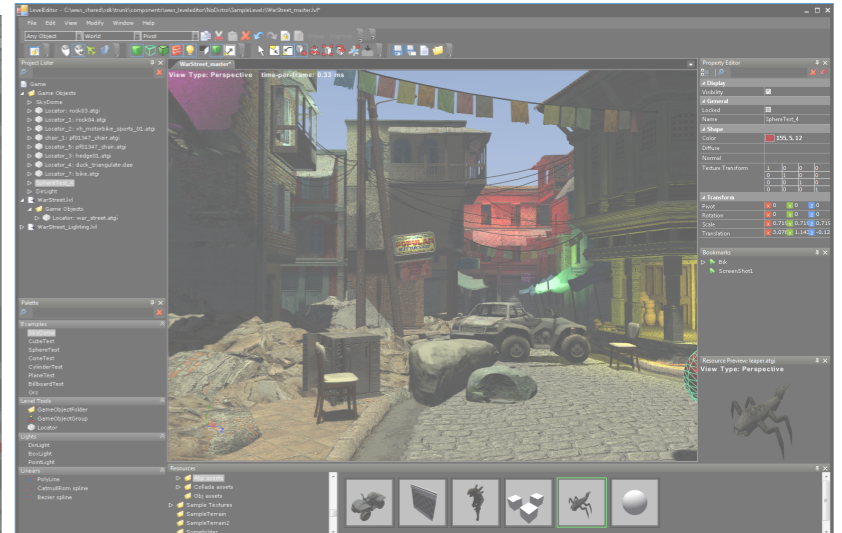
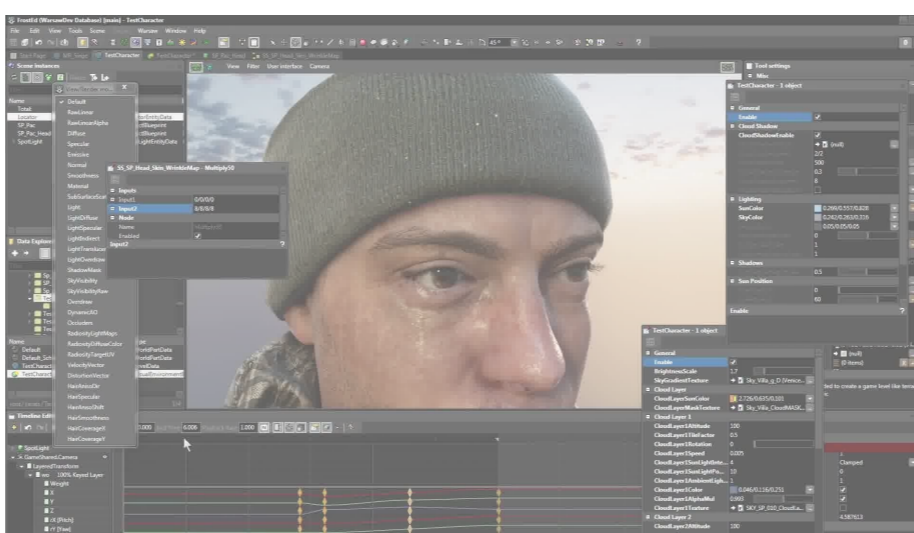
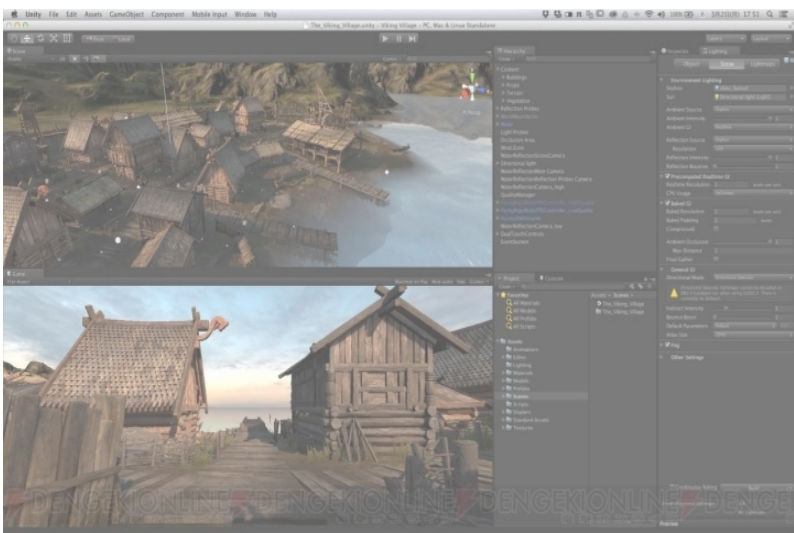


# ゲーム開発のためのツール制作入門





# 目次

---

- **なんでやるうと思ったのか**
- **どんなことが出来るのか**
- **開発環境の導入**
- **Qtとは、メリットとデメリット**
- **実際に触れてみる**
- **実践的に使うための準備**
- **DirectXのレンダラーをツールと連動させる**
- **ツール上でレンダリングする**
- **自分のフレームワーク上にレンダリングする**
- **入力とレンダリングの間**
- **配布時の注意点**
- **さらに実践的なQtツール制作上のノウハウについて**

# なんでやるうとおもったのか

---

**未来想像展で授業の内容で頑張る▶ミドルウェアに負ける**

**なぜ？**

**多くのゲームエンジンにはレベルエディターと呼ばれるツールが存在する。（レベルデザインを行うためのツール）**

**自分たちの授業のフレームワークで作るうとすると、このレベルデザインを行うための環境が貧弱なため、ゲームの面白さを形づくるための工程において生産性が著しく減少してしまいます。**

# なんでやるうとおもったのか

自分たちでレベルデザインを行うための環境を用意する方法がいくつかあります。

- ソースコードのみでごり押しで全て作る

最もダメな方法 変更が脆くステージは大量生産不可能

- テキスト,CSV等で形式立てて用意する

ソースコードより良い方法。他の人に頼める。しかし、マップチップベースになりがちで、ゲームの全体的デザインがセルベースに固定化されてしまいがち。

- レベルデザインツールを作成する

最も良い方法だがHALにおいては今だに出来る人任せ

▲今回はこの部分を解決したい



# なんでやるうとおもったのか

海外では・・・



▶これが大学の授業の教科書として  
つかわれている。(2010年時点)

ゲームエンジンの内部知識、制作手法が教育レベルで  
行われており、技術レベルは日本はかなり遅れています。

**なんでやろうとおもったのか**

---

**しかし、レベルデザインツールを作ろうとすると...**

**ええ、しってます。  
面倒なんですよ**

**なんでやろうとおもったのか**

---

**しかし、レベルデザインツールを作ろうとすると...**

**そうです。**

**いにしえのWindowsです。**



## なんでやろうとおもったのか

---

しかし、レベルデザインツールを作ろうとすると...

非常に使いにくい基本ツールに、  
鬼のようなSwitchCase文、いずれ大量のネストコードで  
脳をやられてしまい、何をしていたのか忘れ、  
これでは、レベルエディタを作っていたらプロジェクトが  
終わってしまう！



**そうだ！ C#があるじゃないか！**

**なんでやろうとおもったのか**

---

**しかし、C#でレベルデザインツールを作ろうとすると...**

**ええ、しってます。  
面倒なんですよ**

**なんでやろうとおもったのか**

---

**しかし、C#でレベルデザインツールを作ろうとすると...**

**そうです。**

**いにしえのDirectXです。**



## なんでやるうとおもったのか

---

しかし、C#でレベルデザインツールを作ろうとすると・・・  
正攻法でやる場合、C#で動作するManaged DirectX  
と呼ばれるC#用のAPIを利用して、描画のコードを書かな  
ければならず、C++でのDirectXとはソースコードが違い  
C++とC#で別のDirectXコードを記述しないと  
いけない。超めんどい。

(一応DLLインジェクションという裏技っぽい方法でC++側  
のBegin,Endをフックするという荒技がある。超むずい)



そうだ！ Qtがあるじゃないか！

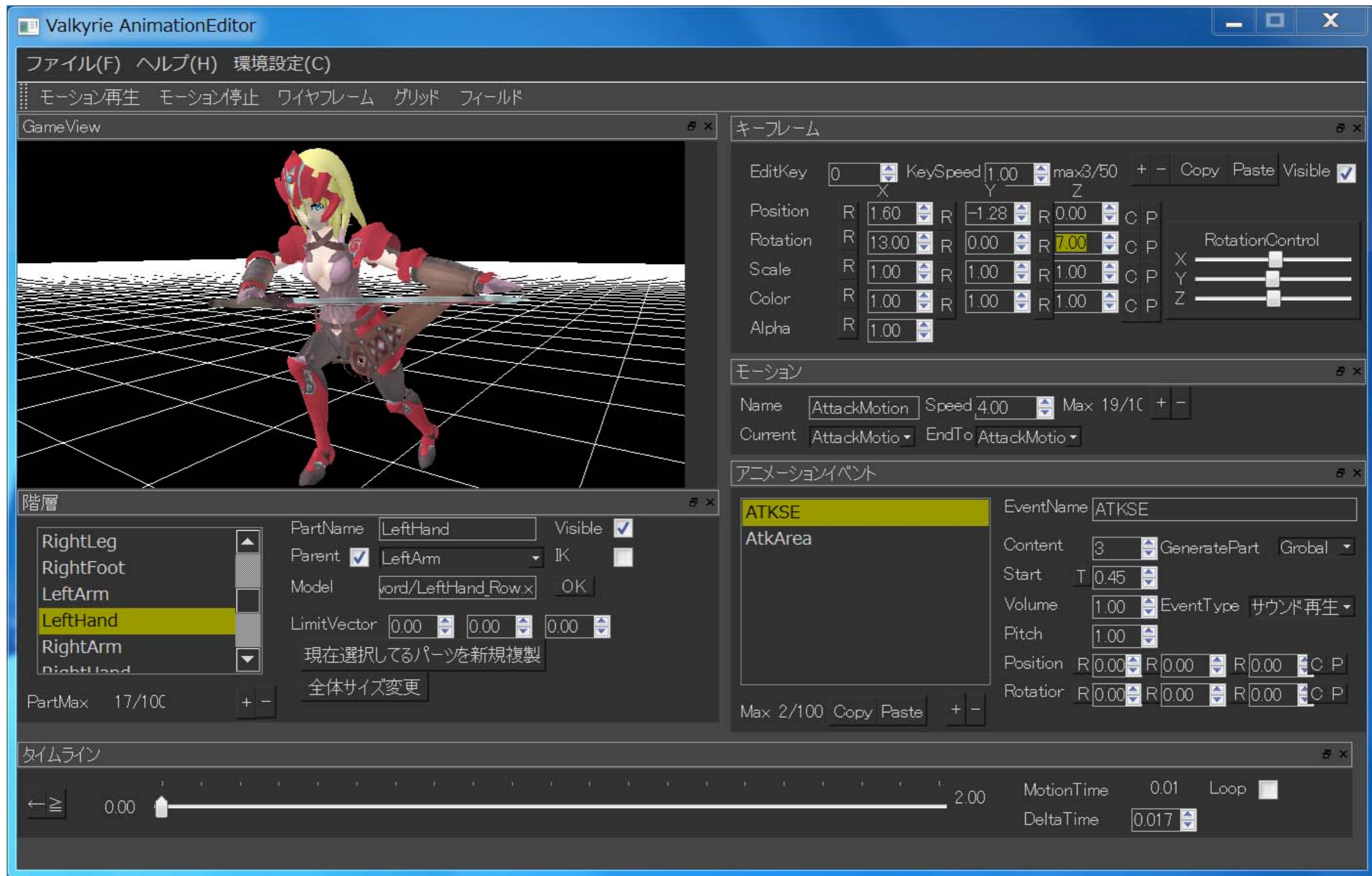
# 目次

---

- **なんでやるうと思ったのか**
- **どんなことが出来るのか**
- **開発環境の導入**
- **Qtとは、メリットとデメリット**
- **実際に触れてみる**
- **実践的に使うための準備**
- **DirectXのレンダラーをツールと連動させる**
- **ツール上でレンダリングする**
- **自分のフレームワーク上にレンダリングする**
- **入力とレンダリングの間**
- **配布時の注意点**
- **さらに実践的なQtツール制作上のノウハウについて**

# どんなことができるのか

## ・アニメーションエディタ

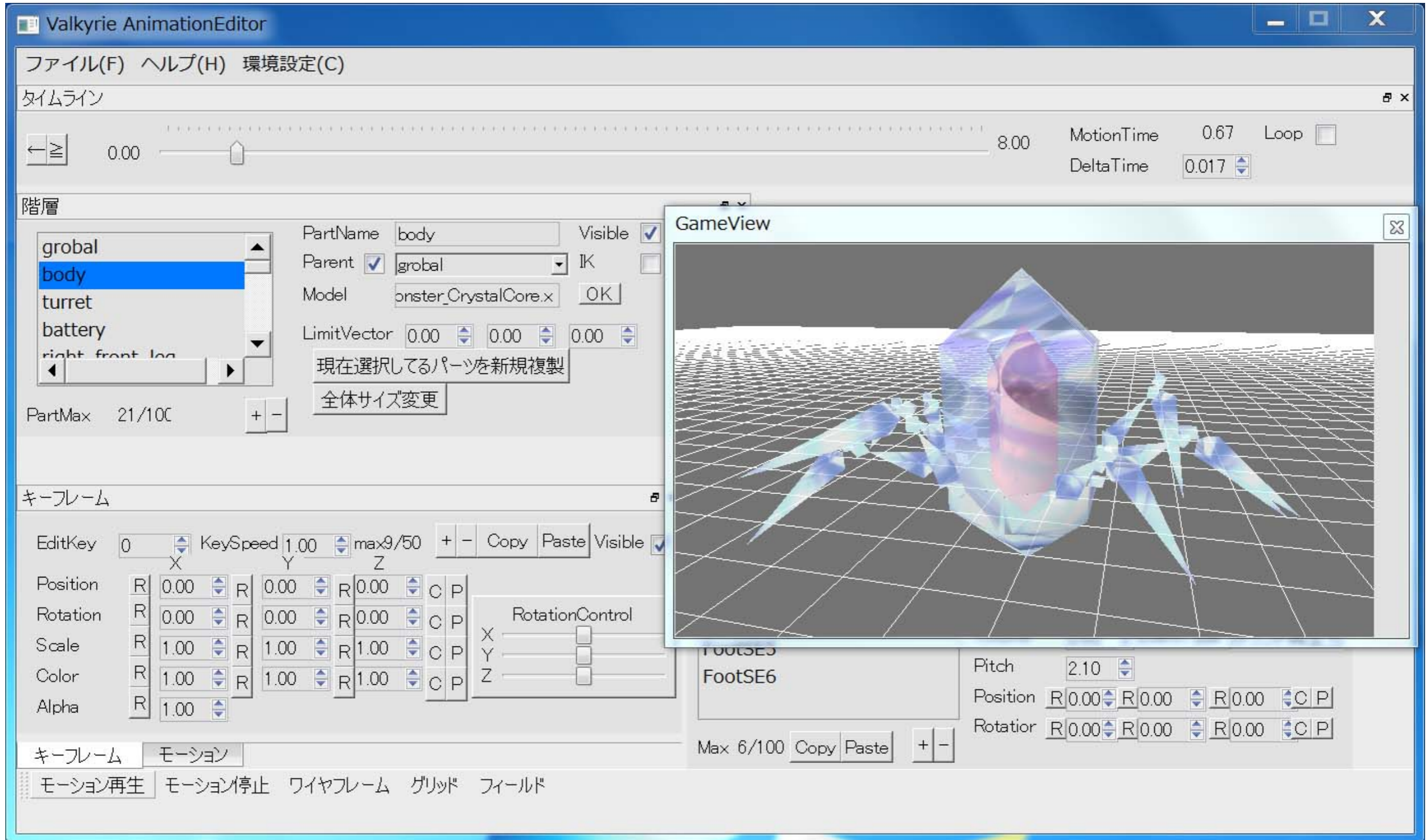


## ・制作期間:3週間



# どんなことができるのか

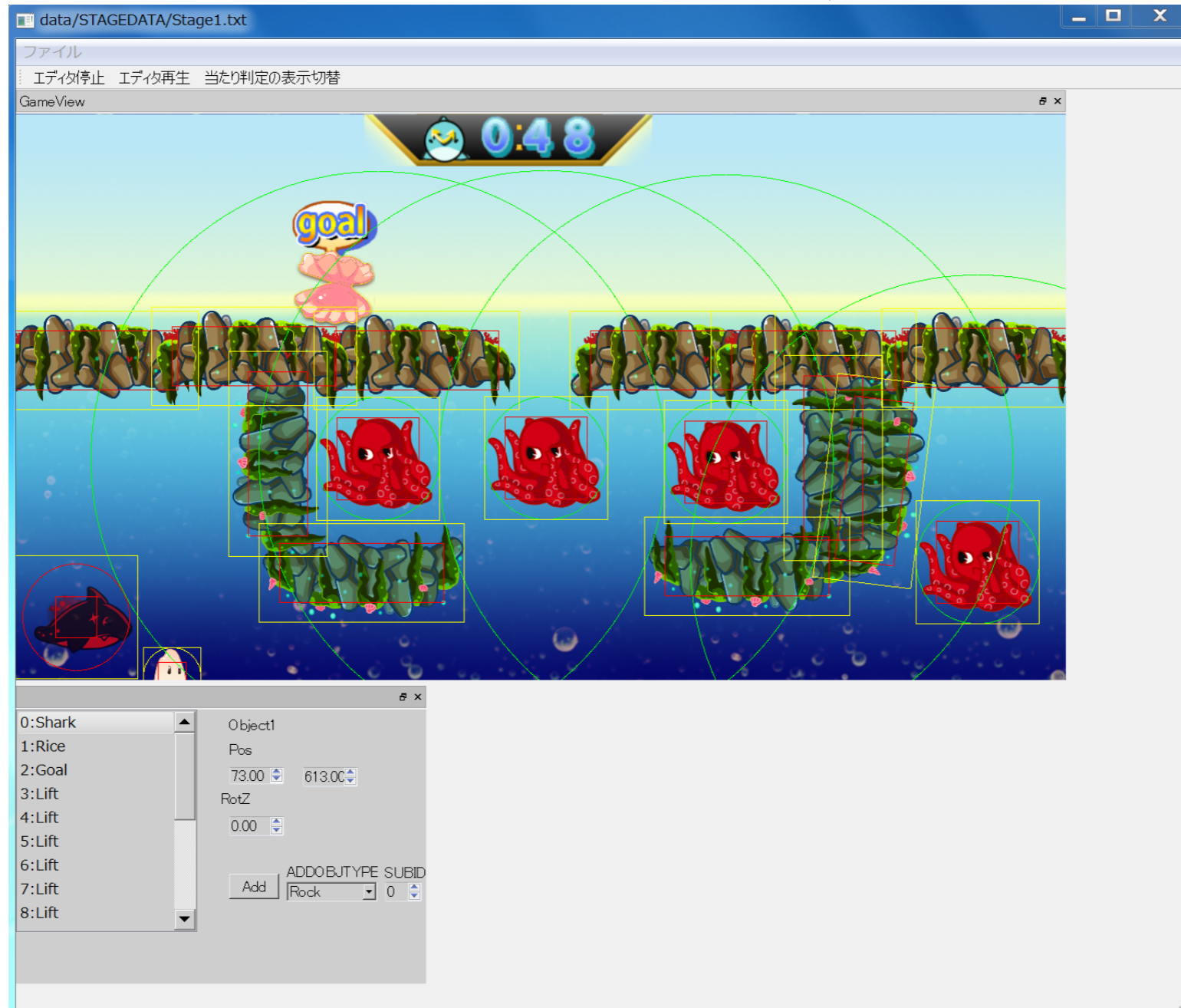
## ・アニメーションエディタ



## ・DirectX9,DirectX11に対応

# どんなことができるのか

## ・2Dレベルエディタ



## ・制作期間: 3日

(最低限の座標編集,オブジェクト選択,セーブロードまで4時間)

# 目次

---

- **なんでやるうと思ったのか**
- **どんなことが出来るのか**
- **開発環境の導入**
- **Qtとは、メリットとデメリット**
- **実際に触れてみる**
- **実践的に使うための準備**
- **DirectXのレンダラーをツールと連動させる**
- **ツール上でレンダリングする**
- **自分のフレームワーク上にレンダリングする**
- **入力とレンダリングの間**
- **配布時の注意点**
- **さらに実践的なQtツール制作上のノウハウについて**



# 開発環境の導入

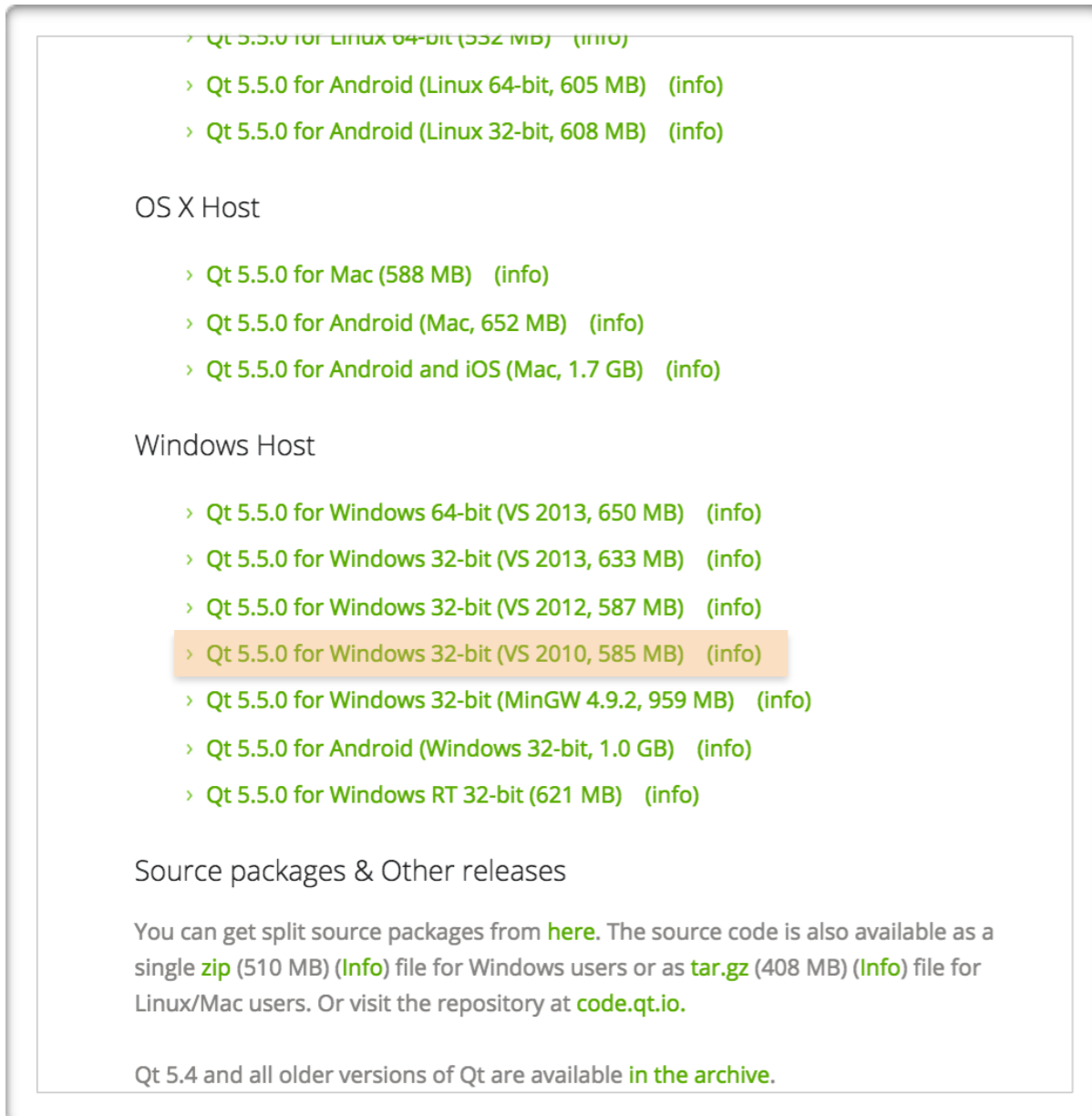
## ・まずはQtでぐぐる

The screenshot shows the Qt Developers website homepage. At the top, there is a navigation bar with the Qt logo on the left and links for Download, Device Creation, Application Development, Services, Developers (highlighted), Blog, Partners, and Sign in on the right. A search icon and the word 'Search' are also present. Below the navigation bar, there are three main sections: 'Get started' with a flag icon and links for Installation, Examples & tutorials, and Tools; 'Use Qt' with a document icon and links for Supported platforms, Reference pages, and Wiki; and 'Join the Community' with a network icon and links for Forums, Contribute, and Mailing lists. Each section has a green button at the bottom: 'Download now', 'Documentation', and 'Showcase your app' respectively. Language options (EN, RU, ZH) are visible in the top right corner.

・ホームページは毎回リニューアルされて非常に厄介なため注意！

# 開発環境の導入

## ・インストーラを回収！



The screenshot shows the Qt 5.5.0 download page with the following content:

- Qt 5.5.0 for Linux 64-bit (552 MB) (info)
- Qt 5.5.0 for Android (Linux 64-bit, 605 MB) (info)
- Qt 5.5.0 for Android (Linux 32-bit, 608 MB) (info)

OS X Host

- Qt 5.5.0 for Mac (588 MB) (info)
- Qt 5.5.0 for Android (Mac, 652 MB) (info)
- Qt 5.5.0 for Android and iOS (Mac, 1.7 GB) (info)

Windows Host

- Qt 5.5.0 for Windows 64-bit (VS 2013, 650 MB) (info)
- Qt 5.5.0 for Windows 32-bit (VS 2013, 633 MB) (info)
- Qt 5.5.0 for Windows 32-bit (VS 2012, 587 MB) (info)
- Qt 5.5.0 for Windows 32-bit (VS 2010, 585 MB) (info)
- Qt 5.5.0 for Windows 32-bit (MinGW 4.9.2, 959 MB) (info)
- Qt 5.5.0 for Android (Windows 32-bit, 1.0 GB) (info)
- Qt 5.5.0 for Windows RT 32-bit (621 MB) (info)

Source packages & Other releases

You can get split source packages from [here](#). The source code is also available as a single [zip](#) (510 MB) (Info) file for Windows users or as [tar.gz](#) (408 MB) (Info) file for Linux/Mac users. Or visit the repository at [code.qt.io](#).

Qt 5.4 and all older versions of Qt are available [in the archive](#).

・Qt本体を回収  
とりあえず32bitにしとく。  
各自が持つてる  
VisualStuidoの  
やつに合わせて回収をすること。

入れない時<http://www.qt.io/download-open-source/#section-2>



# 開発環境の導入

## • VisualStudioを拡張する！

The source code is available as a [zip](#) (30 MB) ([Info](#)) or a [tar.gz](#) (22 MB) ([Info](#)). Or visit the repository at [code.qt.io](#).

Be sure to check if Qt is supported on your platform and read the installation notes that are located in the [Qt Documentation](#).

### Other downloads

- > [Visual Studio Add-in 1.1.11 for Qt4](#) (112 MB) ([info](#))
- > [Visual Studio Add-in 1.2.4 for Qt5](#) (156 MB) ([info](#))
- > [JOM 1.0.14](#) (684 KB) ([info](#))
- > [Qt Build Suite \(QBS\)](#)
- > [Qt Installer Framework](#)
- > [Qt repositories at code.qt.io](#)
- > [Archive for old versions](#)

Please check the individual downloads for licensing information.

### Pre-releases

- Qt VisualStudioアドインを  
回収とりあえずQt5にしとく。  
2つとも回収後にインストールを  
行う（まあまあ時間かかる）

**入れない時**<http://www.qt.io/download-open-source/#section-2>

# 目次

---

- **なんでやるうと思ったのか**
- **どんなことが出来るのか**
- **開発環境の導入**
- **Qtとは、メリットとデメリット**
- **実際に触れてみる**
- **実践的に使うための準備**
- **DirectXのレンダラーをツールと連動させる**
- **ツール上でレンダリングする**
- **自分のフレームワーク上にレンダリングする**
- **入力とレンダリングの間**
- **配布時の注意点**
- **さらに実践的なQtツール制作上のノウハウについて**

# Qtとは、メリットとデメリット

---

- クロスプラットフォームGUIツールフレームワークです。  
そのためOpenGLについても標準で対応しています。  
よみかたは「**キュート**」と呼ぶそうです。  
キューティーじゃないよ

# Qとは、メリットとデメリット

---

## • Qtで作られているもの

### Skype



### Doxygen



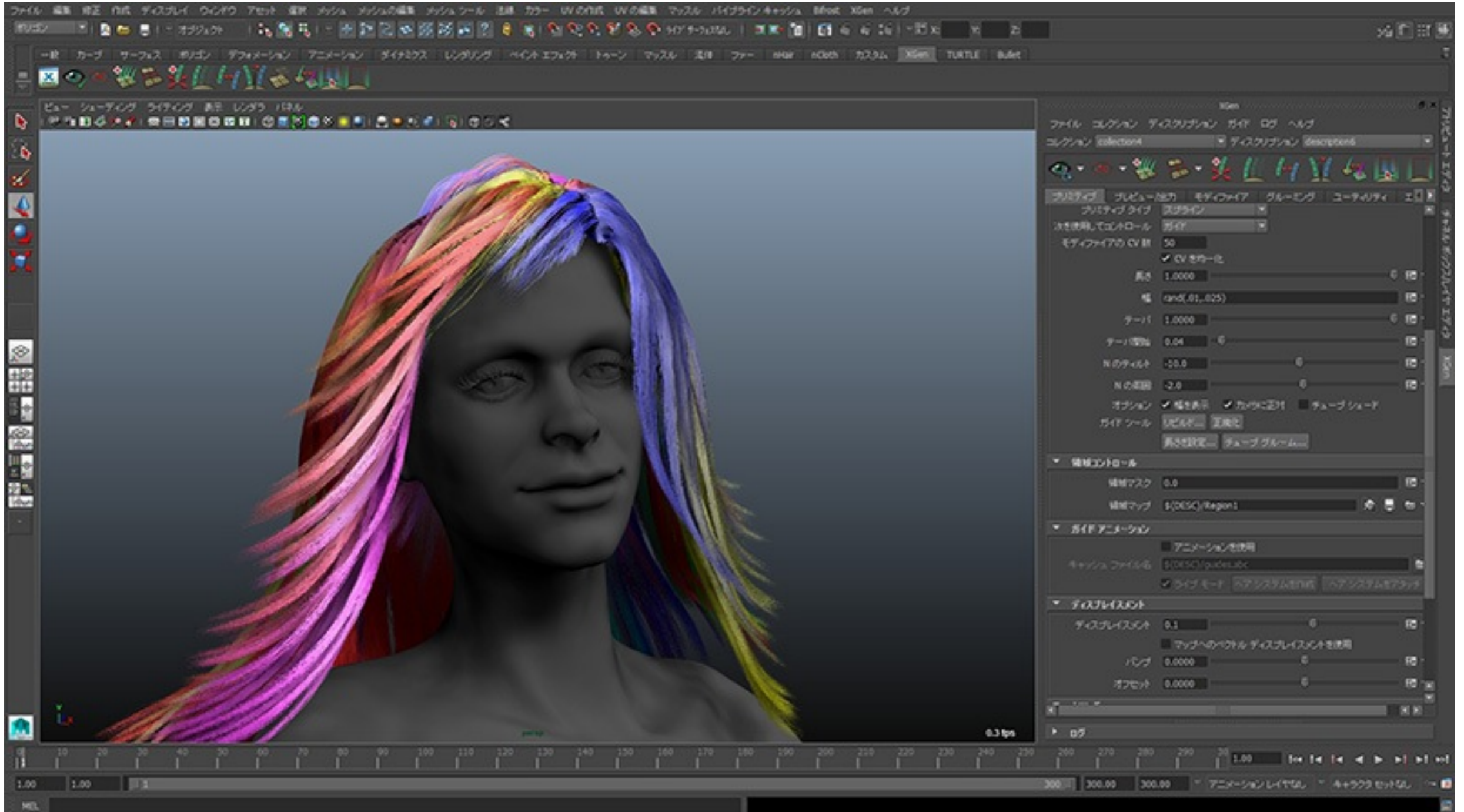
### GoogleEarth



# Qとは、メリットとデメリット

## • Qtで作られているもの

# Maya





# Qtとは、メリットとデメリット

---

## <利用によるメリット>

- **驚異的なスピードでツールプログラムを記述できる。  
1,2分もかからずに、新しい機能を取り込むことが  
不可能ではない。**
- **C#のような豊富なツールキットをC++のコードでやれる**
- **C#よりも便利な機能がある。**
- **Visual Studio と連携が可能**
- **セーブデータをfread ,fwrite  
してしまえば証拠が闇に飲まれてしまう  
(セーブデータがスクリプトだと本当に闇の中)**

# Qtとは、メリットとデメリット

## <利用によるデメリット>

- 商用や企業で利用するとなると、ライセンスについて確認を行わなければならない。
- Qtのオブジェクトはnewを拡張してるため、企業での利用等でカスタムメモリアロケートされてる状況の場合、併用できない可能性がある。
- まだQtライブラリにバグが残ってる  
(ある関数がメモリリークの原因となりうる)
- FPS制御が難しい
- 日本語の資料が少ない
- **DirectXを組み込めた人間をほとんど知らない(苦行)**

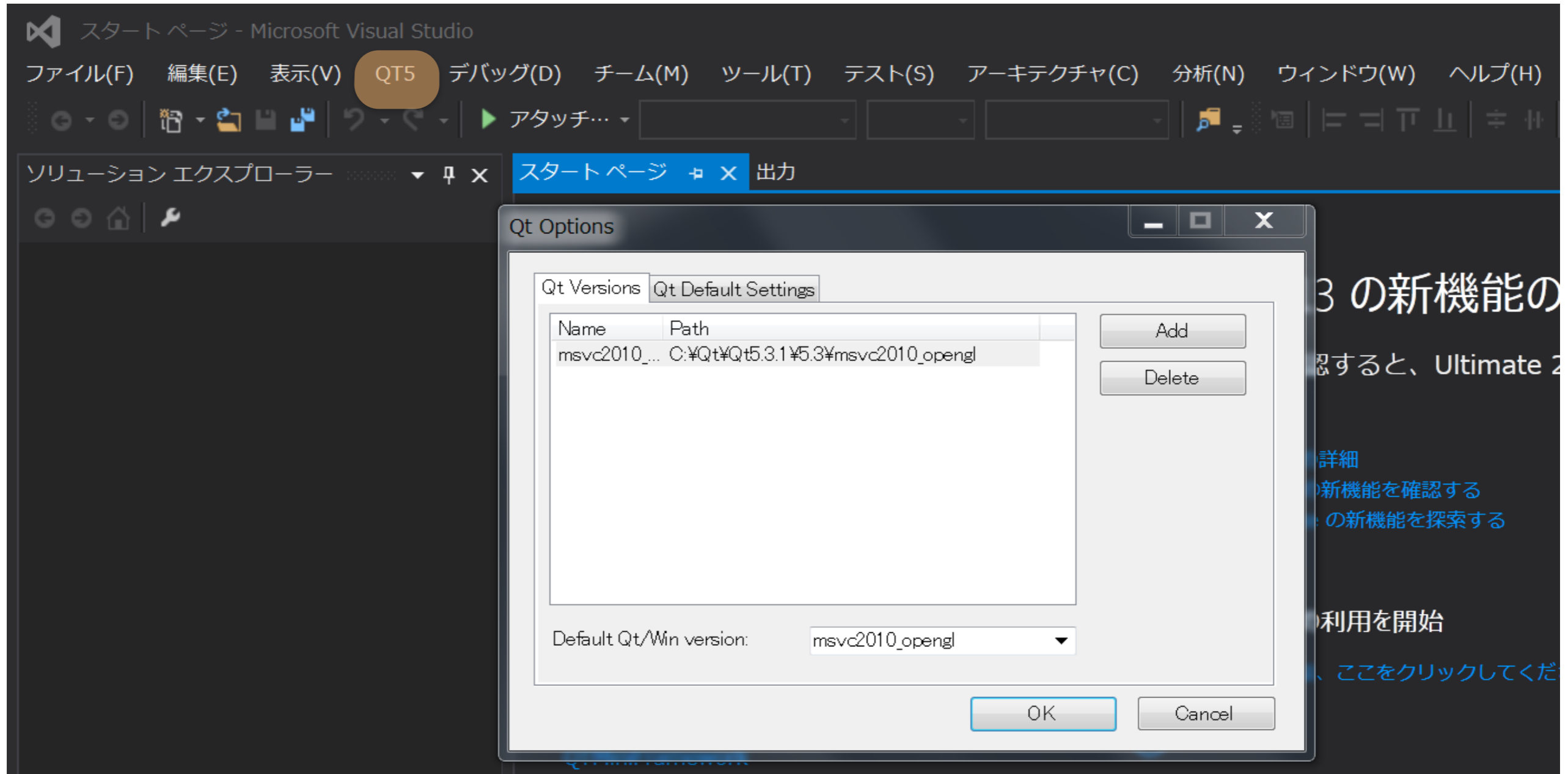
# 目次

---

- **なんでやるうと思ったのか**
- **どんなことが出来るのか**
- **開発環境の導入**
- **Qtとは、メリットとデメリット**
- **実際に触れてみる**
- **実践的に使うための準備**
- **DirectXのレンダラーをツールと連動させる**
- **ツール上でレンダリングする**
- **自分のフレームワーク上にレンダリングする**
- **入力とレンダリングの間**
- **配布時の注意点**
- **さらに実践的なQtツール制作上のノウハウについて**

# 実際に触れてみる

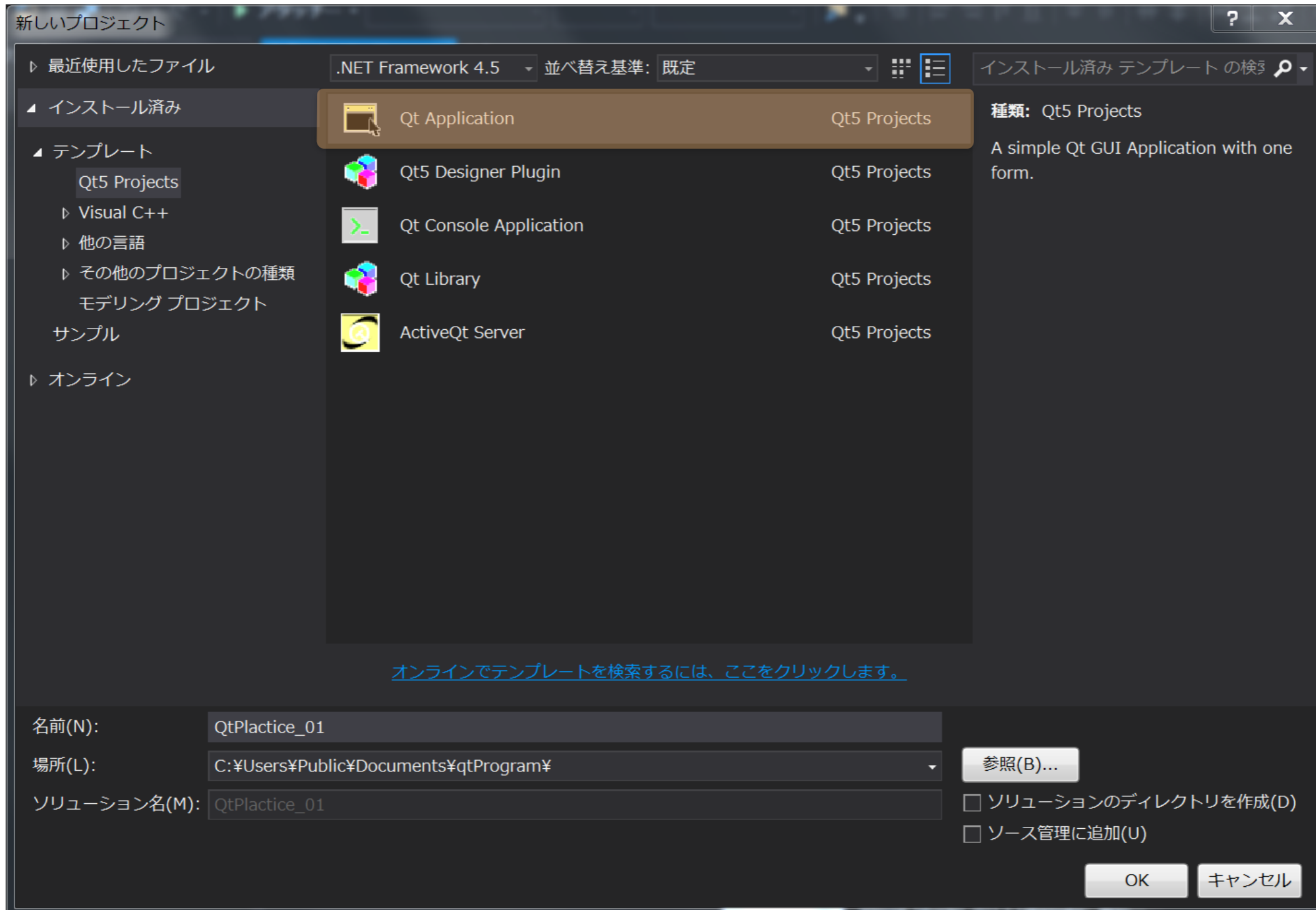
- プロジェクトにQT5という項目があることを確認します。



QTのプロジェクト設定より、QT本体のパスを設定  
もしかしたら現在はOpenGL Verも統合されてるかも

# 実際に触れてみる

- プロジェクトの新規作成をすると見たことないコマンドが

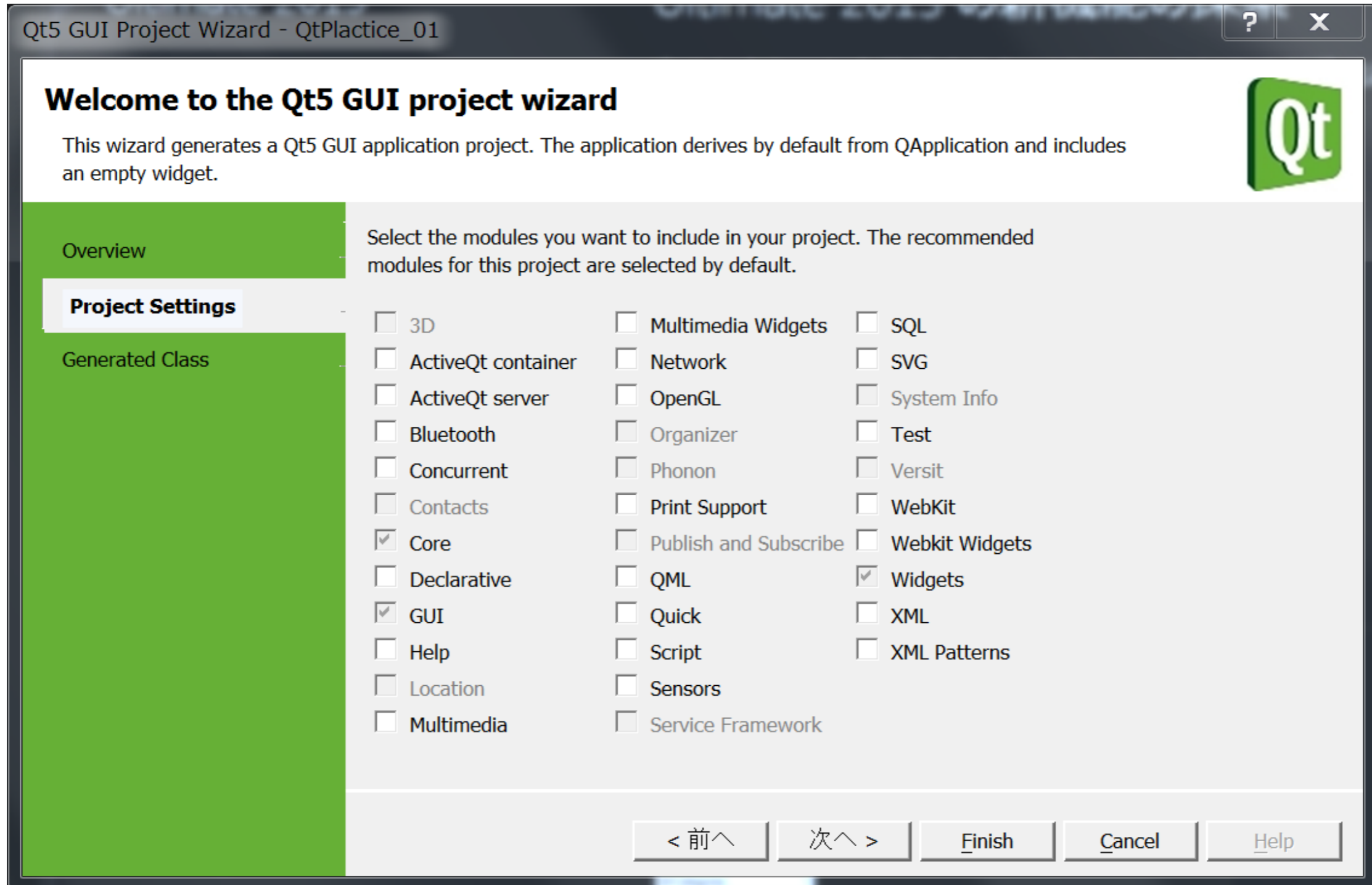


**Qt Application** を選択



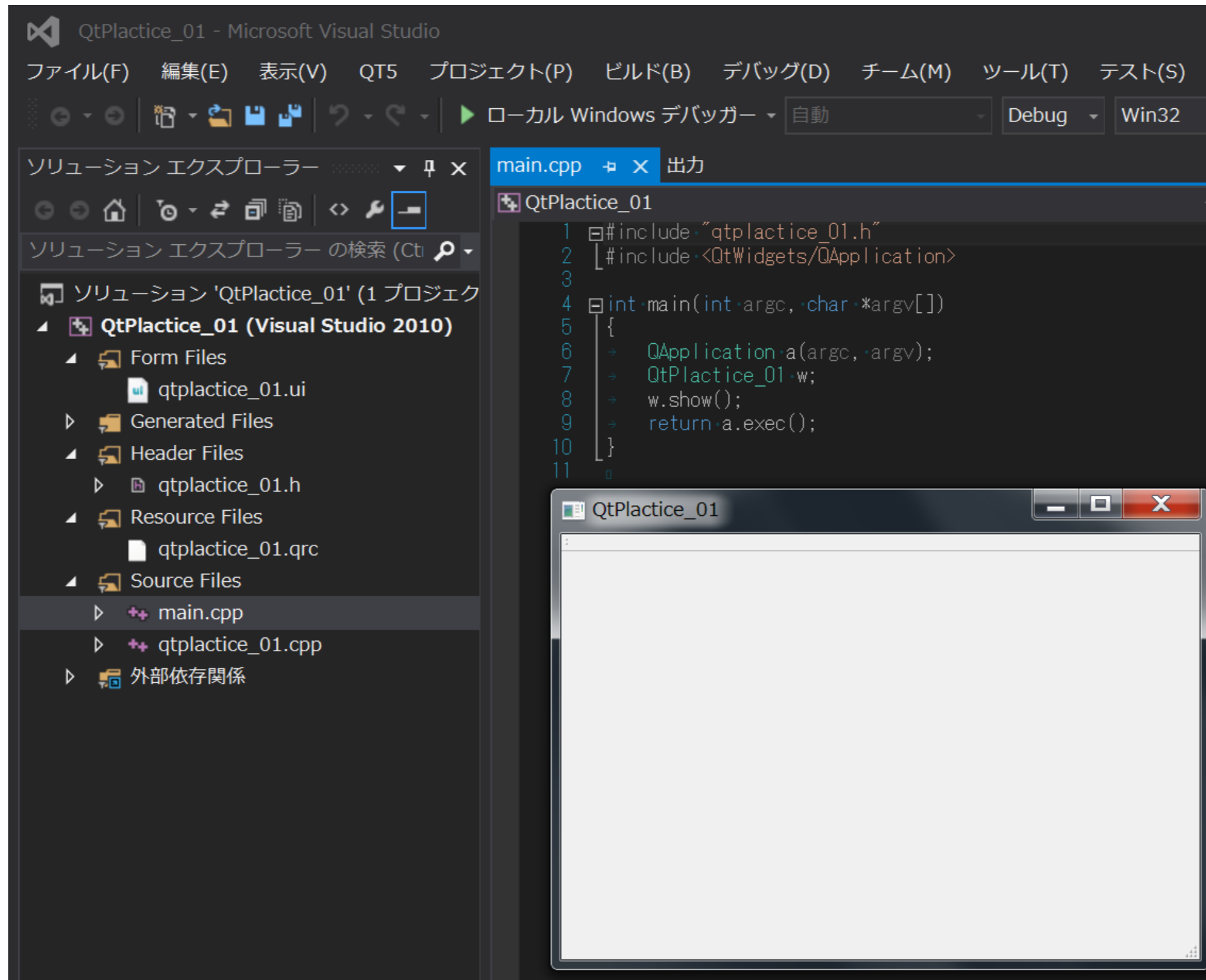
# 実際に触れてみる

- 色々オプションが存在するが、今回特につけなくてもOK



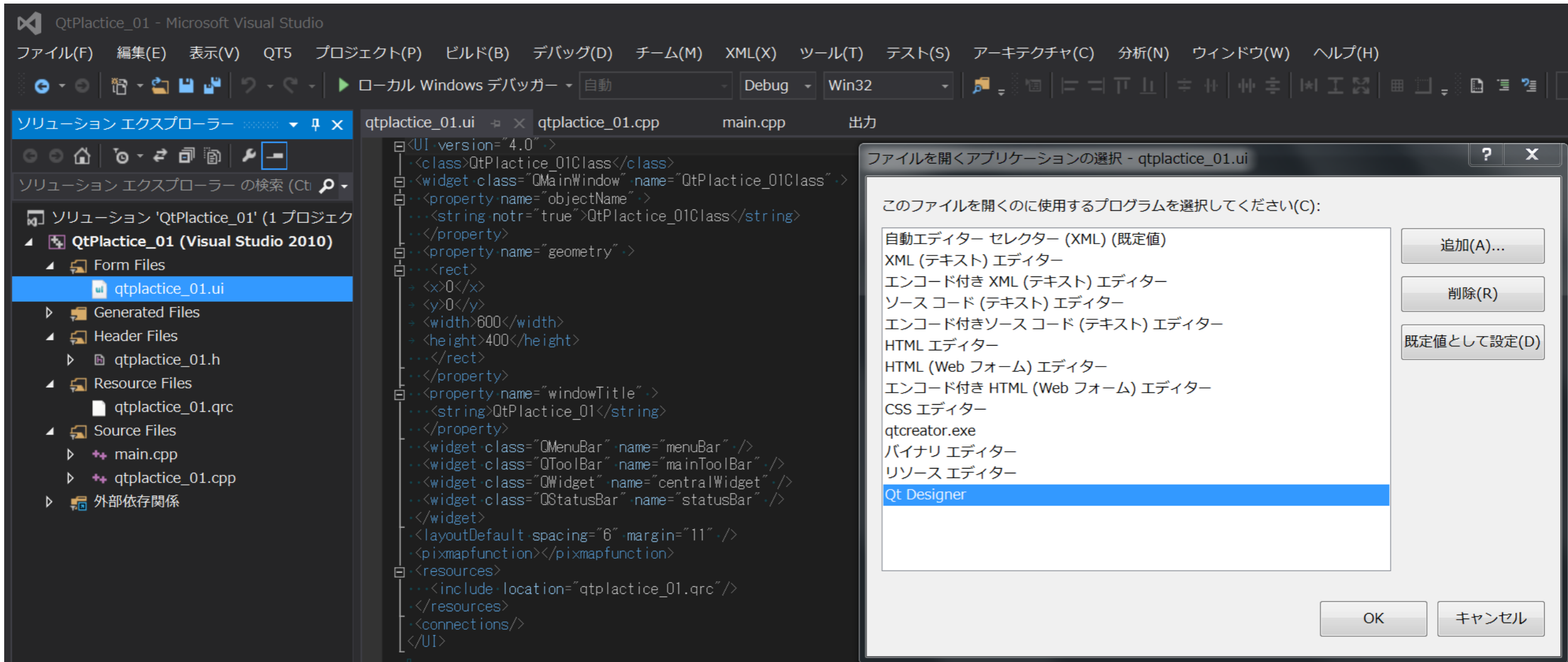
とりあえず次へ次へ送り最後にFinishする。

# 実際に触れてみる



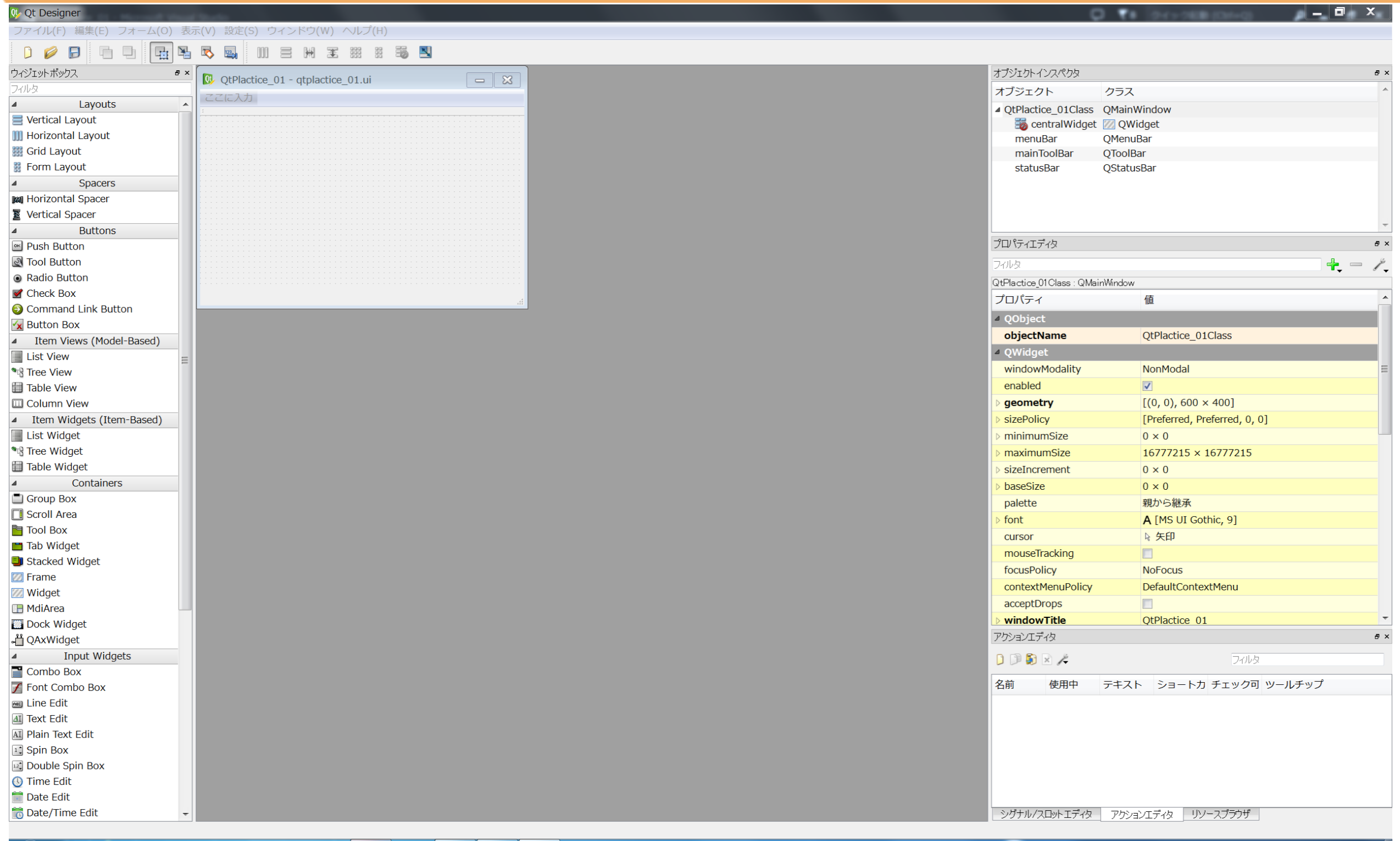
無事に実行できるとこのようなウィンドウが出現する

# 実際に触れてみる



**動作が確認できたら、ソリューションエクスプローラ内にある、.uiファイルを右クリックして図のように開くアプリケーションをQt Designer に既定値として設定した上でOKを押すと...**

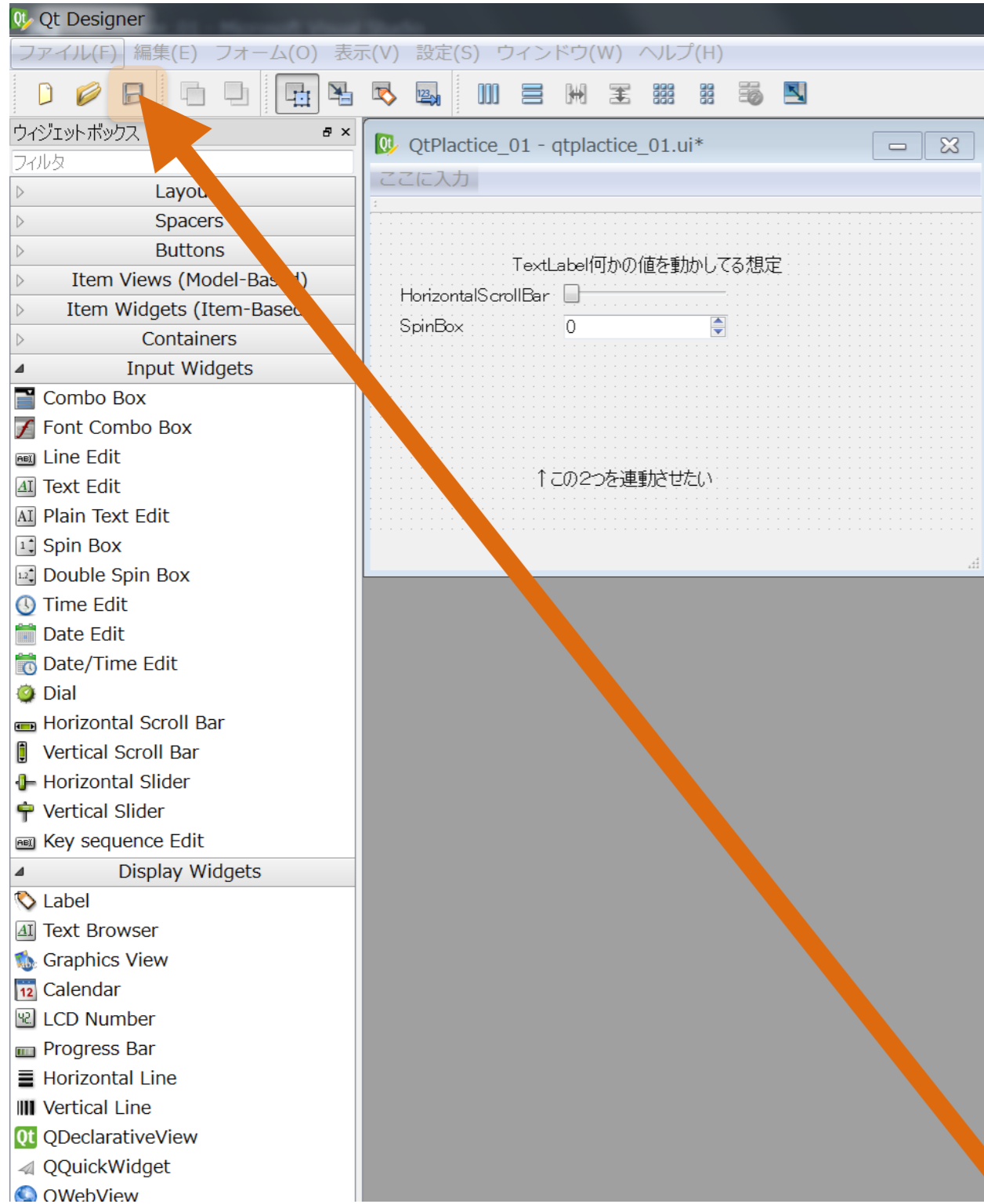
# 実際に触れてみる



なんと！C#のようなGUIエディターが出現！

一体何が起きてるんだ...

# 実際に触れてみる

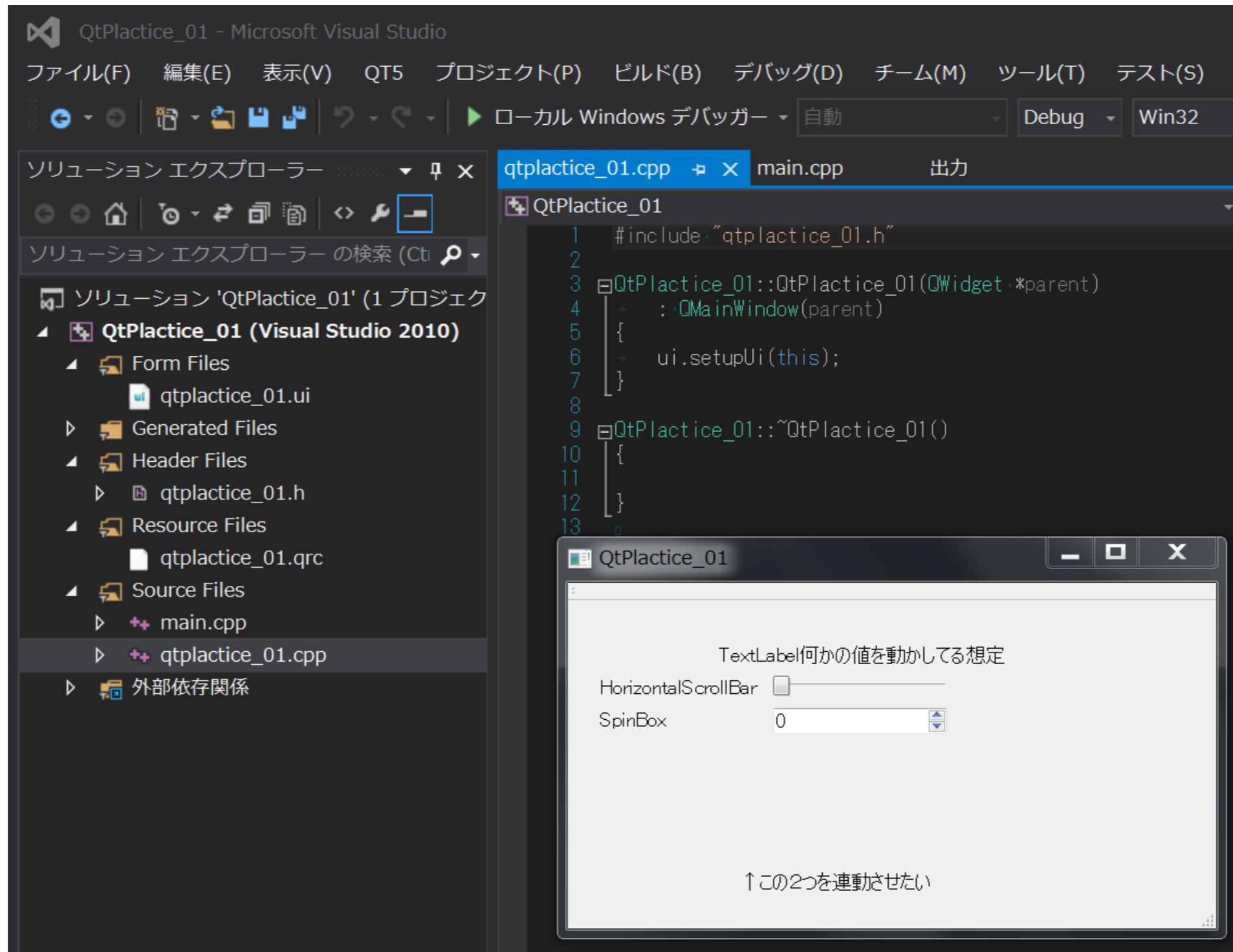


おもむろにいじってみる。  
左にあるアイコンは  
ドラッグしてウィンドウに  
いれられる。  
まずは図に従い、  
**HorizontalScrollBarと  
SpinBox,**  
**Label**等を入れてみる。

問題なければ、最後に必ず、  
保存をすること。  
(ここで保存)



# 実際に触れてみる



**ビルドするとウィンドウの設定が反映されているのを確認**

# 実際に触れてみる

## ・シグナルスロットエディター

ファイル(F) 編集(E) フォーム(O) 表示(V) 設定(S) ウィンドウ(W) ヘルプ(H)

Widget Box

- Layouts
- Spacers
- Buttons
- Item Views (Model-Based)
- Item Widgets (Item-Based)
- Containers
- Input Widgets
  - Combo Box
  - Font Combo Box
  - Line Edit
  - Text Edit
  - Plain Text Edit
  - Spin Box
  - Double Spin Box

QtPlactice\_01 - qtplactice\_01.ui

ここに入力

### ドラッグするとこんな感じに変化

TextLabel何かの値を動かしてる想定

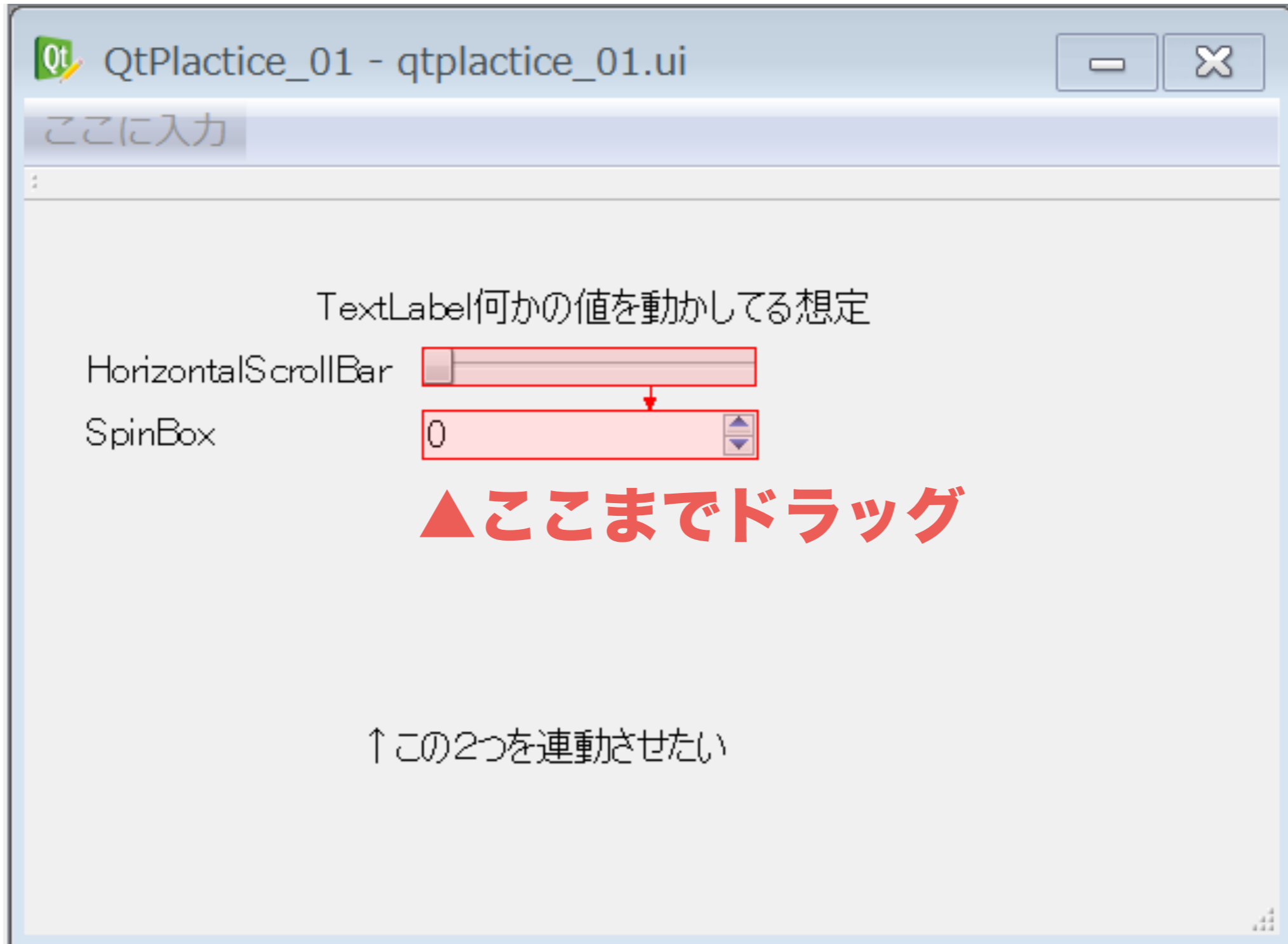
HorizontalScrollBar

SpinBox 0

↑この2つを連動させたい

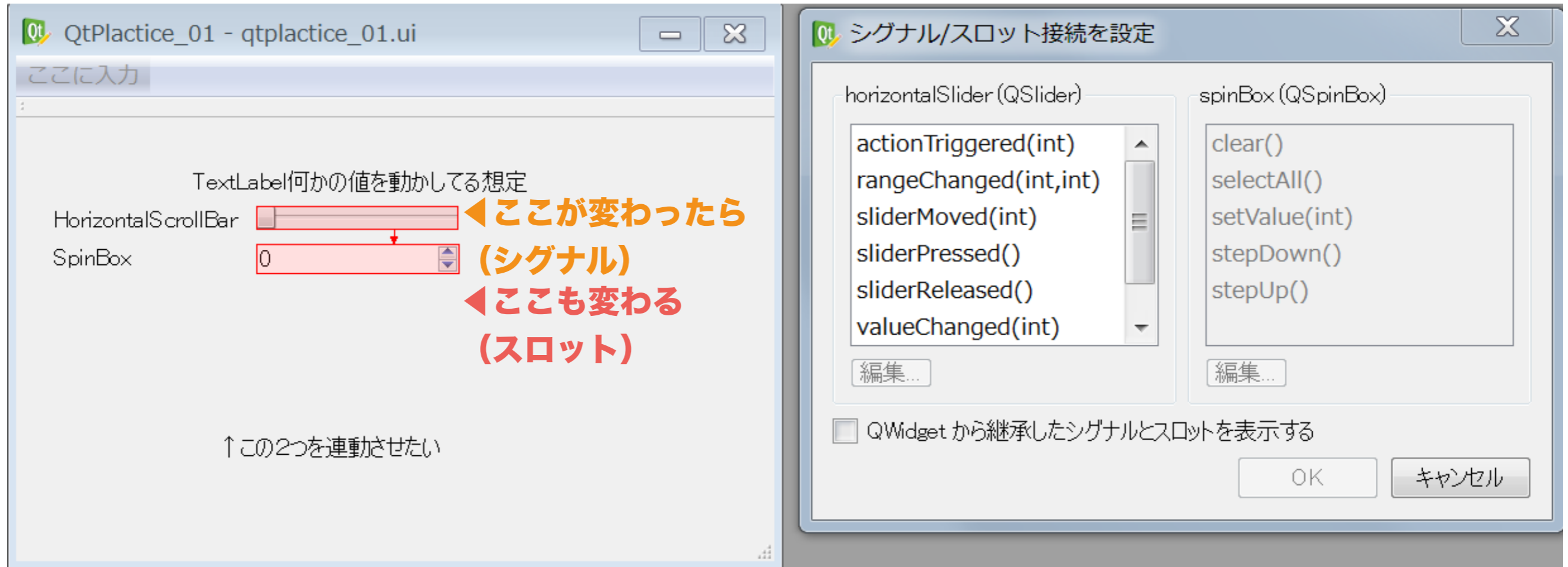
**このゲージがかわったらスピンの値と連動させたい**

# 実際に触れてみる



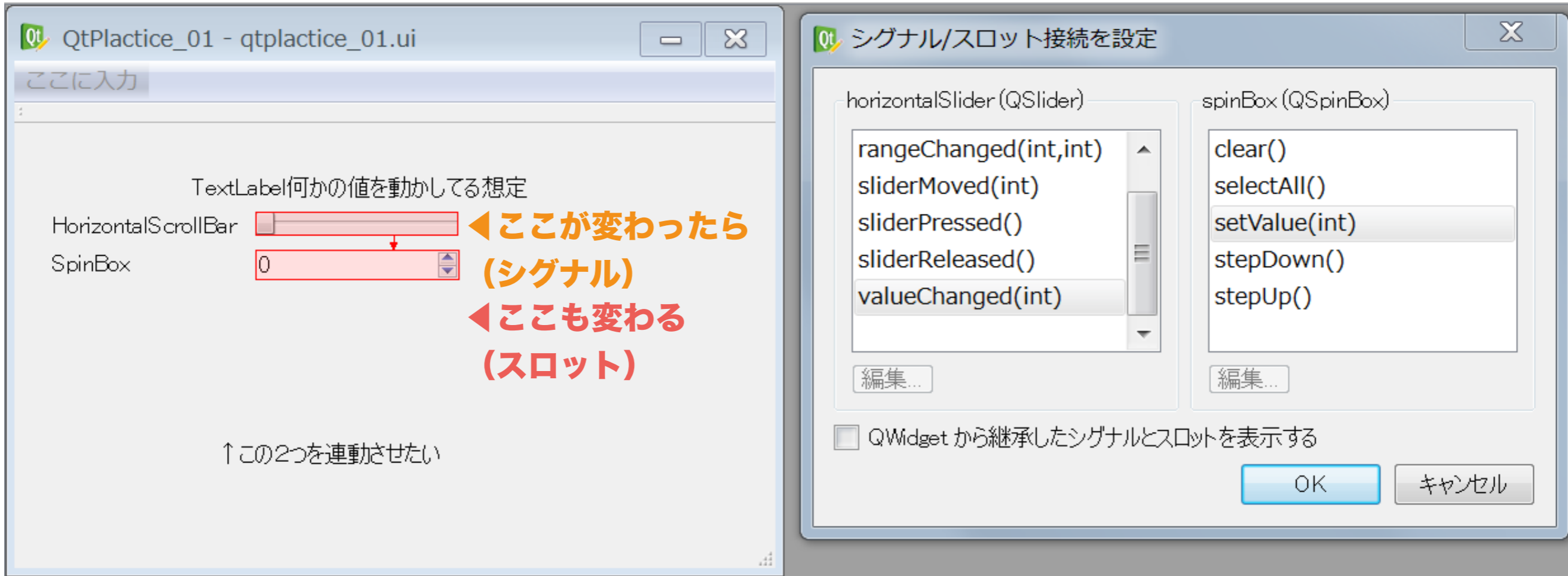
図に従い、HorizontalScrollBar=>SpinBoxまでドラッグしてきます

# 実際に触れてみる



すると図のように何やら不思議なウィンドウが出現します。  
以下にも怪しいです。続けましょう。

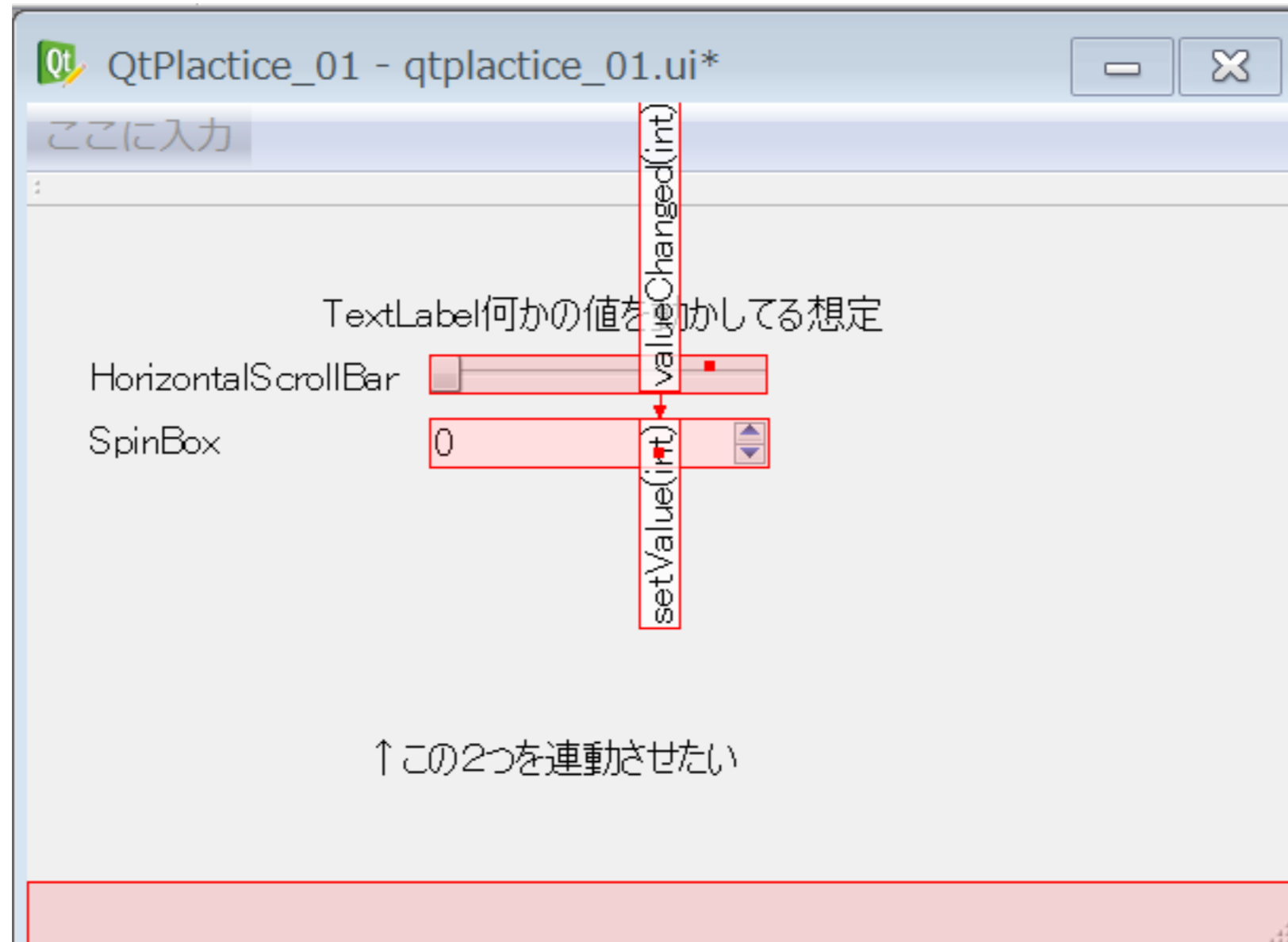
# 実際に触れてみる



図のようにシグナル(呼び出される条件)を  
**valueChanged(int)** :値が変わった時に設定して、  
スロット(呼び出された結果おこる現象) を  
**setValue(int)**:その値を入れるに設定します。

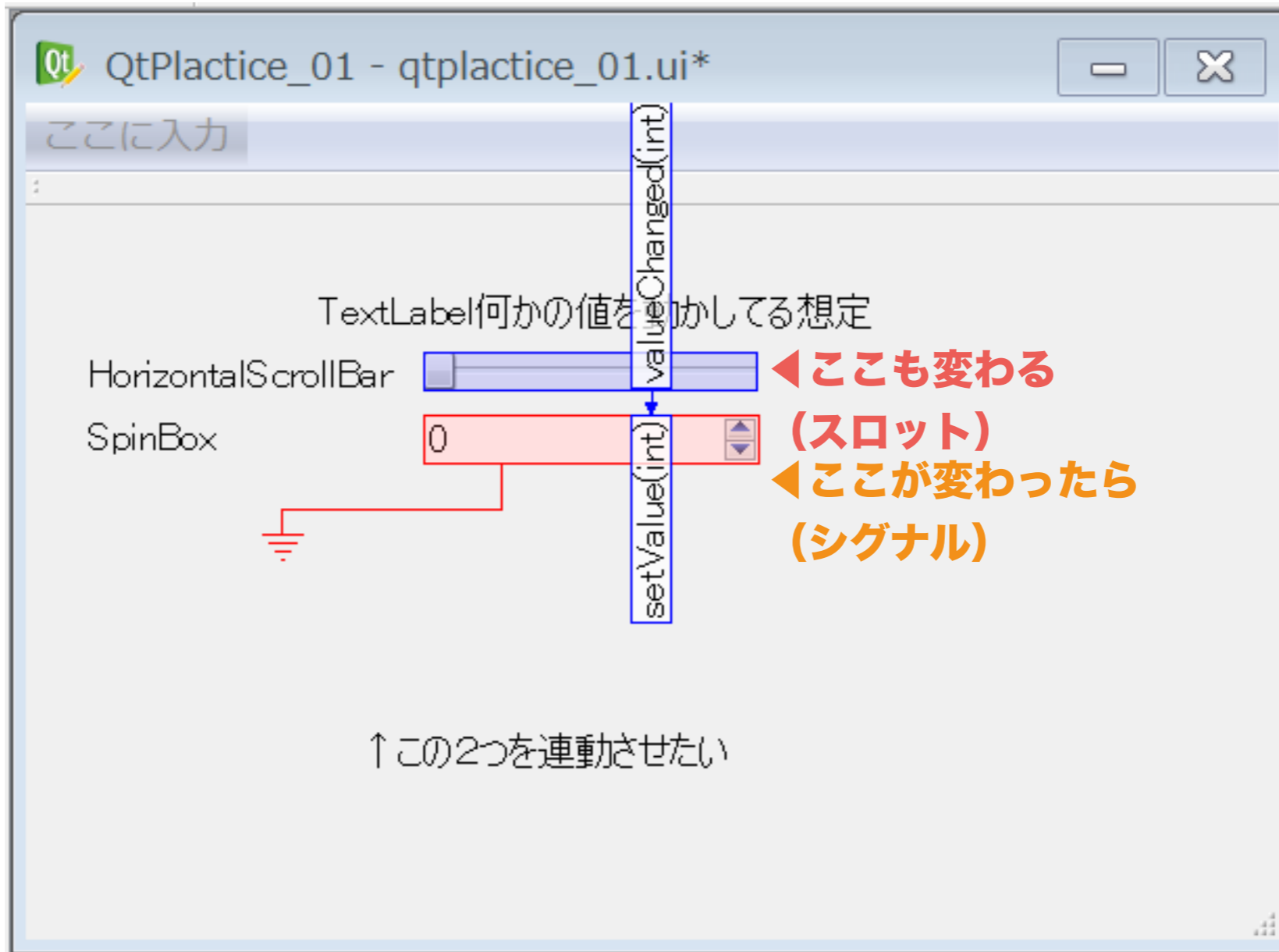


# 実際に触れてみる



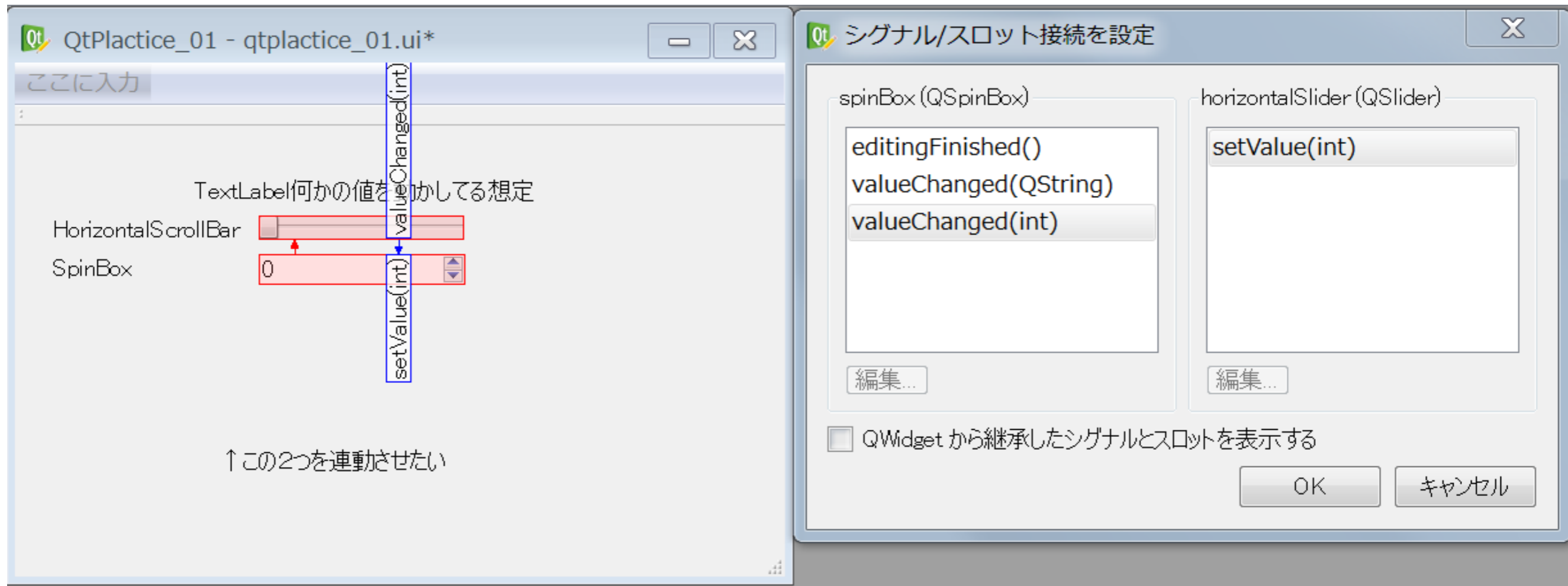
**謎のマークが追加されました。しかしこれではScrollBarが変化したときにしか値が連動しないため、逆側にも同様にシグナルとスロットを設定します。**

# 実際に触れてみる



**今度は下のSpinBoxからドラッグしてきます。**

# 実際に触れてみる



図に従ってシグナルとスロットを設定

# 実際に触れてみる

The screenshot shows the Microsoft Visual Studio interface for a project named 'QtPlactice\_01'. The menu bar includes 'ファイル(F)', '編集(E)', '表示(V)', 'QT5', 'プロジェクト(P)', 'ビルド(B)', 'デバッグ(D)', 'チーム(M)', 'ツール(T)', and 'テスト(T)'. The toolbar shows various icons, including a red box around the 'Build' icon. The Solution Explorer on the left displays the project structure:

- ソリューション 'QtPlactice\_01' (1 プロジェクト)
  - QtPlactice\_01 (Visual Studio 2010)
    - Form Files
      - qtplactice\_01.ui
    - Generated Files
    - Header Files
      - qtplactice\_01.h
    - Resource Files
      - qtplactice\_01.qrc
    - Source Files
      - main.cpp
      - qtplactice\_01.cpp
    - 外部依存関係

The Output window on the right shows the build output for 'ビルド' (Build). The output text is:

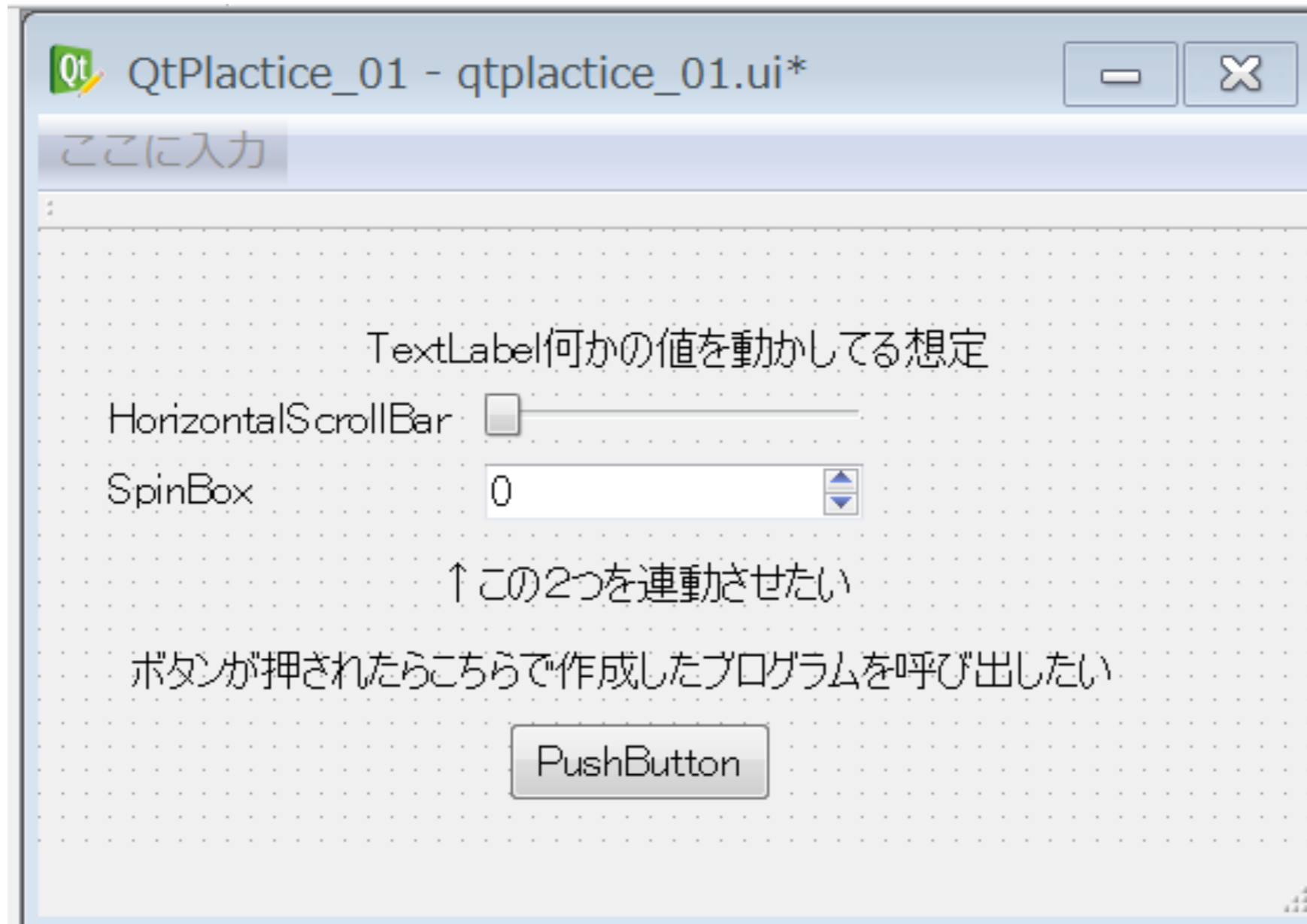
```
TextLabel何かの値を動かしてる想定
HorizontalScrollBar
SpinBox 70
```

Below the output, there is a note: '↑この2つを連動させたい' (↑I want to link these two).

**保存後、ビルドすると値の相互連動を確認できます。**

# 実際に触れてみる

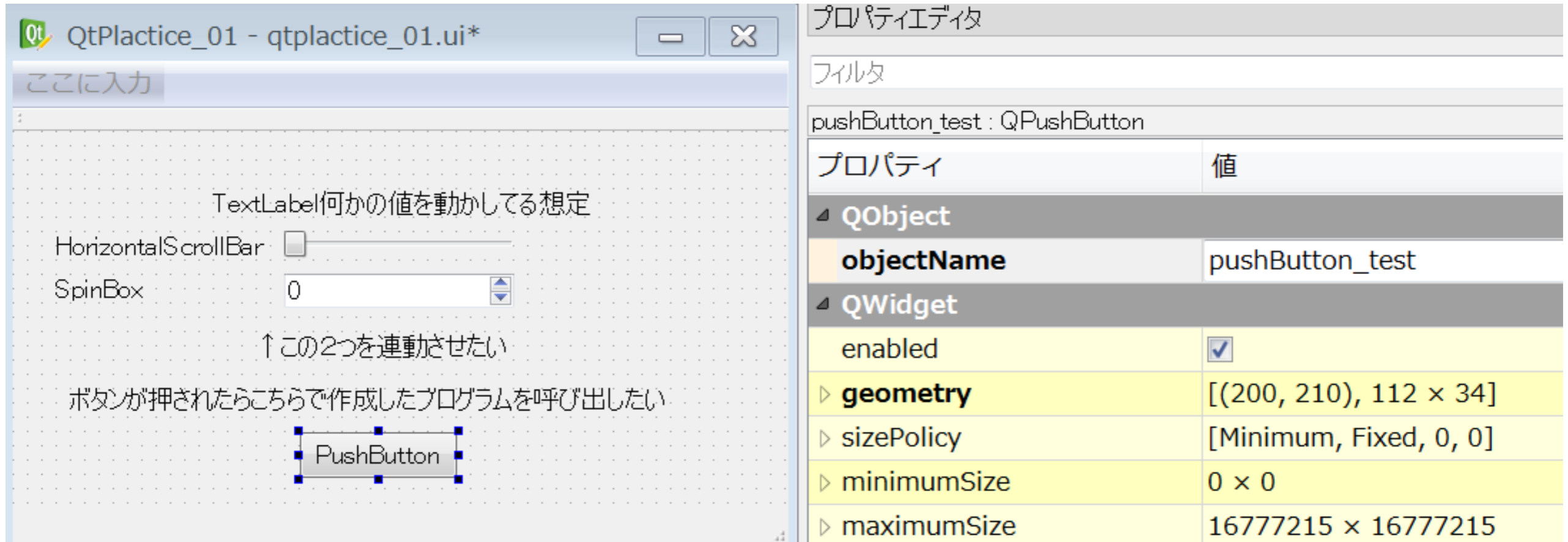
## ・シグナルスロットの自作プログラムによる制御



**次にボタンを入力するとこちらで作成したプログラムを起動させるようなものを作ります。**



# 実際に触れてみる

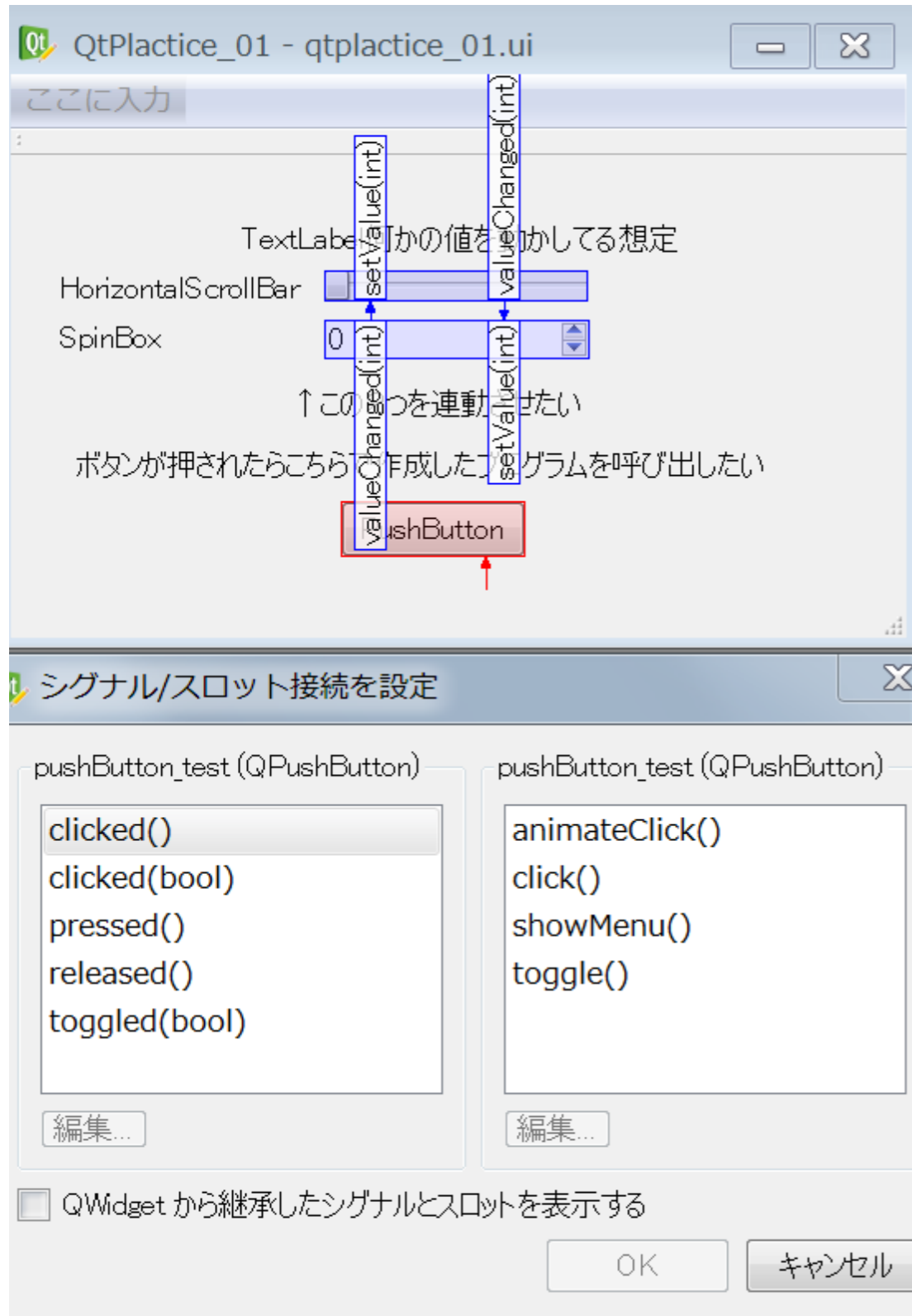


The screenshot shows the Qt Designer interface. On the left, a window titled "QtPlactice\_01 - qtplactice\_01.ui\*" contains a GUI with a text label, a horizontal scrollbar, a spin box, and a push button. The text label says "TextLabel何かの値を動かしてる想定". The spin box is set to 0. Below the spin box, there is a text label "↑この2つを連動させたい" and another text label "ボタンが押されたらこちらで作成したプログラムを呼び出したい". A push button is positioned below these labels. On the right, the "プロパティエディタ" (Property Editor) is open, showing the properties for the selected "pushButton\_test : QPushButton".

プロパティ	値
pushButton_test : QPushButton	
フィルタ	
pushButton_test : QPushButton	
プロパティ	値
▾ QObject	
objectName	pushButton_test
▾ QWidget	
enabled	<input checked="" type="checkbox"/>
▶ geometry	[(200, 210), 112 × 34]
▶ sizePolicy	[Minimum, Fixed, 0, 0]
▶ minimumSize	0 × 0
▶ maximumSize	16777215 × 16777215

**QTのGUIツールにはプロパティエディタと呼ばれるウィンドウがあります。そのObjectNameに注目してください。これがこのボタンのインスタンス名になります。この名前は任意に記述できるので、今回は図のような名前に設定しておきます。**

# 実際に触れてみる



いったん、モードを再度シグナルスロットエディットモードに設定してボタン自身をドラッグアンドドロップして、ボタンに存在している、シグナルメソッド名を確認します。

ボタンをクリックしたときに呼び出したいので、メソッド名がclicked()であることを確認したらキャンセルします。ここでは設定しません。

# 実際に触れてみる

The image shows a screenshot of the Visual Studio 2010 IDE. On the left, the Solution Explorer displays the project structure for 'QtPlactice\_01'. The project is expanded to show the 'Source Files' folder, which contains 'main.cpp' and 'qtplactice\_01.cpp'. The 'Header Files' folder contains 'qtplactice\_01.h'. The 'Form Files' folder contains 'qtplactice\_01.ui'. The 'Generated Files' folder is also visible. The main editor window shows the source code for 'qtplactice\_01.h'. The code is as follows:

```
1 #ifndef QTPLACTICE_01_H
2 #define QTPLACTICE_01_H
3
4 #include <QtWidgets/QMainWindow>
5 #include "ui_qtplactice_01.h"
6
7 class QtPlactice_01 : public QMainWindow
8 {
9     Q_OBJECT
10
11 public:
12     QtPlactice_01(QWidget *parent = 0);
13     ~QtPlactice_01();
14
15 private:
16     Ui::QtPlactice_01Class ui;
17
18 };
19
20 #endif // QTPLACTICE_01_H
21
```

**プロジェクト作成時に生成されたヘッダファイルを確認**

# 実際に触れてみる

```
qtpractice_01.h* x qtpractice_01.cpp main.cpp 出力
QtPlactice_01 QtPlactice_01
1 #ifndef QTPLACTICE_01_H
2 #define QTPLACTICE_01_H
3
4 #include <QtWidgets/QMainWindow>
5 #include "ui_qtpractice_01.h"
6
7 class QtPlactice_01 : public QMainWindow
8 {
9     Q_OBJECT
10
11 public:
12     QtPlactice_01(QWidget *parent = 0);
13     ~QtPlactice_01();
14     Ui::QtPlactice_01Class ui;
15 private:
16
17 private slots:
18 };
19
20 #endif // QTPLACTICE_01_H
21
```

あまり良くはないですが、説明の都合で一度、uiメンバ変数をパブリックメンバに移します。

# 実際に触れてみる

```
qtplactice_01.h  x qtplactice_01.cpp  main.cpp  出力
QtPlactice_01  (グローバル スコープ)
4  #include <QtWidgets/QMainWindow>
5  #include "ui_qtplactice_01.h"
6
7  class QtPlactice_01 : public QMainWindow
8  {
9  →   Q_OBJECT
10
11  public:
12  →   QtPlactice_01 (QWidget *parent = 0);
13  →   ~QtPlactice_01 ();
14  →   Ui::QtPlactice_01Class ui;
15  private:
16  →
17  →   private slots:
18  →   →   // ボタンを押した
19  →   →   void on_pushButton_test_clicked();
20  };
21
22 #endif // QTPLACTICE_01_H
23
```

おもむろに**private slots:** と記述して、  
その下に**on\_オブジェクト名\_シグナル名(シグナル引数型)**で  
メソッドを宣言します。



# 実際に触れてみる

```
qtplactice_01.h  x qtplactice_01.cpp  main.cpp  出力
QtPlactice_01  (グローバル スコープ)
4  #include <QtWidgets/QMainWindow>
5  #include "ui_qtplactice_01.h"
6
7  class QtPlactice_01 : public QMainWindow
8  {
9  →   Q_OBJECT
10
11  public:
12  →   QtPlactice_01 (QWidget *parent = 0);
13  →   ~QtPlactice_01 ();
14  →   Ui::QtPlactice_01Class ui;
15  private:
16  →
17  →   private slots:
18  →     →   // ボタンを押した
19  →     →   void on_pushButton_test_clicked();
20  };
21
22 #endif // QTPLACTICE_01_H
23
```

※ **private slots:** について これはC++の文法とは関係がないため注意してください。QT独自のアクセス指定子となっています。

# 実際に触れてみる

```
qtplactice_01.h  qtplactice_01.cpp  main.cpp  出力
QtPlactice_01
1  #include "qtplactice_01.h"
2
3  QtPlactice_01::QtPlactice_01 (QWidget *parent)
4      : QMainWindow (parent)
5      {
6      →  ui.setupUi (this);
7      }
8
9  QtPlactice_01::~QtPlactice_01 ()
10     {
11     }
12
13
14  void QtPlactice_01::on_pushButton_test_clicked ()
15     {
16     →  // 適当にブレークポイント置いて
17     →  // 呼ばれたかを確認
18     →  int test = 0;
19     →  test = 1;
20     }
```

クラス定義はいつも通りのC++と一緒にです。

戻り値型 クラス名::メソッド名{ 処理 } です。

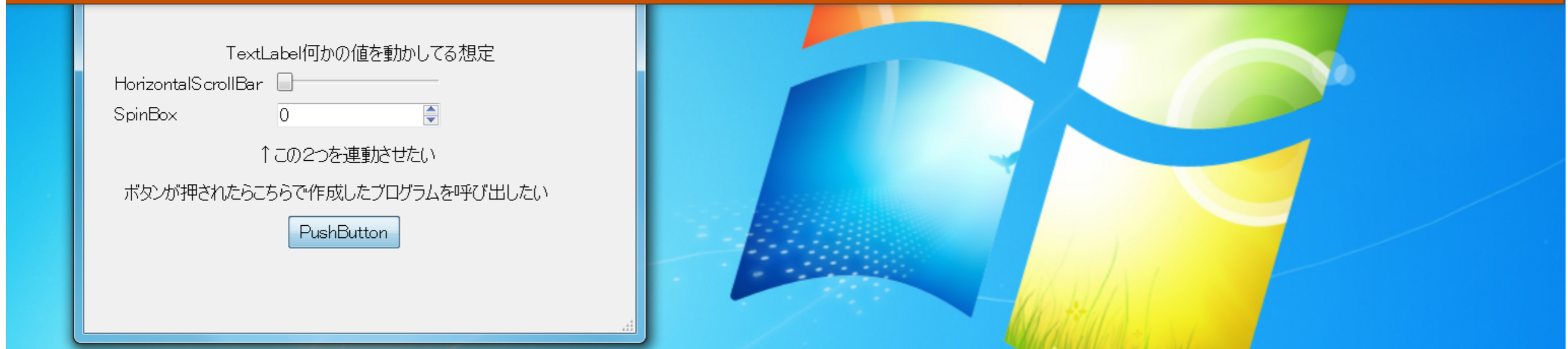
ブレークポイントを設置して呼ばれるかを確認してみます。

# 実際に触れてみる

```
qtplactice_01.h  qtplactice_01.cpp  main.cpp
QtPlactice_01    QtPlactice_01    on_pushButton_test_clicked()

13
14 void QtPlactice_01::on_pushButton_test_clicked()
15 {
16     // 適当にブレークポイント置いて
17     // 呼ばれたかを確認
18     int test = 0;
19     test = 1;
20 }
```

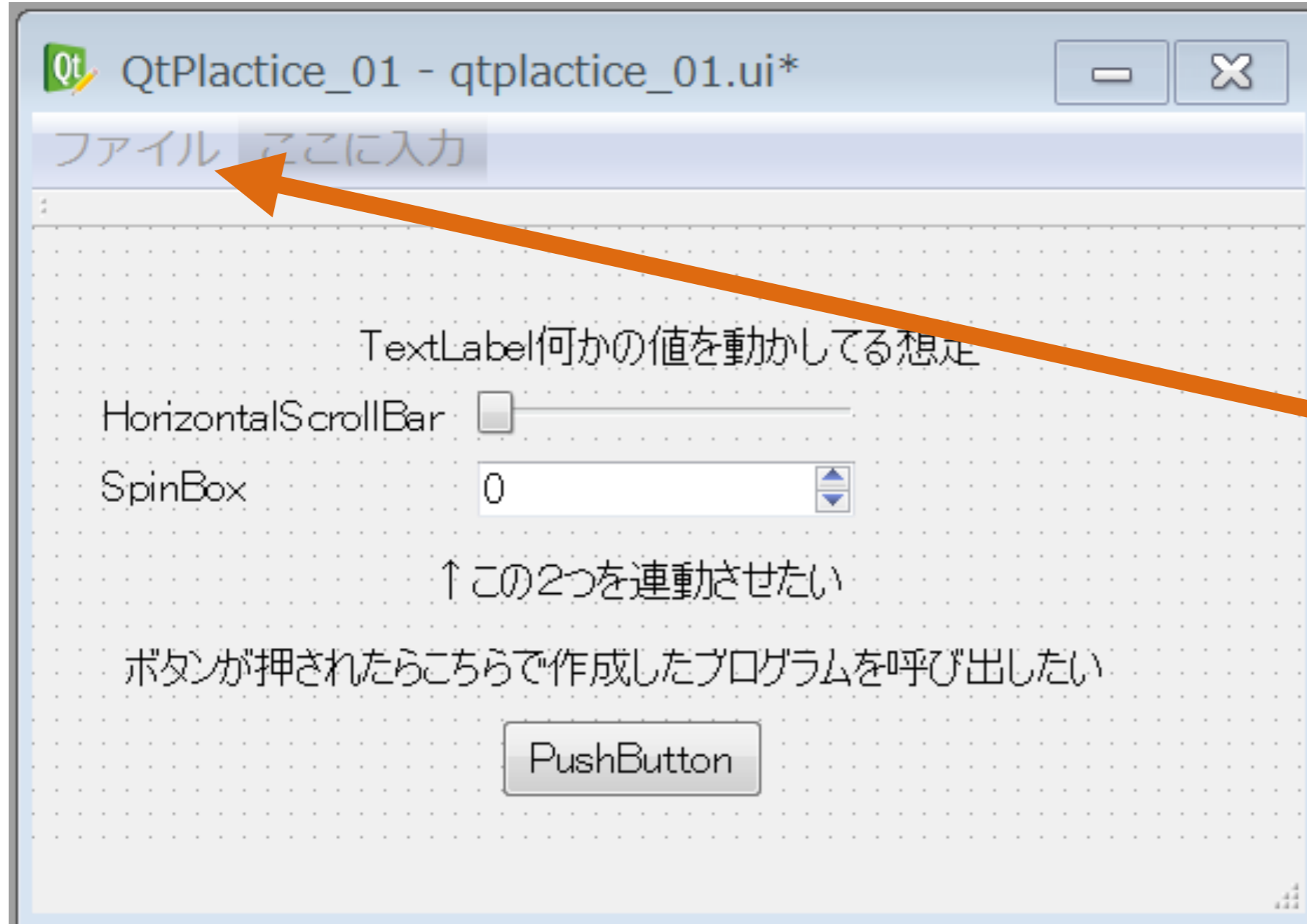
214 % 19行 1列 1文字 挿入



**確かに、ボタンを入力したことで、プログラムが実行されたことが確認できました。**

# 実際に触れてみる

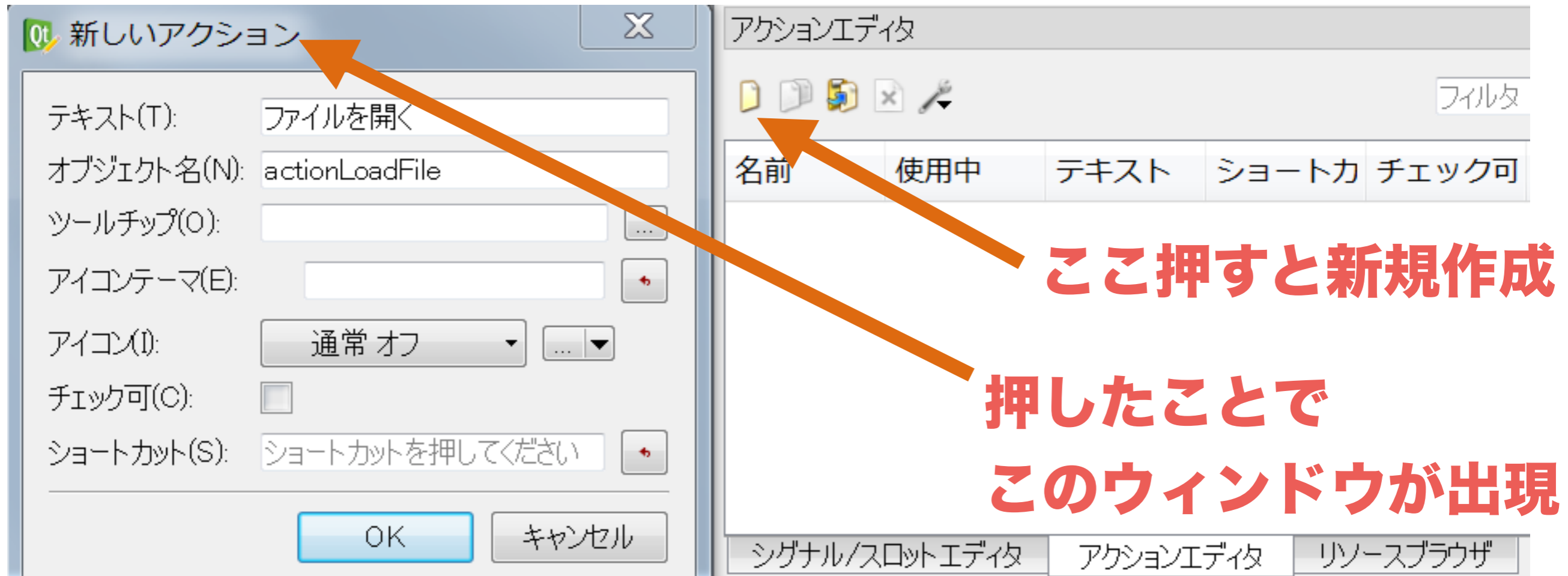
## ・ファイルメニューのシグナルスロット制御



ここは先に  
記述しておく  
選択したら  
編集できるはず

次はボタンではなくファイルメニュー等から、  
同様にプログラムを呼び出す方法について説明します。

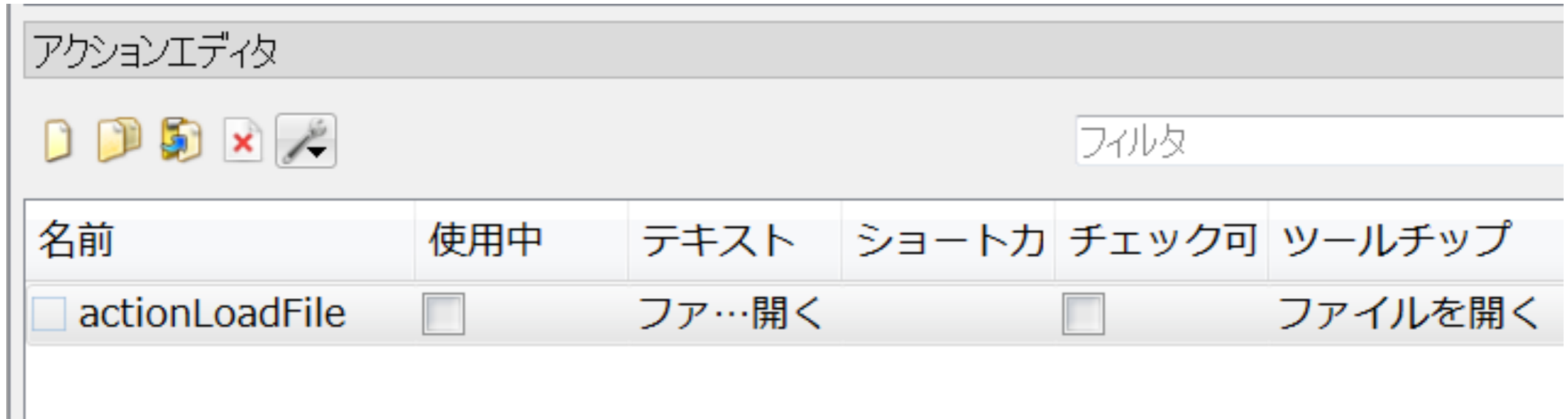
# 実際に触れてみる



ウィンドウ内にアクションエディターというのがあるので  
これを使います。ここにある新規作成アイコンを入力すると、  
ウィンドウが出現するので図のように記述してください。

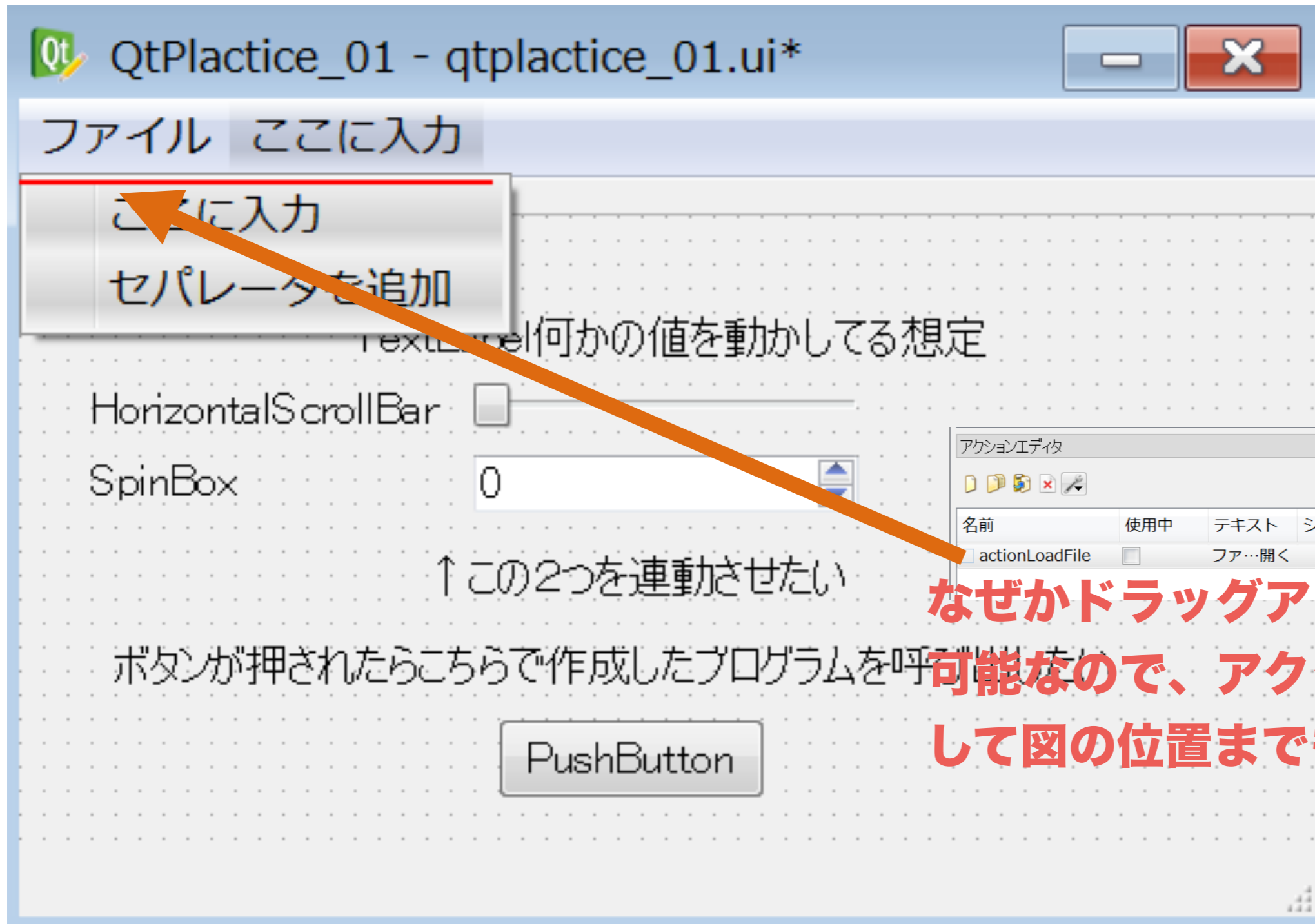


# 実際に触れてみる



**入力を確認すると、図のように新しいアクションと呼ばれる物が追加されていることが確認できます。**

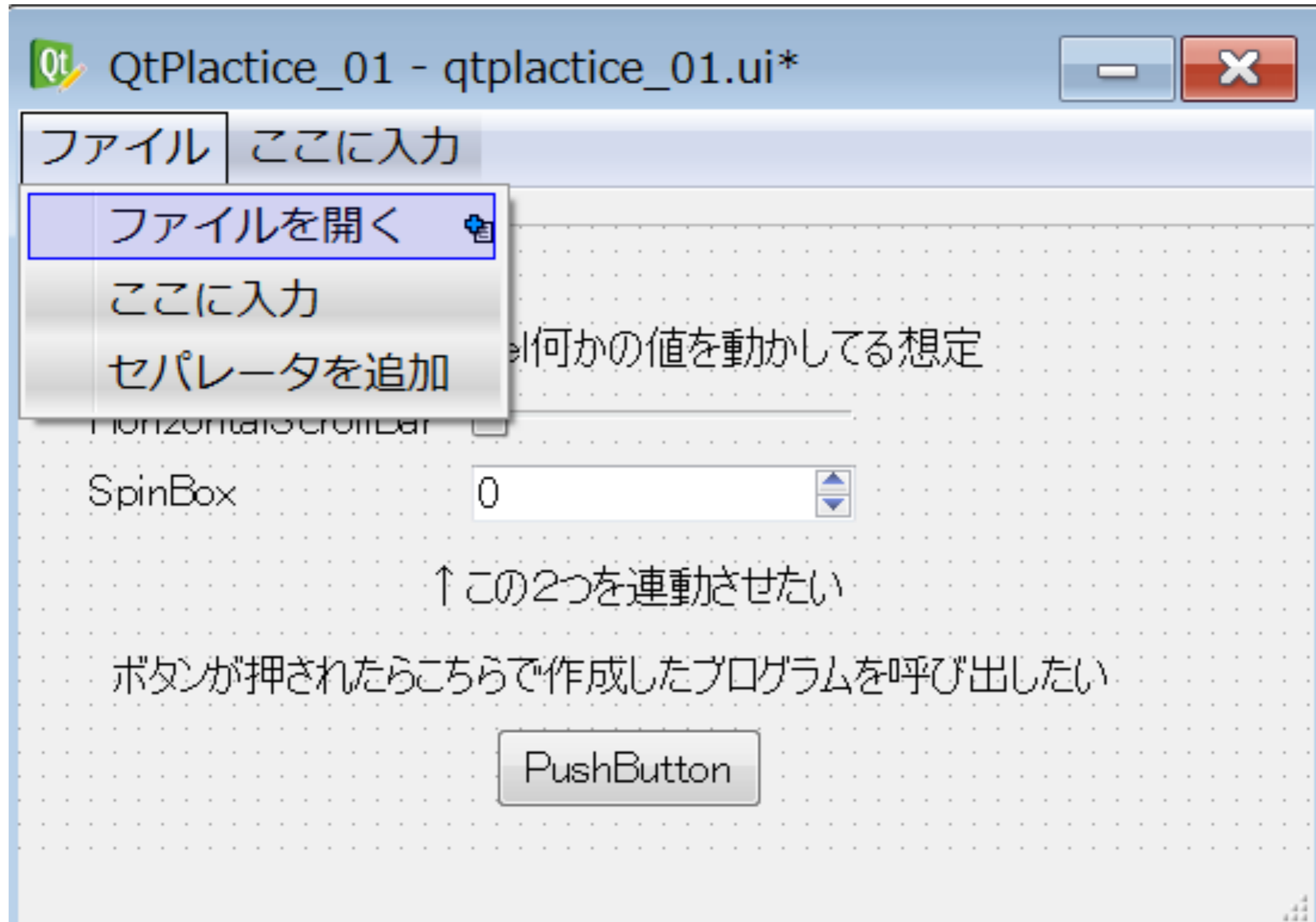
# 実際に触れてみる



**なぜかドラッグアンドドロップ可能なので、アクションをドラッグして図の位置までもってく**

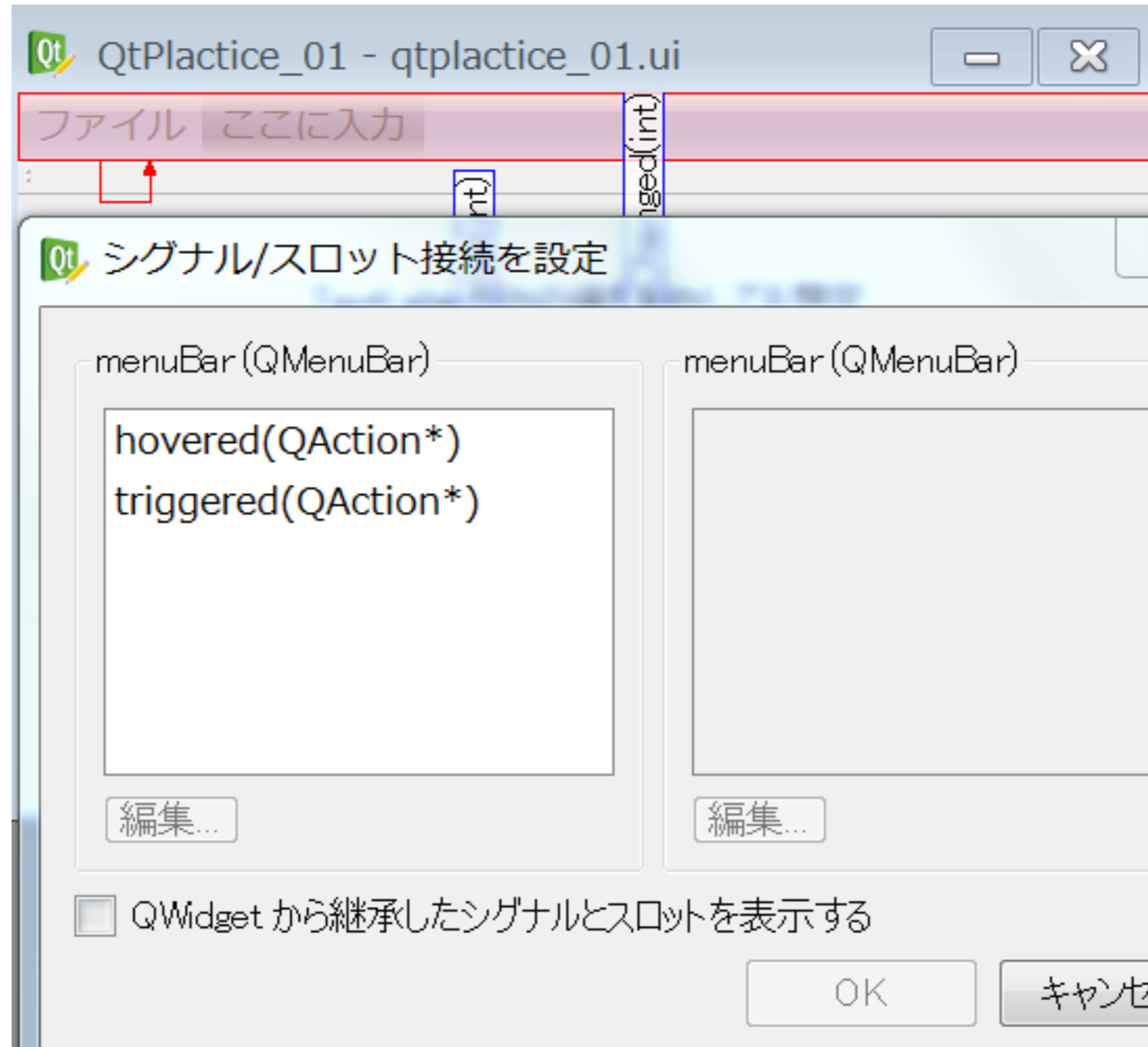
**新規作成したアクションを図の位置までドラッグしてきます。**

# 実際に触れてみる



**アクションがメニューに項目として追加されました。**

# 実際に触れてみる



**呼び出されるシグナルメソッド名を確認しておきます。  
押されたときはtriggered(今回は省略可能)のようです。**

# 実際に触れてみる

```
QtPlactice_01 QtPlactice_01
4 #include <QtWidgets/QMainWindow>
5 #include "ui_qtplactice_01.h"
6
7 class QtPlactice_01 : public QMainWindow
8 {
9     Q_OBJECT
10
11 public:
12     QtPlactice_01(QWidget *parent = 0);
13     ~QtPlactice_01();
14     Ui::QtPlactice_01Class ui;
15 private:
16
17 private slots:
18     // ボタンを押した
19     void on_pushButton_test_clicked();
20
21     // ファイルを開く
22     void on_actionLoadFile_triggered();
23 };
24
25 #endif // QTPLACTICE_01_H
26
```

ファイルを開いたとき用の処理を追加します。ボタンと同様 **on\_オブジェクト名\_シグナル名(シグナル引数型)** でメソッドを宣言します。

# 実際に触れてみる

```
QtPlactice_01 QtPlactice_01
4 #include <QtWidgets/QMainWindow>
5 #include "ui_qtplactice_01.h"
6
7 class QtPlactice_01 : public QMainWindow
8 {
9     Q_OBJECT
10
11 public:
12     QtPlactice_01(QWidget *parent = 0);
13     ~QtPlactice_01();
14     Ui::QtPlactice_01Class ui;
15 private:
16
17 private slots:
18     // ボタンを押した
19     void on_pushButton_test_clicked();
20
21     // ファイルを開く
22     void on_actionLoadFile_triggered();
23 };
24
25 #endif // QTPLACTICE_01_H
26
```

※命名規則は決まっておき一文字でも間違うと呼び出されない。



# 実際に触れてみる

```
13
14 void QtPlactice_01::on_pushButton_test_clicked()
15 {
16     → // 適当にブレークポイント置いて
17     → // 呼ばれたかを確認
18     → int test = 0;
19     → test = 1;
20 }
21
22 // ファイルを開く
23 void QtPlactice_01::on_actionLoadFile_triggered()
24 {
25     → // 適当にブレークポイント置いて
26     → // 再度呼ばれたかを確認
27     → int test2 = 0;
28     → test2 = 1;
29 }
```

定義はボタンと同様。再度ブレークポイントで確認を行う

# 実際に触れてみる

The screenshot shows the Qt Creator IDE with the following code in `QtPlactice_01.cpp`:

```
14 void QtPlactice_01::on_pushButton_test_clicked()
15 {
16     // 適当にブレークポイント置いて
17     // 呼ばれたかを確認
18     int test = 0;
19     test = 1;
20 }
21
22 // ファイルを開く
23 void QtPlactice_01::on_actionLoadFile_triggered()
24 {
25     // 適当にブレークポイント置いて
26     // 再度呼ばれたかを確認
27     int test2 = 0;
28     test2 = 1;
29 }
```

The Qt widget window is open, showing a "ファイル" (File) menu with "ファイルを開く" (Open File) selected. The window contains a text label, a horizontal scrollbar, a spin box with the value 0, and a push button labeled "PushButton".

The screenshot shows the Qt Creator IDE with the following code in `QtPlactice_01.cpp`:

```
21
22 // ファイルを開く
23 void QtPlactice_01::on_actionLoadFile_triggered()
24 {
25     // 適当にブレークポイント置いて
26     // 再度呼ばれたかを確認
27     int test2 = 0;
28     test2 = 1;
29 }
```

The Qt widget window is open, showing a "ファイル" (File) menu with "ファイルを開く" (Open File) selected. The window contains a text label, a horizontal scrollbar, a spin box with the value 0, and a push button labeled "PushButton".

**無事に呼び出されたことが確認できました。**

# 目次

---

- **なんでやるうと思ったのか**
- **どんなことが出来るのか**
- **開発環境の導入**
- **Qtとは、メリットとデメリット**
- **実際に触れてみる**
- **実践的に使うための準備**
- **DirectXのレンダラーをツールと連動させる**
- **ツール上でレンダリングする**
- **自分のフレームワーク上にレンダリングする**
- **入力とレンダリングの間**
- **配布時の注意点**
- **さらに実践的なQtツール制作上のノウハウについて**

## 実践的に使うための準備

これで、やりたい放題できるかに見えましたが、現実には甘くはないです。DirectXをまだ組み込んでいません。そして、これについて説明されたドキュメントはほぼなく一子相伝状態になってしまっています。今回はDirectX9の導入方法について、解説します。



さあ、死ぬがよい！

# 実践的に使うための準備

```
出力 qtplactice_01.h x qtplactice_01.cpp main.cpp
QtPlactice_01 (グローバルスコープ)
10
11 public:
12   → QtPlactice_01 (QWidget *parent = 0);
13   → ~QtPlactice_01 ();
14   → Ui::QtPlactice_01Class ui;
15 private:
16   →
17 private slots:
18   → // ボタンを押した
19   → void on_pushButton_test_clicked ();
20   →
21   → // ファイルを開く
22   → void on_actionLoadFile_triggered ();
23 };
24
25 // DirectXをレンダリングするための準備
26 // 本当はここでおくのはよろしくないが説明の都合
27 HINSTANCE → GetHinstance (void);
28 HWND → GetHwnd (void);
29 HWND → GetQtHwnd (void);
30 bool → GetWindowMode (void);
31 void → SetHinstance (HINSTANCE hInstance);
32 void → SetHwnd (HWND hwnd);
33 void → SetQtHwnd (HWND hwnd);
34 void → SetWindowMode (bool flag);
35
36 #endif // QTPLACTICE_01_H
```

下準備として、次のようなプロトタイプ宣言をしてください。

# 実践的に使うための準備

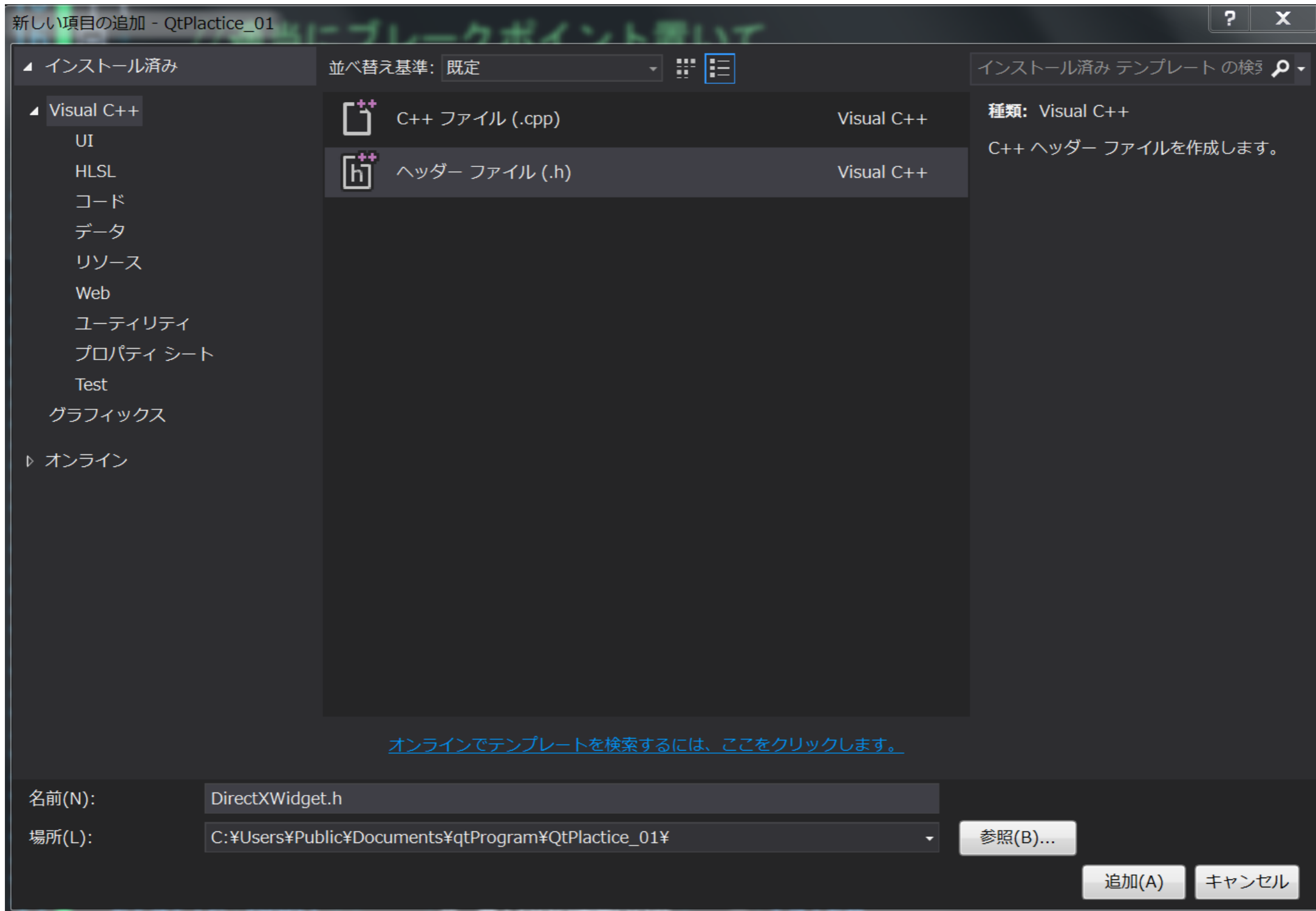
```
32 //→ グローバル変数
33 -----
34 static HINSTANCE g_Hinstance = nullptr;
35 static HWND g_Hwnd = nullptr;
36 static HWND g_QtHwnd = nullptr;
37 static bool g_WindowMode = false;
38
39 /*-----
40 //→ 関数定義
41 -----
42 HINSTANCE GetHinstance(void)
43 {
44     return g_Hinstance;
45 }
46
47 HWND GetHwnd(void)
48 {
49     return g_Hwnd;
50 }
51
52 HWND GetQtHwnd(void)
53 {
54     return g_QtHwnd;
55 }
56 bool GetWindowMode(void)
57 {
58     return g_WindowMode;
59 }
```

```
出力 qtpractice_01.h qtpractice_01.cpp main.cpp
QtPractice_01 (グローバルスコープ)
53 {
54     return g_QtHwnd;
55 }
56 bool GetWindowMode(void)
57 {
58     return g_WindowMode;
59 }
60
61 void SetHinstance(HINSTANCE hInstance)
62 {
63     g_Hinstance = hInstance;
64 }
65
66 void SetHwnd(HWND Hwnd)
67 {
68     g_Hwnd = Hwnd;
69 }
70
71 void SetQtHwnd(HWND Hwnd)
72 {
73     g_QtHwnd = Hwnd;
74 }
75
76 void SetWindowMode(bool flag)
77 {
78     g_WindowMode = flag;
79 }
```

中身は単純なグローバルなセッター、ゲッターになっています。



# 実践的に使うための準備



記述できたら、新しいファイルを作成します。図の通りです。

# 実践的に使うための準備

```
DirectXWidget.cpp  DirectXWidget.h* 出力  qtpractice_01.h  qtpractice_01.cpp  main.cpp
QtPactice_01
CDirectXWidget
1  #pragma once
2  #ifndef _DIRECTX_WIDGET_H_
3  | #define _DIRECTX_WIDGET_H_
4  | #include <QtWidgets/Qwidget>
5  | #include <QTimer>
6  | class CDirectXWidget : public QWidget {
7  |     → Q_OBJECT
8  |     public:
9  |     → CDirectXWidget(QWidget *parent = 0);
10 |     → ~CDirectXWidget();
11 |     → void Initialize(void);
12 |     → void Finalize(void);
13 |     → void Update(void);
14 |     → void Draw(void);
15 |     → void SetUpdateFlag(bool flag);
16 |     → bool GetUpdateFlag(void);
17 |     private:
18 |     → bool m_bUpdateFlag; → //> 更新フラグ
19 |     → QTimer m_Timer; → //> タイマー
20 | };
21 #endif
```

図に従ってクラス定義を記述してください。

# 実践的に使うための準備

```
DirectXWidget.cpp  DirectXWidget.h  出力  qtpractice_01.h  qtpractice_01.cpp  main.cpp
QtPactice_01      (グローバル スコープ)
1  #pragma once
2  #ifndef _DIRECTX_WIDGET_H_
3  | #define _DIRECTX_WIDGET_H_
4  | #include <QtWidgets/Qwidget>
5  | #include <QTimer>
6  |
7  | //DirectX組み込み
8  | #include <d3d9.h>
9  | #include <d3dx9.h>
10 | #pragma comment(lib, "d3d9.lib")
11 | #pragma comment(lib, "d3dx9.lib")
12 | #pragma comment(lib, "dxguid.lib")
13 | #pragma comment(lib, "dinput8.lib")
14 | #pragma comment(lib, "winmm.lib")
15 |
16 | class CDirectXWidget : public QWidget {
17 |     → Q_OBJECT
18 |     public:
19 |     → CDirectXWidget(QWidget *parent = 0);
20 |     → ~CDirectXWidget();
21 |     → void Initialize(void);
```

**QTは問題なくDirectXと併用可能です。  
ライブラリのリンクとインクルードをします。**

# 実践的に使うための準備

```
DirectXWidget.cpp  DirectXWidget.h*  出力  qtpractice_01.h  qtpractice_01.cpp  main.cpp
QtPractice_01  CDirectXWidget
16 class CDirectXWidget : public QWidget {
17     Q_OBJECT
18 public:
19     CDirectXWidget(QWidget *parent = 0);
20     ~CDirectXWidget();
21     void Initialize(void);
22     void Finalize(void);
23     void Update(void);
24     void Draw(void);
25     void SetUpdateFlag(bool flag);
26     bool GetUpdateFlag(void);
27 private:
28     bool m_bUpdateFlag; //更新フラグ
29     QTimer m_Timer; //タイマー
30     LPDIRECT3D9 m_pD3D; //DirectX9オブジェクト
31     LPDIRECT3DDEVICE9 m_pDevice; //DirectX9デバイスオブジェクト
32 };
33 #endif
```

当然デバイスとDirectX9オブジェクトが必要なので用意しましょう。

# 実践的に使うための準備

```
DirectXWidget.cpp  ×  DirectXWidget.h  出力  qtplactice_01.h  qtplactice_01.cpp  main.cpp
QtPactice_01  →  CDirectXWidget  GetUpdateFlag(void)
1  #include "DirectXWidget.h"
2
3  CDirectXWidget::CDirectXWidget (QWidget *parent = 0)
4  {
5
6  }
7  CDirectXWidget::~~CDirectXWidget ()
8  {
9
10 }
11 void CDirectXWidget::Initialize(void) {}
12 void CDirectXWidget::Finalize(void) {}
13 void CDirectXWidget::Update(void) {}
14 void CDirectXWidget::Draw(void) {}
15 void CDirectXWidget::SetUpdateFlag (bool flag) {}
16 bool CDirectXWidget::GetUpdateFlag (void) {}
```

コピペミス!  
=0 は  
消しておいてね。

まずは、通常どおりクラス定義を記述しましょう。

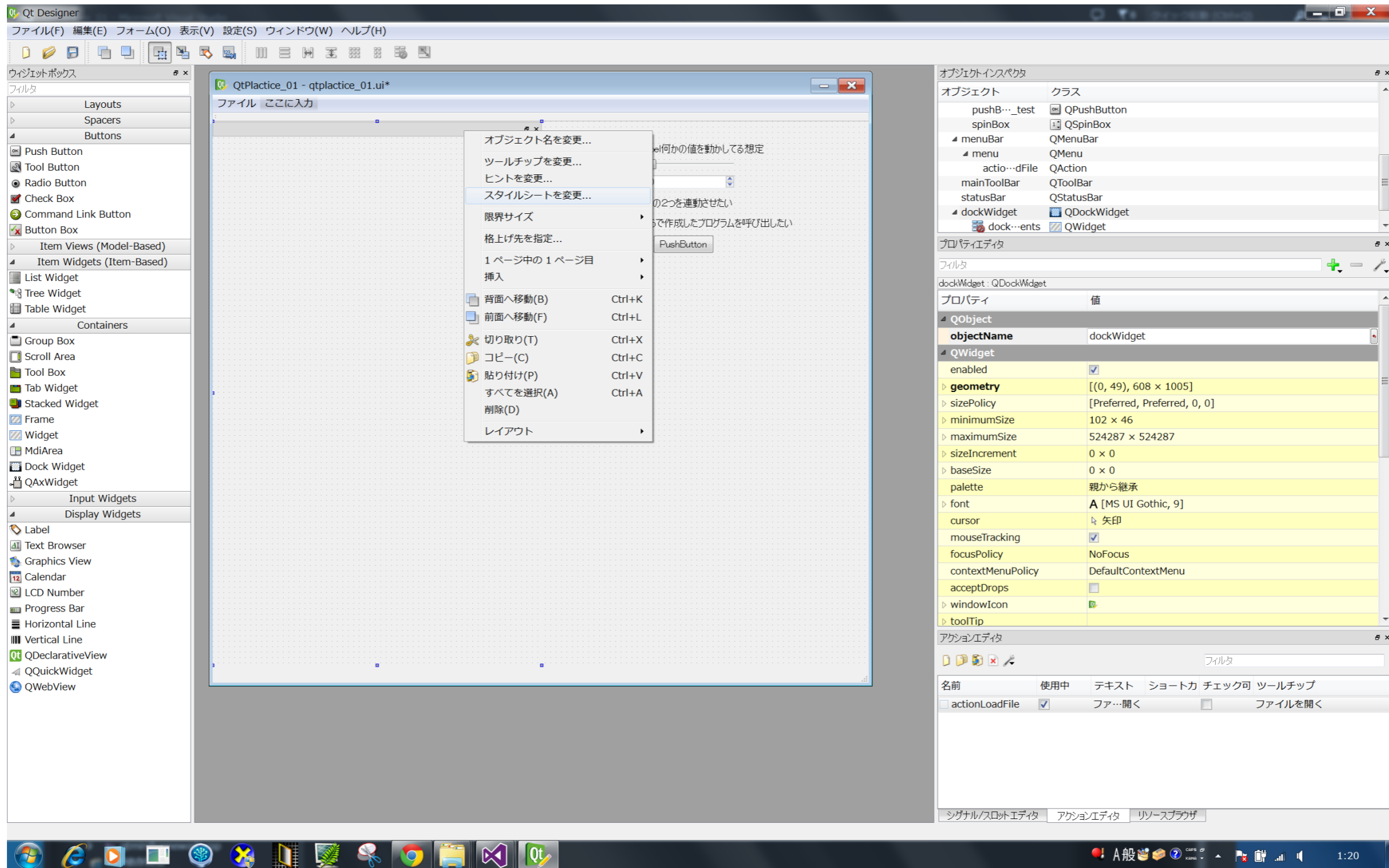
# 実践的に使うための準備

```
QtPlactice_01 (グローバルスコープ)
1 #include "DirectXWidget.h"
2 #include "qtplactice_01.h"
3
4 //どこかで定義しておくこと
5 static const float SCREEN_WIDTH = 600;
6 static const float SCREEN_HEIGHT = 800;
7
8 CDirectXWidget::CDirectXWidget(QWidget *parent = 0)
9     :QWidget(parent)
10    {
11    → //最低限のQT上でのDirectXRendering設定
12    → setFixedSize(SCREEN_WIDTH, SCREEN_HEIGHT);
13    → setAttribute(Qt::WA_PaintOnScreen, true);
14    → m_bUpdateFlag = true;
15    }
16 CDirectXWidget::~~CDirectXWidget()
17 {
18 }
19 }
20 void CDirectXWidget::Initialize(void)
21 {
```

図に従い、コードを記述します。



# 実践的に使うための準備



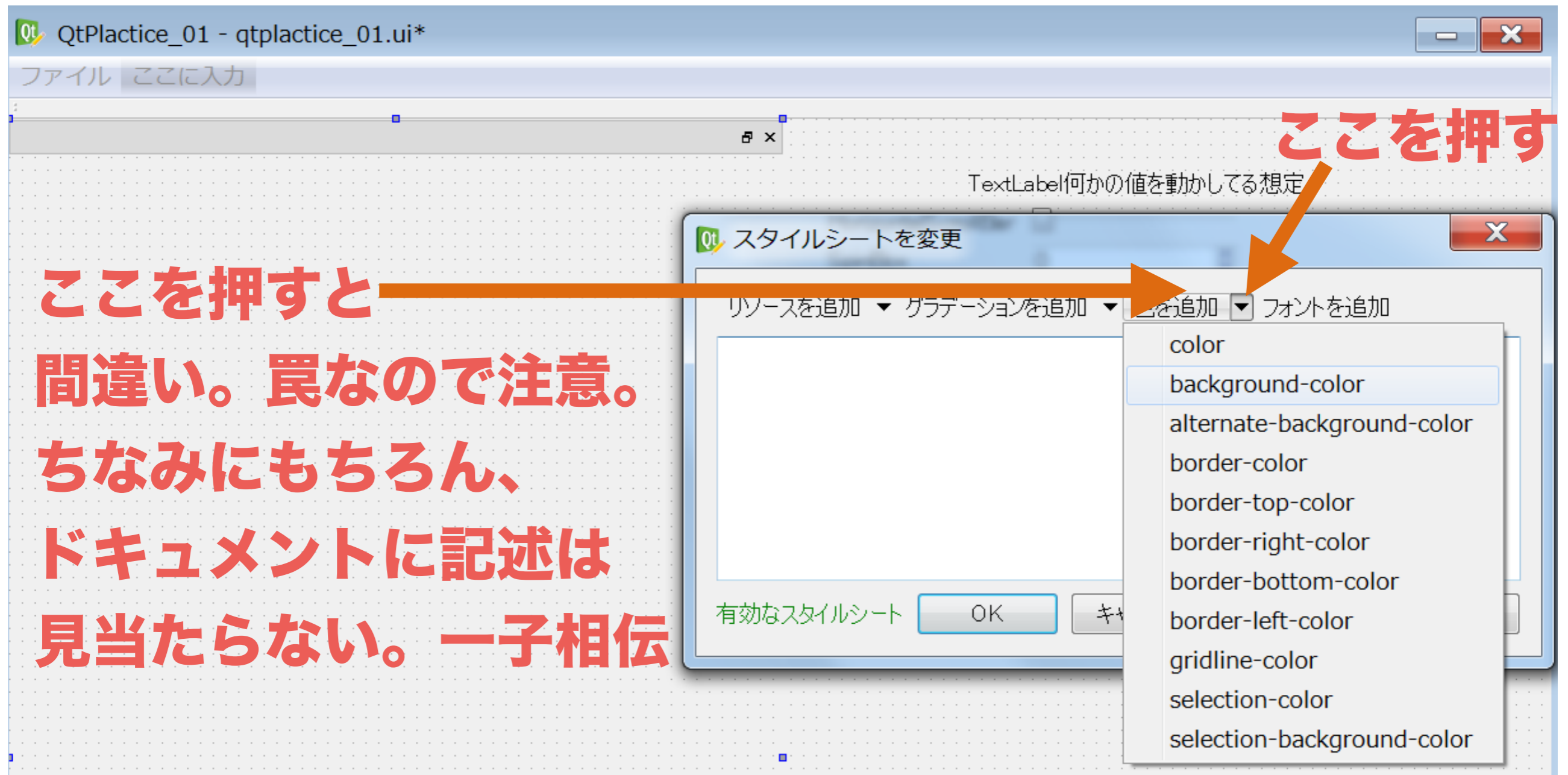
次にUIの設定をしておきます。dockWidget  
をウィンドウ内に入れたら、スタイルシートを変更します。

# 目次

---

- **なんでやろうと思ったのか**
- **どんなことが出来るのか**
- **開発環境の導入**
- **Qtとは、メリットとデメリット**
- **実際に触れてみる**
- **実践的に使うための準備**
- **DirectXのレンダラーをツールと連動させる**
- **ツール上でレンダリングする**
- **自分のフレームワーク上にレンダリングする**
- **入力とレンダリングの間**
- **配布時の注意点**
- **さらに実践的なQtツール制作上のノウハウについて**

# DirectXのレンダラーをツールと連動させる



The screenshot shows the Qt Designer interface. A dialog box titled "スタイルシートを変更" (Change Style Sheet) is open. It has a dropdown menu with "スタイルシートを追加" (Add Style Sheet) selected, and a list of style sheet properties is displayed. The property "background-color" is highlighted. Two red arrows point to the "スタイルシートを追加" dropdown and the "background-color" item, with the text "ここを押す" (Press here) written in red. The background of the designer shows a text label with the text "TextLabel何かの値を動かしてる想定".

ここを押すと間違い。異なるので注意。ちなみにもちろん、ドキュメントに記述は見当たらない。一子相伝

ここを押す

background-color

background-color

alternate-background-color

border-color

border-top-color

border-right-color

border-bottom-color

border-left-color

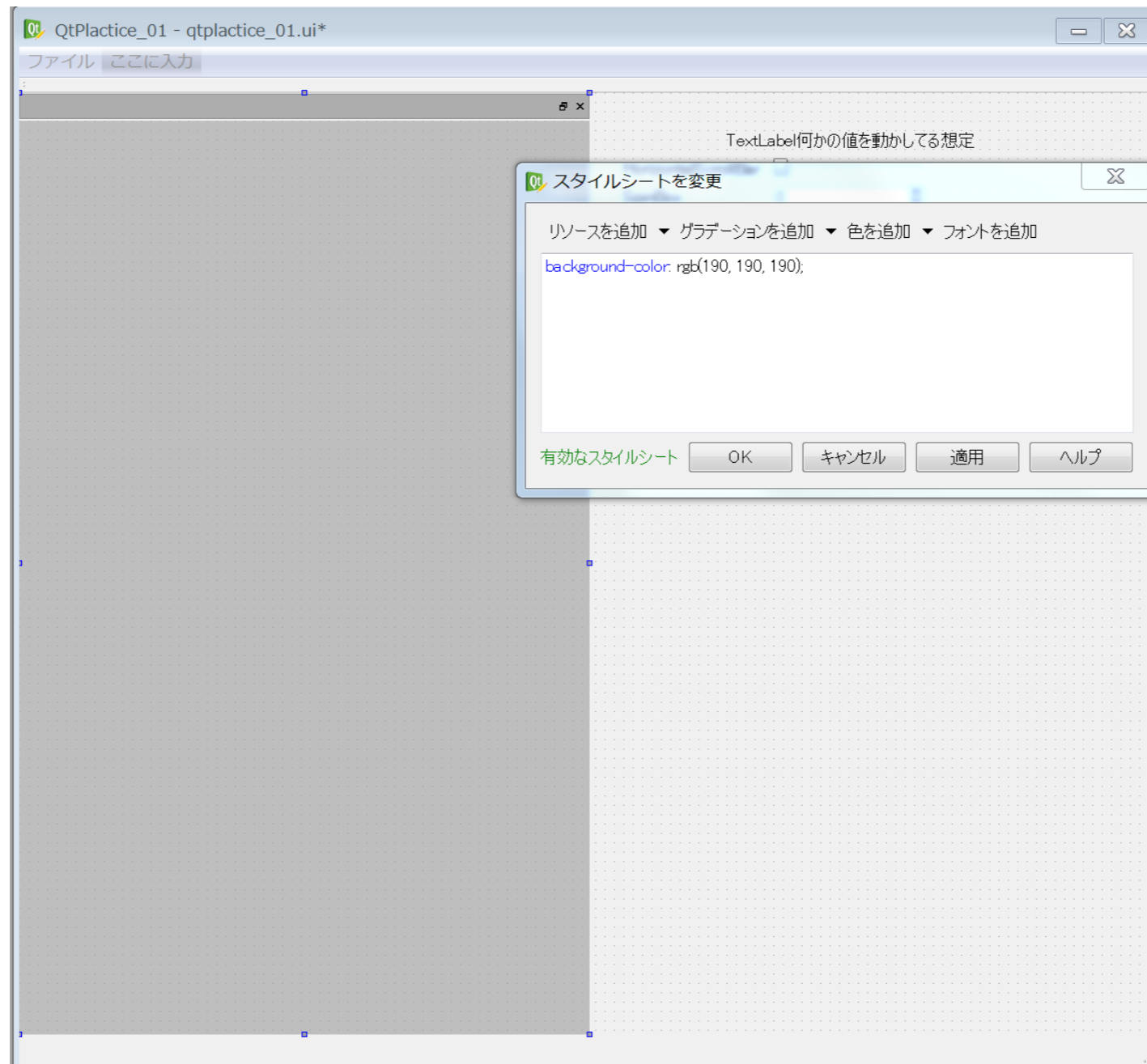
gridline-color

selection-color

selection-background-color

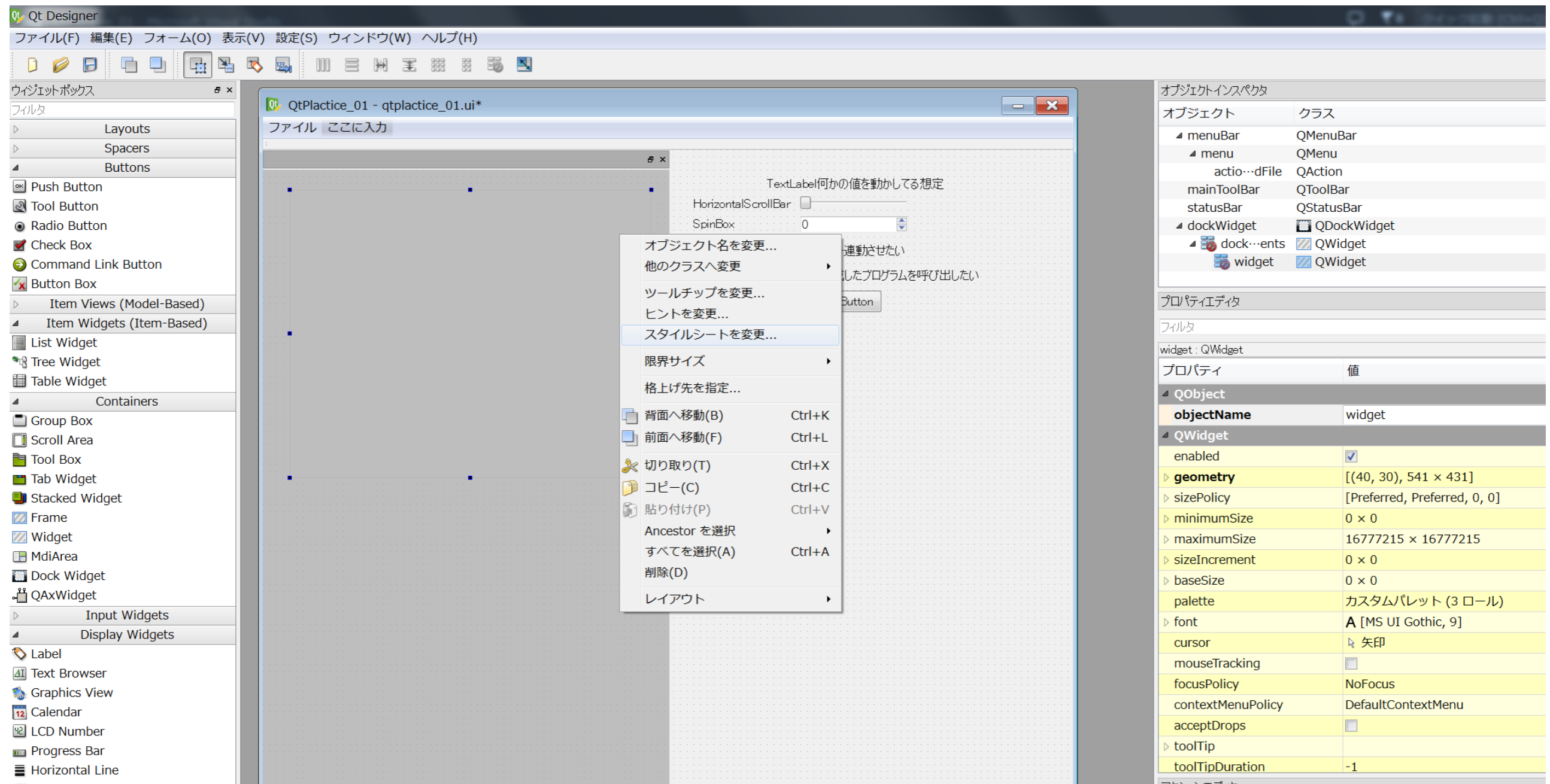
背景色を変更します。background-colorを選択します。

# DirectXのレンダラーをツールと連動させる



**図のように文章が打ち込まれるので（自分で記述してもいい）適用を押すと図のように背景色が変化します。**

# DirectXのレンダラーをツールと連動させる



次にDirectXを描画するためのWidgetを作ります。

WidgetをDockWidget内に入れたら、同様にスタイルシートで背景色を変更して描画領域をわかりやすくします。

# DirectXのレンダラーをツールと連動させる



The screenshot shows the Qt Designer interface. On the left, a cyan-colored rectangular widget is highlighted on a grid. The right side features a property editor and an object inspector. The object inspector lists the following objects and their classes:

オブジェクト	クラス
menuBar	QMenuBar
menu	QMenu
action...dFile	QAction
mainToolBar	QToolBar
statusBar	QStatusBar
dockWidget	QDockWidget
dock...ents	QWidget
widget	QWidget

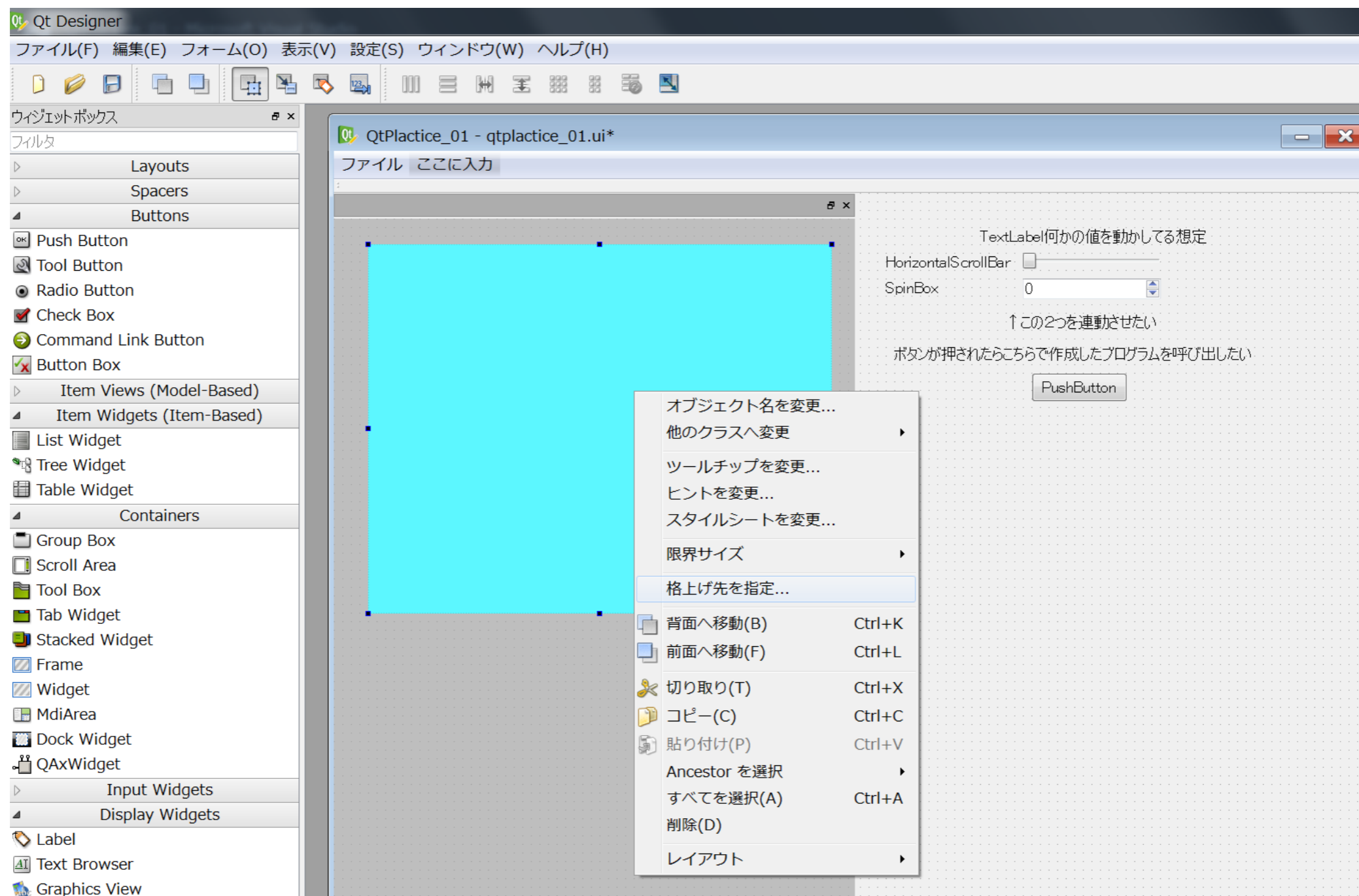
The property editor shows the following properties for the selected widget:

プロパティ	値
objectName	widget
enabled	<input checked="" type="checkbox"/>
geometry	[(40, 30), 541 × 431]
sizePolicy	[Preferred, Preferred, 0, 0]
minimumSize	0 × 0
maximumSize	16777215 × 16777215
sizeIncrement	0 × 0

ここではレンダリングされる領域がわかりやすいように、水色にしています。色はなんでもいいです。

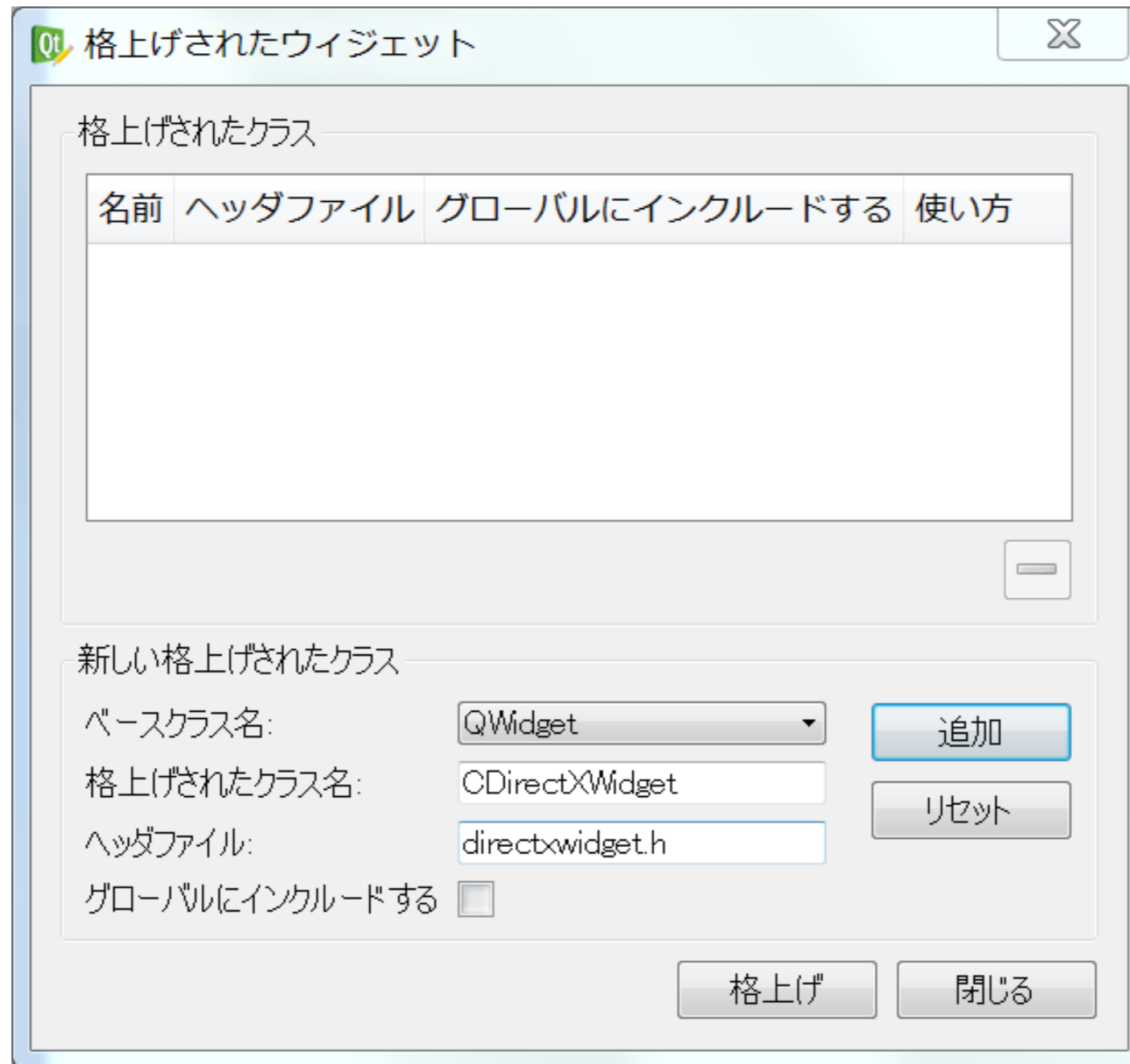


# DirectXのレンダラーをツールと連動させる



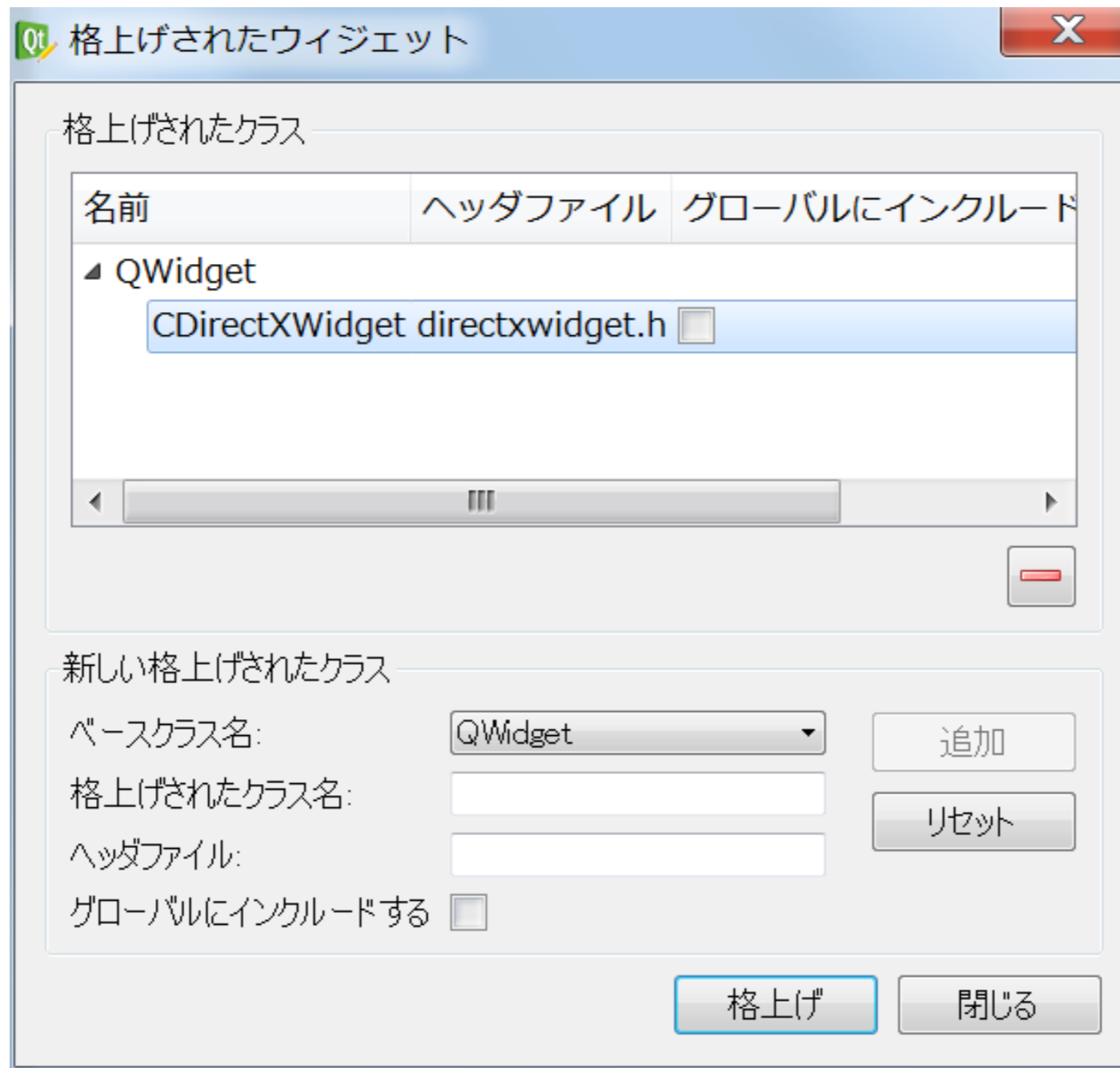
**次にwidgetを右クリックして格上げ先の指定を選択します。**

# DirectXのレンダラーをツールと連動させる



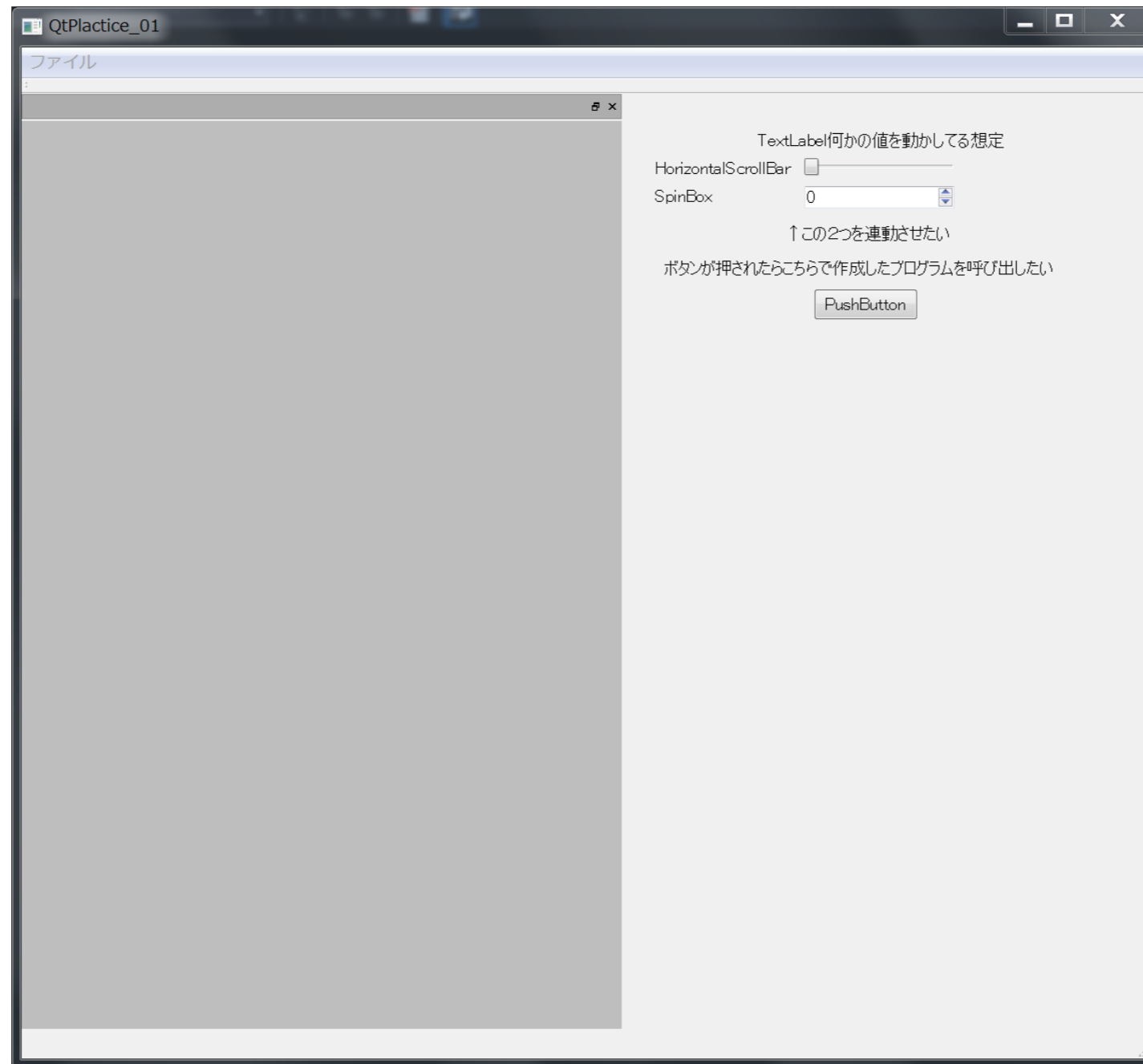
**図のようなウィンドウが出現するので、作成したクラス名とファイル名を正しくしていしたら追加ボタンを押します。**

# DirectXのレンダラーをツールと連動させる



**追加されたら、その項目を選択した状態で、格上げのボタンを押します。**

# DirectXのレンダラーをツールと連動させる



**保存して、実行すると現在は、まだこの状態になっています。**

# 目次

---

- **なんでやるうと思ったのか**
- **どんなことが出来るのか**
- **開発環境の導入**
- **Qtとは、メリットとデメリット**
- **実際に触れてみる**
- **実践的に使うための準備**
- **DirectXのレンダラーをツールと連動させる**
- **ツール上でレンダリングする**
- **自分のフレームワーク上にレンダリングする**
- **入力とレンダリングの間**
- **配布時の注意点**
- **さらに実践的なQtツール制作上のノウハウについて**

# ツール上でレンダリングする

```
1 #include "qtplactice_01.h"
2 #include <QtWidgets/QApplication>
3
4 int main(int argc, char *argv[])
5 {
6     QApplication a(argc, argv);
7     QtPlactice_01 w;
8     SetQtHwnd((HWND)w.winId());
9     w.ui.widget->Initialize();
10    w.show();
11    return a.exec();
12 }
13
```

準備完了 13行 1列 1文字 挿入

**Mainを変更します。ここでは一度QTPlactice\_01クラスインスタンスを作成した後に得られるQT用のHWNDを取得します。こうしないと諸々の事情で後々フレームワークの制御に失敗する**



# ツール上でレンダリングする

```
DirectXWidget.cpp  DirectXWidget.h  出力  qtplactice_01.h  qtplactice_01.cpp  main.cpp
QtPlactice_01
1  #ifndef QTPLACTICE_01_H
2  #define QTPLACTICE_01_H
3
4  #include <QtWidgets/QMainWindow>
5  #include "ui_qtplactice_01.h"
6  #include <QTimer>
7
8  class QtPlactice_01 : public QMainWindow
9  {
10     Q_OBJECT
11
12     public:
13     QtPlactice_01 (QWidget *parent = 0);
14     ~QtPlactice_01 ();
15     Ui::QtPlactice_01Class ui;
16     QTimer m_Timer;
17     private:
18
19     private slots :
20     void timer_start ();
21     // ボタンを押した
22     void on_pushButton_test_clicked ();
23
24     // ファイルを開く
25     void on_actionLoadFile_triggered ();
26 };
27
28 // DirectXをレンダリングするための準備
```

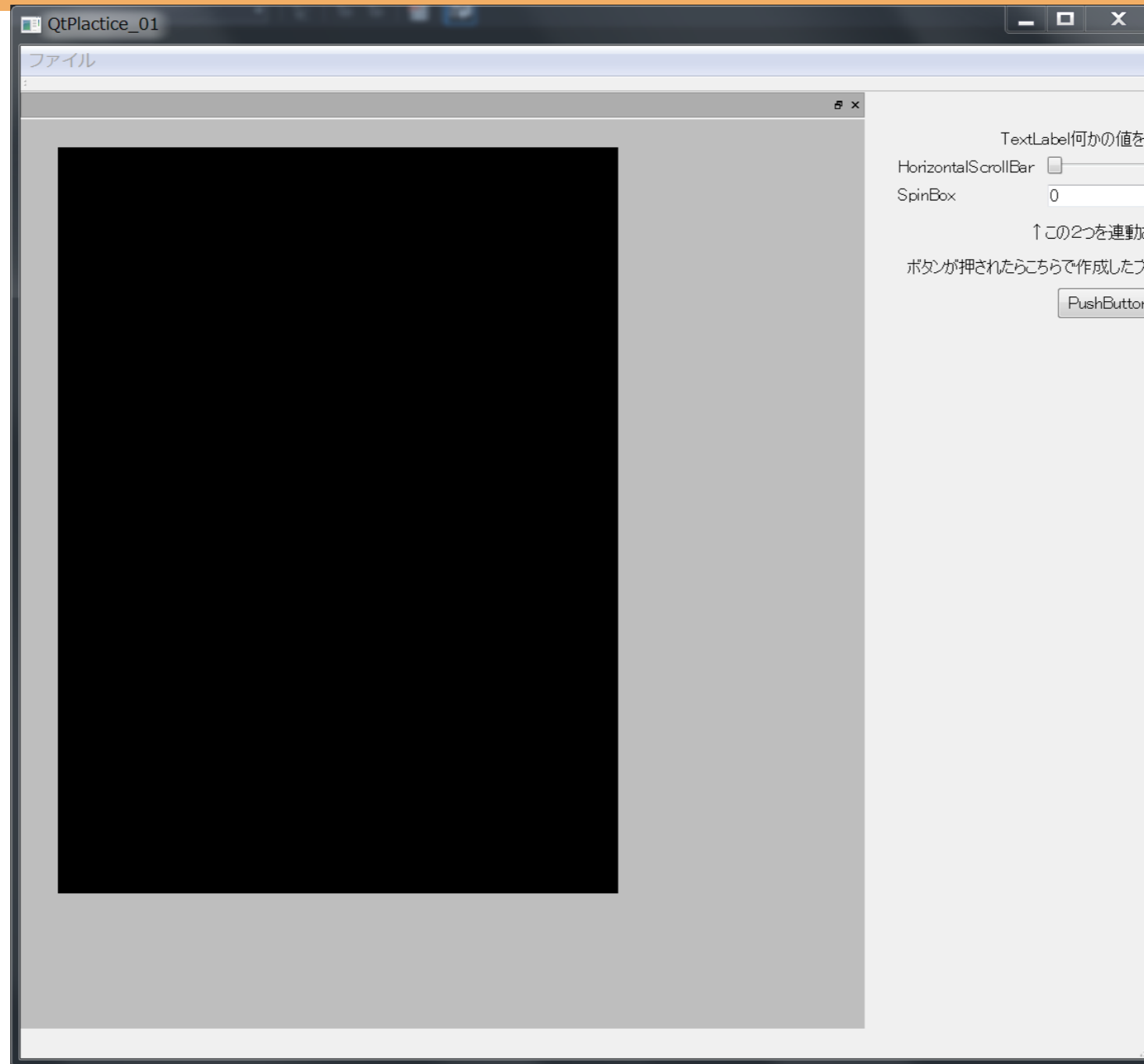
**描画ループ制御用にタイマークラスオブジェクトを追加。  
タイマー用のシグナルも追加しておきます。**

# ツール上でレンダリングする

```
DirectXWidget.cpp  DirectXWidget.h  出力  qtplactice_01.h  qtplactice_01.cpp  main.cpp
QtPlactice_01
1  #include "qtplactice_01.h"
2
3  QtPlactice_01::QtPlactice_01(QWidget *parent)
4  : QMainWindow(parent)
5  {
6  →  ui.setupUi(this);
7
8  →  //Timerクラス独自のスロットに接続する
9  →  connect(&m_Timer, SIGNAL(timeout()), this, SLOT(timer_start()));
10
11 →  //FPS用の内部キュータイマーをスタートさせる
12 →  m_Timer.start(0);
13 }
14
15 QtPlactice_01::~QtPlactice_01()
16 {
17
18 }
19
20 void QtPlactice_01::timer_start()
21 {
22 →  Sleep(11);
23 →  m_Timer.setInterval(1);
24 →  ui.widget->Update();
25 →  ui.widget->Draw();
26 }
27
```

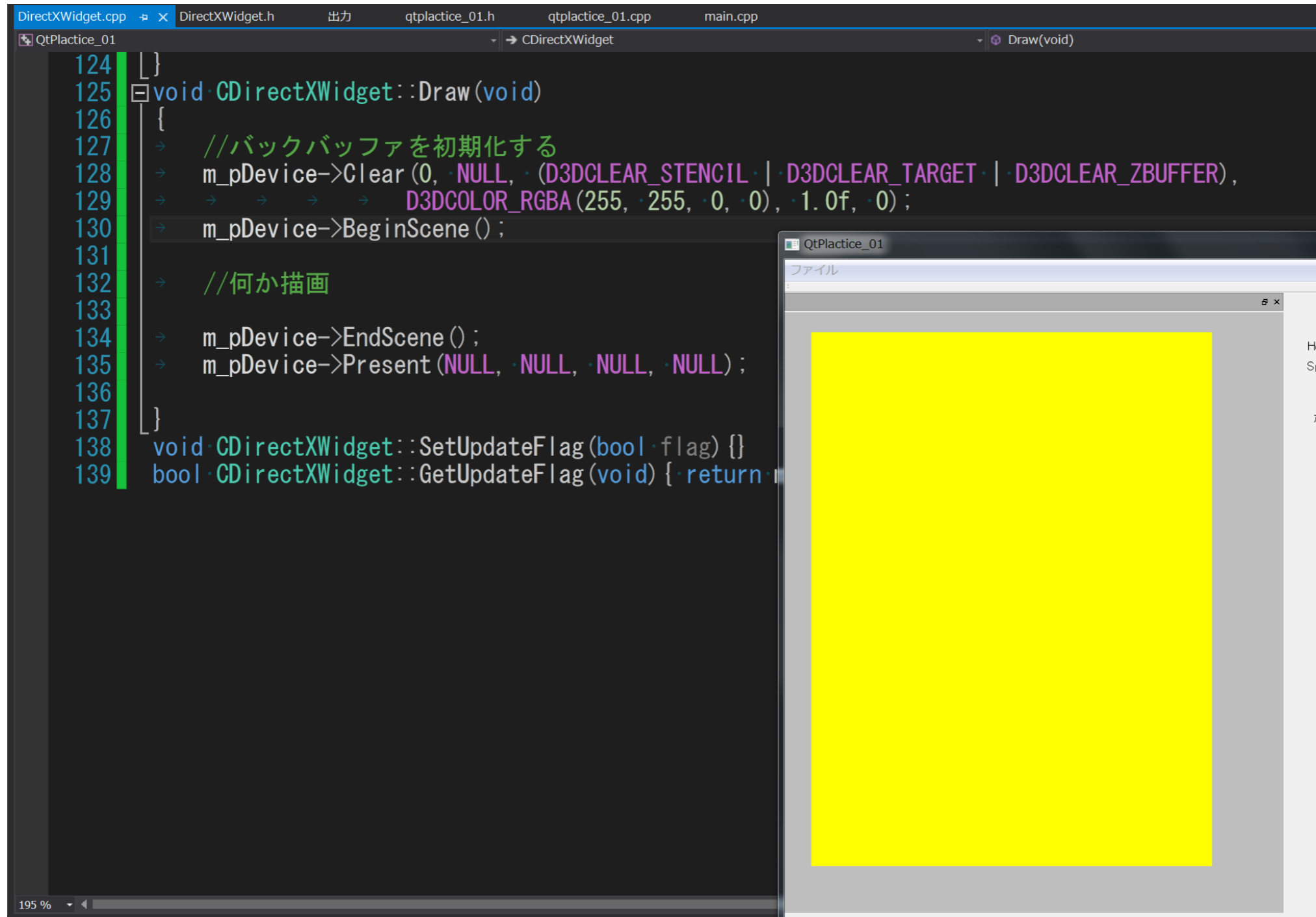
**図のように処理を追加します。ごり押しではあるのですが、一応これで毎回描画、更新がされるようになります。**

# ツール上でレンダリングする



**DirectXのツール内描画に成功！**

# ツール上でレンダリングする



```
124 }
125 void CDirectXWidget::Draw(void)
126 {
127     //バックバッファを初期化する
128     m_pDevice->Clear(0, NULL, (D3DCLEAR_STENCIL | D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER),
129     D3DCOLOR_RGBA(255, 255, 0, 0), 1.0f, 0);
130     m_pDevice->BeginScene();
131
132     //何か描画
133
134     m_pDevice->EndScene();
135     m_pDevice->Present(NULL, NULL, NULL, NULL);
136 }
137
138 void CDirectXWidget::SetUpdateFlag(bool flag) {}
139 bool CDirectXWidget::GetUpdateFlag(void) { return
```

The screenshot shows a code editor with a dark theme. The code is for a C++ class `CDirectXWidget`. The `Draw` method is highlighted, showing a call to `Clear` with a yellow color (`D3DCOLOR_RGBA(255, 255, 0, 0)`). To the right, a preview window titled `QtPlactice_01` displays a solid yellow rectangle, indicating that the clear operation is successful.

**クリアカラーどうりに描画されていることを確認！**

# ツール上でレンダリングする

```
qtplactice_01.ui DirectXWidget.cpp DirectXWidget.h 出力 qtplactice_01.h qtplactice_01.cpp main.cpp
QtPlactice_01 → CDirectXWidget Initialize(void)
20 void CDirectXWidget::Initialize(void)
21 {
22     SetWindowMode(true);
23     SetQtHwnd((HWND) this->winId());
24
25     HWND hWnd = GetQtHwnd();
26
27     //DirectXの初期化コードをコピーしてくる
28     D3DPRESENT_PARAMETERS d3dpp;
29     D3DDISPLAYMODE d3ddm;
30     m_pD3D = Direct3DCreate9(D3D_SDK_VERSION);
31
32     if (m_pD3D == NULL) { this->close(); }
33     if (FAILED(m_pD3D->GetAdapterDisplayMode(D3DADAPTER_DEFAULT, &d3ddm))) { this->close(); }
34
35     //デバイスのプレゼンテーションパラメータの設定
36     ZeroMemory(&d3dpp, sizeof(d3dpp)); //ワークをゼロクリア
37     d3dpp.BackBufferCount = 1; //バックバッファの数
38     d3dpp.BackBufferWidth = SCREEN_WIDTH; //ゲーム画面サイズ(幅)
39     d3dpp.BackBufferHeight = SCREEN_HEIGHT; //ゲーム画面サイズ(高さ)
40     d3dpp.BackBufferFormat = D3DFMT_UNKNOWN; //バックバッファのフォーマットは現在設定され
41     d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD; //映像信号に同期してフリップする
42     d3dpp.Windowed = GetWindowMode(); //ウィンドウモード
43     d3dpp.EnableAutoDepthStencil = TRUE; //デプスバッファ(Zバッファ)とステンシルバ
44     d3dpp.AutoDepthStencilFormat = D3DFMT_D24S8; //デプスバッファとして24bitを使う
45
46     if (GetWindowMode())
47     {
48         //ウィンドウモード
49         d3dpp.BackBufferFormat = D3DFMT_UNKNOWN; //バックバッファ
50         d3dpp.FullScreen_RefreshRateInHz = 0; //リフレッシュレート
```

```
qtplactice_01.ui DirectXWidget.cpp DirectXWidget.h 出力 qtplactice_01.h qtplactice_01.cpp main.cpp
QtPlactice_01 → CDirectXWidget Initialize(void)
45
46     if (GetWindowMode())
47     {
48         //ウィンドウモード
49         d3dpp.BackBufferFormat = D3DFMT_UNKNOWN; //バックバッファ
50         d3dpp.FullScreen_RefreshRateInHz = 0; //リフレッシュレート
51         d3dpp.PresentationInterval = D3DPRESENT_INTERVAL_IMMEDIATE; //インターバル
52     }
53     else
54     {
55         //フルスクリーンモード
56         d3dpp.BackBufferFormat = D3DFMT_R5G6B5; //バックバッファ
57         d3dpp.FullScreen_RefreshRateInHz = D3DPRESENT_RATE_DEFAULT; //リフレッシュレート
58         d3dpp.PresentationInterval = D3DPRESENT_INTERVAL_DEFAULT; //インターバル
59     }
60
61     //デバイスオブジェクトの生成
62     if (FAILED(m_pD3D->CreateDevice(D3DADAPTER_DEFAULT, //デバイス作成制御<描画>と
63         D3DDEVTYPE_HAL, // ディスプレイアダプタ
64         hWnd, // ディスプレイタイプ
65         D3DCREATE_HARDWARE_VERTEXPROCESSING, // フォーカスするウィンドウ
66         &d3dpp, // デバイス作成制御の組み合わせ
67         &m_pDevice))) // デバイスのプレゼンテーション
68     {
69         //上記の設定が失敗したら
70         // [デバイス作成制御<描画>をハードウェアで行い、<頂点処理>はCPUで行なう
71         if (FAILED(m_pD3D->CreateDevice(D3DADAPTER_DEFAULT,
72             D3DDEVTYPE_HAL,
73             hWnd,
74             D3DCREATE_SOFTWARE_VERTEXPROCESSING,
75             &m_pDevice)))
```

```
qtplactice_01.ui DirectXWidget.cpp DirectXWidget.h 出力 qtplactice_01.h qtplactice_01.cpp main.cpp
QtPlactice_01 → CDirectXWidget Initialize(void)
63     if (FAILED(m_pD3D->CreateDevice(D3DADAPTER_DEFAULT, // ディスプレ
64         D3DDEVTYPE_HAL, // ディスプレ
65         hWnd, // フォーカス
66         D3DCREATE_HARDWARE_VERTEXPROCESSING, // デバイス作
67         &d3dpp, // デバイスの
68         &m_pDevice))) // デバイス
69     {
70         //上記の設定が失敗したら
71         // [デバイス作成制御<描画>をハードウェアで行い、<頂点処理>はCPUで行なう
72         if (FAILED(m_pD3D->CreateDevice(D3DADAPTER_DEFAULT,
73             D3DDEVTYPE_HAL,
74             hWnd,
75             D3DCREATE_SOFTWARE_VERTEXPROCESSING,
76             &d3dpp,
77             &m_pDevice)))
78     {
79         //上記の設定が失敗したら
80         // [デバイス作成制御<描画>と<頂点処理>をCPUで行なう
81         if (FAILED(m_pD3D->CreateDevice(D3DADAPTER_DEFAULT,
82             D3DDEVTYPE_REF,
83             hWnd,
84             D3DCREATE_SOFTWARE_VERTEXPROCESSING,
85             &d3dpp,
86             &m_pDevice)))
87     {
88         //初期化失敗
89         this->close();
90     }
91 }
92 }
93 }
```

DirectXの初期化はいつも通りのため、コピペしてくる

# ツール上でレンダリングする

```
qtplactice_01.ui  DirectXWidget.cpp  DirectXWidget.h  出力  qtplactice_01.h  qtplactice_01.cpp  main.cpp
QtPlactice_01    -> CDirectXWidget  Initialize(void)
111
112  → //デバイス作ったらDirect3Dオブジェクトをすてていい。
113  → m_pD3D->Release();
114  → m_pD3D = nullptr;
115  }
116  void CDirectXWidget::Finalize(void)
117  {
118  → m_pDevice->Release();
119  → m_pDevice = nullptr;
120  }
121  void CDirectXWidget::Update(void)
122  {
123  →
124  }
125  void CDirectXWidget::Draw(void)
126  {
127  → //バックバッファを初期化する
128  → m_pDevice->Clear(0, NULL, (D3DCLEAR_STENCIL | D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER),
129  → → → → D3DCOLOR_RGBA(255, 255, 0, 0), 1.0f, 0);
130  → m_pDevice->BeginScene();
131
132  → //何か描画
133
134  → m_pDevice->EndScene();
135  → m_pDevice->Present(NULL, NULL, NULL, NULL);
136
137  }
138  void CDirectXWidget::SetUpdateFlag(bool flag) {}
139  bool CDirectXWidget::GetUpdateFlag(void) { return m_bUpdateFlag; }
```

**解放処理も忘れずに！ メモリリークチェックも併用可能！**



# 目次

---

- **なんでやるうと思ったのか**
- **どんなことが出来るのか**
- **開発環境の導入**
- **Qtとは、メリットとデメリット**
- **実際に触れてみる**
- **実践的に使うための準備**
- **DirectXのレンダラーをツールと連動させる**
- **ツール上でレンダリングする**
- **自分のフレームワーク上にレンダリングする**
- **入力とレンダリングの間**
- **配布時の注意点**
- **さらに実践的なQtツール制作上のノウハウについて**

# 自分のフレームワーク上にレンダリングする

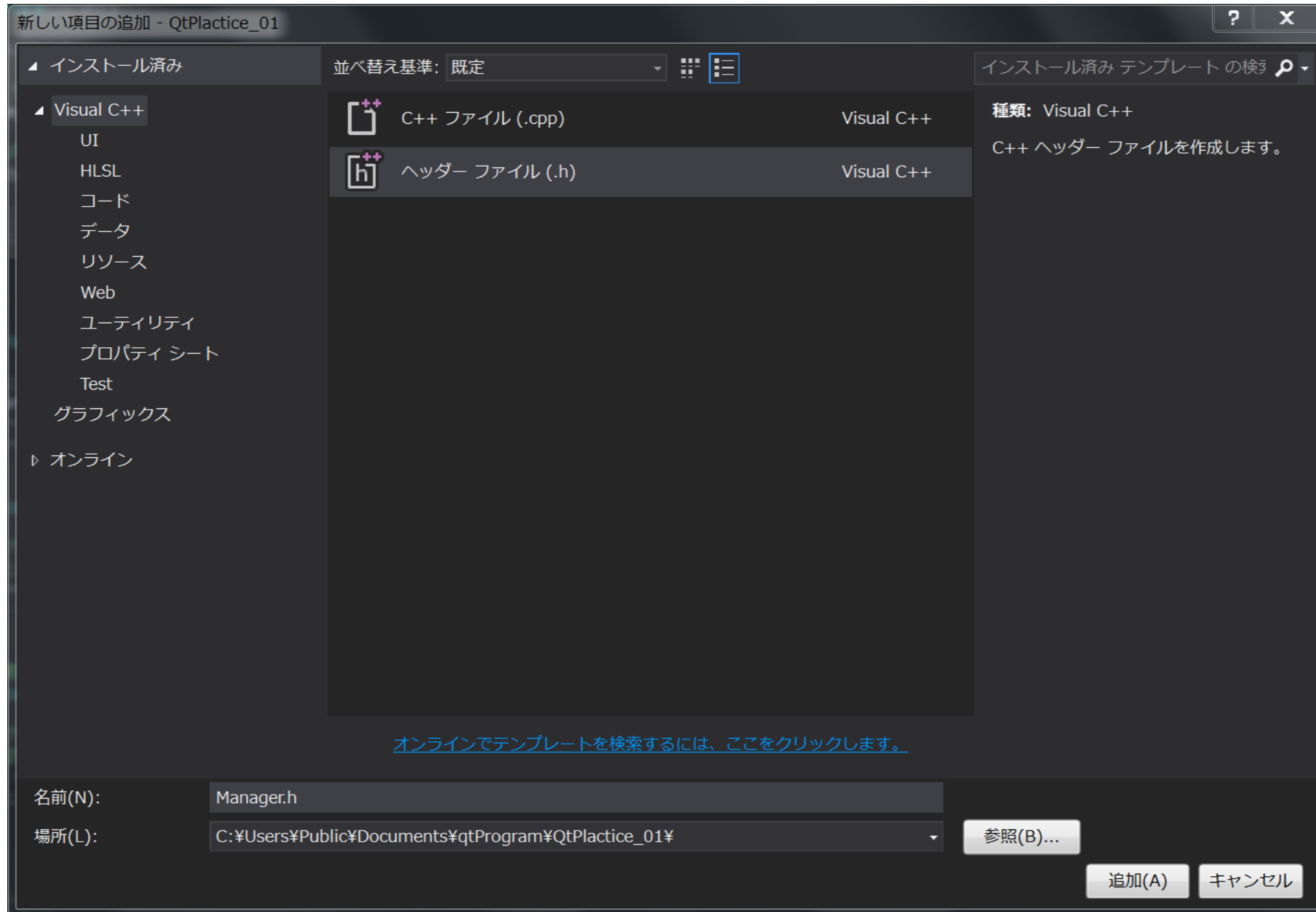
なんとか、DirectXをツール上で動かす仕組みについて理解しました。これを応用して、マネージャクラスに対して、描画、更新をDirectXWidget上で行えるように変更することで、各自で今使っているDirectXのフレームワークをツール上でレンダリングできるはずです。

ここでは、その手助けとして、Managerクラスを新規に作成して、DirectXWidgetのクラスを変更する例を示したいと思います。



まだ生きてたか

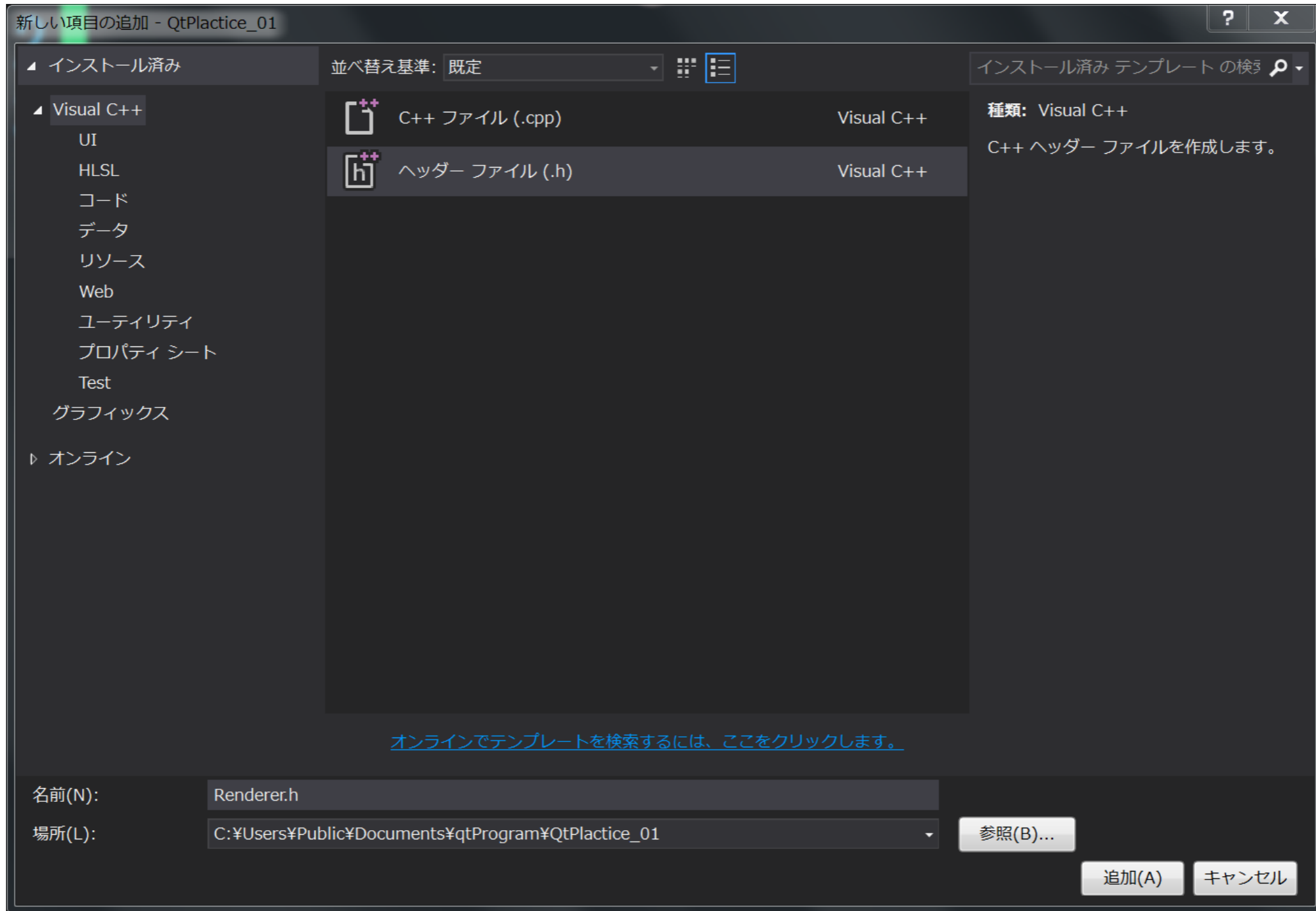
# 自分のフレームワーク上にレンダリングする



そのままのプロジェクトから続けていきます。

Managerクラスを新規作成します。

# 自分のフレームワーク上にレンダリングする



**DirectXWidget内に直接あったDirectXクラスを  
Rendererクラスを新規に作り、そこに移します。**

# 自分のフレームワーク上にレンダリングする

```
Manager.cpp  Renderer.cpp  Renderer.h  Manager.h  出力  qtpractice_01.ui  DirectXWid
QtPractice_01
CManager
1  #pragma once
2
3  #ifndef _MANAGER_H_
4  #define _MANAGER_H_
5
6  //DirectX描画制御用クラス
7  class CRenderer;
8
9  class CManager
10 {
11 public:
12     CManager ();
13     ~CManager ();
14     void Update ();
15     void Draw ();
16 private:
17     CRenderer* m_pRenderer;
18 };
19
20 #endif
```

```
Manager.cpp  Renderer.cpp  Renderer.h  Manager.h  出力  qtpractice_01.ui  DirectXWidget.cpp
QtPractice_01
CRenderer
1  #pragma once
2
3  #ifndef _RENDERER_H_
4  #define _RENDERER_H_
5
6  #include "d3dx9.h"
7
8  //どこかで定義しておくこと
9  static const float SCREEN_WIDTH = 600;
10 static const float SCREEN_HEIGHT = 800;
11
12 class CRenderer
13 {
14 public:
15     CRenderer ();
16     ~CRenderer ();
17     void Update ();
18     void Begin ();
19     void End ();
20 private:
21     LPDIRECT3D9 m_pD3D;
22     LPDIRECT3DDEVICE9 m_pDevice;
23 };
24
25 #endif
```

ヘッダファイルは図のように記述しました。

# 自分のフレームワーク上にレンダリングする

```
Manager.cpp* x Renderer.cpp Renderer.h Manager.h 出力 qtpractice_01.ui
QtPractice_01 -> CManager
1 #include "Manager.h"
2 #include "Renderer.h"
3 CManager::CManager() {
4     m_pRenderer = new CRenderer();
5 }
6 CManager::~CManager() {
7     delete m_pRenderer;
8     m_pRenderer = nullptr;
9 }
10 void CManager::Update() {
11     m_pRenderer->Update();
12     //ここで何か更新する
13     //m_pCamera->Update();
14     //m_pLight->Update();
15     //m_pMeshField->Update();
16     //m_pPlayer->Update();
17     //.....
18 }
19 void CManager::Draw() {
20     m_pRenderer->Begin();
21
22     //ここで何か描画する
23     //m_pMeshField->Draw();
24     //m_pPlayer->Draw();
25     //.....
26     m_pRenderer->End();
27 }
```

```
QtPractice_01 -> CRenderer()
1 #include "Renderer.h"
2 #include "qtpractice_01.h"
3
4 CRenderer::CRenderer()
5 {
6     HWND hWnd = GetQtHwnd();
7
8     //DirectXの初期化コードをコピーしてくる
9     D3DPRESENT_PARAMETERS d3dpp;
10    D3DDISPLAYMODE d3ddm;
11    m_pD3D = Direct3DCreate9(D3D_SDK_VERSION);
12
13    if (m_pD3D == NULL) { return; }
14    if (FAILED(m_pD3D->GetAdapterDisplayMode(D3DADAPTER_DEFAULT, &d3ddm)) { return; }
15
16    //デバイスのプレゼンテーションパラメータの設定
17    ZeroMemory(&d3dpp, sizeof(d3dpp)); //ワークをゼロクリア
18    d3dpp.BackBufferCount = 1; //バックバッファの数
19    d3dpp.BackBufferWidth = SCREEN_WIDTH; //ゲーム画面サイズ(幅)
20    d3dpp.BackBufferHeight = SCREEN_HEIGHT; //ゲーム画面サイズ(高さ)
21    d3dpp.BackBufferFormat = D3DFMT_UNKNOWN; //バックバッファのフォーマットは現在設定されているものを使う
22    d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD; //映像信号に同期してフリップする
23    d3dpp.Windowed = GetWindowMode(); //ウィンドウモード
24    d3dpp.EnableAutoDepthStencil = TRUE; //デプスバッファ (Zバッファ) とステンシルバッファを作成
25    d3dpp.AutoDepthStencilFormat = D3DFMT_D24S8; //デプスバッファとして24bitを使う
26
27    if (GetWindowMode())
28    {
29        //ウィンドウモード
30        d3dpp.BackBufferFormat = D3DFMT_UNKNOWN; //バックバッファ
31        d3dpp.FullScreen_RefreshRateInHz = 0; //リフレッシュレート
32        d3dpp.PresentationInterval = D3DPRESENT_INTERVAL_IMMEDIATE; //インターバル
33    }
34    else
35    {
36        //フルスクリーンモード
37        d3dpp.BackBufferFormat = D3DFMT_R5G6B5; //バックバッファ
38        d3dpp.FullScreen_RefreshRateInHz = D3DPRESENT_RATE_DEFAULT; //リフレッシュレート
39        d3dpp.PresentationInterval = D3DPRESENT_INTERVAL_DEFAULT; //インターバル
40    }
41 }
```

**Managerクラスは各クラスの制御をしている想定です。**



# 自分のフレームワーク上にレンダリングする

```
Manager.cpp  Renderer.cpp  Renderer.h  Manager.h  出力  qtplactice_01.ui  DirectXWidget.cpp  DirectXWidget.h
QtPlactice_01  CRenderer
85  > m_pDevice->SetSamplerState (0, D3DSAMP_MINFILTER, D3DTEXF_LINEAR); > // テクスチャ縮
86  > m_pDevice->SetSamplerState (0, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR); > // テクスチャ拡
87
88  > // テクスチャステージステートパラメータの設定
89  > m_pDevice->SetTextureStageState (0, D3DTSS_ALPHAOP, D3DTOP_MODULATE); > // アルファ
90  > m_pDevice->SetTextureStageState (0, D3DTSS_ALPHAARG1, D3DTA_TEXTURE); > // 最初のアル
91  > m_pDevice->SetTextureStageState (0, D3DTSS_ALPHAARG2, D3DTA_CURRENT); > // 2番目の
92
93  > // デバイス作ったらDirect3Dオブジェクトをすてていい。
94  > m_pD3D->Release ();
95  > m_pD3D = nullptr;
96  }
97
98  CRenderer::~CRenderer ()
99  {
100  > m_pDevice->Release ();
101  > m_pDevice = nullptr;
102  }
103
104  void CRenderer::Update ()
105  {
106  }
107  }
108  void CRenderer::Begin ()
109  {
110  > // バックバッファを初期化する
111  > m_pDevice->Clear (0, NULL, (D3DCLEAR_STENCIL | D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER),
112  > D3DCOLOR_RGBA (255, 255, 0, 0), 1.0f, 0);
113  > m_pDevice->BeginScene ();
114  }
115
116  void CRenderer::End ()
117  {
118  > m_pDevice->EndScene ();
119  > m_pDevice->Present (NULL, NULL, NULL, NULL);
120  }
```

単純にRendererクラスに描画コードを移していきます。

# 自分のフレームワーク上にレンダリングする

```
Manager.cpp  DirectXWidget.h  Renderer.cpp  Renderer.h  Manager.h  出力  qtpractice_01
QtPractice_01  CDirectXWidget
7 //DirectX組み込み
8 #include <d3d9.h>
9 #include <d3dx9.h>
10 #pragma comment(lib, "d3d9.lib")
11 #pragma comment(lib, "d3dx9.lib")
12 #pragma comment(lib, "dxguid.lib")
13 #pragma comment(lib, "dinput8.lib")
14 #pragma comment(lib, "winmm.lib")
15
16 class CManager;
17
18 class CDirectXWidget : public QWidget {
19     Q_OBJECT
20 public:
21     CDirectXWidget(QWidget *parent = 0);
22     ~CDirectXWidget();
23     void Initialize(void);
24     void Finalize(void);
25     void Update(void);
26     void Draw(void);
27     void SetUpdateFlag(bool flag);
28     bool GetUpdateFlag(void);
29 private:
30     bool m_bUpdateFlag; //更新フラグ
31     QTimer m_Timer; //タイマー
32     CManager* m_pManager; //マネージャ
33 };
34 #endif
```

本当はこいつらも別の場所に移した方がいいです。

DirectXWidget内でマネージャクラスを制御

DirectXWidgetクラスの定義を変更します。

# 自分のフレームワーク上にレンダリングする

```
Manager.cpp DirectXWidget.h Renderer.cpp Renderer.h Manager.h 出力 qtpractice_01.ui DirectXWidget.cpp
QtPactice_01 → CDirectXWidget
13     → m_bUpdateFlag = true;
14     }
15     CDirectXWidget::~CDirectXWidget()
16     {
17     → Finalize();
18     }
19     void CDirectXWidget::Initialize(void)
20     {
21     → SetWindowMode(true);
22     → SetQtHwnd((HWND) this->winId());
23     }
24     → m_pManager = new CManager();
25     }
26     void CDirectXWidget::Finalize(void)
27     {
28     → delete m_pManager;
29     → m_pManager = nullptr;
30     }
31     void CDirectXWidget::Update(void)
32     {
33     → m_pManager->Update();
34     }
35     void CDirectXWidget::Draw(void)
36     {
37     → m_pManager->Draw();
38     }
39     void CDirectXWidget::SetUpdateFlag(bool flag) {}
40     bool CDirectXWidget::GetUpdateFlag(void) { return m_bUpdateFlag; }
```

**DirectXWidgetクラス内がだいぶスッキリしました。**

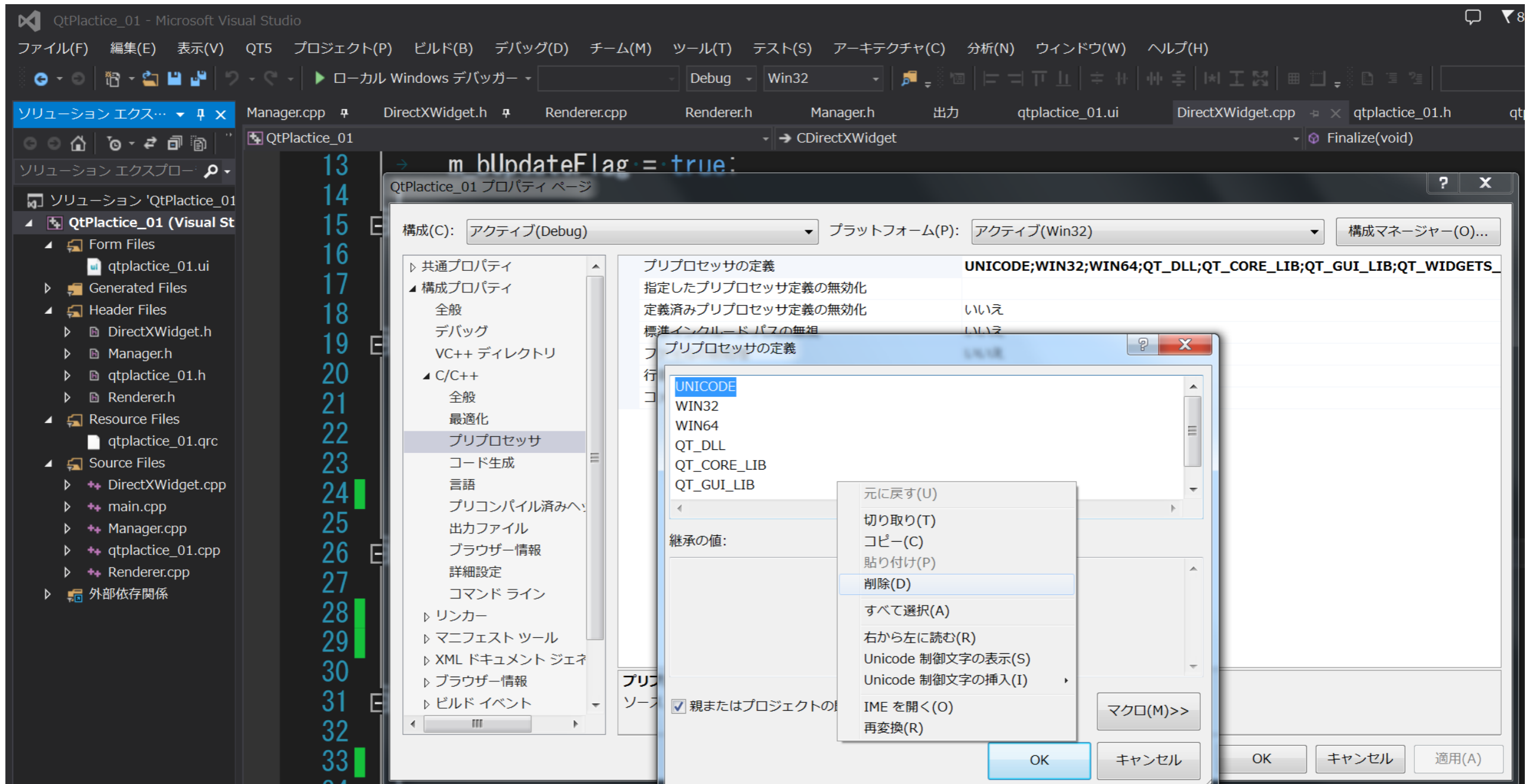
# 自分のフレームワーク上にレンダリングする

The screenshot displays a Qt IDE environment with several source files open: Manager.cpp, DirectXWidget.h, Renderer.cpp, Renderer.h, Manager.h, 出力, qtpractice\_01.ui, DirectXWidget.cpp, qtpractice\_01.h, qtpractice\_01.cpp, and main.cpp. The active file is DirectXWidget.cpp, showing the implementation of the CDirectXWidget class. The code includes methods for initialization, finalization, update, and draw, all delegating to a CManager object. A yellow rectangle is rendered in the QtPlactice\_01 window, demonstrating the output of the rendering process.

```
13     m_bUpdateFlag = true;
14 }
15 CDirectXWidget::~CDirectXWidget()
16 {
17     Finalize();
18 }
19 void CDirectXWidget::Initialize(void)
20 {
21     SetWindowMode(true);
22     SetQtHwnd((HWND) this->winId());
23 }
24 m_pManager = new CManager();
25 }
26 void CDirectXWidget::Finalize(void)
27 {
28     delete m_pManager;
29     m_pManager = nullptr;
30 }
31 void CDirectXWidget::Update(void)
32 {
33     m_pManager->Update();
34 }
35 void CDirectXWidget::Draw(void)
36 {
37     m_pManager->Draw();
38 }
39 void CDirectXWidget::SetUpdateFlag(bool)
40 bool CDirectXWidget::GetUpdateFlag()
```

無事に描画プログラムをマネージャ内で制御できました。

# 自分のフレームワーク上にレンダリングする



各自で実際にフレームワークを取り込む際は、QT側の文字コード設定を解除してください。方法はプリプロセッサ内にあるUNICODEの記述を消せばOK

# 目次

---

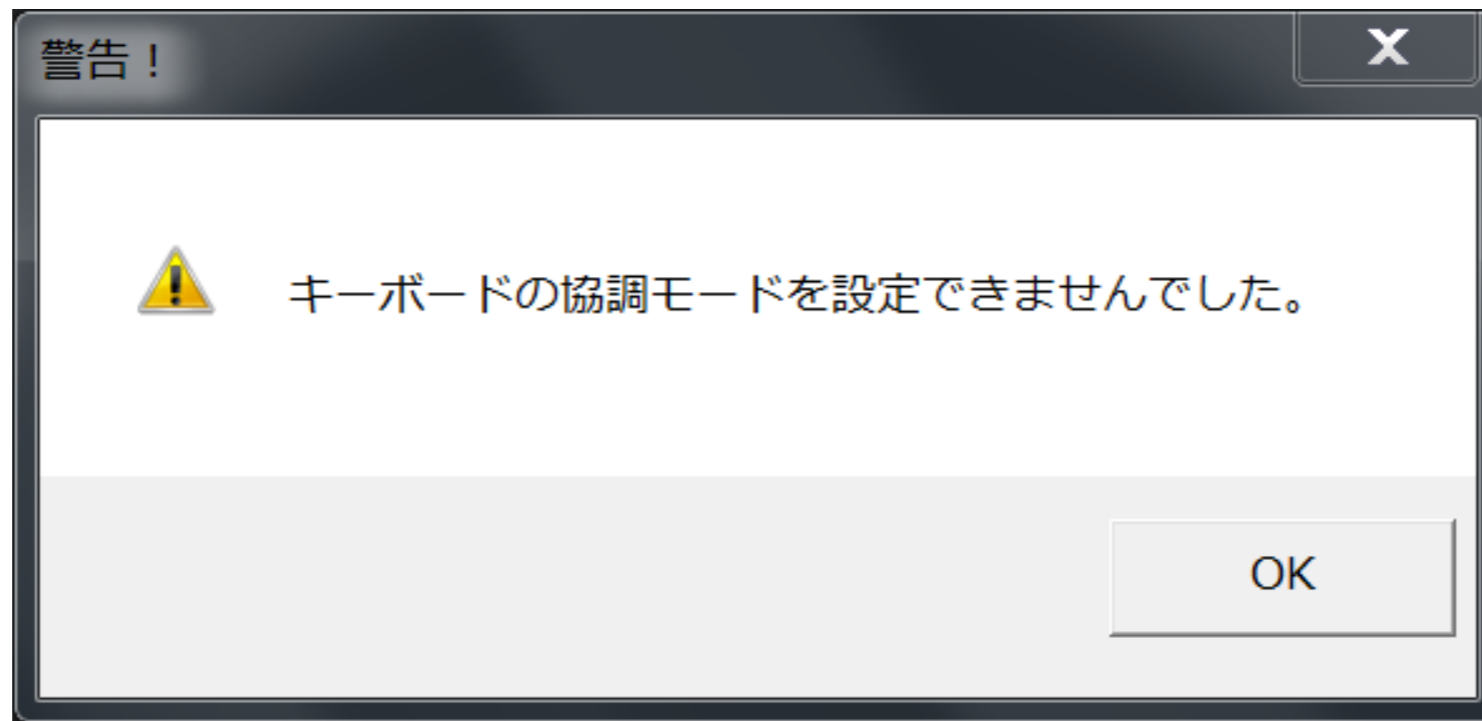
- **なんでやろうと思ったのか**
- **どんなことが出来るのか**
- **開発環境の導入**
- **Qtとは、メリットとデメリット**
- **実際に触れてみる**
- **実践的に使うための準備**
- **DirectXのレンダラーをツールと連動させる**
- **ツール上でレンダリングする**
- **自分のフレームワーク上にレンダリングする**
- **入力とレンダリングの間**
- **配布時の注意点**
- **さらに実践的なQtツール制作上のノウハウについて**



# 入力とレンダリングの間

なんとかフレームワークをエディターに組み込むことができました。しかし...

フレームワークによっては以下のようなエラーが出現する可能性があります。



結果レンダリングされるが、キー入力を受け付けない...  
(ここから本当に一子相伝。日本語英語ともに資料皆無)

# 入力とレンダリングの間

---

**結論だけいいいます。**

# 入力とレンダリングの闇

---

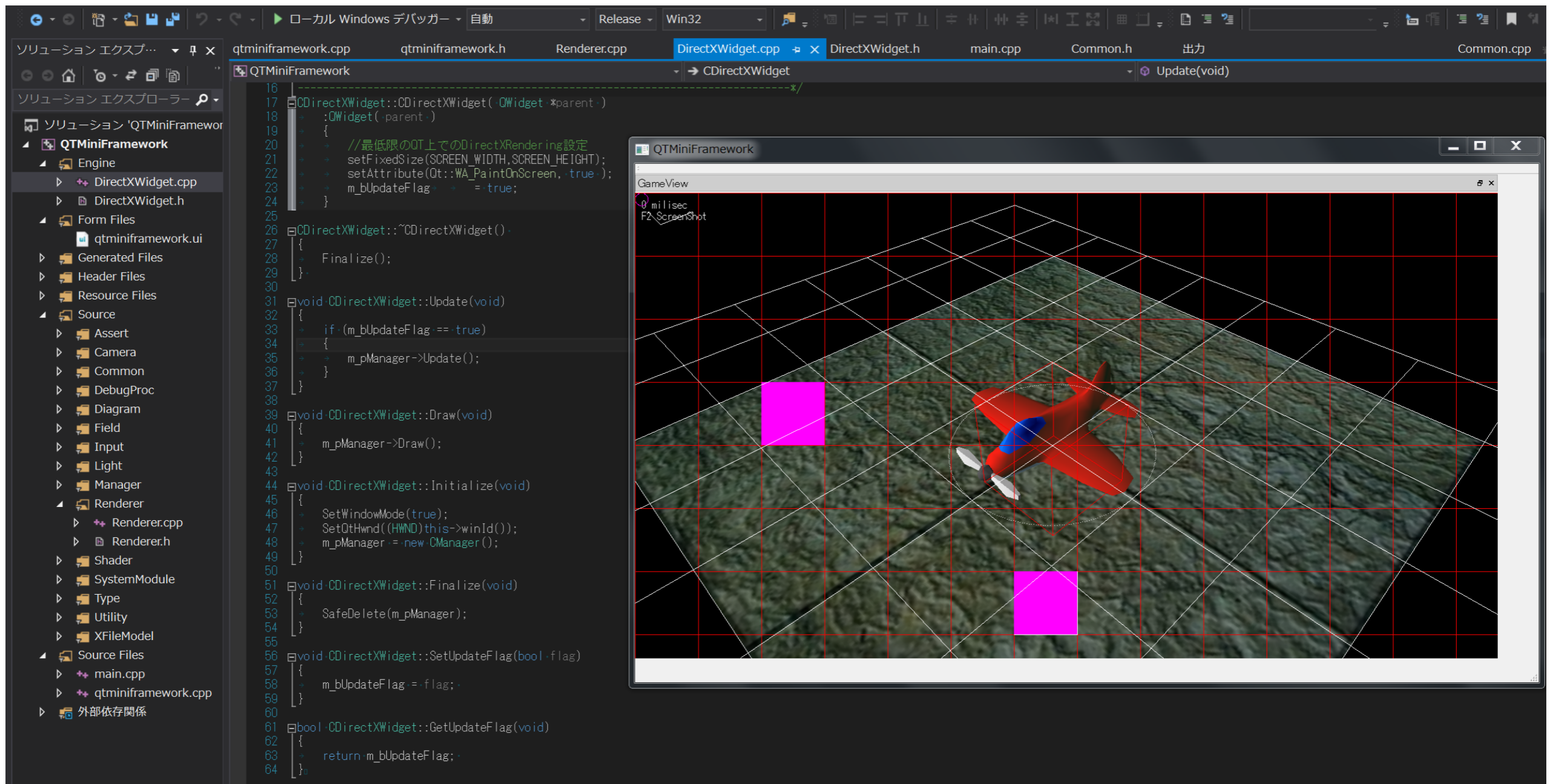
**レンダラーと入力側で初期化のときに代入するHWNDを分けてください。**

**DirectXRenderer-> Qt由来のHWNDを利用**

**Input系全般-> 通常どおりの元々のHWNDを利用**

**こうすることで描画処理とキー入力を両立できます。**

# 入力とレンダリングの闇



**無事,自作したフレームワークを組み込むことができました。**  
**フレームワークの組み込みは手元にいいのがなければ、**  
**授業のシェーダ用の物を使うといいかも**

# 目次

---

- **なんでやろうと思ったのか**
- **どんなことが出来るのか**
- **開発環境の導入**
- **Qtとは、メリットとデメリット**
- **実際に触れてみる**
- **実践的に使うための準備**
- **DirectXのレンダラーをツールと連動させる**
- **ツール上でレンダリングする**
- **自分のフレームワーク上にレンダリングする**
- **入力とレンダリングの間**
- **配布時の注意点**
- **さらに実践的なQtツール制作上のノウハウについて**

## 配布時の注意点

---

**便利なことにQTで作成した実行ファイルを相手が使ったとき、QTのインストール等は必要ありません。**

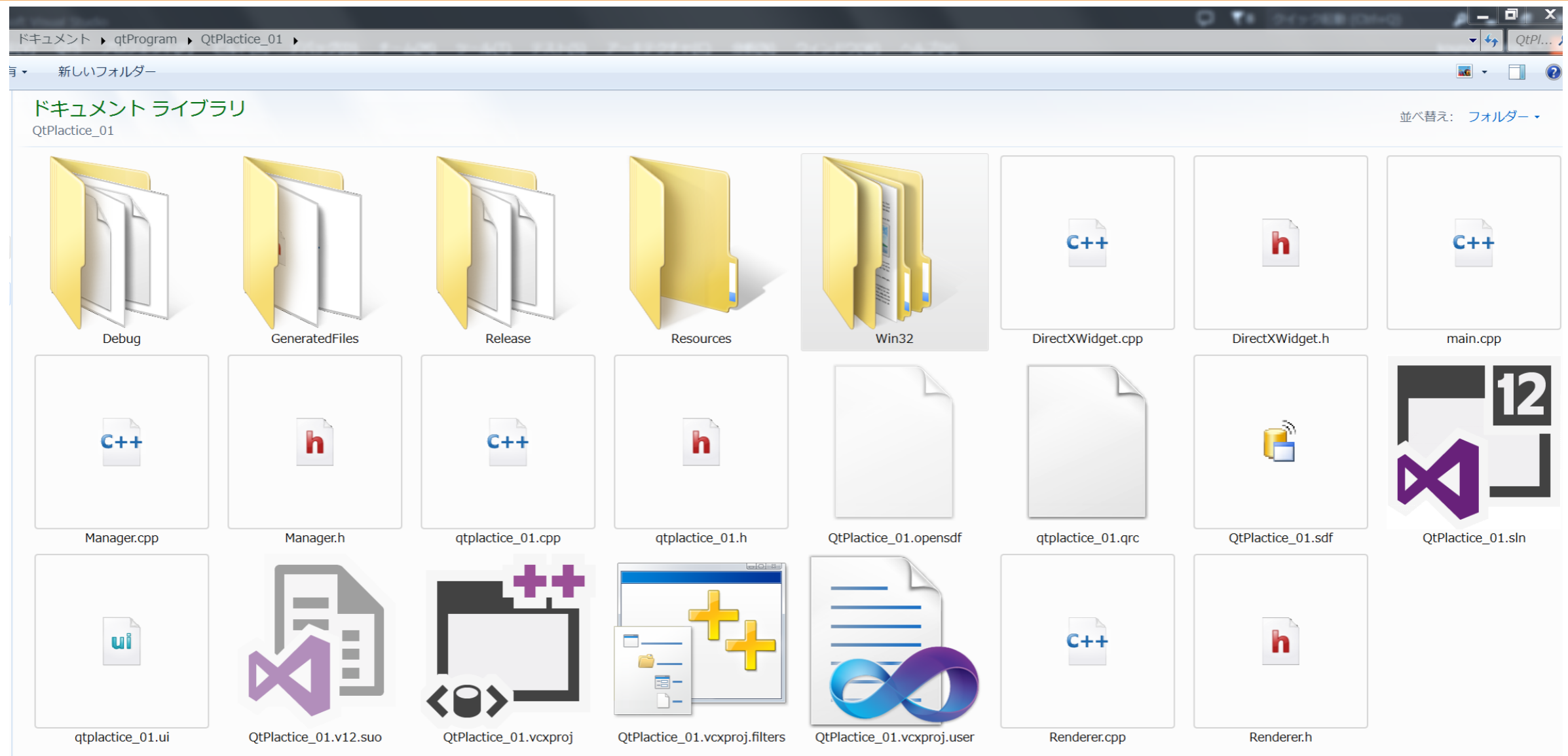
**しかし...**

**ここにも、厄介な罫が隠されています。**

**次に、この罫への対処法を示し、配布先で実行できる方法について解説します。**

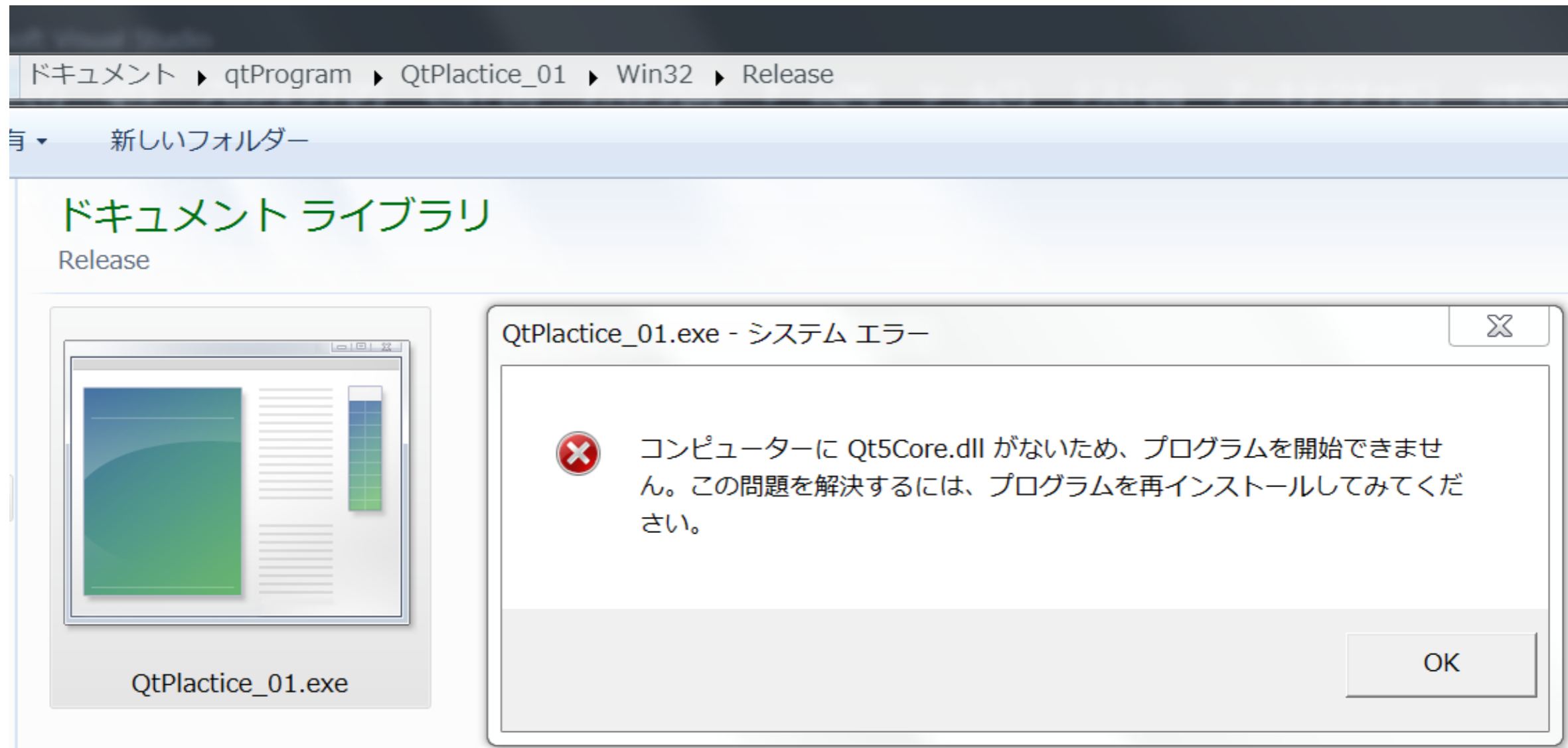


# 配布時の注意点



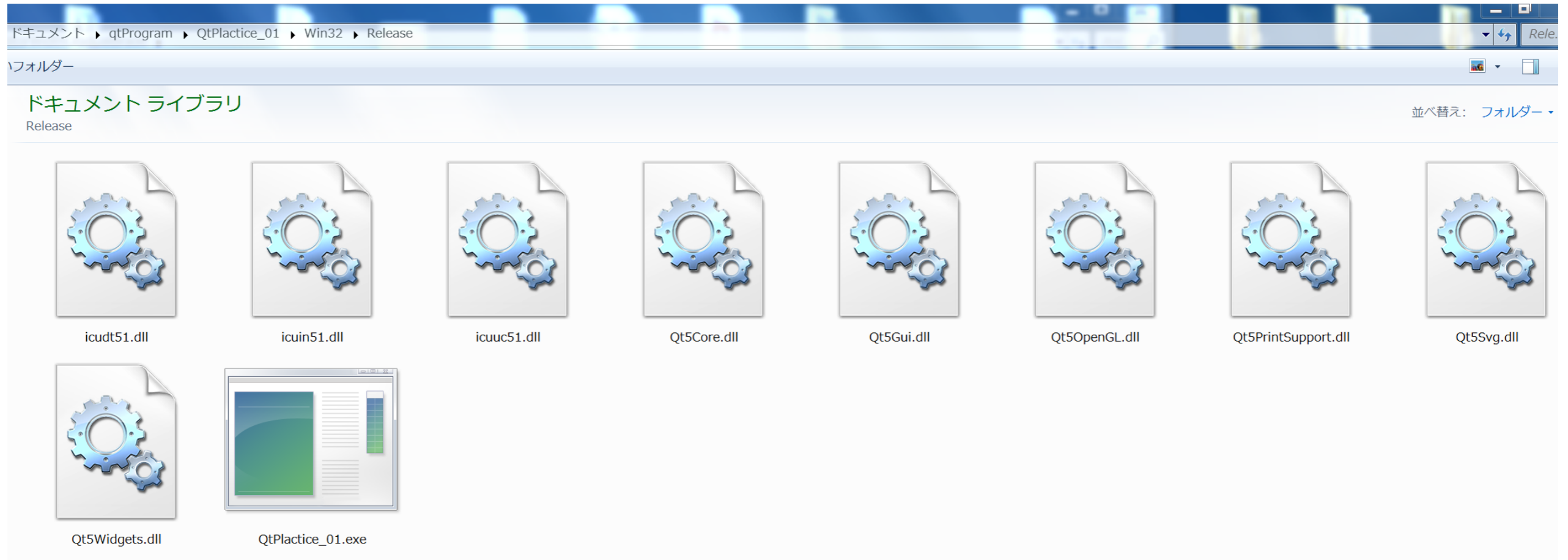
**デフォルトの設定の場合,Releaseビルドを行うと、実行ファイルはWin32フォルダへ出力されます。**

# 配布時の注意点



**そして実行すると、次のようなエラーが出るため、実行できません。エラーのとおりQT用のdllを含めておく必要があるからです。**

# 配布時の注意点

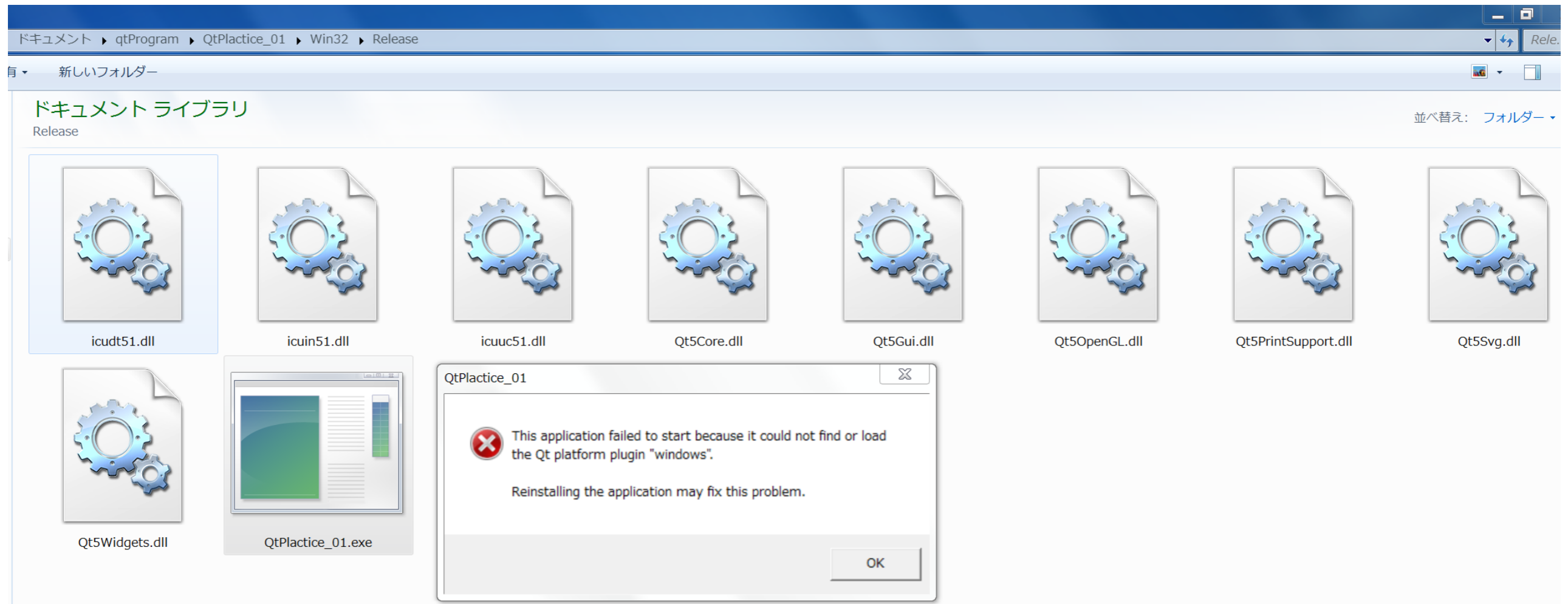


**QTのインストールした場所の中にdllが封印されてるので回収して同じフォルダに入れます。**

**これで作成者の環境ではめでたく起動しますが...**

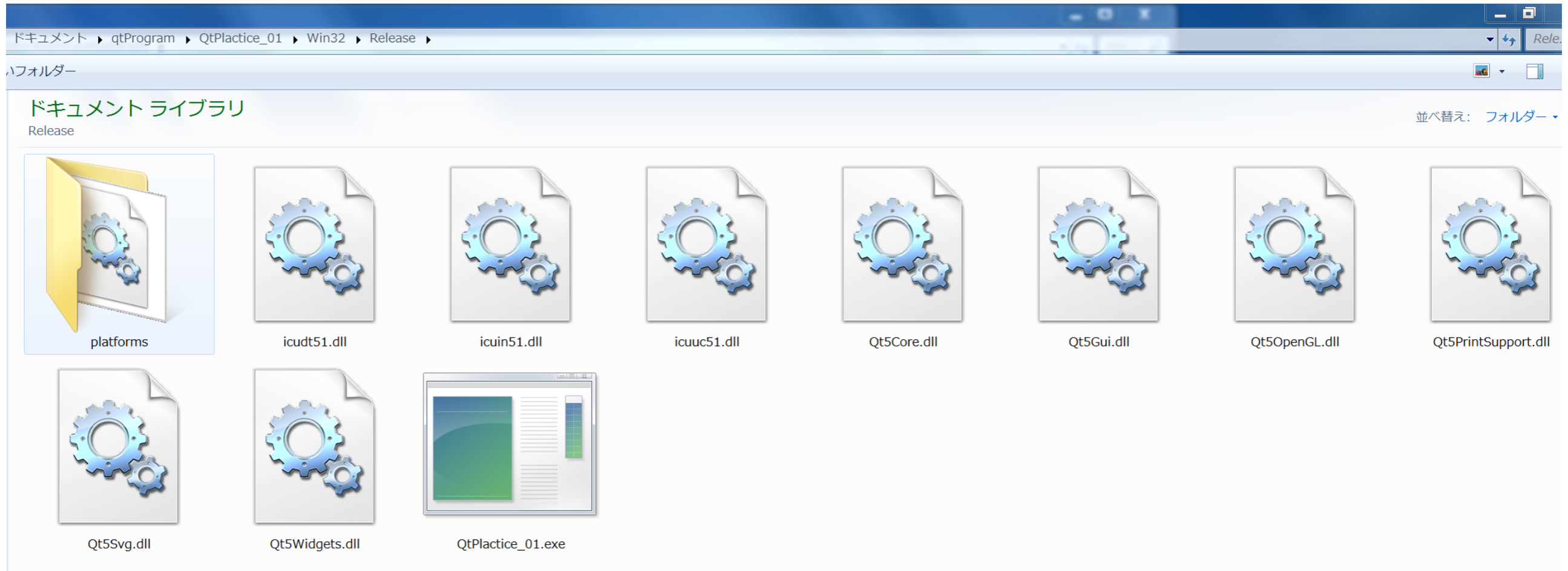
**配布時に最後の罫が起動します。**

# 配布時の注意点



**エラーも意味不明なことを供述しており、だいぶやっかいです。しかし、英語のサイトより解放が示されていたので助かりました。（日本語は皆無）**

# 配布時の注意点



**おもむろにplatformsというフォルダを作ります。  
(決まっているので1文字も間違えてはいけない)**

# 配布時の注意点

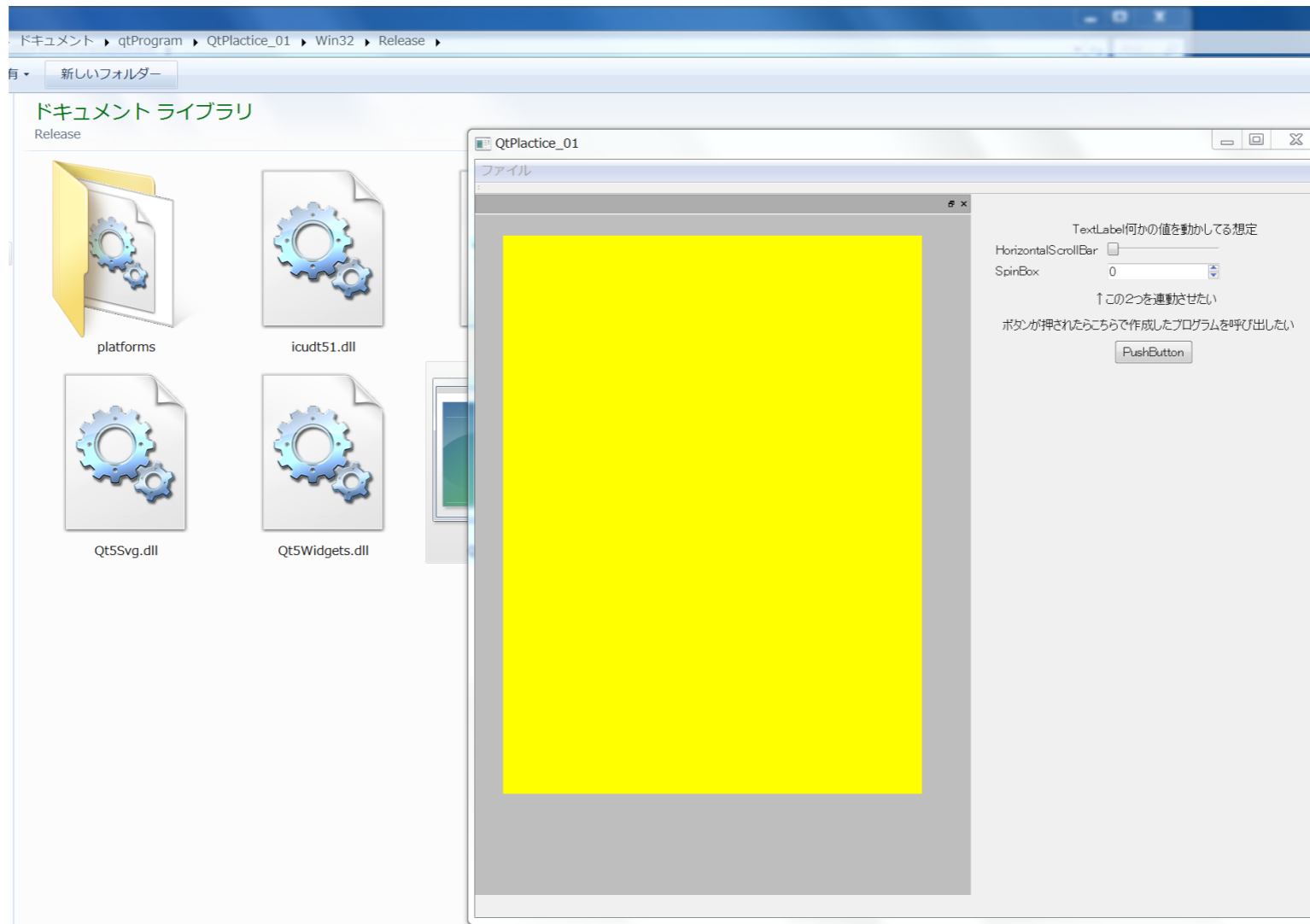


**その中にこいつをいれます。**

**これもインストール時にPC内にくみこまれています。**



# 配布時の注意点



**無事、配布したデータが実行できるようになりました。**

# 目次

---

- **なんでやるうと思ったのか**
- **どんなことが出来るのか**
- **開発環境の導入**
- **Qtとは、メリットとデメリット**
- **実際に触れてみる**
- **実践的に使うための準備**
- **DirectXのレンダラーをツールと連動させる**
- **ツール上でレンダリングする**
- **自分のフレームワーク上にレンダリングする**
- **入力とレンダリングの間**
- **配布時の注意点**
- **さらに実践的なQtツール制作上のノウハウについて**

## さらに実践的なQtツール制作上のノウハウについて

もう十分じゃないかと思われる方もいるかもしれませんが、甘いです。これだけの知識ではQtでDirectXが組み込めて配布できるようになったただけなのです。

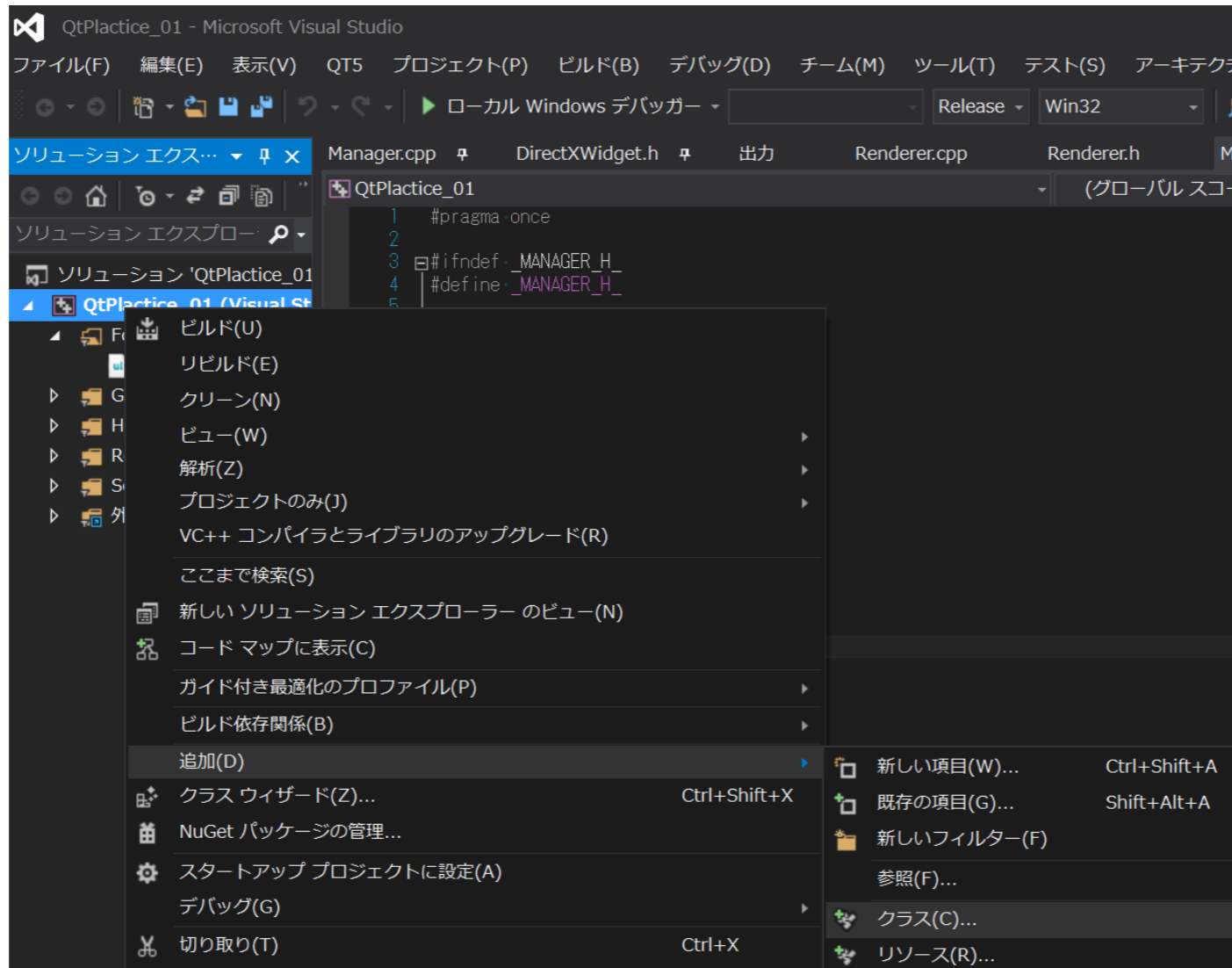
最後にレベルエディターを作るために必要となるであろう知識について解説して、終了したいと思います。



本当の戦いは  
ここからだ！

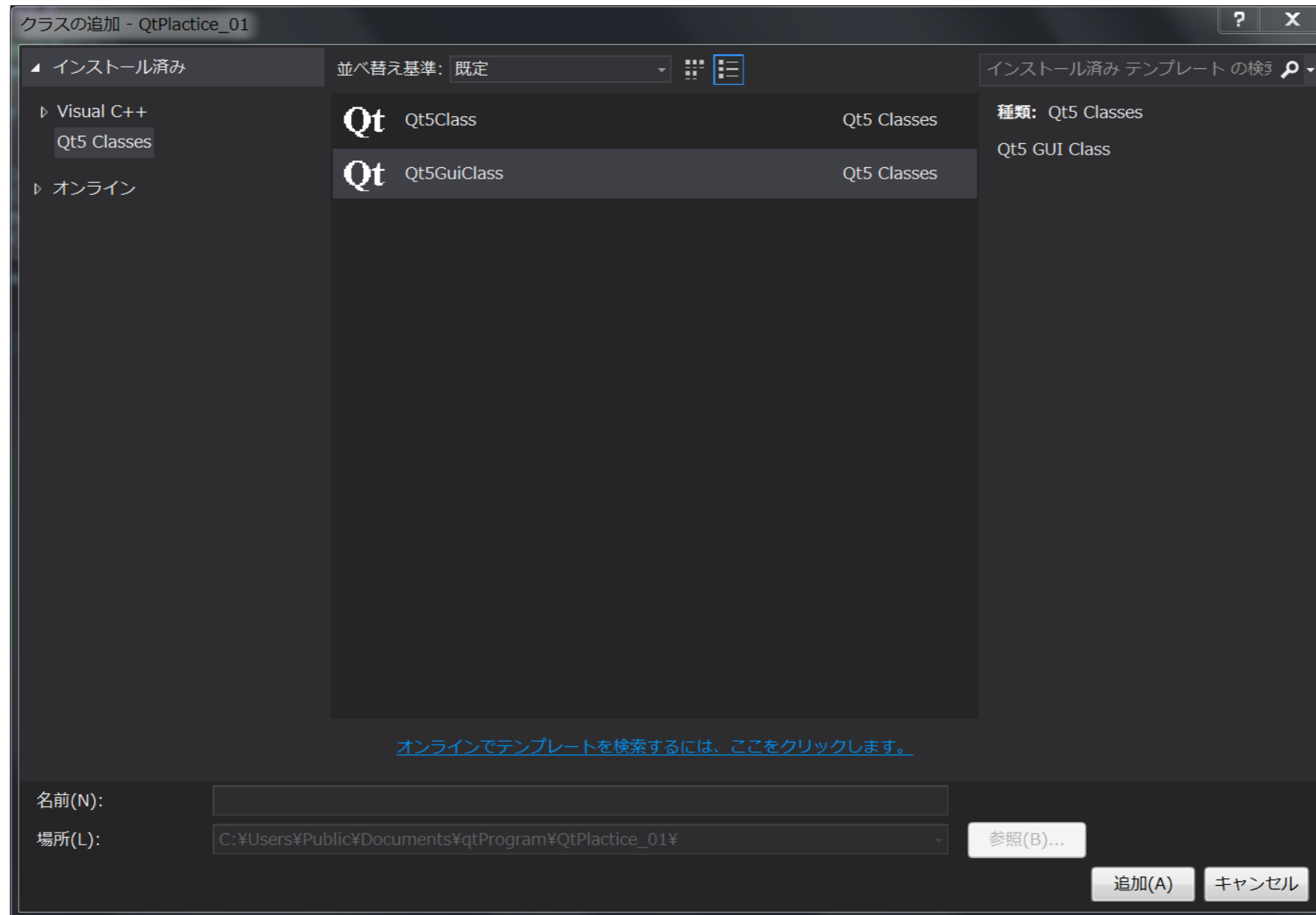
# さらに実践的なQtツール制作上のノウハウについて

## GUIエディターからさらに別のGUIエディターを呼び出す



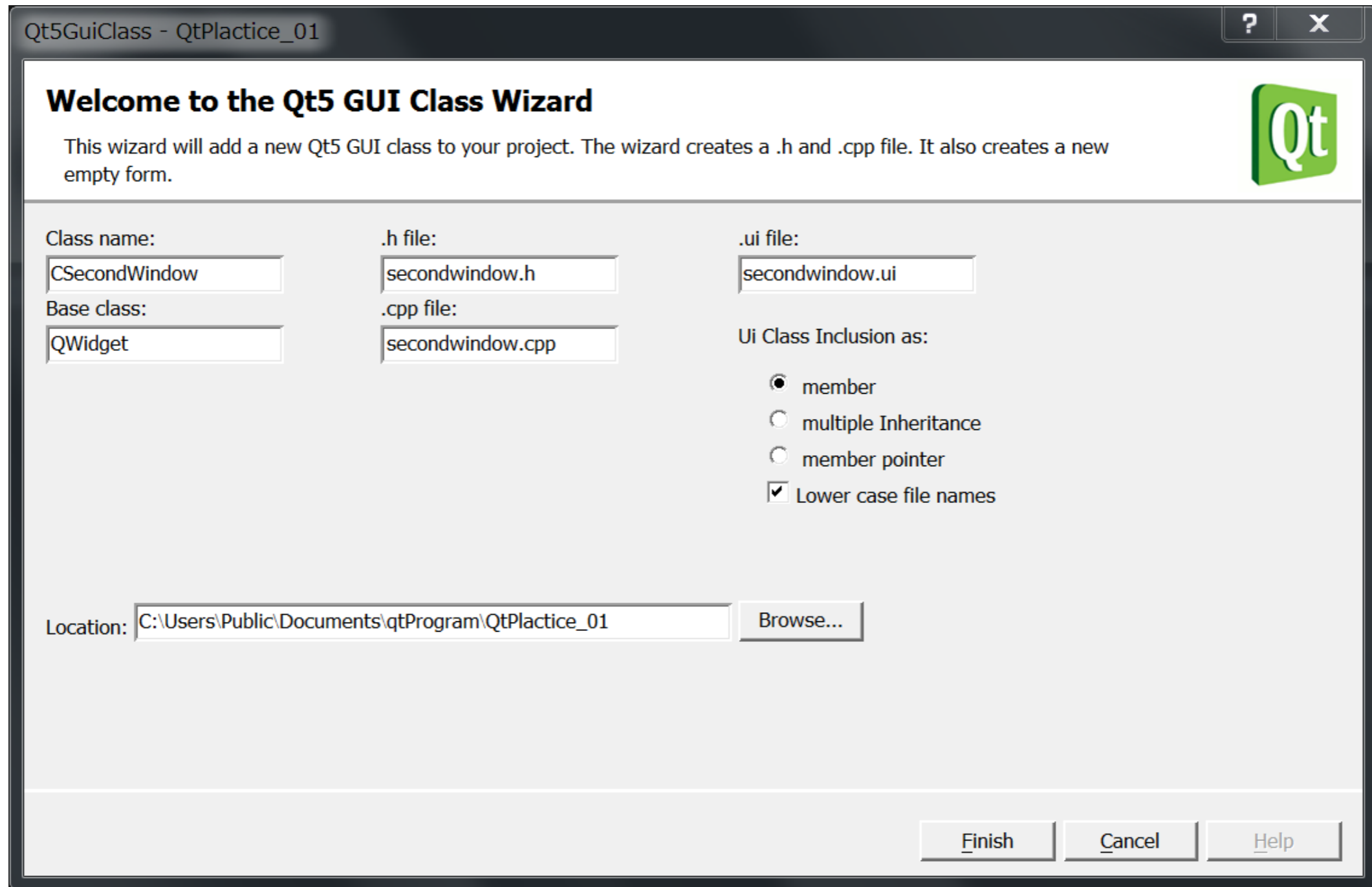
**先ほど作成していたプロジェクトをそのまま使います。  
まずプロジェクトの追加よりクラスを選択してください。**

# さらに実践的なQtツール制作上のノウハウについて



ここではQt5GuiClassを選択してください。

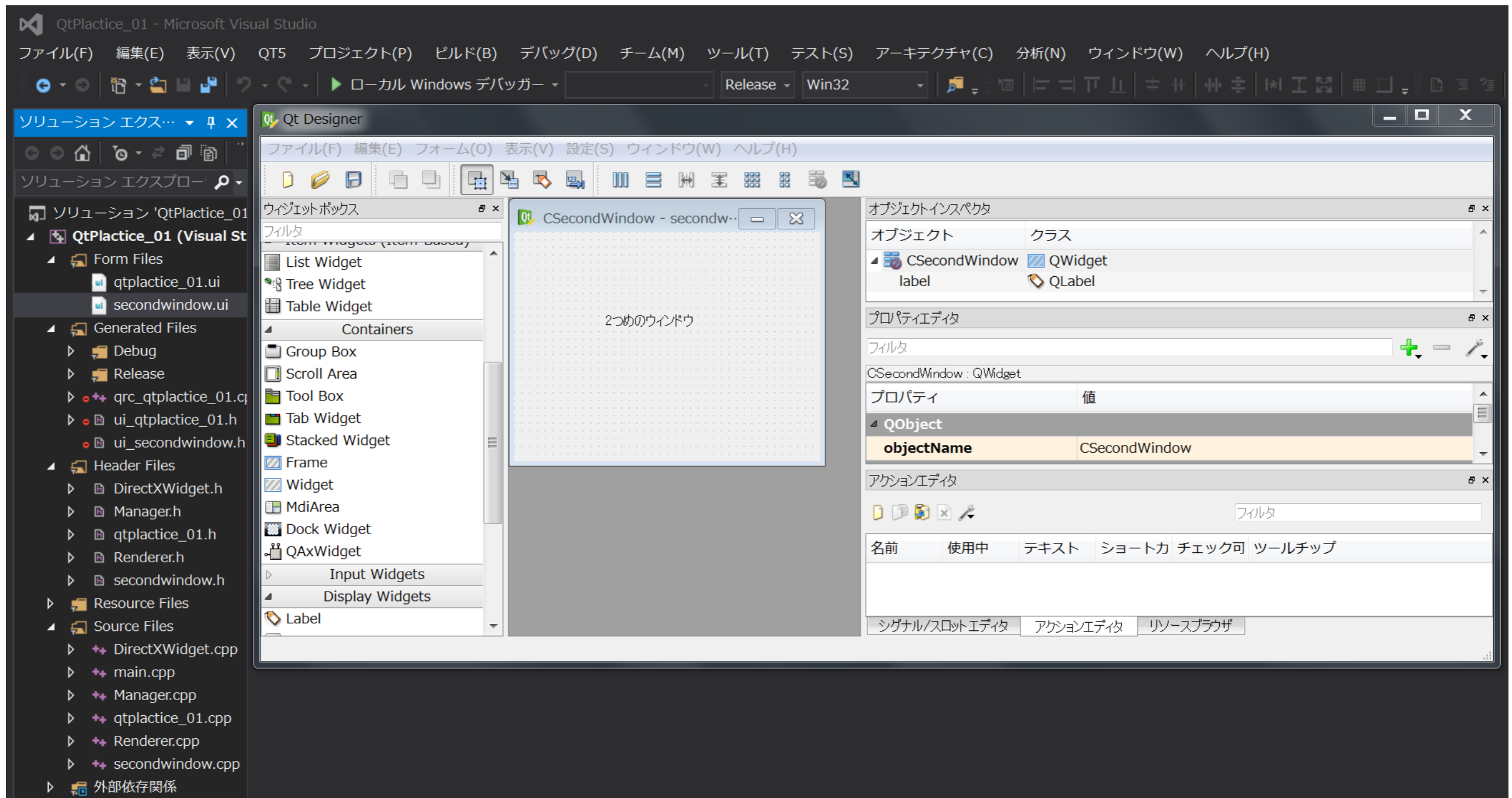
# さらに実践的なQtツール制作上のノウハウについて



**名前は任意のものでいいです。そのGUIエディターでの目的から適切と思われる命名をしてください。**



# さらに実践的なQtツール制作上のノウハウについて



そうすると新しく別の.uiファイルが出来ているので、開くと、新規のGUIエディターが出現します。

# さらに実践的なQtツール制作上のノウハウについて

```
Manager.h  qtpractice_01.cpp  secondwindow.h  出力  qtpractice_01.h  ×
QtPlactice_01
7
8 class CSecondWindow;
9
10 class QtPlactice_01 : public QMainWindow
11 {
12     Q_OBJECT
13
14 public:
15     QtPlactice_01(QWidget *parent = 0);
16     ~QtPlactice_01();
17     Ui::QtPlactice_01Class ui;
18     QTimer m_Timer;
19 private:
20     CSecondWindow* m_pSecondWindow;
21 private slots:
22     void timer_start();
23     // ボタンを押した
24     void on_pushButton_test_clicked();
25     →
26     // ファイルを開く
27     void on_actionLoadFile_triggered();
28 };
29
30 // DirectXをレンダリングするための準備
31 // 本当はここでおくのはよろしくないが説明の都合
32 HINSTANCE → GetHInstance(void);
33 HWND → GetHwnd(void);
34 HWND → GetQtHwnd(void);
35 bool → GetWindowMode(void);
36 void → SetHInstance(HINSTANCE hInstance);
37 void → SetHwnd(HWND hwnd);
38 void → SetQtHwnd(HWND hwnd);
39 void → SetWindowMode(bool flag);
40
```

前方宣言

メンバ変数として  
ポインタで追加する

呼び出すためのプログラムを記述していきます。

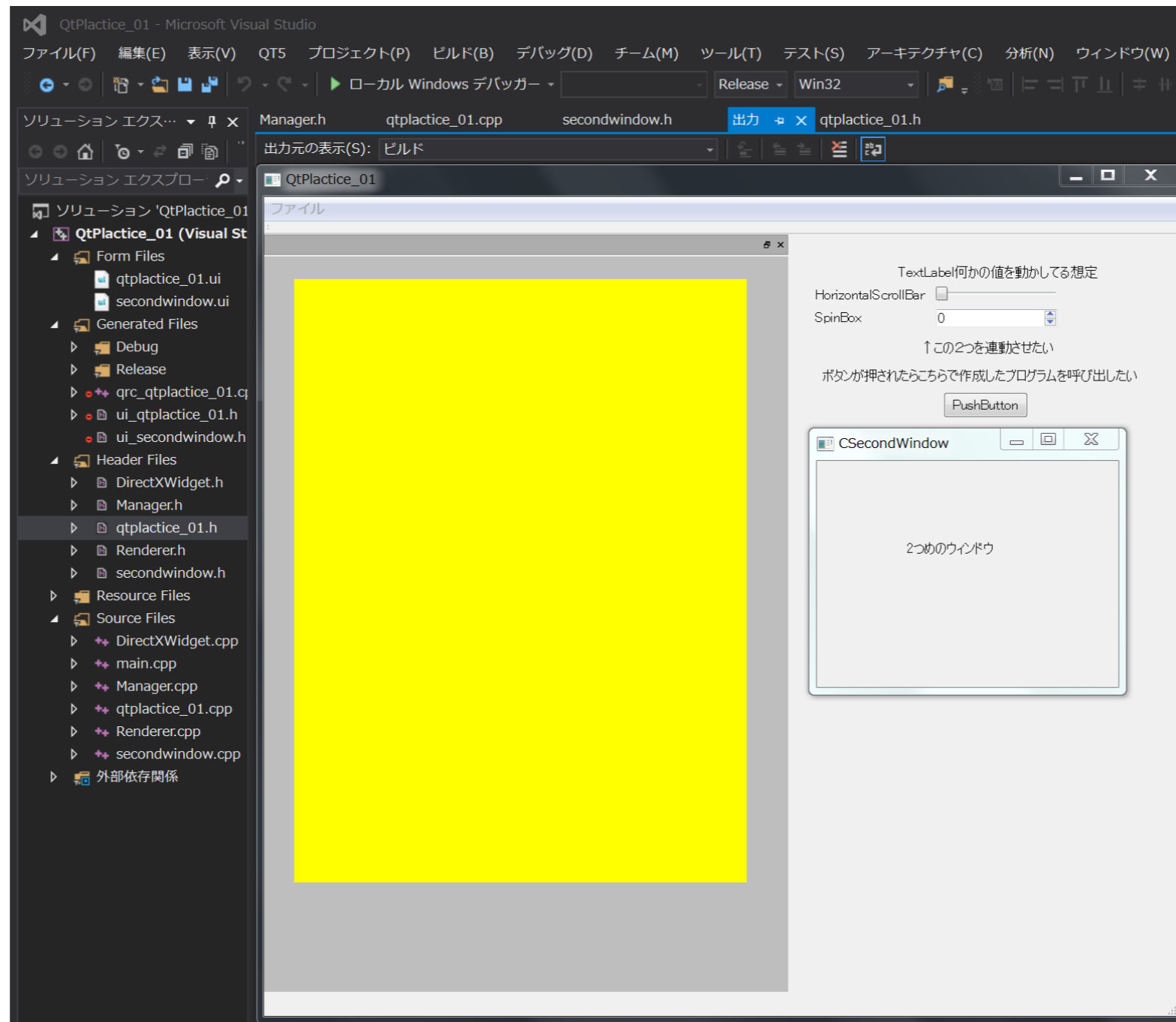
# さらに実践的なQtツール制作上のノウハウについて

```
Manager.h  qtplactice_01.cpp*  secondwindow.h  出力  qtplactice_01.h
QtPlactice_01  QtPlactice_01  timer_start()
1  #include "qtplactice_01.h"
2  #include "secondwindow.h"
3
4  QtPlactice_01::QtPlactice_01 (QWidget *parent)
5  : QMainWindow (parent)
6  {
7  ui.setupUi (this);
8
9  //Timerクラス独自のスロットに接続する
10 connect (&m_Timer, SIGNAL (timeout ()), this, SLOT (timer_start ()));
11
12 //FPS用の内部キュータイマーをスタートさせる
13 m_Timer.start (0);
14
15 m_pSecondWindow = new CSecondWindow;
16 }
17 QtPlactice_01::~QtPlactice_01 ()
18 {
19 delete m_pSecondWindow;
20 m_pSecondWindow = nullptr;
21 }
22 void QtPlactice_01::timer_start () { ... }
29 //ボタンを押したらもうひとつウィンドウが出現
30 void QtPlactice_01::on_pushButton_test_clicked ()
31 {
32 m_pSecondWindow->show ();
33 }
```

インクルード

**ボタンを押すとウィンドウが表示されるようにします。  
(new しただけでは表示はされません。内部的にいるだけ)**

# さらに実践的なQtツール制作上のノウハウについて



**ボタンを押すともう一つのウィンドウが表示されました。  
これでいくらかでも複雑なエディターが作れます。**

# さらに実践的なQtツール制作上のノウハウについて

## セーブ,ロードダイアログの表示

```
DirectXWidget.cpp  DirectXWidget.h  Manager.h  qtpractice_01.cpp  secondwindow.h  出力  qtpractice_01.h
QtPractice_01
1  #ifndef QTPLACTICE_01_H
2  #define QTPLACTICE_01_H
3
4  #include <QtWidgets/QMainWindow>
5  #include "ui_qtpractice_01.h"
6  #include <QTimer>
7
8  class CSecondWindow;
9
10 class QtPractice_01 : public QMainWindow
11 {
12     Q_OBJECT
13
14 public:
15     QtPractice_01(QWidget *parent = 0);
16     ~QtPractice_01();
17     Ui::QtPractice_01Class ui;
18     QTimer m_Timer;
19 private:
20     CSecondWindow* m_pSecondWindow;
21 private slots:
22     void timer_start();
23     // ボタンを押した
24     void on_pushButton_test_clicked();
25     //
26     // ファイルを開く
27     void on_actionLoadFile_triggered();
28
29     // ファイルの保存
30     void on_actionSaveFile_triggered();
31 };
```

ファイルの読み込みと保存のアクションを作成して、シグナルメソッドから呼び出されるようにします。

# さらに実践的なQtツール制作上のノウハウについて

## セーブ,ロードダイアログの表示

```
DirectXWidget.cpp  DirectXWidget.h  Manager.h  qtplactice_01.cpp  secondwindow.h  出力  qtplactice_01.h
QtPlactice_01
QtPlactice_01
on_actionSaveFile_triggered()
1 #include "qtplactice_01.h"
2 #include "secondwindow.h"
3 #include <QFileDialog>
4 #include <stdio.h>
5 QtPlactice_01::QtPlactice_01(QWidget *parent) { ... }
18 QtPlactice_01::~QtPlactice_01() { ... }
23 void QtPlactice_01::timer_start() { ... }
30 void QtPlactice_01::on_pushButton_test_clicked() { ... }
34 //ファイルを開く
35 void QtPlactice_01::on_actionLoadFile_triggered()
36 {
37     QString loadtext = QFileDialog::getOpenFileName(this, tr("Load file"), ".", tr("Test File (*.scene)"));
38     if (loadtext.isEmpty()) {return;}
39     FILE* pFile = NULL;
40     pFile = fopen(loadtext.toLocal8Bit().data(), "rb");
41     //いろいろ読み込んで
42     fclose(pFile);
43 }
44 //ファイル保存
45 void QtPlactice_01::on_actionSaveFile_triggered()
46 {
47     QString savetext = QFileDialog::getSaveFileName(this, tr("Save file"), ".", tr("Test File (*.scene)"));
48     if (savetext.isEmpty()) {return;}
49     FILE* pFile = NULL;
50     pFile = fopen(savetext.toLocal8Bit().data(), "wb");
51     //いろいろ書き込んで
52     fclose(pFile);
53 }
```

図のような風に記述するとファイル名が得られるため、それを用いて、FILEポインタを使ってfread,fwrite等すればいいだけ。カンタン!



# さらに実践的なQtツール制作上のノウハウについて

## セーブ,ロードダイアログの表示

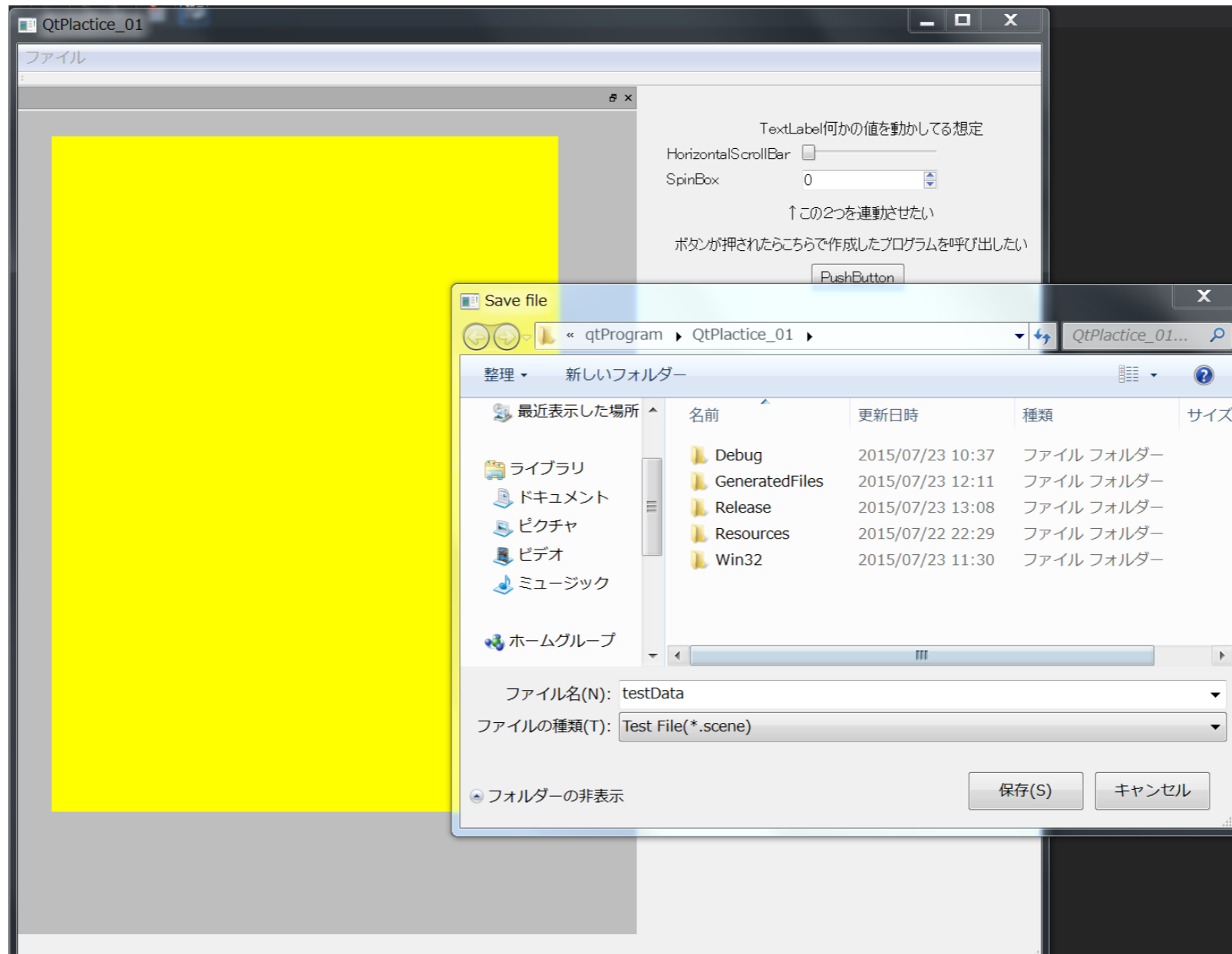
```
DirectXWidget.cpp  DirectXWidget.h  Manager.h  qtplactice_01.cpp  secondwindow.h  出力  qtplactice_01.h
QtPlactice_01
QtPlactice_01
on_actionSaveFile_triggered()
1  #include "qtplactice_01.h"
2  #include "secondwindow.h"
3  #include <QFileDialog>
4  #include <stdio.h>
5  QtPlactice_01::QtPlactice_01(QWidget *parent) { ... }
18 QtPlactice_01::~QtPlactice_01() { ... }
23 void QtPlactice_01::timer_start() { ... }
30 void QtPlactice_01::on_pushButton_test_clicked() { ... }
34 //ファイルを開く
35 void QtPlactice_01::on_actionLoadFile_triggered()
36 {
37     QString loadtext = QFileDialog::getOpenFileName(this, tr("Load file"), ".", tr("Test File (*.scene)"));
38     if (loadtext.isEmpty()) {return;}
39     FILE* pFile = NULL;
40     pFile = fopen(loadtext.toLocal8Bit().data(), "rb");
41     //いろいろ読み込んで
42     fclose(pFile);
43 }
44 //ファイル保存
45 void QtPlactice_01::on_actionSaveFile_triggered()
46 {
47     QString savetext = QFileDialog::getSaveFileName(this, tr("Save file"), ".", tr("Test File (*.scene)"));
48     if (savetext.isEmpty()) {return;}
49     FILE* pFile = NULL;
50     pFile = fopen(savetext.toLocal8Bit().data(), "wb");
51     //いろいろ書き込んで
52     fclose(pFile);
53 }
```

※ **QStringObject.toLocal8Bit().data()**で  
Qt用の文字列オブジェクトからchar\*に変換できる



# さらに実践的なQtツール制作上のノウハウについて

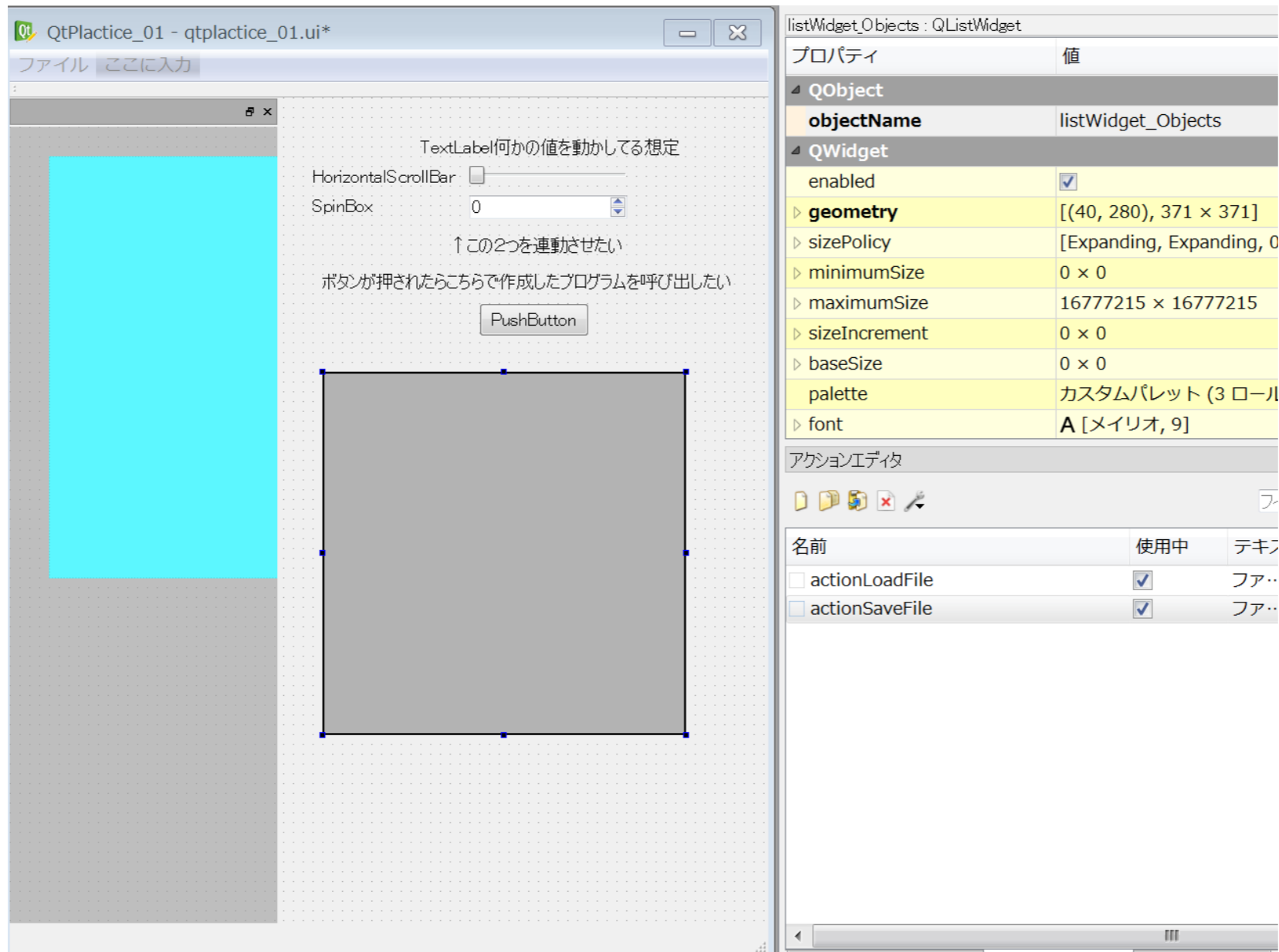
## セーブ,ロードダイアログの表示



問題なくファイルダイアログが開けました。

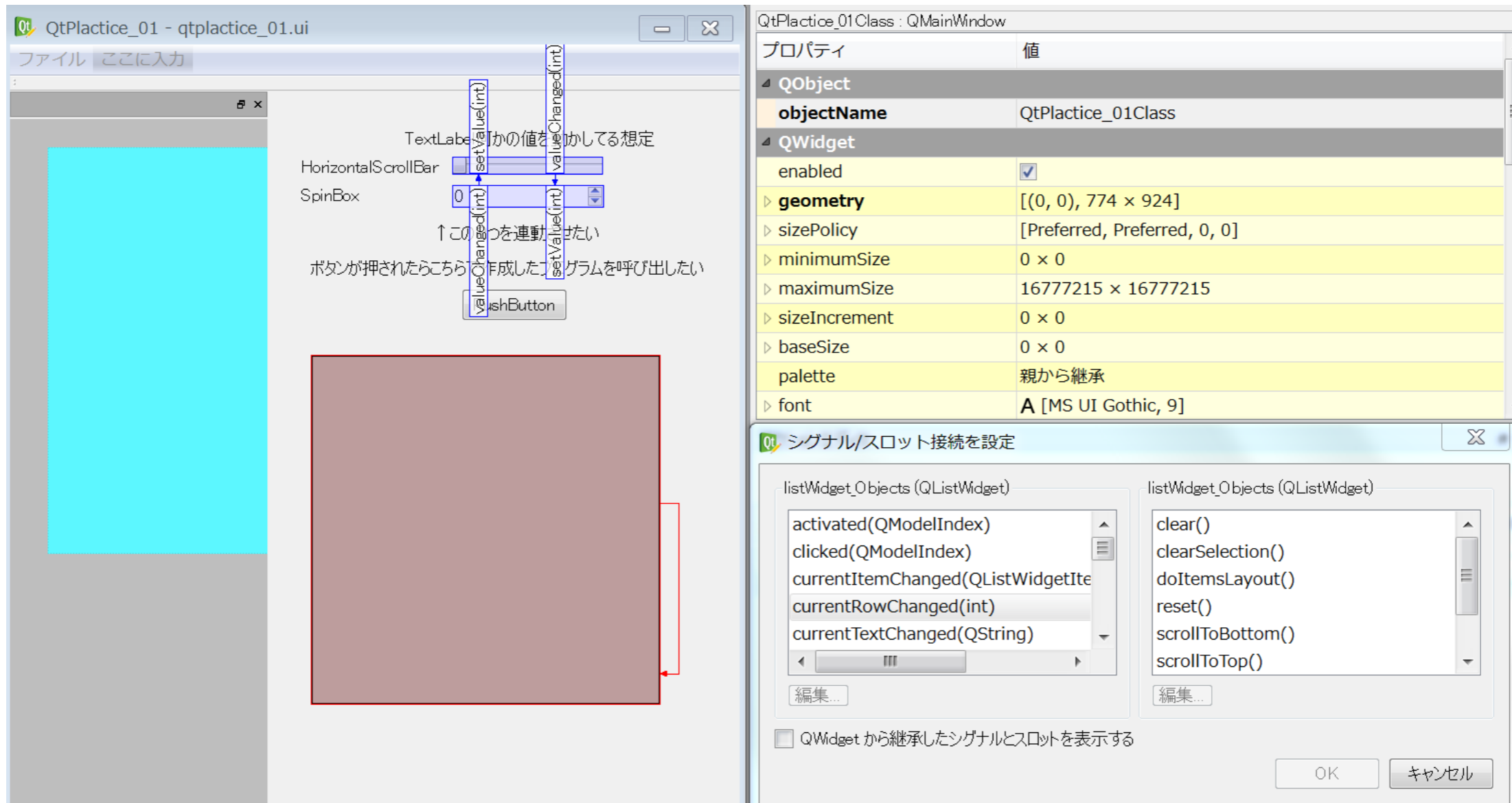
# さらに実践的なQtツール制作上のノウハウについて

## オブジェクトリスト管理



**listwidgetと呼ばれるウィジェットを取り込みます。  
オブジェクト名は図の通りとしました。**

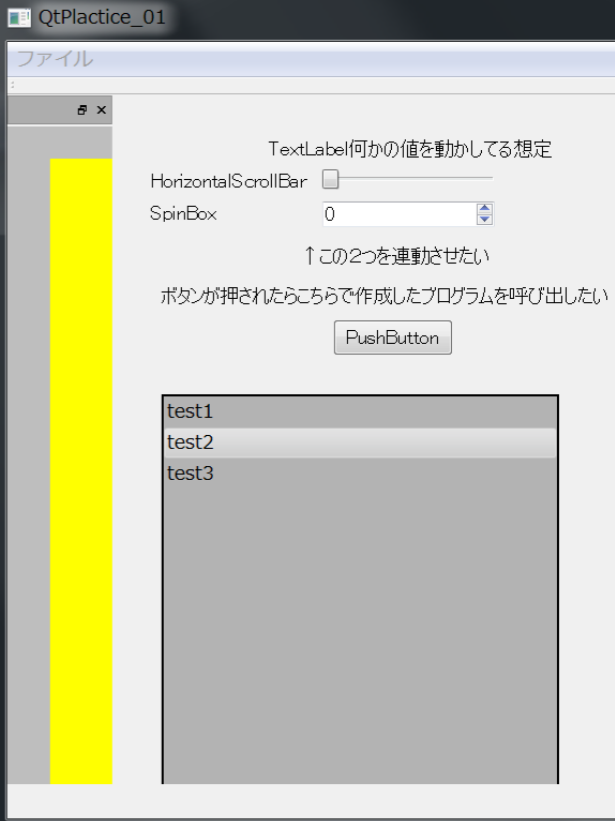
# さらに実践的なQtツール制作上のノウハウについて



**リストの選択が変わったらをシグナルの条件としたいので、メソッド名の確認だけしておきます。**

# さらに実践的なQtツール制作上のノウハウについて

```
main.cpp DirectXWidget.cpp DirectXWidget.h Manager.h qtpractice_01.cpp secondwindow.h 出力 qtpractice_01.h
QtPlactice_01 QtPlactice_01 on_actionLoadFile_triggered()
34 //ファイルを開く
35 void QtPlactice_01::on_actionLoadFile_triggered()
36 {
37     QString loadtext = QFileDialog::getOpenFileName(this, tr("Load file"), ".", tr("Tes
38     if (loadtext.isEmpty()) {return;}
39     FILE* pFile = NULL;
40     pFile = fopen(loadtext.toLocal8Bit().data(), "rb");
41
42     //仮にファイルからいろいろ読み込んで
43     //それをリストに追加するとしよう
44
45     static char* testArray[] = {"test1", "test2", "test3"};
46
47     int MaxFileObjectNum = 3;
48     for (int i = 0; i < MaxFileObjectNum; i++)
49     {
50         ui.listWidget_Objects->addItem(testArray[i]);
51     }
52
53     fclose(pFile);
54 }
55
```



ファイルからオブジェクトのリストがわかると想定して、  
ここではファイル読み込みの処理のあとに  
リストにオブジェクトを追加してみることにします。

# さらに実践的なQtツール制作上のノウハウについて

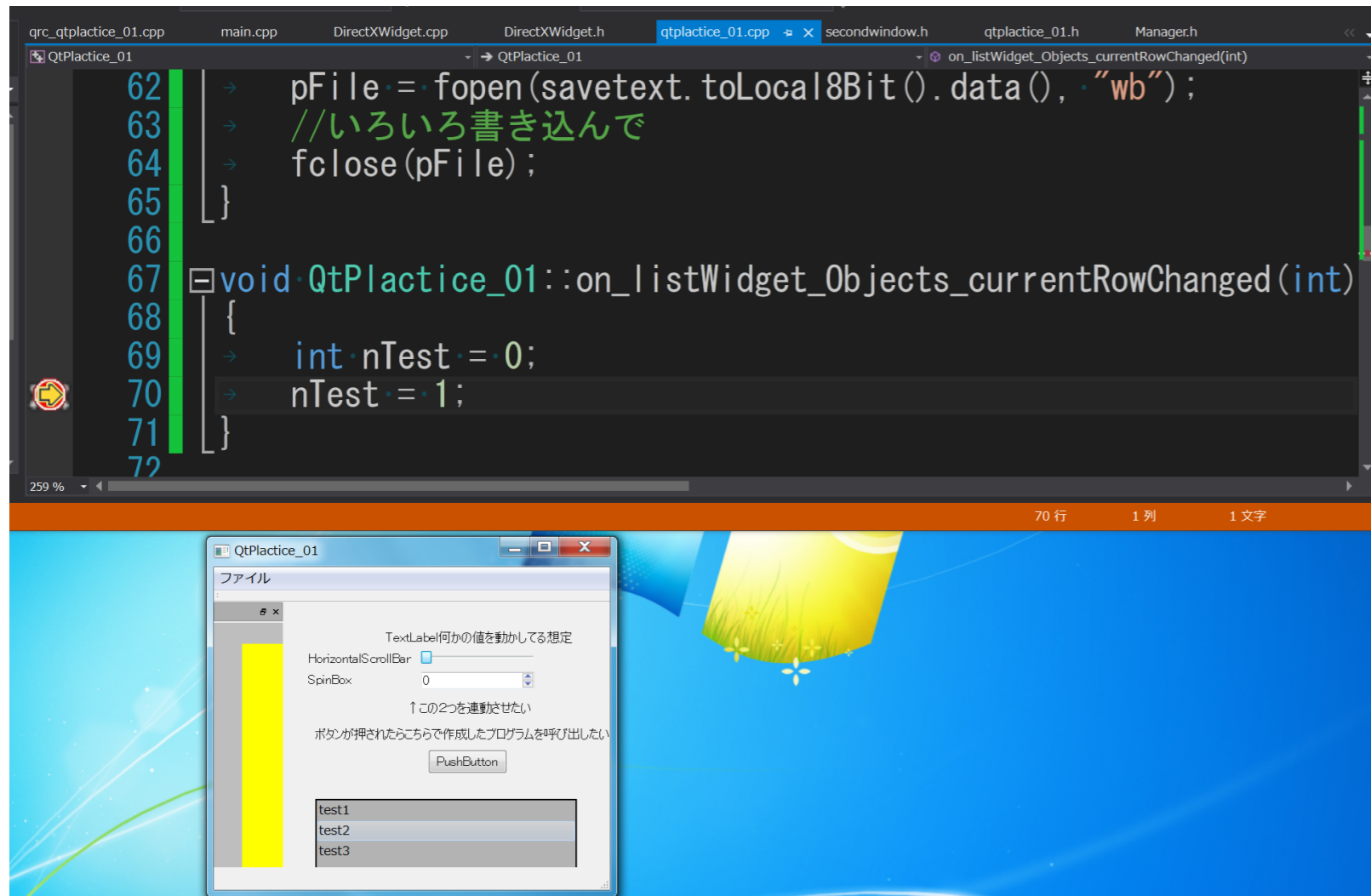
```
出力 qrc_qtpractice_01.cpp main.cpp DirectXWidget.cpp DirectXWidget.h qtpractice_01.cpp secondwindow.h qtplacti
QtPractice_01
16 ~QtPractice_01();
17 Ui::QtPractice_01Class ui;
18 QTimer m_Timer;
19 private:
20 CSecondWindow* m_pSecondWindow;
21 private slots:
22 void timer_start();
23 // ボタンを押した
24 void on_pushButton_test_clicked();
25
26 // ファイルを開く
27 void on_actionLoadFile_triggered();
28
29 // ファイルの保存
30 void on_actionSaveFile_triggered();
31
32 // リストの選択が切り替わった
33 void on_listWidget_Objects_currentRowChanged(int);
34 };
```

ここで書いてない  
場合も呼び出され  
なくなるので注意！

リストの変更時の命名規則も同様です。

**on\_オブジェクト名\_シグナル名(シグナル引数型)**で  
宣言します。今回は(int)の引数型に気をつけてください。  
この型も一致してないと呼ばれません。

# さらに実践的なQtツール制作上のノウハウについて



**対象のインデックスが変化したことを検出しました。**



# さらに実践的なQtツール制作上のノウハウについて

## その他注意点等

- **PosX,Y,Zを設定する用なSpinBoxが会った場合、それらを一度にまとめて変更するような関数を作るとたいていの場合、後で厄介なことになるので、必ずXはXだけ、YはYだけZはZだけが変わるように細かくシグナルメソッドごとに処理分けをすること。**
- **リストのオブジェクトを消すときはBlockSignal(bool)という関数でシグナルを無効にしておかないとエラーで落ちる可能性がある。**