



PyQGIS 3.10 developer cookbook

QGIS Project

2020年12月09日

目次

| | | |
|-------|---|----|
| 第 1 章 | はじめに | 1 |
| 1.1 | Python コンソールでのスクリプティング | 2 |
| 1.2 | Python プラグイン | 3 |
| 1.3 | Running Python code when QGIS starts | 3 |
| 1.3.1 | startup.py ファイル | 3 |
| 1.3.2 | The PYQGIS_STARTUP environment variable | 4 |
| 1.4 | Python アプリケーション | 4 |
| 1.4.1 | スタンドアロンスクリプトで PyQGIS を使用する | 4 |
| 1.4.2 | カスタムアプリケーションで PyQGIS を使用する | 5 |
| 1.4.3 | カスタムアプリケーションを実行する | 6 |
| 1.5 | Technical notes on PyQt and SIP | 7 |
| 第 2 章 | プロジェクトをロードする | 9 |
| 2.1 | Resolving bad paths | 10 |
| 第 3 章 | レイヤーをロードする | 13 |
| 3.1 | ベクターレイヤー | 13 |
| 3.2 | ラスターレイヤー | 17 |
| 3.3 | QgsProject instance | 19 |
| 第 4 章 | Accessing the Table Of Contents (TOC) | 21 |
| 4.1 | The QgsProject class | 21 |
| 4.2 | QgsLayerTreeGroup class | 22 |
| 第 5 章 | ラスターレイヤーを使う | 27 |
| 5.1 | レイヤーについて | 27 |
| 5.2 | レンダラー | 28 |
| 5.2.1 | シングルバンドラスター | 29 |
| 5.2.2 | マルチバンドラスター | 29 |
| 5.3 | 値の検索 | 30 |
| 第 6 章 | ベクターレイヤーを使う | 33 |
| 6.1 | 属性に関する情報を取得する | 34 |
| 6.2 | ベクターレイヤーの反復処理 | 35 |
| 6.3 | 地物の選択 | 36 |
| 6.3.1 | 属性にアクセスする | 37 |
| 6.3.2 | 選択された地物への反復処理 | 37 |

| | | |
|--------|--|----|
| 6.3.3 | 一部の地物への反復処理 | 37 |
| 6.4 | ベクターレイヤーを修正する | 38 |
| 6.4.1 | 地物の追加 | 39 |
| 6.4.2 | 地物の削除 | 40 |
| 6.4.3 | 地物の修正 | 40 |
| 6.4.4 | ベクターレイヤーを編集バッファで修正する | 40 |
| 6.4.5 | フィールドを追加または削除する | 42 |
| 6.5 | 空間索引を使う | 43 |
| 6.6 | ベクターレイヤを作る | 44 |
| 6.6.1 | From an instance of <code>QgsVectorFileWriter</code> | 44 |
| 6.6.2 | Directly from features | 46 |
| 6.6.3 | <code>QgsVectorLayer</code> クラスのインスタンスから作成する | 47 |
| 6.7 | ベクタレイヤの表現 (シンボロジ) | 49 |
| 6.7.1 | 単一シンボルレンダラー | 50 |
| 6.7.2 | 分類シンボルレンダラー | 51 |
| 6.7.3 | 段階シンボルレンダラー | 52 |
| 6.7.4 | シンボルの操作 | 53 |
| 6.7.5 | カスタムレンダラーの作成 | 57 |
| 6.8 | より詳しいトピック | 59 |
| 第 7 章 | ジオメトリの操作 | 61 |
| 7.1 | ジオメトリの構成 | 61 |
| 7.2 | ジオメトリにアクセス | 62 |
| 7.3 | ジオメトリの述語と操作 | 63 |
| 第 8 章 | 投影法サポート | 67 |
| 8.1 | 空間参照系 | 67 |
| 8.2 | CRS Transformation | 68 |
| 第 9 章 | マップキャンバスを使う | 71 |
| 9.1 | 地図キャンバスを埋め込む | 72 |
| 9.2 | ラバーバンドと頂点マーカー | 73 |
| 9.3 | 地図キャンバスで地図ツールを使用する | 74 |
| 9.4 | カスタム地図ツールを書く | 75 |
| 9.5 | カスタム地図キャンバスアイテムを書く | 77 |
| 第 10 章 | 地図のレンダリングと印刷 | 79 |
| 10.1 | 単純なレンダリング | 79 |
| 10.2 | 異なる CRS を持つレイヤーをレンダリングする | 80 |
| 10.3 | 印刷レイアウトを使用して出力する | 80 |
| 10.3.1 | Exporting the layout | 82 |
| 10.3.2 | Exporting a layout atlas | 82 |
| 第 11 章 | 式、フィルタ適用および値の算出 | 85 |
| 11.1 | 式を構文解析する | 86 |

| | | |
|--------|---|-----|
| 11.2 | 式を評価する | 87 |
| 11.2.1 | 基本的な式 | 87 |
| 11.2.2 | 地物に関わる式 | 87 |
| 11.2.3 | 式を使ってレイヤをフィルタする | 89 |
| 11.3 | 式エラーを扱う | 89 |
| 第 12 章 | 設定の読み込みと保存 | 91 |
| 第 13 章 | ユーザーとのコミュニケーション | 95 |
| 13.1 | Showing messages. The QgsMessageBar class | 95 |
| 13.2 | プロセスを表示する | 97 |
| 13.3 | ログを作成する | 99 |
| 13.3.1 | QgsMessageLog | 99 |
| 13.3.2 | The python built in logging module | 100 |
| 第 14 章 | 認証インフラストラクチャ | 103 |
| 14.1 | 前書き | 104 |
| 14.2 | 用語集 | 104 |
| 14.3 | エントリーポイント QgsAuthManager | 105 |
| 14.3.1 | マネージャを初期化し、マスターパスワードを設定する | 105 |
| 14.3.2 | 認証データベースに新しい認証構成項目を設定する | 105 |
| 14.3.3 | Remove an entry from authdb | 107 |
| 14.3.4 | QgsAuthManager に authcfg 展開を残す | 107 |
| 14.4 | 認証インフラストラクチャを使用するようにプラグインを適応させる | 108 |
| 14.5 | 認証の GUI | 109 |
| 14.5.1 | 資格情報を選択するための GUI | 109 |
| 14.5.2 | 認証エディタの GUI | 110 |
| 14.5.3 | 認証局エディタの GUI | 110 |
| 第 15 章 | タスク - バックグラウンドで重い仕事をする | 113 |
| 15.1 | はじめに | 113 |
| 15.2 | 例 | 114 |
| 15.2.1 | QgsTask を拡張する | 114 |
| 15.2.2 | 関数からのタスク | 117 |
| 15.2.3 | プロセッシングアルゴリズムからのタスク | 119 |
| 第 16 章 | Python プラグインを開発する | 121 |
| 16.1 | Python プラグインを構成する | 121 |
| 16.1.1 | プラグインを書く | 122 |
| 16.1.2 | プラグインの内容 | 123 |
| 16.1.3 | Documentation | 129 |
| 16.1.4 | Translation | 129 |
| 16.1.5 | Tips and Tricks | 132 |
| 16.2 | 短いコード | 132 |
| 16.2.1 | How to call a method by a key shortcut | 133 |

| | | |
|---------------|---|------------|
| 16.2.2 | How to toggle Layers | 133 |
| 16.2.3 | How to access attribute table of selected features | 133 |
| 16.2.4 | Interface for plugin in the options dialog | 134 |
| 16.3 | Using Plugin Layers | 135 |
| 16.3.1 | Subclassing QgsPluginLayer | 135 |
| 16.4 | プラグインを書いてデバッグするための IDE 設定 | 137 |
| 16.4.1 | Useful plugins for writing Python plugins | 137 |
| 16.4.2 | A note on configuring your IDE on Linux and Windows | 138 |
| 16.4.3 | Debugging using Pyscripter IDE (Windows) | 138 |
| 16.4.4 | Debugging using Eclipse and PyDev | 139 |
| 16.4.5 | Debugging with PyCharm on Ubuntu with a compiled QGIS | 143 |
| 16.4.6 | PDB を利用してデバッグする | 145 |
| 16.5 | Releasing your plugin | 145 |
| 16.5.1 | Metadata and names | 146 |
| 16.5.2 | Code and help | 146 |
| 16.5.3 | Official Python plugin repository | 147 |
| 第 17 章 | プロセッシングプラグインを書く | 149 |
| 17.1 | イチから作る | 149 |
| 17.2 | プラグインをアップデートする | 150 |
| 第 18 章 | ネットワーク分析ライブラリ | 153 |
| 18.1 | 一般情報 | 153 |
| 18.2 | グラフを構築する | 154 |
| 18.3 | グラフ分析 | 156 |
| 18.3.1 | 最短経路を見つける | 159 |
| 18.3.2 | 利用可能領域 | 161 |
| 第 19 章 | QGIS Server and Python | 163 |
| 19.1 | はじめに | 163 |
| 19.2 | Server API basics | 164 |
| 19.3 | Standalone or embedding | 164 |
| 19.4 | Server plugins | 165 |
| 19.4.1 | Server filter plugins | 165 |
| 19.4.2 | Custom services | 175 |
| 19.4.3 | Custom APIs | 176 |
| 第 20 章 | PyQGIS チートシート | 179 |
| 20.1 | ユーザーインターフェース | 179 |
| 20.2 | 設定 | 179 |
| 20.3 | ツールバー | 180 |
| 20.4 | メニュー | 180 |
| 20.5 | キャンバス | 180 |
| 20.6 | レイヤー | 181 |
| 20.7 | Table of contents | 185 |

| | | |
|-------|------------------------|-----|
| 20.8 | Advanced TOC | 185 |
| 20.9 | プロセシングアルゴリズム | 189 |
| 20.10 | 装飾類 | 190 |
| 20.11 | Composer | 192 |
| 20.12 | 出典 | 192 |

第 1 章

はじめに

この文書は、チュートリアルとリファレンスガイドの両方を意図して書かれています。可能な事例をすべて網羅しているわけではありませんが、主要な機能を概観するには役に立つはずで

- *Python* コンソールでのスクリプティング
- *Python* プラグイン
- *Running Python code when QGIS starts*
 - *startup.py* ファイル
 - *The PYQGIS_STARTUP environment variable*
- *Python* アプリケーション
 - スタンドアロンスクリプトで *PyQGIS* を使用する
 - カスタムアプリケーションで *PyQGIS* を使用する
 - カスタムアプリケーションを実行する
- *Technical notes on PyQt and SIP*

Python のサポートは QGIS 0.9 で初めて導入されました。デスクトップ版 QGIS で Python を利用するにはいくつかの方法があります (この後のセクションで説明します)。

- QGIS に付属する Python コンソールでのコマンドの実行
- プラグインの作成と利用
- QGIS 起動時の Python コードの自動実行
- プロセッシングアルゴリズムの作成
- QGIS の「式」で使用する関数の作成
- QGIS API を利用するカスタムアプリケーションの作成

QGIS サーバでも Python プラグインを含む Python バインディングが提供されています (参照 [QGIS Server and Python](#)) そして Python バインディングを使うことによって Python アプリケーションに QGIS サーバを埋め込むこともできます。

QGIS ライブラリのクラスのドキュメントである [complete QGIS API リファレンス](#) があります。Python 用の QGIS API (`pyqgis`) は C++ API とほぼ同じです。

頻出タスクの実行方法を学習するには、[プラグインのリポジトリ](#) から既存のプラグインをダウンロードして、そのコードを研究するのがよい方法です。

1.1 Python コンソールでのスクリプティング

QGIS はスクリプティングのための Python console を内蔵しています。プラグイン `Python` コンソールメニューから開くことができます。

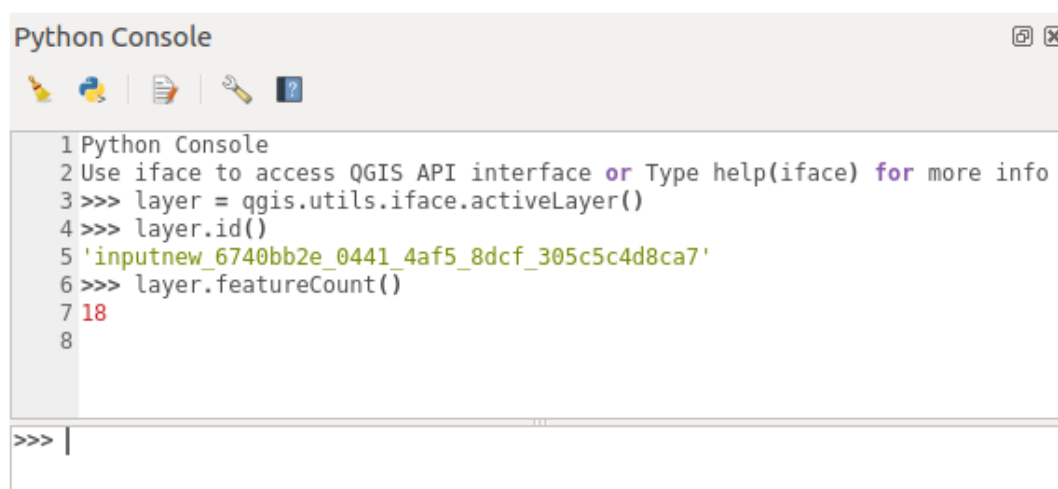


図 1.1 QGIS Python コンソール

上のスクリーンショットは、レイヤリストで現在選択されているレイヤを取得し、その ID を表示し、オプションでベクタレイヤの場合は地物数を表示する方法を示しています。QGIS 環境とのやりとりのために、`QgisInterface` のインスタンスである `iface` 変数があります。このインターフェースにより、地図 キャンパス、メニュー、ツールバー、および QGIS アプリケーションのその他の部分へのアクセスが可能になります。

For user convenience, the following statements are executed when the console is started (in the future it will be possible to set further initial commands)

```
from qgis.core import *
import qgis.utils
```

For those which use the console often, it may be useful to set a shortcut for triggering the console (within *Settings Keyboard shortcuts...*)

1.2 Python プラグイン

QGIS の機能はプラグインを使って拡張することができます。プラグインは Python で書くことができます。C++ プラグインに比較しての主要な利点は配布の単純さ（プラットフォームごとにコンパイルする必要がありません）と開発の容易さです。

様々な機能をカバーする多くのプラグインが Python サポートが導入されてから書かれました。プラグインのインストーラは Python プラグインの取得、アップグレード、削除を簡単に行えます。プラグインとプラグイン開発の詳細については、[Python プラグイン](#) ページを参照してください。

Python でプラグインを作るのはとても簡単です。詳細は [Python プラグインを開発する](#) を見てください。

注釈: Python プラグインは QGIS サーバーでも使うことができます。より詳しくは [QGIS Server and Python](#) をご覧ください。

1.3 Running Python code when QGIS starts

QGIS を起動するたびに Python コードを実行するには、2 つの異なる方法があります。

1. Creating a startup.py script
2. Setting the PYQGIS_STARTUP environment variable to an existing Python file

1.3.1 startup.py ファイル

Every time QGIS starts, the user's Python home directory

- Linux: `./local/share/QGIS/QGIS3`
- Windows: `AppData\Roaming\QGIS\QGIS3`
- macOS: `Library/Application Support/QGIS/QGIS3`

is searched for a file named `startup.py`. If that file exists, it is executed by the embedded Python interpreter.

注釈: デフォルトパスはオペレーティングシステムによって異なります。ご自身の環境に合ったパスを見つけるには、Python コンソールを開いて `QStandardPaths.standardLocations(QStandardPaths.AppDataLocation)` を実行し、デフォルトディレクトリのリストを見てください。

1.3.2 The PYQGIS_STARTUP environment variable

You can run Python code just before QGIS initialization completes by setting the PYQGIS_STARTUP environment variable to the path of an existing Python file.

This code will run before QGIS initialization is complete. This method is very useful for cleaning sys.path, which may have undesirable paths, or for isolating/loading the initial environment without requiring a virtual environment, e.g. homebrew or MacPorts installs on Mac.

1.4 Python アプリケーション

It is often handy to create scripts for automating processes. With PyQGIS, this is perfectly possible --- import the `qgis.core` module, initialize it and you are ready for the processing.

Or you may want to create an interactive application that uses GIS functionality --- perform measurements, export a map as PDF, ... The `qgis.gui` module provides various GUI components, most notably the map canvas widget that can be incorporated into the application with support for zooming, panning and/or any further custom map tools.

PyQGIS custom applications or standalone scripts must be configured to locate the QGIS resources, such as projection information and providers for reading vector and raster layers. QGIS Resources are initialized by adding a few lines to the beginning of your application or script. The code to initialize QGIS for custom applications and standalone scripts is similar. Examples of each are provided below.

注釈: Do *not* use `qgis.py` as a name for your script. Python will not be able to import the bindings as the script's name will shadow them.

1.4.1 スタンドアロンスクリプトで PyQGIS を使用する

To start a standalone script, initialize the QGIS resources at the beginning of the script:

```
1 from qgis.core import *
2
3 # Supply path to qgis install location
4 QgsApplication.setPrefixPath("/path/to/qgis/installation", True)
5
6 # Create a reference to the QgsApplication. Setting the
7 # second argument to False disables the GUI.
8 qgs = QgsApplication([], False)
9
10 # Load providers
11 qgs.initQgis()
12
13 # Write your code here to load some layers, use processing
```

(次のページに続く)

(前のページからの続き)

```

14 # algorithms, etc.
15
16 # Finally, exitQgis() is called to remove the
17 # provider and layer registries from memory
18 qgs.exitQgis()

```

First we import the `qgis.core` module and configure the prefix path. The prefix path is the location where QGIS is installed on your system. It is configured in the script by calling the `setPrefixPath` method. The second argument of `setPrefixPath` is set to `True`, specifying that default paths are to be used.

The QGIS install path varies by platform; the easiest way to find it for your system is to use the *Python コンソールでのスクリプティング* from within QGIS and look at the output from running `QgsApplication.prefixPath()`.

After the prefix path is configured, we save a reference to `QgsApplication` in the variable `qgs`. The second argument is set to `False`, specifying that we do not plan to use the GUI since we are writing a standalone script. With `QgsApplication` configured, we load the QGIS data providers and layer registry by calling the `qgs.initQgis()` method. With QGIS initialized, we are ready to write the rest of the script. Finally, we wrap up by calling `qgs.exitQgis()` to remove the data providers and layer registry from memory.

1.4.2 カスタムアプリケーションで PyQGIS を使用する

The only difference between *スタンドアロンスクリプトで PyQGIS を使用する* and a custom PyQGIS application is the second argument when instantiating the `QgsApplication`. Pass `True` instead of `False` to indicate that we plan to use a GUI.

```

1 from qgis.core import *
2
3 # Supply the path to the qgis install location
4 QgsApplication.setPrefixPath("/path/to/qgis/installation", True)
5
6 # Create a reference to the QgsApplication.
7 # Setting the second argument to True enables the GUI. We need
8 # this since this is a custom application.
9
10 qgs = QgsApplication([], True)
11
12 # load providers
13 qgs.initQgis()
14
15 # Write your code here to load some layers, use processing
16 # algorithms, etc.
17
18 # Finally, exitQgis() is called to remove the
19 # provider and layer registries from memory
20 qgs.exitQgis()

```

Now you can work with the QGIS API - load layers and do some processing or fire up a GUI with a map canvas.

The possibilities are endless :-)

1.4.3 カスタムアプリケーションを実行する

You need to tell your system where to search for QGIS libraries and appropriate Python modules if they are not in a well-known location - otherwise Python will complain:

```
>>> import qgis.core
ImportError: No module named qgis.core
```

This can be fixed by setting the PYTHONPATH environment variable. In the following commands, `<qgispath>` should be replaced with your actual QGIS installation path:

- on Linux: **export PYTHONPATH=/`<qgispath>`/share/qgis/python**
- on Windows: **set PYTHONPATH=c:\code<qgispath>\python**
- on macOS: **export PYTHONPATH=/`<qgispath>`/Contents/Resources/python**

Now, the path to the PyQGIS modules is known, but they depend on the `qgis_core` and `qgis_gui` libraries (the Python modules serve only as wrappers). The path to these libraries may be unknown to the operating system, and then you will get an import error again (the message might vary depending on the system):

```
>>> import qgis.core
ImportError: libqgis_core.so.3.2.0: cannot open shared object file:
  No such file or directory
```

Fix this by adding the directories where the QGIS libraries reside to the search path of the dynamic linker:

- on Linux: **export LD_LIBRARY_PATH=/`<qgispath>`/lib**
- on Windows: **set PATH=C:\code<qgispath>\bin;C:\code<qgispath>\apps\code<qgisrelease>\bin;%PATH%** where `<qgisrelease>` should be replaced with the type of release you are targeting (eg, `qgis-ltr`, `qgis`, `qgis-dev`)

これらのコマンドはブートストラップスクリプトに入れておくことができます。PyQGIS を使ったカスタムアプリケーションを配布するには、これらの二つの方法が可能でしょう:

- require the user to install QGIS prior to installing your application. The application installer should look for default locations of QGIS libraries and allow the user to set the path if not found. This approach has the advantage of being simpler, however it requires the user to do more steps.
- アプリケーションと一緒に QGIS のパッケージを配布する方法です。アプリケーションのリリースにはいろいろやる必要があるし、パッケージも大きくなりますが、ユーザーが追加ソフトウェアをダウンロードしてインストールするという負荷を避けられるでしょう。

The two deployment models can be mixed. You can provide a standalone applications on Windows and macOS, but for Linux leave the installation of GIS up to the user and his package manager.

1.5 Technical notes on PyQt and SIP

We've decided for Python as it's one of the most favoured languages for scripting. PyQGIS bindings in QGIS 3 depend on SIP and PyQt5. The reason for using SIP instead of the more widely used SWIG is that the QGIS code depends on Qt libraries. Python bindings for Qt (PyQt) are done using SIP and this allows seamless integration of PyQGIS with PyQt.

The code snippets on this page need the following imports if you're outside the pyqgis console:

```
1 from qgis.core import (  
2     QgsProject,  
3     QgsPathResolver  
4 )  
5  
6 from qgis.gui import (  
7     QgsLayerTreeMapCanvasBridge,  
8 )
```


第2章

プロジェクトをロードする

時々プラグインから、または(しばしば)スタンドアロンの QGIS Python アプリケーションを開発する時に既存のプロジェクトをロードする必要があります(参照: [Python アプリケーション](#))。

プロジェクトを現在の QGIS アプリケーションにロードするには、`QgsProject` クラスのインスタンスを作成する必要があります。これはシングルトンクラスなので、それを行うには `instance()` メソッドを使わなければなりません。 `read()` メソッドを呼び出して、読み込むプロジェクトのパスを渡すことができます。

```
1 # If you are not inside a QGIS console you first need to import
2 # qgis and PyQt classes you will use in this script as shown below:
3 from qgis.core import QgsProject
4 # Get the project instance
5 project = QgsProject.instance()
6 # Print the current project file name (might be empty in case no projects have
7 #   ↳ been loaded)
8 # print(project.fileName())
9
10 # Load another project
11 import os
12 print(os.getcwd())
13 project.read('testdata/01_project.qgs')
14 print(project.fileName())
```

```
...
testdata/01_project.qgs
```

プロジェクトに変更(たとえばレイヤーの追加や削除)を加え、その変更を保存する必要がある場合は、プロジェクトインスタンスの `write()` メソッドを呼び出します。 `write()` メソッドにパスを指定すれば、プロジェクトを新しい場所に保存することもできます。

```
# Save the project to the same
project.write()
# ... or to a new file
project.write('testdata/my_new_qgis_project.qgs')
```

`read()` と `write()` の両方の関数は操作が成功したかどうかをチェックするために使用できるブール値を返します。

注釈: QGIS スタンドアロンアプリケーションを作成している場合は、ロードされたプロジェクトをキャンバスと同期させるために、`QgsLayerTreeMapCanvasBridge` を以下の例のようにインスタンス化する必要があります:

```
bridge = QgsLayerTreeMapCanvasBridge( \
    QgsProject.instance().layerTreeRoot(), canvas)
# Now you can safely load your project and see it in the canvas
project.read('testdata/my_new_qgis_project.qgs')
```

```
...
```

2.1 Resolving bad paths

It can happen that layers loaded in the project are moved to another location. When the project is loaded again all the layer paths are broken.

The `QgsPathResolver` class with the `setPathPreprocessor()` allows setting a custom path pre-processor function, which allows for manipulation of paths and data sources prior to resolving them to file references or layer sources.

The processor function must accept a single string argument (representing the original file path or data source) and return a processed version of this path.

The path pre-processor function is called **before** any bad layer handler.

Some use cases:

1. replace an outdated path:

```
def my_processor(path):
    return path.replace('c:/Users/ClintBarton/Documents/Projects', 'x:/
↳Projects/')

QgsPathResolver.setPathPreprocessor(my_processor)
```

2. replace a database host address with a new one:

```
def my_processor(path):
    return path.replace('host=10.1.1.115', 'host=10.1.1.116')

QgsPathResolver.setPathPreprocessor(my_processor)
```

3. replace stored database credentials with new ones:

```
1 def my_processor(path):
2     path= path.replace("user='gis_team'", "user='team_awesome'")
```

(次のページに続く)

(前のページからの続き)

```
3     path = path.replace("password='cats'", "password='g7as!m*')  
4     return path  
5  
6 QgsPathResolver.setPathPreprocessor(my_processor)
```


第3章

レイヤーをロードする

The code snippets on this page need the following imports:

```
import os # This is is needed in the pyqgis console also
from qgis.core import (
    QgsVectorLayer
)
```

- ベクターレイヤー
- ラスターレイヤー
- *QgsProject instance*

データのレイヤーを開きましょう。QGIS はベクターおよびラスターレイヤーを認識できます。加えてカスタムレイヤータイプを利用することもできますが、それについてここでは述べません。

3.1 ベクターレイヤー

To create and add a vector layer instance to the project, specify the layer's data source identifier, name for the layer and provider's name:

```
1 # get the path to the shapefile e.g. /home/project/data/ports.shp
2 path_to_airports_layer = "testdata/airports.shp"
3
4 # The format is:
5 # vlayer = QgsVectorLayer(data_source, layer_name, provider_name)
6
7 vlayer = QgsVectorLayer(path_to_airports_layer, "Airports layer", "ogr")
8 if not vlayer.isValid():
9     print("Layer failed to load!")
10 else:
11     QgsProject.instance().addMapLayer(vlayer)
```

データソース識別子は文字列でそれぞれのデータプロバイダーを表します。レイヤー名はレイヤーリストウィジェットの中で使われます。レイヤーが正常にロードされたかどうかをチェックすることは重要です。正しくロードされていない場合は不正なレイヤーインスタンスが返ります。

geopackage ベクターレイヤなら次のようになります。

```
1 # get the path to a geopackage e.g. /usr/share/qgis/resources/data/world_map.gpkg
2 path_to_gpkg = os.path.join(QgsApplication.pkgDataPath(), "resources", "data",
   ↳ "world_map.gpkg")
3 # append the layername part
4 gpkg_countries_layer = path_to_gpkg + "|layername=countries"
5 # e.g. gpkg_places_layer = "/usr/share/qgis/resources/data/world_map.
   ↳ gpkg|layername=countries"
6 vlayer = QgsVectorLayer(gpkg_countries_layer, "Countries layer", "ogr")
7 if not vlayer.isValid():
8     print("Layer failed to load!")
9 else:
10    QgsProject.instance().addMapLayer(vlayer)
```

The quickest way to open and display a vector layer in QGIS is the `addVectorLayer()` method of the `QgisInterface`:

```
vlayer = iface.addVectorLayer(path_to_airports_layer, "Airports layer", "ogr")
if not vlayer:
    print("Layer failed to load!")
```

This creates a new layer and adds it to the current QGIS project (making it appear in the layer list) in one step. The function returns the layer instance or `None` if the layer couldn't be loaded.

以下のリストはベクターデータプロバイダーを使って様々なデータソースにアクセスする方法が記述されています。

- OGR library (Shapefile and many other file formats) --- data source is the path to the file:
 - for Shapefile:

```
vlayer = QgsVectorLayer("testdata/airports.shp", "layer_name_you_like",
   ↳ "ogr")
QgsProject.instance().addMapLayer(vlayer)
```

- for dxf (note the internal options in data source uri):

```
uri = "testdata/sample.dxf|layername=entities|geometrytype=Polygon"
vlayer = QgsVectorLayer(uri, "layer_name_you_like", "ogr")
QgsProject.instance().addMapLayer(vlayer)
```

- PostGIS database - data source is a string with all information needed to create a connection to PostgreSQL database.

`QgsDataSourceUri` class can generate this string for you. Note that QGIS has to be compiled with Postgres support, otherwise this provider isn't available:

```

1 uri = QgsDataSourceUri()
2 # set host name, port, database name, username and password
3 uri.setConnection("localhost", "5432", "dbname", "johny", "xxx")
4 # set database schema, table name, geometry column and optionally
5 # subset (WHERE clause)
6 uri.setDataSource("public", "roads", "the_geom", "cityid = 2643", "primary_
↳key_field")
7
8 vlayer = QgsVectorLayer(uri.uri(False), "layer name you like", "postgres")

```

注釈: `uri.uri(False)` に渡される `False` 引数は、認証構成パラメーターの拡張を防ぎます。もし何も認証構成を使用していなければ、この引数は何の違いもありません。

- CSV or other delimited text files --- to open a file with a semicolon as a delimiter, with field "x" for X coordinate and field "y" for Y coordinate you would use something like this:

```

uri = "file://{}/testdata/delimited_xy.csv?delimiter={}&xField={}&yField={}".
↳format(os.getcwd(), ";", "x", "y")
vlayer = QgsVectorLayer(uri, "layer name you like", "delimitedtext")
QgsProject.instance().addMapLayer(vlayer)

```

注釈: プロバイダーの文字列は URL として構造化されているので、パスには `file://` という接頭辞を付ける必要があります。また、`x` や `y` フィールドの代わりに WKT (well-known text) 形式のジオメトリを使用でき、座標参照系を指定できます。例えば:

```

uri = "file:///some/path/file.csv?delimiter={}&crs=epsg:4723&wktField={}".
↳format(";", "shape")

```

- GPX ファイル---「GPX」データプロバイダーは、GPX ファイルからトラック、ルートやウェイポイントを読み込みます。ファイルを開くには、タイプ(トラック/ルート/ウェイポイント)を URL の一部として指定する必要があります:

```

uri = "testdata/layers.gpx?type=track"
vlayer = QgsVectorLayer(uri, "layer name you like", "gpx")
QgsProject.instance().addMapLayer(vlayer)

```

- SpatiaLite database --- Similarly to PostGIS databases, `QgsDataSourceUri` can be used for generation of data source identifier:

```

1 uri = QgsDataSourceUri()
2 uri.setDatabase('/home/martin/test-2.3.sqlite')
3 schema = ''
4 table = 'Towns'
5 geom_column = 'Geometry'

```

(次のページに続く)

(前のページからの続き)

```

6 uri.setDataSource(schema, table, geom_column)
7
8 display_name = 'Towns'
9 vlayer = QgsVectorLayer(uri.uri(), display_name, 'spatialite')
10 QgsProject.instance().addMapLayer(vlayer)

```

- MySQL の WKB ベースジオメトリ、OGR 経由 --- データソースはテーブルへの接続文字列です。

```

uri = "MySQL:dbname,host=localhost,port=3306,user=root,
↳password=xxx|layername=my_table"
vlayer = QgsVectorLayer( uri, "my table", "ogr" )
QgsProject.instance().addMapLayer(vlayer)

```

- WFS connection: the connection is defined with a URI and using the WFS provider:

```

uri = "https://demo.geo-solutions.it/geoserver/ows?service=WFS&version=1.1.0&
↳request=GetFeature&typename=geosolutions:regioni"
vlayer = QgsVectorLayer(uri, "my wfs layer", "WFS")
QgsProject.instance().addMapLayer(vlayer)

```

URI は標準の `urllib` ライブラリを使用して作成できます。

```

1 import urllib
2
3 params = {
4     'service': 'WFS',
5     'version': '1.1.0',
6     'request': 'GetFeature',
7     'typename': 'geosolutions:regioni',
8     'srsname': "EPSG:4326"
9 }
10 uri2 = 'https://demo.geo-solutions.it/geoserver/ows?' + urllib.parse.
↳unquote(urllib.parse.urlencode(params))

```

注釈: You can change the data source of an existing layer by calling `setDataSource()` on a `QgsVectorLayer` instance, as in the following example:

```

1 uri = "https://demo.geo-solutions.it/geoserver/ows?service=WFS&version=1.1.0&
↳request=GetFeature&typename=geosolutions:regioni"
2 provider_options = QgsDataProvider.ProviderOptions()
3 # Use project's transform context
4 provider_options.transformContext = QgsProject.instance().transformContext()
5 vlayer.setDataSource(uri, "layer name you like", "WFS", provider_options)
6 QgsProject.instance().addMapLayer(vlayer)

```


3.2 ラスターレイヤー

For accessing raster files, GDAL library is used. It supports a wide range of file formats. In case you have troubles with opening some files, check whether your GDAL has support for the particular format (not all formats are available by default). To load a raster from a file, specify its filename and display name:

```

1 # get the path to a tif file e.g. /home/project/data/srtm.tif
2 path_to_tif = "qgis-projects/python_cookbook/data/srtm.tif"
3 rlayer = QgsRasterLayer(path_to_tif, "SRTM layer name")
4 if not rlayer.isValid():
5     print("Layer failed to load!")

```

To load a raster from a geopackage:

```

1 # get the path to a geopackage e.g. /home/project/data/data.gpkg
2 path_to_gpkg = os.path.join(os.getcwd(), "testdata", "sublayers.gpkg")
3 # gpkg_raster_layer = "GPKG:/home/project/data/data.gpkg:srtm"
4 gpkg_raster_layer = "GPKG:" + path_to_gpkg + ":srtm"
5
6 rlayer = QgsRasterLayer(gpkg_raster_layer, "layer name you like", "gdal")
7
8 if not rlayer.isValid():
9     print("Layer failed to load!")

```

Similarly to vector layers, raster layers can be loaded using the `addRasterLayer` function of the `QgisInterface` object:

```
iface.addRasterLayer(path_to_tif, "layer name you like")
```

This creates a new layer and adds it to the current project (making it appear in the layer list) in one step.

To load a PostGIS raster:

PostGIS rasters, similar to PostGIS vectors, can be added to a project using a URI string. It is efficient to create a dictionary of strings for the database connection parameters. The dictionary is then loaded into an empty URI, before adding the raster. Note that `None` should be used when it is desired to leave the parameter blank:

```

1 uri_config = {#
2 # a dictionary of database parameters
3 'dbname':'gis_db', # The PostgreSQL database to connect to.
4 'host':'localhost', # The host IP address or localhost.
5 'port':'5432', # The port to connect on.
6 'sslmode':'disable', # The SSL/TLS mode. Options: allow, disable, prefer,
↳require, verify-ca, verify-full
7 # user and password are not needed if stored in the authcfg or service
8 'user':None, # The PostgreSQL user name, also accepts the new WFS
↳provider naming.
9 'password':None, # The PostgreSQL password for the user.
10 'service':None, # The PostgreSQL service to be used for connection to the
↳database.

```

(次のページに続く)

```

11 'authcfg':'QconfigId', # The QGIS authentication database ID holding connection_
    ↳details.
12 # table and raster column details
13 'schema':'public', # The database schema that the table is located in.
14 'table':'my_rasters', # The database table to be loaded.
15 'column':'rast', # raster column in PostGIS table
16 'mode':'2', # GDAL 'mode' parameter, 2 union raster tiles, 1 separate_
    ↳tiles (may require user input)
17 'sql':None, # An SQL WHERE clause.
18 'key':None, # A key column from the table.
19 'srid':None, # A string designating the SRID of the coordinate_
    ↳reference system.
20 'estimatedmetadata':'False', # A boolean value telling if the metadata is_
    ↳estimated.
21 'type':None, # A WKT string designating the WKB Type.
22 'selectatid':None, # Set to True to disable selection by feature ID.
23 'options':None, # other PostgreSQL connection options not in this list.
24 'connect_timeout':None,
25 'hostaddr':None,
26 'driver':None,
27 'tty':None,
28 'requiresssl':None,
29 'krbsrvname':None,
30 'gsslib':None,
31 }
32 # configure the URI string with the dictionary
33 uri = QgsDataSourceUri()
34 for param in uri_config:
35     if (uri_config[param] != None):
36         uri.setParam(param, uri_config[param]) # add parameters to the URI
37
38 # the raster can now be loaded into the project using the URI string and GDAL data_
    ↳provider
39 rlayer = iface.addRasterLayer('PG: ' + uri.uri(False), "raster layer name", "gdal")

```

ラスターレイヤーも WCS サービスから作成できます。

```

layer_name = 'nurc:mosaic'
uri = "https://demo.geo-solutions.it/geoserver/ows?identifier={}".format(layer_
    ↳name)
rlayer = QgsRasterLayer(uri, 'my wcs layer', 'wcs')

```

Here is a description of the parameters that the WCS URI can contain:

WCS URI is composed of **key=value** pairs separated by &. It is the same format like query string in URL, encoded the same way. `QgsDataSourceUri` should be used to construct the URI to ensure that special characters are encoded properly.

- **url** (required) : WCS Server URL. Do not use VERSION in URL, because each version of WCS is using different parameter name for **GetCapabilities** version, see param version.

- **identifier** (required) : Coverage name
- **time** (optional) : time position or time period (beginPosition/endPosition[/timeResolution])
- **format** (optional) : Supported format name. Default is the first supported format with tif in name or the first supported format.
- **crs** (optional) : CRS in form AUTHORITY:ID, e.g. EPSG:4326. Default is EPSG:4326 if supported or the first supported CRS.
- **username** (optional) : Username for basic authentication.
- **password** (optional) : Password for basic authentication.
- **IgnoreGetMapUrl** (optional, hack) : If specified (set to 1), ignore GetCoverage URL advertised by GetCapabilities. May be necessary if a server is not configured properly.
- **InvertAxisOrientation** (optional, hack) : If specified (set to 1), switch axis in GetCoverage request. May be necessary for geographic CRS if a server is using wrong axis order.
- **IgnoreAxisOrientation** (optional, hack) : If specified (set to 1), do not invert axis orientation according to WCS standard for geographic CRS.
- **cache** (optional) : cache load control, as described in QNetworkRequest::CacheLoadControl, but request is resend as PreferCache if failed with AlwaysCache. Allowed values: AlwaysCache, PreferCache, PreferNetwork, AlwaysNetwork. Default is AlwaysCache.

別の方法としては、WMS サーバーからラスターレイヤーを読み込むことができます。しかし現在では、API から GetCapabilities レスポンスにアクセスすることはできません---どのレイヤが必要か知っている必要があります。

```
urlWithParams = "crs=EPSG:4326&format=image/png&layers=tasmania&styles&url=https://
→demo.geo-solutions.it/geoserver/ows"
rlayer = QgsRasterLayer(urlWithParams, 'some layer name', 'wms')
if not rlayer.isValid():
    print("Layer failed to load!")
```

3.3 QgsProject instance

If you would like to use the opened layers for rendering, do not forget to add them to the `QgsProject` instance. The `QgsProject` instance takes ownership of layers and they can be later accessed from any part of the application by their unique ID. When the layer is removed from the project, it gets deleted, too. Layers can be removed by the user in the QGIS interface, or via Python using the `removeMapLayer()` method.

Adding a layer to the current project is done using the `addMapLayer()` method:

```
QgsProject.instance().addMapLayer(rlayer)
```

To add a layer at an absolute position:

```
1 # first add the layer without showing it
2 QgsProject.instance().addMapLayer(rlayer, False)
3 # obtain the layer tree of the top-level group in the project
4 layerTree = iface.layerTreeCanvasBridge().rootGroup()
5 # the position is a number starting from 0, with -1 an alias for the end
6 layerTree.insertChildNode(-1, QgsLayerTreeLayer(rlayer))
```

If you want to delete the layer use the `removeMapLayer()` method:

```
# QgsProject.instance().removeMapLayer(layer_id)
QgsProject.instance().removeMapLayer(rlayer.id())
```

In the above code, the layer id is passed (you can get it calling the `id()` method of the layer), but you can also pass the layer object itself.

For a list of loaded layers and layer ids, use the `mapLayers()` method:

```
QgsProject.instance().mapLayers()
```

The code snippets on this page need the following imports if you're outside the pyqgis console:

```
from qgis.core import (
    QgsProject,
    QgsVectorLayer,
)
```

第 4 章

Accessing the Table Of Contents (TOC)

- *The QgsProject class*
- *QgsLayerTreeGroup class*

You can use different classes to access all the loaded layers in the TOC and use them to retrieve information:

- `QgsProject`
- `QgsLayerTreeGroup`

4.1 The QgsProject class

You can use `QgsProject` to retrieve information about the TOC and all the layers loaded.

You have to create an instance of `QgsProject` and use its methods to get the loaded layers.

The main method is `mapLayers()`. It will return a dictionary of the loaded layers:

```
layers = QgsProject.instance().mapLayers()
print(layers)
```

```
{'countries_89aeb0f_f41b_4f42_bca4_caf55ddbe4b6': <QgsMapLayer: 'countries' (ogr)>
↔}
```

The dictionary keys are the unique layer ids while the values are the related objects.

It is now straightforward to obtain any other information about the layers:

```
1 # list of layer names using list comprehension
2 l = [layer.name() for layer in QgsProject.instance().mapLayers().values()]
3 # dictionary with key = layer name and value = layer object
4 layers_list = {}
5 for l in QgsProject.instance().mapLayers().values():
```

(次のページに続く)

(前のページからの続き)

```
6     layers_list[l.name()] = l
7
8 print(layers_list)
```

```
{'countries': <QgsMapLayer: 'countries' (ogr)>}
```

You can also query the TOC using the name of the layer:

```
country_layer = QgsProject.instance().mapLayersByName("countries")[0]
```

注釈: A list with all the matching layers is returned, so we index with [0] to get the first layer with this name.

4.2 QgsLayerTreeGroup class

The layer tree is a classical tree structure built of nodes. There are currently two types of nodes: group nodes (`QgsLayerTreeGroup`) and layer nodes (`QgsLayerTreeLayer`).

注釈: for more information you can read these blog posts of Martin Dobias: [Part 1](#) [Part 2](#) [Part 3](#)

The project layer tree can be accessed easily with the method `layerTreeRoot()` of the `QgsProject` class:

```
root = QgsProject.instance().layerTreeRoot()
```

`root` is a group node and has *children*:

```
root.children()
```

A list of direct children is returned. Sub group children should be accessed from their own direct parent.

We can retrieve one of the children:

```
child0 = root.children()[0]
print(child0)
```

```
<qgis._core.QgsLayerTreeLayer object at 0x7f1e1ea54168>
```

Layers can also be retrieved using their (unique) id:

```
ids = root.findLayerIds()
# access the first layer of the ids list
root.findLayer(ids[0])
```

And groups can also be searched using their names:

```
root.findGroup('Group Name')
```

`QgsLayerTreeGroup` has many other useful methods that can be used to obtain more information about the TOC:

```
# list of all the checked layers in the TOC
checked_layers = root.checkedLayers()
print(checked_layers)
```

```
[<QgsMapLayer: 'countries' (ogr)>]
```

Now let 's add some layers to the project 's layer tree. There are two ways of doing that:

1. **Explicit addition** using the `addLayer()` or `insertLayer()` functions:

```
1 # create a temporary layer
2 layer1 = QgsVectorLayer("path_to_layer", "Layer 1", "memory")
3 # add the layer to the legend, last position
4 root.addLayer(layer1)
5 # add the layer at given position
6 root.insertLayer(5, layer1)
```

2. **Implicit addition:** since the project's layer tree is connected to the layer registry it is enough to add a layer to the map layer registry:

```
QgsProject.instance().addMapLayer(layer1)
```

You can switch between `QgsVectorLayer` and `QgsLayerTreeLayer` easily:

```
node_layer = root.findLayer(country_layer.id())
print("Layer node:", node_layer)
print("Map layer:", node_layer.layer())
```

```
Layer node: <qgis._core.QgsLayerTreeLayer object at 0x7fecceb46ca8>
Map layer: <QgsMapLayer: 'countries' (ogr)>
```

Groups can be added with the `addGroup()` method. In the example below, the former will add a group to the end of the TOC while for the latter you can add another group within an existing one:

```
node_group1 = root.addGroup('Simple Group')
# add a sub-group to Simple Group
node_subgroup1 = node_group1.addGroup("I'm a sub group")
```

To moving nodes and groups there are many useful methods.

Moving an existing node is done in three steps:

1. cloning the existing node
2. moving the cloned node to the desired position

3. deleting the original node

```
1 # clone the group
2 cloned_group1 = node_group1.clone()
3 # move the node (along with sub-groups and layers) to the top
4 root.insertChildNode(0, cloned_group1)
5 # remove the original node
6 root.removeChildNode(node_group1)
```

It is a little bit more *complicated* to move a layer around in the legend:

```
1 # get a QgsVectorLayer
2 vl = QgsProject.instance().mapLayersByName("countries")[0]
3 # create a QgsLayerTreeLayer object from vl by its id
4 myvl = root.findLayer(vl.id())
5 # clone the myvl QgsLayerTreeLayer object
6 myvlclone = myvl.clone()
7 # get the parent. If None (layer is not in group) returns ''
8 parent = myvl.parent()
9 # move the cloned layer to the top (0)
10 parent.insertChildNode(0, myvlclone)
11 # remove the original myvl
12 root.removeChildNode(myvl)
```

or moving it to an existing group:

```
1 # get a QgsVectorLayer
2 vl = QgsProject.instance().mapLayersByName("countries")[0]
3 # create a QgsLayerTreeLayer object from vl by its id
4 myvl = root.findLayer(vl.id())
5 # clone the myvl QgsLayerTreeLayer object
6 myvlclone = myvl.clone()
7 # create a new group
8 group1 = root.addGroup("Group1")
9 # get the parent. If None (layer is not in group) returns ''
10 parent = myvl.parent()
11 # move the cloned layer to the top (0)
12 group1.insertChildNode(0, myvlclone)
13 # remove the QgsLayerTreeLayer from its parent
14 parent.removeChildNode(myvl)
```

Some other methods that can be used to modify the groups and layers:

```
1 node_group1 = root.findGroup("Group1")
2 # change the name of the group
3 node_group1.setName("Group X")
4 node_layer2 = root.findLayer(country_layer.id())
5 # change the name of the layer
6 node_layer2.setName("Layer X")
7 # change the visibility of a layer
8 node_group1.setItemVisibilityChecked(True)
9 node_layer2.setItemVisibilityChecked(False)
```

(次のページに続く)

(前のページからの続き)

```
10 # expand/collapse the group view
11 node_group1.setExpanded(True)
12 node_group1.setExpanded(False)
```

The code snippets on this page need the following imports if you're outside the pyqgis console:

```
1 from qgis.core import (
2     QgsRasterLayer,
3     QgsProject,
4     QgsPointXY,
5     QgsRaster,
6     QgsRasterShader,
7     QgsColorRampShader,
8     QgsSingleBandPseudoColorRenderer,
9     QgsSingleBandColorDataRenderer,
10    QgsSingleBandGrayRenderer,
11 )
12
13 from qgis.PyQt.QtGui import (
14     QColor,
15 )
```


第5章

ラスターレイヤーを使う

5.1 レイヤーについて

A raster layer consists of one or more raster bands --- referred to as single band and multi band rasters. One band represents a matrix of values. A color image (e.g. aerial photo) is a raster consisting of red, blue and green bands. Single band rasters typically represent either continuous variables (e.g. elevation) or discrete variables (e.g. land use). In some cases, a raster layer comes with a palette and the raster values refer to the colors stored in the palette.

The following code assumes `rlayer` is a `QgsRasterLayer` object.

```
rlayer = QgsProject.instance().mapLayersByName('srtm')[0]
# get the resolution of the raster in layer unit
print(rlayer.width(), rlayer.height())
```

```
919 619
```

```
# get the extent of the layer as QgsRectangle
print(rlayer.extent())
```

```
<QgsRectangle: 20.06856808199999875 -34.27001076999999896, 20.83945284300000012 -
↪33.750775007000000144>
```

```
# get the extent of the layer as Strings
print(rlayer.extent().toString())
```

```
20.0685680819999988,-34.2700107699999990 : 20.8394528430000001,-33.7507750070000014
```

```
# get the raster type: 0 = GrayOrUndefined (single band), 1 = Palette (single_
↪band), 2 = Multiband
print(rlayer.rasterType())
```

```
0
```

```
# get the total band count of the raster
print(rlayer.bandCount())
```

```
1
```

```
# get all the available metadata as a QgsLayerMetadata object
print(rlayer.metadata())
```

```
<qgis._core.QgsLayerMetadata object at 0x13711d558>
```

5.2 レンダラー

When a raster layer is loaded, it gets a default renderer based on its type. It can be altered either in the layer properties or programmatically.

To query the current renderer:

```
print(rlayer.renderer())
```

```
<qgis._core.QgsSingleBandGrayRenderer object at 0x7f471c1da8a0>
```

```
print(rlayer.renderer().type())
```

```
singlebandgray
```

To set a renderer, use the `setRenderer` method of `QgsRasterLayer`. There are a number of renderer classes (derived from `QgsRasterRenderer`):

- `QgsMultiBandColorRenderer`
- `QgsPalettedRasterRenderer`
- `QgsSingleBandColorDataRenderer`
- `QgsSingleBandGrayRenderer`
- `QgsSingleBandPseudoColorRenderer`

Single band raster layers can be drawn either in gray colors (low values = black, high values = white) or with a pseudocolor algorithm that assigns colors to the values. Single band rasters with a palette can also be drawn using the palette. Multiband layers are typically drawn by mapping the bands to RGB colors. Another possibility is to use just one band for drawing.

5.2.1 シングルバンドラスター

Let's say we want a render single band raster layer with colors ranging from green to yellow (corresponding to pixel values from 0 to 255). In the first stage we will prepare a `QgsRasterShader` object and configure its shader function:

```

1 fcn = QgsColorRampShader()
2 fcn.setColorRampType(QgsColorRampShader.Interpolated)
3 lst = [ QgsColorRampShader.ColorRampItem(0, QColor(0,255,0)),
4         QgsColorRampShader.ColorRampItem(255, QColor(255,255,0)) ]
5 fcn.setColorRampItemList(lst)
6 shader = QgsRasterShader()
7 shader.setRasterShaderFunction(fcn)

```

The shader maps the colors as specified by its color map. The color map is provided as a list of pixel values with associated colors. There are three modes of interpolation:

- **linear (Interpolated):** the color is linearly interpolated from the color map entries above and below the pixel value
- **discrete (Discrete):** the color is taken from the closest color map entry with equal or higher value
- **exact (Exact):** the color is not interpolated, only pixels with values equal to color map entries will be drawn

In the second step we will associate this shader with the raster layer:

```

renderer = QgsSingleBandPseudoColorRenderer(rlayer.dataProvider(), 1, shader)
rlayer.setRenderer(renderer)

```

The number 1 in the code above is the band number (raster bands are indexed from one).

Finally we have to use the `triggerRepaint` method to see the results:

```

rlayer.triggerRepaint()

```

5.2.2 マルチバンドラスター

By default, QGIS maps the first three bands to red, green and blue to create a color image (this is the `MultiBandColor` drawing style). In some cases you might want to override these setting. The following code interchanges red band (1) and green band (2):

```

rlayer_multi = QgsProject.instance().mapLayersByName('multiband')[0]
rlayer_multi.renderer().setGreenBand(1)
rlayer_multi.renderer().setRedBand(2)

```

In case only one band is necessary for visualization of the raster, single band drawing can be chosen, either gray levels or pseudocolor.

We have to use `triggerRepaint` to update the map and see the result:

```
rlayer_multi.triggerRepaint()
```

5.3 値の検索

Raster values can be queried using the `sample` method of the `QgsRasterDataProvider` class. You have to specify a `QgsPointXY` and the band number of the raster layer you want to query. The method returns a tuple with the value and `True` or `False` depending on the results:

```
val, res = rlayer.dataProvider().sample(QgsPointXY(20.50, -34), 1)
```

Another method to query raster values is using the `identify` method that returns a `QgsRasterIdentifyResult` object.

```
ident = rlayer.dataProvider().identify(QgsPointXY(20.5, -34), QgsRaster.  
→IdentifyFormatValue)  
  
if ident.isValid():  
    print(ident.results())
```

```
{1: 323.0}
```

In this case, the `results` method returns a dictionary, with band indices as keys, and band values as values. For instance, something like `{1: 323.0}`

The code snippets on this page need the following imports if you're outside the pyqgis console:

```
1 from qgis.core import (  
2     QgsApplication,  
3     QgsDataSourceUri,  
4     QgsCategorizedSymbolRenderer,  
5     QgsClassificationRange,  
6     QgsPointXY,  
7     QgsProject,  
8     QgsExpression,  
9     QgsField,  
10    QgsFields,  
11    QgsFeature,  
12    QgsFeatureRequest,  
13    QgsFeatureRenderer,  
14    QgsGeometry,  
15    QgsGraduatedSymbolRenderer,  
16    QgsMarkerSymbol,  
17    QgsMessageLog,  
18    QgsRectangle,  
19    QgsRendererCategory,  
20    QgsRendererRange,
```

(次のページに続く)

(前のページからの続き)

```
21 QgsSymbol,  
22 QgsVectorDataProvider,  
23 QgsVectorLayer,  
24 QgsVectorFileWriter,  
25 QgsWkbTypes,  
26 QgsSpatialIndex,  
27 )  
28  
29 from qgis.core.additions.edit import edit  
30  
31 from qgis.PyQt.QtGui import (  
32     QColor,  
33 )
```


第 6 章

ベクターレイヤーを使う

- 属性に関する情報を取得する
- ベクターレイヤーの反復処理
- 地物の選択
 - 属性にアクセスする
 - 選択された地物への反復処理
 - 一部の地物への反復処理
- ベクターレイヤーを修正する
 - 地物の追加
 - 地物の削除
 - 地物の修正
 - ベクターレイヤーを編集バッファで修正する
 - フィールドを追加または削除する
- 空間索引を使う
- ベクターレイヤを作る
 - *From an instance of `QgsVectorFileWriter`*
 - *Directly from features*
 - `QgsVectorLayer` クラスのインスタンスから作成する
- ベクタレイヤの表現 (シンボロジ)
 - 単一シンボルレンダラー
 - 分類シンボルレンダラー

- 段階シンボルレンダラー
- シンボルの操作
 - * シンボルレイヤーの操作
 - * カスタムシンボルレイヤータイプの作成
- カスタムレンダラーの作成
- より詳しいトピック

このセクションではベクターレイヤーに対して行える様々な操作について紹介していきます。

Most work here is based on the methods of the `QgsVectorLayer` class.

6.1 属性に関する情報を取得する

You can retrieve information about the fields associated with a vector layer by calling `fields()` on a `QgsVectorLayer` object:

```
vlayer = QgsVectorLayer("testdata/airports.shp", "airports", "ogr")
for field in vlayer.fields():
    print(field.name(), field.typeName())
```

```
1 ID Integer64
2 fk_region Integer64
3 ELEV Real
4 NAME String
5 USE String
```

The `displayField()` and `mapTipTemplate()` methods of the `QgsVectorLayer` class provide information on the field and template used in the maptips tab.

When you load a vector layer, a field is always chosen by QGIS as the Display Name, while the HTML Map Tip is empty by default. With these methods you can easily get both:

```
vlayer = QgsVectorLayer("testdata/airports.shp", "airports", "ogr")
print(vlayer.displayField())
```

```
NAME
```

注釈: If you change the Display Name from a field to an expression, you have to use `displayExpression()` instead of `displayField()`.

6.2 ベクターレイヤーの反復処理

Iterating over the features in a vector layer is one of the most common tasks. Below is an example of the simple basic code to perform this task and showing some information about each feature. The `layer` variable is assumed to have a `QgsVectorLayer` object.

```

1 # "layer" is a QgsVectorLayer instance
2 layer = iface.activeLayer()
3 features = layer.getFeatures()
4
5 for feature in features:
6     # retrieve every feature with its geometry and attributes
7     print("Feature ID: ", feature.id())
8     # fetch geometry
9     # show some information about the feature geometry
10    geom = feature.geometry()
11    geomSingleType = QgsWkbTypes.isSingleType(geom.wkbType())
12    if geom.type() == QgsWkbTypes.PointGeometry:
13        # the geometry type can be of single or multi type
14        if geomSingleType:
15            x = geom.asPoint()
16            print("Point: ", x)
17        else:
18            x = geom.asMultiPoint()
19            print("MultiPoint: ", x)
20    elif geom.type() == QgsWkbTypes.LineGeometry:
21        if geomSingleType:
22            x = geom.asPolyline()
23            print("Line: ", x, "length: ", geom.length())
24        else:
25            x = geom.asMultiPolyline()
26            print("MultiLine: ", x, "length: ", geom.length())
27    elif geom.type() == QgsWkbTypes.PolygonGeometry:
28        if geomSingleType:
29            x = geom.asPolygon()
30            print("Polygon: ", x, "Area: ", geom.area())
31        else:
32            x = geom.asMultiPolygon()
33            print("MultiPolygon: ", x, "Area: ", geom.area())
34    else:
35        print("Unknown or invalid geometry")
36    # fetch attributes
37    attrs = feature.attributes()
38    # attrs is a list. It contains all the attribute values of this feature
39    print(attrs)
40    # for this test only print the first feature
41    break

```

```

Feature ID: 1
Point: <QgsPointXY: POINT(7 45)>
[1, 'First feature']

```

6.3 地物の選択

In QGIS desktop, features can be selected in different ways: the user can click on a feature, draw a rectangle on the map canvas or use an expression filter. Selected features are normally highlighted in a different color (default is yellow) to draw user's attention on the selection.

Sometimes it can be useful to programmatically select features or to change the default color.

To select all the features, the `selectAll()` method can be used:

```
# Get the active layer (must be a vector layer)
layer = iface.activeLayer()
layer.selectAll()
```

To select using an expression, use the `selectByExpression()` method:

```
# Assumes that the active layer is points.shp file from the QGIS test suite
# (Class (string) and Heading (number) are attributes in points.shp)
layer = iface.activeLayer()
layer.selectByExpression('"Class"=\'B52\' and "Heading" > 10 and "Heading" <70',
↳QgsVectorLayer.SetSelection)
```

To change the selection color you can use `setSelectionColor()` method of `QgsMapCanvas` as shown in the following example:

```
iface.mapCanvas().setSelectionColor( QColor("red") )
```

To add features to the selected features list for a given layer, you can call `select()` passing to it the list of features IDs:

```
1 selected_fid = []
2
3 # Get the first feature id from the layer
4 for feature in layer.getFeatures():
5     selected_fid.append(feature.id())
6     break
7
8 # Add these features to the selected list
9 layer.select(selected_fid)
```

To clear the selection:

```
layer.removeSelection()
```

6.3.1 属性にアクセスする

Attributes can be referred to by their name:

```
print(feature['name'])
```

```
First feature
```

Alternatively, attributes can be referred to by index. This is a bit faster than using the name. For example, to get the second attribute:

```
print(feature[1])
```

```
First feature
```

6.3.2 選択された地物への反復処理

If you only need selected features, you can use the `selectedFeatures()` method from the vector layer:

```
selection = layer.selectedFeatures()
for feature in selection:
    # do whatever you need with the feature
    pass
```

6.3.3 一部の地物への反復処理

If you want to iterate over a given subset of features in a layer, such as those within a given area, you have to add a `QgsFeatureRequest` object to the `getFeatures()` call. Here's an example:

```
1 areaOfInterest = QgsRectangle(450290,400520, 450750,400780)
2
3 request = QgsFeatureRequest().setFilterRect(areaOfInterest)
4
5 for feature in layer.getFeatures(request):
6     # do whatever you need with the feature
7     pass
```

For the sake of speed, the intersection is often done only using feature 's bounding box. There is however a flag `ExactIntersect` that makes sure that only intersecting features will be returned:

```
request = QgsFeatureRequest().setFilterRect(areaOfInterest) \
    .setFlags(QgsFeatureRequest.ExactIntersect)
```

With `setLimit()` you can limit the number of requested features. Here's an example:

```
request = QgsFeatureRequest()
request.setLimit(2)
for feature in layer.getFeatures(request):
    print(feature)
```

```
<qgis._core.QgsFeature object at 0x7f9b78590948>
```

If you need an attribute-based filter instead (or in addition) of a spatial one like shown in the examples above, you can build a `QgsExpression` object and pass it to the `QgsFeatureRequest` constructor. Here's an example:

```
# The expression will filter the features where the field "location_name"
# contains the word "Lake" (case insensitive)
exp = QgsExpression('location_name ILIKE \'%Lake%\')
request = QgsFeatureRequest(exp)
```

See [式、フィルタ適用および値の算出](#) for the details about the syntax supported by `QgsExpression`.

要求は、地物ごとに取得したデータを定義するために使用できるので、反復子はすべての地物を返しますが、それぞれの地物については部分的データを返します。

```
1 # Only return selected fields to increase the "speed" of the request
2 request.setSubsetOfAttributes([0,2])
3
4 # More user friendly version
5 request.setSubsetOfAttributes(['name','id'],layer.fields())
6
7 # Don't return geometry objects to increase the "speed" of the request
8 request.setFlags(QgsFeatureRequest.NoGeometry)
9
10 # Fetch only the feature with id 45
11 request.setFilterFid(45)
12
13 # The options may be chained
14 request.setFilterRect(areaOfInterest).setFlags(QgsFeatureRequest.NoGeometry).
    ↪setFilterFid(45).setSubsetOfAttributes([0,2])
```

6.4 ベクターレイヤーを修正する

大部分のベクターデータプロバイダーは、レイヤーの編集をサポートしています。プロバイダーによっては、可能な編集操作の一部だけしかサポートしていないこともあります。どんな機能をサポートしているかを知るには、`capabilities()` 関数を使ってください。

```
caps = layer.dataProvider().capabilities()
# Check if a particular capability is supported:
if caps & QgsVectorDataProvider.DeleteFeatures:
    print('The layer supports DeleteFeatures')
```

```
The layer supports DeleteFeatures
```

可能な性能をすべて知るには、[API Documentation of QgsVectorDataProvider](#) を参照してください。

`capabilitiesString()` を使うと、下記の例に見るように、レイヤーの性能の説明文をコンマで区切られたリストの形で表示することができます。

```
1 caps_string = layer.dataProvider().capabilitiesString()
2 # Print:
3 # 'Add Features, Delete Features, Change Attribute Values, Add Attributes,
4 # Delete Attributes, Rename Attributes, Fast Access to Features at ID,
5 # Presimplify Geometries, Presimplify Geometries with Validity Check,
6 # Transactions, Curved Geometries'
```

ベクターレイヤーを編集する以下の方法はいずれも、変更が直接、レイヤーの裏にあるデータストア（ファイルやデータベースなど）にコミットされます。一時的な変更をしたいだけの場合にどうすればよいかの説明は、次のセクション [ベクターレイヤーを編集バッファで修正する](#) でしているので、以下を飛ばしてそちらに進んでください。

注釈: QGIS の内部（コンソールまたはプラグインのいずれか）で作業している場合、ジオメトリ、スタイル、属性に加えられた変更を確認するために、以下のように地図キャンパスの強制的な再描画が必要になることもあります。

```
1 # If caching is enabled, a simple canvas refresh might not be sufficient
2 # to trigger a redraw and you must clear the cached image for the layer
3 if iface.mapCanvas().isCachingEnabled():
4     layer.triggerRepaint()
5 else:
6     iface.mapCanvas().refresh()
```

6.4.1 地物の追加

Create some `QgsFeature` instances and pass a list of them to provider's `addFeatures()` method. It will return two values: result (true/false) and list of added features (their ID is set by the data store).

To set up the attributes of the feature, you can either initialize the feature passing a `QgsFields` object (you can obtain that from the `fields()` method of the vector layer) or call `initAttributes()` passing the number of fields you want to be added.

```
1 if caps & QgsVectorDataProvider.AddFeatures:
2     feat = QgsFeature(layer.fields())
3     feat.setAttributes([0, 'hello'])
4     # Or set a single attribute by key or by index:
5     feat.setAttribute('name', 'hello')
```

(次のページに続く)

```
6 feat.setAttribute(0, 'hello')
7 feat.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(123, 456)))
8 (res, outFeats) = layer.dataProvider().addFeatures([feat])
```

6.4.2 地物の削除

To delete some features, just provide a list of their feature IDs.

```
if caps & QgsVectorDataProvider.DeleteFeatures:
    res = layer.dataProvider().deleteFeatures([5, 10])
```

6.4.3 地物の修正

It is possible to either change feature's geometry or to change some attributes. The following example first changes values of attributes with index 0 and 1, then it changes the feature's geometry.

```
1 fid = 100 # ID of the feature we will modify
2
3 if caps & QgsVectorDataProvider.ChangeAttributeValues:
4     attrs = { 0 : "hello", 1 : 123 }
5     layer.dataProvider().changeAttributeValues({ fid : attrs })
6
7 if caps & QgsVectorDataProvider.ChangeGeometries:
8     geom = QgsGeometry.fromPointXY(QgsPointXY(111,222))
9     layer.dataProvider().changeGeometryValues({ fid : geom })
```

ちなみに: Favor `QgsVectorLayerEditUtils` class for geometry-only edits

If you only need to change geometries, you might consider using the `QgsVectorLayerEditUtils` which provides some useful methods to edit geometries (translate, insert or move vertex, etc.).

6.4.4 ベクターレイヤーを編集バッファで修正する

When editing vectors within QGIS application, you have to first start editing mode for a particular layer, then do some modifications and finally commit (or rollback) the changes. All the changes you make are not written until you commit them --- they stay in layer's in-memory editing buffer. It is possible to use this functionality also programmatically --- it is just another method for vector layer editing that complements the direct usage of data providers. Use this option when providing some GUI tools for vector layer editing, since this will allow user to decide whether to commit/rollback and allows the usage of undo/redo. When changes are committed, all changes from the editing buffer are saved to data provider.

メソッドはすでに見たプロバイダーにおけるものとよく似ていますが、プロバイダーではなく `QgsVectorLayer` オブジェクトで呼び出されます。

これらのメソッドが機能するためには、そのレイヤーは編集モードでなければいけません。編集モードを開始するには、`startEditing()` メソッドを使用します。編集を終了するには、`commitChanges()` メソッドか、もしくは `rollBack()` メソッドを使用します。前者はすべての変更をデータソースにコミットします。一方後者は変更をすべて破棄し、データソースには一切、手をつけません。

あるレイヤーが編集モードかどうかを知るには、`isEditable()` メソッドを使用してください。

では、これら編集メソッドの使用方法を示す実例をいくつか見てもらいます。

```

1 from qgis.PyQt.QtCore import QVariant
2
3 feat1 = feat2 = QgsFeature(layer.fields())
4 fid = 99
5 feat1.setId(fid)
6
7 # add two features (QgsFeature instances)
8 layer.addFeatures([feat1, feat2])
9 # delete a feature with specified ID
10 layer.deleteFeature(fid)
11
12 # set new geometry (QgsGeometry instance) for a feature
13 geometry = QgsGeometry.fromWkt("POINT(7 45)")
14 layer.changeGeometry(fid, geometry)
15 # update an attribute with given field index (int) to a given value
16 fieldIndex = 1
17 value = 'My new name'
18 layer.changeAttributeValue(fid, fieldIndex, value)
19
20 # add new field
21 layer.addAttribute(QgsField("mytext", QVariant.String))
22 # remove a field
23 layer.deleteAttribute(fieldIndex)

```

取り消し/やり直しを適切に機能させるためには、上記のメソッド呼び出しを `undo` コマンドでラップしなければなりません。取り消し/やり直し機能が不要で、変更を即座に保存したい場合は、**データプロバイダを使って編集**したほうが手軽でしょう。

取り消し機能を使用するには次のように行います。

```

1 layer.beginEditCommand("Feature triangulation")
2
3 # ... call layer's editing methods ...
4
5 if problem_occurred:
6     layer.destroyEditCommand()
7     # ... tell the user that there was a problem
8     # and return
9
10 # ... more editing ...

```

(次のページに続く)

```
11
12 layer.endEditCommand()
```

`beginEditCommand()` メソッドは内部的に「アクティブな」コマンドを生成し、ベクターレイヤーでその後に行われる変化を記録し続けます。`endEditCommand()` メソッドの呼び出しによって、コマンドはアンドゥスタックにプッシュされ、ユーザーが GUI から取り消し/やり直しをすることができるようになります。変更の最中に何か不具合が生じたときは、`destroyEditCommand()` メソッドによってコマンドは削除され、コマンドがアクティブな間に行われたすべての変更はロールバックされます。

次の例に示すように、よりセマンティックなコードブロックにコミットとロールバックをラップする `with edit(layer)` 文も使用できます。

```
with edit(layer):
    feat = next(layer.getFeatures())
    feat[0] = 5
    layer.updateFeature(feat)
```

これは最後に `commitChanges()` メソッドを自動的に呼び出します。もし何らかの例外が発生したときは、`rollBack()` メソッドを呼び出してすべての変更をロールバックします。`commitChanges()` メソッドの実行の最中に問題に遭遇したとき(メソッドが `False` を返したとき)は、`QgsEditError` 例外を送出します。

6.4.5 フィールドを追加または削除する

フィールド(属性)を追加するには、フィールドの定義を配列で指定する必要があります。フィールドを削除するにはフィールドのインデックスを配列で渡すだけです。

```
1 from qgis.PyQt.QtCore import QVariant
2
3 if caps & QgsVectorDataProvider.AddAttributes:
4     res = layer.dataProvider().addAttributes(
5         [QgsField("mytext", QVariant.String),
6          QgsField("myint", QVariant.Int)])
7
8 if caps & QgsVectorDataProvider.DeleteAttributes:
9     res = layer.dataProvider().deleteAttributes([0])
```

```
1 # Alternate methods for removing fields
2 # first create temporary fields to be removed (f1-3)
3 layer.dataProvider().addAttributes([QgsField("f1", QVariant.Int), QgsField("f2",
4     ↳QVariant.Int), QgsField("f3", QVariant.Int)])
5 layer.updateFields()
6 count=layer.fields().count() # count of layer fields
7 ind_list=list((count-3, count-2)) # create list
8
9 # remove a single field with an index
10 layer.dataProvider().deleteAttributes([count-1])
```

(次のページに続く)

(前のページからの続き)

```

11 # remove multiple fields with a list of indices
12 layer.dataProvider().deleteAttributes(ind_list)

```

データプロバイダーのフィールドを追加または削除した後、レイヤーのフィールドは、変更が自動的に反映されていないため、更新する必要があります。

```
layer.updateFields()
```

ちなみに: `with` に基づくコマンドを使って変更を直接保存する

Using `with edit(layer)`: the changes will be committed automatically calling `commitChanges()` at the end. If any exception occurs, it will `rollback()` all the changes. See [ベクターレイヤーを編集バッファで修正する](#).

6.5 空間索引を使う

空間索引は、頻繁にベクターレイヤーに問い合わせをする必要がある場合、コードのパフォーマンスを劇的に改善します。例えば、補間アルゴリズムを書いていて、補間値の計算に使用するために与えられた位置に対して最も近い10点をポイントレイヤーから求める必要がある、と想像してください。空間索引が無いと、QGISがこれらの10点を求める方法は、すべての点から指定の場所への距離を計算してそれらの距離を比較することしかありません。これは、いくつかの場所について繰り返す必要がある場合は特に、非常に時間のかかる処理となります。もし空間索引がレイヤーに作成されていれば、処理はもっと効率的になります。

空間索引の無いレイヤーは、電話番号が順番に並んでいない、もしくは索引の無い電話帳とってください。所定の人の電話番号を見つける唯一の方法は、巻頭からその番号を見つけるまで読むだけです。

空間索引は、QGIS ベクターレイヤーに対してデフォルトでは作成されていませんが、簡単に作成できます。しなければいけないことはこうです：

- create spatial index using the `QgsSpatialIndex()` class:

```
index = QgsSpatialIndex()
```

- add features to index --- index takes `QgsFeature` object and adds it to the internal data structure. You can create the object manually or use one from a previous call to the provider's `getFeatures()` method.

```
index.addFeature(feats)
```

- 代わりに、一括読み込みを使用してレイヤーのすべての地物を一度に読み込むことができます

```
index = QgsSpatialIndex(layer.getFeatures())
```

- 空間索引に何かしらの値が入れると検索ができるようになります

```

1 # returns array of feature IDs of five nearest features
2 nearest = index.nearestNeighbor(QgsPointXY(25.4, 12.7), 5)
3
4 # returns array of IDs of features which intersect the rectangle
5 intersect = index.intersects(QgsRectangle(22.5, 15.3, 23.1, 17.2))

```

6.6 ベクターレイヤを作る

ベクターレイヤデータセットを作るには幾つかの方法があります。

- `QgsVectorFileWriter` クラス：ベクターファイルをディスクに書きこむための便利なクラスです。静的メソッド `writeAsVectorFormat()` を呼び出してすべてのベクターレイヤを保存するか、もしくはクラスインスタンスを作ってから `addFeature()` メソッドを呼び出すか、いずれかの方法を使うことができます。このクラスは GeoPackage、Shapefile、GeoJSON、KML その他のすべての OGR がサポートするベクターフォーマットをサポートしています。
- `QgsVectorLayer` クラス：データソースの指定されたパス (url) を解釈してデータに接続し、アクセスしたデータプロバイダをインスタンス化します。メモリ上の一時的なレイヤ (memory) を作ったり、OGR データセット (ogr) やデータベース (postgres, spatialite, mysql, mssql) やその他 (wfs, gpx, delimitedtext...) に接続するために使うことができます。

6.6.1 From an instance of `QgsVectorFileWriter`

```

1 # SaveVectorOptions contains many settings for the writer process
2 save_options = QgsVectorFileWriter.SaveVectorOptions()
3 transform_context = QgsProject.instance().transformContext()
4 # Write to a GeoPackage (default)
5 error = QgsVectorFileWriter.writeAsVectorFormatV2(layer,
6
7                                     "testdata/my_new_file.gpkg",
8                                     transform_context,
9                                     save_options)
10
11 if error[0] == QgsVectorFileWriter.NoError:
12     print("success!")
13
14 else:
15     print(error)

```

```

1 # Write to an ESRI Shapefile format dataset using UTF-8 text encoding
2 save_options = QgsVectorFileWriter.SaveVectorOptions()
3 save_options.driverName = "ESRI Shapefile"
4 save_options.fileEncoding = "UTF-8"
5 transform_context = QgsProject.instance().transformContext()
6 error = QgsVectorFileWriter.writeAsVectorFormatV2(layer,
7
8                                     "testdata/my_new_shapefile",
9                                     transform_context,
10                                     save_options)
11
12 if error[0] == QgsVectorFileWriter.NoError:

```

(次のページに続く)

(前のページからの続き)

```

11     print("success again!")
12 else:
13     print(error)

```

```

1  # Write to an ESRI GDB file
2  save_options = QgsVectorFileWriter.SaveVectorOptions()
3  save_options.driverName = "FileGDB"
4  # if no geometry
5  save_options.overrideGeometryType = QgsWkbTypes.Unknown
6  save_options.actionOnExistingFile = QgsVectorFileWriter.CreateOrOverwriteLayer
7  save_options.layerName = 'my_new_layer_name'
8  transform_context = QgsProject.instance().transformContext()
9  gdb_path = "testdata/my_example.gdb"
10 error = QgsVectorFileWriter.writeAsVectorFormatV2(layer,
11                                                    gdb_path,
12                                                    transform_context,
13                                                    save_options)
14 if error[0] == QgsVectorFileWriter.NoError:
15     print("success!")
16 else:
17     print(error)

```

You can also convert fields to make them compatible with different formats by using the `FieldValueConverter`. For example, to convert array variable types (e.g. in Postgres) to a text type, you can do the following:

```

1  LIST_FIELD_NAME = 'xxxx'
2
3  class ESRIValueConverter(QgsVectorFileWriter.FieldValueConverter):
4
5      def __init__(self, layer, list_field):
6          QgsVectorFileWriter.FieldValueConverter.__init__(self)
7          self.layer = layer
8          self.list_field_idx = self.layer.fields().indexOfName(list_field)
9
10     def convert(self, fieldIdxInLayer, value):
11         if fieldIdxInLayer == self.list_field_idx:
12             return QgsListFieldFormatter().representValue(layer=vlayer,
13                                                         fieldIndex=self.list_field_idx,
14                                                         config={},
15                                                         cache=None,
16                                                         value=value)
17         else:
18             return value
19
20     def fieldDefinition(self, field):
21         idx = self.layer.fields().indexOfName(field.name())
22         if idx == self.list_field_idx:
23             return QgsField(LIST_FIELD_NAME, QVariant.String)
24         else:

```

(次のページに続く)

(前のページからの続き)

```
25     return self.layer.fields()[idx]
26
27 converter = ESRIValueConverter(vlayer, LIST_FIELD_NAME)
28 opts = QgsVectorFileWriter.SaveVectorOptions()
29 opts.fieldValueConverter = converter
```

A destination CRS may also be specified --- if a valid instance of `QgsCoordinateReferenceSystem` is passed as the fourth parameter, the layer is transformed to that CRS.

For valid driver names please call the `supportedFiltersAndFormats` method or consult the `supported formats by OGR` --- you should pass the value in the "Code" column as the driver name.

Optionally you can set whether to export only selected features, pass further driver-specific options for creation or tell the writer not to create attributes... There are a number of other (optional) parameters; see the `QgsVectorFileWriter` documentation for details.

6.6.2 Directly from features

```
1 from qgis.PyQt.QtCore import QVariant
2
3 # define fields for feature attributes. A QgsFields object is needed
4 fields = QgsFields()
5 fields.append(QgsField("first", QVariant.Int))
6 fields.append(QgsField("second", QVariant.String))
7
8 """ create an instance of vector file writer, which will create the vector file.
9 Arguments:
10 1. path to new file (will fail if exists already)
11 2. field map
12 3. geometry type - from WKBTYPENUM enum
13 4. layer's spatial reference (instance of
14    QgsCoordinateReferenceSystem)
15 5. coordinate transform context
16 6. save options (driver name for the output file, encoding etc.)
17 """
18
19 crs = QgsProject.instance().crs()
20 transform_context = QgsProject.instance().transformContext()
21 save_options = QgsVectorFileWriter.SaveVectorOptions()
22 save_options.driverName = "ESRI Shapefile"
23 save_options.fileEncoding = "UTF-8"
24
25 writer = QgsVectorFileWriter.create(
26     "testdata/my_new_shapefile.shp",
27     fields,
28     QgsWkbTypes.Point,
29     crs,
30     transform_context,
31     save_options
```

(次のページに続く)

(前のページからの続き)

```

32 )
33
34 if writer.hasError() != QgsVectorFileWriter.NoError:
35     print("Error when creating shapefile: ", writer.errorMessage())
36
37 # add a feature
38 fet = QgsFeature()
39
40 fet.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(10,10)))
41 fet.setAttributes([1, "text"])
42 writer.addFeature(fet)
43
44 # delete the writer to flush features to disk
45 del writer

```

6.6.3 QgsVectorLayer クラスのインスタンスから作成する

`QgsVectorLayer` クラスによってサポートされているすべてのデータプロバイダのうちから、ここではメモリレイヤに焦点をあてましょう。メモリプロバイダは主にプラグインやサードパーティ製アプリの開発者に使われることを意図しています。ディスクにデータを格納することをしないため、開発者はなんらかの一時的なレイヤのための手っ取り早いバックエンドとしてこれを使うことができます。

このプロバイダは属性フィールドの型として `string`、`int`、`double` をサポートします。

メモリプロバイダは空間インデックスもサポートしています。これはプロバイダの `createSpatialIndex()` 関数を呼び出すことによって有効になります。空間インデックスが作成されると、複数の地物にわたって行う処理を、より小さな領域内でより速く行うことができます。これはあらためて地物すべてを走査する必要がなく、指定された領域内のみを走査すればよいからです。

メモリプロバイダは `QgsVectorLayer` コンストラクタにプロバイダ文字列として `"memory"` を渡すと作ることができます。

コンストラクタはレイヤのジオメトリタイプを定義する URI も必要とします。これは `"Point"`、`"LineString"`、`"Polygon"`、`"MultiPoint"`、`"MultiLineString"`、`"MultiPolygon"`、`"None"` のうちのひとつです。

URI ではメモリプロバイダの座標参照系、属性フィールド、URI 内でのメモリプロバイダのインデックスも指定できます。構文は、

`crs=definition` 座標参照系を指定します。ここでの定義は `QgsCoordinateReferenceSystem.createFromString` で有効な形式のいずれかを使用します。

`index=yes` プロバイダーが空間インデックスを使うように指定します。

`field=name:type(length,precision)` レイヤーの属性を指定します。属性は名前を持ち、オプションとして型 (`integer`、`double`、`string`)、長さ、および精度を持ちます。フィールドの定義は複数あってもかまいません。

次のサンプルは全てのこれらのオプションを含んだ URL です:

```
"Point?crs=epsg:4326&field=id:integer&field=name:string(20)&index=yes"
```

次のサンプルコードはメモリプロバイダーを作成してデータ投入をしている様子です:

```
1 from qgis.PyQt.QtCore import QVariant
2
3 # create layer
4 vl = QgsVectorLayer("Point", "temporary_points", "memory")
5 pr = vl.dataProvider()
6
7 # add fields
8 pr.addAttributes([QgsField("name", QVariant.String),
9                    QgsField("age", QVariant.Int),
10                   QgsField("size", QVariant.Double)])
11 vl.updateFields() # tell the vector layer to fetch changes from the provider
12
13 # add a feature
14 fet = QgsFeature()
15 fet.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(10,10)))
16 fet.setAttributes(["Johny", 2, 0.3])
17 pr.addFeatures([fet])
18
19 # update layer's extent when new features have been added
20 # because change of extent in provider is not propagated to the layer
21 vl.updateExtents()
```

最後にやったことを全て確認していきましょう:

```
1 # show some stats
2 print("fields:", len(pr.fields()))
3 print("features:", pr.featureCount())
4 e = vl.extent()
5 print("extent:", e.xMinimum(), e.yMinimum(), e.xMaximum(), e.yMaximum())
6
7 # iterate over features
8 features = vl.getFeatures()
9 for fet in features:
10     print("F:", fet.id(), fet.attributes(), fet.geometry().asPoint())
```

```
fields: 3
features: 1
extent: 10.0 10.0 10.0 10.0
F: 1 ['Johny', 2, 0.3] <QgsPointXY: POINT(10 10)>
```


6.7 ベクタレイヤの表現 (シンボロジ)

ベクタレイヤがレンダリングされる時、データの表現はレイヤに関連付けられた レンダラー と シンボル によって決定されます。シンボルは地物の視覚的表現を処理するクラスで、レンダラはそれぞれの地物でどのシンボルが使われるかを決定します。

The renderer for a given layer can be obtained as shown below:

```
renderer = layer.renderer()
```

この参照を利用して、少しだけ探索してみましょう:

```
print("Type:", renderer.type())
```

```
Type: singleSymbol
```

There are several known renderer types available in the QGIS core library:

| タイプ | クラス | 詳細 |
|-------------------|------------------------------|--|
| singleSymbol | QgsSingleSymbolRenderer | 単一シンボル。全ての地物を同じシンボルでレンダリングします |
| categorizedSymbol | QgsCategorizedSymbolRenderer | 分類されたシンボル。カテゴリごとに違うシンボルを使って地物をレンダリングします |
| graduatedSymbol | QgsGraduatedSymbolRenderer | 段階に分けられたシンボル。それぞれの範囲の値によって違うシンボルを使って地物をレンダリングします |

There might be also some custom renderer types, so never make an assumption there are just these types. You can query the application's `QgsRendererRegistry` to find out currently available renderers:

```
print(QgsApplication.rendererRegistry().renderersList())
```

```
['nullSymbol', 'singleSymbol', 'categorizedSymbol', 'graduatedSymbol',
 → 'RuleRenderer', 'pointDisplacement', 'pointCluster', 'invertedPolygonRenderer',
 → 'heatmapRenderer', '25dRenderer']
```

レンダラーの中身をテキストフォームにダンプできます --- デバッグ時に役に立つでしょう:

```
renderer.dump()
```

```
SINGLE: MARKER SYMBOL (1 layers) color 190,207,80,255
```

6.7.1 単一シンボルレンダラー

You can get the symbol used for rendering by calling `symbol()` method and change it with `setSymbol()` method (note for C++ devs: the renderer takes ownership of the symbol.)

You can change the symbol used by a particular vector layer by calling `setSymbol()` passing an instance of the appropriate symbol instance. Symbols for *point*, *line* and *polygon* layers can be created by calling the `createSimple()` function of the corresponding classes `QgsMarkerSymbol`, `QgsLineSymbol` and `QgsFillSymbol`.

The dictionary passed to `createSimple()` sets the style properties of the symbol.

For example you can replace the symbol used by a particular **point** layer by calling `setSymbol()` passing an instance of a `QgsMarkerSymbol`, as in the following code example:

```
symbol = QgsMarkerSymbol.createSimple({'name': 'square', 'color': 'red'})
layer.renderer().setSymbol(symbol)
# show the change
layer.triggerRepaint()
```

`name` は、マーカーの形状を示しており、以下のいずれかとすることができます。

- circle
- square
- cross
- rectangle
- diamond
- pentagon
- triangle
- equilateral_triangle
- star
- regular_star
- arrow
- filled_arrowhead
- x

To get the full list of properties for the first symbol layer of a symbol instance you can follow the example code:

```
print(layer.renderer().symbol().symbolLayers()[0].properties())
```

```
{'angle': '0', 'color': '255,0,0,255', 'horizontal_anchor_point': '1', 'joinstyle
↪': 'bevel', 'name': 'square', 'offset': '0,0', 'offset_map_unit_scale': '3x:0,0,
↪0,0,0,0', 'offset_unit': 'MM', 'outline_color': '35,35,35,255', 'outline_style':
↪'solid', 'outline_width': '0', 'outline_width_map_unit_scale': '3x:0,0,0,0,0,0',
↪'outline_width_unit': 'MM', 'scale_method': 'diameter', 'size': '2', 'size_map_
↪unit_scale': '3x:0,0,0,0,0,0', 'size_unit': 'MM', 'vertical_anchor_point': '1'}
```

いくつかのプロパティを変更したい場合に便利です:

```
1 # You can alter a single property...
2 layer.renderer().symbol().symbolLayer(0).setSize(3)
3 # ... but not all properties are accessible from methods,
4 # you can also replace the symbol completely:
5 props = layer.renderer().symbol().symbolLayer(0).properties()
6 props['color'] = 'yellow'
7 props['name'] = 'square'
8 layer.renderer().setSymbol(QgsMarkerSymbol.createSimple(props))
9 # show the changes
10 layer.triggerRepaint()
```

6.7.2 分類シンボルレンダラー

When using a categorized renderer, you can query and set the attribute that is used for classification: use the `classAttribute()` and `setClassAttribute()` methods.

カテゴリの配列を取得するには

```
1 categorized_renderer = QgsCategorizedSymbolRenderer()
2 # Add a few categories
3 cat1 = QgsRendererCategory('1', QgsMarkerSymbol(), 'category 1')
4 cat2 = QgsRendererCategory('2', QgsMarkerSymbol(), 'category 2')
5 categorized_renderer.addCategory(cat1)
6 categorized_renderer.addCategory(cat2)
7
8 for cat in categorized_renderer.categories():
9     print("{}: {} :: {}".format(cat.value(), cat.label(), cat.symbol()))
```

```
1: category 1 :: <qgis._core.QgsMarkerSymbol object at 0x7f378ffcd9d8>
2: category 2 :: <qgis._core.QgsMarkerSymbol object at 0x7f378ffcd9d8>
```

Where `value()` is the value used for discrimination between categories, `label()` is a text used for category description and `symbol()` method returns the assigned symbol.

The renderer usually stores also original symbol and color ramp which were used for the classification: `sourceColorRamp()` and `sourceSymbol()` methods.

6.7.3 段階シンボルレンダラー

このレンダラーは先ほど扱ったカテゴリ分けシンボルのレンダラーととても似ていますが、クラスごとの一つの属性値の代わりに領域の値として動作し、そのため数字の属性のみ使うことができます。

レンダラーで使われている領域の多くの情報を見つけるには

```

1 graduated_renderer = QgsGraduatedSymbolRenderer()
2 # Add a few categories
3 graduated_renderer.addClassRange(QgsRendererRange(QgsClassificationRange('class 0-
   ↳100', 0, 100), QgsMarkerSymbol()))
4 graduated_renderer.addClassRange(QgsRendererRange(QgsClassificationRange('class_
   ↳101-200', 101, 200), QgsMarkerSymbol()))
5
6 for ran in graduated_renderer.ranges():
7     print("{} - {}: {}".format(
8         ran.lowerValue(),
9         ran.upperValue(),
10        ran.label(),
11        ran.symbol()
12    ))

```

```

0.0 - 100.0: class 0-100 <qgis._core.QgsMarkerSymbol object at 0x7f8bad281b88>
101.0 - 200.0: class 101-200 <qgis._core.QgsMarkerSymbol object at 0x7f8bad281b88>

```

you can again use the `classAttribute` (to find the classification attribute name), `sourceSymbol` and `sourceColorRamp` methods. Additionally there is the `mode` method which determines how the ranges were created: using equal intervals, quantiles or some other method.

もし連続値シンボルレンダラーを作ろうとしているのであれば次のスニペットの例で書かれているようにします (これはシンプルな二つのクラスを作成するものを取り上げています):

```

1 from qgis.PyQt import QtGui
2
3 myVectorLayer = QgsVectorLayer("testdata/airports.shp", "airports", "ogr")
4 myTargetField = 'ELEV'
5 myRangeList = []
6 myOpacity = 1
7 # Make our first symbol and range...
8 myMin = 0.0
9 myMax = 50.0
10 myLabel = 'Group 1'
11 myColour = QtGui.QColor('#ffee00')
12 mySymbol1 = QgsSymbol.defaultSymbol(myVectorLayer.geometryType())
13 mySymbol1.setColor(myColour)
14 mySymbol1.setOpacity(myOpacity)
15 myRange1 = QgsRendererRange(myMin, myMax, mySymbol1, myLabel)
16 myRangeList.append(myRange1)
17 #now make another symbol and range...
18 myMin = 50.1
19 myMax = 100

```

(次のページに続く)

(前のページからの続き)

```

20 myLabel = 'Group 2'
21 myColour = QtGui.QColor('#00eeff')
22 mySymbol2 = QgsSymbol.defaultSymbol(
23     myVectorLayer.geometryType())
24 mySymbol2.setColor(myColour)
25 mySymbol2.setOpacity(myOpacity)
26 myRange2 = QgsRendererRange(myMin, myMax, mySymbol2, myLabel)
27 myRangeList.append(myRange2)
28 myRenderer = QgsGraduatedSymbolRenderer('', myRangeList)
29 myClassificationMethod = QgsApplication.classificationMethodRegistry().method(
    ↪ "EqualInterval")
30 myRenderer.setClassificationMethod(myClassificationMethod)
31 myRenderer.setClassAttribute(myTargetField)
32
33 myVectorLayer.setRenderer(myRenderer)

```

6.7.4 シンボルの操作

For representation of symbols, there is `QgsSymbol` base class with three derived classes:

- `QgsMarkerSymbol` --- for point features
- `QgsLineSymbol` --- for line features
- `QgsFillSymbol` --- for polygon features

Every symbol consists of one or more symbol layers (classes derived from `QgsSymbolLayer`). The symbol layers do the actual rendering, the symbol class itself serves only as a container for the symbol layers.

Having an instance of a symbol (e.g. from a renderer), it is possible to explore it: the `type` method says whether it is a marker, line or fill symbol. There is a `dump` method which returns a brief description of the symbol. To get a list of symbol layers:

```

marker_symbol = QgsMarkerSymbol()
for i in range(marker_symbol.symbolLayerCount()):
    lyr = marker_symbol.symbolLayer(i)
    print("{}: {}".format(i, lyr.layerType()))

```

```
0: SimpleMarker
```

To find out symbol's color use `color` method and `setColor` to change its color. With marker symbols additionally you can query for the symbol size and rotation with the `size` and `angle` methods. For line symbols the `width` method returns the line width.

サイズと幅は標準でミリメートルが使われ、角度は度が使われます。

シンボルレイヤーの操作

As said before, symbol layers (subclasses of `QgsSymbolLayer`) determine the appearance of the features. There are several basic symbol layer classes for general use. It is possible to implement new symbol layer types and thus arbitrarily customize how features will be rendered. The `layerType()` method uniquely identifies the symbol layer class --- the basic and default ones are `SimpleMarker`, `SimpleLine` and `SimpleFill` symbol layers types.

You can get a complete list of the types of symbol layers you can create for a given symbol layer class with the following code:

```
1 from qgis.core import QgsSymbolLayerRegistry
2 myRegistry = QgsApplication.symbolLayerRegistry()
3 myMetadata = myRegistry.symbolLayerMetadata("SimpleFill")
4 for item in myRegistry.symbolLayersForType(QgsSymbol.Marker):
5     print(item)
```

```
1 EllipseMarker
2 FilledMarker
3 FontMarker
4 GeometryGenerator
5 RasterMarker
6 SimpleMarker
7 SvgMarker
8 VectorField
```

The `QgsSymbolLayerRegistry` class manages a database of all available symbol layer types.

To access symbol layer data, use its `properties()` method that returns a key-value dictionary of properties which determine the appearance. Each symbol layer type has a specific set of properties that it uses. Additionally, there are the generic methods `color`, `size`, `angle` and `width`, with their setter counterparts. Of course size and angle are available only for marker symbol layers and width for line symbol layers.

カスタムシンボルレイヤータイプの作成

データをどうレンダリングするかをカスタマイズしたいと考えているとします。思うままに地物を描画する独自のシンボルレイヤークラスを作成できます。次の例は指定した半径で赤い円を描画するマーカーを示しています:

```
1 from qgis.core import QgsMarkerSymbolLayer
2 from qgis.PyQt.QtGui import QColor
3
4 class FooSymbolLayer(QgsMarkerSymbolLayer):
5
6     def __init__(self, radius=4.0):
7         QgsMarkerSymbolLayer.__init__(self)
8         self.radius = radius
9         self.color = QColor(255,0,0)
10
```

(次のページに続く)

(前のページからの続き)

```

11 def layerType(self):
12     return "FooMarker"
13
14 def properties(self):
15     return { "radius" : str(self.radius) }
16
17 def startRender(self, context):
18     pass
19
20 def stopRender(self, context):
21     pass
22
23 def renderPoint(self, point, context):
24     # Rendering depends on whether the symbol is selected (QGIS >= 1.5)
25     color = context.selectionColor() if context.selected() else self.color
26     p = context.renderContext().painter()
27     p.setPen(color)
28     p.drawEllipse(point, self.radius, self.radius)
29
30 def clone(self):
31     return FooSymbolLayer(self.radius)

```

The `layerType` method determines the name of the symbol layer; it has to be unique among all symbol layers. The `properties` method is used for persistence of attributes. The `clone` method must return a copy of the symbol layer with all attributes being exactly the same. Finally there are rendering methods: `startRender` is called before rendering the first feature, `stopRender` when the rendering is done, and `renderPoint` is called to do the rendering. The coordinates of the point(s) are already transformed to the output coordinates.

For polylines and polygons the only difference would be in the rendering method: you would use `renderPolyline` which receives a list of lines, while `renderPolygon` receives a list of points on the outer ring as the first parameter and a list of inner rings (or None) as a second parameter.

普通はユーザーに外観をカスタマイズさせるためにシンボルレイヤータイプの属性を設定する GUI を追加すると使いやすくなります: 上記の例であればユーザーは円の半径を設定できます。次のコードはそのようなウィジェットの実装となります:

```

1 from qgis.gui import QgsSymbolLayerWidget
2
3 class FooSymbolLayerWidget(QgsSymbolLayerWidget):
4     def __init__(self, parent=None):
5         QgsSymbolLayerWidget.__init__(self, parent)
6
7         self.layer = None
8
9         # setup a simple UI
10        self.label = QLabel("Radius:")
11        self.spinRadius = QDoubleSpinBox()
12        self.hbox = QHBoxLayout()
13        self.hbox.addWidget(self.label)
14        self.hbox.addWidget(self.spinRadius)

```

(次のページに続く)

```

15     self.setLayout(self.hbox)
16     self.connect(self.spinRadius, SIGNAL("valueChanged(double)"), \
17                 self.radiusChanged)
18
19     def setSymbolLayer(self, layer):
20         if layer.layerType() != "FooMarker":
21             return
22         self.layer = layer
23         self.spinRadius.setValue(layer.radius)
24
25     def symbolLayer(self):
26         return self.layer
27
28     def radiusChanged(self, value):
29         self.layer.radius = value
30         self.emit(SIGNAL("changed()"))

```

This widget can be embedded into the symbol properties dialog. When the symbol layer type is selected in symbol properties dialog, it creates an instance of the symbol layer and an instance of the symbol layer widget. Then it calls the `setSymbolLayer` method to assign the symbol layer to the widget. In that method the widget should update the UI to reflect the attributes of the symbol layer. The `symbolLayer` method is used to retrieve the symbol layer again by the properties dialog to use it for the symbol.

On every change of attributes, the widget should emit the `changed()` signal to let the properties dialog update the symbol preview.

私達は最後につなげるところだけまだ扱っていません: QGIS にこれらの新しいクラスを知らせる方法です。これはレジストリにシンボルレイヤーを追加すれば完了です。レジストリに追加しなくてもシンボルレイヤーを使うことはできますが、いくつかの機能が動かないでしょう: 例えばカスタムシンボルレイヤーを使ってプロジェクトファイルを読み込んだり、GUI でレイヤーの属性を編集できないなど。

シンボルレイヤーのメタデータを作る必要があるでしょう

```

1 from qgis.core import QgsSymbol, QgsSymbolLayerAbstractMetadata, \
2     ↳QgsSymbolLayerRegistry
3
4 class FooSymbolLayerMetadata(QgsSymbolLayerAbstractMetadata):
5
6     def __init__(self):
7         super().__init__("FooMarker", "My new Foo marker", QgsSymbol.Marker)
8
9     def createSymbolLayer(self, props):
10        radius = float(props["radius"]) if "radius" in props else 4.0
11        return FooSymbolLayer(radius)
12
13 QgsApplication.symbolLayerRegistry().addSymbolLayerType(FooSymbolLayerMetadata())

```

You should pass layer type (the same as returned by the layer) and symbol type (marker/line/fill) to the constructor of the parent class. The `createSymbolLayer()` method takes care of creating an instance of symbol layer with attributes specified in the `props` dictionary. And there is the `createSymbolLayerWidget()` method

which returns the settings widget for this symbol layer type.

最後にこのシンボルレイヤーをレジストリに追加します --- これで完了です。

6.7.5 カスタムレンダラーの作成

もし地物をレンダリングするためのシンボルをどう選択するかをカスタマイズしたいのであれば、新しいレンダラーの実装を作ると便利かもしれません。いくつかのユースケースとしてこんなことをしたいのかもしれませんが: フィールドの組み合わせからシンボルを決定する、現在の縮尺に合わせてシンボルのサイズを変更するなどなど。

次のコードは二つのマーカーシンボルを作成して全ての地物からランダムに一つ選ぶ簡単なカスタムレンダラーです

```

1 import random
2 from qgis.core import QgsWkbTypes, QgsSymbol, QgsFeatureRenderer
3
4
5 class RandomRenderer(QgsFeatureRenderer):
6     def __init__(self, syms=None):
7         super().__init__("RandomRenderer")
8         self.syms = syms if syms else [
9             QgsSymbol.defaultSymbol(QgsWkbTypes.geometryType(QgsWkbTypes.Point)),
10            QgsSymbol.defaultSymbol(QgsWkbTypes.geometryType(QgsWkbTypes.Point))
11        ]
12
13    def symbolForFeature(self, feature, context):
14        return random.choice(self.syms)
15
16    def startRender(self, context, fields):
17        super().startRender(context, fields)
18        for s in self.syms:
19            s.startRender(context, fields)
20
21    def stopRender(self, context):
22        super().stopRender(context)
23        for s in self.syms:
24            s.stopRender(context)
25
26    def usedAttributes(self, context):
27        return []
28
29    def clone(self):
30        return RandomRenderer(self.syms)

```

The constructor of the parent `QgsFeatureRenderer` class needs a renderer name (which has to be unique among renderers). The `symbolForFeature` method is the one that decides what symbol will be used for a particular feature. `startRender` and `stopRender` take care of initialization/finalization of symbol rendering. The `usedAttributes` method can return a list of field names that the renderer expects to be present. Finally, the `clone` method should return a copy of the renderer.

Like with symbol layers, it is possible to attach a GUI for configuration of the renderer. It has to be derived from `QgsRendererWidget`. The following sample code creates a button that allows the user to set the first symbol

```

1 from qgis.gui import QgsRendererWidget, QgsColorButton
2
3
4 class RandomRendererWidget(QgsRendererWidget):
5     def __init__(self, layer, style, renderer):
6         super().__init__(layer, style)
7         if renderer is None or renderer.type() != "RandomRenderer":
8             self.r = RandomRenderer()
9         else:
10            self.r = renderer
11            # setup UI
12            self.btn1 = QgsColorButton()
13            self.btn1.setColor(self.r.syms[0].color())
14            self.vbox = QVBoxLayout()
15            self.vbox.addWidget(self.btn1)
16            self.setLayout(self.vbox)
17            self.btn1.colorChanged.connect(self.setColor1)
18
19    def setColor1(self):
20        color = self.btn1.color()
21        if not color.isValid(): return
22        self.r.syms[0].setColor(color)
23
24    def renderer(self):
25        return self.r

```

The constructor receives instances of the active layer (`QgsVectorLayer`), the global style (`QgsStyle`) and the current renderer. If there is no renderer or the renderer has different type, it will be replaced with our new renderer, otherwise we will use the current renderer (which has already the type we need). The widget contents should be updated to show current state of the renderer. When the renderer dialog is accepted, the widget's `renderer` method is called to get the current renderer --- it will be assigned to the layer.

最後のちょっとした作業はレンダラーのメタデータとレジストリへの登録です。これらをしないとレンダラーのレイヤーの読み込みは動かず、ユーザーはレンダラーのリストから選択できないでしょう。では、私達の `RandomRenderer` の例を終わらせましょう

```

1 from qgis.core import (
2     QgsRendererAbstractMetadata,
3     QgsRendererRegistry,
4     QgsApplication
5 )
6
7 class RandomRendererMetadata(QgsRendererAbstractMetadata):
8
9     def __init__(self):
10        super().__init__("RandomRenderer", "Random renderer")
11
12    def createRenderer(self, element):

```

(次のページに続く)

(前のページからの続き)

```

13     return RandomRenderer()
14
15     def createRendererWidget(self, layer, style, renderer):
16         return RandomRendererWidget(layer, style, renderer)
17
18 QgsApplication.rendererRegistry().addRenderer(RandomRendererMetadata())

```

Similarly as with symbol layers, abstract metadata constructor awaits renderer name, name visible for users and optionally name of renderer's icon. The `createRenderer` method passes a `QDomElement` instance that can be used to restore the renderer's state from the DOM tree. The `createRendererWidget` method creates the configuration widget. It does not have to be present or can return `None` if the renderer does not come with GUI.

To associate an icon with the renderer you can assign it in the `QgsRendererAbstractMetadata` constructor as a third (optional) argument --- the base class constructor in the `RandomRendererMetadata` `__init__()` function becomes

```

QgsRendererAbstractMetadata.__init__(self,
    "RandomRenderer",
    "Random renderer",
    QIcon(QPixmap("RandomRendererIcon.png", "png")))

```

The icon can also be associated at any later time using the `setIcon` method of the metadata class. The icon can be loaded from a file (as shown above) or can be loaded from a `Qt resource` (PyQt5 includes `.qrc` compiler for Python).

6.8 より詳しいトピック

TODO:

- シンボルの作成や修正
- working with style (`QgsStyle`)
- working with color ramps (`QgsColorRamp`)
- シンボルレイヤーとレンダラーのレジストリを調べる方法

The code snippets on this page need the following imports if you're outside the pyqgis console:

```

1 from qgis.core import (
2     QgsGeometry,
3     QgsPoint,
4     QgsPointXY,
5     QgsWkbTypes,
6     QgsProject,
7     QgsFeatureRequest,
8     QgsVectorLayer,
9     QgsDistanceArea,

```

(次のページに続く)

(前のページからの続き)

```
10     QgsUnitTypes,  
11 )
```

第 7 章

ジオメトリの操作

- ジオメトリの構成
- ジオメトリにアクセス
- ジオメトリの述語と操作

Points, linestrings and polygons that represent a spatial feature are commonly referred to as geometries. In QGIS they are represented with the `QgsGeometry` class.

時には 1 つのジオメトリは実際に単純な (シングルパート) ジオメトリの集合です。このような幾何学的形状は、マルチパートジオメトリと呼ばれています。単純ジオメトリが 1 種類だけ含まれている場合は、マルチポイント、マルチラインまたはマルチポリゴンと呼んでいます。例えば、複数の島からなる国は、マルチポリゴンとして表現できます。

ジオメトリの座標値はどの座標参照系 (CRS) も利用できます。レイヤーから地物を持ってきたときに、ジオメトリの座標値はレイヤーの CRS のものを持つでしょう。

Description and specifications of all possible geometries construction and relationships are available in the [OGC Simple Feature Access Standards](#) for advanced details.

7.1 ジオメトリの構成

PyQGIS provides several options for creating a geometry:

- 座標から

```
1 gPnt = QgsGeometry.fromPointXY(QgsPointXY(1,1))
2 print(gPnt)
3 gLine = QgsGeometry.fromPolyline([QgsPoint(1, 1), QgsPoint(2, 2)])
4 print(gLine)
5 gPolygon = QgsGeometry.fromPolygonXY([[QgsPointXY(1, 1),
6     QgsPointXY(2, 2), QgsPointXY(2, 1)]])
7 print(gPolygon)
```

Coordinates are given using `QgsPoint` class or `QgsPointXY` class. The difference between these classes is that `QgsPoint` supports M and Z dimensions.

A Polyline (Linestring) is represented by a list of points.

A Polygon is represented by a list of linear rings (i.e. closed linestrings). The first ring is the outer ring (boundary), optional subsequent rings are holes in the polygon. Note that unlike some programs, QGIS will close the ring for you so there is no need to duplicate the first point as the last.

マルチパートジオメトリはさらに上のレベルです: マルチポイントはポイントのリストで、マルチラインストリングはラインストリングのリストで、マルチポリゴンはポリゴンのリストです。

- well-known テキスト (WKT) から

```
geom = QgsGeometry.fromWkt("POINT(3 4)")
print(geom)
```

- Well-Known バイナリ (WKB) から

```
1 g = QgsGeometry()
2 wkb = bytes.fromhex("01010000000000000000000045400000000000001440")
3 g.fromWkb(wkb)
4
5 # print WKT representation of the geometry
6 print(g.asWkt())
```

7.2 ジオメトリにアクセス

First, you should find out the geometry type. The `wkbType()` method is the one to use. It returns a value from the `QgsWkbTypes.Type` enumeration.

```
1 if gPnt.wkbType() == QgsWkbTypes.Point:
2     print(gPnt.wkbType())
3     # output: 1 for Point
4 if gLine.wkbType() == QgsWkbTypes.LineString:
5     print(gLine.wkbType())
6     # output: 2 for LineString
7 if gPolygon.wkbType() == QgsWkbTypes.Polygon:
8     print(gPolygon.wkbType())
9     # output: 3 for Polygon
```

As an alternative, one can use the `type()` method which returns a value from the `QgsWkbTypes.GeometryType` enumeration.

You can use the `displayString()` function to get a human readable geometry type.

```
1 print(QgsWkbTypes.displayString(gPnt.wkbType()))
2 # output: 'Point'
3 print(QgsWkbTypes.displayString(gLine.wkbType()))
```

(次のページに続く)

(前のページからの続き)

```

4 # output: 'LineString'
5 print(QgsWkbTypes.displayString(gPolygon.wkbType()))
6 # output: 'Polygon'

```

```

Point
LineString
Polygon

```

There is also a helper function `isMultipart()` to find out whether a geometry is multipart or not.

To extract information from a geometry there are accessor functions for every vector type. Here's an example on how to use these accessors:

```

1 print(gPnt.asPoint())
2 # output: <QgsPointXY: POINT(1 1)>
3 print(gLine.asPolyline())
4 # output: [<QgsPointXY: POINT(1 1)>, <QgsPointXY: POINT(2 2)>]
5 print(gPolygon.asPolygon())
6 # output: [[<QgsPointXY: POINT(1 1)>, <QgsPointXY: POINT(2 2)>, <QgsPointXY:
↳POINT(2 1)>, <QgsPointXY: POINT(1 1)>]]

```

注釈: The tuples (x,y) are not real tuples, they are `QgsPoint` objects, the values are accessible with `x()` and `y()` methods.

For multipart geometries there are similar accessor functions: `asMultiPoint()`, `asMultiPolyline()` and `asMultiPolygon()`.

7.3 ジオメトリの述語と操作

QGIS uses GEOS library for advanced geometry operations such as geometry predicates (`contains()`, `intersects()`, ...) and set operations (`combine()`, `difference()`, ...). It can also compute geometric properties of geometries, such as area (in the case of polygons) or lengths (for polygons and lines).

Let's see an example that combines iterating over the features in a given layer and performing some geometric computations based on their geometries. The below code will compute and print the area and perimeter of each country in the `countries` layer within our tutorial QGIS project.

The following code assumes `layer` is a `QgsVectorLayer` object that has Polygon feature type.

```

1 # let's access the 'countries' layer
2 layer = QgsProject.instance().mapLayersByName('countries')[0]
3
4 # let's filter for countries that begin with Z, then get their features
5 query = '"name" LIKE \'Zu%\''
6 features = layer.getFeatures(QgsFeatureRequest().setFilterExpression(query))

```

(次のページに続く)

```

7
8 # now loop through the features, perform geometry computation and print the results
9 for f in features:
10     geom = f.geometry()
11     name = f.attribute('NAME')
12     print(name)
13     print('Area: ', geom.area())
14     print('Perimeter: ', geom.length())

```

```

1 Zubin Potok
2 Area: 0.040717371293465573
3 Perimeter: 0.9406133328077781
4 Zulia
5 Area: 3.708060762610232
6 Perimeter: 17.172123598311487
7 Zuid-Holland
8 Area: 0.4204687950359031
9 Perimeter: 4.098878517120812
10 Zug
11 Area: 0.027573510374275363
12 Perimeter: 0.7756605461489624

```

Now you have calculated and printed the areas and perimeters of the geometries. You may however quickly notice that the values are strange. That is because areas and perimeters don't take CRS into account when computed using the `area()` and `length()` methods from the `QgsGeometry` class. For a more powerful area and distance calculation, the `QgsDistanceArea` class can be used, which can perform ellipsoid based calculations:

The following code assumes `layer` is a `QgsVectorLayer` object that has Polygon feature type.

```

1 d = QgsDistanceArea()
2 d.setEllipsoid('WGS84')
3
4 layer = QgsProject.instance().mapLayersByName('countries')[0]
5
6 # let's filter for countries that begin with Z, then get their features
7 query = '"name" LIKE \'Zu%\''
8 features = layer.getFeatures(QgsFeatureRequest().setFilterExpression(query))
9
10 for f in features:
11     geom = f.geometry()
12     name = f.attribute('NAME')
13     print(name)
14     print("Perimeter (m):", d.measurePerimeter(geom))
15     print("Area (m2):", d.measureArea(geom))
16
17 # let's calculate and print the area again, but this time in square kilometers
18 print("Area (km2):", d.convertAreaMeasurement(d.measureArea(geom), QgsUnitTypes.
    ↪AreaSquareKilometers))

```



```

1 Zubin Potok
2 Perimeter (m): 87581.40256396442
3 Area (m2): 369302069.18814206
4 Area (km2): 369.30206918814207
5 Zulia
6 Perimeter (m): 1891227.0945423362
7 Area (m2): 44973645460.19726
8 Area (km2): 44973.64546019726
9 Zuid-Holland
10 Perimeter (m): 331941.8000214341
11 Area (m2): 3217213408.4100943
12 Area (km2): 3217.213408410094
13 Zug
14 Perimeter (m): 67440.22483063207
15 Area (m2): 232457391.52097562
16 Area (km2): 232.45739152097562

```

Alternatively, you may want to know the distance and bearing between two points.

```

1 d = QgsDistanceArea()
2 d.setEllipsoid('WGS84')
3
4 # Let's create two points.
5 # Santa claus is a workaholic and needs a summer break,
6 # lets see how far is Tenerife from his home
7 santa = QgsPointXY(25.847899, 66.543456)
8 tenerife = QgsPointXY(-16.5735, 28.0443)
9
10 print("Distance in meters: ", d.measureLine(santa, tenerife))

```

QGIS に含まれているアルゴリズムの多くの例を見つけて、これらのメソッドをベクターデータを分析し変換するために使用できます。ここにそれらのコードのいくつかへのリンクを記載します。

- Distance and area using the `QgsDistanceArea` class: [Distance matrix algorithm](#)
- Lines to polygons algorithm

The code snippets on this page need the following imports if you're outside the pyqgis console:

```

1 from qgis.core import (
2     QgsCoordinateReferenceSystem,
3     QgsCoordinateTransform,
4     QgsProject,
5     QgsPointXY,
6 )

```


第 8 章

投影法サポート

8.1 空間参照系

Coordinate reference systems (CRS) are encapsulated by the `QgsCoordinateReferenceSystem` class. Instances of this class can be created in several different ways:

- CRS を ID によって指定する

```
# EPSG 4326 is allocated for WGS84
crs = QgsCoordinateReferenceSystem("EPSG:4326")
assert crs.isValid()
```

QGIS supports different CRS identifiers with the following formats:

- EPSG:<code> --- ID assigned by the EPSG organization - handled with `createFromOgcWms()`
- POSTGIS:<srid>--- ID used in PostGIS databases - handled with `createFromSrid()`
- INTERNAL:<srsid> --- ID used in the internal QGIS database - handled with `createFromSrsId()`
- PROJ:<proj> - handled with `createFromProj()`
- WKT:<wkt> - handled with `createFromWkt()`

If no prefix is specified, WKT definition is assumed.

- CRS を well-known テキスト (WKT) で指定する

```
1 wkt = 'GEOGCS["WGS84", DATUM["WGS84", SPHEROID["WGS84", 6378137.0, 298.
↪257223563]],' \
2     'PRIMEM["Greenwich", 0.0], UNIT["degree",0.017453292519943295],' \
3     'AXIS["Longitude",EAST], AXIS["Latitude",NORTH]]'
4 crs = QgsCoordinateReferenceSystem(wkt)
5 assert crs.isValid()
```

- create an invalid CRS and then use one of the `create*` functions to initialize it. In the following example we use a Proj string to initialize the projection.

```
crs = QgsCoordinateReferenceSystem()
crs.createFromProj("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs")
assert crs.isValid()
```

It's wise to check whether creation (i.e. lookup in the database) of the CRS has been successful: `isValid()` must return `True`.

Note that for initialization of spatial reference systems QGIS needs to look up appropriate values in its internal database `srs.db`. Thus in case you create an independent application you need to set paths correctly with `QgsApplication.setPrefixPath()`, otherwise it will fail to find the database. If you are running the commands from the QGIS Python console or developing a plugin you do not care: everything is already set up for you.

Accessing spatial reference system information:

```
1 crs = QgsCoordinateReferenceSystem("EPSG:4326")
2
3 print("QGIS CRS ID:", crs.srsid())
4 print("PostGIS SRID:", crs.postgisSrid())
5 print("Description:", crs.description())
6 print("Projection Acronym:", crs.projectionAcronym())
7 print("Ellipsoid Acronym:", crs.ellipsoidAcronym())
8 print("Proj String:", crs.toProj())
9 # check whether it's geographic or projected coordinate system
10 print("Is geographic:", crs.isGeographic())
11 # check type of map units in this CRS (values defined in QGis::units enum)
12 print("Map units:", crs.mapUnits())
```

Output:

```
1 QGIS CRS ID: 3452
2 PostGIS SRID: 4326
3 Description: WGS 84
4 Projection Acronym: longlat
5 Ellipsoid Acronym: WGS84
6 Proj String: +proj=longlat +datum=WGS84 +no_defs
7 Is geographic: True
8 Map units: 6
```

8.2 CRS Transformation

You can do transformation between different spatial reference systems by using the `QgsCoordinateTransform` class. The easiest way to use it is to create a source and destination CRS and construct a `QgsCoordinateTransform` instance with them and the current project. Then just repeatedly call `transform()` function to do the transformation. By default it does forward transformation, but it is capable to do also inverse transformation.

```

1 crsSrc = QgsCoordinateReferenceSystem("EPSG:4326")    # WGS 84
2 crsDest = QgsCoordinateReferenceSystem("EPSG:32633")  # WGS 84 / UTM zone 33N
3 transformContext = QgsProject.instance().transformContext()
4 xform = QgsCoordinateTransform(crsSrc, crsDest, transformContext)
5
6 # forward transformation: src -> dest
7 pt1 = xform.transform(QgsPointXY(18,5))
8 print("Transformed point:", pt1)
9
10 # inverse transformation: dest -> src
11 pt2 = xform.transform(pt1, QgsCoordinateTransform.ReverseTransform)
12 print("Transformed back:", pt2)

```

Output:

```

Transformed point: <QgsPointXY: POINT(832713.79873844375833869 553423.
↔98688333143945783)>
Transformed back: <QgsPointXY: POINT(18 5)>

```

The code snippets on this page need the following imports if you're outside the pyqgis console:

```

1 from qgis.PyQt.QtGui import (
2     QColor,
3 )
4
5 from qgis.PyQt.QtCore import Qt, QRectF
6
7 from qgis.core import (
8     QgsVectorLayer,
9     QgsPoint,
10    QgsPointXY,
11    QgsProject,
12    QgsGeometry,
13    QgsMapRendererJob,
14 )
15
16 from qgis.gui import (
17     QgsMapCanvas,
18     QgsVertexMarker,
19     QgsMapCanvasItem,
20     QgsRubberBand,
21 )

```


第 9 章

マップキャンバスを使う

- 地図キャンバスを埋め込む
- ラバーバンドと頂点マーカー
- 地図キャンバスで地図ツールを使用する
- カスタム地図ツールを書く
- カスタム地図キャンバスアイテムを書く

The Map canvas widget is probably the most important widget within QGIS because it shows the map composed from overlaid map layers and allows interaction with the map and layers. The canvas always shows a part of the map defined by the current canvas extent. The interaction is done through the use of **map tools**: there are tools for panning, zooming, identifying layers, measuring, vector editing and others. Similar to other graphics programs, there is always one tool active and the user can switch between the available tools.

マップキャンバスは、`qgis.gui` モジュール内の `QgsMapCanvas` クラスで実装されています。実装は Qt Graphics View フレームワークに基づいています。このフレームワークは一般的に、カスタムグラフィックスアイテムが配置され、ユーザがそれら进行操作することができるサーフェスとビューを提供します。ここでは、グラフィックスシーン、ビュー、アイテムの概念を理解できるほど Qt に精通していることを前提とします。そうでない場合は [フレームワークの概要](#) をお読みください。

Whenever the map has been panned, zoomed in/out (or some other action that triggers a refresh), the map is rendered again within the current extent. The layers are rendered to an image (using the `QgsMapRendererJob` class) and that image is displayed on the canvas. The `QgsMapCanvas` class also controls refreshing of the rendered map. Besides this item which acts as a background, there may be more **map canvas items**.

Typical map canvas items are rubber bands (used for measuring, vector editing etc.) or vertex markers. The canvas items are usually used to give visual feedback for map tools, for example, when creating a new polygon, the map tool creates a rubber band canvas item that shows the current shape of the polygon. All map canvas items are subclasses of `QgsMapCanvasItem` which adds some more functionality to the basic `QGraphicsItem` objects.

要約すると、地図キャンバスアーキテクチャは3つのコンセプトからなります：

- map canvas --- 地図の可視化
- map canvas items --- additional items that can be displayed on the map canvas
- map tools --- for interaction with the map canvas

9.1 地図キャンバスを埋め込む

Map canvas is a widget like any other Qt widget, so using it is as simple as creating and showing it.

```
canvas = QgsMapCanvas()
canvas.show()
```

This produces a standalone window with map canvas. It can be also embedded into an existing widget or window. When using .ui files and Qt Designer, place a QWidget on the form and promote it to a new class: set QgsMapCanvas as class name and set qgis.gui as header file. The pyuic5 utility will take care of it. This is a very convenient way of embedding the canvas. The other possibility is to manually write the code to construct map canvas and other widgets (as children of a main window or dialog) and create a layout.

デフォルトでは、地図キャンバスの背景色は黒でありアンチエイリアスは使用されません。背景を白に設定し、投影をなめらかにするためのアンチエイリアスを有効にするには

```
canvas.setCanvasColor(Qt.white)
canvas.enableAntiAliasing(True)
```

(In case you are wondering, Qt comes from PyQt.QtCore module and Qt.white is one of the predefined QColor instances.)

それでは、地図レイヤを追加していきましょう。まずレイヤを開いて、現在のプロジェクトに追加します。次に、キャンバスの範囲を設定し、キャンバスのレイヤのリストを設定します。

```
1 vlayer = QgsVectorLayer('testdata/airports.shp', "Airports layer", "ogr")
2 if not vlayer.isValid():
3     print("Layer failed to load!")
4
5 # add layer to the registry
6 QgsProject.instance().addMapLayer(vlayer)
7
8 # set extent to the extent of our layer
9 canvas.setExtent(vlayer.extent())
10
11 # set the map canvas layer set
12 canvas.setLayers([vlayer])
```

これらのコマンドを実行した後、キャンバスには読み込んだレイヤーが表示されているはずです。

9.2 ラバーバンドと頂点マーカー

To show some additional data on top of the map in canvas, use map canvas items. It is possible to create custom canvas item classes (covered below), however there are two useful canvas item classes for convenience: `QgsRubberBand` for drawing polylines or polygons, and `QgsVertexMarker` for drawing points. They both work with map coordinates, so the shape is moved/scaled automatically when the canvas is being panned or zoomed.

To show a polyline:

```
r = QgsRubberBand(canvas, False) # False = not a polygon
points = [QgsPoint(-100, 45), QgsPoint(10, 60), QgsPoint(120, 45)]
r.setToGeometry(QgsGeometry.fromPolyline(points), None)
```

ポリゴンを表示するには

```
r = QgsRubberBand(canvas, True) # True = a polygon
points = [[QgsPointXY(-100, 35), QgsPointXY(10, 50), QgsPointXY(120, 35)]]
r.setToGeometry(QgsGeometry.fromPolygonXY(points), None)
```

ポリゴンの点が普通のリストではないことに注意してください。実際には、ポリゴンの線状のリングを含有するリングのリストです：最初のリングは外側の境界であり、さらに（オプションの）リングはポリゴンの穴に対応します。

ラバーバンドはいくらカスタマイズできます、すなわち、その色と線幅を変更することが可能です

```
r.setColor(QColor(0, 0, 255))
r.setWidth(3)
```

The canvas items are bound to the canvas scene. To temporarily hide them (and show them again), use the `hide()` and `show()` combo. To completely remove the item, you have to remove it from the scene of the canvas

```
canvas.scene().removeItem(r)
```

(C++ ではアイテムを削除することだけ可能ですが、Python では `del r` は参照を削除するだけでありオブジェクトはキャンバスの所有物なのでそのまま残ります)

Rubber band can be also used for drawing points, but the `QgsVertexMarker` class is better suited for this (`QgsRubberBand` would only draw a rectangle around the desired point).

You can use the vertex marker like this:

```
m = QgsVertexMarker(canvas)
m.setCenter(QgsPointXY(10, 40))
```

This will draw a red cross on position **[10,45]**. It is possible to customize the icon type, size, color and pen width

```
m.setColor(QColor(0, 255, 0))
m.setIconSize(5)
```

(次のページに続く)

```
m.setIconType(QgsVertexMarker.ICON_BOX) # or ICON_CROSS, ICON_X
m.setPenWidth(3)
```

For temporary hiding of vertex markers and removing them from canvas, use the same methods as for rubber bands.

9.3 地図キャンバスで地図ツールを使用する

以下の例では、地図キャンバスと、地図のパンニングとズームのための基本的な地図ツールを含むウィンドウを作成します。パンニングは `QgsMapToolPan` で行い、ズームイン/ズームアウトは `QgsMapToolZoom` インスタンスのペアで行います。アクションはチェック可能に設定されており、後からツールに割り当てられ、アクションのチェック済み/チェック解除状態の自動処理を可能にします -- 地図ツールがアクティブになると、そのアクションは選択されたら印が付き、前の地図ツールのアクションは選択解除されます。地図ツールは `setMapTool()` メソッドを使って起動します。

```
1 from qgis.gui import *
2 from qgis.PyQt.QtWidgets import QAction, QMainWindow
3 from qgis.PyQt.QtCore import Qt
4
5 class MyWnd(QMainWindow):
6     def __init__(self, layer):
7         QMainWindow.__init__(self)
8
9         self.canvas = QgsMapCanvas()
10        self.canvas.setCanvasColor(Qt.white)
11
12        self.canvas.setExtent(layer.extent())
13        self.canvas.setLayers([layer])
14
15        self.setCentralWidget(self.canvas)
16
17        self.actionZoomIn = QAction("Zoom in", self)
18        self.actionZoomOut = QAction("Zoom out", self)
19        self.actionPan = QAction("Pan", self)
20
21        self.actionZoomIn.setCheckable(True)
22        self.actionZoomOut.setCheckable(True)
23        self.actionPan.setCheckable(True)
24
25        self.actionZoomIn.triggered.connect(self.zoomIn)
26        self.actionZoomOut.triggered.connect(self.zoomOut)
27        self.actionPan.triggered.connect(self.pan)
28
29        self.toolbar = self.addToolBar("Canvas actions")
30        self.toolbar.addAction(self.actionZoomIn)
31        self.toolbar.addAction(self.actionZoomOut)
32        self.toolbar.addAction(self.actionPan)
33
```

(次のページに続く)

(前のページからの続き)

```

34     # create the map tools
35     self.toolPan = QgsMapToolPan(self.canvas)
36     self.toolPan.setAction(self.actionPan)
37     self.toolZoomIn = QgsMapToolZoom(self.canvas, False) # false = in
38     self.toolZoomIn.setAction(self.actionZoomIn)
39     self.toolZoomOut = QgsMapToolZoom(self.canvas, True) # true = out
40     self.toolZoomOut.setAction(self.actionZoomOut)
41
42     self.pan()
43
44     def zoomIn(self):
45         self.canvas.setMapTool(self.toolZoomIn)
46
47     def zoomOut(self):
48         self.canvas.setMapTool(self.toolZoomOut)
49
50     def pan(self):
51         self.canvas.setMapTool(self.toolPan)

```

You can try the above code in the Python console editor. To invoke the canvas window, add the following lines to instantiate the `MyWnd` class. They will render the currently selected layer on the newly created canvas

```

w = MyWnd iface.activeLayer()
w.show()

```

9.4 カスタム地図ツールを書く

You can write your custom tools, to implement a custom behavior to actions performed by users on the canvas.

Map tools should inherit from the `QgsMapTool` class or any derived class, and selected as active tools in the canvas using the `setMapTool()` method as we have already seen.

キャンバスをクリックしてドラッグすることで矩形範囲を定義できる地図ツールの例を次に示します。矩形が定義されると、境界座標がコンソールに表示されます。前述のラバーバンド要素を使用して、選択されている矩形が定義されていることを示します。

```

1 class RectangleMapTool(QgsMapToolEmitPoint):
2     def __init__(self, canvas):
3         self.canvas = canvas
4         QgsMapToolEmitPoint.__init__(self, self.canvas)
5         self.rubberBand = QgsRubberBand(self.canvas, True)
6         self.rubberBand.setColor(Qt.red)
7         self.rubberBand.setWidth(1)
8         self.reset()
9
10    def reset(self):
11        self.startPoint = self.endPoint = None
12        self.isEmittingPoint = False

```

(次のページに続く)

```
13     self.rubberBand.reset(True)
14
15     def canvasPressEvent(self, e):
16         self.startPoint = self.toMapCoordinates(e.pos())
17         self.endPoint = self.startPoint
18         self.isEmittingPoint = True
19         self.showRect(self.startPoint, self.endPoint)
20
21     def canvasReleaseEvent(self, e):
22         self.isEmittingPoint = False
23         r = self.rectangle()
24         if r is not None:
25             print("Rectangle:", r.xMinimum(),
26                   r.yMinimum(), r.xMaximum(), r.yMaximum()
27                   )
28
29     def canvasMoveEvent(self, e):
30         if not self.isEmittingPoint:
31             return
32
33         self.endPoint = self.toMapCoordinates(e.pos())
34         self.showRect(self.startPoint, self.endPoint)
35
36     def showRect(self, startPoint, endPoint):
37         self.rubberBand.reset(Qgis.Polygon)
38         if startPoint.x() == endPoint.x() or startPoint.y() == endPoint.y():
39             return
40
41         point1 = QgsPoint(startPoint.x(), startPoint.y())
42         point2 = QgsPoint(startPoint.x(), endPoint.y())
43         point3 = QgsPoint(endPoint.x(), endPoint.y())
44         point4 = QgsPoint(endPoint.x(), startPoint.y())
45
46         self.rubberBand.addPoint(point1, False)
47         self.rubberBand.addPoint(point2, False)
48         self.rubberBand.addPoint(point3, False)
49         self.rubberBand.addPoint(point4, True) # true to update canvas
50         self.rubberBand.show()
51
52     def rectangle(self):
53         if self.startPoint is None or self.endPoint is None:
54             return None
55         elif (self.startPoint.x() == self.endPoint.x() or \
56               self.startPoint.y() == self.endPoint.y()):
57             return None
58
59         return QgsRectangle(self.startPoint, self.endPoint)
60
61     def deactivate(self):
62         QgsMapTool.deactivate(self)
63         self.deactivated.emit()
```

9.5 カスタム地図キャンバスアイテムを書く

Here is an example of a custom canvas item that draws a circle:

```

1 class CircleCanvasItem(QgsMapCanvasItem):
2     def __init__(self, canvas):
3         super().__init__(canvas)
4         self.center = QgsPoint(0, 0)
5         self.size = 100
6
7     def setCenter(self, center):
8         self.center = center
9
10    def center(self):
11        return self.center
12
13    def setSize(self, size):
14        self.size = size
15
16    def size(self):
17        return self.size
18
19    def boundingRect(self):
20        return QRectF(self.center.x() - self.size/2,
21                      self.center.y() - self.size/2,
22                      self.center.x() + self.size/2,
23                      self.center.y() + self.size/2)
24
25    def paint(self, painter, option, widget):
26        path = QPainterPath()
27        path.moveTo(self.center.x(), self.center.y());
28        path.arcTo(self.boundingRect(), 0.0, 360.0)
29        painter.fillPath(path, QColor("red"))
30
31
32    # Using the custom item:
33    item = CircleCanvasItem(iface.mapCanvas())
34    item.setCenter(QgsPointXY(200,200))
35    item.setSize(80)

```

The code snippets on this page need the following imports:

```

1 import os
2
3 from qgis.core import (
4     QgsGeometry,
5     QgsMapSettings,
6     QgsPrintLayout,
7     QgsMapSettings,
8     QgsMapRendererParallelJob,
9     QgsLayoutItemLabel,
10    QgsLayoutItemLegend,

```

(次のページに続く)

```
11     QgsLayoutItemMap,  
12     QgsLayoutItemPolygon,  
13     QgsLayoutItemScaleBar,  
14     QgsLayoutExporter,  
15     QgsLayoutItem,  
16     QgsLayoutPoint,  
17     QgsLayoutSize,  
18     QgsUnitTypes,  
19     QgsProject,  
20     QgsFillSymbol,  
21 )  
22  
23 from qgis.PyQt.QtGui import (  
24     QPolygonF,  
25     QColor,  
26 )  
27  
28 from qgis.PyQt.QtCore import (  
29     QPointF,  
30     QRectF,  
31     QSize,  
32 )
```

第 10 章

地図のレンダリングと印刷

- 単純なレンダリング
- 異なる CRS を持つレイヤーをレンダリングする
- 印刷レイアウトを使用して出力する
 - *Exporting the layout*
 - *Exporting a layout atlas*

入力データを地図として描画せねばならないときには、総じてふたつのアプローチがあります。 `QgsMapRendererJob` を使って手早く済ませるか、もしくは `:class:`QgsLayout`` クラスで地図を構成し、より精密に調整された出力を作成するかです。

10.1 単純なレンダリング

The rendering is done creating a `QgsMapSettings` object to define the rendering settings, and then constructing a `QgsMapRendererJob` with those settings. The latter is then used to create the resulting image.

こちらがサンプルです。

```
1 image_location = os.path.join(QgsProject.instance().homePath(), "render.png")
2
3 vlayer = iface.activeLayer()
4 settings = QgsMapSettings()
5 settings.setLayers([vlayer])
6 settings.setBackgroundColor(QColor(255, 255, 255))
7 settings.setOutputSize(QSize(800, 600))
8 settings.setExtent(vlayer.extent())
9
10 render = QgsMapRendererParallelJob(settings)
11
12 def finished():
```

(次のページに続く)

```
13     img = render.renderedImage()
14     # save the image; e.g. img.save("/Users/myuser/render.png", "png")
15     img.save(image_location, "png")
16
17 render.finished.connect(finished)
18
19 render.start()
```

10.2 異なる CRS を持つレイヤーをレンダリングする

レイヤが複数あり、それぞれの CRS が異なっている場合は、上記の単純な例ではおそらく求める結果は得られません。範囲計算から正しい値を得るためには、明示的に目的の CRS を設定する必要があります。

```
layers = [iface.activeLayer()]
settings.setLayers(layers)
settings.setDestinationCrs(layers[0].crs())
```

10.3 印刷レイアウトを使用して出力する

Print layout is a very handy tool if you would like to do a more sophisticated output than the simple rendering shown above. It is possible to create complex map layouts consisting of map views, labels, legend, tables and other elements that are usually present on paper maps. The layouts can be then exported to PDF, raster images or directly printed on a printer.

The layout consists of a bunch of classes. They all belong to the core library. QGIS application has a convenient GUI for placement of the elements, though it is not available in the GUI library. If you are not familiar with [Qt Graphics View framework](#), then you are encouraged to check the documentation now, because the layout is based on it.

The central class of the layout is the `QgsLayout` class, which is derived from the Qt `QGraphicsScene` class. Let us create an instance of it:

```
project = QgsProject()
layout = QgsPrintLayout(project)
layout.initializeDefaults()
```

Now we can add various elements (map, label, ...) to the layout. All these objects are represented by classes that inherit from the base `QgsLayoutItem` class.

Here's a description of some of the main layout items that can be added to a layout.

- map --- このアイテムは地図自体を置くためのライブラリを伝えます。ここでは、地図を作成し、全体の用紙サイズの上に伸ばします


```
map = QgsLayoutItemMap(layout)
layout.addItem(map)
```

- label --- ラベルを表示できます。そのフォント、色、配置及びマージンを変更することが可能です

```
label = QgsLayoutItemLabel(layout)
label.setText("Hello world")
label.adjustSizeToText()
layout.addItem(label)
```

- 凡例

```
legend = QgsLayoutItemLegend(layout)
legend.setLinkedMap(map) # map is an instance of QgsLayoutItemMap
layout.addItem(legend)
```

- スケールバー

```
1 item = QgsLayoutItemScaleBar(layout)
2 item.setStyle('Numeric') # optionally modify the style
3 item.setLinkedMap(map) # map is an instance of QgsLayoutItemMap
4 item.applyDefaultSize()
5 layout.addItem(item)
```

- 矢印
- picture
- 基本図形
- ノードに基づく図形

```
1 polygon = QPolygonF()
2 polygon.append(QPointF(0.0, 0.0))
3 polygon.append(QPointF(100.0, 0.0))
4 polygon.append(QPointF(200.0, 100.0))
5 polygon.append(QPointF(100.0, 200.0))
6
7 polygonItem = QgsLayoutItemPolygon(polygon, layout)
8 layout.addItem(polygonItem)
9
10 props = {}
11 props["color"] = "green"
12 props["style"] = "solid"
13 props["style_border"] = "solid"
14 props["color_border"] = "black"
15 props["width_border"] = "10.0"
16 props["joinstyle"] = "miter"
17
18 symbol = QgsFillSymbol.createSimple(props)
19 polygonItem.setSymbol(symbol)
```

- 表

Once an item is added to the layout, it can be moved and resized:

```
item.attemptMove(QgsLayoutPoint(1.4, 1.8, QgsUnitTypes.LayoutCentimeters))
item.attemptResize(QgsLayoutSize(2.8, 2.2, QgsUnitTypes.LayoutCentimeters))
```

A frame is drawn around each item by default. You can remove it as follows:

```
# for a composer label
label.setFrameEnabled(False)
```

Besides creating the layout items by hand, QGIS has support for layout templates which are essentially compositions with all their items saved to a .qpt file (with XML syntax).

Once the composition is ready (the layout items have been created and added to the composition), we can proceed to produce a raster and/or vector output.

10.3.1 Exporting the layout

To export a layout, the `QgsLayoutExporter` class must be used.

```
1 base_path = os.path.join(QgsProject.instance().homePath())
2 pdf_path = os.path.join(base_path, "output.pdf")
3
4 exporter = QgsLayoutExporter(layout)
5 exporter.exportToPdf(pdf_path, QgsLayoutExporter.PdfExportSettings())
```

Use the `exportToImage()` in case you want to export to an image instead of a PDF file.

10.3.2 Exporting a layout atlas

If you want to export all pages from a layout that has the atlas option configured and enabled, you need to use the `atlas()` method in the exporter (`QgsLayoutExporter`) with small adjustments. In the following example, the pages are exported to PNG images:

```
exporter.exportToImage(layout.atlas(), base_path, 'png', QgsLayoutExporter.
↳ ImageExportSettings())
```

Notice that the outputs will be saved in the base path folder, using the output filename expression configured on atlas.

The code snippets on this page need the following imports if you're outside the pyqgis console:

```
1 from qgis.core import (
2     edit,
3     QgsExpression,
```

(次のページに続く)

(前のページからの続き)

```
4     QgsExpressionContext,  
5     QgsFeature,  
6     QgsFeatureRequest,  
7     QgsField,  
8     QgsFields,  
9     QgsVectorLayer,  
10    QgsPointXY,  
11    QgsGeometry,  
12    QgsProject,  
13    QgsExpressionContextUtils  
14 )
```


第 11 章

式、フィルタ適用および値の算出

- 式を構文解析する
- 式を評価する
 - 基本的な式
 - 地物に関わる式
 - 式を使ってレイヤをフィルタする
- 式エラーを扱う

QGIS では、SQL 風の式の構文解析について少しサポートしています。サポートされるのは SQL 構文の小さなサブセットのみです。式は、ブール述語 (真または偽を返す) または関数 (スカラー値を返す) のどちらかとして評価できます。使用可能な関数の完全なリストについては、ユーザーマニュアル中の `vector_expressions` を参照。

3 つの基本的な型がサポートされています。

- 数 --- 整数および小数。例. 123, 3.14
- 文字列 --- 一重引用符で囲む必要があります: 'hello world'
- 列参照 --- 評価する際に、参照はフィールドの実際の値で置き換えられます。名前はエスケープされません。

次の演算子が利用可能です:

- 算術演算子: +, -, *, /, ^
- 丸括弧: 演算を優先します: (1 + 1) * 3
- 単項のプラスとマイナス: -12, +5
- 数学的関数: sqrt, sin, cos, tan, asin, acos, atan
- 変換関数: to_int, to_real, to_string, to_date

- ジオメトリ関数: \$area, \$length
- ジオメトリ処理関数: \$x、\$y、\$geometry、num_geometries、centroid

以下の述語がサポートされています:

- 比較: =, !=, >, >=, <, <=
- パターンマッチング: LIKE (% と _ を使用), ~ (正規表現)
- 論理述語: AND, OR, NOT
- NULL 値チェック: IS NULL, IS NOT NULL

述語の例:

- $1 + 2 = 3$
- $\sin(\text{angle}) > 0$
- 'Hello' LIKE 'He%'
- $(x > 10 \text{ AND } y > 10) \text{ OR } z = 0$

スカラー式の例:

- $2 ^ 10$
- $\text{sqrt}(\text{val})$
- $\$length + 1$

11.1 式を構文解析する

与えられた式が正しくパースできるかどうかは、以下の例で示す方法で確認します。

```
1 exp = QgsExpression('1 + 1 = 2')
2 assert(not exp.hasParserError())
3
4 exp = QgsExpression('1 + 1 = ')
5 assert(exp.hasParserError())
6
7 assert(exp.parserErrorString() == '\nsyntax error, unexpected $end')
```

11.2 式を評価する

式は、例えば地物をフィルタしたり、新しいフィールド値を計算するなど、異なったコンテキストで使うことができます。いずれの場合においても、式は評価されなければなりません。つまり式の値は、単純な算術式から集約式まで、指定された計算ステップを実行することによって計算されます。

11.2.1 基本的な式

この基本的な式は 1、つまり true と評価されます。

```
exp = QgsExpression('1 + 1 = 2')
assert (exp.evaluate()) # exp.evaluate() returns 1 and assert() recognizes this as
↳ True
```

11.2.2 地物に関わる式

地物に関わる式を評価するためには、式が地物のフィールド値にアクセスできるようにするために、`QgsExpressionContext` オブジェクトを生成し、評価関数に渡さなければなりません。

以下の例は、"Column" という名前のフィールドを持つ地物を作り、この地物を式のコンテキストに加える方法を示しています。

```
1 fields = QgsFields()
2 field = QgsField('Column')
3 fields.append(field)
4 feature = QgsFeature()
5 feature.setFields(fields)
6 feature.setAttribute(0, 99)
7
8 exp = QgsExpression('"Column"')
9 context = QgsExpressionContext()
10 context.setFeature(feature)
11 assert (exp.evaluate(context) == 99)
```

以下の例は、ベクターレイヤのコンテキストにおいて、新しいフィールド値を計算するためにどのように式を使うかを示す、より完成された例です。

```
1 from qgis.PyQt.QtCore import QVariant
2
3 # create a vector layer
4 vl = QgsVectorLayer("Point", "Companies", "memory")
5 pr = vl.dataProvider()
6 pr.addAttributes([QgsField("Name", QVariant.String),
7                     QgsField("Employees", QVariant.Int),
8                     QgsField("Revenue", QVariant.Double),
9                     QgsField("Rev. per employee", QVariant.Double),
10                    QgsField("Sum", QVariant.Double),
```

(次のページに続く)

```
11         QgsField("Fun", QVariant.Double)])
12 vl.updateFields()
13
14 # add data to the first three fields
15 my_data = [
16     {'x': 0, 'y': 0, 'name': 'ABC', 'emp': 10, 'rev': 100.1},
17     {'x': 1, 'y': 1, 'name': 'DEF', 'emp': 2, 'rev': 50.5},
18     {'x': 5, 'y': 5, 'name': 'GHI', 'emp': 100, 'rev': 725.9}]
19
20 for rec in my_data:
21     f = QgsFeature()
22     pt = QgsPointXY(rec['x'], rec['y'])
23     f.setGeometry(QgsGeometry.fromPointXY(pt))
24     f.setAttributes([rec['name'], rec['emp'], rec['rev']])
25     pr.addFeature(f)
26
27 vl.updateExtents()
28 QgsProject.instance().addMapLayer(vl)
29
30 # The first expression computes the revenue per employee.
31 # The second one computes the sum of all revenue values in the layer.
32 # The final third expression doesn't really make sense but illustrates
33 # the fact that we can use a wide range of expression functions, such
34 # as area and buffer in our expressions:
35 expression1 = QgsExpression('"Revenue"/"Employees"')
36 expression2 = QgsExpression('sum("Revenue")')
37 expression3 = QgsExpression('area(buffer($geometry,"Employees"))')
38
39 # QgsExpressionContextUtils.globalProjectLayerScopes() is a convenience
40 # function that adds the global, project, and layer scopes all at once.
41 # Alternatively, those scopes can also be added manually. In any case,
42 # it is important to always go from "most generic" to "most specific"
43 # scope, i.e. from global to project to layer
44 context = QgsExpressionContext()
45 context.appendScopes(QgsExpressionContextUtils.globalProjectLayerScopes(vl))
46
47 with edit(vl):
48     for f in vl.getFeatures():
49         context.setFeature(f)
50         f['Rev. per employee'] = expression1.evaluate(context)
51         f['Sum'] = expression2.evaluate(context)
52         f['Fun'] = expression3.evaluate(context)
53         vl.updateFeature(f)
54
55 print( f['Sum'])
```

876.5

11.2.3 式を使ってレイヤをフィルタする

次の例はレイヤーをフィルタリングして述語に一致する任意の地物を返します。

```

1 layer = QgsVectorLayer("Point?field=Test:integer",
2                       "addfeat", "memory")
3
4 layer.startEditing()
5
6 for i in range(10):
7     feature = QgsFeature()
8     feature.setAttributes([i])
9     assert(layer.addFeature(feature))
10 layer.commitChanges()
11
12 expression = 'Test >= 3'
13 request = QgsFeatureRequest().setFilterExpression(expression)
14
15 matches = 0
16 for f in layer.getFeatures(request):
17     matches += 1
18
19 assert(matches == 7)

```

11.3 式エラーを扱う

式をパースする過程、もしくは式を評価する過程で、式に関連するエラーが生じる可能性があります。

```

1 exp = QgsExpression("1 + 1 = 2")
2 if exp.hasParserError():
3     raise Exception(exp.parserErrorString())
4
5 value = exp.evaluate()
6 if exp.hasEvalError():
7     raise ValueError(exp.evalErrorString())

```

The code snippets on this page need the following imports if you're outside the pyqgis console:

```

1 from qgis.core import (
2     QgsProject,
3     QgsSettings,
4     QgsVectorLayer
5 )

```


第 12 章

設定の読み込みと保存

多くの場合、プラグインでいくつかの変数が保存されて、そのプラグインを次に実行したときにユーザーがそれらの変数を入力したり選択したりする必要がないようにすると便利です。

これらの変数は保存され、Qt と QGIS API の助けを借りて取得できます。各変数について、変数にアクセスするために使用されるキーを選択する必要があります---ユーザの好みの色のためにキー「favourite_color」またはその他の意味のある文字列を使用できます。キーの名前をつけるときは何らかの構造を持たせることをお勧めします。

We can differentiate between several types of settings:

- **global settings** --- they are bound to the user at a particular machine. QGIS itself stores a lot of global settings, for example, main window size or default snapping tolerance. Settings are handled using the `QgsSettings` class, through for example the `setValue()` and `value()` methods.

Here you can see an example of how these methods are used.

```

1 def store():
2     s = QgsSettings()
3     s.setValue("myplugin/mytext", "hello world")
4     s.setValue("myplugin/myint", 10)
5     s.setValue("myplugin/myreal", 3.14)
6
7 def read():
8     s = QgsSettings()
9     mytext = s.value("myplugin/mytext", "default text")
10    myint = s.value("myplugin/myint", 123)
11    myreal = s.value("myplugin/myreal", 2.71)
12    nonexistent = s.value("myplugin/nonexistent", None)
13    print(mytext)
14    print(myint)
15    print(myreal)
16    print(nonexistent)

```

The second parameter of the `value()` method is optional and specifies the default value that is returned if there is no previous value set for the passed setting name.

For a method to pre-configure the default values of the global settings through the `global_settings`.

ini file, see `deploying_organization` for further details.

- **project settings** --- vary between different projects and therefore they are connected with a project file. Map canvas background color or destination coordinate reference system (CRS) are examples --- white background and WGS84 might be suitable for one project, while yellow background and UTM projection are better for another one.

An example of usage follows.

```
1 proj = QgsProject.instance()
2
3 # store values
4 proj.writeEntry("myplugin", "mytext", "hello world")
5 proj.writeEntry("myplugin", "myint", 10)
6 proj.writeEntry("myplugin", "mydouble", 0.01)
7 proj.writeEntry("myplugin", "mybool", True)
8
9 # read values (returns a tuple with the value, and a status boolean
10 # which communicates whether the value retrieved could be converted to
11 # its type, in these cases a string, an integer, a double and a boolean
12 # respectively)
13
14 mytext, type_conversion_ok = proj.readEntry("myplugin",
15                                             "mytext",
16                                             "default text")
17 myint, type_conversion_ok = proj.readNumEntry("myplugin",
18                                               "myint",
19                                               123)
20 mydouble, type_conversion_ok = proj.readDoubleEntry("myplugin",
21                                                     "mydouble",
22                                                     123)
23 mybool, type_conversion_ok = proj.readBoolEntry("myplugin",
24                                                  "mybool",
25                                                  123)
```

As you can see, the `writeEntry()` method is used for all data types, but several methods exist for reading the setting value back, and the corresponding one has to be selected for each data type.

- **map layer settings** --- these settings are related to a particular instance of a map layer with a project. They are *not* connected with underlying data source of a layer, so if you create two map layer instances of one shapefile, they will not share the settings. The settings are stored inside the project file, so if the user opens the project again, the layer-related settings will be there again. The value for a given setting is retrieved using the `customProperty()` method, and can be set using the `setCustomProperty()` one.

```
1 vlayer = QgsVectorLayer()
2 # save a value
3 vlayer.setCustomProperty("mytext", "hello world")
4
5 # read the value again (returning "default text" if not found)
6 mytext = vlayer.customProperty("mytext", "default text")
```

The code snippets on this page need the following imports if you're outside the pyqgis console:

```
1 from qgis.core import (  
2     QgsMessageLog,  
3     QgsGeometry,  
4 )  
5  
6 from qgis.gui import (  
7     QgsMessageBar,  
8 )  
9  
10 from qgis.PyQt.QtWidgets import (  
11     QSizePolicy,  
12     QPushButton,  
13     QDialog,  
14     QGridLayout,  
15     QDialogButtonBox,  
16 )
```


第 13 章

ユーザーとのコミュニケーション

- *Showing messages. The QgsMessageBar class*
- プロセスを表示する
- ログを作成する
 - *QgsMessageLog*
 - *The python built in logging module*

このセクションでは、ユーザーインターフェイスにおいて一貫性を維持するためにユーザーとのコミュニケーション時に使うべき方法と要素をいくつか示します。

13.1 Showing messages. The QgsMessageBar class

メッセージボックスを使用するのはユーザー体験の見地からは良いアイデアではありません。警告/エラー用に小さな情報行を表示するには、たいてい QGIS メッセージバーが良い選択肢です。

QGIS インターフェイスオブジェクトへの参照を利用すると、次のようなコードでメッセージバー内にメッセージを表示できます。

```
from qgis.core import Qgs
iface.messageBar().pushMessage("Error", "I'm sorry Dave, I'm afraid I can't do that
↔", level=Qgs.Critical)
```

```
Messages (2): Error : I'm sorry Dave, I'm afraid I can't do that
```

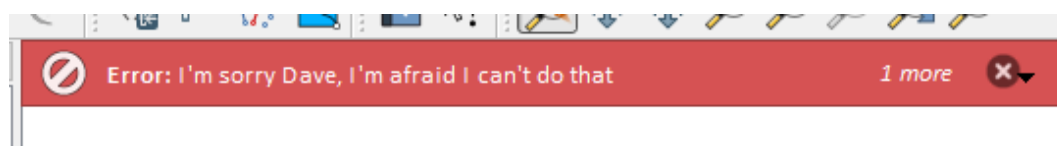


図 13.1 QGIS メッセージバー

表示期間を設定して時間を限定することができます。

```
iface.messageBar().pushMessage("Oops", "The plugin is not working as it should",
    level=Qgis.Critical, duration=3)
```

```
Messages(2): Oops : The plugin is not working as it should
```

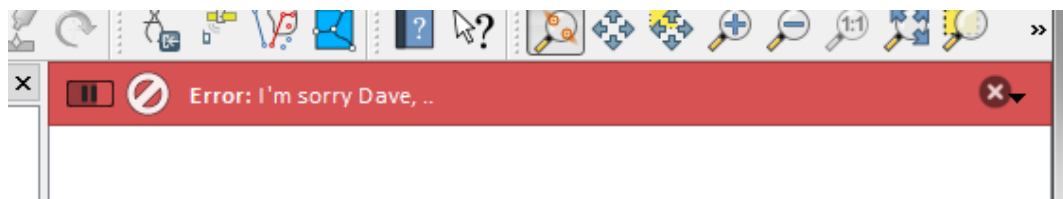


図 13.2 タイマー付き QGIS メッセージバー

The examples above show an error bar, but the `level` parameter can be used to creating warning messages or info messages, using the `Qgis.MessageLevel` enumeration. You can use up to 4 different levels:

0. Info
1. Warning
2. Critical
3. Success

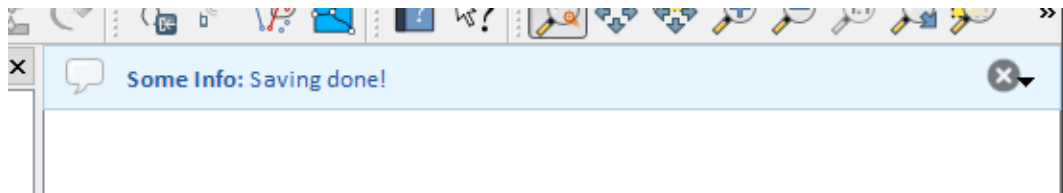


図 13.3 QGIS メッセージバー (お知らせ)

ウィジェットは、例えば詳細情報の表示用ボタンのように、メッセージバーに追加することができます

```
1 def showError():
2     pass
3
4 widget = iface.messageBar().createMessage("Missing Layers", "Show Me")
5 button = QPushButton(widget)
6 button.setText("Show Me")
7 button.pressed.connect(showError)
8 widget.layout().addWidget(button)
9 iface.messageBar().pushWidget(widget, Qgis.Warning)
```

```
Messages(1): Missing Layers : Show Me
```

メッセージバーは自分のダイアログの中でも使えるため、メッセージボックスを表示する必要はありませんし、メインの QGIS ウィンドウ内に表示する意味がない時にも使えます。

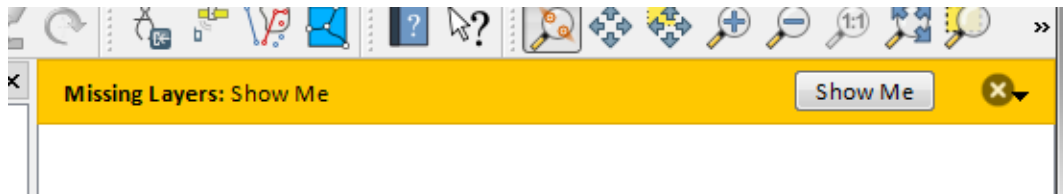


図 13.4 ボタン付きの QGIS メッセージバー

```

1 class MyDialog(QDialog):
2     def __init__(self):
3         QDialog.__init__(self)
4         self.bar = QgsMessageBar()
5         self.bar.setSizePolicy(QSizePolicy.Minimum, QSizePolicy.Fixed)
6         self.setLayout(QGridLayout())
7         self.layout().setContentsMargins(0, 0, 0, 0)
8         self.buttonbox = QDialogButtonBox(QDialogButtonBox.Ok)
9         self.buttonbox.accepted.connect(self.run)
10        self.layout().addWidget(self.buttonbox, 0, 0, 2, 1)
11        self.layout().addWidget(self.bar, 0, 0, 1, 1)
12    def run(self):
13        self.bar.pushMessage("Hello", "World", level=Qgis.Info)
14
15 myDlg = MyDialog()
16 myDlg.show()

```

13.2 プロセスを表示する

プログレスバーはご覧のとおりウィジェットを受け入れるので、QGIS メッセージバーに置くこともできます。コンソール内で試すことができる例はこちらです。

```

1 import time
2 from qgis.PyQt.QtWidgets import QProgressBar
3 from qgis.PyQt.QtCore import *
4 progressMessageBar = iface.messageBar().createMessage("Doing something boring...")
5 progress = QProgressBar()
6 progress.setMaximum(10)
7 progress.setAlignment(Qt.AlignLeft|Qt.AlignVCenter)
8 progressMessageBar.layout().addWidget(progress)
9 iface.messageBar().pushWidget(progressMessageBar, Qgis.Info)
10
11 for i in range(10):
12     time.sleep(1)
13     progress.setValue(i + 1)
14
15 iface.messageBar().clearWidgets()

```

```
Messages(0): Doing something boring...
```

Also, you can use the built-in status bar to report progress, as in the next example:



図 13.5 カスタムダイアログ内の QGIS メッセージバー

```
1 vlayer = iface.activeLayer()
2
3 count = vlayer.featureCount()
4 features = vlayer.getFeatures()
5
6 for i, feature in enumerate(features):
7     # do something time-consuming here
8     print('.') # printing should give enough time to present the progress
9
10    percent = i / float(count) * 100
11    # iface.mainWindow().statusBar().showMessage("Processed {} %".
12    ↪format(int(percent)))
13    iface.statusBarIface().showMessage("Processed {} %".format(int(percent)))
14
15 iface.statusBarIface().clearMessage()
```

13.3 ログを作成する

There are three different types of logging available in QGIS to log and save all the information about the execution of your code. Each has its specific output location. Please consider to use the correct way of logging for your purpose:

- `QgsMessageLog` is for messages to communicate issues to the user. The output of the `QgsMessageLog` is shown in the Log Messages Panel.
- The python built in **logging** module is for debugging on the level of the QGIS Python API (PyQGIS). It is recommended for Python script developers that need to debug their python code, e.g. feature ids or geometries
- `QgsLogger` is for messages for *QGIS internal* debugging / developers (i.e. you suspect something is triggered by some broken code). Messages are only visible with developer versions of QGIS.

Examples for the different logging types are shown in the following sections below.

警告: Use of the Python `print` statement is unsafe to do in any code which may be multithreaded and **extremely slows down the algorithm**. This includes **expression functions, renderers, symbol layers** and **Processing algorithms** (amongst others). In these cases you should always use the python **logging** module or thread safe classes (`QgsLogger` or `QgsMessageLog`) instead.

13.3.1 QgsMessageLog

```
# You can optionally pass a 'tag' and a 'level' parameters
QgsMessageLog.logMessage("Your plugin code has been executed correctly", 'MyPlugin
↪', level=Qgis.Info)
QgsMessageLog.logMessage("Your plugin code might have some problems", level=Qgis.
↪Warning)
QgsMessageLog.logMessage("Your plugin code has crashed!", level=Qgis.Critical)
```

```
MyPlugin(0): Your plugin code has been executed correctly
(1): Your plugin code might have some problems
(2): Your plugin code has crashed!
```

注釈: You can see the output of the `QgsMessageLog` in the `log_message_panel`

13.3.2 The python built in logging module

```

1 import logging
2 formatter = '%(asctime)s - %(name)s - %(levelname)s - %(message)s'
3 logfilename=r'c:\temp\example.log'
4 logging.basicConfig(filename=logfilename, level=logging.DEBUG, format=formatter)
5 logging.info("This logging info text goes into the file")
6 logging.debug("This logging debug text goes into the file as well")

```

The basicConfig method configures the basic setup of the logging. In the above code the filename, logging level and the format are defined. The filename refers to where to write the logfile to, the logging level defines what levels to output and the format defines the format in which each message is output.

```

2020-10-08 13:14:42,998 - root - INFO - This logging text goes into the file
2020-10-08 13:14:42,998 - root - DEBUG - This logging debug text goes into the_
↪file as well

```

If you want to erase the log file every time you execute your script you can do something like:

```

if os.path.isfile(logfilename):
    with open(logfilename, 'w') as file:
        pass

```

Further resources on how to use the python logging facility are available at:

- <https://docs.python.org/3/library/logging.html>
- <https://docs.python.org/3/howto/logging.html>
- <https://docs.python.org/3/howto/logging-cookbook.html>

警告: Please note that without logging to a file by setting a filename the logging may be multithreaded which heavily slows down the output.

The code snippets on this page need the following imports if you're outside the pyqgis console:

```

1 from qgis.core import (
2     QgsApplication,
3     QgsRasterLayer,
4     QgsAuthMethodConfig,
5     QgsDataSourceUri,
6     QgsPkiBundle,
7     QgsMessageLog,
8 )
9
10 from qgis.gui import (
11     QgsAuthAuthoritiesEditor,
12     QgsAuthConfigEditor,

```

(次のページに続く)

(前のページからの続き)

```
13     QgsAuthConfigSelect,  
14     QgsAuthSettingsWidget,  
15 )  
16  
17 from qgis.PyQt.QtWidgets import (  
18     QWidget,  
19     QTabWidget,  
20 )  
21  
22 from qgis.PyQt.QtNetwork import QSslCertificate
```


第 14 章

認証インフラストラクチャ

- 前書き
- 用語集
- エントリポイント *QgsAuthManager*
 - マネージャを初期化し、マスターパスワードを設定する
 - 認証データベースに新しい認証構成項目を設定する
 - * 利用可能な認証方法
 - * 認証局の導入
 - * *QgsPkiBundle* で *PKI* バンドルを管理する
 - *Remove an entry from authdb*
 - *QgsAuthManager* に *authcfg* 展開を残す
 - * 他のデータプロバイダーと *PKI* の例
- 認証インフラストラクチャを使用するようにプラグインを適応させる
- 認証の *GUI*
 - 資格情報を選択するための *GUI*
 - 認証エディタの *GUI*
 - 認証局エディタの *GUI*

14.1 前書き

認証基盤のユーザーリファレンスはユーザーマニュアル中で `authentication_overview` 段落を参照して下さい。

この章では、開発者の観点から、認証システムを使用するベストプラクティスについて説明します。

The authentication system is widely used in QGIS Desktop by data providers whenever credentials are required to access a particular resource, for example when a layer establishes a connection to a Postgres database.

There are also a few widgets in the QGIS gui library that plugin developers can use to easily integrate the authentication infrastructure into their code:

- `QgsAuthConfigEditor`
- `QgsAuthConfigSelect`
- `QgsAuthSettingsWidget`

A good code reference can be read from the authentication infrastructure `tests code`.

警告: Due to the security constraints that were taken into account during the authentication infrastructure design, only a selected subset of the internal methods are exposed to Python.

14.2 用語集

これはこの章で扱われる最も一般的なオブジェクトのいくつかの定義です。

マスターパスワード アクセスを許可し、QGIS 認証 DB に保存された資格情報を復号化するパスワードです

認証データベース A *Master Password* crypted sqlite db `qgis-auth.db` where *Authentication Configuration* are stored. e.g user/password, personal certificates and keys, Certificate Authorities

認証 DB 認証データベース

認証の設定 認証データの構成は認証メソッドによって異なります。例えばベーシック認証の場合は、ユーザー/パスワードの対が格納されます。

Authentication Config 認証設定

認証方法 認証されるためには特別な方法が利用されています。各方法は、認証されるためにそれぞれ独自のプロトコルを利用しています。それぞれの方法は共有ライブラリとして QGIS 認証基盤の初期化中に動的にロードされるように実装されています。

14.3 エントリーポイント QgsAuthManager

The `QgsAuthManager` singleton is the entry point to use the credentials stored in the QGIS encrypted *Authentication DB*, i.e. the `qgis-auth.db` file under the active user profile folder.

This class takes care of the user interaction: by asking to set a master password or by transparently using it to access encrypted stored information.

14.3.1 マネージャを初期化し、マスターパスワードを設定する

次のコード例は、認証設定へのアクセスを開くために、マスターパスワードを設定する例を示します。コードのコメントは、このコード例を理解するために重要です。

```

1 authMgr = QgsApplication.authManager()
2
3 # check if QgsAuthManager has already been initialized... a side effect
4 # of the QgsAuthManager.init() is that AuthDbPath is set.
5 # QgsAuthManager.init() is executed during QGIS application init and hence
6 # you do not normally need to call it directly.
7 if authMgr.authenticationDatabasePath():
8     # already initilised => we are inside a QGIS app.
9     if authMgr.masterPasswordIsSet():
10        msg = 'Authentication master password not recognized'
11        assert authMgr.masterPasswordSame("your master password"), msg
12    else:
13        msg = 'Master password could not be set'
14        # The verify parameter check if the hash of the password was
15        # already saved in the authentication db
16        assert authMgr.setMasterPassword("your master password",
17                                         verify=True), msg
18 else:
19     # outside qgis, e.g. in a testing environment => setup env var before
20     # db init
21     os.environ['QGIS_AUTH_DB_DIR_PATH'] = "/path/where/located/qgis-auth.db"
22     msg = 'Master password could not be set'
23     assert authMgr.setMasterPassword("your master password", True), msg
24     authMgr.init("/path/where/located/qgis-auth.db")

```

14.3.2 認証データベースに新しい認証構成項目を設定する

Any stored credential is a *Authentication Configuration* instance of the `QgsAuthMethodConfig` class accessed using a unique string like the following one:

```
authcfg = 'fm1s770'
```

that string is generated automatically when creating an entry using the QGIS API or GUI, but it might be useful to manually set it to a known value in case the configuration must be shared (with different credentials) between multiple users within an organization.

`QgsAuthMethodConfig` is the base class for any *Authentication Method*. Any Authentication Method sets a configuration hash map where authentication informations will be stored. Hereafter an useful snippet to store PKI-path credentials for an hypothetical alice user:

```
1 authMgr = QgsApplication.authManager()
2 # set alice PKI data
3 config = QgsAuthMethodConfig()
4 config.setName("alice")
5 config.setMethod("PKI-Paths")
6 config.setUri("https://example.com")
7 config.setConfig("certpath", "path/to/alice-cert.pem" )
8 config.setConfig("keypath", "path/to/alice-key.pem" )
9 # check if method parameters are correctly set
10 assert config.isValid()
11
12 # register alice data in authdb returning the ``authcfg`` of the stored
13 # configuration
14 authMgr.storeAuthenticationConfig(config)
15 newAuthCfgId = config.id()
16 assert newAuthCfgId
```

利用可能な認証方法

Authentication Method libraries are loaded dynamically during authentication manager init. Available authentication methods are:

1. Basic ユーザーとパスワード認証
2. Esri-Token ESRI token based authentication
3. Identity-Cert アイデンティティ証明書認証
4. OAuth2 OAuth2 authentication
5. PKI-Paths PKI パス認証
6. PKI-PKCS # 12 PKI PKCS # 12 認証

認証局の導入

```
1 authMgr = QgsApplication.authManager()
2 # add authorities
3 cacerts = QSslCertificate.fromPath( "/path/to/ca_chains.pem" )
4 assert cacerts is not None
5 # store CA
6 authMgr.storeCertAuthorities(cacerts)
7 # and rebuild CA caches
8 authMgr.rebuildCaCertsCache()
9 authMgr.rebuildTrustedCaCertsCache()
```

QgsPkiBundle で PKI バンドルを管理する

A convenience class to pack PKI bundles composed on SslCert, SslKey and CA chain is the `QgsPkiBundle` class. Hereafter a snippet to get password protected:

```

1 # add alice cert in case of key with pwd
2 caBundlesList = [] # List of CA bundles
3 bundle = QgsPkiBundle.fromPemPaths( "/path/to/alice-cert.pem",
4                                     "/path/to/alice-key_w-pass.pem",
5                                     "unlock_pwd",
6                                     caBundlesList )
7 assert bundle is not None
8 # You can check bundle validity by calling:
9 # bundle.isValid()
```

Refer to `QgsPkiBundle` class documentation to extract cert/key/CAs from the bundle.

14.3.3 Remove an entry from authdb

以下が 認証データベース からエントリをその `authcfg` 識別子を使用して削除するコード例です :

```

authMgr = QgsApplication.authManager()
authMgr.removeAuthenticationConfig( "authCfg_Id_to_remove" )
```

14.3.4 QgsAuthManager に authcfg 展開を残す

The best way to use an *Authentication Config* stored in the *Authentication DB* is referring it with the unique identifier `authcfg`. Expanding, means convert it from an identifier to a complete set of credentials. The best practice to use stored *Authentication Configs*, is to leave it managed automatically by the Authentication manager. The common use of a stored configuration is to connect to an authentication enabled service like a WMS or WFS or to a DB connection.

注釈: Take into account that not all QGIS data providers are integrated with the Authentication infrastructure. Each authentication method, derived from the base class `QgsAuthMethod` and support a different set of Providers. For example the `certIdentity` () method supports the following list of providers:

```

authM = QgsApplication.authManager()
print(authM.authMethod("Identity-Cert").supportedDataProviders())
```

Sample output:

```
['ows', 'wfs', 'wcs', 'wms', 'postgres']
```

例えば、`authcfg = 'fmls770'` で識別保存された資格情報を使用して WMS サービスにアクセスするためには、次のコードのように、データ・ソースの URL に `authcfg` を使用する必要があります。

```
1 authCfg = 'fmls770'
2 quri = QgsDataSourceUri()
3 quri.setParam("layers", 'usa:states')
4 quri.setParam("styles", '')
5 quri.setParam("format", 'image/png')
6 quri.setParam("crs", 'EPSG:4326')
7 quri.setParam("dpiMode", '7')
8 quri.setParam("featureCount", '10')
9 quri.setParam("authcfg", authCfg) # <---- here my authCfg url parameter
10 quri.setParam("contextualWMSLegend", '0')
11 quri.setParam("url", 'https://my_auth_enabled_server_ip/wms')
12 rlayer = QgsRasterLayer(str(quri.encodedUri(), "utf-8"), 'states', 'wms')
```

上の場合には、wms プロバイダーは、単に HTTP 接続を設定する前に、資格を authcfg URI パラメーターを拡張するために世話をします。

警告: The developer would have to leave authcfg expansion to the `QgsAuthManager`, in this way he will be sure that expansion is not done too early.

Usually an URI string, built using the `QgsDataSourceURI` class, is used to set a data source in the following way:

```
authCfg = 'fmls770'
quri = QgsDataSourceUri("my WMS uri here")
quri.setParam("authcfg", authCfg)
rlayer = QgsRasterLayer(quri.uri(False), 'states', 'wms')
```

注釈: False パラメーターは、URI で authcfg ID の存在の URI 完全な展開を避けるために重要です。

他のデータプロバイダーと PKI の例

Other example can be read directly in the QGIS tests upstream as in `test_authmanager_pki_ows` or `test_authmanager_pki_postgres`.

14.4 認証インフラストラクチャを使用するようにプラグインを適応させる

Many third party plugins are using `httplib2` or other Python networking libraries to manage HTTP connections instead of integrating with `QgsNetworkAccessManager` and its related Authentication Infrastructure integration.

To facilitate this integration a helper Python function has been created called `NetworkAccessManager`. Its code can be found [here](#).

このヘルパークラスは、次のコードのように使用できます：

```

1 http = NetworkAccessManager(authid="my_authCfg", exception_class=My_
  ↳FailedRequestError)
2 try:
3     response, content = http.request( "my_rest_url" )
4 except My_FailedRequestError, e:
5     # Handle exception
6     pass

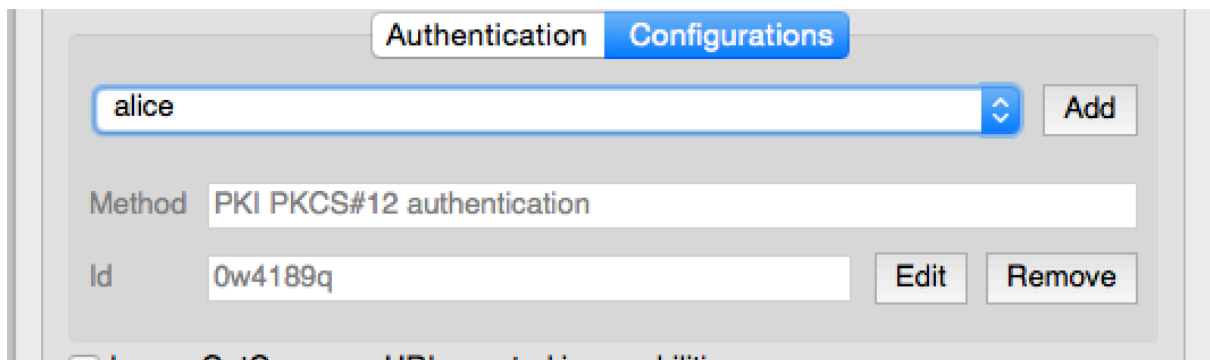
```

14.5 認証の GUI

この段落では、カスタムインターフェイスで認証インフラストラクチャを統合するために役立つ利用可能な GUI が記載されています。

14.5.1 資格情報を選択するための GUI

If it's necessary to select a *Authentication Configuration* from the set stored in the *Authentication DB* it is available in the GUI class `QgsAuthConfigSelect`.



そして次のコードのように使用できます：

```

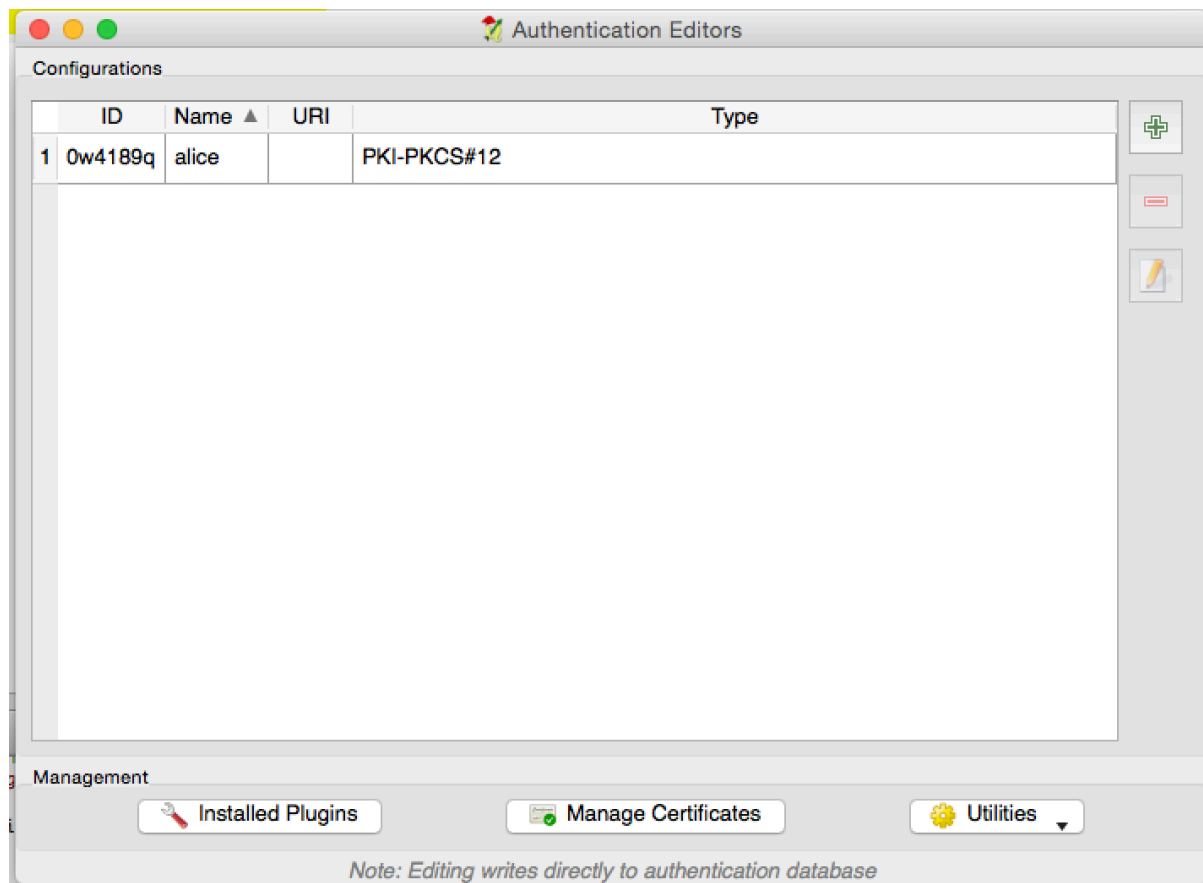
1 # create the instance of the QgsAuthConfigSelect GUI hierarchically linked to
2 # the widget referred with `parent`
3 parent = QWidget() # Your GUI parent widget
4 gui = QgsAuthConfigSelect( parent, "postgres" )
5 # add the above created gui in a new tab of the interface where the
6 # GUI has to be integrated
7 tabGui = QTabWidget()
8 tabGui.insertTab( 1, gui, "Configurations" )

```

The above example is taken from the QGIS source code. The second parameter of the GUI constructor refers to data provider type. The parameter is used to restrict the compatible *Authentication Methods* with the specified provider.

14.5.2 認証エディタの GUI

The complete GUI used to manage credentials, authorities and to access to Authentication utilities is managed by the `QgsAuthEditorWidgets` class.



そして次のコードのように使用できます：

```

1 # create the instance of the QgsAuthEditorWidgets GUI hierarchically linked to
2 # the widget referred with `parent`
3 parent = QWidget() # Your GUI parent widget
4 gui = QgsAuthConfigSelect( parent )
5 gui.show()

```

An integrated example can be found in the related test.

14.5.3 認証局エディタの GUI

A GUI used to manage only authorities is managed by the `QgsAuthAuthoritiesEditor` class.

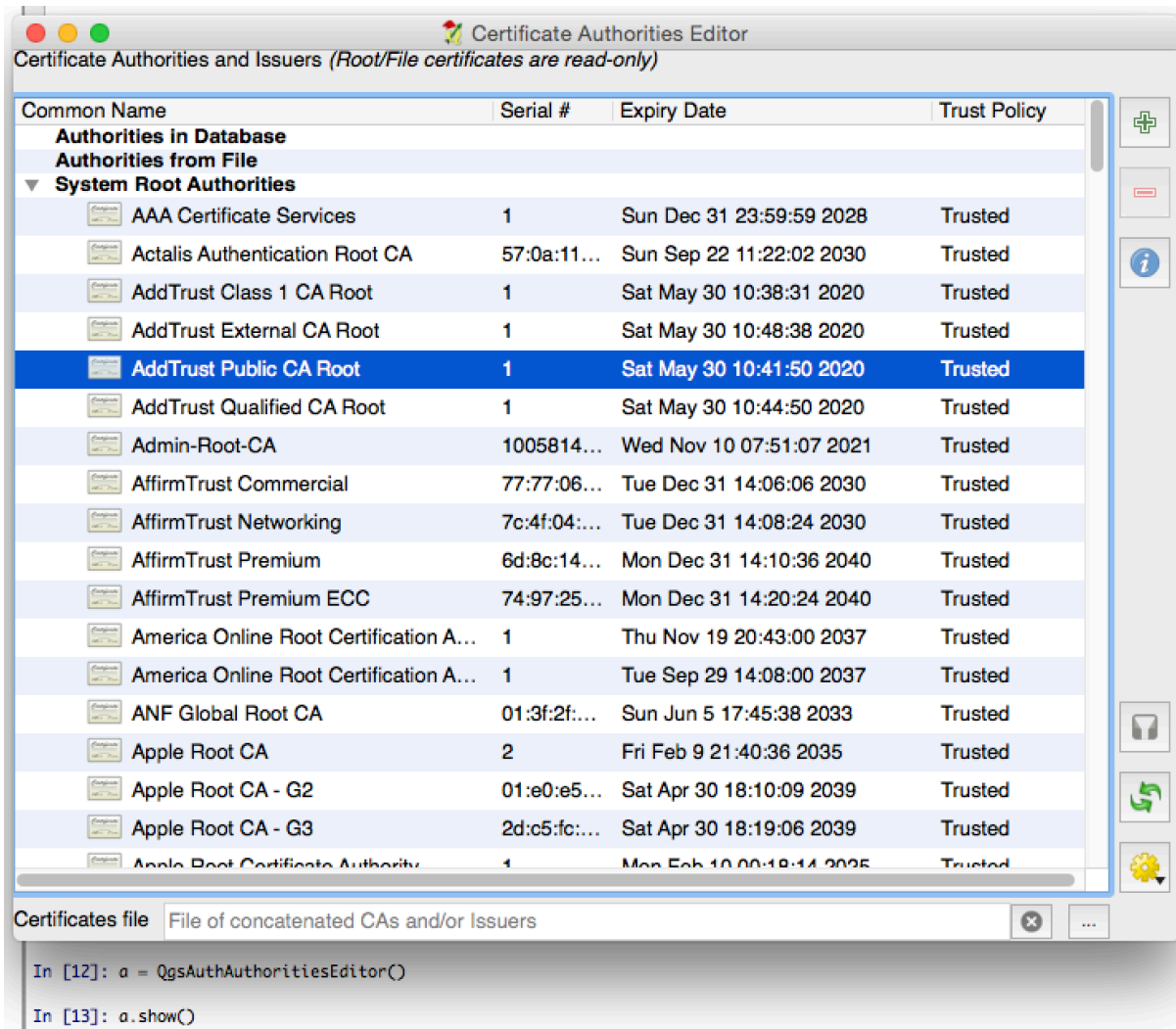
そして次のコードのように使用できます：

```

1 # create the instance of the QgsAuthAuthoritiesEditor GUI hierarchically
2 # linked to the widget referred with `parent`
3 parent = QWidget() # Your GUI parent widget

```

(次のページに続く)



(前のページからの続き)

```
4 gui = QgsAuthAuthoritiesEditor( parent )
5 gui.show()
```

The code snippets on this page need the following imports if you're outside the pyqgis console:

```
1 from qgis.core import (
2     QgsProcessingContext,
3     QgsTaskManager,
4     QgsTask,
5     QgsProcessingAlgRunnerTask,
6     Qgis,
7     QgsProcessingFeedback,
8     QgsApplication,
9     QgsMessageLog,
10 )
```


第 15 章

タスク - バックグラウンドで重い仕事をする

15.1 はじめに

スレッドを使用したバックグラウンド処理は、重い処理が行われているときに応答性の高いユーザーインターフェイスを維持するための方法です。タスクは QGIS でスレッドを実行するために使用できます。

タスク (`QgsTask`) はバックグラウンドで実行されるコードのコンテナです。そしてタスクマネージャ (`QgsTaskManager`) はタスクの実行を制御するために使用されます。これらのクラスは、シグナリング、進捗報告、およびバックグラウンドプロセスのステータスへアクセスするためのメカニズムを提供することによって、QGIS のバックグラウンド処理を単純化します。タスクはサブタスクを使用してグループ化できます。

The global task manager (found with `QgsApplication.taskManager()`) is normally used. This means that your tasks may not be the only tasks that are controlled by the task manager.

QGIS タスクを作成する方法はいくつかあります：

- `QgsTask` を拡張することで自分のタスクを作成する

```
class SpecialisedTask(QgsTask):
    pass
```

- 関数からタスクを作成

```
1 def heavyFunction():
2     # Some CPU intensive processing ...
3     pass
4
5 def workdone():
6     # ... do something useful with the results
7     pass
8
9 task = QgsTask.fromFunction('heavy function', heavyFunction,
10                             onfinished=workdone)
```

- プロセッシングアルゴリズムからタスクを作成

```

1 params = dict()
2 context = QgsProcessingContext()
3 feedback = QgsProcessingFeedback()
4
5 buffer_alg = QgsApplication.instance().processingRegistry().algorithmById(
6     ↪ 'native:buffer')
7 task = QgsProcessingAlgRunnerTask(buffer_alg, params, context,
                                   feedback)

```

警告: Any background task (regardless of how it is created) must NEVER use any QObject that lives on the main thread, such as accessing QgsVectorLayer, QgsProject or perform any GUI based operations like creating new widgets or interacting with existing widgets. Qt widgets must only be accessed or modified from the main thread. Data that is used in a task must be copied before the task is started. Attempting to use them from background threads will result in crashes.

Dependencies between tasks can be described using the `addSubTask` function of `QgsTask`. When a dependency is stated, the task manager will automatically determine how these dependencies will be executed. Whenever possible dependencies will be executed in parallel in order to satisfy them as quickly as possible. If a task on which another task depends is canceled, the dependent task will also be canceled. Circular dependencies can make deadlocks possible, so be careful.

If a task depends on a layer being available, this can be stated using the `setDependentLayers` function of `QgsTask`. If a layer on which a task depends is not available, the task will be canceled.

Once the task has been created it can be scheduled for running using the `addTask` function of the task manager. Adding a task to the manager automatically transfers ownership of that task to the manager, and the manager will cleanup and delete tasks after they have executed. The scheduling of the tasks is influenced by the task priority, which is set in `addTask`.

タスクの状態は `QgsTask` および `QgsTaskManager` のシグナルと関数を使って監視できます。

15.2 例

15.2.1 QgsTask を拡張する

この例では `RandomIntegerSumTask` は `QgsTask` を拡張し、指定された期間中に 0 から 500 の間の 100 個のランダムな整数を生成します。乱数が 42 の場合、タスクは中止され、例外が発生します。(サブタスク付きの) `RandomIntegerSumTask` のいくつかのインスタンスが生成されてタスクマネージャに追加され、2 種類の依存関係を実証します。

```

1 import random
2 from time import sleep

```

(次のページに続く)

(前のページからの続き)

```

3
4 from qgis.core import (
5     QgsApplication, QgsTask, QgsMessageLog,
6     )
7
8 MESSAGE_CATEGORY = 'RandomIntegerSumTask'
9
10 class RandomIntegerSumTask(QgsTask):
11     """This shows how to subclass QgsTask"""
12
13     def __init__(self, description, duration):
14         super().__init__(description, QgsTask.CanCancel)
15         self.duration = duration
16         self.total = 0
17         self.iterations = 0
18         self.exception = None
19
20     def run(self):
21         """Here you implement your heavy lifting.
22         Should periodically test for isCanceled() to gracefully
23         abort.
24         This method MUST return True or False.
25         Raising exceptions will crash QGIS, so we handle them
26         internally and raise them in self.finished
27         """
28         QgsMessageLog.logMessage('Started task "{}".format(
29             self.description()),
30             MESSAGE_CATEGORY, Qgs.Info)
31         wait_time = self.duration / 100
32         for i in range(100):
33             sleep(wait_time)
34             # use setProgress to report progress
35             self.setProgress(i)
36             arandominteger = random.randint(0, 500)
37             self.total += arandominteger
38             self.iterations += 1
39             # check isCanceled() to handle cancellation
40             if self.isCanceled():
41                 return False
42             # simulate exceptions to show how to abort task
43             if arandominteger == 42:
44                 # DO NOT raise Exception('bad value!')
45                 # this would crash QGIS
46                 self.exception = Exception('bad value!')
47                 return False
48             return True
49
50     def finished(self, result):
51         """
52         This function is automatically called when the task has
53         completed (successfully or not).

```

(次のページに続く)

```

54     You implement finished() to do whatever follow-up stuff
55     should happen after the task is complete.
56     finished is always called from the main thread, so it's safe
57     to do GUI operations and raise Python exceptions here.
58     result is the return value from self.run.
59     """
60     if result:
61         QgsMessageLog.logMessage(
62             'RandomTask "{name}" completed\n' \
63             'RandomTotal: {total} (with {iterations} '\
64             'iterations)'.format(
65                 name=self.description(),
66                 total=self.total,
67                 iterations=self.iterations),
68             MESSAGE_CATEGORY, Qgs.Success)
69     else:
70         if self.exception is None:
71             QgsMessageLog.logMessage(
72                 'RandomTask "{name}" not successful but without '\
73                 'exception (probably the task was manually '\
74                 'canceled by the user)'.format(
75                     name=self.description()),
76                 MESSAGE_CATEGORY, Qgs.Warning)
77         else:
78             QgsMessageLog.logMessage(
79                 'RandomTask "{name}" Exception: {exception}'.format(
80                     name=self.description(),
81                     exception=self.exception),
82                 MESSAGE_CATEGORY, Qgs.Critical)
83         raise self.exception
84
85     def cancel(self):
86         QgsMessageLog.logMessage(
87             'RandomTask "{name}" was canceled'.format(
88                 name=self.description()),
89                 MESSAGE_CATEGORY, Qgs.Info)
90         super().cancel()
91
92
93     longtask = RandomIntegerSumTask('waste cpu long', 20)
94     shorttask = RandomIntegerSumTask('waste cpu short', 10)
95     minitask = RandomIntegerSumTask('waste cpu mini', 5)
96     shortsubtask = RandomIntegerSumTask('waste cpu subtask short', 5)
97     longsubtask = RandomIntegerSumTask('waste cpu subtask long', 10)
98     shortestsubtask = RandomIntegerSumTask('waste cpu subtask shortest', 4)
99
100     # Add a subtask (shortsubtask) to shorttask that must run after
101     # minitask and longtask has finished
102     shorttask.addSubTask(shortsubtask, [minitask, longtask])
103     # Add a subtask (longsubtask) to longtask that must be run
104     # before the parent task

```

(前のページからの続き)

```

105 longtask.addSubTask(longsubtask, [], QgsTask.ParentDependsOnSubTask)
106 # Add a subtask (shortestsubtask) to longtask
107 longtask.addSubTask(shortestsubtask)
108
109 QgsApplication.taskManager().addTask(longtask)
110 QgsApplication.taskManager().addTask(shorttask)
111 QgsApplication.taskManager().addTask(minitask)

```

```

1 RandomIntegerSumTask(0): Started task "waste cpu subtask shortest"
2 RandomIntegerSumTask(0): Started task "waste cpu short"
3 RandomIntegerSumTask(0): Started task "waste cpu mini"
4 RandomIntegerSumTask(0): Started task "waste cpu subtask long"
5 RandomIntegerSumTask(3): Task "waste cpu subtask shortest" completed
6 RandomTotal: 25452 (with 100 iterations)
7 RandomIntegerSumTask(3): Task "waste cpu mini" completed
8 RandomTotal: 23810 (with 100 iterations)
9 RandomIntegerSumTask(3): Task "waste cpu subtask long" completed
10 RandomTotal: 26308 (with 100 iterations)
11 RandomIntegerSumTask(0): Started task "waste cpu long"
12 RandomIntegerSumTask(3): Task "waste cpu long" completed
13 RandomTotal: 22534 (with 100 iterations)

```

15.2.2 関数からのタスク

関数からタスクを作成します(この例では `doSomething`)。関数の最初のパラメータは関数の `QgsTask` を持ちます。重要な(名前付き)パラメータは `on_finished` です。これはタスクが完了したときに呼ばれる関数を指定します。この例の `doSomething` 関数は追加の名前付きパラメータ `wait_time` を持っています。

```

1 import random
2 from time import sleep
3
4 MESSAGE_CATEGORY = 'TaskFromFunction'
5
6 def doSomething(task, wait_time):
7     """
8     Raises an exception to abort the task.
9     Returns a result if success.
10    The result will be passed, together with the exception (None in
11    the case of success), to the on_finished method.
12    If there is an exception, there will be no result.
13    """
14    QgsMessageLog.logMessage('Started task {}'.format(task.description()),
15                             MESSAGE_CATEGORY, QgsInfo)
16    wait_time = wait_time / 100
17    total = 0
18    iterations = 0
19    for i in range(100):

```

(次のページに続く)

```
20     sleep(wait_time)
21     # use task.setProgress to report progress
22     task.setProgress(i)
23     arandominteger = random.randint(0, 500)
24     total += arandominteger
25     iterations += 1
26     # check task.isCanceled() to handle cancellation
27     if task.isCanceled():
28         stopped(task)
29         return None
30     # raise an exception to abort the task
31     if arandominteger == 42:
32         raise Exception('bad value!')
33     return {'total': total, 'iterations': iterations,
34           'task': task.description()}
35
36 def stopped(task):
37     QgsMessageLog.logMessage(
38         'Task "{name}" was canceled'.format(
39             name=task.description()),
40         MESSAGE_CATEGORY, Qgis.Info)
41
42 def completed(exception, result=None):
43     """This is called when doSomething is finished.
44     Exception is not None if doSomething raises an exception.
45     result is the return value of doSomething."""
46     if exception is None:
47         if result is None:
48             QgsMessageLog.logMessage(
49                 'Completed with no exception and no result '\
50                 '(probably manually canceled by the user)',
51                 MESSAGE_CATEGORY, Qgis.Warning)
52         else:
53             QgsMessageLog.logMessage(
54                 'Task {name} completed\n'
55                 'Total: {total} ( with {iterations} '\
56                 'iterations)'.format(
57                     name=result['task'],
58                     total=result['total'],
59                     iterations=result['iterations']),
60                 MESSAGE_CATEGORY, Qgis.Info)
61     else:
62         QgsMessageLog.logMessage("Exception: {}".format(exception),
63                                 MESSAGE_CATEGORY, Qgis.Critical)
64     raise exception
65
66 # Create a few tasks
67 task1 = QgsTask.fromFunction('Waste cpu 1', doSomething,
68                             on_finished=completed, wait_time=4)
69 task2 = QgsTask.fromFunction('Waste cpu 2', doSomething,
70                             on_finished=completed, wait_time=3)
```

(前のページからの続き)

```

71 QgsApplication.taskManager().addTask(task1)
72 QgsApplication.taskManager().addTask(task2)

```

```

1 RandomIntegerSumTask(0): Started task "waste cpu subtask short"
2 RandomTaskFromFunction(0): Started task Waste cpu 1
3 RandomTaskFromFunction(0): Started task Waste cpu 2
4 RandomTaskFromFunction(0): Task Waste cpu 2 completed
5 RandomTotal: 23263 ( with 100 iterations)
6 RandomTaskFromFunction(0): Task Waste cpu 1 completed
7 RandomTotal: 25044 ( with 100 iterations)

```

15.2.3 プロセッシングアルゴリズムからのタスク

Create a task that uses the algorithm `qgis:randompointsinextent` to generate 50000 random points inside a specified extent. The result is added to the project in a safe way.

```

1 from functools import partial
2 from qgis.core import (QgsTaskManager, QgsMessageLog,
3                       QgsProcessingAlgRunnerTask, QgsApplication,
4                       QgsProcessingContext, QgsProcessingFeedback,
5                       QgsProject)
6
7 MESSAGE_CATEGORY = 'AlgRunnerTask'
8
9 def task_finished(context, successful, results):
10     if not successful:
11         QgsMessageLog.logMessage('Task finished unsuccessfully',
12                                 MESSAGE_CATEGORY, Qgs.Warning)
13     output_layer = context.getMapLayer(results['OUTPUT'])
14     # because getMapLayer doesn't transfer ownership, the layer will
15     # be deleted when context goes out of scope and you'll get a
16     # crash.
17     # takeMapLayer transfers ownership so it's then safe to add it
18     # to the project and give the project ownership.
19     if output_layer and output_layer.isValid():
20         QgsProject.instance().addMapLayer(
21             context.takeResultLayer(output_layer.id()))
22
23 alg = QgsApplication.processingRegistry().algorithmById(
24         'qgis:randompointsinextent')
25 context = QgsProcessingContext()
26 feedback = QgsProcessingFeedback()
27 params = {
28     'EXTENT': '0.0,10.0,40,50 [EPSG:4326]',
29     'MIN_DISTANCE': 0.0,
30     'POINTS_NUMBER': 50000,
31     'TARGET_CRS': 'EPSG:4326',
32     'OUTPUT': 'memory:My random points'
33 }

```

(次のページに続く)

(前のページからの続き)

```
34 task = QgsProcessingAlgRunnerTask(alg, params, context, feedback)
35 task.executed.connect(partial(task_finished, context))
36 QgsApplication.taskManager().addTask(task)
```

See also: <https://opengis.ch/2018/06/22/threads-in-pyqgis3/>.

第 16 章

Python プラグインを開発する

16.1 Python プラグインを構成する

- プラグインを書く
 - プラグインファイル
- プラグインの内容
 - プラグインメタデータ
 - `__init__.py`
 - `mainPlugin.py`
 - Resource ファイル
- *Documentation*
- *Translation*
 - *Software requirements*
 - *Files and directory*
 - * *.pro file*
 - * *.ts file*
 - * *.qm file*
 - *Translate using Makefile*
 - *Load the plugin*
- *Tips and Tricks*
 - *Plugin Reloader*

- *Accessing Plugins*
- *Log Messages*
- *Share your plugin*

プラグインを作成するには、以下の手順に従います。

1. アイデア : 新しい QGIS プラグインで何をしたいか考えます。なぜそれを行うのですか？ あなたが解決したいのはどのような問題ですか？ その問題に対してすでに他のプラグインはないですか？
2. ファイルの作成 : ファイルのうちいくつかは必須です ([プラグインファイル](#) を参照してください)。
3. コードを書く : それぞれのファイルに適切なコードを書きます。
4. テスト : すべてがうまく行くかどうかを [プラグイン](#) をリロードして確認します。
5. 公開 : 出来上がったプラグインを QGIS リポジトリに公開するか、ご自身のリポジトリを作って個人的な「GIS ウェポン」の兵器廠にしましょう。

16.1.1 プラグインを書く

QGIS に Python プラグインが導入されて以来、数多くのプラグインが登場しました。QGIS チームは *Official Python plugin repository* を保守しています。それらのソースを使用して、PyQGIS を使ったプログラミングについてもっと学んだり、開発努力が重複していないか調べたりできます。

プラグインファイル

ここで例として見ていくプラグインのディレクトリ構造はこのようになります。

```
PYTHON_PLUGINS_PATH/  
  MyPlugin/  
    __init__.py    --> *required*  
    mainPlugin.py --> *core code*  
    metadata.txt   --> *required*  
    resources.qrc  --> *likely useful*  
    resources.py   --> *compiled version, likely useful*  
    form.ui        --> *likely useful*  
    form.py        --> *compiled version, likely useful*
```

各ファイルの意味は以下の通りです。

- `__init__.py` = プラグインの起点です。 `classFactory()` メソッドが必須ですが、それ以外にもどんな初期化コードでも含むことができます。
- `mainPlugin.py` = プラグインの中心として作動するコードです。プラグインの全ての機能に関する情報と主要コードを含みます。
- `resources.qrc` = Qt Designer によって作られる `.xml` ドキュメントです。外観のリソースへの相対パスを含みます。

- `resources.py` = 上記の `.qrc` ファイルを Python へと変換したものです。
- `form.ui` = Qt Designer によって作られた GUI です。
- `form.py` = 上記の `form.ui` を Python に変換したものです。
- `metadata.txt` = プラグインのウェブサイトとインフラストラクチャで使われる、一般的な情報、バージョン、名前、その他のメタデータを含みます。

こちら `_` に典型的な QGIS Python プラグインの基本的なファイル (スケルトン) を作る方法があります。

Plugin Builder 3 `_` という名前の、QGIS のプラグインテンプレートを作る QGIS プラグインもあります。3.x 互換のソースを作れますので、こちらが推奨される選択肢となります。

警告: プラグインを *Official Python plugin repository* にアップロードする予定なら、プラグインの *Validation* で必要とされる幾つかの付加的なルールに、そのプラグインが従っていることを確認せねばなりません。

16.1.2 プラグインの内容

ここでは、上記のファイル構造中のそれぞれのファイルに何を書けばよいか、その情報と実例を提供します。

プラグインメタデータ

最初に、そのプラグインの名前や説明といった基本的な情報を、プラグインマネージャは検索する必要があります。 `metadata.txt` ファイルがその情報を格納しておく場所になります。

注釈: メタデータのエンコーディングはすべて UTF-8 でなければなりません。

| メタデータ名 | 必須 | 注 |
|-----------------------|----|---|
| name | Y | そのプラグインの名前から成る短い文字列 |
| qgisMinimumVersion | Y | ドット記法による最小 QGIS バージョン |
| qgisMaximumVersion | N | ドット記法による最大 QGIS バージョン |
| description | Y | プラグインを説明する短いテキスト。HTML は不可 |
| about | Y | プラグインの詳細を説明するより長いテキスト。HTML は不可 |
| version | Y | バージョンをドット記法で記した短い文字列 |
| author | Y | 著者の名前 |
| email | Y | 著者の E メール。ウェブサイトでログインしたユーザのみに表示されるが、プラグインをインストールした後はプラグインマネージャで見ることができる |
| changelog | N | 文字列。改行して複数行になってもよい。HTML 不可 |
| experimental | N | boolean flag, <i>True</i> or <i>False</i> - <i>True</i> if this version is experimental |
| deprecated | N | <i>True</i> か <i>False</i> のいずれかの値。単にアップロードしたバージョンに対してではなく、そのプラグインすべてに適用される |
| tags | N | コンマで区切ったリスト。個々のタグの内部ではスペース可 |
| homepage | N | プラグインのホームページを示す有効な URL |
| repository | Y | ソースコードのリポジトリの有効な URL |
| tracker | N | チケットとバグ報告のための有効な URL |
| icon | N | 画像のファイル名もしくは (プラグインの圧縮パッケージのベースフォルダからの) 相対パス。画像はウェブに適したフォーマット (PNG, JPEG) で |
| category | N | <i>Raster</i> 、 <i>Vector</i> 、 <i>Database</i> 、 <i>Web</i> のうちのいずれか |
| plugin_dependencies | N | pip と同様の、インストールされるべきその他のプラグインのコンマ区切りリスト |
| server | N | boolean flag, <i>True</i> or <i>False</i> , determines if the plugin has a server interface |
| hasProcessingProvider | N | <i>True</i> か <i>False</i> のいずれかの値。このプラグインがプロセッシングアルゴリズムを提供するかどうかを特定する。 |

デフォルトでプラグインはプラグインメニューに置かれます (プラグインをメニューエントリに追加する方法は次のセクションで見ます)。他にもラスターメニューやベクターメニュー、データベースメニュー、Webメニューに置くこともできます。

表示するメニューを特定するために、対応する "category" メタデータエントリが存在します。これに従ってプラグインは分類することができます。このメタデータエントリは、ユーザへのヒントとして使われ、このプラグインを使うためにはどのメニューを探せばよいかを伝えます。"category" に使うことのできる値は、Vector、Raster、Database、Web です。例えば、あなたのプラグインを *Raster* メニューから使えるようにするには、`metadata.txt` にそのように追加します。

```
category=Raster
```

注釈: `qgisMaximumVersion` が空欄の場合、*Official Python plugin repository* にアップロードする際に、自動的にメジャーバージョン + .99 に設定されます。

metadata.txt のための例です。

```
; the next section is mandatory

[general]
name=HelloWorld
email=me@example.com
author=Just Me
qgisMinimumVersion=3.0
description=This is an example plugin for greeting the world.
    Multiline is allowed:
    lines starting with spaces belong to the same
    field, in this case to the "description" field.
    HTML formatting is not allowed.
about=This paragraph can contain a detailed description
    of the plugin. Multiline is allowed, HTML is not.
version=version 1.2
tracker=http://bugs.itopen.it
repository=http://www.itopen.it/repo
; end of mandatory metadata

; start of optional metadata
category=Raster
changelog=The changelog lists the plugin versions
    and their changes as in the example below:
    1.0 - First stable release
    0.9 - All features implemented
    0.8 - First testing release

; Tags are in comma separated value format, spaces are allowed within the
; tag name.
; Tags should be in English language. Please also check for existing tags and
; synonyms before creating a new one.
tags=wkt,raster,hello world

; these metadata can be empty, they will eventually become mandatory.
homepage=https://www.itopen.it
icon=icon.png

; experimental flag (applies to the single version)
experimental=True

; deprecated flag (applies to the whole plugin and not only to the uploaded
↪version)
```

(次のページに続く)

```

deprecated=False

; if empty, it will be automatically set to major version + .99
qgisMaximumVersion=3.99

; Since QGIS 3.8, a comma separated list of plugins to be installed
; (or upgraded) can be specified.
; The example below will try to install (or upgrade) "MyOtherPlugin" version 1.12
; and any version of "YetAnotherPlugin"
plugin_dependencies=MyOtherPlugin==1.12,YetAnotherPlugin

```

__init__.py

このファイルは Python のインポートシステムで必要とされます。同時に QGIS はこのファイルの `classFactory()` 関数を必要とします。この関数はプラグインが QGIS に読み込まれる時に呼ばれます。この関数は `QgisInterface` のインスタンスへの参照を受け取り、`:file:`mainplugin.py` ファイルのあなたのプラグインオブジェクトを返さなければなりません (ここでの事例では `TestPlugin` がそれに当たります。以下を見てください)。 `__init__.py` は次のようにならなければなりません。

```

def classFactory(iface):
    from .mainPlugin import TestPlugin
    return TestPlugin(iface)

# any other initialisation needed

```

mainPlugin.py

このファイルが魔法の起こる場所です。魔法はこんな風です (例えば `mainPlugin.py` の場合)。

```

from qgis.PyQt.QtGui import *
from qgis.PyQt.QtWidgets import *

# initialize Qt resources from file resources.py
from . import resources

class TestPlugin:

    def __init__(self, iface):
        # save reference to the QGIS interface
        self.iface = iface

    def initGui(self):
        # create action that will start plugin configuration
        self.action = QAction(QIcon(":/plugins/testplug/icon.png"),
                               "Test plugin",
                               self.iface.mainWindow())
        self.action.setObjectName("testAction")
        self.action.setWhatsThis("Configuration for test plugin")

```

(次のページに続く)

(前のページからの続き)

```

self.action.setStatusTip("This is status tip")
self.action.triggered.connect(self.run)

# add toolbar button and menu item
self.iface.addToolBarIcon(self.action)
self.iface.addPluginToMenu("&Test plugins", self.action)

# connect to signal renderComplete which is emitted when canvas
# rendering is done
self.iface.mapCanvas().renderComplete.connect(self.renderTest)

def unload(self):
    # remove the plugin menu item and icon
    self.iface.removePluginMenu("&Test plugins", self.action)
    self.iface.removeToolBarIcon(self.action)

    # disconnect from signal of the canvas
    self.iface.mapCanvas().renderComplete.disconnect(self.renderTest)

def run(self):
    # create and show a configuration dialog or something similar
    print("TestPlugin: run called!")

def renderTest(self, painter):
    # use painter for drawing to map canvas
    print("TestPlugin: renderTest called!")

```

メインプラグインソースファイル（例えば `mainPlugin.py`）に必須のプラグイン関数は以下の 3 つだけです。

- `__init__` は QGIS interface へのアクセスを与えます。
- `initGui()` はプラグインが読み込まれた時に呼ばれます。
- `unload()` はプラグインがアンロードされた時に呼ばれます。

上記の例では、`addPluginToMenu` が使われています。これは対応するメニューアクションを `:menuselection:` プラグイン () メニューに追加します。他のメニューにアクションを追加するにはまた別のメソッドがあります。以下はそれらのメソッドのリストです。

- `addPluginToRasterMenu()`
- `addPluginToVectorMenu()`
- `addPluginToDatabaseMenu()`
- `addPluginToWebMenu()`

これらはすべて `:meth: addPluginToMenu` メソッドと同じシンタックスを持ちます。

これらのあらかじめ定義されたメソッドのうちのひとつに、あなたのプラグインメニューを追加することは、プラグインエントリがどのように組織されているかの一貫性を保つためにも推奨されます。しかし、次の例が

示すように、独自のカスタムメニューグループを直接メニューバーに追加することもできます。

```
def initGui(self):
    self.menu = QMenu(self.iface.mainWindow())
    self.menu.setObjectName("testMenu")
    self.menu.setTitle("MyMenu")

    self.action = QAction(QIcon(":/plugins/testplug/icon.png"),
                           "Test plugin",
                           self.iface.mainWindow())
    self.action.setObjectName("testAction")
    self.action.setWhatsThis("Configuration for test plugin")
    self.action.setStatusTip("This is status tip")
    self.action.triggered.connect(self.run)
    self.menu.addAction(self.action)

    menuBar = self.iface.mainWindow().menuBar()
    menuBar.insertMenu(self.iface.firstRightStandardMenu().menuAction(),
                       self.menu)

def unload(self):
    self.menu.deleteLater()
```

カスタマイズが可能になるように、QAction クラスと QMenu クラスの objectName にあなたのプラグインの固有名を設定するのを忘れないでください。

Resource ファイル

initGui() 関数の中に resource ファイル (この例では resources.qrc という名前です) からのアイコンが使われているのを見ることができます。

```
<RCC>
  <qresource prefix="/plugins/testplug" >
    <file>icon.png</file>
  </qresource>
</RCC>
```

他のプラグインや他の QGIS のパーツと衝突しないよう接頭辞を使うのは良いことです。さもないと望んでいないリソースを読み込んでしまうかもしれません。さてあと必要なのはこのリソースを含む Python ファイルを生成することです。これは **pyrcc5** コマンドを使って次のようにして行われます。

```
pyrcc5 -o resources.py resources.qrc
```

注釈: In Windows environments, attempting to run the **pyrcc5** from Command Prompt or Powershell will probably result in the error "Windows cannot access the specified device, path, or file [...]". The easiest solution is probably to use the OSGeo4W Shell but if you are comfortable modifying the PATH environment variable or specifying the path to the executable explicitly you should be able to find it at <Your QGIS Install


```
Directory>\bin\pyrcc5.exe.
```

はい、これですべてです。...何も複雑なことはありませんね (^_^)

すべてを正しく行っていれば、プラグインマネージャであなたのプラグインを見つけて読み込むことができます。そしてツールバーのアイコンが適切なメニュー項目が選択された時には、コンソールにメッセージが表示されるはずですが。

実際にプラグイン作成の作業を行うときは、別の（作業用の）ディレクトリでプラグインを書いて、UI とリソースファイルを生成すると、プラグインを実際のインストールディレクトリにインストールするのは、make ファイルを書いてそれにさせるのが賢いやり方でしょう。

16.1.3 Documentation

The documentation for the plugin can be written as HTML help files. The `qgis.utils` module provides a function, `showPluginHelp()` which will open the help file browser, in the same way as other QGIS help.

The `showPluginHelp()` function looks for help files in the same directory as the calling module. It will look for, in turn, `index-ll_cc.html`, `index-ll.html`, `index-en.html`, `index-en_us.html` and `index.html`, displaying whichever it finds first. Here `ll_cc` is the QGIS locale. This allows multiple translations of the documentation to be included with the plugin.

The `showPluginHelp()` function can also take parameters `packageName`, which identifies a specific plugin for which the help will be displayed, `filename`, which can replace "index" in the names of files being searched, and `section`, which is the name of an html anchor tag in the document on which the browser will be positioned.

16.1.4 Translation

With a few steps you can set up the environment for the plugin localization so that depending on the locale settings of your computer the plugin will be loaded in different languages.

Software requirements

The easiest way to create and manage all the translation files is to install [Qt Linguist](#). In a Debian-based GNU/Linux environment you can install it typing:

```
sudo apt install qttools5-dev-tools
```

Files and directory

When you create the plugin you will find the `i18n` folder within the main plugin directory.

All the translation files have to be within this directory.

.pro file

First you should create a `.pro` file, that is a *project* file that can be managed by **Qt Linguist**.

In this `.pro` file you have to specify all the files and forms you want to translate. This file is used to set up the localization files and variables. A possible project file, matching the structure of our *example plugin*:

```
FORMS = ../form.ui
SOURCES = ../your_plugin.py
TRANSLATIONS = your_plugin_it.ts
```

Your plugin might follow a more complex structure, and it might be distributed across several files. If this is the case, keep in mind that `pylupdate5`, the program we use to read the `.pro` file and update the translatable string, does not expand wild card characters, so you need to place every file explicitly in the `.pro` file. Your project file might then look like something like this:

```
FORMS = ../ui/about.ui ../ui/feedback.ui \
        ../ui/main_dialog.ui
SOURCES = ../your_plugin.py ../computation.py \
        ../utils.py
```

Furthermore, the `your_plugin.py` file is the file that *calls* all the menu and sub-menus of your plugin in the QGIS toolbar and you want to translate them all.

Finally with the `TRANSLATIONS` variable you can specify the translation languages you want.

警告: Be sure to name the `ts` file like `your_plugin_ + language + .ts` otherwise the language loading will fail! Use the 2 letter shortcut for the language (**it** for Italian, **de** for German, etc...)

.ts file

Once you have created the `.pro` you are ready to generate the `.ts` file(s) for the language(s) of your plugin.

Open a terminal, go to `your_plugin/i18n` directory and type:

```
pylupdate5 your_plugin.pro
```

you should see the `your_plugin_language.ts` file(s).

Open the `.ts` file with **Qt Linguist** and start to translate.

.qm file

When you finish to translate your plugin (if some strings are not completed the source language for those strings will be used) you have to create the `.qm` file (the compiled `.ts` file that will be used by QGIS).

Just open a terminal `cd` in `your_plugin/i18n` directory and type:

```
lrelease your_plugin.ts
```

now, in the `i18n` directory you will see the `your_plugin.qm` file(s).

Translate using Makefile

Alternatively you can use the makefile to extract messages from python code and Qt dialogs, if you created your plugin with Plugin Builder. At the beginning of the Makefile there is a `LOCALES` variable:

```
LOCALES = en
```

Add the abbreviation of the language to this variable, for example for Hungarian language:

```
LOCALES = en hu
```

Now you can generate or update the `hu.ts` file (and the `en.ts` too) from the sources by:

```
make transup
```

After this, you have updated `.ts` file for all languages set in the `LOCALES` variable. Use **Qt Linguist** to translate the program messages. Finishing the translation the `.qm` files can be created by the `transcompile`:

```
make transcompile
```

You have to distribute `.ts` files with your plugin.

Load the plugin

In order to see the translation of your plugin, open QGIS, change the language (*Settings Options General*) and restart QGIS.

You should see your plugin in the correct language.

警告: If you change something in your plugin (new UIs, new menu, etc..) you have to **generate again** the update version of both `.ts` and `.qm` file, so run again the command of above.

16.1.5 Tips and Tricks

Plugin Reloader

During development of your plugin you will frequently need to reload it in QGIS for testing. This is very easy using the **Plugin Reloader** plugin. You can find it with the Plugin Manager.

Accessing Plugins

You can access all the classes of installed plugins from within QGIS using python, which can be handy for debugging purposes.

```
my_plugin = qgis.utils.plugins['My Plugin']
```

Log Messages

Plugins have their own tab within the log_message_panel.

Share your plugin

QGIS is hosting hundreds of plugins in the plugin repository. Consider sharing yours! It will extend the possibilities of QGIS and people will be able to learn from your code. All hosted plugins can be found and installed from within QGIS with the Plugin Manager.

Information and requirements are here: plugins.qgis.org.

16.2 短いコード

- *How to call a method by a key shortcut*
- *How to toggle Layers*
- *How to access attribute table of selected features*
- *Interface for plugin in the options dialog*

This section features code snippets to facilitate plugin development.

16.2.1 How to call a method by a key shortcut

In the plug-in add to the `initGui()`

```
self.key_action = QAction("Test Plugin", self.iface.mainWindow())
self.iface.registerMainWindowAction(self.key_action, "Ctrl+I") # action triggered_
↳by Ctrl+I
self.iface.addPluginToMenu("&Test plugins", self.key_action)
self.key_action.triggered.connect(self.key_action_triggered)
```

To `unload()` add

```
self.iface.unregisterMainWindowAction(self.key_action)
```

The method that is called when CTRL+I is pressed

```
def key_action_triggered(self):
    QMessageBox.information(self.iface.mainWindow(), "Ok", "You pressed Ctrl+I")
```

16.2.2 How to toggle Layers

There is an API to access layers in the legend. Here is an example that toggles the visibility of the active layer

```
root = QgsProject.instance().layerTreeRoot()
node = root.findLayer(iface.activeLayer().id())
new_state = Qt.Checked if node.isVisible() == Qt.Unchecked else Qt.Unchecked
node.setItemVisibilityChecked(new_state)
```

16.2.3 How to access attribute table of selected features

```
1 def change_value(value):
2     """Change the value in the second column for all selected features.
3
4     :param value: The new value.
5     """
6     layer = iface.activeLayer()
7     if layer:
8         count_selected = layer.selectedFeatureCount()
9         if count_selected > 0:
10            layer.startEditing()
11            id_features = layer.selectedFeatureIds()
12            for i in id_features:
13                layer.changeAttributeValue(i, 1, value) # 1 being the second column
14            layer.commitChanges()
15        else:
16            iface.messageBar().pushCritical("Error",
17                "Please select at least one feature from current layer")
```

(次のページに続く)

```

18     else:
19         iface.messageBar().pushCritical("Error", "Please select a layer")
20
21     # The method requires one parameter (the new value for the second
22     # field of the selected feature(s)) and can be called by
23     change_value(50)

```

16.2.4 Interface for plugin in the options dialog

You can add a custom plugin options tab to *Settings Options*. This is preferable over adding a specific main menu entry for your plugin's options, as it keeps all of the QGIS application settings and plugin settings in a single place which is easy for users to discover and navigate.

The following snippet will just add a new blank tab for the plugin's settings, ready for you to populate with all the options and settings specific to your plugin. You can split the following classes into different files. In this example, we are adding two classes into the main `mainPlugin.py` file.

```

1 class MyPluginOptionsFactory(QgsOptionsWidgetFactory):
2
3     def __init__(self):
4         super().__init__()
5
6     def icon(self):
7         return QIcon('icons/my_plugin_icon.svg')
8
9     def createWidget(self, parent):
10        return ConfigOptionsPage(parent)
11
12
13 class ConfigOptionsPage(QgsOptionsPageWidget):
14
15     def __init__(self, parent):
16         super().__init__(parent)
17         layout = QHBoxLayout()
18         layout.setContentsMargins(0, 0, 0, 0)
19         self.setLayout(layout)

```

Finally we are adding the imports and modifying the `__init__` function:

```

1 from qgis.PyQt.QtWidgets import QHBoxLayout
2 from qgis.gui import QgsOptionsWidgetFactory, QgsOptionsPageWidget
3
4
5 class MyPlugin:
6     """QGIS Plugin Implementation."""
7
8     def __init__(self, iface):
9         """Constructor.

```

(前のページからの続き)

```

10
11     :param iface: An interface instance that will be passed to this class
12                   which provides the hook by which you can manipulate the QGIS
13                   application at run time.
14     :type iface: QgsInterface
15     """
16     # Save reference to the QGIS interface
17     self.iface = iface
18
19
20     def initGui(self):
21         self.options_factory = MyPluginOptionsFactory()
22         self.options_factory.setTitle(self.tr('My Plugin'))
23         iface.registerOptionsWidgetFactory(self.options_factory)
24
25     def unload(self):
26         iface.unregisterOptionsWidgetFactory(self.options_factory)

```

ちなみに: You can apply a similar logic to add the plugin custom option to the layer properties dialog using the classes `QgsMapLayerConfigWidgetFactory` and `QgsMapLayerConfigWidget`.

16.3 Using Plugin Layers

If your plugin uses its own methods to render a map layer, writing your own layer type based on `QgsPluginLayer` might be the best way to implement that.

16.3.1 Subclassing QgsPluginLayer

Below is an example of a minimal `QgsPluginLayer` implementation. It is based on the original code of the [Watermark example plugin](#).

The custom renderer is the part of the implement that defines the actual drawing on the canvas.

```

1 class WatermarkLayerRenderer(QgsMapLayerRenderer):
2
3     def __init__(self, layerId, rendererContext):
4         super().__init__(layerId, rendererContext)
5
6     def render(self):
7         image = QImage("/usr/share/icons/hicolor/128x128/apps/qgis.png")
8         painter = self.rendererContext().painter()
9         painter.save()
10        painter.drawImage(10, 10, image)
11        painter.restore()
12        return True

```

(次のページに続く)

```
13
14 class WatermarkPluginLayer(QgsPluginLayer):
15
16     LAYER_TYPE="watermark"
17
18     def __init__(self):
19         super().__init__(WatermarkPluginLayer.LAYER_TYPE, "Watermark plugin layer")
20         self.setValid(True)
21
22     def createMapRenderer(self, rendererContext):
23         return WatermarkLayerRenderer(self.id(), rendererContext)
24
25     def setTransformContext(self, ct):
26         pass
27
28     # Methods for reading and writing specific information to the project file can
29     # also be added:
30
31     def readXml(self, node, context):
32         pass
33
34     def writeXml(self, node, doc, context):
35         pass
```

The plugin layer can be added to the project and to the canvas as any other map layer:

```
plugin_layer = WatermarkPluginLayer()
QgsProject.instance().addMapLayer(plugin_layer)
```

When loading a project containing such a layer, a factory class is needed:

```
1 class WatermarkPluginLayerType(QgsPluginLayerType):
2
3     def __init__(self):
4         super().__init__(WatermarkPluginLayer.LAYER_TYPE)
5
6     def createLayer(self):
7         return WatermarkPluginLayer()
8
9     # You can also add GUI code for displaying custom information
10    # in the layer properties
11    def showLayerProperties(self, layer):
12        pass
13
14
15    # Keep a reference to the instance in Python so it won't
16    # be garbage collected
17    plt = WatermarkPluginLayerType()
18
19    assert QgsApplication.pluginLayerRegistry().addPluginLayerType(plt)
```


16.4 プラグインを書いてデバッグするための IDE 設定

- *Useful plugins for writing Python plugins*
- *A note on configuring your IDE on Linux and Windows*
- *Debugging using Pyscripter IDE (Windows)*
- *Debugging using Eclipse and PyDev*
 - インストール
 - *Eclipse* をセットアップする
 - *Configuring the debugger*
 - *Eclipse* に API を理解させる
- *Debugging with PyCharm on Ubuntu with a compiled QGIS*
- *PDB を利用してデバッグする*

プログラマには皆それぞれ自分の好みの IDE /テキストエディタがありますが、ここに人気の IDE を設定し、QGIS の Python プラグインを書いたりデバッグするためのいくつかの推奨事項があります。

16.4.1 Useful plugins for writing Python plugins

Some plugins are convenient when writing Python plugins. From *Plugins Manage and Install plugins...*, install:

- *Plugin reloader*: This will let you reload a plugin and pull new changes without restarting QGIS.
- *First Aid*: This will add a Python console and local debugger to inspect variables when an exception is raised from a plugin.

警告: *Despite our constant efforts, information beyond this line may not be updated for QGIS 3. Refer to <https://qgis.org/pyqgis/master> for the python API documentation or, give a hand to update the chapters you know about. Thanks.*

16.4.2 A note on configuring your IDE on Linux and Windows

On Linux, all that usually needs to be done is to add the QGIS library locations to the user's `PYTHONPATH` environment variable. Under most distributions, this can be done by editing `~/.bashrc` or `~/.bash-profile` with the following line (tested on OpenSUSE Tumbleweed):

```
export PYTHONPATH="$PYTHONPATH:/usr/share/qgis/python/plugins:/usr/share/qgis/
↳python"
```

Save the file and implement the environment settings by using the following shell command:

```
source ~/.bashrc
```

On Windows, you need to make sure that you have the same environment settings and use the same libraries and interpreter as QGIS. The fastest way to do this is to modify the startup batch file of QGIS.

OSGeo4W インストーラを使用した場合は OSGeo4W インストールの `bin` フォルダの下にこれを見つけることができます。 `C:\OSGeo4W\bin\qgis-unstable.bat` のようなものを探してください。

16.4.3 Debugging using Pyscripter IDE (Windows)

For using Pyscripter IDE, here's what you have to do:

1. Make a copy of `qgis-unstable.bat` and rename it `pyscripter.bat`.
2. Open it in an editor. And remove the last line, the one that starts QGIS.
3. Add a line that points to your Pyscripter executable and add the command line argument that sets the version of Python to be used
4. さらに、QGIS が使用する Python の DLL を Pyscripter が見つけられるフォルダを指す引数を追加します。これは OSGeoW インストールの `bin` フォルダの下に見つけることができます

```
@echo off
SET OSGEO4W_ROOT=C:\OSGeo4W
call "%OSGEO4W_ROOT%"\bin\o4w_env.bat
call "%OSGEO4W_ROOT%"\bin\gdal16.bat
@echo off
path %PATH%;%GISBASE%\bin
Start C:\pyscripter\pyscripter.exe --python25 --pythondllpath=C:\OSGeo4W\bin
```


5. Now when you double click this batch file it will start Pyscripter, with the correct path.

More popular than Pyscripter, Eclipse is a common choice among developers. In the following section, we will be explaining how to configure it for developing and testing plugins.

16.4.4 Debugging using Eclipse and PyDev

インストール

To use Eclipse, make sure you have installed the following

- Eclipse
- Aptana Studio 3 Plugin or PyDev
- QGIS 2.x
- You may also want to install **Remote Debug**, a QGIS plugin. At the moment it's still experimental so enable  *Experimental plugins* under *Plugins Manage and Install plugins... Options* beforehand.

To prepare your environment for using Eclipse in Windows, you should also create a batch file and use it to start Eclipse:

1. Locate the folder where `qgis_core.dll` resides in. Normally this is `C:\OSGeo4W\apps\qgis\bin`, but if you compiled your own QGIS application this is in your build folder in `output/bin/RelWithDebInfo`
2. Locate your `eclipse.exe` executable.
3. Create the following script and use this to start eclipse when developing QGIS plugins.

```
call "C:\OSGeo4W\bin\o4w_env.bat"
set PATH=%PATH%;C:\path\to\your\qgis_core.dll\parent\folder
C:\path\to\your\eclipse.exe
```

Eclipse をセットアップする

1. Eclipse で、新しいプロジェクトを作成します。一般的なプロジェクトを選択しておいて本当のソースを後でリンクできるので、このプロジェクトをどこに配置するかは実際は問題になりません。
2. Right-click your new project and choose *New Folder*.
3. 詳細 をクリックし、別の場所にリンク（リンクフォルダ）を選択します。すでにデバッグしたいソースがある場合はこれらを選択してください。そうでない場合は、すでに説明したようにフォルダを作成してください。

するとビュー プロジェクトエクスプローラ で、ソースツリーがポップアップしますので、コードでの作業を開始できます。すでに利用可能な構文の強調表示や他のすべての強力な IDE ツールが使用できるようになっています。

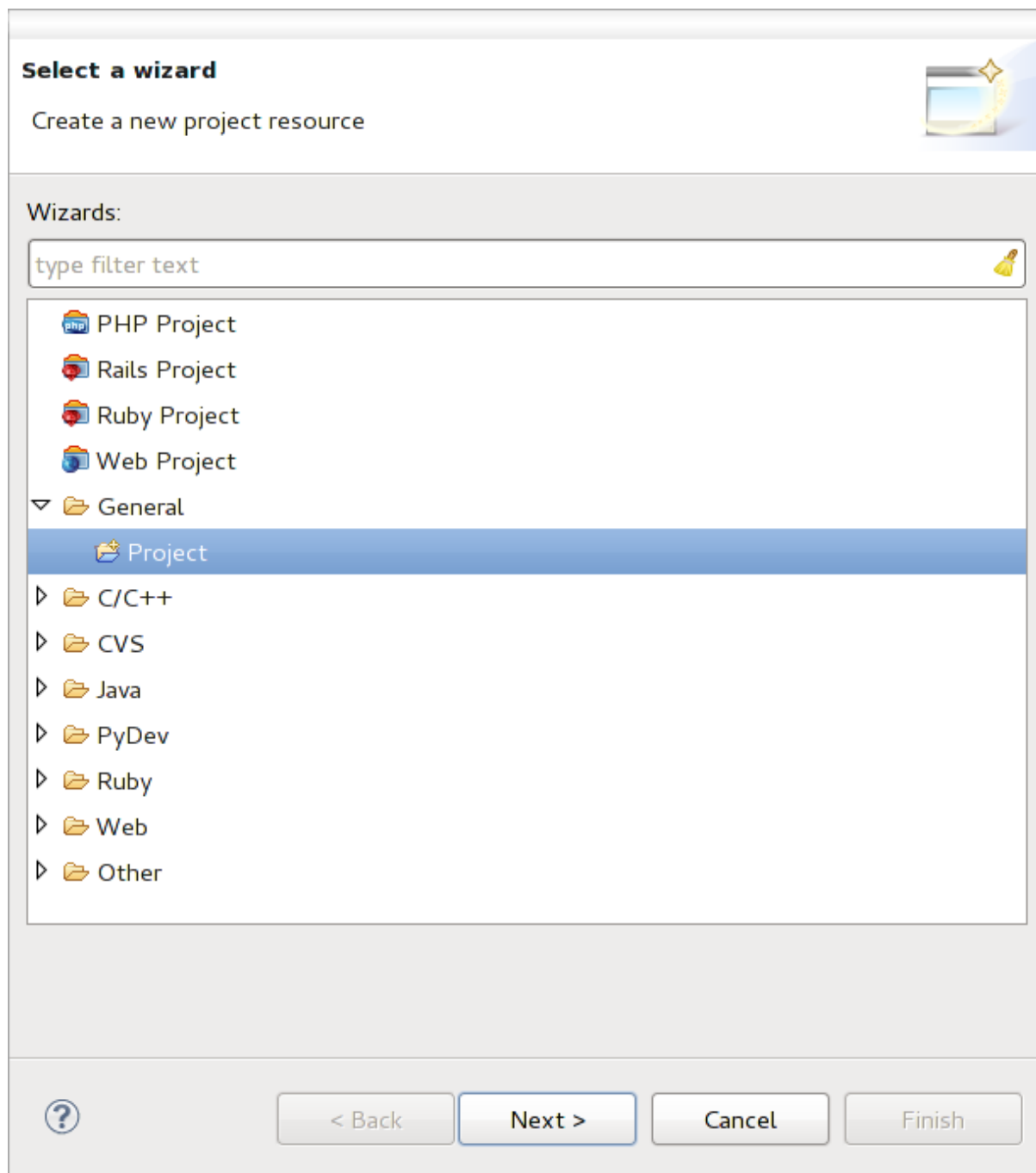


図 16.1 Eclipse project

Configuring the debugger

To get the debugger working:

1. Switch to the Debug perspective in Eclipse (*Window Open Perspective Other Debug*).
2. start the PyDev debug server by choosing *PyDev Start Debug Server*.
3. Eclipse は QGIS からデバッグサーバーへの接続を待っています。QGIS がデバッグサーバーに接続すると、Python スクリプトを制御できます。それはまさに私たちが *Remote Debug* プラグインをインストールしたものです。だからまだ起動していなければ QGIS を起動し、バグのシンボルをクリックして

ください。

ここでブレークポイントを設定できます。コードがそこに達すると実行は停止し、プラグインの現在の状態を検査できます。(ブレークポイントは下の画像の緑色の点です。ブレークポイントを設定したい行の左空白でダブルクリックすることで設定します)。

```

87         self.verticalExaggerationChanged.emit(val)
88
89     def printProfile(self):
90         printer = QPrinter( QPrinter.HighResolution )
91         printer.setOutputFormat( QPrinter.PdfFormat )
92         printer.setPaperSize( QPrinter.A4 )
93         printer.setOrientation( QPrinter.Landscape )
94
95         printPreviewDlg = QPrintPreviewDialog( )
96         printPreviewDlg.paintRequested.connect( self.printRequested )
97
98         printPreviewDlg.exec_()
99
100     @pyqtSlot( QPrinter )
101     def printRequested( self, printer ):
102         self.webView.print_( printer )

```

図 16.2 ブレークポイント

今利用できる非常に興味深いものはデバッグコンソールです。先に進む前に、現在実行がブレークポイントで停止していることを確認してください。

1. Open the Console view (*Window Show view*). It will show the *Debug Server* console which is not very interesting. But there is a button *Open Console* which lets you change to a more interesting *PyDev Debug Console*.
2. Click the arrow next to the *Open Console* button and choose *PyDev Console*. A window opens up to ask you which console you want to start.
3. Choose *PyDev Debug Console*. In case its greyed out and tells you to Start the debugger and select the valid frame, make sure that you've got the remote debugger attached and are currently on a breakpoint.

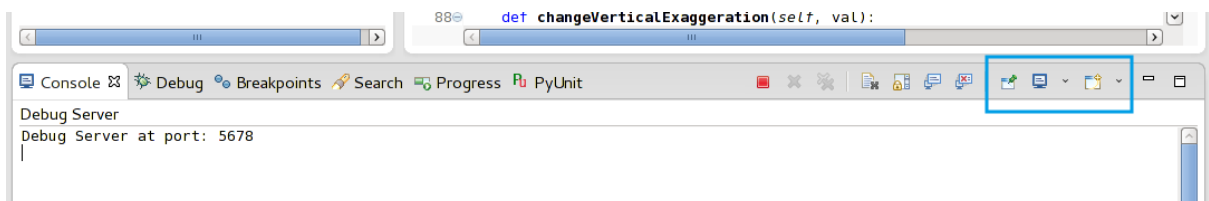


図 16.3 PyDev Debug Console

You have now an interactive console which lets you test any commands from within the current context. You can manipulate variables or make API calls or whatever you like.

ちなみに: ちょっと面倒なのですが、コマンドを入力するたびコンソールはデバッグサーバーに戻ります。この動作を停止するには、デバッグサーバーページで *Pin Console* ボタンをクリックしますが、少なくとも現在のデバッグセッションではこの決定は記憶されます。

Eclipse に API を理解させる

非常に便利な機能は、Eclipse が実際に QGIS の API についてわかっているようにすることです。これにより、タイプミスがないかコードを確認できます。これだけでなく、Eclipse でのインポートから API 呼び出しへ自動入力する支援を可能にします。

これを行うため、Eclipse では QGIS ライブラリファイルを解析し、そこにすべての情報を取得します。しなければならないことは、どこのライブラリを検索するかを Eclipse に伝えることです。

1. ウィンドウ 設定 *PyDev* インタプリタ *Python* をクリック。

ウィンドウの上部と下部にいくつかのタブで設定済みの Python インタプリタ（現在 QGIS のための python2.7）が表示されます。私たちにとって興味深いタブは、ライブラリ および 強制ビルトイン です。

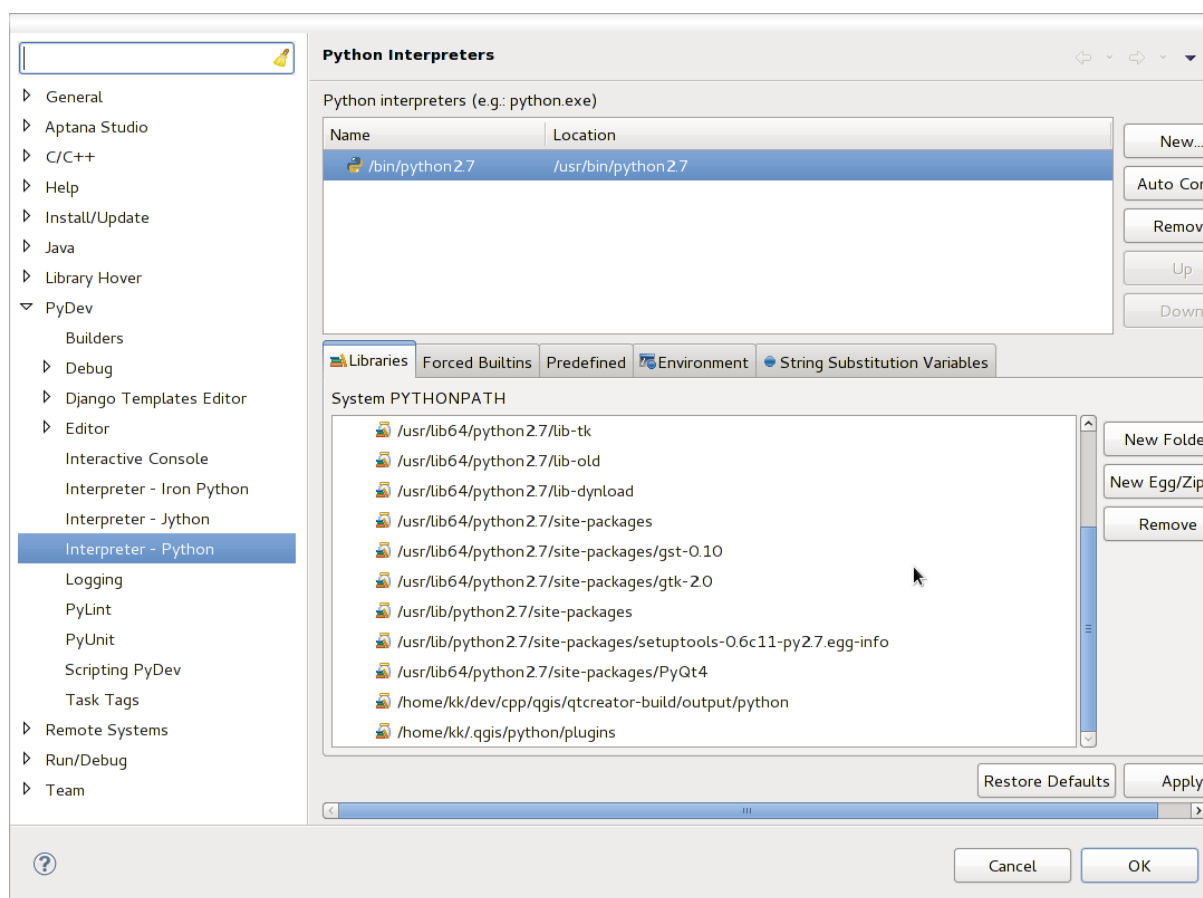


図 16.4 PyDev Debug Console

2. First open the Libraries tab.
3. Add a New Folder and choose the python folder of your QGIS installation. If you do not know where this folder is (it's not the plugins folder):
 1. Open QGIS
 2. Start a python console
 3. Enter `qgis`

4. and press Enter. It will show you which QGIS module it uses and its path.
5. Strip the trailing `/qgis/__init__.pyc` from this path and you've got the path you are looking for.
4. You should also add your plugins folder here (it is in `python/plugins` under the user profile folder).
5. Next jump to the *Forced Builtins* tab, click on *New...* and enter `qgis`. This will make Eclipse parse the QGIS API. You probably also want Eclipse to know about the PyQt API. Therefore also add PyQt as forced builtin. That should probably already be present in your libraries tab.
6. Click *OK* and you're done.

注釈: Every time the QGIS API changes (e.g. if you're compiling QGIS master and the SIP file changed), you should go back to this page and simply click *Apply*. This will let Eclipse parse all the libraries again.

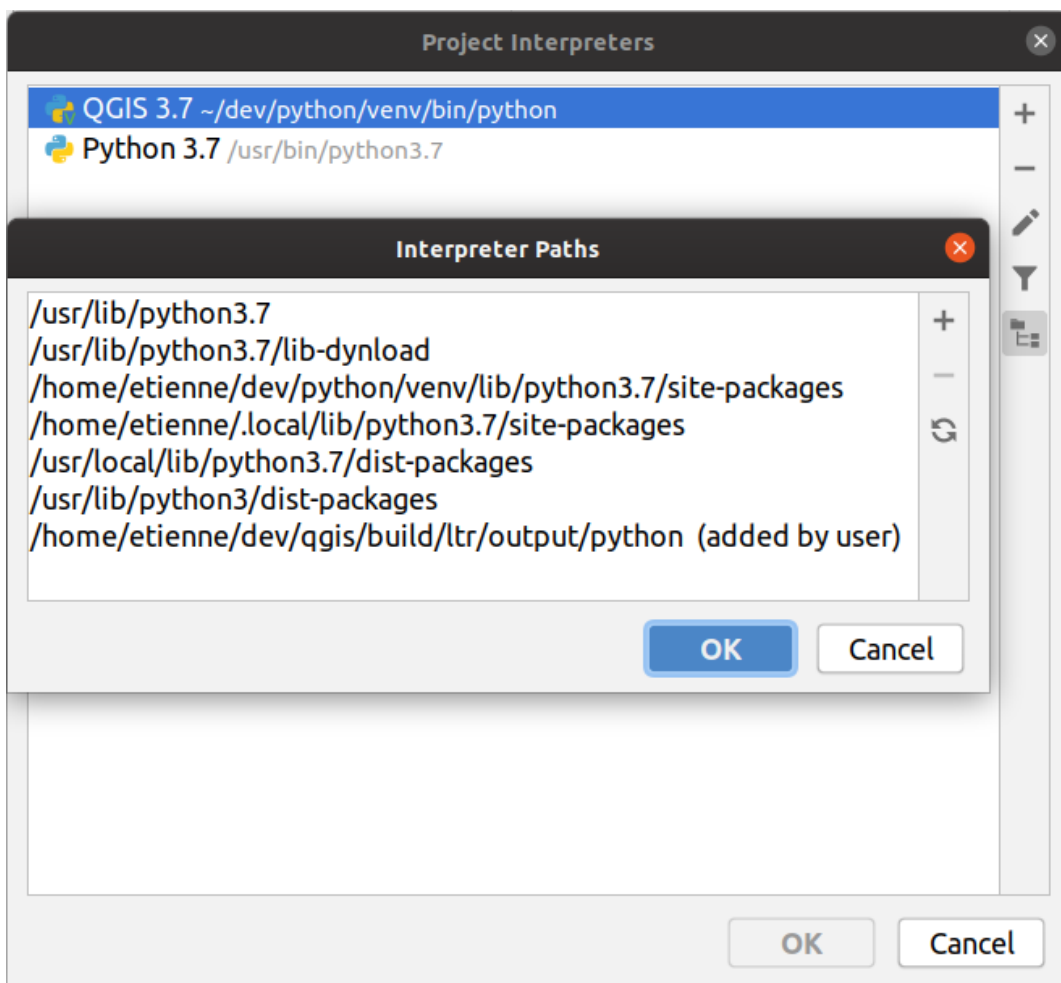
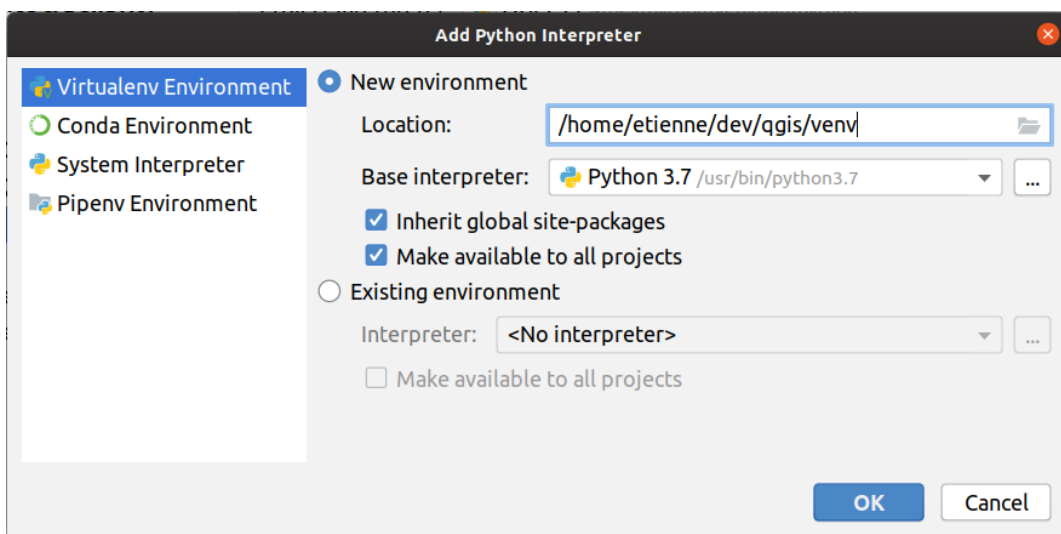
16.4.5 Debugging with PyCharm on Ubuntu with a compiled QGIS

PyCharm is an IDE for Python developed by JetBrains. There is a free version called Community Edition and a paid one called Professional. You can download PyCharm on the website: <https://www.jetbrains.com/pycharm/download>

We are assuming that you have compiled QGIS on Ubuntu with the given build directory `~/dev/qgis/build/master`. It's not compulsory to have a self compiled QGIS, but only this has been tested. Paths must be adapted.

1. In PyCharm, in your *Project Properties, Project Interpreter*, we are going to create a Python Virtual environment called QGIS.
2. Click the small gear and then *Add*.
3. Select *Virtualenv environment*.
4. Select a generic location for all your Python projects such as `~/dev/qgis/venv` because we will use this Python interpreter for all our plugins.
5. Choose a Python 3 base interpreter available on your system and check the next two options *Inherit global site-packages* and *Make available to all projects*.
 1. Click *OK*, come back on the small gear and click *Show all*.
 2. In the new window, select your new interpreter QGIS and click the last icon in the vertical menu *Show paths for the selected interpreter*.
 3. Finally, add the following absolute path to the list `~/dev/qgis/build/master/output/python`.
 1. Restart PyCharm and you can start using this new Python virtual environment for all your plugins.

PyCharm will be aware of the QGIS API and also of the PyQt API if you use Qt provided by QGIS like `from qgis.PyQt.QtCore import QDir`. The autocompletion should work and PyCharm can inspect your code.



In the professional version of PyCharm, remote debugging is working well. For the Community edition, remote debugging is not available. You can only have access to a local debugger, meaning that the code must run *inside* PyCharm (as script or unittest), not in QGIS itself. For Python code running *in* QGIS, you might use the *First Aid* plugin mentioned above.

16.4.6 PDB を利用してデバッグする

If you do not use an IDE such as Eclipse or PyCharm, you can debug using PDB, following these steps.

1. First add this code in the spot where you would like to debug

```
# Use pdb for debugging
import pdb
# also import PyQtRemoveInputHook
from qgis.PyQt.QtCore import PyQtRemoveInputHook
# These lines allow you to set a breakpoint in the app
PyQtRemoveInputHook()
pdb.set_trace()
```

2. Then run QGIS from the command line.

On Linux do:

```
$ ./Qgis
```

On macOS do:

```
$ /Applications/Qgis.app/Contents/MacOS/Qgis
```

3. And when the application hits your breakpoint you can type in the console!

TODO: Add testing information

16.5 Releasing your plugin

- *Metadata and names*
- *Code and help*
- *Official Python plugin repository*
 - *Permissions*
 - *Trust management*
 - *Validation*

プラグインの準備ができ、そのプラグインが誰かのために役立つことあると思ったら、躊躇わずに *Official Python plugin repository* にアップロードしてください。そのページでは、プラグインのインストーラでうまく動作するプラグインを準備する方法についてパッケージ化のガイドラインがあります。あるいは、独自のプラグインのリポジトリを設定したい場合は、プラグインとそのメタデータの一覧を表示する単純な XML ファイルを作成してください。

Please take special care to the following suggestions:

16.5.1 Metadata and names

- avoid using a name too similar to existing plugins
- if your plugin has a similar functionality to an existing plugin, please explain the differences in the About field, so the user will know which one to use without the need to install and test it
- avoid repeating "plugin" in the name of the plugin itself
- use the description field in metadata for a 1 line description, the About field for more detailed instructions
- コードリポジトリ、バグトラッカー、およびホーム・ページを含めてください。これは非常に協働作業の可能性を向上させますし、利用可能な Web インフラストラクチャの 1 つ (GitHub、GitLab、Bitbucket など) で非常に簡単に行うことができます
- タグは注意して選択してください：情報価値のないもの（例：ベクター）を避け、すでに他のユーザーによって使用されているもの（プラグインの Web サイトを参照）を選んでください
- add a proper icon, do not leave the default one; see QGIS interface for a suggestion of the style to be used

16.5.2 Code and help

- do not include generated file (ui_*.py, resources_rc.py, generated help files...) and useless stuff (e.g. .gitignore) in repository
- add the plugin to the appropriate menu (Vector, Raster, Web, Database)
- 適切な場合（解析を実行するプラグイン）、処理フレームワークのサブプラグインとしてプラグインを追加することを検討してください：これによってユーザーはバッチでそれを実行し、より複雑なワークフローにそれを統合できるようになり、インターフェイスを設計する負担がなくなります
- 最低限の文書を、そしてテストと理解に役立つ場合はサンプルデータを含めてください。

16.5.3 Official Python plugin repository

You can find the *official* Python plugin repository at <https://plugins.qgis.org/>.

In order to use the official repository you must obtain an OSGEO ID from the [OSGEO web portal](#).

Once you have uploaded your plugin it will be approved by a staff member and you will be notified.

TODO: Insert a link to the governance document

Permissions

These rules have been implemented in the official plugin repository:

- every registered user can add a new plugin
- *staff* users can approve or disapprove all plugin versions
- users which have the special permission *plugins.can_approve* get the versions they upload automatically approved
- users which have the special permission *plugins.can_approve* can approve versions uploaded by others as long as they are in the list of the plugin *owners*
- a particular plugin can be deleted and edited only by *staff* users and plugin *owners*
- if a user without *plugins.can_approve* permission uploads a new version, the plugin version is automatically unapproved.

Trust management

Staff members can grant *trust* to selected plugin creators setting *plugins.can_approve* permission through the front-end application.

The plugin details view offers direct links to grant trust to the plugin creator or the plugin *owners*.

Validation

Plugin's metadata are automatically imported and validated from the compressed package when the plugin is uploaded.

Here are some validation rules that you should aware of when you want to upload a plugin on the official repository:

1. プラグインを含むメインフォルダの名前は、ASCII 文字 (A-Z および a-z)、数字、アンダースコア (_) とマイナス (-) しか含んではならず、また数字で始めることはできません。
2. `metadata.txt` is required
3. all required metadata listed in *metadata table* must be present

4. the *version* metadata field must be unique

Plugin structure

Following the validation rules the compressed (.zip) package of your plugin must have a specific structure to validate as a functional plugin. As the plugin will be unzipped inside the users plugins folder it must have it's own directory inside the .zip file to not interfere with other plugins. Mandatory files are: `metadata.txt` and `__init__.py`. But it would be nice to have a `README` and of course an icon to represent the plugin (`resources.qrc`). Following is an example of how a `plugin.zip` should look like.

```
plugin.zip
pluginfolder/
|-- i18n
|   |-- translation_file_de.ts
|-- img
|   |-- icon.png
|   `-- iconsource.svg
|-- __init__.py
|-- Makefile
|-- metadata.txt
|-- more_code.py
|-- main_code.py
|-- README
|-- resources.qrc
|-- resources_rc.py
`-- ui_Qt_user_interface_file.ui
```

Python プログラミング言語を使ってプラグインを作ることができます。C++ で書かれた従来のプラグインと比べると、Python 言語の動的な性質のおかげで、より書きやすく、より理解しやすく、そしてより配布しやすくなっています。

Python プラグインは、C++ プラグインとともに、QGIS プラグインマネージャに表示されます。これらのプラグインは、`~/ (UserProfile) /python/plugins` と、以下のパスから検索されます。

- UNIX/Mac: `(qgis_prefix) /share/qgis/python/plugins`
- Windows: `(qgis_prefix) /python/plugins`

~ と `(UserProfile)` の定義は、`core_and_external_plugins` を参照してください。

注釈: `QGIS_PLUGINPATH` に既存のディレクトリ・パスを設定することにより、プラグインが検索されるパスのリストに、既存のパスを加えることができます。

第 17 章

プロセッシングプラグインを書く

- イチから作る
- プラグインをアップデートする

開発しようとしているプラグインの種類によっては、プロセッシングアルゴリズム（またはそれらのセット）として機能を追加する方が良い場合もあるでしょう。そうすれば、QGIS 内でのより良い統合がなされ（これは、モデラーやバッチ処理インターフェイスといった、「プロセッシング」のコンポーネントの中で実行できるためです）、また開発時間の短縮も期待できます（「プロセッシング」が作業の大部分を肩代わりしてくれるからです）。

開発したアルゴリズムを配布するためには、アルゴリズムをプロセッシングツールボックスに追加するためのプラグインを新しく作る必要があります。このプラグインにはアルゴリズムプロバイダを含ませるとともに、プラグインの初期化の際にアルゴリズムがツールボックスに登録されるようにする必要があります。

17.1 イチから作る

アルゴリズムプロバイダを含むプラグインをイチから作るには、Plugin Builder を使って以下のステップに従います。

1. **Plugin Builder** プラグインをインストールする
2. Plugin Builder を使用して新しくプラグインを作成します。Plugin Builder が使用するテンプレートをきいてきたら、「プロセッシングプロバイダ」を選択します。
3. 生成されたプラグインには、アルゴリズムをひとつ持つプロバイダが含まれています。プロバイダファイルおよびアルゴリズムファイルには両方ともに十分なコメントがついていて、プロバイダを修正したりさらにアルゴリズムを追加する方法についての情報が含まれています。詳細については、それらを参照してください。

17.2 プラグインをアップデートする

すでに作成済みのプラグインをプロセッシングに追加したい場合は、さらにコードを追加する必要があります。

1. `metadata.txt` ファイルに以下の変数を追加する必要があります。

```
hasProcessingProvider=yes
```

2. `initGui` メソッドによってプラグインのセットアップを担う Python ファイルでは、幾つかのコードを直す必要があります。

```
1 from qgis.core import QgsApplication
2 from processing_provider.provider import Provider
3
4 class YourPluginName():
5
6     def __init__(self):
7         self.provider = None
8
9     def initProcessing(self):
10        self.provider = Provider()
11        QgsApplication.processingRegistry().addProvider(self.provider)
12
13    def initGui(self):
14        self.initProcessing()
15
16    def unload(self):
17        QgsApplication.processingRegistry().removeProvider(self.provider)
```

3. `processing_provider` フォルダを作ってそこに次の3つのファイルを納めることもできます。

- 白紙の `__init__.py` ファイル。このファイルは妥当な Python パッケージを作るために必要です。
- `provider.py` ファイルはプロセッシングプロバイダを生成しあなたのアルゴリズムを外部から使えるようにします。

```
1 from qgis.core import QgsProcessingProvider
2
3 from processing_provider.example_processing_algorithm import _
4 ↪ExampleProcessingAlgorithm
5
6 class Provider(QgsProcessingProvider):
7
8     def loadAlgorithms(self, *args, **kwargs):
9         self.addAlgorithm(ExampleProcessingAlgorithm())
10        # add additional algorithms here
11        # self.addAlgorithm(MyOtherAlgorithm())
12
```

(次のページに続く)

(前のページからの続き)

```

13     def id(self, *args, **kwargs):
14         """The ID of your plugin, used for identifying the provider.
15
16         This string should be a unique, short, character only string,
17         eg "qgis" or "gdal". This string should not be localised.
18         """
19         return 'yourplugin'
20
21     def name(self, *args, **kwargs):
22         """The human friendly name of your plugin in Processing.
23
24         This string should be as short as possible (e.g. "Lastools", not
25         "Lastools version 1.0.1 64-bit") and localised.
26         """
27         return self.tr('Your plugin')
28
29     def icon(self):
30         """Should return a QIcon which is used for your provider inside
31         the Processing toolbox.
32         """
33         return QgsProcessingProvider.icon(self)

```

- example_processing_algorithm.py ファイルはサンプルアルゴリズムを含みます。 script template file の内容をコピー&ペーストして、自分の必要に合わせて修正してください。

4. ここまできたら QGIS でプラグインをリロードすれば、プロセッシングツールボックスとモデラーの中にあなたのスクリプトを見つけることができます。

The code snippets on this page need the following imports if you're outside the pyqgis console:

```

from qgis.core import (
    QgsVectorLayer,
    QgsPointXY,
)

```


第 18 章

ネットワーク分析ライブラリ

- 一般情報
- グラフを構築する
- グラフ分析
 - 最短経路を見つける
 - 利用可能領域

The network analysis library can be used to:

- create mathematical graph from geographical data (polyline vector layers)
- implement basic methods from graph theory (currently only Dijkstra's algorithm)

ネットワーク解析ライブラリは RoadGraph コアプラグインから基本機能をエクスポートすることによって作成されました。今はそのメソッドをプラグインで、または Python のコンソールから直接使用できます。

18.1 一般情報

手短に言えば、一般的なユースケースは、次のように記述できます。

1. 地理データから地理的なグラフ（たいていはポリラインベクターレイヤー）を作成する
2. グラフ分析の実行
3. 分析結果の利用（例えば、これらの可視化）

18.2 グラフを構築する

最初にする必要がある事は---入力データを準備することです、つまりベクターレイヤーをグラフに変換することです。これからのすべての操作は、レイヤーではなく、このグラフを使用します。

ソースとしてどんなポリラインベクターレイヤーも使用できます。ポリラインの頂点は、グラフの頂点となり、ポリラインのセグメントは、グラフの辺です。いくつかのノードが同じ座標を持っている場合、それらは同じグラフの頂点です。だから共通のノードを持つ2つの線は接続しています。

さらに、グラフの作成時には、入力ベクターレイヤーに好きな数だけ追加の点を「固定する」(「結びつける」)ことが可能です。追加の点それぞれについて、対応箇所---最も近いグラフの頂点または最も近いグラフの辺、が探し出されます。後者の場合、辺は分割されて新しい頂点が追加されるでしょう。

ベクターレイヤー属性と辺の長さは、辺の性質として使用できます。

Converting from a vector layer to the graph is done using the [Builder](#) programming pattern. A graph is constructed using a so-called Director. There is only one Director for now: [QgsVectorLayerDirector](#). The director sets the basic settings that will be used to construct a graph from a line vector layer, used by the builder to create the graph. Currently, as in the case with the director, only one builder exists: [QgsGraphBuilder](#), that creates [QgsGraph](#) objects. You may want to implement your own builders that will build a graphs compatible with such libraries as [BGL](#) or [NetworkX](#).

To calculate edge properties the programming pattern [strategy](#) is used. For now only [QgsNetworkDistanceStrategy](#) strategy (that takes into account the length of the route) and [QgsNetworkSpeedStrategy](#) (that also considers the speed) are available. You can implement your own strategy that will use all necessary parameters. For example, [RoadGraph](#) plugin uses a strategy that computes travel time using edge length and speed value from attributes.

では手順に行きましょう。

First of all, to use this library we should import the analysis module

```
from qgis.analysis import *
```

それからディレクターを作成するためのいくつかの例

```
1 # don't use information about road direction from layer attributes,
2 # all roads are treated as two-way
3 director = QgsVectorLayerDirector(vectorLayer, -1, '', '', '',
4     ↳QgsVectorLayerDirector.DirectionBoth)
5
6 # use field with index 5 as source of information about road direction.
7 # one-way roads with direct direction have attribute value "yes",
8 # one-way roads with reverse direction have the value "1", and accordingly
9 # bidirectional roads have "no". By default roads are treated as two-way.
10 director = QgsVectorLayerDirector(vectorLayer, 5, 'yes', '1', 'no',
    ↳QgsVectorLayerDirector.DirectionBoth)
```

ディレクターを構築するために、ベクターレイヤーを渡さなければなりません。これはグラフ構造および各道路セグメント上で許される移動(一方向または双方向の動き、順または逆の方向)についての情報のソースと

して使用されるでしょう。呼び出しは次のようになります

```

1 director = QgsVectorLayerDirector(vectorLayer,
2                                     directionFieldId,
3                                     directDirectionValue,
4                                     reverseDirectionValue,
5                                     bothDirectionValue,
6                                     defaultDirection)

```

そして、ここでこれらのパラメーターは何を意味するかの完全なリストは以下のとおりです。

- `vectorLayer` --- vector layer used to build the graph
- `directionFieldId` ---道路の方向に関する情報が格納されている属性テーブルのフィールドのインデックス。 -1、その後、まったくこの情報を使用しない場合。整数。
- `directDirectionValue` ---順方向（線の最初の点から最後の点へ移動）の道に対するフィールド値。文字列。
- `reverseDirectionValue` ---逆方向（線の最後の点から最初の点へ移動）の道に対するフィールド値。文字列。
- `bothDirectionValue` ---双方向道路に対するフィールド値（例えば最初の点から最後まで、また最後までから最初まで移動できる道に対する）。文字列。
- `defaultDirection` --- default road direction. This value will be used for those roads where field `directionFieldId` is not set or has some value different from any of the three values specified above. Possible values are:
 - `QgsVectorLayerDirector.DirectionForward` --- One-way direct
 - `QgsVectorLayerDirector.DirectionBackward` --- One-way reverse
 - `QgsVectorLayerDirector.DirectionBoth` --- Two-way

それから、辺性質を計算するための戦略を作成することが必要です

```

1 # The index of the field that contains information about the edge speed
2 attributeId = 1
3 # Default speed value
4 defaultValue = 50
5 # Conversion from speed to metric units ('1' means no conversion)
6 toMetricFactor = 1
7 strategy = QgsNetworkSpeedStrategy(attributeId, defaultValue, toMetricFactor)

```

そして、ディレクターにこの戦略について教えます

```

director = QgsVectorLayerDirector(vectorLayer, -1, '', '', '', 3)
director.addStrategy(strategy)

```

Now we can use the builder, which will create the graph. The `QgsGraphBuilder` class constructor takes several arguments:

- `crs` --- coordinate reference system to use. Mandatory argument.
- `otfEnabled` --- use "on the fly" reprojection or no. By default `const:True` (use OTF).
- `topologyTolerance` --- topological tolerance. Default value is 0.
- `ellipsoidID` --- ellipsoid to use. By default "WGS84".

```
# only CRS is set, all other values are defaults
builder = QgsGraphBuilder(vectorLayer.crs())
```

分析に使用されるいくつかのポイントを定義することもできます。例えば

```
startPoint = QgsPointXY(1179720.1871, 5419067.3507)
endPoint = QgsPointXY(1180616.0205, 5419745.7839)
```

これですべてが整いましたので、グラフを構築し、それにこれらの点を「結びつける」ことができます。

```
tiedPoints = director.makeGraph(builder, [startPoint, endPoint])
```

グラフを構築するには（レイヤー中の地物数とレイヤーのサイズに応じて）いくらか時間がかかることがあります。 `tiedPoints` は「結びつけられた」点の座標を持つリストです。ビルド操作が完了するとグラフが得られ、それを分析のために使用できます

```
graph = builder.graph()
```

次のコードで、点の頂点インデックスを取得できます

```
startId = graph.findVertex(tiedPoints[0])
endId = graph.findVertex(tiedPoints[1])
```

18.3 グラフ分析

ネットワーク分析はこの二つの質問に対する答えを見つけるために使用されます：どの頂点が接続されているか、どのように最短経路を検索するか。これらの問題を解決するため、ネットワーク解析ライブラリではダイクストラのアルゴリズムを提供しています。

ダイクストラ法は、グラフの1つの頂点から他のすべての頂点への最短ルートおよび最適化パラメーターの値を見つけます。結果は、最短経路木として表現できます。

The shortest path tree is a directed weighted graph (or more precisely a tree) with the following properties:

- 流入する辺がない頂点が1つだけあります - 木の根
- 他のすべての頂点には流入する辺が1つだけあります
- 頂点 B が頂点 A から到達可能である場合、このグラフ上の A から B への経路は、単一の利用可能な経路であり、それは最適（最短）です

To get the shortest path tree use the methods `shortestTree` and `dijkstra` of the `QgsGraphAnalyzer` class. It is recommended to use the `dijkstra` method because it works faster and uses memory more efficiently.

The `shortestTree` method is useful when you want to walk around the shortest path tree. It always creates a new graph object (`QgsGraph`) and accepts three variables:

- `source` --- input graph
- `startVertexIdx` --- index of the point on the tree (the root of the tree)
- `criterionNum` --- number of edge property to use (started from 0).

```
tree = QgsGraphAnalyzer.shortestTree(graph, startId, 0)
```

The `dijkstra` method has the same arguments, but returns two arrays. In the first array element `n` contains index of the incoming edge or -1 if there are no incoming edges. In the second array element `n` contains the distance from the root of the tree to vertex `n` or `DOUBLE_MAX` if vertex `n` is unreachable from the root.

```
(tree, cost) = QgsGraphAnalyzer.dijkstra(graph, startId, 0)
```

Here is some very simple code to display the shortest path tree using the graph created with the `shortestTree` method (select linestring layer in *Layers* panel and replace coordinates with your own).

警告: Use this code only as an example, it creates a lot of `QgsRubberBand` objects and may be slow on large datasets.

```
1 from qgis.core import *
2 from qgis.gui import *
3 from qgis.analysis import *
4 from qgis.PyQt.QtCore import *
5 from qgis.PyQt.QtGui import *
6
7 vectorLayer = QgsVectorLayer('testdata/network.gpkg|layername=network_lines',
8 ↪ 'lines')
9 director = QgsVectorLayerDirector(vectorLayer, -1, '', '', '',
10 ↪ QgsVectorLayerDirector.DirectionBoth)
11 strategy = QgsNetworkDistanceStrategy()
12 director.addStrategy(strategy)
13 builder = QgsGraphBuilder(vectorLayer.crs())
14
15 pStart = QgsPointXY(1179661.925139, 5419188.074362)
16 tiedPoint = director.makeGraph(builder, [pStart])
17 pStart = tiedPoint[0]
18
19 graph = builder.graph()
20
21 idStart = graph.findVertex(pStart)
22
23 tree = QgsGraphAnalyzer.shortestTree(graph, idStart, 0)
```

(次のページに続く)

```
22
23 i = 0
24 while (i < tree.edgeCount()):
25     rb = QgsRubberBand(iface.mapCanvas())
26     rb.setColor (Qt.red)
27     rb.addPoint (tree.vertex(tree.edge(i).fromVertex()).point())
28     rb.addPoint (tree.vertex(tree.edge(i).toVertex()).point())
29     i = i + 1
```

Same thing but using the `dijkstra` method

```
1 from qgis.core import *
2 from qgis.gui import *
3 from qgis.analysis import *
4 from qgis.PyQt.QtCore import *
5 from qgis.PyQt.QtGui import *
6
7 vectorLayer = QgsVectorLayer('testdata/network.gpkg|layername=network_lines',
8 ↪ 'lines')
9
10 director = QgsVectorLayerDirector(vectorLayer, -1, '', '', '',
11 ↪ QgsVectorLayerDirector.DirectionBoth)
12 strategy = QgsNetworkDistanceStrategy()
13 director.addStrategy(strategy)
14 builder = QgsGraphBuilder(vectorLayer.crs())
15
16 pStart = QgsPointXY(1179661.925139, 5419188.074362)
17 tiedPoint = director.makeGraph(builder, [pStart])
18 pStart = tiedPoint[0]
19
20 graph = builder.graph()
21
22 idStart = graph.findVertex(pStart)
23
24 (tree, costs) = QgsGraphAnalyzer.dijkstra(graph, idStart, 0)
25
26 for edgeId in tree:
27     if edgeId == -1:
28         continue
29     rb = QgsRubberBand(iface.mapCanvas())
30     rb.setColor (Qt.red)
31     rb.addPoint (graph.vertex(graph.edge(edgeId).fromVertex()).point())
32     rb.addPoint (graph.vertex(graph.edge(edgeId).toVertex()).point())
```

18.3.1 最短経路を見つける

To find the optimal path between two points the following approach is used. Both points (start A and end B) are "tied" to the graph when it is built. Then using the `shortestTree` or `dijkstra` method we build the shortest path tree with root in the start point A. In the same tree we also find the end point B and start to walk through the tree from point B to point A. The whole algorithm can be written as:

```

1 assign T = B
2 while T != B
3     add point T to path
4     get incoming edge for point T
5     look for point TT, that is start point of this edge
6     assign T = TT
7 add point A to path

```

この時点において、この経路で走行中に訪問される頂点の反転リストの形（頂点は逆順で終点から始点へと列挙されている）で、経路が得られます。

Here is the sample code for QGIS Python Console (you may need to load and select a linestring layer in TOC and replace coordinates in the code with yours) that uses the `shortestTree` method

```

1 from qgis.core import *
2 from qgis.gui import *
3 from qgis.analysis import *
4
5 from qgis.PyQt.QtCore import *
6 from qgis.PyQt.QtGui import *
7
8 vectorLayer = QgsVectorLayer('testdata/network.gpkg|layername=network_lines',
9 ↪ 'lines')
10 builder = QgsGraphBuilder(vectorLayer.sourceCrs())
11 director = QgsVectorLayerDirector(vectorLayer, -1, '|', '|', '|',
12 ↪ QgsVectorLayerDirector.DirectionBoth)
13
14 startPoint = QgsPointXY(1179661.925139, 5419188.074362)
15 endPoint = QgsPointXY(1180942.970617, 5420040.097560)
16
17 tiedPoints = director.makeGraph(builder, [startPoint, endPoint])
18 tStart, tStop = tiedPoints
19
20 graph = builder.graph()
21 idxStart = graph.findVertex(tStart)
22
23 tree = QgsGraphAnalyzer.shortestTree(graph, idxStart, 0)
24
25 idxStart = tree.findVertex(tStart)
26 idxEnd = tree.findVertex(tStop)
27
28 if idxEnd == -1:
29     raise Exception('No route!')

```

(次のページに続く)

(前のページからの続き)

```

29 # Add last point
30 route = [tree.vertex(idxEnd).point()]
31
32 # Iterate the graph
33 while idxEnd != idxStart:
34     edgeIds = tree.vertex(idxEnd).incomingEdges()
35     if len(edgeIds) == 0:
36         break
37     edge = tree.edge(edgeIds[0])
38     route.insert(0, tree.vertex(edge.fromVertex()).point())
39     idxEnd = edge.fromVertex()
40
41 # Display
42 rb = QgsRubberBand(iface.mapCanvas())
43 rb.setColor(Qt.green)
44
45 # This may require coordinate transformation if project's CRS
46 # is different than layer's CRS
47 for p in route:
48     rb.addPoint(p)

```

And here is the same sample but using the `dijkstra` method

```

1 from qgis.core import *
2 from qgis.gui import *
3 from qgis.analysis import *
4
5 from qgis.PyQt.QtCore import *
6 from qgis.PyQt.QtGui import *
7
8 vectorLayer = QgsVectorLayer('testdata/network.gpkg|layername=network_lines',
9 ↪ 'lines')
10 director = QgsVectorLayerDirector(vectorLayer, -1, '', '', '',
11 ↪ QgsVectorLayerDirector.DirectionBoth)
12 strategy = QgsNetworkDistanceStrategy()
13 director.addStrategy(strategy)
14
15 builder = QgsGraphBuilder(vectorLayer.sourceCrs())
16
17 startPoint = QgsPointXY(1179661.925139, 5419188.074362)
18 endPoint = QgsPointXY(1180942.970617, 5420040.097560)
19
20 tiedPoints = director.makeGraph(builder, [startPoint, endPoint])
21 tStart, tStop = tiedPoints
22
23 graph = builder.graph()
24 idxStart = graph.findVertex(tStart)
25 idxEnd = graph.findVertex(tStop)
26
27 (tree, costs) = QgsGraphAnalyzer.dijkstra(graph, idxStart, 0)

```

(次のページに続く)

(前のページからの続き)

```

27 if tree[idxEnd] == -1:
28     raise Exception('No route!')
29
30 # Total cost
31 cost = costs[idxEnd]
32
33 # Add last point
34 route = [graph.vertex(idxEnd).point()]
35
36 # Iterate the graph
37 while idxEnd != idxStart:
38     idxEnd = graph.edge(tree[idxEnd]).fromVertex()
39     route.insert(0, graph.vertex(idxEnd).point())
40
41 # Display
42 rb = QgsRubberBand(iface.mapCanvas())
43 rb.setColor(Qt.red)
44
45 # This may require coordinate transformation if project's CRS
46 # is different than layer's CRS
47 for p in route:
48     rb.addPoint(p)

```

18.3.2 利用可能領域

頂点 A に対する利用可能領域とは、頂点 A から到達可能であり、A からこれらの頂点までの経路のコストがある値以下になるような、グラフの頂点の部分集合です。

より明確に、これは次の例で示すことができます。「消防署があります。消防車が 5 分、10 分、15 分で到達できるのは市内のどの部分ですか？」。これらの質問への回答が消防署の利用可能領域です。

To find the areas of availability we can use the `dijkstra` method of the `QgsGraphAnalyzer` class. It is enough to compare the elements of the cost array with a predefined value. If `cost[i]` is less than or equal to a predefined value, then vertex `i` is inside the area of availability, otherwise it is outside.

より難しい問題は、利用可能領域の境界を取得することです。下限はまだ到達できる頂点の集合であり、上限は到達できない頂点の集合です。実際にはこれは簡単です：それは、辺の元頂点が到達可能であり辺の先頂点が到達可能ではないような、最短経路木の辺に基づく利用可能境界です。

例です

```

1 director = QgsVectorLayerDirector(vectorLayer, -1, '|', '|', '|',
  ↳QgsVectorLayerDirector.DirectionBoth)
2 strategy = QgsNetworkDistanceStrategy()
3 director.addStrategy(strategy)
4 builder = QgsGraphBuilder(vectorLayer.crs())
5
6
7 pStart = QgsPointXY(1179661.925139, 5419188.074362)

```

(次のページに続く)

```
8 delta = iface.mapCanvas().getCoordinateTransform().mapUnitsPerPixel() * 1
9
10 rb = QgsRubberBand(iface.mapCanvas(), True)
11 rb.setColor(Qt.green)
12 rb.addPoint(QgsPointXY(pStart.x() - delta, pStart.y() - delta))
13 rb.addPoint(QgsPointXY(pStart.x() + delta, pStart.y() - delta))
14 rb.addPoint(QgsPointXY(pStart.x() + delta, pStart.y() + delta))
15 rb.addPoint(QgsPointXY(pStart.x() - delta, pStart.y() + delta))
16
17 tiedPoints = director.makeGraph(builder, [pStart])
18 graph = builder.graph()
19 tStart = tiedPoints[0]
20
21 idStart = graph.findVertex(tStart)
22
23 (tree, cost) = QgsGraphAnalyzer.dijkstra(graph, idStart, 0)
24
25 upperBound = []
26 r = 1500.0
27 i = 0
28 tree.reverse()
29
30 while i < len(cost):
31     if cost[i] > r and tree[i] != -1:
32         outVertexId = graph.edge(tree[i]).toVertex()
33         if cost[outVertexId] < r:
34             upperBound.append(i)
35         i = i + 1
36
37 for i in upperBound:
38     centerPoint = graph.vertex(i).point()
39     rb = QgsRubberBand(iface.mapCanvas(), True)
40     rb.setColor(Qt.red)
41     rb.addPoint(QgsPointXY(centerPoint.x() - delta, centerPoint.y() - delta))
42     rb.addPoint(QgsPointXY(centerPoint.x() + delta, centerPoint.y() - delta))
43     rb.addPoint(QgsPointXY(centerPoint.x() + delta, centerPoint.y() + delta))
44     rb.addPoint(QgsPointXY(centerPoint.x() - delta, centerPoint.y() + delta))
```

第 19 章

QGIS Server and Python

19.1 はじめに

QGIS Server is three different things:

1. QGIS Server library: a library that provides an API for creating OGC web services
2. QGIS Server FCGI: a FCGI binary application `qgis_maserv.fcgi` that together with a web server implements a set of OGC services (WMS, WFS, WCS etc.) and OGC APIs (WFS3/OAPIF)
3. QGIS Development Server: a development server binary application `qgis_mapserver` that implements a set of OGC services (WMS, WFS, WCS etc.) and OGC APIs (WFS3/OAPIF)

This chapter of the cookbook focuses on the first topic and by explaining the usage of QGIS Server API it shows how it is possible to use Python to extend, enhance or customize the server behavior or how to use the QGIS Server API to embed QGIS server into another application.

There are a few different ways you can alter the behavior of QGIS Server or extend its capabilities to offer new custom services or APIs, these are the main scenarios you may face:

- EMBEDDING → Use QGIS Server API from another Python application
- STANDALONE → Run QGIS Server as a standalone WSGI/HTTP service
- FILTERS → Enhance/Customize QGIS Server with filter plugins
- SERVICES → Add a new *SERVICE*
- OGC APIs → Add a new *OGC API*

Embedding and standalone applications require using the QGIS Server Python API directly from another Python script or application while the remaining options are better suited for when you want to add custom features to a standard QGIS Server binary application (FCGI or development server): in this case you'll need to write a Python plugin for the server application and register your custom filters, services or APIs.

19.2 Server API basics

The fundamental classes involved in a typical QGIS Server application are:

- `QgsServer` the server instance (typically a single instance for the whole application life)
- `QgsServerRequest` the request object (typically recreated on each request)
- `QgsServerResponse` the response object (typically recreated on each request)
- `QgsServer.handleRequest(request, response)` processes the request and populates the response

The QGIS Server FCGI or development server workflow can be summarized as follows:

```
1 initialize the QgsApplication
2 create the QgsServer
3 the main server loop waits forever for client requests:
4     for each incoming request:
5         create a QgsServerRequest request
6         create a QgsServerResponse response
7         call QgsServer.handleRequest(request, response)
8             filter plugins may be executed
9         send the output to the client
```

Inside the `QgsServer.handleRequest(request, response)` method the filter plugins callbacks are called and `QgsServerRequest` and `QgsServerResponse` are made available to the plugins through the `QgsServerInterface`.

警告: QGIS server classes are not thread safe, you should always use a multiprocessing model or containers when building scalable applications based on QGIS Server API.

19.3 Standalone or embedding

For standalone server applications or embedding, you will need to use the above mentioned server classes directly, wrapping them up into a web server implementation that manages all the HTTP protocol interactions with the client.

A minimal example of the QGIS Server API usage (without the HTTP part) follows:

```
1 from qgis.core import QgsApplication
2 from qgis.server import *
3 app = QgsApplication([], False)
4
5 # Create the server instance, it may be a single one that
6 # is reused on multiple requests
7 server = QgsServer()
```

(次のページに続く)

(前のページからの続き)

```

8
9 # Create the request by specifying the full URL and an optional body
10 # (for example for POST requests)
11 request = QgsBufferServerRequest (
12     'http://localhost:8081/?MAP=/qgis-server/projects/helloworld.qgs' +
13     '&SERVICE=WMS&REQUEST=GetCapabilities')
14
15 # Create a response objects
16 response = QgsBufferServerResponse ()
17
18 # Handle the request
19 server.handleRequest (request, response)
20
21 print (response.headers ())
22 print (response.body () .data () .decode ('utf8'))
23
24 app.exitQgis ()

```

Here is a complete standalone application example developed for the continuous integrations testing on QGIS source code repository, it showcases a wide set of different plugin filters and authentication schemes (not mean for production because they were developed for testing purposes only but still interesting for learning):

https://github.com/qgis/QGIS/blob/master/tests/src/python/qgis_wrapped_server.py

19.4 Server plugins

Server python plugins are loaded once when the QGIS Server application starts and can be used to register filters, services or APIs.

The structure of a server plugin is very similar to their desktop counterpart, a `QgsServerInterface` object is made available to the plugins and the plugins can register one or more custom filters, services or APIs to the corresponding registry by using one of the methods exposed by the server interface.

19.4.1 Server filter plugins

Filters come in three different flavors and they can be instantiated by subclassing one of the classes below and by calling the corresponding method of `QgsServerInterface`:

| Filter Type | Base Class | QgsServerInterface registration |
|----------------|-------------------------------------|------------------------------------|
| I/O | <code>QgsServerFilter</code> | <code>registerFilter</code> |
| Access Control | <code>QgsAccessControlFilter</code> | <code>registerAccessControl</code> |
| Cache | <code>QgsServerCacheFilter</code> | <code>registerServerCache</code> |

I/O filters

I/O filters can modify the server input and output (the request and the response) of the core services (WMS, WFS etc.) allowing to do any kind of manipulation of the services workflow, it is possible for example to restrict the access to selected layers, to inject an XSL stylesheet to the XML response, to add a watermark to a generated WMS image and so on.

From this point, you might find useful a quick look to the [server plugins API docs](#).

Each filter should implement at least one of three callbacks:

- `requestReady()`
- `responseComplete()`
- `sendResponse()`

All filters have access to the request/response object (`QgsRequestHandler`) and can manipulate all its properties (input/output) and raise exceptions (while in a quite particular way as we 'll see below).

Here is the pseudo code showing how the server handles a typical request and when the filter 's callbacks are called:

```
1 for each incoming request:
2     create GET/POST request handler
3     pass request to an instance of QgsServerInterface
4     call requestReady filters
5     if there is not a response:
6         if SERVICE is WMS/WFS/WCS:
7             create WMS/WFS/WCS service
8             call service 's executeRequest
9                 possibly call sendResponse for each chunk of bytes
10                sent to the client by a streaming services (WFS)
11         call responseComplete
12         call sendResponse
13     request handler sends the response to the client
```

次の段落では、利用可能なコールバックを詳細に説明します。

requestReady

要求の準備ができたときに呼び出されます。受信 URL とデータが解析され、コアサービス (WMS、WFS など) スイッチに入る前に、これは入力进行操作するなどのアクションを実行できるポイントです。

- 認証/認可
- リダイレクト
- 特定のパラメーター (例えば、型名) を追加 / 除去
- 例外を発生させる

SERVICE パラメーターを変更することでコアサービスを完全に置き換え、それによりコアサービスを完全にバイパスすることさえできるかもしれません (とはいえ、これはあまり意味がないということ)。

sendResponse

This is called whenever any output is sent to **FCGI** `stdout` (and from there, to the client), this is normally done after core services have finished their process and after `responseComplete` hook was called, but in a few cases XML can become so huge that a streaming XML implementation was needed (WFS `GetFeature` is one of them), in this case, `sendResponse` is called multiple times before the response is complete (and before `responseComplete` is called). The obvious consequence is that `sendResponse` is normally called once but might be exceptionally called multiple times and in that case (and only in that case) it is also called before `responseComplete`.

`sendResponse` is the best place for direct manipulation of core service's output and while `responseComplete` is typically also an option, `sendResponse` is the only viable option in case of streaming services.

responseComplete

This is called once when core services (if hit) finish their process and the request is ready to be sent to the client. As discussed above, this is normally called before `sendResponse` except for streaming services (or other plugin filters) that might have called `sendResponse` earlier.

`responseComplete` is the ideal place to provide new services implementation (WPS or custom services) and to perform direct manipulation of the output coming from core services (for example to add a watermark upon a WMS image).

Raising exceptions from a plugin

Some work has still to be done on this topic: the current implementation can distinguish between handled and unhandled exceptions by setting a `QgsRequestHandler` property to an instance of `QgsMapServiceException`, this way the main C++ code can catch handled python exceptions and ignore unhandled exceptions (or better: log them).

このアプローチは、基本的に動作しますが、それは非常に「パイソンの」ではありません：より良いアプローチは、Python コードから例外を発生し、それらがそこで処理されるために C++ ループに湧き上がるのを見ることでしょう。

サーバー・プラグインを書く

A server plugin is a standard QGIS Python plugin as described in *Python プラグインを開発する*, that just provides an additional (or alternative) interface: a typical QGIS desktop plugin has access to QGIS application through the `QgisInterface` instance, a server plugin has only access to a `QgsServerInterface` when it is executed within the QGIS Server application context.

To make QGIS Server aware that a plugin has a server interface, a special metadata entry is needed (in *metadata.txt*)

```
server=True
```

重要: Only plugins that have the `server=True` metadata set will be loaded and executed by QGIS Server.

The example plugin discussed here (with many more) is available on github at <https://github.com/elpaso/qgis3-server-vagrant/tree/master/resources/web/plugins>, a few server plugins are also published in the official QGIS plugins repository.

プラグインファイル

私たちの例のサーバー・プラグインのディレクトリ構造はこちらです

```
1 PYTHON_PLUGINS_PATH/  
2   HelloServer/  
3     __init__.py    --> *required*  
4     HelloServer.py --> *required*  
5     metadata.txt  --> *required*
```

`__init__.py`

This file is required by Python's import system. Also, QGIS Server requires that this file contains a `serverClassFactory()` function, which is called when the plugin gets loaded into QGIS Server when the server starts. It receives reference to instance of `QgsServerInterface` and must return instance of your plugin's class. This is how the example plugin `__init__.py` looks like

```
def serverClassFactory(serverIface):  
    from .HelloServer import HelloServerServer  
    return HelloServerServer(serverIface)
```


HelloServer.py

魔法が起こると、これは魔法がどのように見えるかであるところである : (例 `HelloServer.py`)

A server plugin typically consists in one or more callbacks packed into instances of a `QgsServerFilter`.

Each `QgsServerFilter` implements one or more of the following callbacks:

- `requestReady()`
- `responseComplete()`
- `sendResponse()`

The following example implements a minimal filter which prints *HelloServer!* in case the **SERVICE** parameter equals to " HELLO "

```

1 class HelloFilter(QgsServerFilter):
2
3     def __init__(self, serverIface):
4         super().__init__(serverIface)
5
6     def requestReady(self):
7         QgsMessageLog.logMessage("HelloFilter.requestReady")
8
9     def sendResponse(self):
10        QgsMessageLog.logMessage("HelloFilter.sendResponse")
11
12    def responseComplete(self):
13        QgsMessageLog.logMessage("HelloFilter.responseComplete")
14        request = self.serverInterface().requestHandler()
15        params = request.parameterMap()
16        if params.get('SERVICE', '').upper() == 'HELLO':
17            request.clear()
18            request.setResponseHeader('Content-type', 'text/plain')
19            # Note that the content is of type "bytes"
20            request.appendBody(b'HelloServer!')
```

The filters must be registered into the **serverIface** as in the following example:

```

class HelloServerServer:
    def __init__(self, serverIface):
        serverIface.registerFilter(HelloFilter(), 100)
```

The second parameter of `registerFilter` sets a priority which defines the order for the callbacks with the same name (the lower priority is invoked first).

By using the three callbacks, plugins can manipulate the input and/or the output of the server in many different ways. In every moment, the plugin instance has access to the `QgsRequestHandler` through the `QgsServerInterface`. The `QgsRequestHandler` class has plenty of methods that can be used to alter the input parameters before entering the core processing of the server (by using `requestReady()`) or after the request has been processed by the core services (by using `sendResponse()`).

次の例は、いくつかの一般的なユースケースをカバーします：

入力を変更する

The example plugin contains a test example that changes input parameters coming from the query string, in this example a new parameter is injected into the (already parsed) `parameterMap`, this parameter is then visible by core services (WMS etc.), at the end of core services processing we check that the parameter is still there:

```

1 class ParamsFilter(QgsServerFilter):
2
3     def __init__(self, serverIface):
4         super(ParamsFilter, self).__init__(serverIface)
5
6     def requestReady(self):
7         request = self.serverInterface().requestHandler()
8         params = request.parameterMap()
9         request.setParameter('TEST_NEW_PARAM', 'ParamsFilter')
10
11    def responseComplete(self):
12        request = self.serverInterface().requestHandler()
13        params = request.parameterMap()
14        if params.get('TEST_NEW_PARAM') == 'ParamsFilter':
15            QgsMessageLog.logMessage("SUCCESS - ParamsFilter.responseComplete")
16        else:
17            QgsMessageLog.logMessage("FAIL - ParamsFilter.responseComplete")

```

これは、ログファイルに見るものの抽出物である：

```

1 src/core/qgsmessagelog.cpp: 45: (logMessage) [0ms] 2014-12-12T12:39:29 plugin[0]
↳HelloServerServer - loading filter ParamsFilter
2 src/core/qgsmessagelog.cpp: 45: (logMessage) [1ms] 2014-12-12T12:39:29 Server[0]
↳Server plugin HelloServer loaded!
3 src/core/qgsmessagelog.cpp: 45: (logMessage) [0ms] 2014-12-12T12:39:29 Server[0]
↳Server python plugins loaded
4 src/mapserver/qgshttprequesthandler.cpp: 547: (requestStringToParameterMap) [1ms]
↳inserting pair SERVICE // HELLO into the parameter map
5 src/mapserver/qgsserverfilter.cpp: 42: (requestReady) [0ms] QgsServerFilter
↳plugin default requestReady called
6 src/core/qgsmessagelog.cpp: 45: (logMessage) [0ms] 2014-12-12T12:39:29 plugin[0]
↳SUCCESS - ParamsFilter.responseComplete

```

強調表示された行の「SUCCESS」の文字列は、プラグインがテストに合格したことを示しています。

同じ手法が、コアのサービスでなくカスタムサービスを利用するために利用できます：たとえば **WFS SERVICE** 要求または任意の他のコア要求を **SERVICE** パラメーターを別の何かに変更するだけでスキップできます、そしてコアサービスはスキップされ、それからカスタム結果を出力に注入してそれらをクライアントに送信できます（これはここで以下に説明される）。

ちなみに： If you really want to implement a custom service it is recommended to subclass `QgsService` and

register your service on `registerFilter` by calling its `registerService(service)`

出力を変更または置き換えする

透かしフィルタの例は、WMS コアサービスによって作成された WMS 画像の上に透かし画像を加算した新たな画像で WMS 出力を置き換える方法を示しています：

```

1 from qgis.server import *
2 from qgis.PyQt.QtCore import *
3 from qgis.PyQt.QtGui import *
4
5 class WatermarkFilter(QgsServerFilter):
6
7     def __init__(self, serverIface):
8         super().__init__(serverIface)
9
10    def responseComplete(self):
11        request = self.serverInterface().requestHandler()
12        params = request.parameterMap()
13        # Do some checks
14        if (params.get('SERVICE').upper() == 'WMS' \
15            and params.get('REQUEST').upper() == 'GETMAP' \
16            and not request.exceptionRaised()):
17            QgsMessageLog.logMessage("WatermarkFilter.responseComplete: image_
↳ready %s" % request.parameter("FORMAT"))
18            # Get the image
19            img = QImage()
20            img.loadFromData(request.body())
21            # Adds the watermark
22            watermark = QImage(os.path.join(os.path.dirname(__file__), 'media/
↳watermark.png'))
23            p = QPainter(img)
24            p.drawImage(QRect( 20, 20, 40, 40), watermark)
25            p.end()
26            ba = QByteArray()
27            buffer = QBuffer(ba)
28            buffer.open(QIODevice.WriteOnly)
29            img.save(buffer, "PNG" if "png" in request.parameter("FORMAT") else
↳"JPG")
30            # Set the body
31            request.clearBody()
32            request.appendBody(ba)

```

In this example the **SERVICE** parameter value is checked and if the incoming request is a **WMS GETMAP** and no exceptions have been set by a previously executed plugin or by the core service (WMS in this case), the WMS generated image is retrieved from the output buffer and the watermark image is added. The final step is to clear the output buffer and replace it with the newly generated image. Please note that in a real-world situation we should also check for the requested image type instead of supporting PNG or JPG only.

Access control filters

Access control filters gives the developer a fine-grained control over which layers, features and attributes can be accessed, the following callbacks can be implemented in an access control filter:

- `layerFilterExpression(layer)`
- `layerFilterSubsetString(layer)`
- `layerPermissions(layer)`
- `authorizedLayerAttributes(layer, attributes)`
- `allowToEdit(layer, feature)`
- `cacheKey()`

プラグインファイル

Here's the directory structure of our example plugin:

```

1 PYTHON_PLUGINS_PATH/
2   MyAccessControl/
3     __init__.py    --> *required*
4     AccessControl.py --> *required*
5     metadata.txt  --> *required*
```

`__init__.py`

This file is required by Python's import system. As for all QGIS server plugins, this file contains a `serverClassFactory()` function, which is called when the plugin gets loaded into QGIS Server at startup. It receives a reference to an instance of `QgsServerInterface` and must return an instance of your plugin's class. This is how the example plugin `__init__.py` looks like:

```

def serverClassFactory(serverIface):
    from MyAccessControl.AccessControl import AccessControlServer
    return AccessControlServer(serverIface)
```

`AccessControl.py`

```

1 class AccessControlFilter(QgsAccessControlFilter):
2
3     def __init__(self, server_iface):
4         super().__init__(server_iface)
5
6     def layerFilterExpression(self, layer):
7         """ Return an additional expression filter """
8         return super().layerFilterExpression(layer)
9
```

(次のページに続く)

(前のページからの続き)

```

10 def layerFilterSubsetString(self, layer):
11     """ Return an additional subset string (typically SQL) filter """
12     return super().layerFilterSubsetString(layer)
13
14 def layerPermissions(self, layer):
15     """ Return the layer rights """
16     return super().layerPermissions(layer)
17
18 def authorizedLayerAttributes(self, layer, attributes):
19     """ Return the authorised layer attributes """
20     return super().authorizedLayerAttributes(layer, attributes)
21
22 def allowToEdit(self, layer, feature):
23     """ Are we authorise to modify the following geometry """
24     return super().allowToEdit(layer, feature)
25
26 def cacheKey(self):
27     return super().cacheKey()
28
29 class AccessControlServer:
30
31     def __init__(self, serverIface):
32         """ Register AccessControlFilter """
33         serverIface.registerAccessControl(AccessControlFilter(self.serverIface), 100)

```

この例では全員に完全なアクセス権を与えています。

誰がログオンしているかを知るのはこのプラグインの役割です。

これらすべての方法で私達は、レイヤーごとの制限をカスタマイズできるようにするには、引数のレイヤーを持っています。

layerFilterExpression

結果を制限するために式を追加するために使用し、例えば：

```

def layerFilterExpression(self, layer):
    return "$role = 'user'"

```

属性の役割が「ユーザー」に等しい地物に制限するため。

layerFilterSubsetString

以前よりも同じですが、(データベース内で実行) SubsetString を使用

```
def layerFilterSubsetString(self, layer):  
    return "role = 'user'"
```

属性の役割が「ユーザー」に等しい地物に制限するため。

layerPermissions

レイヤーへのアクセスを制限します。

Return an object of type `LayerPermissions`, which has the properties:

- `canRead` to see it in the `GetCapabilities` and have read access.
- `canInsert` to be able to insert a new feature.
- `canUpdate` to be able to update a feature.
- `canDelete` to be able to delete a feature.

例:

```
1 def layerPermissions(self, layer):  
2     rights = QgsAccessControlFilter.LayerPermissions()  
3     rights.canRead = True  
4     rights.canInsert = rights.canUpdate = rights.canDelete = False  
5     return rights
```

読み取り専用のアクセスのすべてを制限します。

authorizedLayerAttributes

属性の特定のサブセットの可視性を制限するために使用します。

引数の属性が表示属性の現在のセットを返します。

例:

```
def authorizedLayerAttributes(self, layer, attributes):  
    return [a for a in attributes if a != "role"]
```

「role」属性を非表示にします。

allowToEdit

これは、地物のサブセットに編集を制限するために使用されます。

これは、WFS-Transaction プロトコルで使用されています。

例:

```
def allowToEdit(self, layer, feature):
    return feature.attribute('role') == 'user'
```

値「user」の属性「role」を持つ地物だけを編集できます。

cacheKey

QGIS サーバーは、このメソッド中に役割を返すことができる役割ごとにキャッシュを持っている能力のキャッシュを維持します。または None を返し、完全にキャッシュを無効にします。

19.4.2 Custom services

In QGIS Server, core services such as WMS, WFS and WCS are implemented as subclasses of `QgsService`.

To implement a new service that will be executed when the query string parameter `SERVICE` matches the service name, you can implement your own `QgsService` and register your service on the `serviceRegistry` by calling its `registerService(service)`.

Here is an example of a custom service named CUSTOM:

```
1 from qgis.server import QgsService
2 from qgis.core import QgsMessageLog
3
4 class CustomServiceService(QgsService):
5
6     def __init__(self):
7         QgsService.__init__(self)
8
9     def name(self):
10        return "CUSTOM"
11
12    def version(self):
13        return "1.0.0"
14
15    def allowMethod(method):
16        return True
17
18    def executeRequest(self, request, response, project):
19        response.setStatuscode(200)
20        QgsMessageLog.logMessage('Custom service executeRequest')
21        response.write("Custom service executeRequest")
```

(次のページに続く)

```
22
23
24 class CustomService():
25
26     def __init__(self, serverIface):
27         serverIface.serviceRegistry().registerService(CustomServiceService())
```

19.4.3 Custom APIs

In QGIS Server, core OGC APIs such OAPIF (aka WFS3) are implemented as collections of `QgsServerOgcApiHandler` subclasses that are registered to an instance of `QgsServerOgcApi` (or it's parent class `QgsServerApi`).

To implemented a new API that will be executed when the url path matches a certain URL, you can implemented your own `QgsServerOgcApiHandler` instances, add them to an `QgsServerOgcApi` and register the API on the `serviceRegistry` by calling its `registerApi(api)`.

Here is an example of a custom API that will be executed when the URL contains `/customapi`:

```
1 import json
2 import os
3
4 from qgis.PyQt.QtCore import QBuffer, QIODevice, QTextStream, QRegularExpression
5 from qgis.server import (
6     QgsServiceRegistry,
7     QgsService,
8     QgsServerFilter,
9     QgsServerOgcApi,
10    QgsServerQueryStringParameter,
11    QgsServerOgcApiHandler,
12 )
13
14 from qgis.core import (
15     QgsMessageLog,
16     QgsJsonExporter,
17     QgsCircle,
18     QgsFeature,
19     QgsPoint,
20     QgsGeometry,
21 )
22
23
24 class CustomApiHandler(QgsServerOgcApiHandler):
25
26     def __init__(self):
27         super(CustomApiHandler, self).__init__()
28         self.setContentTypes([QgsServerOgcApi.HTML, QgsServerOgcApi.JSON])
29
30     def path(self):
```

(次のページに続く)


```

31     return QRegularExpression("/customapi")
32
33     def operationId(self):
34         return "CustomApiXYCircle"
35
36     def summary(self):
37         return "Creates a circle around a point"
38
39     def description(self):
40         return "Creates a circle around a point"
41
42     def linkTitle(self):
43         return "Custom Api XY Circle"
44
45     def linkType(self):
46         return QgsServerOgcApi.data
47
48     def handleRequest(self, context):
49         """Simple Circle"""
50
51         values = self.values(context)
52         x = values['x']
53         y = values['y']
54         r = values['r']
55         f = QgsFeature()
56         f.setAttributes([x, y, r])
57         f.setGeometry(QgsCircle(QgsPoint(x, y), r).toCircularString())
58         exporter = QgsJsonExporter()
59         self.write(json.loads(exporter.exportFeature(f)), context)
60
61     def templatePath(self, context):
62         # The template path is used to serve HTML content
63         return os.path.join(os.path.dirname(__file__), 'circle.html')
64
65     def parameters(self, context):
66         return [QgsServerQueryStringParameter('x', True,
↪QgsServerQueryStringParameter.Type.Double, 'X coordinate'),
67                 QgsServerQueryStringParameter(
68                     'y', True, QgsServerQueryStringParameter.Type.Double, 'Y_
↪coordinate'),
69                 QgsServerQueryStringParameter('r', True,
↪QgsServerQueryStringParameter.Type.Double, 'radius')]
70
71
72 class CustomApi():
73
74     def __init__(self, serverIface):
75         api = QgsServerOgcApi(serverIface, '/customapi',
76                               'custom api', 'a custom api', '1.1')
77         handler = CustomApiHandler()
78         api.registerHandler(handler)
79         serverIface.serviceRegistry().registerApi(api)

```

The code snippets on this page need the following imports if you're outside the pyqgis console:

```
1 from qgis.PyQt.QtCore import (  
2     QRectF,  
3 )  
4  
5 from qgis.core import (  
6     QgsProject,  
7     QgsLayerTreeModel,  
8 )  
9  
10 from qgis.gui import (  
11     QgsLayerTreeView,  
12 )
```

第 20 章

PyQGIS チートシート

20.1 ユーザーインターフェース

ルックアンドフィールの変更

```
1 from qgis.PyQt.QtWidgets import QApplication
2
3 app = QApplication.instance()
4 app.setStyleSheet(".QWidget {color: blue; background-color: yellow;}")
5 # You can even read the stylesheet from a file
6 with open("testdata/file.qss") as qss_file_content:
7     app.setStyleSheet(qss_file_content.read())
```

アイコンとタイトルの変更

```
1 from qgis.PyQt.QtGui import QIcon
2
3 icon = QIcon("/path/to/logo/file.png")
4 iface.mainWindow().setWindowIcon(icon)
5 iface.mainWindow().setWindowTitle("My QGIS")
```

20.2 設定

Get QgsSettings list

```
1 from qgis.core import QgsSettings
2
3 qs = QgsSettings()
4
5 for k in sorted(qs.allKeys()):
6     print(k)
```

20.3 ツールバー

Remove toolbar

```
1 toolbar = iface.helpToolBar()
2 parent = toolbar.parentWidget()
3 parent.removeToolBar(toolbar)
4
5 # and add again
6 parent.addToolBar(toolbar)
```

Remove actions toolbar

```
actions = iface.attributesToolBar().actions()
iface.attributesToolBar().clear()
iface.attributesToolBar().addAction(actions[4])
iface.attributesToolBar().addAction(actions[3])
```

20.4 メニュー

Remove menu

```
1 # for example Help Menu
2 menu = iface.helpMenu()
3 menubar = menu.parentWidget()
4 menubar.removeAction(menu.menuAction())
5
6 # and add again
7 menubar.addAction(menu.menuAction())
```

20.5 キャンバス

キャンバスにアクセスする

```
canvas = iface.mapCanvas()
```

キャンバスの色を変更する

```
from qgis.PyQt.QtCore import Qt

iface.mapCanvas().setCanvasColor(Qt.black)
iface.mapCanvas().refresh()
```

Map Update interval

```

from qgis.core import QgsSettings
# Set milliseconds (150 milliseconds)
QgsSettings().setValue("/qgis/map_update_interval", 150)

```

20.6 レイヤー

ベクターレイヤーを追加する

```

layer = iface.addVectorLayer("testdata/airports.shp", "layer name you like", "ogr")
if not layer or not layer.isValid():
    print("Layer failed to load!")

```

アクティブなレイヤーを取得する

```

layer = iface.activeLayer()

```

レイヤーをすべて一覧する

```

from qgis.core import QgsProject

QgsProject.instance().mapLayers().values()

```

レイヤー名を取得する

```

1 from qgis.core import QgsVectorLayer
2 layer = QgsVectorLayer("Point?crs=EPSG:4326", "layer name you like", "memory")
3 QgsProject.instance().addMapLayer(layer)
4
5 layers_names = []
6 for layer in QgsProject.instance().mapLayers().values():
7     layers_names.append(layer.name())
8
9 print("layers TOC = {}".format(layers_names))

```

```

layers TOC = ['layer name you like']

```

もしくは

```

layers_names = [layer.name() for layer in QgsProject.instance().mapLayers().
↳values()]
print("layers TOC = {}".format(layers_names))

```

```

layers TOC = ['layer name you like']

```

名前からレイヤーを見つける

```
from qgis.core import QgsProject

layer = QgsProject.instance().mapLayersByName("layer name you like")[0]
print(layer.name())
```

```
layer name you like
```

アクティブなレイヤーを設定する

```
from qgis.core import QgsProject

layer = QgsProject.instance().mapLayersByName("layer name you like")[0]
iface.setActiveLayer(layer)
```

間隔をおいてレイヤーを更新する

```
1 from qgis.core import QgsProject
2
3 layer = QgsProject.instance().mapLayersByName("layer name you like")[0]
4 # Set seconds (5 seconds)
5 layer.setAutoRefreshInterval(5000)
6 # Enable auto refresh
7 layer.setAutoRefreshEnabled(True)
```

メソッドを表示する

```
dir(layer)
```

地物フォームを使って新たに地物を追加する

```
1 from qgis.core import QgsFeature, QgsGeometry
2
3 feat = QgsFeature()
4 geom = QgsGeometry()
5 feat.setGeometry(geom)
6 feat.setFields(layer.fields())
7
8 iface.openFeatureForm(layer, feat, False)
```

地物フォームを使わずに新たに地物を追加する

```
1 from qgis.core import QgsGeometry, QgsPointXY, QgsFeature
2
3 pr = layer.dataProvider()
4 feat = QgsFeature()
5 feat.setGeometry(QgsGeometry.fromPointXY(QgsPointXY(10,10)))
6 pr.addFeatures([feat])
```

地物を取得する

```
for f in layer.getFeatures():
    print (f)
```

```
<qgis._core.QgsFeature object at 0x7f45cc64b678>
```

選択された地物を取得する

```
for f in layer.selectedFeatures():
    print (f)
```

選択された地物の ID を取得する

```
selected_ids = layer.selectedFeatureIds()
print(selected_ids)
```

選択された地物の ID からメモリに一時レイヤを作成する

```
from qgis.core import QgsFeatureRequest

memory_layer = layer.materialize(QgsFeatureRequest().setFilterFids(layer.
↪selectedFeatureIds()))
QgsProject.instance().addMapLayer(memory_layer)
```

ジオメトリを取得する

```
# Point layer
for f in layer.getFeatures():
    geom = f.geometry()
    print ('%f, %f' % (geom.asPoint().y(), geom.asPoint().x()))
```

```
10.000000, 10.000000
```

ジオメトリを移動する

```
1 from qgis.core import QgsFeature, QgsGeometry
2 poly = QgsFeature()
3 geom = QgsGeometry.fromWkt("POINT(7 45)")
4 geom.translate(1, 1)
5 poly.setGeometry(geom)
6 print(poly.geometry())
```

```
<QgsGeometry: Point (8 46)>
```

CRS を設定する

```
from qgis.core import QgsProject, QgsCoordinateReferenceSystem

for layer in QgsProject.instance().mapLayers().values():
    layer.setCrs(QgsCoordinateReferenceSystem('EPSG:4326'))
```

CRSを確認する

```

1 from qgis.core import QgsProject
2
3 for layer in QgsProject.instance().mapLayers().values():
4     crs = layer.crs().authid()
5     layer.setName('{} ({}).format(layer.name(), crs)

```

特定のフィールド列を隠す

```

1 from qgis.core import QgsEditorWidgetSetup
2
3 def fieldVisibility (layer, fname):
4     setup = QgsEditorWidgetSetup('Hidden', {})
5     for i, column in enumerate(layer.fields()):
6         if column.name() == fname:
7             layer.setEditorWidgetSetup(idx, setup)
8             break
9         else:
10            continue

```

Layer from WKT

```

1 from qgis.core import QgsVectorLayer, QgsFeature, QgsGeometry, QgsProject
2
3 layer = QgsVectorLayer('Polygon?crs=epsg:4326', 'Mississippi', 'memory')
4 pr = layer.dataProvider()
5 poly = QgsFeature()
6 geom = QgsGeometry.fromWkt("POLYGON ((-88.82 34.99,-88.0934.89,-88.39 30.34,-89.57_
↳30.18,-89.73 31,-91.63 30.99,-90.8732.37,-91.23 33.44,-90.93 34.23,-90.30 34.99,-
↳88.82 34.99))")
7 poly.setGeometry(geom)
8 pr.addFeatures([poly])
9 layer.updateExtents()
10 QgsProject.instance().addMapLayers([layer])

```

Load all vector layers from GeoPackage

```

1 fileName = "testdata/sublayers.gpkg"
2 layer = QgsVectorLayer(fileName, "test", "ogr")
3 subLayers = layer.dataProvider().subLayers()
4
5 for subLayer in subLayers:
6     name = subLayer.split('!::!!')[1]
7     uri = "%s|layername=%s" % (fileName, name,)
8     # Create layer
9     sub_vlayer = QgsVectorLayer(uri, name, 'ogr')
10    # Add layer to map
11    QgsProject.instance().addMapLayer(sub_vlayer)

```

タイルレイヤ (XYZ-Layer) を読み込む


```

1 from qgis.core import QgsRasterLayer, QgsProject
2
3 def loadXYZ(url, name):
4     rasterLyr = QgsRasterLayer("type=xyz&url=" + url, name, "wms")
5     QgsProject.instance().addMapLayer(rasterLyr)
6
7 urlWithParams = 'type=xyz&url=https://a.tile.openstreetmap.org/%7Bz%7D/%7Bx%7D/%7By
  ↳%7D.png&zmax=19&zmin=0&crs=EPSG3857'
8 loadXYZ(urlWithParams, 'OpenStreetMap')

```

すべてのレイヤを削除する

```
QgsProject.instance().removeAllMapLayers()
```

すべてを削除する

```
QgsProject.instance().clear()
```

20.7 Table of contents

チェックの入ったレイヤにアクセスする

```
iface.mapCanvas().layers()
```

Remove contextual menu

```

1 ltv = iface.layerTreeView()
2 mp = ltv.menuProvider()
3 ltv.setMenuProvider(None)
4 # Restore
5 ltv.setMenuProvider(mp)

```

20.8 Advanced TOC

Root ノード

```

1 from qgis.core import QgsVectorLayer, QgsProject, QgsLayerTreeLayer
2
3 root = QgsProject.instance().layerTreeRoot()
4 node_group = root.addGroup("My Group")
5
6 layer = QgsVectorLayer("Point?crs=EPSG:4326", "layer name you like", "memory")
7 QgsProject.instance().addMapLayer(layer, False)
8
9 node_group.addLayer(layer)
10

```

(次のページに続く)

(前のページからの続き)

```

11 print(root)
12 print(root.children())

```

最初の子ノードにアクセスする

```

1 from qgis.core import QgsLayerTreeGroup, QgsLayerTreeLayer, QgsLayerTree
2
3 child0 = root.children()[0]
4 print(child0.name())
5 print(type(child0))
6 print(isinstance(child0, QgsLayerTreeLayer))
7 print(isinstance(child0.parent(), QgsLayerTree))

```

```

My Group
<class 'qgis._core.QgsLayerTreeGroup'>
False
True

```

グループとそのノードを取得する

```

1 from qgis.core import QgsLayerTreeGroup, QgsLayerTreeLayer
2
3 def get_group_layers(group):
4     print('- group: ' + group.name())
5     for child in group.children():
6         if isinstance(child, QgsLayerTreeGroup):
7             # Recursive call to get nested groups
8             get_group_layers(child)
9         else:
10            print(' - layer: ' + child.name())
11
12
13 root = QgsProject.instance().layerTreeRoot()
14 for child in root.children():
15     if isinstance(child, QgsLayerTreeGroup):
16         get_group_layers(child)
17     elif isinstance(child, QgsLayerTreeLayer):
18         print('- layer: ' + child.name())

```

```

- group: My Group
- layer: layer name you like

```

名前からグループを取得する

```
print(root.findGroup("My Group"))
```

```
<qgis._core.QgsLayerTreeGroup object at 0x7fd75560cee8>
```

ID からレイヤを取得する

```
print(root.findLayer(layer.id()))
```

```
<qgis._core.QgsLayerTreeLayer object at 0x7f56087af288>
```

レイヤを追加する

```
1 from qgis.core import QgsVectorLayer, QgsProject
2
3 layer1 = QgsVectorLayer("Point?crs=EPSG:4326", "layer name you like 2", "memory")
4 QgsProject.instance().addMapLayer(layer1, False)
5 node_layer1 = root.addLayer(layer1)
6 # Remove it
7 QgsProject.instance().removeMapLayer(layer1)
```

グループを追加する

```
1 from qgis.core import QgsLayerTreeGroup
2
3 node_group2 = QgsLayerTreeGroup("Group 2")
4 root.addChildNode(node_group2)
5 QgsProject.instance().mapLayersByName("layer name you like")[0]
```

読み込んだレイヤを移動する

```
1 layer = QgsProject.instance().mapLayersByName("layer name you like")[0]
2 root = QgsProject.instance().layerTreeRoot()
3
4 myLayer = root.findLayer(layer.id())
5 myClone = myLayer.clone()
6 parent = myLayer.parent()
7
8 myGroup = root.findGroup("My Group")
9 # Insert in first position
10 myGroup.insertChildNode(0, myClone)
11
12 parent.removeChildNode(myLayer)
```

Move loaded layer to a specific group

```
1 QgsProject.instance().addMapLayer(layer, False)
2
3 root = QgsProject.instance().layerTreeRoot()
4 myGroup = root.findGroup("My Group")
5 myOriginalLayer = root.findLayer(layer.id())
6 myLayer = myOriginalLayer.clone()
7 myGroup.insertChildNode(0, myLayer)
8 parent.removeChildNode(myOriginalLayer)
```

表示/非表示を変更する

```
myGroup.setItemVisibilityChecked(False)
myLayer.setItemVisibilityChecked(False)
```

あるグループが選択されているかどうか

```
1 def isMyGroupSelected( groupName ):
2     myGroup = QgsProject.instance().layerTreeRoot().findGroup( groupName )
3     return myGroup in iface.layerTreeView().selectedNodes()
4
5 print(isMyGroupSelected( 'my group name' ))
```

```
False
```

ノードを展開する/折り畳む

```
print(myGroup.isExpanded())
myGroup.setExpanded(False)
```

Hidden node trick

```
1 from qgis.core import QgsProject
2
3 model = iface.layerTreeView().layerTreeModel()
4 ltv = iface.layerTreeView()
5 root = QgsProject.instance().layerTreeRoot()
6
7 layer = QgsProject.instance().mapLayersByName('layer name you like')[0]
8 node = root.findLayer( layer.id() )
9
10 index = model.node2index( node )
11 ltv.setRowHidden( index.row(), index.parent(), True )
12 node.setCustomProperty( 'nodeHidden', 'true' )
13 ltv.setCurrentIndex( model.node2index( root ) )
```

Node signals

```
1 def onWillAddChildren(node, indexFrom, indexTo):
2     print ("WILL ADD", node, indexFrom, indexTo)
3
4 def onAddedChildren(node, indexFrom, indexTo):
5     print ("ADDED", node, indexFrom, indexTo)
6
7 root.willAddChildren.connect( onWillAddChildren )
8 root.addedChildren.connect( onAddedChildren )
```

レイヤを削除する

```
root.removeLayer(layer)
```

グループを削除する

```
root.removeChildNode(node_group2)
```

Create new table of contents (TOC)

```
1 root = QgsProject.instance().layerTreeRoot()
2 model = QgsLayerTreeModel(root)
3 view = QgsLayerTreeView()
4 view.setModel(model)
5 view.show()
```

ノードを移動する

```
cloned_group1 = node_group.clone()
root.insertChildNode(0, cloned_group1)
root.removeChildNode(node_group)
```

ノード名を変更する

```
cloned_group1.setName("Group X")
node_layer1.setName("Layer X")
```

20.9 プロセシングアルゴリズム

アルゴリズムの一覧を取得する

```
1 from qgis.core import QgsApplication
2
3 for alg in QgsApplication.processingRegistry().algorithms():
4     if 'buffer' == alg.name():
5         print("{}: {} --> {}".format(alg.provider().name(), alg.name(), alg.
↳ displayName()))
```

```
QGIS (native c++):buffer --> Buffer
```

アルゴリズムのヘルプを取得する

Random selection

```
from qgis import processing
processing.algorithmHelp("native:buffer")
```

```
...
```

アルゴリズムを実行する

この例では、プロジェクトに追加された一時メモリレイヤに結果が格納される。

```
from qgis import processing
result = processing.run("native:buffer", {'INPUT': layer, 'OUTPUT': 'memory:'})
QgsProject.instance().addMapLayer(result['OUTPUT'])
```

```
Processing(0): Results: {'OUTPUT': 'output_d27a2008_970c_4687_b025_f057abbd7319'}
```

アルゴリズムはいくつあるか？

```
len(QgsApplication.processingRegistry().algorithms())
```

プロバイダはいくつあるか？

```
from qgis.core import QgsApplication

len(QgsApplication.processingRegistry().providers())
```

式はいくつあるか？

```
from qgis.core import QgsExpression

len(QgsExpression.Functions())
```

20.10 装飾類

著作権

```
1 from qgis.PyQt.Qt import QTextDocument
2 from qgis.PyQt.QtGui import QFont
3
4 mQFont = "Sans Serif"
5 mQFontsize = 9
6 mLabelQString = "^^c2^^a9 QGIS 2019"
7 mMarginHorizontal = 0
8 mMarginVertical = 0
9 mLabelQColor = "#FF0000"
10
11 INCHES_TO_MM = 0.0393700787402 # 1 millimeter = 0.0393700787402 inches
12 case = 2
13
14 def add_copyright(p, text, xOffset, yOffset):
15     p.translate( xOffset , yOffset )
16     text.drawContents(p)
17     p.setWorldTransform( p.worldTransform() )
18
19 def _on_render_complete(p):
20     deviceHeight = p.device().height() # Get paint device height on which this_
↳painter is currently painting
21     deviceWidth = p.device().width() # Get paint device width on which this_
↳painter is currently painting
```

(次のページに続く)

(前のページからの続き)

```

22  # Create new container for structured rich text
23  text = QTextDocument()
24  font = QFont()
25  font.setFamily(mQFont)
26  font.setPointSize(int(mQFontsize))
27  text.setDefaultFont(font)
28  style = "<style type=\"text/css\"> p {color: " + mLabelQColor + "</style>"
29  text.setHtml( style + "<p>" + mLabelQString + "</p>" )
30  # Text Size
31  size = text.size()
32
33  # RenderMillimeters
34  pixelsInchX = p.device().logicalDpiX()
35  pixelsInchY = p.device().logicalDpiY()
36  xOffset = pixelsInchX * INCHES_TO_MM * int(mMarginHorizontal)
37  yOffset = pixelsInchY * INCHES_TO_MM * int(mMarginVertical)
38
39  # Calculate positions
40  if case == 0:
41      # Top Left
42      add_copyright(p, text, xOffset, yOffset)
43
44  elif case == 1:
45      # Bottom Left
46      yOffset = deviceHeight - yOffset - size.height()
47      add_copyright(p, text, xOffset, yOffset)
48
49  elif case == 2:
50      # Top Right
51      xOffset = deviceWidth - xOffset - size.width()
52      add_copyright(p, text, xOffset, yOffset)
53
54  elif case == 3:
55      # Bottom Right
56      yOffset = deviceHeight - yOffset - size.height()
57      xOffset = deviceWidth - xOffset - size.width()
58      add_copyright(p, text, xOffset, yOffset)
59
60  elif case == 4:
61      # Top Center
62      xOffset = deviceWidth / 2
63      add_copyright(p, text, xOffset, yOffset)
64
65  else:
66      # Bottom Center
67      yOffset = deviceHeight - yOffset - size.height()
68      xOffset = deviceWidth / 2
69      add_copyright(p, text, xOffset, yOffset)
70
71  # Emitted when the canvas has rendered
72  iface.mapCanvas().renderComplete.connect(_on_render_complete)

```

(次のページに続く)

```
73 # Repaint the canvas map
74 iface.mapCanvas().refresh()
```

20.11 Composer

印刷レイアウトを名前で取得する

```
1 composerTitle = 'MyComposer' # Name of the composer
2
3 project = QgsProject.instance()
4 projectLayoutManager = project.layoutManager()
5 layout = projectLayoutManager.layoutByName(composerTitle)
```

20.12 出典

- [QGIS Python \(PyQGIS\) API](#)
- [QGIS C++ API](#)
- [StackOverFlow QGIS questions](#)
- [Script by Klas Karlsson](#)