

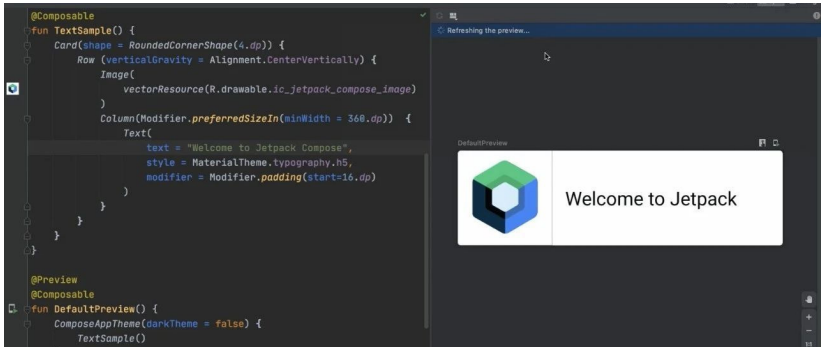
GUI

Bernd Porr

GUI: Introduction

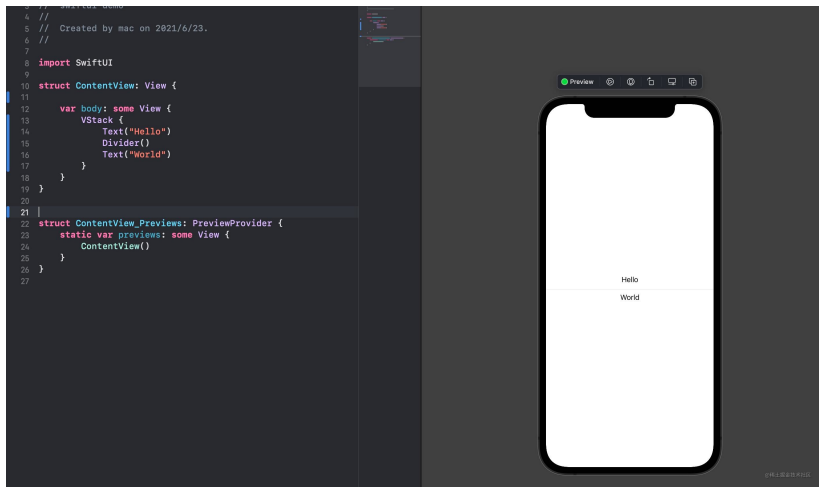
1. Old approach: XML for Layout, code for rest
2. New approach: Everything is code

Example: Jetpack compose



Definition of the layout with KOTLIN class instances.

Example: SwiftUI



Again, definition of the layout with SWIFT class instances.

- ▶ **Qt** is a cross-platform windows development environment for Linux, Windows and Mac written in C++.
- ▶ Elements in Qt are *Widgets* which can contain anything from plots, buttons, text fields or the layout themselves. They are classes.
- ▶ QT works with callbacks using a QT-specific signal/slot concept.

Layout

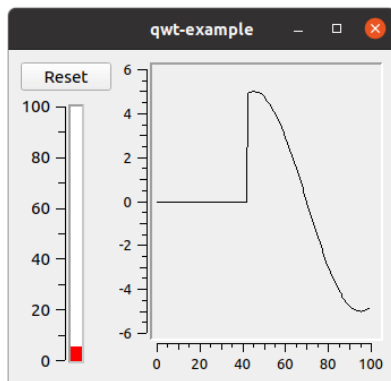


Figure: QT example layout

Widgets are usually organised into nested vertical and horizontal layouts for the result). A lot of other options available.

Layout II

```
// create 3 widgets
button = new QPushButton;
thermo = new QwtThermo;
plot = new QwtPlot;

// vertical layout
vLayout = new QVBoxLayout;
vLayout->addWidget(button);
vLayout->addWidget(thermo);

// horizontal layout
hLayout = new QHBoxLayout;
hLayout->addLayout(vLayout);
hLayout->addWidget(plot);

// main layout
setLayout(hLayout);
```

Events from widgets

A method of a class needs to be combined with the instance pointer. The Qt method “connect” does exactly that:

```
connect(button, &QPushButton::clicked,  
         this, &Window::reset);
```

The QPushButton instance `button` has a method called `clicked()` which is called whenever the user clicks on the button. This is then forwarded to the method `reset()` in the application `Widget`.

Plotting realtime data arriving via a callback

A callback `addSample()` is called in real-time whenever a sample has arrived:

```
void Window::addSample( float v ) {  
    // add the new input to the plot  
    std::move( yData, yData + plotDataSize - 1, yData+1 );  
    curve->setSamples(xData, yData, plotDataSize);  
    yData[0] = v;  
    plot->replot(); // triggers replot but not now  
}
```

which stores the sample `v` in the shift buffer `yData`.

Plotting realtime data arriving via a callback

Then the screen refresh (which is slow) is done at a lower and unreliable rate:

```
void Window::timerEvent( QTimerEvent * )  
{  
    update(); // triggers the update of all Widgets  
}
```

the `paintEvent()` callback

`update()` triggers a paint event and then Qt calls the callback `paintEvent()` “as soon as possible” (which is to say, not in real-time) to repaint the canvas of the widget:

```
void ScopeWindow::paintEvent(QPaintEvent *) {  
    QPainter paint( this );  
  
    paint.drawLine( ... )  
}
```

In Qt timing is not guaranteed

Note that neither the timer callback nor the `update()` function provide reliable realtime timing. So Qt timers cannot be used to sample data but should only be used for screen refresh and other non-time-critical tasks.

...but what about QML???

GUI in QML (javascript-ish) \Leftrightarrow C++ application

The problem is that QML needs to call C++ but we have two very different languages.

... but what about QML? Part 2

QML:

```
Backend {  
    id: backend  
}  
  
TextField {  
    text: backend.userName  
    placeholderText: qsTr("User name")  
    anchors.centerIn: parent  
  
    onEditingFinished: backend.userName = text  
}
```

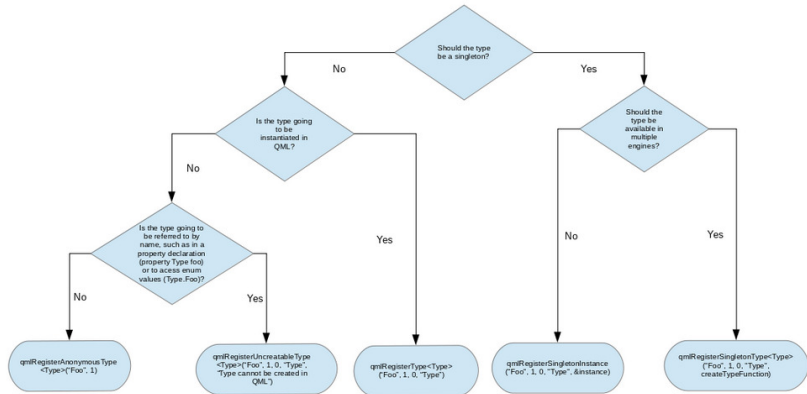
C++

```
class Backend : public QObject  
{  
    Q_OBJECT  
    Q_PROPERTY(QString userName READ userName WRITE setUsername NOTIFY userNameChanged)  
    QML_ELEMENT  
  
public:  
    explicit Backend(QObject *parent = nullptr);  
  
    QString userName();  
    void setUsername(const QString &userName);
```

... but what about QML? Part 3

Choosing the Correct Integration Method Between C++ and QML

To quickly determine which integration method is appropriate for your situation, the following flowchart can be used:



...so in a nutshell: don't do QML.

GUI summary

Stick with C++ for both frontend (GUI) and backend.
Generally, the trend goes towards GUI- and backend in the same language.