WebKit in Qt® and Qtopia®

WebKit in Qt and Qtopia

February 2008

Abstract

This whitepaper discusses the challenges faced when developing applications that require dynamic user interfaces based on Web technologies.

We describe why Trolltech has chosen WebKit as the preferred option for integration of Web technology in Qt applications, and describe benefits and possible use cases enabled by this integration.

Contents

1. Executive Summary

Forthcoming versions of Trolltech's Qt and Qtopia products feature integration of the WebKit open source rendering and scripting components.

The Qt WebKit Integration abstracts the components with highly intuitive Qt APIs, and exploits Qt's underlying infrastructure to retain its renowned cross-platform portability.

The Qt WebKit Integration allows Web and native content to be easily merged on devices and in desktop applications, allowing new opportunities for service innovation. Use cases may include mixing Facebook content with local address book details, or populating a scripted idle screen weather widget with live updates.

WebKit allows script based – as opposed to code based – development of a UI. This allows for faster design, code and test iterations than compiled code because there is less overhead for deployment. Scripts can easily be downloaded to a device without worrying about run-time linking.

The additional functionality added onto WebKit for Qt enables new approaches to user interface design, deployment of functionality and richness of the user experience. Not only can new services be rapidly developed, they can easily be updated in the field.

WebKit is a key element in the evolution of user interfaces in Qt and Qtopia, where Web and local content is seamlessly blended to provide the best possible user experience.

Demonstration at Mobile World Congress Barcelona: 11-14 February, 2008
The WebKit demonstration on the Trolltech stand at MWC Barcelona shows the benefits of WebKit with scripted live idle screen widgets, Web content integrated with a local application, and a small footprint browser.

2. Rationale

As access to the Internet becomes more widely available in home environments and consumer devices, Web technologies are increasingly becoming more common as part of end-user functionality. Indeed, many applications now require a solution to the problem of accurately rendering Web pages for common features, such as providing interactive help or enabling access to online data.

2.1. Web Technologies Provide Rich Experiences

As the Web technologies used by browsers improve and begin to converge with those traditionally associated with native applications, fully-featured Web browsing components become necessary to handle highly-dynamic information on all platforms. Even "simple" help systems used by applications now take advantage of modern Web standards to generate dynamic and interactive content, and users increasingly expect to be able to define the way information is integrated with their application/appliance and the Web.

More complex applications have a strong interaction between the data fetched online from a Web server and rich client applications running locally. Having fast access to online data and an easy way to integrate this data becomes mandatory. The integration needed here goes much deeper than simply embedding a browser widget; the application may need to call scripts embedded into documents, and documents may need to call back into the application through a set of interfaces. Two prominent examples of these types of applications are iTunes® and Rhapsody®.

2.2. Simplifying the User Interface

At the same time there is a move towards more declarative ways of building user interfaces. HTML-like user interfaces are now commonly used in both desktop applications and embedded scenarios, and using a Web-like design approach for these user interfaces can greatly simplify the process of creating a modern, consistent and unique user experience and its delivery to the end-user. This way of building user interfaces demands easily embedded native components in the declarative environment. In HTML, this can be achieved through the use of the `<object>` tag and scripting bindings to the native component.

3. The Trolltech Solution

Trolltech now offers a way to address these challenges and to deliver hybrid Web/rich client applications that provide the best of both worlds.

With the forthcoming versions of Qt and Qtopia, Trolltech will integrate the fully featured WebKit HTML rendering engine. WebKit is developed and maintained co-operatively by Apple, Trolltech and Open Source community projects. The community involvement ensures rapid development and stabilization of new features, and open source also ensures the continued availability of the WebKit technology.

As well as the benefits listed above, WebKit integration was driven by a number of other factors:

1. **Size:** The WebKit codebase, as well as the compiled engine itself, is relatively compact. This makes the whole codebase easier to understand, modify and fix – as well as adapting for embedded scenarios which are more resource constrained.
2. **Features:** WebKit is a standards-compliant browser engine, supporting all of the features required to be compatible with today's content on the Web.
3. **Existing Skills:** The WebKit project is a fork of KDE's KHTML browser engine. Trolltech employs a number of the key developers that have worked on KHTML, including the founder of the KHTML project who maintained it from 1998 to 2003. This gives a unique inside knowledge of the codebase.
4. **Licensing/Open Source:** WebKit is licensed under the LGPL (and partly BSD), a license that makes it suitable for integration into a commercial framework.
5. **Clean design:** WebKit's internal design is clean and sound, making it easy to work with the code base, modify it to meet requirements, add features and fix bugs.
6. **Industry acceptance:** WebKit has been chosen by Apple, Google, Nokia a number of other leading companies as their preferred browser solution.
7. **Acceptance by content providers:** As WebKit is the engine behind Apple's Safari Web browser, the engine is accepted as a major engine by most content providers.

In summary, WebKit and the Qt WebKit Integration provides commercial Qt and Qtopia users with exciting new user interface possibilities while building on the best qualities of Open Source: high quality, dynamically evolving technology.

4. Features Overview

The features of the Qt WebKit Integration can be divided into two parts: Web-specific features provided by WebKit itself and the benefits of integrating the browser engine into Qt. We do not discuss specific features of Qt here, but instead refer the reader to the *Qt Whitepaper* for more information:

http://trolltech.com/products/qt/learnmore/whitepapers

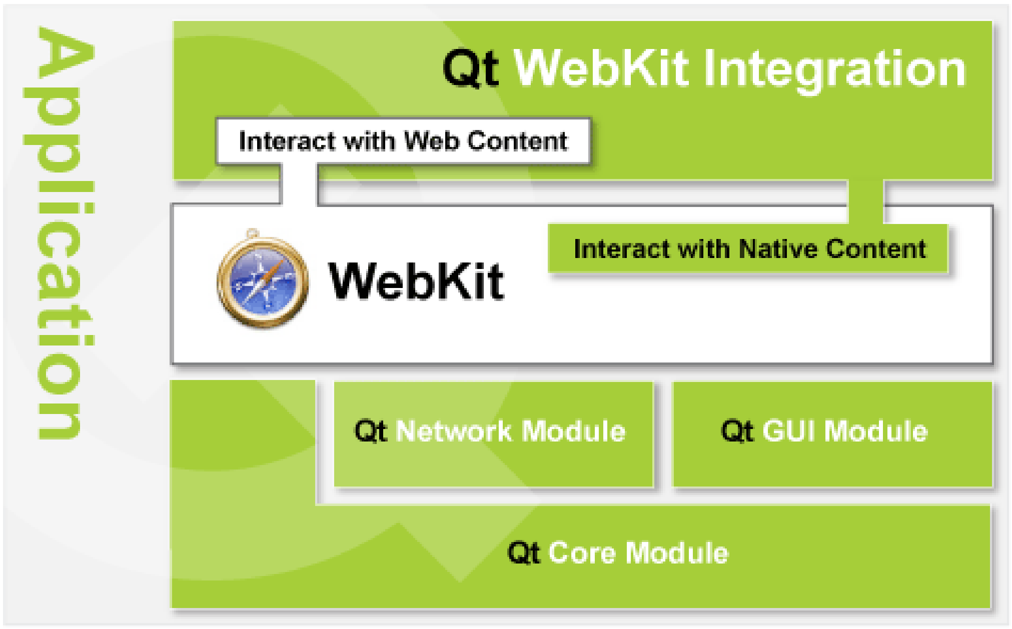

Figure 1: The architecture of the Qt WebKit Integration.

4.1. Web Standards Support

WebKit is a fully-featured open source Web rendering engine with a focus on stability and performance.

The WebKit engine currently used by Trolltech (the Safari-3 branch) supports the following Web standards:

- **HTML 4.01**, **XHTML 1.1** and **XML** for markup of content.
- **CSS 2.1** to describe page layout in a way that separates style from content.
- **JavaScript 1.5** adds scriptability to Web pages.
- **HTML editing**, **HTML canvas** and **AJAX** enable interactive Web applications to be created.
- **XSLT** and **XPath 1.0** allow content to be queried and transformed in the browser.

It also partially supports most of the SVG 1.2 specification and parts of the CSS 3 specification.

4.2. Native Application Integration

The Qt WebKit Integration goes beyond just rendering HTML. To utilize the power of WebKit and Qt together, additional functionality has been added.

Although WebKit is a complete Web rendering and layout engine, it requires platform-specific functionality for infrastructure and display. For this reason, the integration of WebKit in Qt is implemented using Qt primitives. The WebKit backend is implemented using the Qt API to provide drawing and network communication. In addition, parts of the WebKit API are wrapped in Qt-style classes to ensure a uniform and intuitive API across all modules conforming to Trolltech guidelines.

Beyond its ability to show Web pages and execute associated JavaScript functionality, the integration of WebKit in Qt adds the ability to show rich controls as part of a Web page, interact with Web Content using JavaScript from the Application, and interact with native content from the Web environment.

5. In Depth: Technology Details

The Qt WebKit Integration exposes features of WebKit to applications using Qt's paradigms. For example, support for Qt's "signals and slots" inter-object communication mechanism makes it easier for developers to coordinate communication between Web components and traditional rich client objects.

5.1. Examples

The following examples show some of the principal ways in which the WebKit engine can be used via the API provided by the Integration.

Displaying Web Content

When using WebKit in Qt, it is simple to show Web content in the same way as creating a basic control for the user interface:

```
QWebView webView;
webView.load("http://www.host.com/");
webView.show();
```

The Web content will be fully interactive, following links and running JavaScript functionality. The Web component will also provide signals and slots**###**, making it easy to specify functionality and application flow in code and in Qt Designer.

Extending the Web Environment

When it is necessary to reuse existing components, or complement the control set of HTML – the WebKit integration in Qt can be used to allow access of native controls in HTML. This is done by specifying an object tag with an appropriate identifier:

```
<OBJECT type="application/x-qt-plugin" classid="CLASSNAME"
        width="100" height="50"/>
```

With the above type specifier, the CLASSNAME and remaining arguments will be passed to the native WebKit component, where the developer will examine the passed in arguments and return a component as appropriate:

```
QObject* MyWebPage::createPlugin(const QString &classid, const QUrl &url,
         const QStringList &paramNames, const QStringList &paramValues)
{
    if (classid=="CLASSNAME")
        return new MyComponent;
}
```

Interacting with Web Content

The Web environment is also fully available from the native application side. Here, the developer can issue JavaScript commands to interact with dynamic Web content:

```
webView->mainFrame()->evaluateJavaScript("map.setCenter(bounds.getCenter());");
```

In this case, code to center a map component is executed.

Interaction between Script and Native Code

With WebKit there is full bi-directionality between the native code and the Web envionment. JavaScript code can access methods and properties in native Qt objects and native Qt objects can invoke JavaScript functions. The magic behind this is Qt's meta-object system, which is also used to implement inter-object communication with signals and slots.

Below is a simple class with a method and a signal. It is instantiated, given a name, and added to the JavaScript environment. Location::getLocation() is declared as a slot, which automatically makes it accessible from JavaScript. Location::locationChanged() is declared as a signal so it can be "connected" to a JavaScript function – this will automatically invoke the JavaScript function when emitted.

```
class Location :  public QObject
{
    Q_OBJECT;
public:
    Location()
public slots:
    int getLocation()  return location;
signals:
    void locationChanged();
private:
    int location;
};

Location *myLocation = new Location();
myLocation->setObjectName("location");
webView->mainFrame()->addToJavaScriptWindowObject(
    myLocation->objectName(), myObject);
```

The listing below shows the JavaScript code. This first section shows how the Location::getLocation() method is invoked from the Web environment's script context.

```
updateLocation :  function() {
    var myPosition = location.getLocation();
}
```

Next, the Location::locationChanged() signal is connected to a JavaScript function:

```
location.locationChanged.connect(this, this.JSlocationChanged);

JSlocationChanged :  function() {
    // do something useful here
}
```

This bi-directional interaction between JavaScript and Native code is very powerful. An application's user experience can be updated or customized by users without needing to change native code or re-deploy binaries.

5.2. Underlying Technologies

HTTP stack

The Safari-3 branch of WebKit does not fully support HTTP 1.1 (RFC 2616). However, the Qt WebKit Integration provides an HTTP implementation that is 100% compliant with the HTTP 1.1 specification.

SSL Support

Qt provides an easy-to-use API for secure TCP connections using a Secure Sockets Layer (SSL) implementation. The implementation uses OpenSSL* internally and comes with a default certificate bundle that can, of course, be customized. There is a transparent integration between Qt's HTTP implementation and the SSL abstraction, allowing a transparent way of accessing secure content over the HTTP stack.

Proxies

Qt provides an elegant abstraction layer for proxying. Both Socks5 and HTTP transparent proxies, as well as pure HTTP proxies with and without authentication, are supported.

Cookies

Web services and applications issue "cookies" to clients as a means of storing session information. The Qt WebKit Integration supports cookie handling, ensuring that applications correctly interact with popular online services and e-commerce sites.

Web Settings

The Qt WebKit Web settings abstraction allows the developer to specify typical browser defaults that would affect the appearance of Web pages in a browser. The primitives which can be modified include standard font, automatic loading of images, enabling/disabling JavaScript, setting missing graphic icon, user style sheet, etc.

Security

When exposing native functionality to remotely managed resources, there is an immediate risk of compromising local security. WebKit solves this by allowing the developer to manage the capabilities of the Web environment and manage network traffic.

Local objects that are exposed to JavaScript will automatically make their slot methods and properties for themselves and child objects visible to the script environment. By disabling JavaScript altogether this will not be accessible in the Web environment.

With JavaScript is enabled, only objects which are explicitly exposed will be accessible. When developing mixed-mode applications using WebKit, care needs to be taken to only expose necessary methods and objects.

Network traffic in WebKit can be managed by intercepting network requests in the network access manager. The virtual method QNetworkAccessManager::createRequest() can be subclassed, and the network request can be inspected before it is allowed to be initiated.

In summary, WebKit offers a rich set of features to allow development of secure, Web-enabled applications.

* http://www.openssl.org

5.3. Future Improvements

The Netscape plugin API provides an easy way to embed plugins into a browser component. The API contains some platform dependencies requiring a platform-dependent implementation. Trolltech's current plan is to provide a **QWidget** subclass that abstracts the platform dependencies of the Netscape plugin APIs and allows Netscape plugins to be embedded, not only in a Web browser component, but also in a regular rich client application.

The use of embedded Java® applets inside Web browsers has become less common in recent years. For this reason, integration of Java with the Web browsing component is last on the list of priorities, and is currently planned to be done after the Qt 4.4 release.

To truly interact with Web content, the API would have to allow modification of Web page content via a DOM or DOM-like interface from native code. This would allow many common tasks to be performed, such as building up documents dynamically, reading back values from form fields, and exchanging images in-place. The HTML DOM API has proven itself to be rather cumbersome to use and, at the same time, imposes high memory requirements on applications which use it.

Trolltech plans to create a simplified Qt-style API to make programmatic changes of Web documents a lot easier.

6. Benefits

Using WebKit, the same result will be achieved when showing Web content across all platforms. Both the rendering of Web pages and the functionality of JavaScript will be exactly the same. This removes the overhead required to write Web content that has to work across multiple platforms, and lessens the overhead needed for testing.

With WebKit, the boundaries between desktop and Web environments are blurred. This enables graphic and Web designers to participate more actively in the design and implementation of the user experience, and enables more advanced user interfaces to be made with a shorter time to market. Developers will be able to enjoy a clean separation between user interface and core functionality while utilizing the full range of skills that graphic and Web designers provide.

Using a mixed-mode environment that blends local content with remote HTML design, application content providers will be able to centrally manage application look and feel, and perform micro-customizations without the required overhead of a full re-deployment.

Since HTML is widely known, a Web-enabled user experience enables increased user participation in the process of defining content and appearance. This allows each application to evolve beyond its initial scope, a factor which has traditionally limited the development of deployed software.

Enriching applications and appliances with HTML content is easier than ever before thanks to the tight interoperability between Web and native environments that the Qt WebKit integration provides.

7. Use Cases

Mobile Telephony Home Screen

Figure 2 on the following page shows a possible Qt WebKit Integration use case: on the idle screen of a mobile phone. Upcoming appointments are obtained from the device's contact database rather than from a Web service. Missed calls are coming from the device's telephony module, whereas weather conditions and financial data from the Internet.

The real power of using markup to implement the idle screen is that it can so easily be customized. Mobile operators might provide a library of idle screen pages, or an online site could be designed that allows users to redesign what appears, push a button and instantly download their design to their mobile phone. Customization or personalization of a user's phone has now become a simple reality.

Since the whole capability of the phone is now accessible via scripting, whole new applications can be developed that make use of the camera, Bluetooth®, accelerometers and GPS functions on the device. The range of new applications is now limited by just the imagination of the developer.

Prepaid Phone Card Refill

Normally, when a prepaid subscription to a cellular network has been used up, subscribers must buy a refill card or make a trip to an ATM to purchase new credit.

On a WebKit-enabled phone, a subscriber in this position could be automatically directed to a sandboxed Web environment where they are able to enter credit card details to refill the phone. Once the purchase is accepted, the Web page would access phone resources using exposed objects to refill the phone credits for the user.

User-Configurable "Weblets"

Traditional desktop systems have recently enabled the user to configure information content using multiple Web-enabled viewports called "Weblets".

These viewports present specific information, for instance weather updates, stock information, rapid searches, clocks for multiple time zones, and so on. Using WebKit, such information can be created by content providers and graphic designers – and easily shown in specialized view modes which present the "Weblets" the user has decided to show.

iTunes-Like Branded Experience

The iTunes Music Store is a well known service which is enabled by WebKit. The content is centrally managed and does not require re-deployment to enable new content or functionality. A new version of the client software is required only when new local functionality needs to be exposed. This approach allows rich and appealing content to be easily updated transparently for the user. It also allows Web designers to complement the functionality built into the core application logic with great design for a minimum of effort.

Geo-Location of Service Access

Using a map service such as Google Maps™, WebKit could aggregate local network log files with location data to create a map component that would show place marks for each originating network connection.

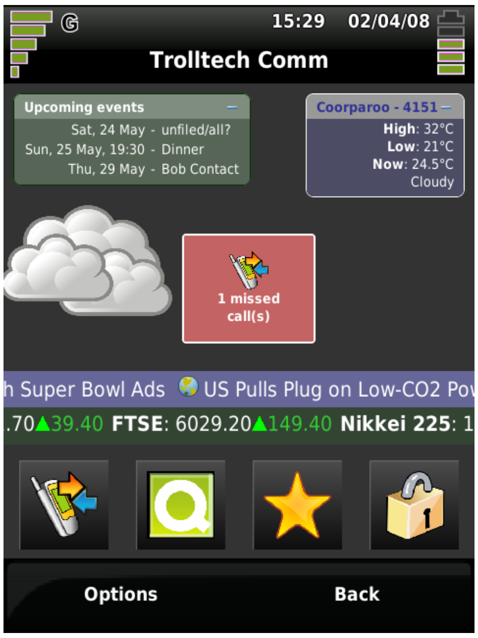
15:29 02/04/08
Trolltech Comm
Upcoming events
Sat, 24 May - unfiled/all?
Sun, 25 May, 19:30 - Dinner
Thu, 29 May - Bob Contact
Coorparoo - 4151
High: 32°C
Low: 21°C
Now: 24.5°C
Cloudy
1 missed call(s)
h Super Bowl Ads US Pulls Plug on Low-CO2 Po
.70▲39.40 FTSE: 6029.20▲149.40 Nikkei 225: 1
Options
Back

Figure 2: An implementation of the idle screen of a mobile phone using the Qt WebKit Integration.

Documentation Viewer

HTML is often the preferred format for end-user and developer documentation supplied with applications and operating systems. Qt also uses HTML to format its documentation and Trolltech has traditionally supplied documentation in HTML format for use with the Qt Assistant application. When configured to use WebKit, Qt Assistant is now able to show any well-formed HTML file.

Social Networking Integration

Many social networking sites expose APIs which allow users to set and query information. Using WebKit, a user's Facebook status could, for instance, be updated with their current country location, interacting with a Facebook application to keep a journal of where in the world they have been.

Central Control of User Interfaces

WebKit can be employed to centrally manage user interface layouts. By exposing native controls in `<object>` tags, different HTML pages can specify different arrangements of controls, and control the application flow by defining signals and slots in JavaScript.

SVG Viewer

WebKit includes support for SVG. With this technology, resolution independence and animation can be achieved with a very small memory footprint.

Reporting

By generating report output as HTML, WebKit can be used to display complex information – additional differentiation can be added by using CSS and traditional HTML formatting to make content stand out.