



Code less.  
Create more.  
Deploy everywhere.

## Server-driven UI with Hybrid Qt - WebKit Integration

A great user experience requires dynamic content, enhanced by dynamic presentation, and supported by fast plumbing. Product and service designs that fall short in any one of these three critical elements may disappoint users and leave a market opening for competitors. Dynamic content and presentation requirements can only be satisfied with server-based solutions that can be updated long after code is released. Fast plumbing can only be delivered through native coding.

Hybrid Qt - WebKit with HTML5 is optimized for designs requiring the responsiveness and power of native code integrated with large amounts of dynamic content. The hybrid architecture extends the standard Qt C++ framework for high performance cross-platform software development with the QtWebkit module to allow developers to freely combine Qt objects built in C++ with JavaScript, style sheets, and web content. In a hybrid Qt - WebKit implementation the native code is developed, tested, and released with all the rigor and controls one would expect of a mass-produced consumer product. The UI, however, is updated from a server and can change quite frequently.

This paper uses the example of a media center with a UI coded in HTML, JavaScript, and CSS supported by a native music library subsystem developed in Qt C++. This allows, for example, a single firmware build to be dynamically skinned for a variety of different customers. It also allows the integration of a music store, with all content and promotions controlled by the server.

## Table of Contents

1. Overview .....	3
1.1 The people .....	3
1.2 The technology .....	3
2. Using Qt - WebKit Integration.....	4
2.1 Define the subsystems.....	4
2.2 Build the Qt objects.....	5
2.3 Add the Qt objects to the web environment.....	5
2.4 Access the Qt objects from JavaScript.....	7
3. Getting started.....	7

## 1. OVERVIEW

### 1.1 *The people*

Hybrid development processes make maximum use of the skills available in today's development teams. Most of the work is done using existing web skills – user experience design, UI implementation with HTML and CSS, and logic embodied in JavaScript. Developers with experience in web design will need to adapt when developing for embedded and mobile platforms, but they need not start from scratch. Consider a User Experience designer with years of practice developing online shopping portals. Designing a shopping experience for a “leaning back” TV viewer, delivered through a set-top box, will be quite different than designing one for a “leaning forward” consumer sitting in front of a full-screen desktop browser. Performance considerations also come into play when developing for embedded platforms. However, web developers can continue to draw on their technical skills in building experiences using web technologies and working with existing web design tools.

Web skills are augmented by C++ work using the Qt framework. When a desired component is not available in the web environment, a Qt object can be created and integrated. It should be mentioned that even though hybrid development combines web and C++ coding, hybrid teams do not necessarily have “web people” and “C++ people”. The precise allocation of tasks and combination of skill sets varies widely from organization to organization.

### 1.2 *The technology*

The hybrid Qt - WebKit architecture is conceptually simple: web content built with HTML, JavaScript, and CSS interact with native objects built in C++ using the Qt cross-platform framework. The web content and native objects share events and data while presenting one seamless interface to the user.

**Native objects** are built using the Qt C++ framework for high performance cross-platform software development. Objects built using Qt can run as native objects on supported platforms including Windows, desktop and embedded Linux, Mac, and Symbian S60.

**Web content** is handled using the QtWebkit module within the Qt framework. This module integrates the popular WebKit engine into applications, making it easy to integrate web content and native functionality.

**Interaction** between the web content and native objects creates a seamless experience for the user.

With Qt - WebKit integration designs can:

- Embed Qt widgets in HTML pages along with native C++ code.
- Access Qt objects from JavaScript, allowing a web page's script to interact directly with C++ data structures. This interaction between JavaScript and C++ data structures is illustrated in code examples later in this paper.
- Call JavaScript functions in their web page context from C++ code and trigger events in the web page.
- Share client-side storage. This provides access to a database accessible both from JavaScript and C++, making it possible to share large amounts of data easily even when offline.

## 2. USING QT - WEBKIT INTEGRATION

Consider a media center as a use case. It provides local storage and playback of music, video, and still images whether or not the center is connected to the Internet. It integrates with online stores from which a user can purchase media tracks.

Some aspects of the media center clearly need to be server-driven, including access to the online stores and stylistic elements such as branding and themes.

Other aspects of the media center clearly need to be optimized for power and performance. These might include video processing or a local cache of pre-scaled thumbnail images.

This leads to two questions. First, what technology should be used to implement the media center? There are three primary options in Qt, and each has a place in the media center.

**Qt widgets** written in C++ provide high performance and cross-platform capability. A playlist manager would be a good use of Qt widgets.

**Qt Quick** is coming with the Qt 4.7 release, and the [Qt Declarative](#) module would make an eye-catching UI easy to code. An animated view through a stream of album covers would be a good use for Qt Quick.

**Qt WebKit** Hybrid design is best for integrating a lot of server-driven content with native functionality. This approach is ideal for managing an online music store and/or a local music library.

The Qt WebKit Hybrid design integrates native components coded in Qt C++ with web capability coded in HTML, JavaScript, and CSS. This begs the second question: What goes where?

There are no hard and fast answers to this question, but most hybrid applications code “the application” using web technologies and leverage a library of custom-written Qt C++ objects. For the media center this would mean using HTML with server-driven UI for the online store, Qt C++ objects for managing a local music library (a subsystem of the media center), and locally-cached HTML to create the interface to previously-downloaded tracks.

Implementing this system consists of three primary steps: build the Qt C++ objects, add those objects to the web environment, and then manipulate the objects in JavaScript.

### 2.1 Define the subsystems

Define each subsystem of the design in terms of content, actions, events, and transient data types. Here are the particulars for the music library:

- **Subsystem** of the media center
- **Content:** A list of all albums in the music library
- **Actions:** Add a song to the music library; add an album to the music library (also delete album and delete song)
- **Events:** The list of all albums has changed
- **Transient data types:** Album, Song

## 2.2 Build the Qt objects

The music library subsystem will be built in Qt as a named child object. See the online documentation for a full explanation of the [Qt Object Model](#).

- **Content** will be defined as Qt properties and accessible as JavaScript properties.
- **Actions** will be defined as Qt slots and accessible as JavaScript functions.
- **Events** will be defined as Qt signals which can be connected to a variety of JavaScript mechanisms, including for example triggering a JavaScript handler that responds to a Qt C++ signal that the albums have changed. More details are provided in [Using Signals and Slots](#) in the online documentation.

Note that the Qt WebKit environment knows how to transform transient data types to and from JavaScript objects and arrays. This means that using those types for passing arguments between the web environment and the native code (either in signals, slots or properties) allows JavaScript to access the internals of the transient type (for example, the song's title). More details are available in [Default Conversion from C++ to QtScript](#).

Here is a snippet of the code for the music library object. Notice the signal that albums have been changed and the use of QVariant and QVariantList to pass data between the native and web environments.

```
class MusicLibrary : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QVariantList allAlbums READ allAlbums)

public:
    MusicLibrary (QObject* parent = NULL);
    QVariantList allAlbums() const;

public slots:
    QVariant addSong(const QVariant &);
    QVariant addAlbum(const QVariant &);

signals:
    void albumsChanged();

private:
    QMap<QUuid, Song> song_map;
    QMap<QUuid, Album> album_map;
};

class MediaCenter : public QObject
{
    Q_OBJECT
public:
    MediaCenter(QObject* o = NULL);
    MusicManager & music() { return *musicManager; }
private:
    MusicManager* musicManager;
};
```

### 2.3 Add the Qt objects to the web environment

Qt C++ objects are not available to the web environment unless explicitly exposed using the `QWebFrame::addToJavaScriptWindowObject` function. This is important for security reasons as the C++ objects may execute outside of the sandbox of the web environment.

In the following snippet the system is named "MediaCenter". Note the connection of a slot to the web-frame's `javaScriptWindowObjectCleared` signal to make sure the object is added every time the frame refreshes.

```
class MyObject : public QObject
{
    Q_OBJECT

public:
    MyObject(MediaCenter* mc, QWebFrame* f) : QObject(mc)
    {
        frame = f;
        mediaCenter = mc;
        addJS();                               connect(f, SIGNAL(javaScriptWindowObjectCleared(
    )), this,
        SLOT(addJS()));
    }
public slots:
    void addJS()
    {
        frame->addToJavaScriptWindowObject
            ("MediaCenter", mediaCenter);
    }
private:
    QWebFrame* frame;
    MediaCenter* mediaCenter;
};
```

#### 2.4 Access the Qt objects from JavaScript

The MediaCenter object is now accessible like any other JavaScript object. JavaScript code can now activate slots, connect to signals, read/write properties and access child objects.

Here is how to access the child named “music” of the main MediaCenter object:

```
var musicManager = MediaCenter.music;
```

Call the addSong slot when the user wants to add a song:

```
musicManager.addSong(s);
```

Read the allAlbums property to show the list:

```
albums = musicManager.allAlbums;
```

And remember the C++ signal generated when the albums changed? Here is how you connect the signal to the JavaScript’s refresh function to update the view:

```
musicManager.albumsChanged.connect(refresh);
```

## 3.GETTING STARTED

For detailed information about the Qt WebKit implementation including a full class reference, see [Integrating Web Content with QtWebkit](#) in the online documentation, or [download the Qt SDK](#) and start hacking on the included examples.

To drill deeper into the steps involved to create a Qt WebKit hybrid design:

- Learn more about Qt C++ objects in the online documentation of the [Qt Object Model](#).
- [Making a C++ object available to Scripts Written in QtScript \[JavaScript\]](#) provides more detail about the relationship between your Qt objects and your web environment.
- Passing data and interactions between Qt objects and JavaScript are explained in [Default Conversion from C++ to QtScript](#) and [Using Signals and Slots](#).

## About Qt

Qt is a cross-platform application framework. Using Qt, you can develop applications and user interfaces once, and deploy them across many desktop and embedded operating systems without rewriting the source code. Qt Development Frameworks, formerly Trolltech, was acquired by Nokia in June 2008. For more details about Qt please visit <http://qt.nokia.com>.

## About Nokia

Nokia is the world leader in mobility, driving the transformation and growth of the converging Internet and communications industries. We make a wide range of mobile devices with services and software that enable people to experience music, navigation, video, television, imaging, games, business mobility and more. Developing and growing our offering of consumer Internet services, as well as our enterprise solutions and software, is a key area of focus. We also provide equipment, solutions and services for communications networks through Nokia Siemens Networks.



**Code less.  
Create more.  
Deploy everywhere.**

**NOKIA**