

# Convertendo um widget WRT em uma aplicação Qt

## Introdução

Este artigo fornece informações aos desenvolvedores Web sobre como converter widgets WRT (Web Runtime) em aplicações Qt. A IDE usada no artigo é a [Qt Creator](#), mas os princípios básicos são válidos também para outras IDEs como [NetBeans](#) ou [Eclipse](#).

## Criando uma aplicação Qt usando o componente QWebView

1. Execute a Qt Creator IDE.
2. Escolha **File > New File or Project... > Projects > Qt4 Gui Application**.
3. Clique em *OK*.
4. Dê um nome a seu projeto e defina o diretório onde ele residirá
5. Clique em *Next*.
6. Na janela seguinte, você pode escolher os módulos a serem incluídos no projeto. Escolha o *QtWebKit* e clique em *Next*.
7. Na janela seguinte, defina o nome da sua classe de janela como *WRTWidgetWindow* (o campo *Class name*) e desmarque *Generate form* (nesse projeto, os componentes da interface gráfica serão criados manualmente).
8. Clique em *Next*
9. Verifique se as alterações estão corretas e clique em *Finish*. O projeto está aberto e existem dois arquivos de código, *main.cpp* e *wrtwidgetwindow.cpp*, que podem ser encontrados na pasta **Sources** que está em **Projects**.

Agora você está pronto para escrever o código! Abra o arquivo *main.cpp* e modifique-o de modo que fique assim:

```
#include <QtGui/QApplication>
#include "wrtwidgetwindow.h"
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    WRTWidgetWindow window;
    window.show();
    return app.exec();
}
```

Agora, abra o arquivo de declaração *header* para a janela principal, o *wrtwidgetwindow.h* (que está na pasta *Headers* dentro de *Projects*), e modifique-o para que fique assim:

```
#ifndef WRTWIDGETWINDOW_H
#define WRTWIDGETWINDOW_H

#include <QtCore/QPointer>
#include <QtGui/QMainWindow>

class QWebView;

class WRTWidgetWindow : public QMainWindow
{
    Q_OBJECT

public:
    WRTWidgetWindow(QWidget *parent = 0);
    ~WRTWidgetWindow();

private:
    void setupUI();
    QWebView* createWebView();

private:
    QPointer<QWebView> webView;
};

#endif // WRTWIDGETWINDOW_H
```

A classe, que foi declarada acima, representa a janela principal da aplicação. Ela contém um widget Qt, *QWebView* (*webView*), que é usado para exibir conteúdo web (HTML, CSS, JavaScript) no Qt. A seguir, abra o arquivo de implementação da classe (*wrtwidgetwindow.cpp*) e modifique-o de modo a ficar assim:

```
#include "wrtwidgetwindow.h"

#include <QtGui/QFrame>
#include <QtGui/QVBoxLayout>
#include <QtWebKit/QWebView>

WRTWidgetWindow::WRTWidgetWindow(QWidget *parent)
    : QMainWindow(parent)
{
    setupUI();
}

WRTWidgetWindow::~WRTWidgetWindow()
{
    webView->deleteLater();
}
```

```

void WRTWidgetWindow::setupUI()
{
    QFrame* cw = new QFrame(this);
    setCentralWidget(cw);

    QVBoxLayout* layout = new QVBoxLayout(cw);
    cw->setLayout(layout);

    webView = createWebView();
    layout->addWidget(webView);
}

QWebView* WRTWidgetWindow::createWebView()
{
    QWebView* view = new QWebView(this);
    return view;
}

```

Depois disso, pode-se compilar e executar a aplicação. Você pode tentar fazê-lo, embora não exista muita funcionalidade ainda. Existe apenas um componente QWebView vazio na tela.

## Exibindo conteúdo Web no Qt

Vamos agora incluir seu widget WRT no projeto (se você não tiver um, use um daqui, por exemplo: [Web Runtime Stub](#)). Copie os arquivos HTML, CSS, Javascript, recursos (como imagens) para seu projeto Qt. Isso pode ser feito mais facilmente usando-se o gerenciador de arquivos para copiar os respectivos diretórios para a raiz do seu projeto (você guarda os conteúdo web em pastas separadas, não é?). A estrutura do diretório do seu widget poderia ser como esta, por exemplo:

```

html/    (arquivos HTML)
style/   (arquivos CSS)
script/  (arquivos JavaScript)
gfx/     (imagens)

```

Para usar o conteúdo web na aplicação Qt, você precisa criar um arquivo de recursos no seu projeto e incluir conteúdo nele. Para criar o arquivo de recursos, selecione **File > New File or Project... > Qt > Qt Resource file** e clique *OK*. Dê um nome para esse arquivo e escolha o diretório onde ele irá ficar. Escolha *Next* e certifique-se de que o arquivo foi incluído no seu projeto (a caixinha **Add to project** e o item **Project** na lista). Escolha *Finish*. O arquivo de recursos será criado e será incluído automaticamente no arquivo `.pro` **Plain Text Editor**. Inclua o conteúdo web no arquivo como a seguir:

```

<?xml version="1.0" encoding="utf-8"?>
<RCC version="1.0">
  <qresource>
    <!-- arquivos HTML -->
    <file>html/index.html</file>

    <!-- arquivos CSS -->
    <file>style/general.css</file>

    <!-- arquivos JavaScript -->
    <file>script/common.js</file>
    <file>script/main.js</file>

    <!-- imagens -->
    <file>gfx/icon.png</file>
  </qresource>
</RCC>

```

Os recursos serão compilados com a aplicação Qt, mas eles ainda não são usados para nada. Para acessar os recursos dentro do arquivo HTML, é preciso usar atalhos de arquivo relativos, usando `"qrc:/"` antes dos respectivos nomes. Por exemplo, se o arquivo HTML principal do widget WRT for assim:

```

<link rel="stylesheet" href="style/general.css" type="text/css" />
<script type="text/javascript" src="script/common.js" charset="utf-8"></script>
<script type="text/javascript" src="script/main.js" charset="utf-8"></script>

```

então, o arquivo HTML principal na aplicação Qt será assim:

```

<link rel="stylesheet" href="qrc:/style/general.css" type="text/css" />
<script type="text/javascript" src="qrc:/script/common.js" charset="utf-8"></script>
<script type="text/javascript" src="qrc:/script/main.js" charset="utf-8"></script>

```

Altere todos os atalhos nos arquivos HTML como explicado acima (isso não afeta os arquivos CSS, que funcionam como era antes), e mais esta última coisa antes de testar o widget: Abra o arquivo `wrtwidgetwindow.cpp` **Rebuild Project**). Junto com os outros arquivos, os arquivos de recursos serão compilados com a aplicação. Então, execute a aplicação (**Build > Run** ou `Ctrl+R`). O arquivo HTML que foi carregado aparecerá no componente QWebView.

## Considerações sobre a conversão

Assumi-se que o arquivo HTML do widget era relativamente simples (como o do exemplo [WRT Stub application](#)). Na prática, não é bem assim. Esta parte agora apresenta algumas questões a serem consideradas quando se converte um WRT widget em uma aplicação Qt.

### Web Runtime API

Junto com os objetos JavaScript padrões, existem APIs no WRT que permitem que widgets acessem propriedades do sistema, por exemplo. Para ver uma lista desses objetos, acesse a [Web Runtime API reference na Web Developer's Library](#). Como esses objetos são específicos do WRT, eles não funcionam no Qt. Por exemplo, o exemplo abaixo que oculta as softkeys:

```

window.menu.hideSoftkeys();

```

Esse código precisa ser removido da aplicação Qt, e uma outra maneira de se fazer a mesma coisa precisa ser implementada de outra maneira. Por exemplo, para ocultar as softkeys pode-se exibir a aplicação em tela cheia:

```
WRTWidgetWindow widget;
widget.showFullScreen();
```

## S60 Platform Services

O WRT provê diversas APIs JavaScript para se acessar os [S60 Platform Services](#). Como o nome sugere, essas APIs são específicas para S60 (e para WRT), de forma que não funcionam no Qt. Numa aplicação QWebKit, existem duas alternativas para os Platform Services. Para desenvolvedores C++, as [Qt Mobility APIs](#) são uma boa escolha. Elas estão em desenvolvimento e irão fornecer o mesmo tipo de funcionalidade. A outra é o [Qt hybrid application framework](#), que a Nokia está desenvolvendo para prover esse tipo de funcionalidade. A vantagem deste último é que os mesmos serviços podem ser acessados usando somente tecnologias web.

## Escalabilidade dinâmica

Lidar com mudanças na orientação do aparelho, dinamicamente, é uma questão importante quando se projeta aplicações móveis. A técnica apresentada em "[Detecting orientation changes in Symbian Web Runtime](#)" e "[Reacting to the changes in screen size in Symbian Web Runtime](#)" funciona bem no Qt, porque o evento do JavaScript `onresize` pode ser usado a partir do componente QWebKit para lidar com a questão da escalabilidade. A única coisa que pode ter que ser levada em consideração é determinar um valor fixo para a orientação em plataformas que não permitem mudanças de orientação (como por exemplo, no Windows e Linux). Aqui está um exemplo que faz isso acontecer:

```
WRTWidgetWindow widget;
#if defined Q_OS_SYMBIAN || defined Q_WS_HILDON
    // No Symbian and Maemo, mostrar o widget em tela cheia
    widget.showFullScreen();
#elif defined Q_OS_WIN32 || defined Q_OS_LINUX
    // No Windows e Linux, mostrar o widget usando resolução WGA
    widget.setFixedSize(800, 480);
    widget.show();
#endif
```

## Executando código Qt a partir do JavaScript

Em alguma hora, pode ser necessário executar código Qt a partir do JavaScript. Um exemplo disso é para encerrar a aplicação. O código JavaScript `"window.close()"`, que normalmente faria a janela ser fechada, não pode ser usada no Qt para fazer isso. Então, é preciso usar um código Qt e chamá-lo a partir do JavaScript. Aqui está uma função que encerra a aplicação:

```
#include <QtGui/QApplication>

void WRTWidgetWindow::close()
{
    QApplication::exit();
}
```

Para poder chamar essa função no JavaScript, é preciso que a classe do Qt esteja disponível 'no lado JavaScript' (mais tecnicamente falando, a dentro do contexto do QWebFrame, no JavaScript). Acrescente a seguinte função na classe:

```
#include <QtWebKit/QWebFrame>

void WRTWidgetWindow::addJavaScriptObject()
{
    // Fazer com que WRTWidgetWindow fique disponível para o QWebFrame,
    // no JavaScript (na variável "clientApp")
    this->webView->page()->mainFrame()->addToJavaScriptWindowObject("clientApp",
        this);
}
```

A função acima deve ser chamada quando o QWebView for criado:

```
QWebView* WRTWidgetWindow::createWebView()
{
    QWebView* view = new QWebView(this);
    // Chamar o slot addJavaScriptObject quando o sinal javascriptWindowObjectCleared
    // for emitido, i.e. antes que a janela seja carregada
    connect(view->page()->mainFrame(), SIGNAL(javascriptWindowObjectCleared()),
        this, SLOT(addJavaScriptObject()));
    view->load(QUrl("qrc:/html/index.html"));
    return view;
}
```

Acrescente também essas declarações no arquivo *header*:

```
public:
    void close();

private slots:
    void addJavaScriptObject();
```

Depois desses passos, pode-se escrever uma função JavaScript que chama a função `close()` da `WRTWidgetWindow`.

```
// Encerrar a aplicação
function exit() {
```

```
    clientApp.close();  
}
```

Finalmente, chame a função JavaScript, no HTML. Por exemplo:

```
<input id="btnExit" type="button" onclick="exit();" value="Exit" />
```

---

## Projeto exemplo

O widget original do BetaLabs pode ser obtido em [http://www.developer.nokia.com/info/sw.nokia.com/id/60bbf194-224a-44eb-b352-da5ad53069fe/Web\\_Run-Time\\_BetaLabsWidget\\_Example.html](http://www.developer.nokia.com/info/sw.nokia.com/id/60bbf194-224a-44eb-b352-da5ad53069fe/Web_Run-Time_BetaLabsWidget_Example.html)

A primeira versão da aplicação convertida está disponível em [File:BetaLabsClientQtWebKit.zip](#).

---

## Documentação relacionada

Para maiores informações, veja os seguintes exemplos:

- [Exposing QObjects to Qt Webkit](#)
- [Calling an exposed QObject slot from Qt WebKit with JavaScript](#)
- [Connecting to a QObjects signal with JavaScript slot in Qt WebKit](#)

---

## QtWebKitStub

Também está disponível um esqueleto de aplicação (template) para que seja mais fácil começar a desenvolver aplicações com QtWebKit. Para obtê-la, acesse [QtWebKitStub](#).

