed integer (qint32)) • 5 — Float32 (Thirty two bit floating point (float)) • 6 — Float64 (Sixty four bit floating point (double)) • 7 — CInt16 (Complex Int16) • 8 — CInt32 (Complex Int32) • 9 — CFloat32 (Complex Float32) • 10 — CFloat64 (Complex Float64) • 11 — Int8 (Eight bit signed integer (qint8)) Available options depend on the GDAL version built with QGIS (see *Help* ► *About* menu)
Calculated	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specify the output (calculated) raster layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).
Additional command-line parameters Optional	`EXTRA`	[string] Default: ''	Add extra GDAL command line options

Outputs

Label	Name	Type	Description
Calculated	`OUTPUT`	[raster]	Output (calculated) raster layer

Python code

Algorithm ID: `gdal:rastercalculator`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Raster information

The gdalinfo program lists various information about a GDAL supported raster dataset.

This algorithm is derived from the GDAL info utility.

Default menu: *Raster ► Miscellaneous*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Input raster layer
Force computation of the actual min/max values for each band	`MIN_MAX`	[boolean] Default: False	Forces computation of the actual min/max values for each band in the dataset
Read and display image statistics (force computation if necessary)	`STATS`	[boolean] Default: False	Reads and displays image statistics. Forces computation if no statistics are stored in an image.
Suppress GCP info	`NO_GCP`	[boolean] Default: False	Suppresses ground control points list printing. It may be useful for datasets with huge amount of GCPs, such as L1B AVHRR or HDF4 MODIS which contain thousands of them.
Suppress metadata info	`NO_METADATA`	[boolean] Default: False	Suppresses metadata printing. Some datasets may contain a lot of metadata strings.
Layer information	`OUTPUT`	[html] Default: `[Save to temporary file]`	Specify the HTML file for output. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Additional command-line parameters Optional	`EXTRA`	[string] Default: Not set	Add extra GDAL command line options

Outputs

Label	Name	Type	Description
Layer information	`OUTPUT`	[html]	The HTML file containing information about the input raster layer

Python code

Algorithm ID: `gdal:gdalinfo`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Retile

Retiles a set of input tiles. All the input tiles must be georeferenced in the same coordinate system and have a matching number of bands. Optionally pyramid levels are generated.

This algorithm is derived from the GDAL Retile utility.

Parameters

Basic parameters

Label	Name	Type	Description
Input files	`INPUT`	[raster] [list]	The input raster files
Tile width	`TILE_SIZE_X`	[numeric: integer] Default: 256	Width of the tiles in pixels (minimum 0)
Tile height	`TILE_SIZE_Y`	[numeric: integer] Default: 256	Height of the tiles in pixels (minimum 0)
Overlap in pixels between consecutive tiles	`OVERLAP`	[numeric: integer] Default: 0	
Number of pyramid levels to build	`LEVELS`	[numeric: integer] Default: 1	Minimum: 0

continues on next page

Table 24.334 – continued from previous page

Label	Name	Type	Description
Output directory	`OUTPUT`	[folder] Default: `[Save to temporary folder]`	Specify the output folder for the tiles. *One of*: • Save to a Temporary Directory • Save to Directory
CSV file containing the tile(s) georeferencing information	`OUTPUT_CSV`	[file] Default: `[Skip output]`	Specify the output file for the tiles. *One of*: • Skip Output • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Source coordinate reference system Optional	`SOURCE_CRS`	[crs] Default: Not set	
Resampling method	`RESAMPLING`	[enumeration] Default: 0	The resampling algorithm to use. Options: • 0 — Nearest Neighbour (`nearest`) • 1 — Bilinear (2x2 kernel) (`bilinear`) • 2 — Cubic (4x4 kernel) (`cubic`) • 3 — Cubic B-Spline (4x4 kernel) (`cubicspline`) • 4 — Lanczos (6x6 kernel) (`lanczos`)
Column delimiter used in the CSV file Optional	`DELIMITER`	[string] Default: ';'	Delimiter to use in the CSV file containing the tile(s) georeferencing information
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (`|`).
Additional command-line parameters Optional	`EXTRA`	[string] Default: ''	Add extra GDAL command line options

continues on next page

Table 24.335 – continued from previous page

Label	Name	Type	Description
Output data type	`DATA_TYPE`	[enumeration] Default: 5	Defines the format of the output raster file. Options: • 0 — Byte (Eight bit unsigned integer (quint8)) • 1 — Int16 (Sixteen bit signed integer (qint16)) • 2 — UInt16 (Sixteen bit unsigned integer (quint16)) • 3 — UInt32 (Thirty two bit unsigned integer (quint32)) • 4 — Int32 (Thirty two bit signed integer (qint32)) • 5 — Float32 (Thirty two bit floating point (float)) • 6 — Float64 (Sixty four bit floating point (double)) • 7 — CInt16 (Complex Int16) • 8 — CInt32 (Complex Int32) • 9 — CFloat32 (Complex Float32) • 10 — CFloat64 (Complex Float64) • 11 — Int8 (Eight bit signed integer (qint8)) Available options depend on the GDAL version built with QGIS (see *Help* ► *About* menu)
Build only the pyramids	`ONLY_PYRAMIDS`	[boolean] Default: False	
Use separate directory for each tile row	`DIR_FOR_ROW`	[boolean] Default: False	

Outputs

Label	Name	Type	Description
Output directory	`OUTPUT`	[folder]	The output folder for the tiles.
CSV file containing the tile(s) georeferencing information	`OUTPUT_CSV`	[file]	The CSV file with georeferencing information for the tiles.

Python code

Algorithm ID: `gdal:retile`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Tile index

Builds a vector layer with a record for each input raster file, an attribute containing the filename, and a polygon geometry outlining the raster. This output is suitable for use with MapServer as a raster tileindex.

This algorithm is derived from the GDAL Tile Index utility.

Default menu: *Raster* ► *Miscellaneous*

Parameters

Basic parameters

Label	Name	Type	Description
Input files	`LAYERS`	[raster] [list]	The input raster files. Can be multiple files.
Field name to hold the file path to the indexed rasters Optional	`PATH_FIELD_NAM`	[string] Default: 'location'	The output field name to hold the file path/location to the indexed rasters.
Store absolute path to the indexed rasters	`ABSOLUTE_PATH`	[boolean] Default: False	Set whether the absolute path to the raster files is stored in the tile index file. By default the raster filenames will be put in the file exactly as they are specified in the command.
Skip files with different projection reference	`PROJ_DIFFERENC`	[boolean] Default: False	Only files with same projection as files already inserted in the tile index will be inserted. Default does not check projection and accepts all inputs.
Tile index	`OUTPUT`	[vector: polygon] Default: `[Save to temporary file]`	Specify the polygon vector layer to write the index to. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Transform geometries to the given CRS Optional	`TARGET_CRS`	[crs]	Geometries of input files will be transformed to the specified target coordinate reference system. Default creates simple rectangular polygons in the same coordinate reference system as the input rasters.
The name of the field to store the SRS of each tile Optional	`CRS_FIELD_NAME`	[string]	The name of the field to store the SRS of each tile
The format in which the CRS of each tile must be written	`CRS_FORMAT`	[enumeration] Default: 0	Format for the CRS. One of: • 0 – Auto (`AUTO`) • 1 – Well-known text (`WKT`) • 2 – EPSG (`EPSG`) • 3 – Proj.4 (`PROJ`)

Outputs

Label	Name	Type	Description
Tile index	`OUTPUT`	[vector: polygon]	The polygon vector layer with the tile index.

Python code

Algorithm ID: `gdal:tileindex`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Viewshed

Calculates a viewshed raster from an input raster DEM using method defined in Wang2000 for a user defined point.

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Input elevation raster layer
Band number	`BAND`	[raster band] Default: 1	The number of the band to use as elevation
Observer location	`OBSERVER`	[point]	The location of the observer
Observer height	`OBSERVER_HEIGHT`	[numeric: double] Default: 1.0	The altitude of the observer, in the DEM units
Target height	`TARGET_HEIGHT`	[numeric: double] Default: 1.0	The altitude of the target element, in the DEM units
Maximum distance from observer to compute visibility	`MAX_DISTANCE`	[numeric: double] Default: 100.0	Maximum distance from observer to compute visibility, in the DEM units
Output	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Output raster layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).
Additional command-line parameters	`EXTRA`	[string] Default: Not set	Add extra GDAL command line options

Outputs

Label	Name	Type	Description
Output	`OUTPUT`	[raster]	The raster layer displaying the viewshed.

Python code

Algorithm ID: `gdal:viewshed`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.2.5 Raster projections

Assign projection

Applies a coordinate system to a raster dataset.

This algorithm is derived from the GDAL edit utility.

Default menu: *Raster* ► *Projections*

Parameters

Label	Name	Type	Description
Input layer	`INPUT_LAYER`	[raster]	Input raster layer
Desired CRS	`CRS`	[crs]	The projection (CRS) of the output layer

Outputs

Label	Name	Type	Description
Layer with projection	`OUTPUT`	[raster]	The output raster layer (with the new projection information)

Python code

Algorithm ID: `gdal:assignprojection`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Extract projection

Extracts the projection of a raster file and writes it into a *world* file with extension `.wld`.

This algorithm is derived from the GDAL srsinfo utility.

Default menu: *Raster* ► *Projections*

Parameters

Label	Name	Type	Description
Input file	`INPUT_LAYER`	[raster]	Input raster The raster layer has to be file based, as the algorithm uses the path to the raster file as the location of the generated `.wld` file. Using a non-file raster layer will lead to an error.
Create also .prj file	`PRJ_FILE_CREAT`	[boolean] Default: False	If this is activated a `.prj` file containing the projection information is also created.

Outputs

Label	Name	Type	Description
World file	`WORLD_FILE`	[file]	Text file with extension `.wld` containing transformation parameters for the raster file.
ESRI Shapefile prj file	`PRJ_FILE`	[file]	Text file with `.prj` extension that describes the CRS. Will be `None` if *Create also .prj file* is False.

Python code

Algorithm ID: `gdal:extractprojection`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Warp (reproject)

Reprojects a raster layer into another Coordinate Reference System (CRS). The output file resolution and the resampling method can be chosen.

This algorithm is derived from the GDAL warp utility.

Default menu: *Raster* ► *Projections*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[raster]	Input raster layer to reproject
Source CRS Optional	`SOURCE_CRS`	[crs]	Defines the CRS of the input raster layer
Target CRS Optional	`TARGET_CRS`	[crs] Default: `EPSG:4326`	The CRS of the output layer

continues on next page

Table 24.340 – continued from previous page

Label	Name	Type	Description
Resampling method to use	`RESAMPLING`	[enumeration] Default: 0	Pixel value resampling method to use. Options: • 0 — Nearest Neighbour • 1 — Bilinear (2x2 kernel) • 2 — Cubic (4x4 kernel) • 3 — Cubic B-Spline (4x4 kernel) • 4 — Lanczos (6x6 kernel) • 5 — Average • 6 — Mode • 7 — Maximum • 8 — Minimum • 9 — Median • 10 — First quartile (Q1) • 11 — Third quartile (Q3)
NoData value for output bands Optional	`NODATA`	[numeric: double] Default: Not set	Sets NoData value for output bands. If not provided, then NoData values will be copied from the source dataset.
Output file resolution in target georeferenced units Optional	`TARGET_RESOLUTION`	[numeric: double] Default: Not set	Defines the output file resolution of reprojection result
Reprojected	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output raster layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).

continues on next page

Table 24.341 – continued from previous page

Label	Name	Type	Description
Output data type	`DATA_TYPE`	[enumeration] Default: 0	Defines the format of the output raster file. Options: • 0 — Use Input Layer Data Type • 1 — Byte (Eight bit unsigned integer (quint8)) • 2 — Int16 (Sixteen bit signed integer (qint16)) • 3 — UInt16 (Sixteen bit unsigned integer (quint16)) • 4 — UInt32 (Thirty two bit unsigned integer (quint32)) • 5 — Int32 (Thirty two bit signed integer (qint32)) • 6 — Float32 (Thirty two bit floating point (float)) • 7 — Float64 (Sixty four bit floating point (double)) • 8 — CInt16 (Complex Int16) • 9 — CInt32 (Complex Int32) • 10 — CFloat32 (Complex Float32) • 11 — CFloat64 (Complex Float64) • 12 — Int8 (Eight bit signed integer (qint8)) Available options depend on the GDAL version built with QGIS (see *Help* ► *About* menu)
Georeferenced extents of output file to be created Optional	`TARGET_EXTENT`	[extent]	Sets the georeferenced extent of the output file to be created (in the *Target CRS* by default. In the *CRS of the target raster extent*, if specified). Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
CRS of the target raster extent Optional	`TARGET_EXTENT_CRS`	[crs]	Specifies the CRS in which to interpret the coordinates given for the extent of the output file. This must not be confused with the target CRS of the output dataset. It is instead a convenience e.g. when knowing the output coordinates in a geodetic long/lat CRS, but wanting a result in a projected coordinate system.

continues on next page

Table 24.341 – continued from previous page

Label	Name	Type	Description
Use multithreaded warping implementation	`MULTITHREADING`	[boolean] Default: False	Two threads will be used to process chunks of the image and perform input/output operations simultaneously. Note that the computation itself is not multithreaded.
Additional command-line parameters Optional	`EXTRA`	[string] Default: Not set	Add extra GDAL command line options.

Outputs

Label	Name	Type	Description
Reprojected	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Reprojected output raster layer

Python code

Algorithm ID: `gdal:warpreproject`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.2.6 Vector conversion

Convert format

Converts any OGR-supported vector layer into another OGR-supported format.

This algorithm is derived from the ogr2ogr utility.

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	Input vector layer
Convert all layers from dataset	`CONVERT_ALL_LAYER`	[boolean] Default: False	Converts the whole dataset. Supported output formats for this option are `GPKG` and `GML`.
Converted	`OUTPUT`	[same as input]	Specification of the output vector layer. *One of*: • Save to a Temporary File • Save to File… For `Save to File`, the output format has to be specified. All GDAL vector formats are supported. For `Save to a Temporary File` the QGIS default vector format will be used.

Advanced parameters

Label	Name	Type	Description
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: '' (no additional options)	Additional GDAL creation options.

Outputs

Label	Name	Type	Description
Converted	`OUTPUT`	[same as input]	The output vector layer

Python code

Algorithm ID: `gdal:convertformat`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Rasterize (overwrite with attribute)

Overwrites a raster layer with values from a vector layer. New values are assigned based on the attribute value of the overlapping vector feature.

This algorithm is derived from the GDAL rasterize utility.

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Input raster layer	`INPUT_RASTER`	[raster]	Input raster layer
Field to use for a burn-in value Optional	`FIELD`	[tablefield: numeric]	Defines the attribute field to use to set the pixels values

Advanced parameters

Label	Name	Type	Description
Add burn in values to existing raster values	`ADD`	[boolean] Default: False	If False, pixels are assigned the selected field's value. If True, the selected field's value is added to the value of the input raster layer.
Additional command-line parameters Optional	`EXTRA`	[string] Default: ''	Add extra GDAL command line options

Outputs

Label	Name	Type	Description
Rasterized	`OUTPUT`	[raster]	The overwritten input raster layer

Python code

Algorithm ID: `gdal:rasterize_over`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Rasterize (overwrite with fixed value)

Overwrites parts of a raster layer with a fixed value. The pixels to overwrite are chosen based on the supplied (overlapping) vector layer.

This algorithm is derived from the GDAL rasterize utility.

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Input raster layer	`INPUT_RASTER`	[raster]	Input raster layer
A fixed value to burn	`BURN`	[numeric: double] Default: 0.0	The value to burn

Advanced parameters

Label	Name	Type	Description
Add burn in values to existing raster values	`ADD`	[boolean] Default: False	If False, pixels are assigned the fixed value. If True, the fixed value is added to the value of the input raster layer.
Additional command-line parameters Optional	`EXTRA`	[string] Default: ''	Add extra GDAL command line options

Outputs

Label	Name	Type	Description
Rasterized	`OUTPUT`	[raster]	The overwritten input raster layer

Python code

Algorithm ID: `gdal:rasterize_over_fixed_value`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Rasterize (vector to raster)

Converts vector geometries (points, lines and polygons) into a raster image.

This algorithm is derived from the GDAL rasterize utility.

Default menu: *Raster* ► *Conversion*

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Field to use for a burn-in value Optional	`FIELD`	[tablefield: numeric]	Defines the attribute field from which the attributes for the pixels should be chosen
A fixed value to burn Optional	`BURN`	[numeric: double] Default: 0.0	A fixed value to burn into a band for all features.
Burn value extracted from the "Z" values of the feature	`USE_Z`	[boolean] Default: False	Indicates that a burn value should be extracted from the "Z" values of the feature. Works with points and lines (linear interpolation along each segment). For polygons, works properly only if they are flat (same Z value for all vertices)
Output raster size units	`UNITS`	[enumeration] Default: 0	Units to use when defining the output raster size/resolution. One of: • 0 — Pixels • 1 — Georeferenced units
Width/Horizontal resolution	`WIDTH`	[numeric: double] Default: 0.0	Sets the width (if size units is "Pixels") or horizontal resolution (if size units is "Georeferenced units") of the output raster. Minimum value: 0.0.
Height/Vertical resolution	`HEIGHT`	[numeric: double] Default: 0.0	Sets the height (if size units is "Pixels") or vertical resolution (if size units is "Georeferenced units") of the output raster.

continues on next page

Table 24.344 – continued from previous page

Label	Name	Type	Description
Output extent Optional	`EXTENT`	[extent]	Extent of the output raster layer. If the extent is not specified, the minimum extent that covers the selected reference layer(s) will be used. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Assign a specified NoData value to output bands Optional	`NODATA`	[numeric: double] Default: 0.0	Assigns a specified NoData value to output bands
Rasterized	`OUTPUT`	[raster] Default: `[Save to temporary file]`	Specification of the output raster layer. *One of*: • Save to a Temporary File • Save to File… For `Save to File`, the output format has to be specified. All GDAL raster formats are supported. For `Save to a Temporary File` the QGIS default raster format will be used.

Advanced parameters

Label	Name	Type	Description
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: ''	For adding one or more creation options that control the raster to be created (colors, block size, file compression…). For convenience, you can rely on predefined profiles (see *GDAL driver options section*). Batch Process and Model Designer: separate multiple options with a pipe character (|).
Output data type	`DATA_TYPE`	[enumeration] Default: 5	Defines the format of the output raster file. Options: • 0 — Byte (Eight bit unsigned integer (quint8)) • 1 — Int16 (Sixteen bit signed integer (qint16)) • 2 — UInt16 (Sixteen bit unsigned integer (quint16)) • 3 — UInt32 (Thirty two bit unsigned integer (quint32)) • 4 — Int32 (Thirty two bit signed integer (qint32)) • 5 — Float32 (Thirty two bit floating point (float)) • 6 — Float64 (Sixty four bit floating point (double)) • 7 — CInt16 (Complex Int16) • 8 — CInt32 (Complex Int32) • 9 — CFloat32 (Complex Float32) • 10 — CFloat64 (Complex Float64) • 11 — Int8 (Eight bit signed integer (qint8)) Available options depend on the GDAL version built with QGIS (see *Help* ► *About* menu)
Pre-initialize the output image with value Optional	`INIT`	[numeric: double]	Pre-initializes the output image bands with this value. Not marked as the NoData value in the output file. The same value is used in all the bands.
Invert rasterization	`INVERT`	[boolean] Default: False	Burns the fixed burn value, or the burn value associated with the first feature into all parts of the image not inside the provided polygon.
Additional command-line parameters Optional	`EXTRA`	[string] Default: ''	Add extra GDAL command line options

Outputs

Label	Name	Type	Description
Rasterized	`OUTPUT`	[raster]	Output raster layer

Python code

Algorithm ID: `gdal:rasterize`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.2.7 Vector geoprocessing

Buffer vectors

Create buffers around the features of a vector layer.

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	The input vector layer
Geometry column name	`GEOMETRY`	[string] Default: 'geometry'	The name of the input layer geometry column to use
Buffer distance	`DISTANCE`	[numeric: double] Default: 10.0	Minimum: 0.0
Dissolve by attribute Optional	`FIELD`	[tablefield: any] Default: Not set	Field to use for dissolving
Dissolve results	`DISSOLVE`	[boolean] Default: False	If set, the result is dissolved. If no field is set for dissolving, all the buffers are dissolved into one feature.
Produce one feature for each geometry in any kind of geometry collection in the source file	`EXPLODE_COLLECTI`	[boolean] Default: False	
Buffer	`OUTPUT`	[vector: polygon] Default: `[Save to temporary file]`	Specify the output buffer layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: '' (no additional options)	Additional GDAL creation options.

Outputs

Label	Name	Type	Description
Buffer	`OUTPUT`	[vector: polygon]	The output buffer layer

Python code

Algorithm ID: `gdal:buffervectors`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Clip vector by extent

Clips any OGR-supported vector file to a given extent.

This algorithm is derived from the GDAL ogr2ogr utility.

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	The input vector layer

continues on next page

Table 24.347 – continued from previous page

Label	Name	Type	Description
Clip extent	`EXTENT`	[extent]	Defines the bounding box that should be used for the output vector file. It has to be defined in target CRS coordinates. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Clipped (extent)	`OUTPUT`	[same as input] Default: `[Save to temporary file]`	Specify the output (clipped) layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: '' (no additional options)	Additional GDAL creation options.

Outputs

Label	Name	Type	Description
Clipped (extent)	`OUTPUT`	[same as input]	The output (clipped) layer. The default format is "ESRI Shapefile".

Python code

Algorithm ID: `gdal:clipvectorbyextent`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Clip vector by mask layer

Clips any OGR-supported vector layer by a mask polygon layer.

This algorithm is derived from the GDAL ogr2ogr utility.

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	The input vector layer
Mask layer	`MASK`	[vector: polygon]	Layer to be used as clipping extent for the input vector layer.
Clipped (mask)	`OUTPUT`	[same as input] Default: `[Save to temporary file]`	The output (masked) layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: '' (no additional options)	Additional GDAL creation options.

Outputs

Label	Name	Type	Description
Clipped (mask)	`OUTPUT`	[same as input]	The output (masked) layer. The default format is "ESRI Shapefile".

Python code

Algorithm ID: `gdal:clipvectorbypolygon`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Dissolve

Dissolve (combine) geometries that have the same value for a given attribute / field. The output geometries are multipart.

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	The input layer to dissolve
Dissolve field Optional	`FIELD`	[tablefield: any]	The field of the input layer to use for dissolving
Geometry column name	`GEOMETRY`	[string] Default: 'geometry'	The name of the input layer geometry column to use for dissolving.
Dissolved	`OUTPUT`	[same as input] Default: `[Save to temporary file]`	Specify the output layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Produce one feature for each geometry in any kind of geometry collection in the source file	`EXPLODE_COLLECTI`	[boolean] Default: False	Produce one feature for each geometry in any kind of geometry collection in the source file
Keep input attributes	`KEEP_ATTRIBUTE`	[boolean] Default: False	Keep all attributes from the input layer
Count dissolved features	`COUNT_FEATURES`	[boolean] Default: False	Count the dissolved features and include it in the output layer.
Compute area and perimeter of dissolved features	`COMPUTE_AREA`	[boolean] Default: False	Compute the area and perimeter of dissolved features and include them in the output layer
Compute min/max/sum/mean for attribute	`COMPUTE_STATISTIC`	[boolean] Default: False	Calculate statistics (min, max, sum and mean) for the numeric attribute specified and include them in the output layer

continues on next page

Table 24.349 – continued from previous page

Label	Name	Type	Description
Numeric attribute to calculate statistics on Optional	STATISTICS_ATTRIBUTE	[tablefield: numeric]	The numeric attribute to calculate statistics on
Additional creation options Optional	CREATION_OPTIONS (for QGIS <= 3.42, this was OPTIONS)	[string] Default: '' (no additional options)	Additional GDAL creation options.

Outputs

Label	Name	Type	Description
Dissolved	OUTPUT	[same as input]	The output multipart geometry layer (with dissolved geometries)

Python code

Algorithm ID: `gdal:dissolve`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Offset curve

Offsets lines by a specified distance. Positive distances will offset lines to the left, and negative distances will offset them to the right.

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	INPUT	[vector: line]	The input line layer
Geometry column name	GEOMETRY	[string] Default: 'geometry'	The name of the input layer geometry column to use
Offset distance (left-sided: positive, right-sided: negative)	DISTANCE	[numeric: double] Default: 10.0	
Offset curve	OUTPUT	[vector: line] Default: [Save to temporary file]	Specify the output line layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: '' (no additional options)	Additional GDAL creation options.

Outputs

Label	Name	Type	Description
Offset curve	`OUTPUT`	[vector: line]	The output offset curve layer

Python code

Algorithm ID: `gdal:offsetcurve`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

One side buffer

Creates a buffer on one side (right or left) of the lines in a line vector layer.

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line]	The input line layer
Geometry column name	`GEOMETRY`	[string] Default: 'geometry'	The name of the input layer geometry column to use
Buffer distance	`DISTANCE`	[numeric: double] Default: 10.0	
Buffer side	`BUFFER_SIDE`	[enumeration] Default: 0	One of: • 0 — Right • 1 — Left
Dissolve by attribute Optional	`FIELD`	[tablefield: any] Default: Not set	Field to use for dissolving

continues on next page

Table 24.351 – continued from previous page

Label	Name	Type	Description
Dissolve all results	`DISSOLVE`	[boolean] Default: False	**If set, the result is dissolved.** If no field is set for dissolving, all the buffers are dissolved into one feature.
Produce one feature for each geometry in any kind of geometry collection in the source file	`EXPLODE_COLLECTI`	[boolean] Default: False	
One-sided buffer	`OUTPUT`	[vector: polygon] Default: `[Save to temporary file]`	Specify the output buffer layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: '' (no additional options)	Additional GDAL creation options.

Outputs

Label	Name	Type	Description
One-sided buffer	`OUTPUT`	[vector: polygon]	The output buffer layer

Python code

Algorithm ID: `gdal:onesidebuffer`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Points along lines

Generates a point on each line of a line vector layer at a distance from start. The distance is provided as a fraction of the line length.

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: line]	The input line layer
Geometry column name	`GEOMETRY`	[string] Default: 'geometry'	The name of the input layer geometry column to use
Distance from line start represented as a fraction of line length	`DISTANCE`	[numeric: double] Default: 0.5	
Points along lines	`OUTPUT`	[vector: point] Default: `[Save to temporary file]`	Specify the output point layer. *One of*: • Save to a Temporary File • Save to File…

Advanced parameters

Label	Name	Type	Description
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: '' (no additional options)	Additional GDAL creation options.

Outputs

Label	Name	Type	Description
Points along line	`OUTPUT`	[vector: point]	The output point layer

Python code

Algorithm ID: `gdal:pointsalonglines`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

24.2.8 Vector miscellaneous

Build virtual vector

Creates a virtual vector layer that contains a set of vector layers.

This algorithm is especially useful in case another algorithm needs multiple layers but accept only one `vrt` in which the layers are specified.

Parameters

Label	Name	Type	Description
Input datasources	`INPUT`	[vector: any] [list]	Select the vector layers you want to use to build the virtual vector
Create "unioned" VRT	`UNIONED`	[boolean] Default: False	Check if you want to unite all the vectors in a single `vrt` file
Virtual vector	`OUTPUT`	[same as input] Default: `[Save to temporary file]`	Specify the output layer containing only the duplicates. *One of*: • Save to a Temporary File • Save to File…

Outputs

Label	Name	Type	Description
Virtual vector	`OUTPUT`	[vector: any]	The output virtual vector made from the chosen sources

Python code

Algorithm ID: `gdal:buildvirtualvector`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Execute SQL

Runs a simple or complex query with SQL syntax on the source layer. The result of the query will be added as a new layer.

This algorithm is derived from the GDAL ogr2ogr utility.

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: any]	OGR-supported input vector layer
SQL expression	`SQL`	[string]	Defines the SQL query, for example `SELECT * FROM my_table WHERE name is not null`.
SQL dialect	`DIALECT`	[enumeration] Default: 0	SQL dialect to use. One of: • 0 — None • 1 — OGR SQL • 2 — SQLite
SQL result	`OUTPUT`	[vector: any]	Specification of the output layer. *One of*: • Save to a Temporary File • Save to File… For `Save to File`, the output format has to be specified. All GDAL vector formats are supported. For `Save to a Temporary File` the default output vector layer format will be used.

Advanced parameters

Label	Name	Type	Description
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: '' (no additional options)	Additional GDAL creation options.

Outputs

Label	Name	Type	Description
SQL result	`OUTPUT`	[vector: any]	Vector layer created by the query

Python code

Algorithm ID: `gdal:executesql`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Export to PostgreSQL (available connections)

Imports vector layers inside a PostgreSQL database on the basis of an available connection. The connection has to *be defined properly* beforehand. Be aware that the checkboxes 'Save Username' and 'Save Password' are activated. Then you can use the algorithm.

This algorithm is derived from the GDAL ogr2ogr utility.

Parameters

Label	Name	Type	Description
Database (connection name)	`DATABASE`	[string]	The PostgreSQL database to connect to
Input layer	`INPUT`	[vector: any]	OGR-supported vector layer to export to the database
Shape encoding Optional	`SHAPE_ENCODING`	[string] Default: ''	Sets the encoding to apply to the data
Output geometry type	`GTYPE`	[enumeration] Default: 0	Defines the output geometry type. One of: • 0 — • 1 — NONE • 2 — GEOMETRY • 3 — POINT • 4 — LINESTRING • 5 — POLYGON • 6 — GEOMETRYCOLLECTION • 7 — MULTIPOINT • 8 — MULTIPOLYGON • 9 — MULTILINESTRING • 10 — CIRCULARSTRING • 11 — COUMPOUNDCURVE • 12 — CURVEPOLYGON • 13 — MULTICURVE • 14 — MULTISURFACE • 15 — CONVERT_TO_LINEAR • 16 — CONVERT_TO_CURVE
Assign an output CRS Optional	`A_SRS`	[crs] Default: Not set	Defines the output CRS of the database table
Reproject to this CRS on output Optional	`T_SRS`	[crs] Default: Not set	Reprojects/transforms to this CRS on output
Override source CRS Optional	`S_SRS`	[crs] Default: Not set	Overrides the input layer CRS
Schema (schema name) Optional	`SCHEMA`	[string] Default: 'public'	Defines the schema for the database table
Table to export to (leave blank to use layer name) Optional	`TABLE`	[string] Default: ''	Defines a name for the table that will be imported into the database. By default the table name is the name of the input vector file.
Primary Key (new field) Optional	`PK`	[string] Default: 'id'	Defines which attribute field will be the primary key of the database table

continues on next page

Table 24.354 – continued from previous page

Label	Name	Type	Description
Primary Key (existing field, used if the above option is left empty) Optional	`PRIMARY_KEY`	[tablefield: any] Default: Not set	Defines which attribute field in the exported layer will be the primary key of the database table
Geometry column name Optional	`GEOCOLUMN`	[string] Default: 'geom'	Defines in which attribute field of the database there will be the geometry information
Vector dimensions Optional	`DIM`	[enumeration] Default: 0 (2D)	Defines the coordinate dimensions of the imported vector data. One of: • 0 — 2 (`XY`) • 1 — 3 (`XYZ`) • 2 — 4 (`XYZM`)
Distance tolerance for simplification Optional	`SIMPLIFY`	[string] Default: ''	Defines a distance tolerance for the simplification of the vector geometries to be imported. By default there is no simplification.
Maximum distance between 2 nodes (densification) Optional	`SEGMENTIZE`	[string] Default: ''	The maximum distance between two nodes. Used to create intermediate points. By default there is no densification.
Select features by extent (defined in input layer CRS) Optional	`SPAT`	[extent] Default: Not set	You can select features from a given extent that will be in the output table. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Clip the input layer using the above (rectangle) extent	`CLIP`	[boolean] Default: False	The input layer will be clipped by the extent you defined before
Select features using a SQL "WHERE" statement (Ex: column="value") Optional	`WHERE`	[string] Default: ''	Defines with a SQL "WHERE" statement which features should be selected from the input layer
Group N features per transaction (Default: 2000) Optional	`GT`	[string] Default: ''	You can group the input features in transactions where N defines the size. By default N limits the transaction size to 20000 features.

continues on next page

Table 24.354 – continued from previous page

Label	Name	Type	Description
Overwrite existing table	`OVERWRITE`	[boolean] Default: True	If there is a table with the same name in the database, and if this option is set to True, the table will be overwritten.
Append to existing table	`APPEND`	[boolean] Default: False	If checked / True the vector data will be appended to an existing table. New fields found in the input layer are ignored. By default a new table will be created.
Append and add new fields to existing table	`ADDFIELDS`	[boolean] Default: False	If activated the vector data will be appended to an existing table, there won't be a new table created. New fields found in input layer are added to the table. By default a new table will be created.
Do not launder columns/table names	`LAUNDER`	[boolean] Default: False	With this option checked you can prevent the default behaviour (converting column names to lowercase, removing spaces and other invalid characters).
Do not create Spatial Index	`INDEX`	[boolean] Default: False	Prevents a spatial index for the output table from being created. By default, a spatial index is added.
Continue after a failure, skipping the failed feature	`SKIPFAILURES`	[boolean] Default: False	Continue after a failure, skipping the failed feature.
Validate geometries based on Simple Features specification	`MAKEVALID`	[boolean] Default: False	Applies cleanup operations on geometries to ensure they are valid regarding the rules of the Simple Features specification.
Promote to Multipart	`PROMOTETOMULTI`	[boolean] Default: True	Casts features geometry type to multipart in the output table
Keep width and precision of input attributes	`PRECISION`	[boolean] Default: True	Avoids modifying column attributes to comply with input data
Additional creation options Optional	`CREATION_OPTIONS` (for QGIS <= 3.42, this was `OPTIONS`)	[string] Default: '' (no additional options)	Additional GDAL creation options.

Outputs

This algorithm has no output.

Python code

Algorithm ID: `gdal:importvectorintopostgisdatabaseavailableconnections`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Export to PostgreSQL (new connection)

Imports vector layers inside a PostgreSQL database. A new connection to the PostgreSQL database must be created.

This algorithm is derived from the GDAL ogr2ogr utility.

Parameters

Label	Name	Type	Description
Input layer	INPUT	[vector: any]	OGR-supported vector layer to export to the database
Shape encoding Optional	SHAPE_ENCODING	[string] Default: ''	Sets the encoding to apply to the data
Output geometry type	GTYPE	[enumeration] Default: 0	Defines the output geometry type. One of: • 0 — • 1 — NONE • 2 — GEOMETRY • 3 — POINT • 4 — LINESTRING • 5 — POLYGON • 6 — GEOMETRYCOLLECTION • 7 — MULTIPOINT • 8 — MULTIPOLYGON • 9 — MULTILINESTRING • 10 — CIRCULARSTRING • 11 — COUMPOUNDCURVE • 12 — CURVEPOLYGON • 13 — MULTICURVE • 14 — MULTISURFACE • 15 — CONVERT_TO_LINEAR • 16 — CONVERT_TO_CURVE
Assign an output CRS Optional	A_SRS	[crs] Default: Not set	Defines the output CRS of the database table
Reproject to this CRS on output Optional	T_SRS	[crs] Default: Not set	Reprojects/transforms to this CRS on output
Override source CRS Optional	S_SRS	[crs] Default: Not set	Overrides the input layer CRS
Host Optional	HOST	[string] Default: 'localhost'	Name of the database host
Port Optional	PORT	[string] Default: '5432'	Port number the PostgreSQL database server listens on
Username Optional	USER	[string] Default: ''	User name used to log in to the database
Database name Optional	DBNAME	[string] Default: ''	Name of the database
Password Optional	PASSWORD	[string] Default: ''	Password used with Username to connect to the database
Schema (schema name) Optional	SCHEMA	[string] Default: 'public'	Defines the schema for the database table

continues on next page

Table 24.355 – continued from previous page

Label	Name	Type	Description
Table name, leave blank to use input name Optional	`TABLE`	[string] Default: ''	Defines a name for the table that will be imported into the database. By default the table name is the name of the input vector file.
Primary Key (new field) Optional	`PK`	[string] Default: 'id'	Defines which attribute field will be the primary key of the database table
Primary Key (existing field, used if the above option is left empty) Optional	`PRIMARY_KEY`	[tablefield: any] Default: Not set	Defines which attribute field in the exported layer will be the primary key of the database table
Geometry column name Optional	`GEOCOLUMN`	[string] Default: 'geom'	Defines in which attribute field to store the geometry information
Vector dimensions Optional	`DIM`	[enumeration] Default: 0 (2D)	Defines the coordinate dimensions of the imported vector data. One of: • 0 — 2 (`XY`) • 1 — 3 (`XYZ`) • 2 — 4 (`XYZM`)
Distance tolerance for simplification Optional	`SIMPLIFY`	[string] Default: ''	Defines a distance tolerance for the simplification of the vector geometries to be imported. By default there is no simplification.
Maximum distance between 2 nodes (densification) Optional	`SEGMENTIZE`	[string] Default: ''	The maximum distance between two nodes. Used to create intermediate points. By default there is no densification.
Select features by extent (defined in input layer CRS) Optional	`SPAT`	[extent] Default: Not set	You can select features from a given extent that will be in the output table. Available methods are: • Calculate from layer…: uses extent of a layer loaded in the current project • Calculate from layout map…: uses extent of a *layout map item* in the active project • Calculate from bookmark…: uses extent of a saved *bookmark* • Use map canvas extent • Draw on canvas: click and drag a rectangle delimiting the area to take into account • Enter the coordinates as `xmin, xmax, ymin, ymax`
Clip the input layer using the above (rectangle) extent	`CLIP`	[boolean] Default: False	The input layer will be clipped by the extent you defined before
Fields to include (leave empty to use all fields) Optional	`FIELDS`	[string] [list] Default: []	Defines fields to keep from the imported vector file. If none is selected, all the fields are imported.

continues on next page

Table 24.355 – continued from previous page

Label	Name	Type	Description
Select features using a SQL "WHERE" statement (Ex: column="value") Optional	WHERE	[string] Default: ''	Defines with a SQL "WHERE" statement which features should be selected for the output table
Group N features per transaction (Default: 2000) Optional	GT	[string] Default: ''	You can group the input features in transactions where N defines the size. By default N limits the transaction size to 20000 features.
Overwrite existing table	OVERWRITE	[boolean] Default: True	If there is a table with the same name in the database, and if this option is set to True, the table will be overwritten.
Append to existing table	APPEND	[boolean] Default: False	If checked / True the vector data will be appended to an existing table. New fields found in the input layer are ignored. By default a new table will be created.
Append and add new fields to existing table	ADDFIELDS	[boolean] Default: False	If activated the vector data will be appended to an existing table, there won't be created a new table. New fields found in input layer are added to the table. By default a new table will be created.
Do not launder columns/table names	LAUNDER	[boolean] Default: False	With this option checked you can prevent the default behaviour (converting column names to lowercase, removing spaces and other invalid characters).
Do not create Spatial Index	INDEX	[boolean] Default: False	Prevents a spatial index for the output table from being created. By default, a spatial index is added.
Continue after a failure, skipping the failed feature	SKIPFAILURES	[boolean] Default: False	Continue after a failure, skipping the failed feature.
Validate geometries based on Simple Features specification	MAKEVALID	[boolean] Default: False	Applies cleanup operations on geometries to ensure they are valid regarding the rules of the Simple Features specification.
Promote to Multipart	PROMOTETOMULTI	[boolean] Default: True	Casts features geometry type to multipart in the output table
Keep width and precision of input attributes	PRECISION	[boolean] Default: True	Avoids modifying column attributes to comply with input data
Additional creation options Optional	CREATION_OPTIONS (for QGIS <= 3.42, this was OPTIONS)	[string] Default: '' (no additional options)	Additional GDAL creation options.

Outputs

This algorithm has no output.

Python code

Algorithm ID: `gdal:importvectorintopostgisdatabasenewconnection`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Vector Information

Creates an information file that lists information about an OGR-supported data source. The output will be shown in a 'Result' window and can be written into a HTML-file. The information includes the geometry type, feature count, the spatial extent, the projection information, the list of attributes and related type, list of relations and field domains, and many more.

This algorithm is derived from the GDAL ogrinfo utility.

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Enable listing of all layers in the dataset	`ALL_LAYERS`	[boolean] Default: False	If checked, QGIS will output the information of all the layers in the input dataset instead of information of the first layer only. Convenient when a multi layer format (GeoPackage, GML, KML, SpatiaLite, SQLite, …) is used as input.
Summary output only	`SUMMARY_ONLY`	[boolean] Default: True	Suppress listing of individual features and show only summary information like projection, schema, feature count and extents.
Suppress metadata info	`NO_METADATA`	[boolean] Default: False	Suppress metadata printing. Some datasets may contain a lot of metadata strings.
Layer information	`OUTPUT`	[html] Default: `[Save to temporary file]`	Specify the output HTML file that includes the file information. *One of*: • Save to a Temporary File • Save to File… If no HTML-file is defined the output will be written to a temporary file

Advanced parameters

Label	Name	Type	Description
Additional command line parameters Optional	`EXTRA`	[string] Default: '' (no additional options)	Add extra GDAL command line options

Outputs

Label	Name	Type	Description
Layer information	`OUTPUT`	[html]	The output HTML-file that includes the file information.

Python code

Algorithm ID: `gdal:ogrinfo`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

Vector Information (JSON)

Creates an information file that lists information about an OGR-supported data source. The output will be shown in a 'Result' window and can be written into a `.JSON` file. The information includes the geometry type, feature count, the spatial extent, the projection information, the list of attributes and related type, list of relations and field domains, and many more.

This algorithm is derived from the GDAL ogrinfo utility.

Requires version of GDAL >= 3.7

Parameters

Basic parameters

Label	Name	Type	Description
Input layer	`INPUT`	[vector: geometry]	Input vector layer
Enable listing of all layers in the dataset	`ALL_LAYERS`	[boolean] Default: False	If checked, QGIS will output the information of all the layers in the input dataset instead of information of the first layer only. Convenient when a multi layer format (GeoPackage, GML, KML, SpatiaLite, SQLite, …) is used as input.

continues on next page

Table 24.357 – continued from previous page

Label	Name	Type	Description
Enable listing of features	`FEATURES`	[boolean] Default: False	
Suppress metadata info	`NO_METADATA`	[boolean] Default: False	Suppress metadata printing. Some datasets may contain a lot of metadata strings.
Layer information	`OUTPUT`	[file] Default: `[Save to temporary file]`	Specify the output JSON file that includes the file information. *One of*: • Save to a Temporary File • Save to File… If no JSON file is defined the output will be written to a temporary file

Advanced parameters

Label	Name	Type	Description
Additional command line parameters Optional	`EXTRA`	[string] Default: '' (no additional options)	Add extra GDAL command line options

Outputs

Label	Name	Type	Description
Layer information	`OUTPUT`	[file]	The output *.JSON* file that includes the file information.

Python code

Algorithm ID: `gdal:ogrinfojson`

```
import processing
processing.run("algorithm_id", {parameter_dictionary})
```

The *algorithm id* is displayed when you hover over the algorithm in the Processing Toolbox. The *parameter dictionary* provides the parameter NAMEs and values. See *Using processing algorithms from the console* for details on how to run processing algorithms from the Python console.

CHAPTER

TWENTYFIVE

PLUGINS

25.1 QGIS Plugins

QGIS has been designed with a plugin architecture. This allows many new features and functions to be easily added to the application. Some of the features in QGIS are actually implemented as plugins.

25.1.1 Core and External plugins

QGIS plugins are implemented either as **Core Plugins** or **External Plugins**.

Core Plugins are maintained by the QGIS Development Team and are automatically part of every QGIS distribution. They are written in one of two languages: **C++** or **Python**.

Most of External Plugins are currently written in Python. They are stored either in the 'Official' QGIS Repository at https://plugins.qgis.org/plugins/ or in external repositories and are maintained by the individual authors. Detailed documentation about the usage, minimum QGIS version, home page, authors, and other important information are provided for the plugins in the Official repository. For other external repositories, documentation might be available with the external plugins themselves. External plugins documentation is not included in this manual.

To install or activate a plugin, go to *Plugins* menu and select  *Manage and install plugins...*. Installed external python plugins are placed under the `python/plugins` folder of the active *user profile* path.

Paths to Custom C++ plugins libraries can also be added under *Settings* ► *Options* ► *System*.

25.1.2 The Plugins Dialog

The Settings tab

At the bottom of the left panel, the  *Settings* tab is the main place you can configure which plugins can be displayed in your application. You can use the following options:

-  *Check for Updates on Startup*. QGIS will automatically check for plugin updates at startup, at most once every 3 days. Whenever an installed plugin has an update available, QGIS will inform you.
-  *Show also Experimental Plugins*. QGIS will show you plugins in early stages of development, which are generally unsuitable for production use. For these plugins, you can install either the stable or the experimental version, and at any moment switch from one to the other.
-  *Show also Deprecated Plugins*. These plugins are usually unmaintained because they have replacement functions in QGIS, a lack of maintainers, they rely on functions that are no longer available in QGIS... They are generally unsuitable for production use and appear grayed in the plugins list.

By default, in the *Plugin Repositories* section, QGIS provides you with its official plugin repository with the URL `https://plugins.qgis.org/plugins/plugins.xml?qgis=version` (where `<version>` represents the exact QGIS version you are running). To add external author repositories, click *Add...* and fill in the *Repository Details* form with a name and the URL. The URL can be of `http://` or `file://` protocol type.

The default QGIS repository is an open repository and you don't need any authentication to access it. You can however deploy your own plugin repository and require an authentication (basic authentication, PKI). You can get more information on QGIS authentication support in *Authentication* chapter.

If you do not want one or more of the added repositories, they can be disabled from the Settings tab via the *Edit...* button, or completely removed with the *Delete* button.

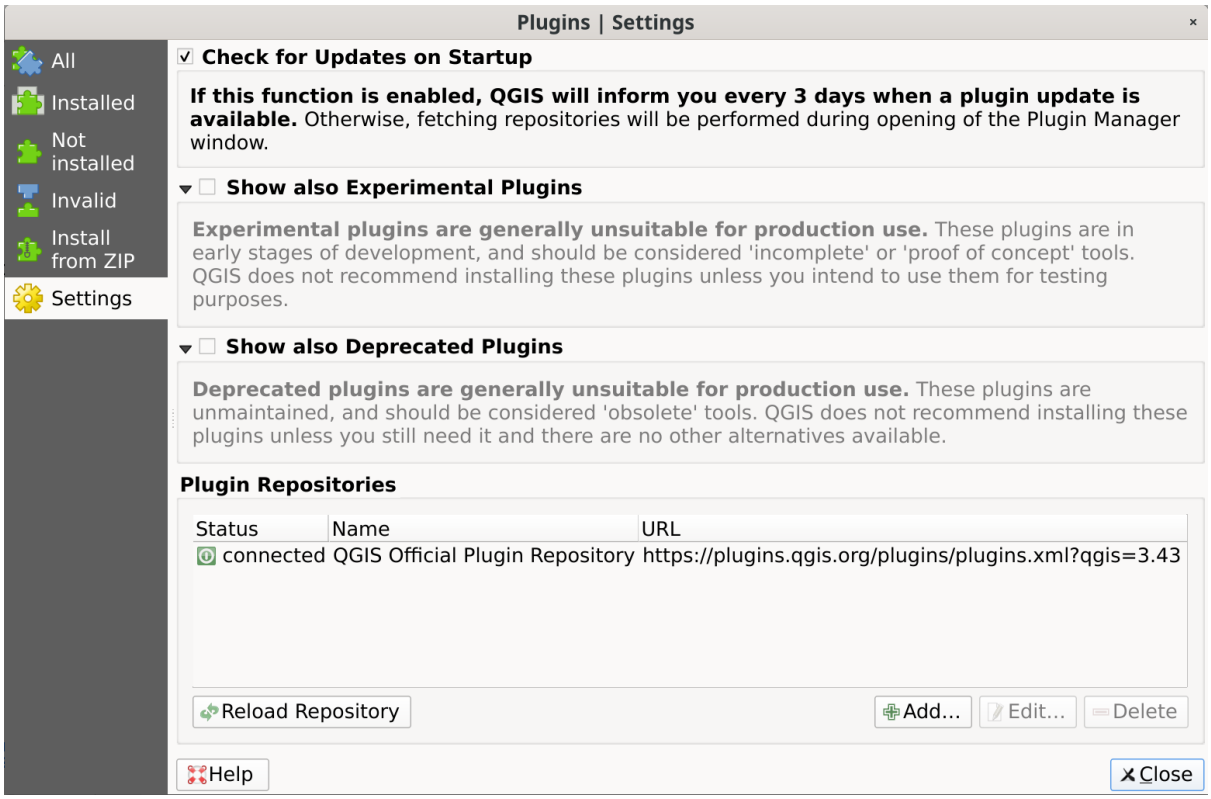


Fig. 25.1: The *Settings* tab

Browsing the plugins

The tabs

The upper tabs in the *Plugins* dialog provide you with lists of plugins based on their install, creation or update status. Depending on the plugins settings, available tabs can be:

- *All*: shows all the available plugins in the enabled repositories
- *Installed*: shows both the plugins you installed and the core plugins that are installed by default and you can not uninstall
- *Not installed*: shows uninstalled or not yet installed plugins in the enabled repositories
- *New*: shows plugins released since the last *Check for Updates on Startup*

- *Upgradeable*: shows installed plugins that have published a more recent version in the repository
- *Invalid*: shows all installed plugins that are currently broken for any reason (missing dependency, errors while loading, incompatible functions with QGIS version…)

At the top of the tabs, a *Search* function helps you find any plugin using metadata information (author, name, description, tag,…).

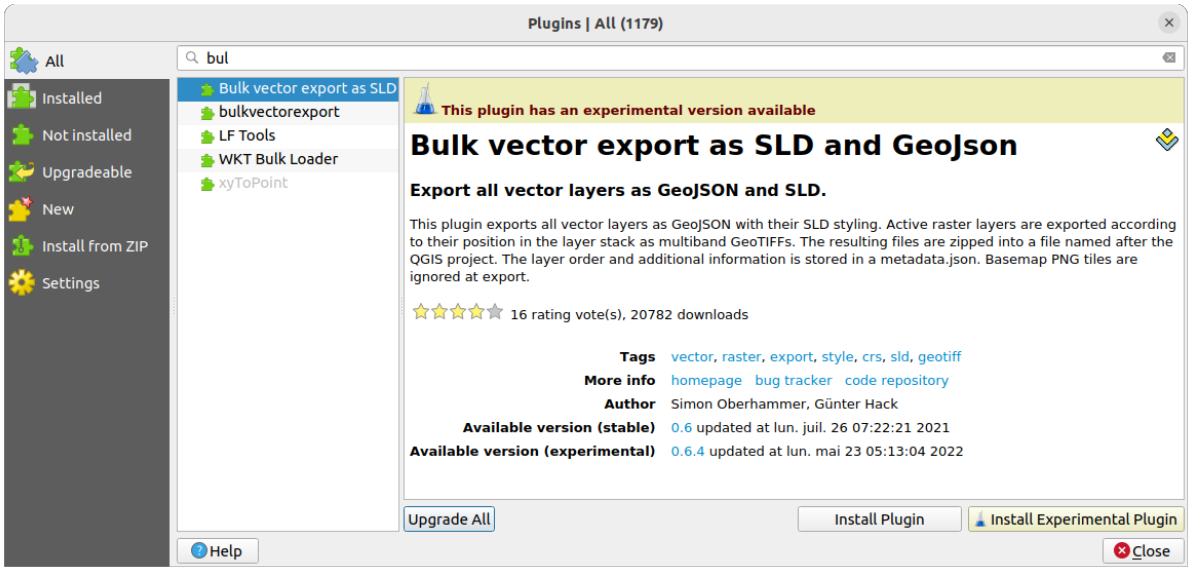


Fig. 25.2: Searching a plugin from the *All* tab

The Plugins

Select a plugin and you will have some metadata displayed in the right panel:

- information on whether the plugin is experimental or has an experimental version available (if *Show also Experimental Plugins* is checked)
- summary and description
- rating vote(s) (you can vote for your preferred plugin!)
- tags
- some useful links to the home page, tracker and code repository
- author(s)
- version(s) available with link to download page in the repository, or path to local folder for installed plugins

The *Plugin Manager* dialog allows you to interact with the latest version of the plugins. When enabled, the experimental version can be shown only if it is more recent than the latest stable version. Depending on the active tab, whether the selected plugin is installed, you will be given some of the following options:

- *Install*: installs the latest stable version of the selected plugin
- *Install Experimental Plugin*: installs the experimental version of the selected plugin
- *Reinstall Plugin*: installs the same stable version of the plugin e.g. after it has failed to load
- *Reinstall Experimental Plugin*: installs the same experimental version of the plugin e.g. after it has failed to load
- *Upgrade Plugin*: upgrades selected plugin to its latest stable version
- *Upgrade Experimental Plugin*: upgrades selected plugin to its experimental version

- *Upgrade All*: upgrades all installed plugins to their more recent stable or experimental version (depending on whether their previously installed version was stable or experimental).
- *Downgrade Plugin*: moves from the experimental version of the plugin to its previous stable version
- *Downgrade Experimental Plugin*: moves from an experimental version of the plugin to its latest published experimental version. This may occur when playing with a not yet published version.
- *Uninstall Plugin*: removes the installed plugin from the user profile

An installed plugin displays a checkbox on its left. Uncheck it to temporarily deactivate the plugin.

Right-click on a plugin in the list and you will be able to sort the plugins list by various metadata. The new order applies to all the tabs. Sort options are:

- *Sort by Name*
- *Sort by Downloads*
- *Sort by Vote*
- *Sort by Status*
- *Sort by Date Created*
- *Sort by Date Updated*

The Install from ZIP tab

The *Install from ZIP* tab provides a file selector widget to import plugins in a zipped format, e.g. plugins downloaded directly from their repository. Encrypted files are supported.

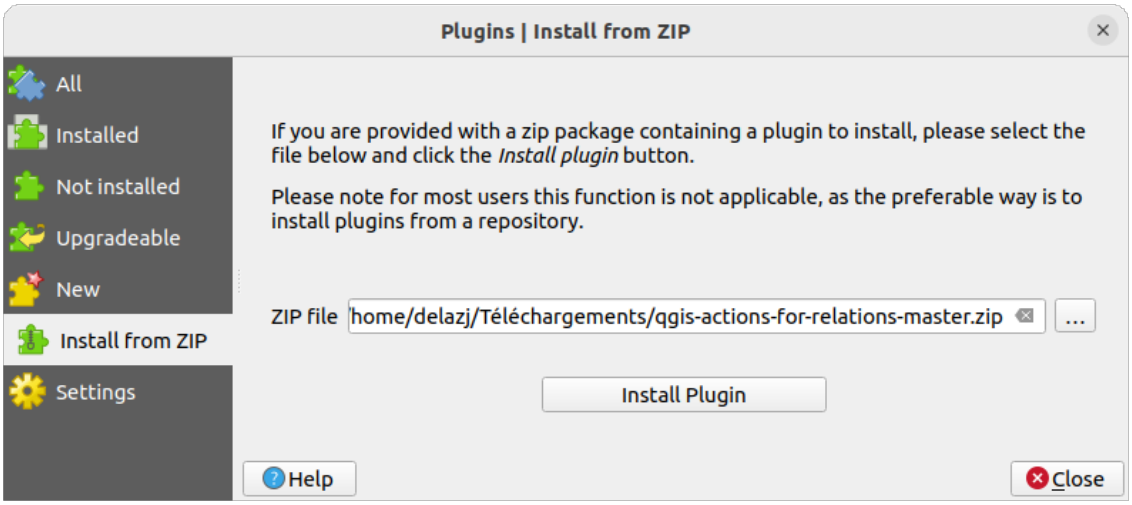


Fig. 25.3: The *Install from zip* tab

25.2 Using QGIS Core Plugins

25.2.1 DB Manager Plugin

The DB Manager Plugin is intended to be the main tool to integrate and manage spatial database formats supported by QGIS (PostgreSQL, SpatiaLite, GeoPackage, Oracle Spatial, Virtual layers) in one user interface. The DB Manager Plugin provides several features. You can drag layers from the QGIS Browser into the DB Manager, and it will import your layer into your spatial database. You can drag and drop tables between spatial databases and they will get imported.

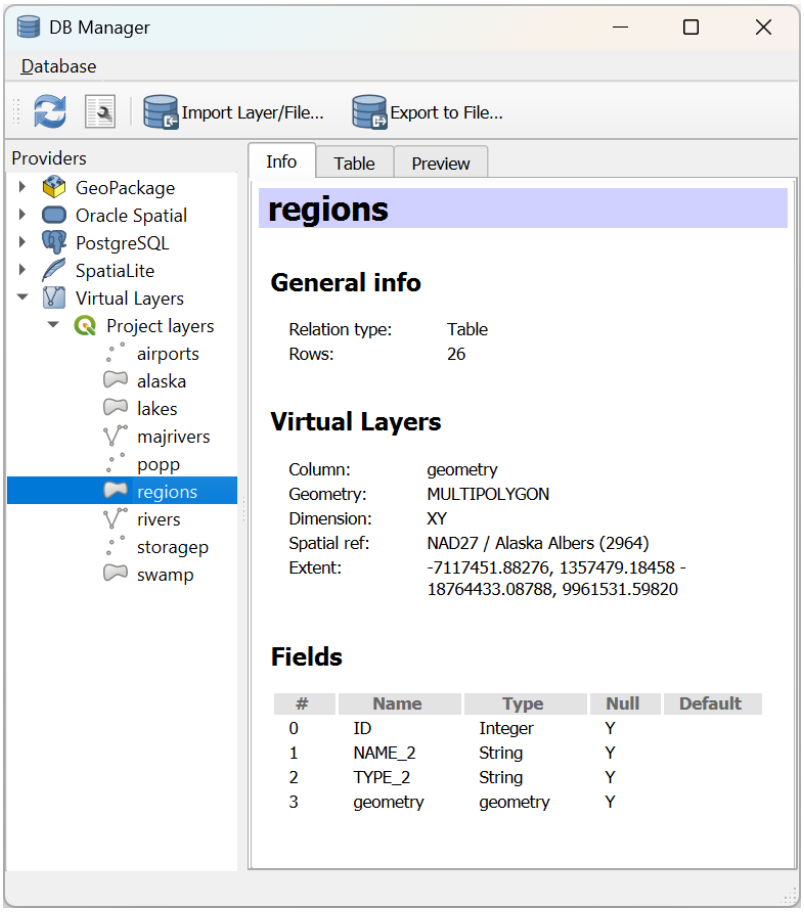


Fig. 25.4: DB Manager dialog

The *Database* menu allows you to connect to an existing database, to start the SQL window and to exit the DB Manager Plugin. Once you are connected to an existing database, the menus *Schema* (relevant for DBMSs, such as PostgreSQL) and *Table* will appear.

The *Schema* menu includes tools to create and delete (only if empty) schemas and, if topology is available (e.g. with PostGIS topology), to start a *TopoViewer*.

The *Table* menu allows you to create and edit tables and to delete tables and views. It is also possible to empty tables and to move tables between schemas. You can *Run Vacuum Analyze* for the selected table. *Vacuum* reclaims space and makes it available for reuse, and *analyze* updates statistics that is used to determine the most efficient way to execute a query. *Change Logging…* allows you to add change logging support to a table. Finally, you can *Import Layer/File…* and *Export to File…*.

> **Note**
>
> Using the DB Manager it is possible to add comments for tables and columns of a PostgreSQL Database.

The *Providers* window lists all existing databases supported by QGIS. With a double-click, you can connect to the database. With the right mouse button, you can rename and delete existing schemas and tables. Tables can also be added to the QGIS canvas with the context menu.

If connected to a database, the **main** window of the DB Manager offers four tabs. The *Info* tab provides information about the table and its geometry, as well as about existing fields, constraints and indexes. It allows you to create a spatial index on a the selected table. The *Table* tab shows the table, and the *Preview* tab renders the geometries as preview. When you open an *SQL Window*, it will be placed in a new tab.

Working with the SQL Window

You can use the DB Manager to execute SQL queries against your spatial database. Queries can be saved and loaded, and there the *SQL Query Builder* will help you formulate your queries. You can even view spatial output by checking *Load as new layer* and specifying *Column(s) with unique values* (IDs), *Geometry column* and *Layer name (prefix)*. It is possible to highlight a portion of the SQL to only execute that portion when pressing `Ctrl+R` or clicking the *Execute* button.

After executing your query, you can select specific cells in the result set. Use the `Ctrl+C` shortcut to copy the selected cells to the clipboard. The copied data is available as a formatted table. This allows you to paste the data into other applications, such as spreadsheet where it will show up as a table.

The *Query History* button stores the last 20 queries of each database and provider.

Double clicking on an entry will add the string to the SQL window.

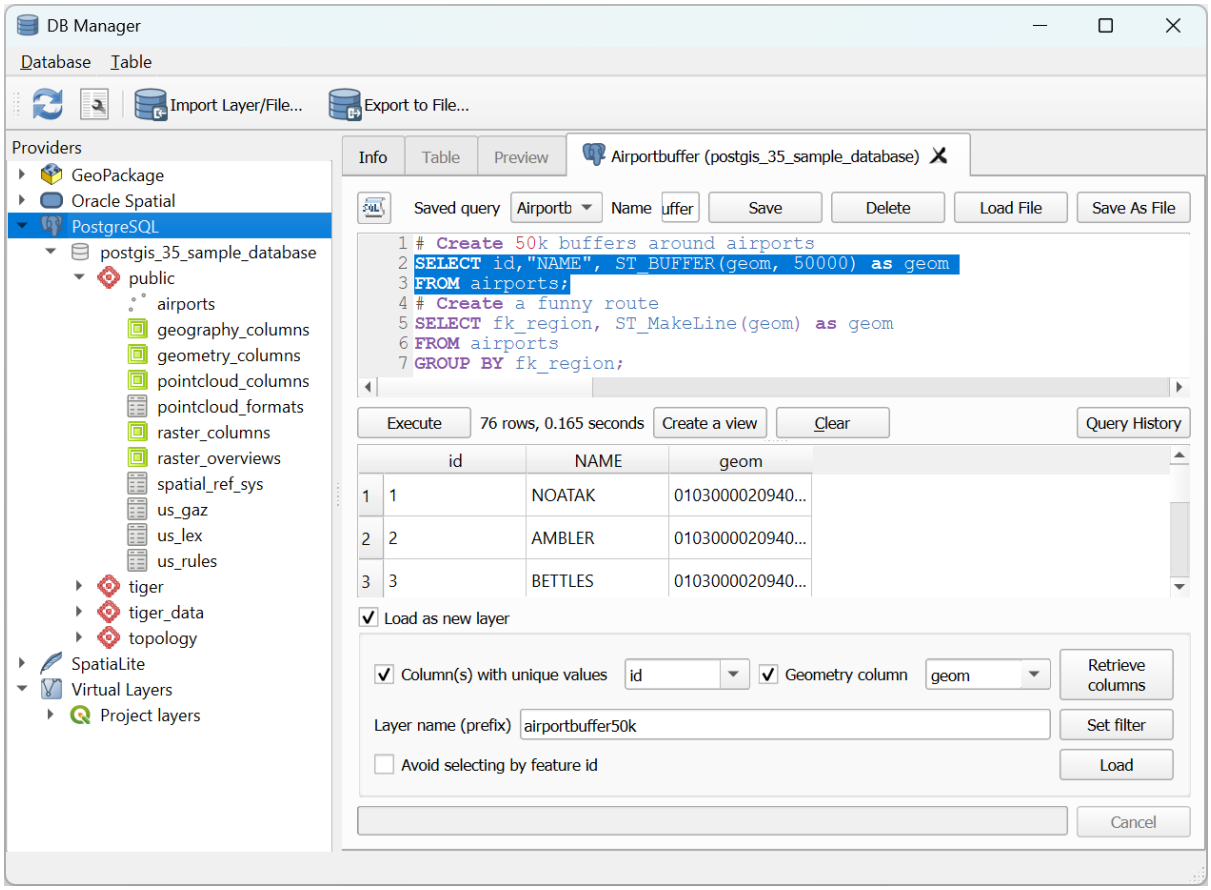


Fig. 25.5: Executing SQL queries in the DB Manager SQL window

> **Note**
>
> The SQL Window can also be used to create Virtual Layers. In that case, instead of selecting a database, select **QGIS Layers** under **Virtual Layers** before opening the SQL Window. See *Creating virtual layers* for instructions on the SQL syntax to use.

25.2.2 Geometry Checker Plugin

Geometry Checker is a powerful core plugin to check and fix the geometry validity of a layer. It is available from the *Vector* menu ( *Check Geometries…*).

Configuring the checks

The *Check Geometries* dialog shows different grouped settings in the first tab (*Setup*):

- *Input vector layers*: to select the layers to check. A  *Only selected features* checkbox can be used to restrict the checking to the geometries of the selected features.
- *Allowed geometry types* gives the chance to restrict the geometry type of the input layer(s) to:
  -  Point
  -  Multipoint
  -  Line
  -  Multiline
  -  Polygon
  -  Multipolygon
- *Geometry validity*. Depending on geometry types you can choose between:
  -  *Self intersections*
  -  *Duplicate nodes*
  -  *Self contacts*
  -  *Polygon with less than 3 nodes*.
- *Geometry properties*. Depending on geometry types, different options are available:
  -  *Polygons and multipolygons may not contain any holes*
  -  *Multipart objects must consist of more than one part*
  -  *Lines must not have dangles*
- *Geometry conditions*. Allows you to add some condition to validate the geometries with:
  -  *Minimal segment length (map units)* 
  -  *Minimum angle between segment (deg)* 
  -  *Minimal polygon area (map units sqr.)* 
  -  *No sliver polygons* with a *Maximum thinness*  and a  *Max. area (map units sqr.)* 

- *Topology checks*. Depending on geometry types, many different options are available:
  -  *Checks for duplicates*
  -  *Checks for features within other features*
  -  *Checks for overlaps smaller than* 
  -  *Checks for gaps smaller than* 
  -  *Points must be covered by lines*
  -  *Points must properly lie inside a polygon*
  -  *Lines must not intersect any other lines*
  -  *Lines must not intersect with features of layer* 
  -  *Polygons must follow boundaries of layer* 
- *Tolerance*. You can define the tolerance of the check in map layer units.
- *Output vector layer* gives the choice to:
  -  *Modify input layer*
  -  *Create new layers*

When you are happy with the configuration, you can click on the *Run* button.

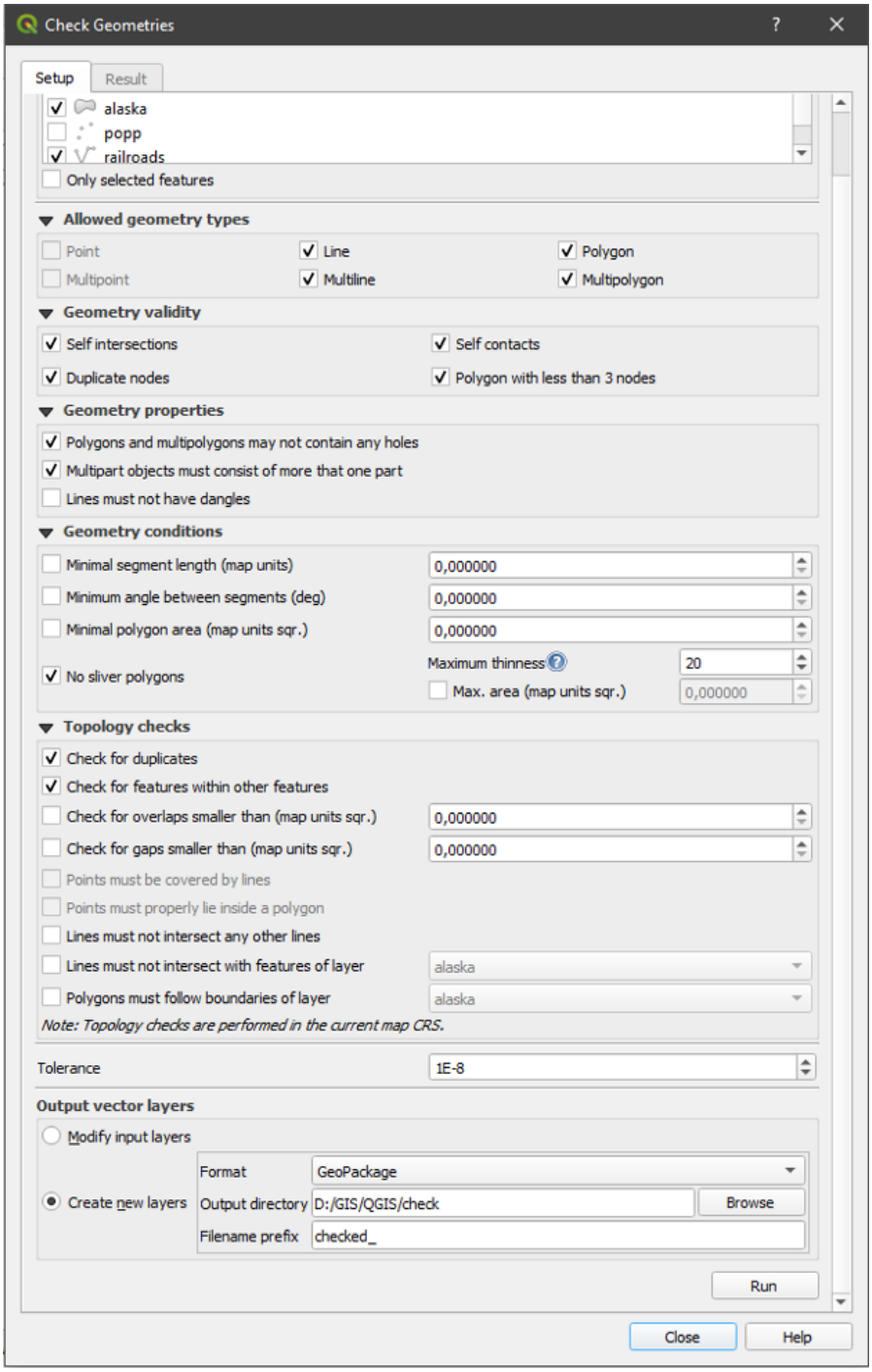


Fig. 25.6: The Geometry Checker Plugin

The *Geometry Checker Plugin* can find the following errors:

- Self intersections: a polygon with a self intersection
- Duplicate nodes: two duplicates nodes in a segment
- Holes: hole in a polygon
- Segment length: a segment length lower than a threshold
- Minimum angle: two segments with an angle lower than a threshold
- Minimum area: polygon area lower than a threshold

- Silver polygon: this error come from very small polygon (with small area) with a large perimeter
- Duplicates features
- Feature within feature
- Overlaps: polygon overlapping
- Gaps: gaps between polygons

The following figure shows the different checks made by the plugin.

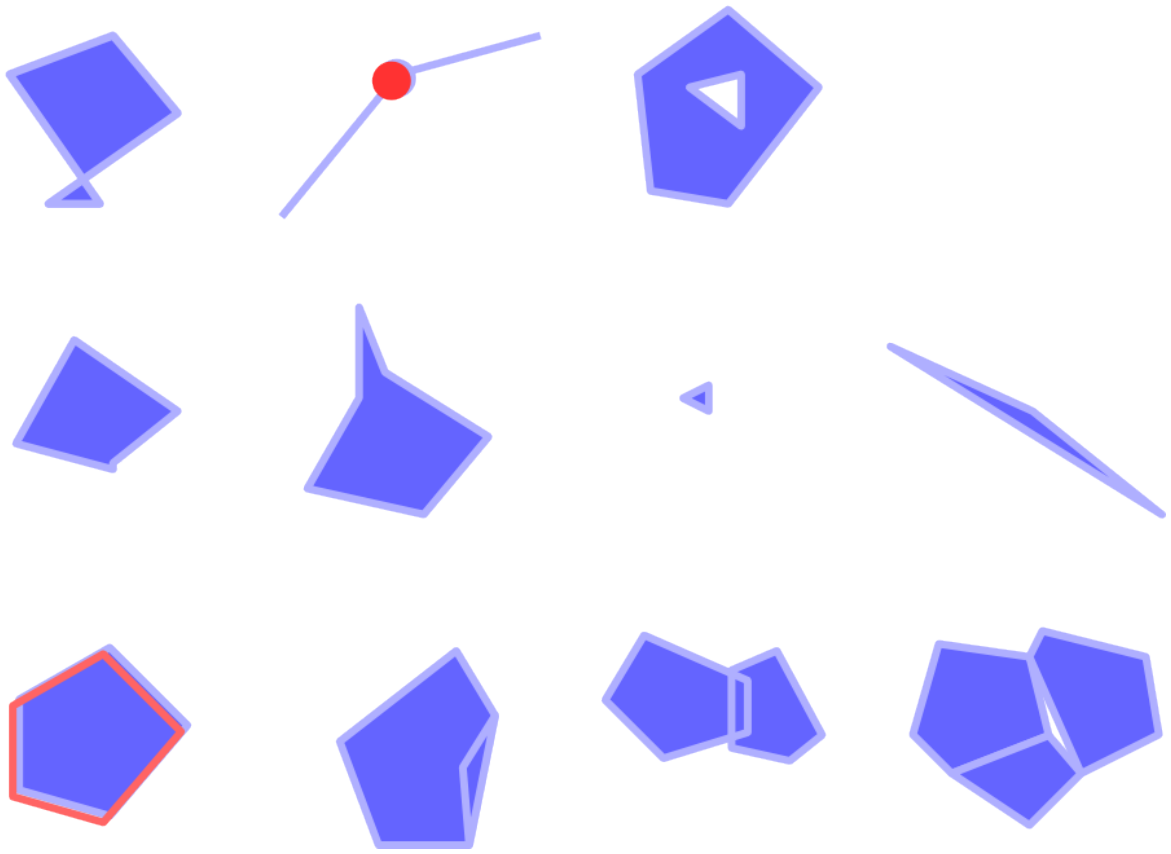


Fig. 25.7: Some checks supported by the plugin

Analysing the results

The results appear in the second tab (*Result*) and as an overview layer of the errors in the canvas (its name has the default prefix `checked_`). A table lists the *Geometry check result* with one error per row and columns containing: the layer name, an ID, the error type, then the coordinates of the error, a value (depending on the type of the error) and finally the resolution column which indicates the resolution of the error. At the bottom of this table, you can *Export* the error into different file formats. You also have a counter with the number of total errors and fixed ones.

You can select a row to see the location of the error. You can change this behavior by selecting another action between *Error* (default), *Feature*, *Don't move*, and *Highlight selected features*.

Below the zoom action when clicking on the table row, you can:

- *Show selected features in attribute table*
- *Fix selected errors using default resolution*
- *Fix selected errors, prompt for resolution method* You will see a window to choose the resolution's method among which:

  - Merge with neighboring polygon with longest shared edge
  - Merge with neighboring polygon with largest area
  - Merge with neighboring polygon with identical attribute value, if any, or leave as is
  - Delete feature
  - No action
- *Error resolution settings* allows you to change the default resolution method depending on the error type

> **Tip**
>
> **Fix multiple errors**
>
> You can fix multiple errors by selecting more than one row in the table with the *CTRL + click* action.

Finally, you can choose which *Attribute to use when merging features by attribute value*.

25.2.3 MetaSearch Catalog Client

Introduction

MetaSearch is a QGIS plugin to interact with metadata catalog services, supporting both the OGC API - Records and OGC Catalog Service for the Web (CSW) standards.

MetaSearch provides an easy and intuitive approach and user-friendly interface to searching metadata catalogs within QGIS.

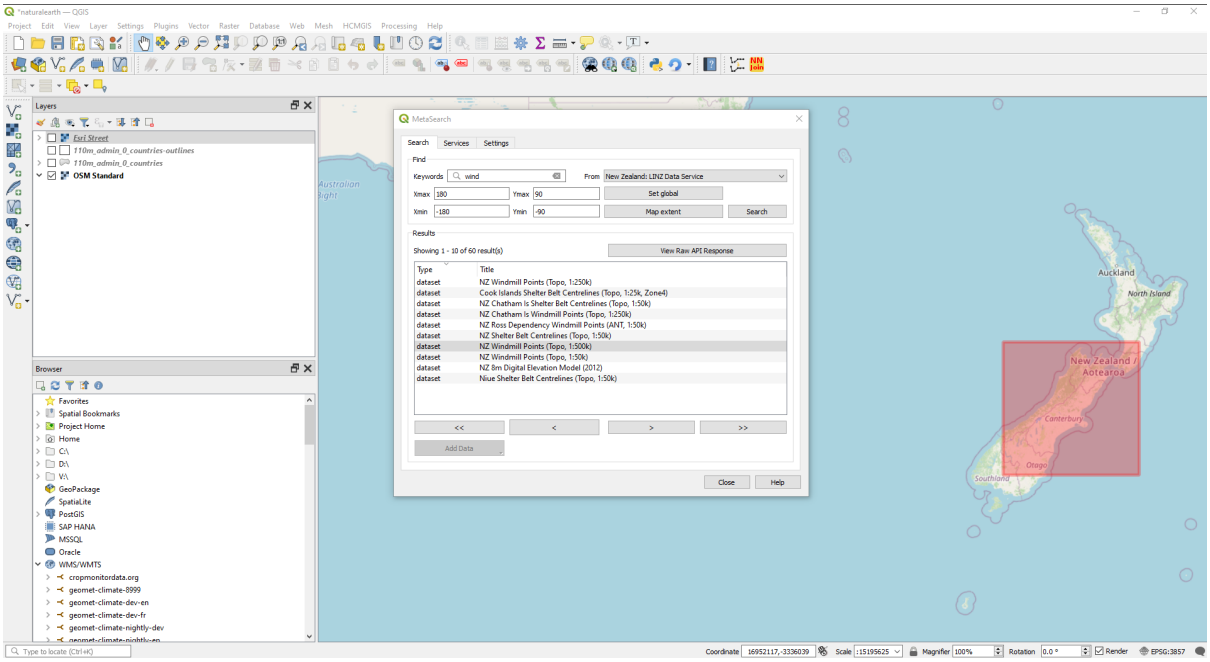


Fig. 25.8: Search and results of Services in MetaSearch

Working with Metadata Catalogs in QGIS

MetaSearch is included by default in QGIS, with all of its dependencies, and can be enabled from the QGIS Plugin Manager.

OGC API - Records

OGC API - Records is an OGC (Open Geospatial Consortium) standard for the discovery of geospatial resources on the Web.

CSW (Catalog Service for the Web)

CSW (Catalog Service for the Web) is an OGC (Open Geospatial Consortium) specification that defines common interfaces to discover, browse and query metadata about data, services, and other potential resources.

Startup

To start MetaSearch, click the  icon or select *Web* ► *MetaSearch* ► *MetaSearch* via the QGIS main menu. The MetaSearch dialog will appear. The main GUI consists of three tabs: *Services*, *Search* and *Settings*.

Managing Catalog Services

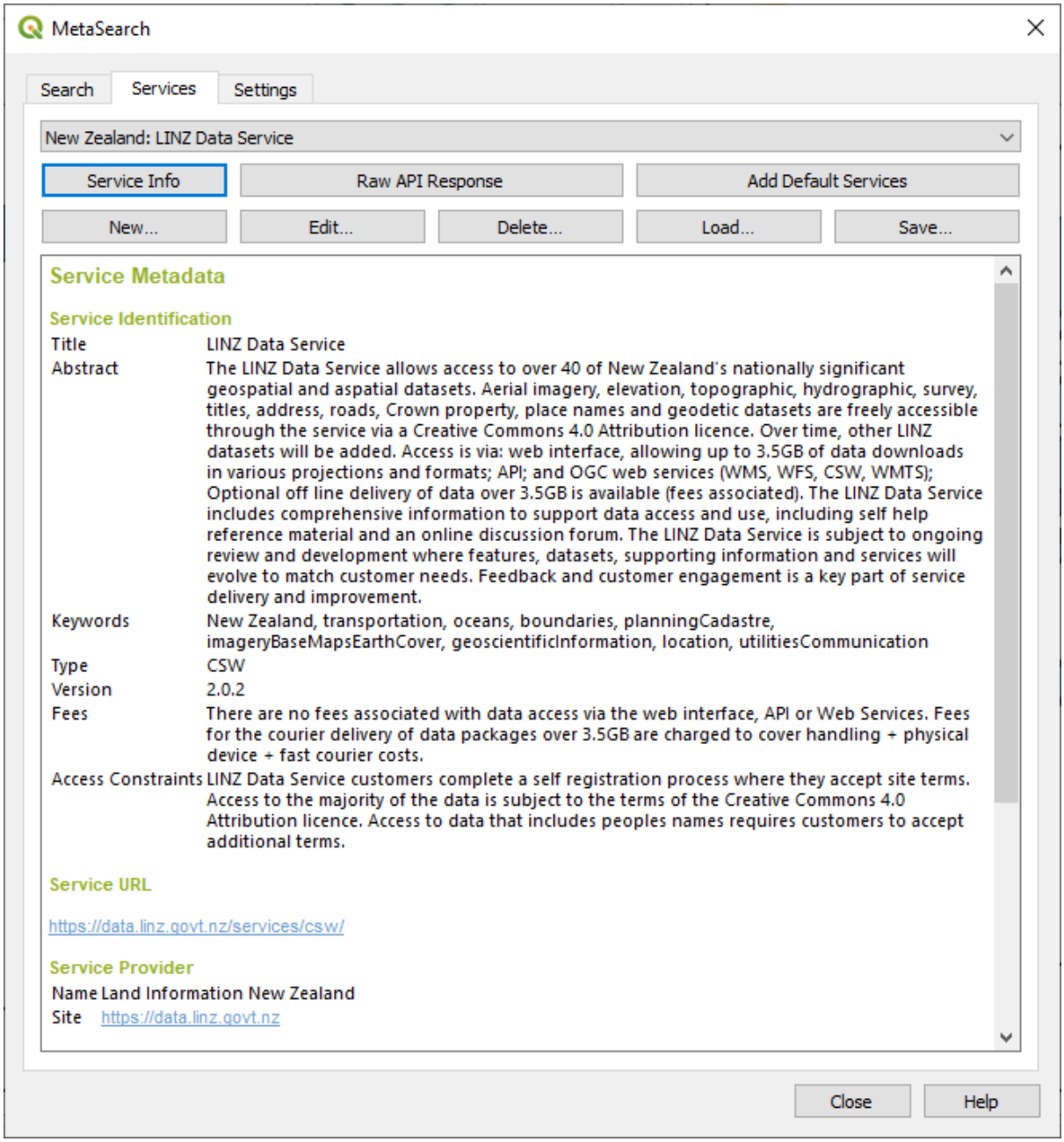


Fig. 25.9: Managing Catalog Services

The *Services* tab allows the user to manage all available catalog services. MetaSearch provides a default list of Catalog Services, which can be added by pressing the *Add Default Services* button.

To find all listed Catalog Service entries, click the dropdown select box.

To add a Catalog Service entry:

1. Click the *New* button
2. Enter a *Name* for the service, as well as the *URL* (endpoint). Note that for OGC CSW 2.0.2 Catalogs, only the base URL is required (not a full GetCapabilities URL). For OGC API - Records Catalogs, the URL should be the path to the collection endpoint
3. If the Catalog requires authentication, enter the appropriate *User name* and *Password* credentials.

4. Click *OK* to add the service to the list of entries.

To edit an existing Catalog Service entry:

1. Select the entry you would like to edit
2. Click the *Edit* button
3. And modify the *Name* or *URL* values
4. Click *OK*.

To delete a Catalog Service entry, select the entry you would like to delete and click the *Delete* button. You will be asked to confirm deleting the entry.

MetaSearch allows loading and saving connections to an XML file. This is useful when you need to share settings between applications. Below is an example of the XML file format.

```
<?xml version="1.0" encoding="UTF-8"?>
<qgsCSWConnections version="1.0">
    <csw type="OGC CSW 2.0.2" name="Data.gov CSW" url="https://catalog.data.gov/
↪csw-all"/>
    <csw type="OGC CSW 2.0.2" name="Geonorge - National CSW service for Norway"␣
↪url="https://www.geonorge.no/geonetwork/srv/eng/csw"/>
    <csw type="OGC CSW 2.0.2" name="Geoportale Nazionale - Servizio di ricerca␣
↪Italiano" url="http://www.pcn.minambiente.it/geoportal/csw"/>
    <csw type="OGC CSW 2.0.2" name="LINZ Data Service" url="http://data.linz.govt.
↪nz/feeds/csw"/>
    <csw type="OGC CSW 2.0.2" name="Nationaal Georegister (Nederland)" url="http://
↪www.nationaalgeoregister.nl/geonetwork/srv/eng/csw"/>
    <csw type="OGC CSW 2.0.2" name="RNDT - Repertorio Nazionale dei Dati␣
↪Territoriali - Servizio di ricerca" url="http://www.rndt.gov.it/RNDT/CSW"/>
    <csw type="OGC CSW 2.0.2" name="UK Location Catalogue Publishing Service" url=
↪"http://csw.data.gov.uk/geonetwork/srv/en/csw"/>
    <csw type="OGC CSW 2.0.2" name="UNEP/GRID-Geneva Metadata Catalog" url="http://
↪metadata.grid.unep.ch:8080/geonetwork/srv/eng/csw"/>
</qgsCSWConnections>
```

To load a list of entries:

1. Click the *Load* button. A new window will appear.
2. Click the *Browse* button and navigate to the XML file of entries you wish to load.
3. Click *Open*. The list of entries will be displayed.
4. Select the entries you wish to add from the list and click *Load*.

Click the *Service Info* button to display information about the selected Catalog Service such as service identification, service provider and contact information. If you would like to view the raw API response, click the *Raw API Response* button. A separate window will open displaying server information in raw JSON or XML format.

Searching Catalog Services

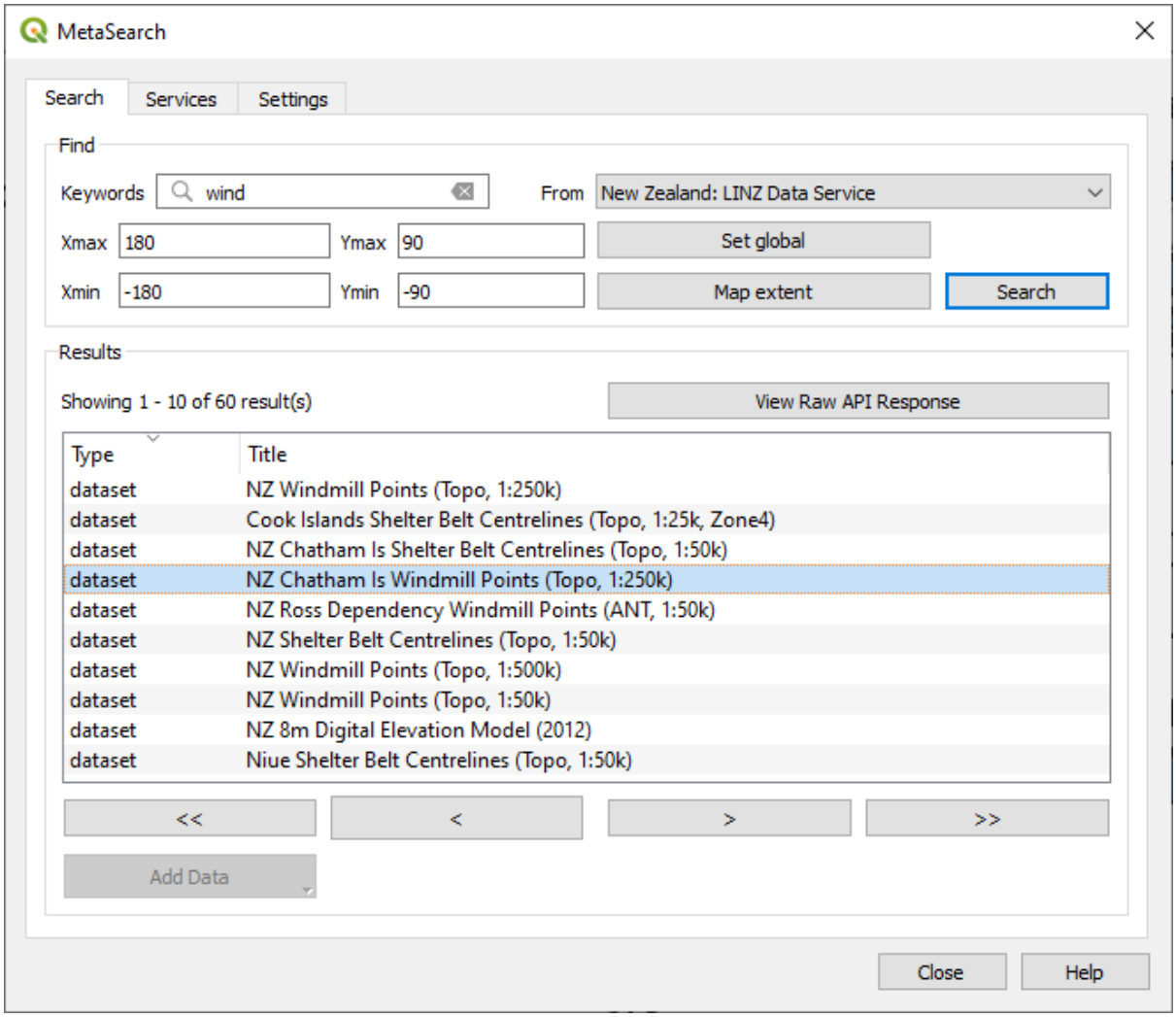


Fig. 25.10: Searching catalog services

The *Search* tab allows the user to query Catalog Services for data and services, set various search parameters and view results.

The following search parameters are available:

- *Keywords*: free text search keywords;
- *From*: the Catalog Service to perform the query against;
- **Bounding box**: the spatial area of interest to filter, defined by *Xmax*, *Xmin*, *Ymax*, and *Ymin*. Click *Set Global* to do a global search, click *Map Extent* to do a search in the visible area, or enter values manually.

Clicking the *Search* button will search the selected Metadata Catalog. Search results are displayed in a list, and can be sorted by clicking on the column header. You can navigate through search results with the directional buttons below the search results.

Select a result and:

- Click the *View Raw API Response* button to open a window with the service response in raw JSON or XML format.
- If the metadata record has an associated bounding box, a footprint of the bounding box will be displayed on the map.

- Double-click the record to display the record metadata with any associated access links. Clicking a link opens the link in the user's web browser.
- If the record is a supported web service (WMS/WMTS, WFS, WCS, ArcGIS REST Service, etc.) or a GIS File, the *Add Data* button will be enabled. When clicking this button, MetaSearch will verify if this is a valid OWS. The service will then be added to the appropriate QGIS connection list, and the appropriate connection dialog will appear. If you choose *Add GIS File* the linked GIS file will be added to the current project.

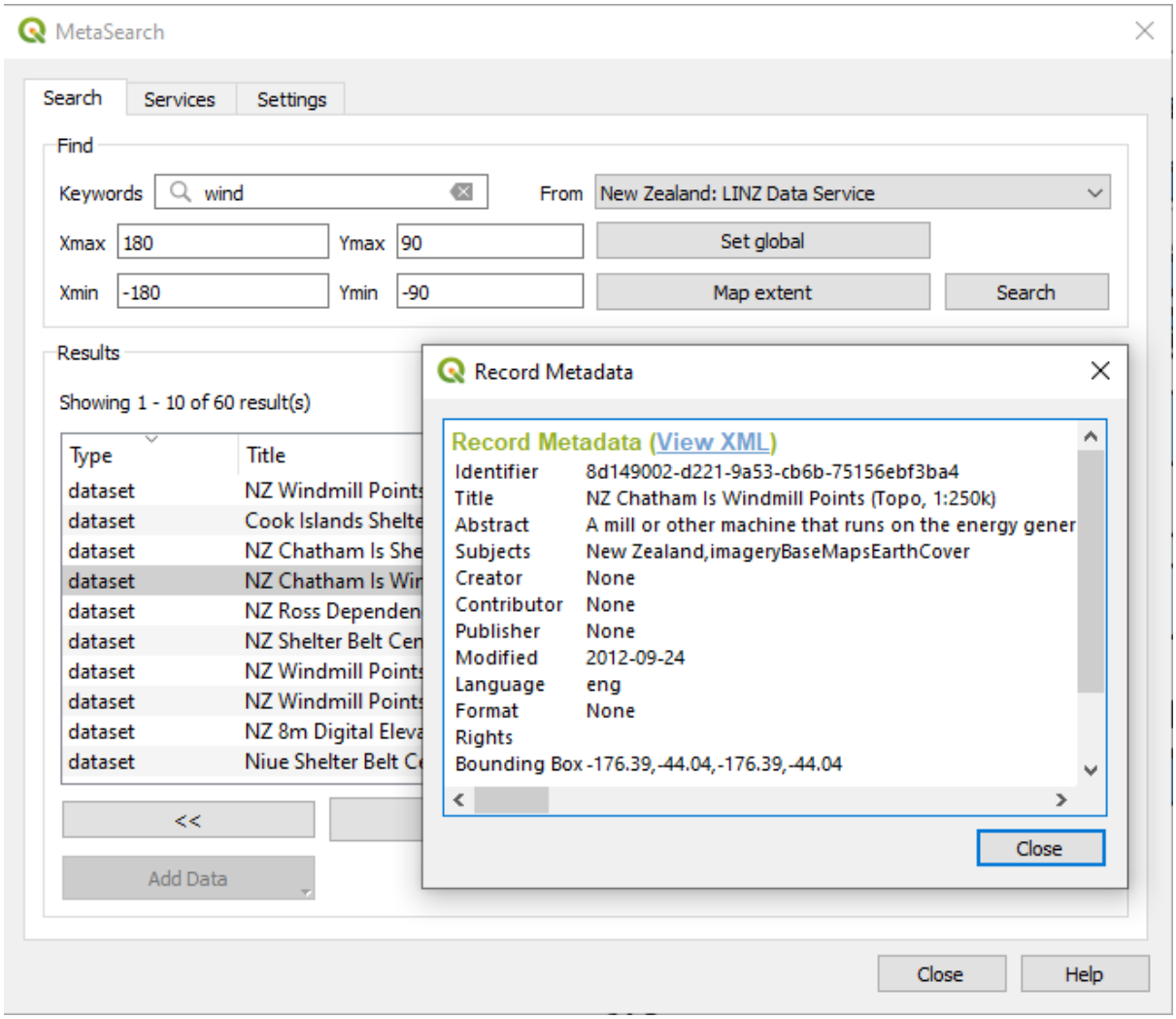


Fig. 25.11: Metadata record display

Settings

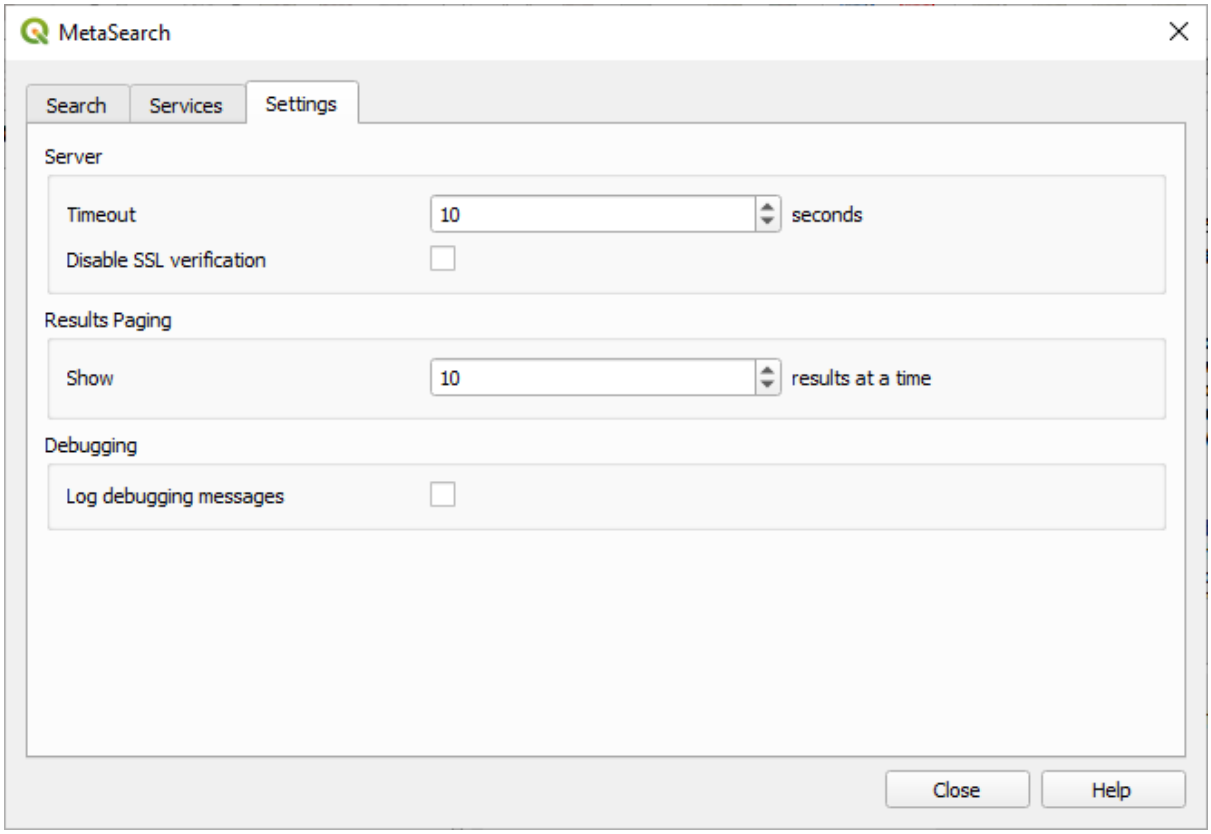


Fig. 25.12: MetaSearch settings

You can fine tune MetaSearch with the following *Settings*:

- *Server Timeout*: when searching metadata catalogs, the number of seconds for blocking connection attempt. Default value is 10.
- *Disable SSL verification*: option to switch off ssl verification.
- *Results paging*: when searching metadata catalogs, the number of results to show per page. Default value is 10.
- *Log debugging messages*: option to enable the logging of debugging messages in the *Log Messages Panel*.

Catalog Server Errors

In some cases, the Catalog will work in a web browser, but not in MetaSearch. This may be due to the Catalog server's configuration/setup. Catalog server providers should ensure URLs are consistent and up to date in their configuration (this is common in HTTP -> HTTPS redirection scenarios). Please see the pycsw FAQ item for a deeper explanation of the issue and fix. Although the FAQ item is pycsw specific it can also apply in general to other Catalog servers.

25.2.4 Offline Editing Plugin

For data collection, it is a common situation to work with a laptop or a cell phone offline in the field. Upon returning to the network, the changes need to be synchronized with the master datasource (e.g., a PostgreSQL database). If several persons are working simultaneously on the same datasets, it is difficult to merge the edits by hand, even if people don't change the same features.

The  Offline Editing Plugin automates the synchronisation by copying the content of the datasource to a SpatiaLite or GeoPackage database and storing the offline edits to dedicated tables. After being connected to the network again, it is possible to apply the offline edits to the master dataset.

To use the plugin:

1. Open a project with some vector layers (e.g., from an Esri Shapefile, PostgreSQL or WFS-T datasource).
2. Assuming you have already enabled the plugin (see *Core and External plugins*) go to *Database* ► *Offline Editing* ►  *Convert to offline project*. The eponym dialog opens.
3. Select the *Storage type*. It can be of *GeoPackage* or *SpatiaLite* database type.
4. Use the *Browse* button to indicate the location of the database in which to store the *Offline data*. It can be an existing file or one to create.
5. In the *Select remote layers* section, check the layers you'd like to save. The content of the layers is saved to database tables.

   Note

   Since target database formats do not have native list support, the offline editing plugin transforms {string, number} list fields into string fields where values are separated by commas. This allows reading and edit of the contents of those fields when offline.

   If you would like to handle both the field from the original layer and the offline layer, you can rely on the *try()* and *array* expression functions, e.g.:

   ```
   try(array_contains("field",1),array_contains(string_to_array("field"),1))
   ```

6. You can check  *Only synchronize selected features if a selection is present* allowing to only save and work on a subset. It can be invaluable in case of large layers.

   This is all!

7. Save your project and bring it on the field.
8. Edit the layers offline.
9. After being connected again, upload the changes using *Database* ► *Offline Editing* ►  *Synchronize*.

Note

Layers that are used offline are marked with the  icon in the *Layers* panel.

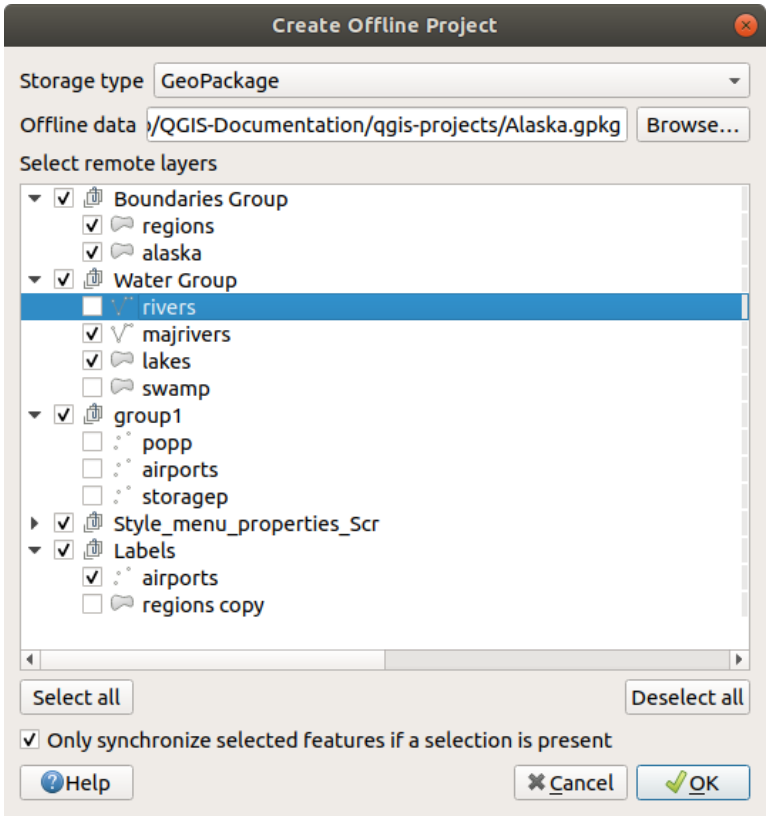


Fig. 25.13: Create an offline project

25.2.5 Topology Checker Plugin

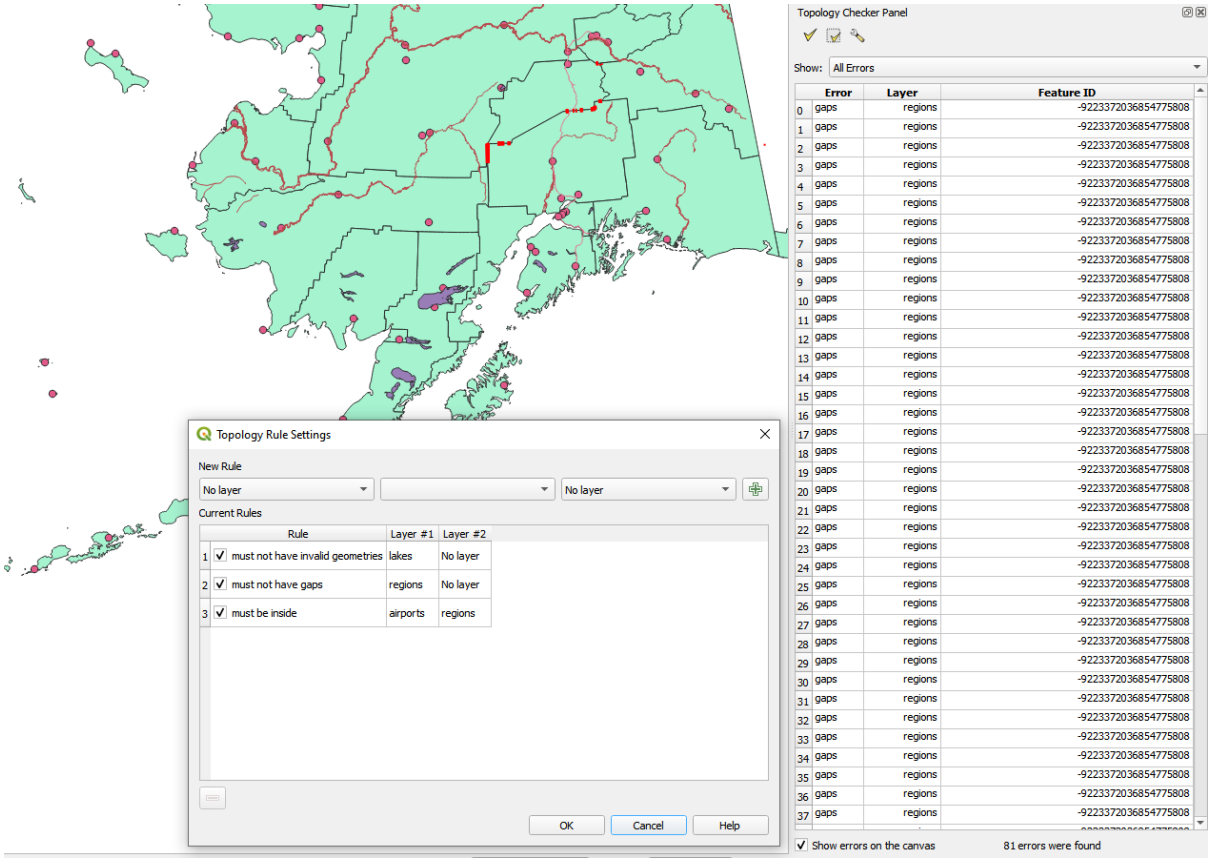


Fig. 25.14: The Topology Checker Plugin

Topology describes the relationships between points, lines and polygons that represent the features of a geographic region. With the Topology Checker plugin, you can look over your vector files and check the topology with several topology rules. These rules check with spatial relations whether your features 'Equal', 'Contain', 'Cover', are 'CoveredBy', 'Cross', are 'Disjoint', 'Intersect', 'Overlap', 'Touch' or are 'Within' each other. It depends on your individual questions which topology rules you apply to your vector data (e.g., normally you won't accept overshoots in line layers, but if they depict dead-end streets you won't remove them from your vector layer).

QGIS has a built-in topological editing feature, which is great for creating new features without errors. But existing data errors and user-induced errors are hard to find. This plugin helps you find such errors through a list of rules.

To enable the *Topology checker* plugin:

1. Go to *Plugins* menu
2. Open *Manage and Install plugins* and choose *Installed*
3. Enable *Topology checker*
4. Close the *Plugin manager* dialog. A *Topology checker* entry is added to the *Vector* menu.

After you enabled *Topology checker* open it and choose *Configure* to create your topology rules.

On **point layers** the following rules are available:

- *Must be covered by*: Here you can choose a vector layer from your project. Points that aren't covered by the given vector layer occur in the 'Error' field.
- *Must be covered by endpoints of*: Here you can choose a line layer from your project.

- *Must be inside*: Here you can choose a polygon layer from your project. The points must be inside a polygon. Otherwise, QGIS writes an 'Error' for the point.
- *Must not have duplicates*: Whenever a point is represented twice or more, it will occur in the 'Error' field.
- *Must not have invalid geometries*: Checks whether the geometries are valid.
- *Must not have multi-part-geometries*: All multi-part points are written into the 'Error' field.

On **line layers**, the following rules are available:

- *End points must be covered by*: Here you can select a point layer from your project.
- *Must not have dangles*: This will show the overshoots in the line layer.
- *Must not have duplicates*: Whenever a line feature is represented twice or more, it will occur in the 'Error' field.
- *Must not have invalid geometries*: Checks whether the geometries are valid.
- *Must not have multi-part geometries*: Sometimes, a geometry is actually a collection of simple (single-part) geometries. Such a geometry is called multi-part geometry. If it contains just one type of simple geometry, we call it multi-point, multi-linestring or multi-polygon. All multi-part lines are written into the 'Error' field.
- *Must not have pseudos*: A line geometry's endpoint should be connected to the endpoints of two other geometries. If the endpoint is connected to only one other geometry's endpoint, the endpoint is called a pseudo node.

On **polygon layers**, the following rules are available:

- *Must contain*: Polygon layer must contain at least one point geometry from the second layer.
- *Must not have duplicates*: Polygons from the same layer must not have identical geometries. Whenever a polygon feature is represented twice or more it will occur in the 'Error' field.
- *Must not have gaps*: Adjacent polygons should not form gaps between them. Administrative boundaries could be mentioned as an example (US state polygons do not have any gaps between them…).
- *Must not have invalid geometries*: Checks whether the geometries are valid. Some of the rules that define a valid geometry are:
  - Polygon rings must close.
  - Rings that define holes should be inside rings that define exterior boundaries.
  - Rings may not self-intersect (they may neither touch nor cross one another).
  - Rings may not touch other rings, except at a point.
- *Must not have multi-part geometries*: Sometimes, a geometry is actually a collection of simple (single-part) geometries. Such a geometry is called multi-part geometry. If it contains just one type of simple geometry, we call it multi-point, multi-linestring or multi-polygon. For example, a country consisting of multiple islands can be represented as a multi-polygon.
- *Must not overlap*: Adjacent polygons should not share common area.
- *Must not overlap with*: Adjacent polygons from one layer should not share common area with polygons from another layer.

When you create a *New rule* click on the  Add rule to include it to the *Current rules*. You can enable or disable individual rules by clicking on the checkbox. Right-clicking over a rule provides the following options:

- *Select All* the rules
- *Activate* or *Deactivate* the selected rules
- *Toggle activation* of selected rules
- *Delete* selected rules. This can also be achieved with the  Delete selected rules button.

Press *OK* and then choose from the *Topology checker* panel:

- Validate All: applies the active rules to all the features of the involved layer(s)
- or Validate Extent: applies the active rules to the features of the involved layer(s), within the current map canvas. The button is kept pushed and the results will update as the map canvas extent changes.

Errors will show up in the table of results containing type of error, layer and feature ID. Use *Filter errors by rule* menu to filter the errors to a specific error type.

Check *Show errors on the canvas* to show error location on the canvas. Clicking a row in the table will zoom the map canvas to the concerned feature, where you can use *QGIS digitizing tools* to fix the error.

Below is the list of Core plugins provided with QGIS. They are not necessarily enabled by default.

Icon	Plugin	Description	Manual Reference
	DB Manager	Manage your databases within QGIS	*DB Manager Plugin*
	Geometry Checker	Check and repair errors in vector geometries	*Geometry Checker Plugin*
	GRASS 7	GRASS functionality	*GRASS GIS Integration*
	GRASS GIS provider	GRASS GIS Processing functionality	*GRASS GIS Integration*
	MetaSearch Catalog Client	Interact with metadata catalog services (CSW)	*MetaSearch Catalog Client*
	Offline Editing	Offline editing and synchronizing with database	*Offline Editing Plugin*
	Processing	Spatial data processing framework	*QGIS processing framework*
	Topology Checker	Find topological errors in vector layers	*Topology Checker Plugin*

Note

To use the Core Plugins GRASS 7, GRASS GIS provider, they have to be configured. Informations can be found *here*.

25.3 QGIS Python console

As you will see later in this chapter, QGIS has been designed with a plugin architecture. Plugins can be written in Python, a very famous language in the geospatial world.

QGIS brings a Python API (see PyQGIS Developer Cookbook for some code sample) to let the user interact with its objects (layers, feature or interface). QGIS also has a Python console.

The QGIS Python Console is an interactive shell for Python command executions. It also has a Python file editor that allows you to edit and save your Python scripts. Both console and editor are based on PyQScintilla2 package. To open the console go to *Plugins* ► *Python Console* (`Ctrl+Alt+P`) or click on the Python Console icon in the *Plugins* toolbar.

25.3.1 The Interactive Console

The console is a Python interpreter that allows you to execute Python commands. Modules from QGIS (analysis, core, gui, server, processing, 3d) and Qt (QtCore, QtGui, QtNetwork, QtWidgets, QtXml) as well as Python's math, os, re and sys modules are already imported and can be used directly.

The interactive console is composed of a toolbar, an input area and an output one.

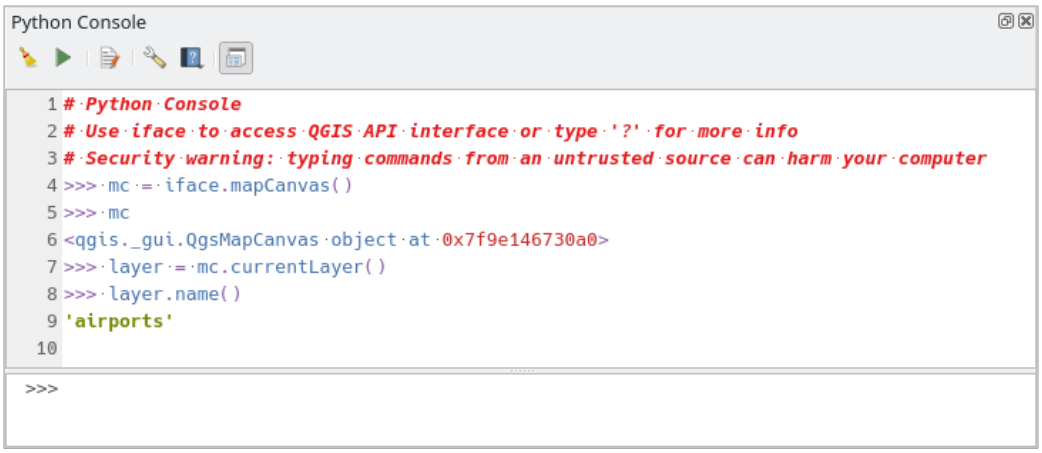


Fig. 25.15: The Python Console

Toolbar

The toolbar proposes the following tools:

- Clear Console to wipe the output area;
- Run Command available in the input area: same as pressing `Enter`;
- Show Editor: toggles *The Code Editor* visibility;
- Options...: opens a dialog to configure *console properties*;
- Help... provides a menu to access various documentation:
  - *Python Console Help* (the current page)
  - PyQGIS API documentation
  - PyQGIS Cookbook
- Dock Code Editor to dock or undock the panel in QGIS interface

Input area

The Console input area main features are:

- Code completion, highlighting syntax and calltips for the following APIs:
  - Python
  - PyQGIS
  - PyQt5
  - QScintilla2
  - osgeo-gdal-ogr

- `Ctrl+Alt+Space` to view the auto-completion list if enabled in the *Python settings*;
- Execute code snippets from the input area by typing and pressing `Enter` or *Run Command*;
- Execute code snippets from the output area using the *Enter Selected* from the contextual menu or pressing `Ctrl+E`;
- Browse the command history from the input area using the `Up` and `Down` arrow keys and execute the command you want;
- `Ctrl+Shift+Space` to view the command history: double-clicking a row will execute the command. The *Command History* dialog can also be accessed from context menu of input area;
- Save and clear the command history. The history will be saved into the `console_history.txt` file under the active *user profile* folder;
- Type the following special commands:
  - `?` to show a help of the Python Console
  - `_api` to open QGIS C++ API documentation or `_api(object)` for a specific object documentation (in QGIS C++ API or Qt API documentation)
  - `_pyqgis` to open QGIS Python API documentation or `_pyqgis(object)` for a specific object documentation (in QGIS Python API or Qt API documentation)
  - `_cookbook` to open PyQGIS Cookbook.
  - `!` followed by a command to execute Shell commands from the Python Console. The console will start a subprocess, and forward its output to the Python Console Output. While the subprocess is running, the Python Console Input switches to STDIN mode and forwards entered character to the child process. This makes it possible to send confirmation when the child program asks for it. When the Console is in STDIN mode, pressing `Ctrl+C` kills the subprocess. It is also possible to affect the result of a command to a variable with the syntax `var = !cmd`

```
>>> !echo QGIS Rocks!
QGIS Rocks

>>> !gdalinfo --version
GDAL 3.6.2, released 2023/01/02

>>> !pip install black
# Install black python formatter using pip (if available)

>>> sql_formats = !ogrinfo --formats | grep SQL
>>> sql_formats
['SQLite -vector- (rw+v): SQLite / Spatialite', '  MSSQLSpatial -vector-␣
↪(rw+): Microsoft SQL Server Spatial Database', '  PostgreSQL -vector-␣
↪(rw+): PostgreSQL/PostGIS', '  MySQL -vector- (rw+): MySQL', '  PGDUMP -
↪vector- (w+v): PostgreSQL SQL dump']
```

Tip

Reuse executed commands from the output panel

You can execute code snippets from the output panel by selecting some text and pressing `Ctrl+E`. No matter if selected text contains the interpreter prompt (`>>>`, `...`).

25.3.2 The Code Editor

Use the Show Editor button in the Interactive Console to enable the editor widget. It allows editing and saving Python scripts and offers advanced functionalities to manage your code. Depending on the *enabled settings*, it provides various capabilities for easier code writing, such as code completion, highlighting syntax and calltips for supported APIs. Automatic indentation, parenthesis insertion, code commenting and syntax checking are also available.

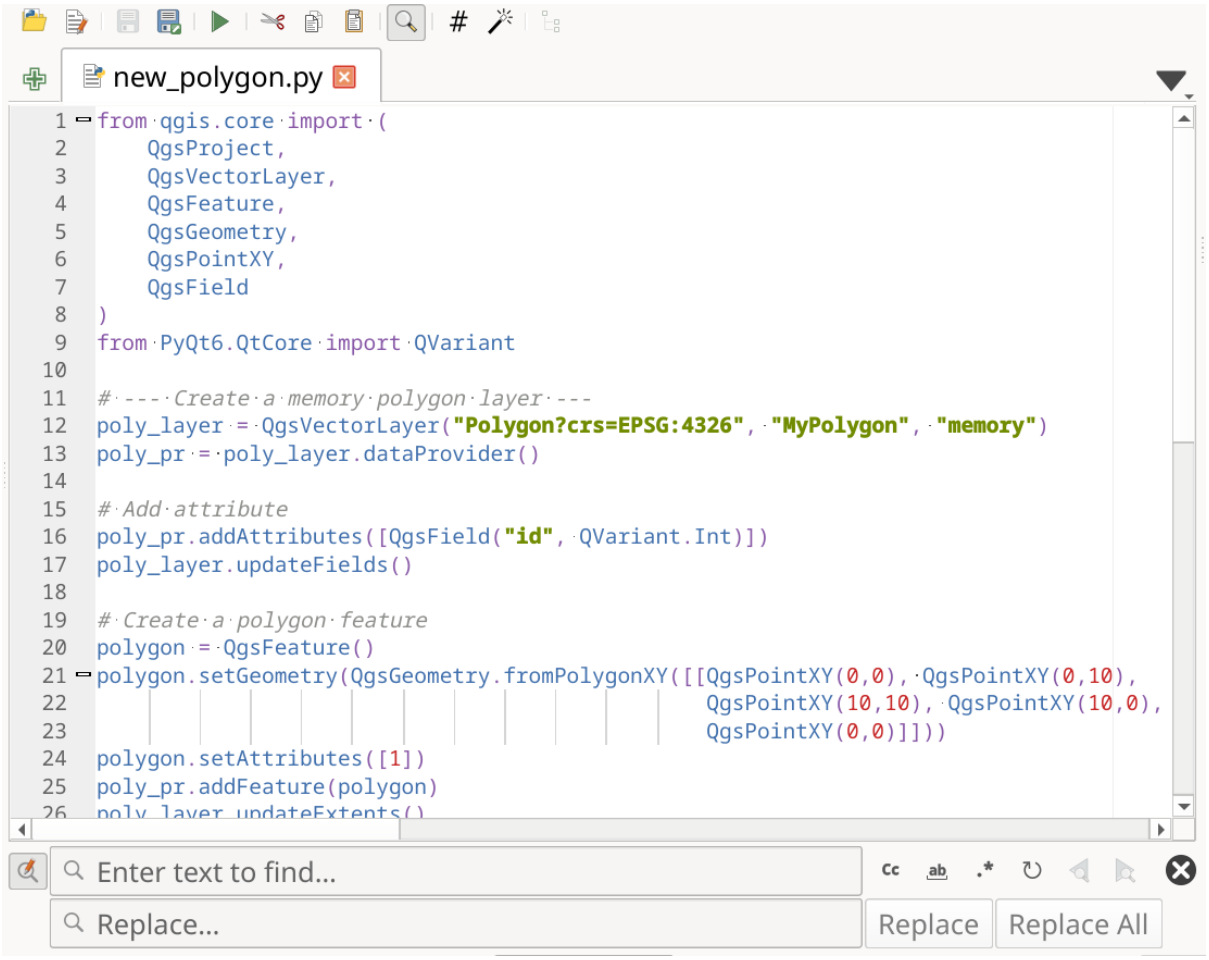


Fig. 25.16: The Python Console editor

The code editor area allows to simultaneously work on different scripts, each in a specific tab. Press New editor to add a new tab. You can run partially or totally a script from the *Code Editor* and output the result in the Interactive Console output area.

> **Tip**
>
> Press `Ctrl+Space` to view the auto-completion list.

At the top of the dialog, a toolbar provides access to a few commands. Right-clicking the editor area provides some more options. All available tools are described in the following table.

Table 25.1: Python Console Editor Tools

Tool name	Function	Location
Open script…	Loads a Python file in the code editor dialog, as a new tab	Toolbar
Open in external editor…	Opens a saved Python script in the default external program set for Python file editing	
Save (`Ctrl+S`)	Saves the current script	
Save as… (`Ctrl+Shift+S`)	Saves the current script as a new file	
Run script (`Ctrl+Shift+E`)	Executes the whole script in the Interactive console (this creates a byte-compiled file with the extension `.pyc`)	Toolbar & Contextual menu
Context help (`F1`)	Attempts to display help on the selected string (class, method, object,…) in its corresponding API documentation	Contextual menu
Run selected (`Ctrl+E`)	Executes in the Interactive console the lines selected in the script	Toolbar & Contextual menu
Cut (`Ctrl+X`)	Cuts selected text to the clipboard	
Copy (`Ctrl+C`)	Copies selected text to the clipboard	
Paste (`Ctrl+V`)	Pastes a cut or copied text	
Find text	Allows to search and replace a text in the script. • Use the default Desktop Environment shortcut to find next/previous (`Ctrl+G` and `Shift+Ctrl+G`); • Automatically find first match when typing in find box; • Set initial find string to selection when opening find; • Enable the *Use Regular Expressions* option in the search bar to use regular expressions instead of plain text matching; • Replace the matching text with the text in the replace box. When using a regular expression, you can reference captured groups (e.g. via `\1`) in the replacement text. Use *Replace All* to replace all matches at once. • Pressing `Esc` closes the find bar.	
Toggle comment (`Ctrl+:`)	Comments out or uncomment selected lines, by adding or removing `#` character (potentially followed by a space) at the start of the lines. If the selection contains both comments and actual commands, adds `#` at the start of each non empty line. Pressing again the button reverts to the initial state.	
Reformat code (`Ctrl+Alt+F`)	Allows to manually apply various formatting rules (sort imports, indentation, line length,…) to the code, following *user-defined settings*. This may require installation of additional Python modules.	
Object inspector…	Shows and hides a dedicated browser with a tree structure for classes and functions available in the script. Click an item fo a quick access to its definition. The tool requires an activation from the *Python settings - Run and Debug*.	
Hide editor	Hides the Python code editor block. To make it visible again, press Show editor button from the interactive console toolbar.	Contextual menu
Check syntax (`Ctrl+4`)	Browses the code and reports syntax errors, such as missing parenthesis, colons, wrong indentation,…	
Undo (`Ctrl+Z`)	Undoes the latest action	
Redo	Reverts undone actions to a more recent	

CHAPTER

TWENTYSIX

APPENDICES

26.1 Appendix C: QGIS File Formats

26.1.1 QGS/QGZ - The QGIS Project File Format

The **QGS** format is an XML format for storing QGIS projects. The **QGZ** format is a compressed (zip) archive containing a QGS file and a QGD file. The **QGD** file is the associated sqlite database of the qgis project that contain auxiliary data for the project. If there are no auxiliary data, the QGD file will be empty.

A QGIS file contains everything that is needed for storing a QGIS project, including:

- project title
- project CRS
- the layer tree
- snapping settings
- relations
- the map canvas extent
- project models
- legend
- mapview docks (2D and 3D)
- the layers with links to the underlying datasets (data sources) and other layer properties including extent, SRS, joins, styles, renderer, blend mode, opacity and more.
- project properties

The figures below show the top level tags in a QGS file and the expanded `ProjectLayers` tag.

```
−<qgis version="3.4.13-Madeira" projectname="">
    <homePath path=""/>
    <title/>
    <autotransaction active="0"/>
    <evaluateDefaultValues active="0"/>
    <trust active="0"/>
  +<projectCrs></projectCrs>
  +<layer-tree-group></layer-tree-group>
  +<snapping-settings tolerance="12" unit="1" enabled="0" type="1" mode="2" intersection-snapping="0">
    </snapping-settings>
    <relations/>
  −<mapcanvas name="theMapCanvas" annotationsVisible="1">
      <units>meters</units>
    +<extent></extent>
      <rotation>0</rotation>
    +<destinationsrs></destinationsrs>
      <rendermaptile>0</rendermaptile>
    </mapcanvas>
    <projectModels/>
  +<legend updateDrawingOrder="true"></legend>
    <mapViewDocks/>
    <mapViewDocks3D/>
  +<projectlayers></projectlayers>
  +<layerorder></layerorder>
  +<properties></properties>
    <visibility-presets/>
    <transformContext/>
  +<projectMetadata></projectMetadata>
    <Annotations/>
    <Layouts/>
  </qgis>
```

Fig. 26.1: The top level tags in a QGS file

```
−<projectlayers>
 −<maplayer styleCategories="AllStyleCategories" readOnly="0" autoRefreshTime="0" autoRefreshEnabled="0" refreshOnNotifyEnabled="0" maxScale="0"
  geometry="Polygon" labelsEnabled="0" type="vector" simplifyDrawingHints="1" hasScaleBasedVisibilityFlag="0" simplifyDrawingTol="1"
  simplifyMaxScale="1" minScale="1e+8" simplifyAlgorithm="0" simplifyLocal="1" refreshOnNotifyMessage="">
  +<extent></extent>
   <id>watersheds_b62efa19_8809_4406_b6ec_2951ac4c94c5</id>
  −<datasource>
     ./QGIS-Training-Data-2.0/exercise_data/processing/generalize/watersheds.shp
   </datasource>
  +<keywordList></keywordList>
   <layername>watersheds</layername>
  +<srs></srs>
  +<resourceMetadata></resourceMetadata>
   <provider encoding="UTF-8">ogr</provider>
   <vectorjoins/>
   <layerDependencies/>
   <dataDependencies/>
   <legend type="default-vector"/>
   <expressionfields/>
  +<map-layer-style-manager current="default"></map-layer-style-manager>
   <auxiliaryLayer/>
  +<flags></flags>
  +<renderer-v2 symbollevels="0" enableorderby="0" type="singleSymbol" forceraster="0"></renderer-v2>
  +<customproperties></customproperties>
   <blendMode>0</blendMode>
   <featureBlendMode>0</featureBlendMode>
   <layerOpacity>1</layerOpacity>
  +<SingleCategoryDiagramRenderer diagramType="Histogram" attributeLegend="1"></SingleCategoryDiagramRenderer>
  +<DiagramLayerSettings priority="0" linePlacementFlags="18" dist="0" showAll="1" placement="1" obstacle="0" zIndex="0"></DiagramLayerSettings>
  +<geometryOptions removeDuplicateNodes="0" geometryPrecision="0"></geometryOptions>
  +<fieldConfiguration></fieldConfiguration>
  +<aliases></aliases>
   <excludeAttributesWMS/>
   <excludeAttributesWFS/>
  +<defaults></defaults>
  +<constraints></constraints>
  +<constraintExpressions></constraintExpressions>
   <expressionfields/>
  +<attributeactions></attributeactions>
  +<attributetableconfig actionWidgetStyle="dropDown" sortExpression="" sortOrder="0"></attributetableconfig>
  +<conditionalstyles></conditionalstyles>
   <editform tolerant="1"/>
   <editforminit/>
   <editforminitcodesource>0</editforminitcodesource>
   <editforminitfilepath/>
  +<editforminitcode></editforminitcode>
   <featformsuppress>0</featformsuppress>
   <editorlayout>generatedlayout</editorlayout>
  +<editable></editable>
  +<labelOnTop></labelOnTop>
   <widgets/>
   <previewExpression>ID</previewExpression>
   <mapTip/>
 </maplayer>
</projectlayers>
```


Fig. 26.2: The expanded top level ProjectLayers tag of a QGS file

26.1.2 QLR - The QGIS Layer Definition file

A Layer Definition file (QLR) is an XML file that contains a pointer to the layer data source in addition to QGIS style information for the layer.

The use case for this file is simple: To have a single file for opening a data source and bringing in all the related style information. QLR files also allow you to mask the underlying datasource in an easy to open file.

An example of QLR usage is for opening MS SQL layers. Rather than having to go to the MS SQL connection dialog, connect, select, load and finally style, you can simply add a .qlr file that points to the correct MS SQL layer with all the necessary style included.

In the future a .qlr file may hold a reference to more than one layer.

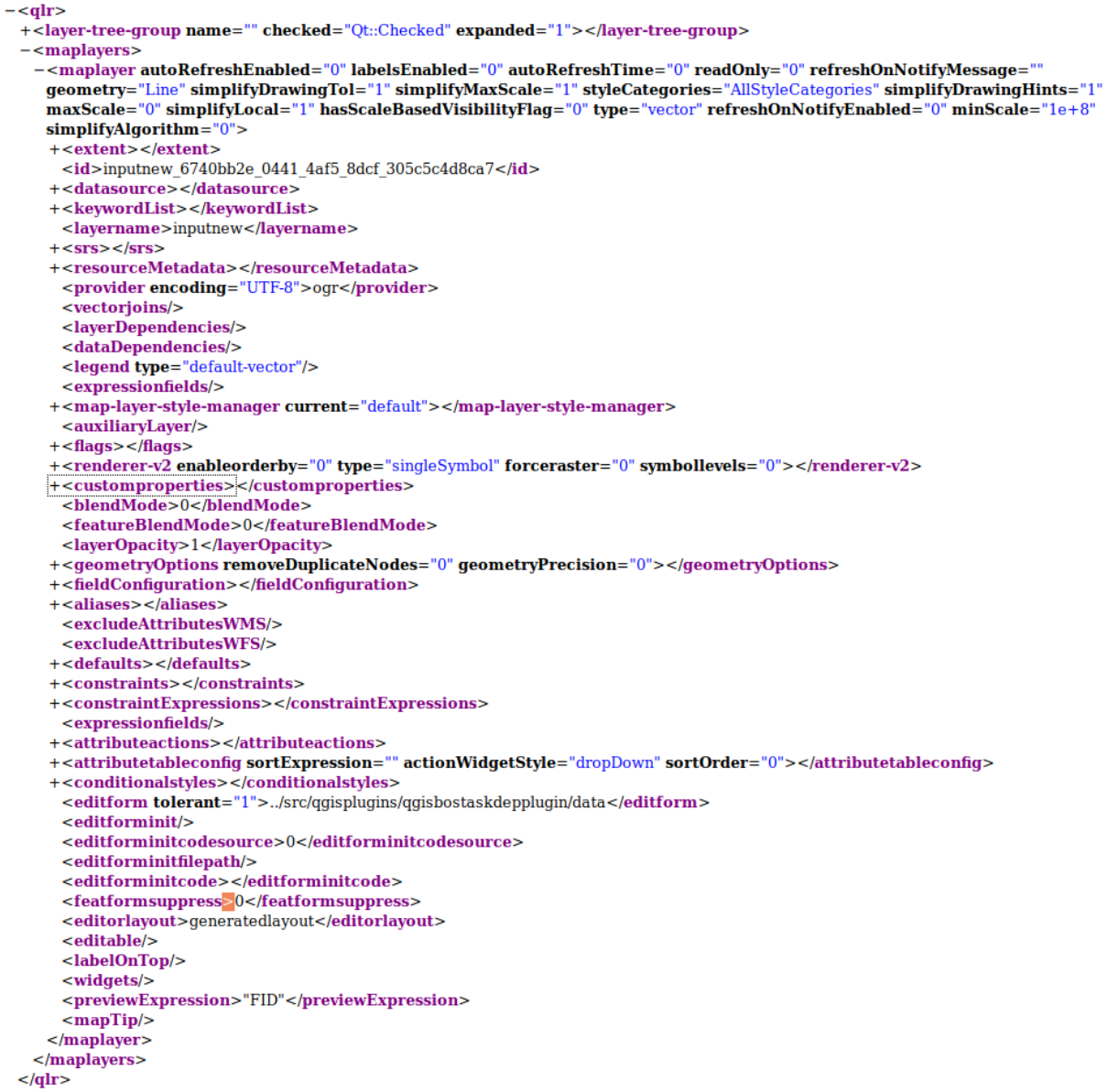


Fig. 26.3: The top level tags of a QLR file

26.1.3 QML - The QGIS Style File Format

QML is an XML format for storing layer styling.

A QML file contains all the information QGIS can handle for the rendering of feature geometries including symbol definitions, sizes and rotations, labelling, opacity and blend mode and more.

The figure below shows the top level tags of a QML file (with only `renderer_v2` and its `symbol` tag expanded).

```
-<qgis version="3.4.13-Madeira" styleCategories="AllStyleCategories" readOnly="0" maxScale="0"
 labelsEnabled="0" simplifyDrawingHints="1" hasScaleBasedVisibilityFlag="0" simplifyDrawingTol="1"
 simplifyMaxScale="1" minScale="1e+8" simplifyAlgorithm="0" simplifyLocal="1">
 +<flags></flags>
 -<renderer-v2 symbollevels="0" enableorderby="0" type="singleSymbol" forceraster="0">
   -<symbols>
     +<symbol clip_to_extent="1" name="0" alpha="1" type="fill" force_rhr="0"></symbol>
     </symbols>
     <rotation/>
     <sizescale/>
   </renderer-v2>
 +<customproperties></customproperties>
   <blendMode>0</blendMode>
   <featureBlendMode>0</featureBlendMode>
   <layerOpacity>1</layerOpacity>
 +<SingleCategoryDiagramRenderer diagramType="Histogram" attributeLegend="1">
   </SingleCategoryDiagramRenderer>
 +<DiagramLayerSettings priority="0" linePlacementFlags="18" dist="0" showAll="1" placement="1"
   obstacle="0" zIndex="0">
   </DiagramLayerSettings>
 +<geometryOptions removeDuplicateNodes="0" geometryPrecision="0"></geometryOptions>
 +<fieldConfiguration></fieldConfiguration>
 +<aliases></aliases>
   <excludeAttributesWMS/>
   <excludeAttributesWFS/>
 +<defaults></defaults>
 +<constraints></constraints>
 +<constraintExpressions></constraintExpressions>
   <expressionfields/>
 +<attributeactions></attributeactions>
 +<attributetableconfig actionWidgetStyle="dropDown" sortExpression="" sortOrder="0">
   </attributetableconfig>
 +<conditionalstyles></conditionalstyles>
   <editform tolerant="1"/>
   <editforminit/>
   <editforminitcodesource>0</editforminitcodesource>
   <editforminitfilepath/>
 +<editforminitcode></editforminitcode>
   <featformsuppress>0</featformsuppress>
   <editorlayout>generatedlayout</editorlayout>
 +<editable></editable>
 +<labelOnTop></labelOnTop>
   <widgets/>
   <previewExpression>ID</previewExpression>
   <mapTip/>
   <layerGeometryType>2</layerGeometryType>
 </qgis>
```

Fig. 26.4: The top level tags of a QML file (only the renderer_v2 tag with its symbol tag is expanded)

26.2 Appendix D: QGIS R script syntax

Contributed by Matteo Ghetta - funded by Scuola Superiore Sant'Anna

Writing R scripts in Processing is a bit tricky because of the special syntax.

A Processing R script starts with defining its **Inputs** and **Outputs**, each preceded with double hash characters (##).

Before the inputs, the group to place the algoritm in can be specified. If the group already exists, the algorithm will be added to it, if not, the group will be created. In the example below, the name of the group is *My group*:

```
##My Group=group
```

26.2.1 Inputs

All input data and parameters have to be specified. There are several types of inputs:

- vector: `##Layer = vector`
- vector field: `##F = Field Layer` (where *Layer* is the name of an input vector layer the field belongs to)
- raster: `##r = raster`
- table: `##t = table`
- number: `##Num = number`
- string: `##Str = string`
- boolean: `##Bol = boolean`
- elements in a dropdown menu. The items must be separated with semicolons `;`: `##type=selection point;lines;point+lines`

26.2.2 Outputs

As for the inputs, each output has to be defined at the beginning of the script:

- vector: `##output= output vector`
- raster: `##output= output raster`
- table: `##output= output table`
- plots: `##output_plots_to_html` (##showplots in earlier versions)
- To show R output in the *Result Viewer*, put > in front of the command whose output you would like to show.

26.2.3 Syntax Summary for QGIS R scripts

A number of input and output parameter types are offered.

Input parameter types

Parameter	Syntax example	Returning objects
vector	Layer = vector	sf object (or SpatialDataFrame object, if ##load_vector_using_rgdal is specified)
vector point	Layer = vector point	sf object (or SpatialDataFrame object, if ##load_vector_using_rgdal is specified)
vector line	Layer = vector line	sf object (or SpatialDataFrame object, if ##load_vector_using_rgdal is specified)
vector polygon	Layer = vector polygon	sf object (or SpatialPolygonsDataFrame object, if ##load_vector_using_rgdal is used)
multiple vector	Layer = multiple vector	sf object (or SpatialDataFrame objects if ##load_vector_using_rgdal is specified)
table	Layer = table	dataframe conversion from csv, default object of `read.csv` function
field	Field = Field Layer	name of the Field selected, e.g. `"Area"`
raster	Layer = raster	RasterBrick object, default object of `raster` package
multiple raster	Layer = multiple raster	RasterBrick objects, default object of `raster` package
number	N = number	integer or floating number chosen
string	S = string	string added in the box
longstring	LS = longstring	string added in the box, could be longer then the normal string
selection	S = selection first;second;third	string of the selected item chosen in the dropdown menu
crs	C = crs	string of the resulting CRS chosen, in the format: `"EPSG:4326"`
extent	E = extent	Extent object of the `raster` package, you can extract values as `E@xmin`
point	P = point	when clicked on the map, you have the coordinates of the point
file	F = file	path of the file chosen, e.g. “/home/matteo/file.txt”
folder	F = folder	path of the folder chosen, e.g. “/home/matteo/Downloads”

A parameter can be **OPTIONAL**, meaning that it can be ignored.

In order to set an input as optional, you add the string `optional` **before** the input, e.g:

```
##Layer = vector
##Field1 = Field Layer
##Field2 = optional Field Layer
```

Output parameter types

Parameter	Syntax example
vector	Output = output vector
raster	Output = output raster
table	Output = output table
file	Output = output file

Note

You can save plots as `png` from the *Processing Result Viewer*, or you can choose to save the plot directly from the algorithm interface.

Script body

The script body follows R syntax and the **Log** panel can help you if there is something wrong with your script.

Remember that you have to load all additional libraries in the script:

```
library(sp)
```

26.2.4 Examples

Example with vector output

Let's take an algorithm from the online collection that creates random points from the extent of an input layer:

```
##Point pattern analysis=group
##Layer=vector polygon
##Size=number 10
##Output=output vector
library(sp)
spatpoly = as(Layer, "Spatial")
pts=spsample(spatpoly,Size,type="random")
spdf=SpatialPointsDataFrame(pts, as.data.frame(pts))
Output=st_as_sf(spdf)
```

Explanation (per line in the script):

1. `Point pattern analysis` is the group of the algorithm
2. `Layer` is the input **vector** layer
3. `Size` is a **numerical** parameter with a default value of 10
4. `Output` is the **vector** layer that will be created by the algorithm
5. `library(sp)` loads the **sp** library
6. `spatpoly = as(Layer, "Spatial")` translate to an sp object
7. Call the `spsample` function of the `sp` library and run it using the input defined above (`Layer` and `Size`)
8. Create a *SpatialPointsDataFrame* object using the `SpatialPointsDataFrame` function
9. Create the output vector layer using the `st_as_sf` function

That's it! Just run the algorithm with a vector layer you have in the QGIS Legend, choose the number of random point. The resulting layer will be added to your map.

Example with raster output

The following script will perform basic ordinary kriging to create a raster map of interpolated values from a specified field of the input point vector layer by using the `autoKrige` function of the `automap` R package. It will first calculate the kriging model and then create a raster. The raster is created with the `raster` function of the raster R package:

```
##Basic statistics=group
##Layer=vector point
##Field=Field Layer
##Output=output raster
##load_vector_using_rgdal
require("automap")
require("sp")
require("raster")
```

(continues on next page)

(continued from previous page)

```
table=as.data.frame(Layer)
coordinates(table)= ~coords.x1+coords.x2
c = Layer[[Field]]
kriging_result = autoKrige(c~1, table)
prediction = raster(kriging_result$krige_output)
Output<-prediction
```

By using `##load_vector_using_rgdal`, the input vector layer will be made available as a `SpatialDataFrame` objects, so we avoid having to translate it from an `sf` object.

Example with table output

Let's edit the `Summary Statistics` algorithm so that the output is a table file (csv).

The script body is the following:

```
##Basic statistics=group
##Layer=vector
##Field=Field Layer
##Stat=Output table
Summary_statistics<-data.frame(rbind(
    sum(Layer[[Field]]),
    length(Layer[[Field]]),
    length(unique(Layer[[Field]])),
    min(Layer[[Field]]),
    max(Layer[[Field]]),
    max(Layer[[Field]])-min(Layer[[Field]]),
    mean(Layer[[Field]]),
    median(Layer[[Field]]),
    sd(Layer[[Field]])),
  row.names=c("Sum:","Count:","Unique values:","Minimum value:","Maximum value:",
↪"Range:","Mean value:","Median value:","Standard deviation:"))
colnames(Summary_statistics)<-c(Field)
Stat<-Summary_statistics
```

The third line specifies the **Vector Field** in input and the fourth line tells the algorithm that the output should be a table.

The last line will take the `Stat` object created in the script and convert it into a `csv` table.

Example with console output

We can use the previous example and instead of creating a table, print the result in the **Result Viewer**:

```
##Basic statistics=group
##Layer=vector
##Field=Field Layer
Summary_statistics<-data.frame(rbind(
sum(Layer[[Field]]),
length(Layer[[Field]]),
length(unique(Layer[[Field]])),
min(Layer[[Field]]),
max(Layer[[Field]]),
max(Layer[[Field]])-min(Layer[[Field]]),
mean(Layer[[Field]]),
median(Layer[[Field]]),
sd(Layer[[Field]])),row.names=c("Sum:","Count:","Unique values:","Minimum value:",
↪"Maximum value:","Range:","Mean value:","Median value:","Standard deviation:"))
colnames(Summary_statistics)<-c(Field)
>Summary_statistics
```

The script is exactly the same as the one above except for two edits:

1. no output specified (the fourth line has been removed)
2. the last line begins with >, telling Processing to make the object available through the result viewer

Example with plot

To create plots, you have to use the `##output_plots_to_html` parameter as in the following script:

```
##Basic statistics=group
##Layer=vector
##Field=Field Layer
##output_plots_to_html
####output_plots_to_html
qqnorm(Layer[[Field]])
qqline(Layer[[Field]])
```

The script uses a field (`Field`) of a vector layer (`Layer`) as input, and creates a *QQ Plot* (to test the normality of the distribution).

The plot is automatically added to the Processing *Result Viewer*.

26.3 Appendix E: QGIS Application Network Connections

This is a list of the pre-configured/automated network connections that QGIS makes. Some of the connections are user initiated because they require an action from the user before they take place, others happen automatically.

Name	Purpose	UX mode	Server	Information sent to server	Information stored on server
qgis.org					
Python API help	Browser PyQGIS documentation	User initiated	https://qgis.org/pyqgis/3.44/search	IP, QGIS version, OS	IP in server log
version.qgi					
New version check	Notify the user on new QGIS versions availability	Automatic	https://version.qgis.org	IP, QGIS version, OS	IP in server log
feed.qgi					
QGIS Feed	Retrieve QGIS news from the feed	Automatic	https://feed.qgis.org	IP, QGIS version, language code, last download timestamp, OS	IP in server log; QGIS version, OS and IP are aggregated and used to collect some statistics
plugins.qgis.					
Check for plugin updates	Notify the user about plugin updates	User initiated/automa (configurable)	https://plugins.qgis.org	IP, QGIS version, OS	IP in server log
Plugins list	Retrieve the list of plugins	User initiated	https://plugins.qgis.org	IP, QGIS version, OS	IP in server log
Plugin installation	Download and install a plugin package	User initiated	https://plugins.qgis.org	IP, QGIS version, OS	Increase plugin download counter by one
Styles	List user contributed styles	User initiated	https://hub.qgis.org/styles	IP, QGIS version, OS	Increase download counter by one
3rd party					
Terrain data	Produce a DEM for 3D views	User initiated	https://s3.amazonaws.com/elevation-tiles-prod/terrarium/{z}/{x}/{y}	IP, QGIS version, OS	see Amazon TOS
Google Map Geocode	Geocoding services	User initiated	https://maps.googleapis.com/maps/api/geocode/json	IP, QGIS version, OS	See google maps API TOS
Nominatim Geocode	Geocoding services	User initiated	https://nominatim.qgis.org/ui/search.html	IP, QGIS version, OS	
Geodetic grid	Add a new PROJ grid	User initiated	https://cdn.proj.org	IP, PROJ version	Access logs are permanently deleted after one day

CHAPTER

TWENTYSEVEN

LITERATURE AND WEB REFERENCES

GDAL-SOFTWARE-SUITE. Geospatial data abstraction library. https://gdal.org/en/latest, 2013.

GRASS-PROJECT. Geographic resource analysis support system. https://grass.osgeo.org, 2013.

NETELER, M., AND MITASOVA, H. Open source gis: A grass gis approach, 2008.

OGR-SOFTWARE-SUITE. Geospatial data abstraction library. https://gdal.org/en/latest, 2013.

OPEN-GEOSPATIAL-CONSORTIUM. Web map service (1.1.1) implementation specification. https://portal.ogc.org/files/?artifact_id=1081&version=1&format=pdf, 2002.

OPEN-GEOSPATIAL-CONSORTIUM. Web map service (1.3.0) implementation specification. https://portal.ogc.org/files/?artifact_id=14416&format=pdf, 2004.

POSTGIS-PROJECT. Spatial support for postgresql. http://www.refractions.net/products/postgis/, 2013.