Code less.
Create more.
Deploy everywhere.

# Qt Extended

This whitepaper describes Qt Extended, a GUI application platform for Linux-based consumer electronic devices.

Qt Extended is comprised of the Qt application framework, an embedded windowing system, a library of integrated applications and technologies, and a complete Software Development Kit for creating applications.

Qt Extended provides an efficient, customizable, and extensible architecture for managing all applications, content, and connectivity for embedded devices.

NOKIA

# Contents

**1. Introduction** **2**

**2. System Requirements** **4**

**3. Overview of Qt Extended Modules** **5**

## Architecture

**4. Application Integration** **6**
- 4.1. Inter-Process Communication . . . 6
- 4.2. Services . . . 6
- 4.3. Data Sharing and Linking . . . 6
- 4.4. The Value Space . . . 7
- 4.5. Plugins . . . 7

**5. Graphical User Interfaces** **8**
- 5.1. Widgets, Layouts and User Input . . . 9
- 5.2. Styles and Themes . . . 10
- 5.3. Internationalization . . . 10
- 5.4. 2D and 3D Graphics . . . 13

**6. Application Life-Cycle Management** **14**
- 6.1. Qt Extended Server . . . 14
- 6.2. Safe Execution Environment (SXE) . . . 15
- 6.3. Package Management . . . 16

**7. Content Management** **17**
- 7.1. Media Management . . . 17
- 7.2. Multimedia Framework . . . 18
- 7.3. Digital Rights Management (DRM) . . . 18
- 7.4. SQL Database Management . . . 19
- 7.5. Personal Information Management (PIM) . . . 19

**8. Connectivity** **20**
- 8.1. VoIP Telephony . . . 21
- 8.2. Messaging . . . 21
- 8.3. Bluetooth . . . 22
- 8.4. Infrared . . . 22
- 8.5. Internet . . . 22

## Development and Deployment

**9. The Qt Extended Development Environment** **24**
- 9.1. The Virtual Framebuffer . . . 25
- 9.2. Qt Designer . . . 26
- 9.3. Qt Linguist . . . 27
- 9.4. Qt Assistant . . . 28
- 9.5. Building and Testing Tools . . . 29

# 1. Introduction

Qt Extended is Qt Software's Embedded Linux® application platform and user interface for consumer electronic devices. Qt Extended is composed of three software layers: Qt for Embedded Linux, a set of platform components, and a comprehensive set of applications which can be included "out of the box" or used as modifiable reference applications. Qt Extended can run wherever Embedded Linux runs. Its modular structure and complete Software Development Kit (SDK) enable agile and efficient software development and system integration while allowing manufacturers to use their own branding throughout the user interface.

Highlights of Qt Extended include:

**Connectivity**

The telephony components support GSM, GPRS, and Voice over Internet Protocol (VoIP). Each of these components is itself modularized to allow system integrators to use their own or a third-party GSM Modem or VoIP stack.

The wireless connectivity components include 802.11, Bluetooth®, infrared (IrDA®), and AT-command modem integration.

The messaging components support sending and receiving SMS, MMS, e-mail and instant messages. Messaging and presence information is exposed to applications via Telepathy services.

**Content Management**

Multimedia components are provided for capture and play of music and videos. A reference integration of the Real Networks® Helix™ DNA client is included that supports several multimedia codecs, including MP3, WAV, 3GP, and AMR-NB; an implementation of the H.263 codec is available separately. A reference integration of the GStreamer media engine is also provided for playback of audio and video.

A reference integration with a Digital Rights Management (DRM) agent is provided, enabling content protection in consumer devices and securing interoperability between the device and Open Mobile Alliance™ (OMA) DRM-enabled services offered by operators.

**User Interface**

The Graphical User Interface (GUI) theme engine gives system integrators complete control of the look and feel and branding used throughout the user interface.

Product internationalization is supported with language translation tools, Unicode™ support, and user-configurable input methods, including handwriting recognition and right-to-left text input and display.

A multi-tasking environment is provided that appears to the user as a natural, single-tasking user experience. In particular, the Qt Extended server ensures that primary tasks, such as incoming call handling, are never degraded by secondary user applications.

**Personal Information Management**

Integrated Qt Extended reference applications include a complete Personal Information Management library (PIM), media player, camera support, image and document managers, and video games. PIM features are integrated throughout the user interface; e.g., contacts can include VoIP number, calls are recorded in Call History, etc.

The complete Software Development Kit includes the following tools:

- Qt Designer for user interface and visual form design.
- Qt Assistant for presenting on-line documentation.
- Qt Linguist for supporting internationalization.
- The Qt Virtual Framebuffer (QVFb) for testing in the desktop environment.
- QBuild, a tool for automating the configuration and build process of systems and applications.
- The QtUiTest framework for System testing.

Additionally, other specialized tools are provided to help with domain-specific problems and issues that developers may encounter:

- A modem simulator (*phonesim*) for testing of telephony applications.
- A value space explorer (*vsexplorer*) for inspecting the operation of applications and services at run-time.

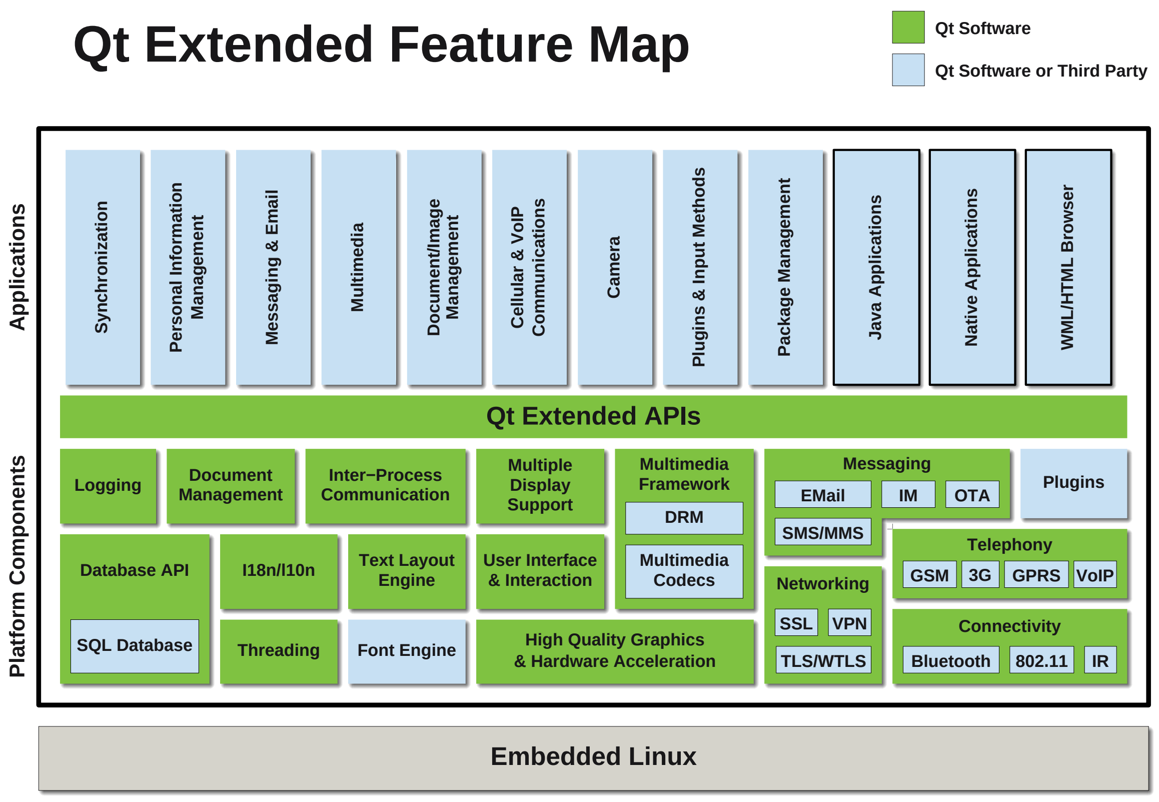


**Figure 1:** Qt Extended Feature Map

## 2. System Requirements

*Qt Extended with the complete Embedded Linux environment and a core selection of applications requires a minimum of 32MB of Flash memory and 32MB of RAM.*

Because Qt Extended is based on Qt, it can run on the most common CPU architectures supported by Linux and a C++ compiler. Supported architectures include Intel® x86 and ARM® (including StrongARM® and XScale™). Qt Extended has been verified on Intel® x86, ARM® 9 and 11 chipsets, including those from Marvell®, Freescale®, NXP™ and NEC®.

Although Qt Extended is targeted at devices with at least 32MB of RAM and 32MB of Flash memory, performance can be improved by adding RAM to devices that offer a rich multi-tasking experience to end-users. In many situations 64MB of RAM is appropriate for devices that run applications with higher memory requirements.

A key strength of Qt Extended is its integrated windowing system: the Qt Extended GUI is directly rendered into the embedded device's hardware framebuffer, saving memory and reducing complexity by eliminating the need for the X11, Xlib, and X Window Server software. Qt Extended is also able to run in a configuration that uses an underlying X11 implementation for rendering, window management, and input methods, enabling the most appropriate display system to be used for each device.

Specialized graphics hardware can be used to further accelerate rendering if suitable drivers are available for the device – Qt Extended provides APIs to enable support for accelerated rendering (see 2D and 3D Graphics on page 13).

More detailed information about the Qt Extended features and requirements can be found on the Qt Extended section of the Qt website:

http://trolltech.com/products

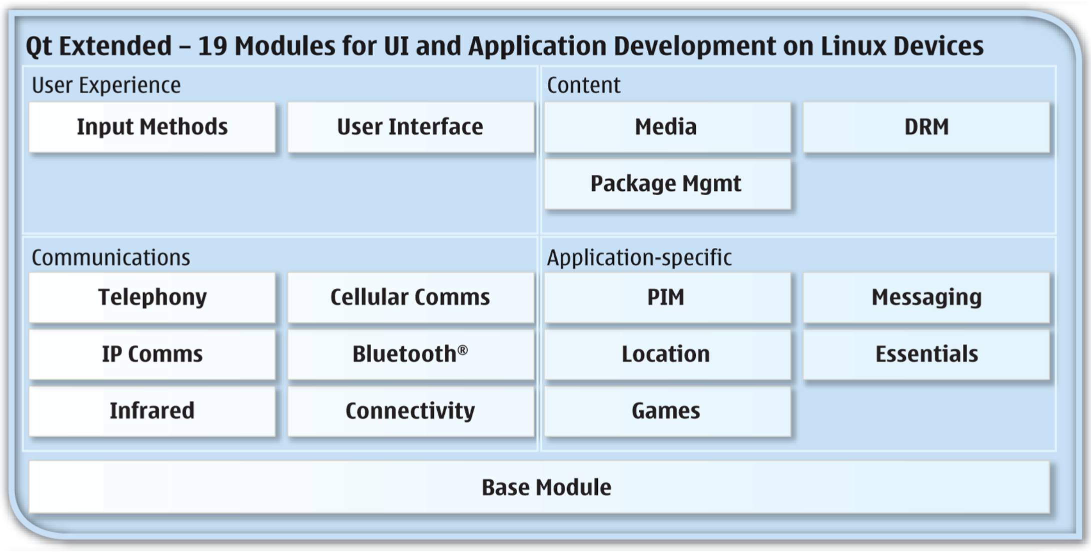
## 3. Overview of Qt Extended Modules

*Qt Extended is divided up into a set of logical modules, covering different areas of functionality, that can be combined to create specific configurations for different devices and deployment scenarios.*

A single base module is needed for the minimal configuration of Qt Extended, and developers can choose from a list of optional modules, selecting those that they need and discarding the rest. A set of premium modules, featuring components provided by third parties, can also be purchased and added to a device's configuration.

Modules are selected when developers create a configuration of the Qt Extended software for a device. Additional applications and services can then be developed against this configuration.

Two reference designs are presented as good starting points with common device profiles: a touchscreen mobile phone and a desktop media phone. These reference configurations for Qt Extended are regularly tested on readily-available hardware which can also be used as reference platforms for development and testing purposes.



In this document, features are grouped together in terms of their appearance in Qt Extended's architecture. This provides a different way of looking at features to complement the grouping shown in the above diagram.

The contents and features of many of the modules are outlined in the following sections. The exception is the **Essentials** module, which contains a set of common user-oriented applications – more information about these can be found in the documentation accompanying Qt Extended and on the Qt website.

# 4. Application Integration

*Qt Extended fulfils a very versitile set of requirements by building on the robust foundations provided by Embedded Linux to provide a flexible client/server architecture for multi-application devices. Its architecture combines rich graphical user interfaces, application and content management mechanisms, and comprehensive connectivity features within a unified and consistent development framework.*

## 4.1. Inter-Process Communication

The Qt Extended architecture is based on the client/server model. Client applications communicate with each other and with the Qt Extended Server (see Qt Extended Server on page 14) using an Inter-Process Communication (IPC) layer called Qt Extended IPC, comprising of *services*, *service requests*, *data sharing and linking*, and the *value space*. New applications can be integrated into Qt Extended using IPC to replace or augment the functionality of existing applications.

The IPC mechanisms provided take full advantage of the "signals and slots" inter-object communication mechanism, a key feature of the underlying Qt architecture, which enables application objects to communicate asynchronously via well-defined interfaces.

## 4.2. Services

The Qt Extended Services architecture allows applications to advertise functionality they can perform for other applications. Services can be requested without knowing which applications or libraries will provide them. This loose coupling of service request and service provider allows system integrators and end-users to select the applications and libraries that the Qt Extended Server will use when brokering service requests.

Software components can become service providers by implementing a standard, public API and naming the service and the specific service requests to which the software component will respond.

## 4.3. Data Sharing and Linking

Qt Extended Data Sharing (QDS) provides an API for efficient sharing of data across applications. QDS is a form of remote procedure call, built on top of the Services mechanism. The QDS API allows a client application to find, and then interact with, another client application that has been registered as the provider of the desired type of data.

For example, an address mapping application for displaying a street map with a particular street address at its center, can search for a QDS service provider that takes a street address as input and returns an image of the street map centered on the desired address.

Qt Extended Data Linking (QDL) is built on top of the Services mechanism, providing an API to associate data items across applications. A QDL Link is conceptually the same as a clickable link on a browser Web page. It contains the information required to uniquely identify and fetch an object from another application. When the user clicks on the QDL Link, the referenced object is accessed.

For example, using QDL, the *New Meeting* function in the Calendar application can create a meeting event for a specific date and time and then create a QDL Link to each person in the Contacts list who should attend the meeting. When the user views the meeting event in the Calendar, a hypertext link appears for each meeting attendee. Activating one of these links sends a message to the Contacts application to display the information for that contact.

## 4.4. The Value Space

The Value Space allows transient data to be efficiently shared between applications. This encourages the development and use of components in Qt Extended without the communication costs traditionally associated with component-based architectures.

It is a hierarchical database of readable, writable, transient data values. A Qt Extended application can create data values in the value space which can then be read, written, and subscribed to by other applications. The values remain in the value space until their creating application and all using applications are terminated.

When an application subscribes to a value in the value space, the application is notified via the Qt Extended "signals and slots" inter-object communication mechanism whenever any application changes the value.

The *vsexplorer* tool is provided with the Qt Extended SDK to allow the Value Space to be inspected. Many components write values into the value space so that they are accessible to other libraries, applications, and themes.

## 4.5. Plugins

The plugin framework provides a well-defined development environment for adding new features to Qt Extended. The range of services provided by the server need not be completely defined at system configure and build time. Services can be added to Qt Extended as Plugins, even after the system has been deployed on devices.

Qt Extended provides APIs for writing extensions to Qt Extended itself, such as custom database drivers, image formats, text codecs and custom styles, and APIs for extending existing Qt Extended applications. Standard plugin interfaces are provided for writing plugins for features at various levels of the Qt Extended architecture. Plugins for user-visible features, such as content recognition, DRM handling, GUI styles, input methods, media codecs and media engines, are complemented by plugins for lower level parts of the Qt Extended infrastructure, such as image and icon format handlers, modem drivers, network interfaces, telephony multiplexers, printing, and keyboard and screen drivers.

## 5. Graphical User Interfaces

*Qt Extended provides excellent support for 2D and 3D graphics, with support for many common raster and vector graphics formats. A variety of GUI elements (widgets) are available, including a fully-featured canvas for highly-interactive applications, and new widgets can be created with ease. A powerful style engine enables the appearance of widgets to be substantially changed, or simply fine-tuned as required.*

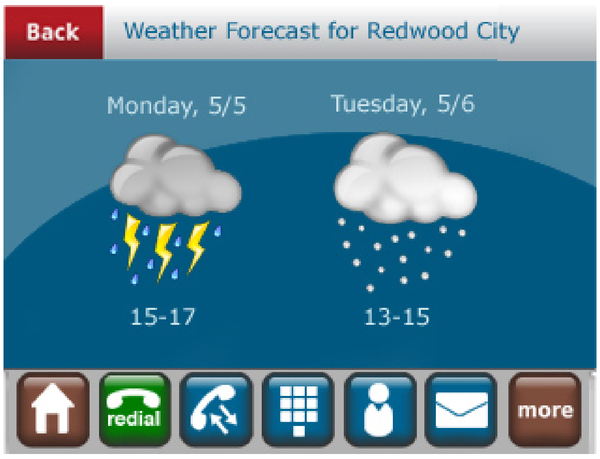
Graphics are drawn using a common 2D painting API for widgets, images (including PNG, JPEG, TIFF and BMP), vector graphics (SVG and PDF), printers (PostScript® and PDF), and OpenGL® contexts. This system provides the foundations for the features highlighted below.

Qt Extended provides extensive theming capabilities, allowing the customization of screens such as the home screen, application launcher, and dialer by adapting simple XML descriptions.

The underlying painting system is common to the Qt Extended range of products and all desktop versions of Qt, and is described in detail in the Qt Whitepaper:

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

Facilities for creating interactive 2D applications are provided by the Graphics View framework (see 2D and 3D Graphics on page 13). Qt Extended also provides support for a comprehensive range of common graphics file formats.



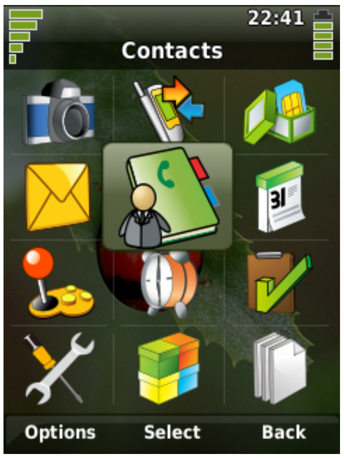


**Figure 2:** Qt Extended themed for deskphone and mobile device screens.

## 5.1. Widgets, Layouts and User Input

GUIs in Qt Extended consist of collections of widgets, such as buttons, label and text editors, arranged within general purpose container widgets.

### Widget Management

Widgets are arranged using well-proven layout managers which remove the need to position user interface elements exactly. Since layout managers automatically position widgets according to their size requirements, Qt Extended applications do not need to be rebuilt to support different written languages (see Internationalization on the next page) or changes to the user interface font.

This flexible approach to rendering makes it possible for the look and feel of a device to be fundamentally changed, while support for resolution independent drawing enables the device environment and applications to adapt to different resolutions and screen sizes. As a result, a series of devices with different form factors can all be made to present a consistent user interface without the need for time-consuming redesigns.

The position, visibility and stacking order of windows are managed by the Qt Extended Server (see Qt Extended Server on page 14). Windows can be opaque, partially transparent, or hidden completely, making it possible for fading transition effects to be used.

### Input Methods

Qt Extended accepts user input in a variety of different forms; most visibly through "input methods" designed to help users enter information into a device with limited input interfaces – perhaps only a keypad or a touch screen. Input methods, such as on-screen keyboards, handwriting recognition systems, or predictive text schemes can be integrated into the environment, allowing existing applications to take advantage of new solutions.

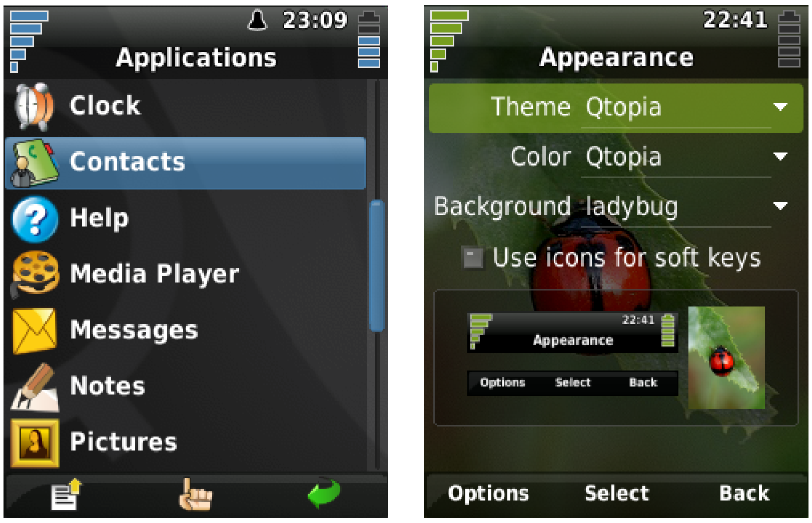
Qt Extended provides standard features to address the issue of text input, given the lack of both screen real-estate and convenient keyboards on mobile devices. The Phone Keys input method is intended for devices with the common 0-9, # and * buttons. The Full Screen Handwriting input method is intended primarily for touch screen devices, but can be used with any sort of pointing device, including a mouse. These input methods are not mutually exclusive, and a device equipped with both phone keys and a pointing device can take advantage of both.

### Keyboards and Predictive Text

Additionally, Qt Extended offers virtual keyboards for devices with touch screens: a standard QWERTY virtual keyboard and a predictive keyboard. These input methods offer a widget displaying the characters that can be touched or clicked to input text.

The predictive keyboard is designed for finger-based input and takes into account the reduced accuracy of this method by analyzing the user input to resolve ambiguities. This keyboard provides dictionary look-up capabilities, allowing it to be used accurately with fingers instead of a stylus.

When a pointing device such as a mouse or touch screen is available, the user can enter text by drawing a character at a time. Qt Extended is capable of composing characters from multiple strokes, for example in the case of "t" or "4". A shadow of the original stroke is left on-screen to help guide the next stroke. Qt Extended uses input hints to guide the input guess, as in the case of the letter "s" and the number "5", where the strokes may be difficult to differentiate.



Theme Qtopia
Color Qtopia
Background ladybug
Use icons for soft keys

**Figure 3:** Example Qt Extended "Applications" and "Appearance" screens.

## 5.2. Styles and Themes

A style implements the look and feel of the user interface, preserving the visual identity of a brand by providing basic drawing functions for drawing elements such as frames and buttons.

Styles are complemented by the user's preferences for colors, fonts, sounds and other customizable properties. As a result, Qt Extended applications automatically adapt to the active theme used on a device. All the built-in widgets are style-aware, and developers can use an extensive style API to ensure that custom widgets fit into the user's preferred theme.

The style engine supports right-to-left languages, and can automatically adapt the user interface to use right-to-left widget layouts as the default for these languages.

Qt Extended has a powerful theming engine, allowing much of the user interface to be customized and personalized using simple, yet rich, XML descriptions. Components of the user interface that are managed by the theme engine include the Title Bar, Home Screen (shown in figure 3), Call Screen, and the Context Bar.

Themes are defined by XML files describing a hierarchical group of visual and formatting elements. Themed Views contain theme elements which define the layout and structure of components. Each of these elements can be assigned attributes to customize visual appearance and placement.

## 5.3. Internationalization

The Qt Extended internationalization support enables complete localization of the user interface and all applications. It covers the following requirements:

- Translation of strings (including localized time and date strings).
- Support for localized dictionaries (including the use of multiple dictionaries at the same time).

- Support for language-specific common dictionaries.
- Adjustments to alignments and layouts where necessary (including right-to-left input support; e.g., Arabic and Hebrew).
- Localization of pictures.
- Localization of user help pages.

In some cases, internationalization might be simple. For example, making a US application accessible to British users might require nothing more than a few spelling corrections. However, to make a US application accessible to Japanese users or a Korean application accessible to German users, requires that the application display texts in different languages and use different input methods, character encodings, and presentation conventions.

All input widgets and text drawing methods offer built-in support for internationalization and localization. The built-in font engine can simultaneously render texts from different writing systems; e.g., English, Arabic and Hebrew, or English and Chinese.

The Qt Extended internationalization and localization support can be used for the following languages:

- All East Asian languages: Chinese, Japanese and Korean
- All Cyrillic languages: Russian, Ukrainian, etc.
- All Western languages using Latin scripts
- All Indic languages
- Thai, Lao, Khmer
- Right-to-left languages: Arabic and Hebrew
- All scripts in Unicode 4.0 that do not require special processing

Qt Extended is capable of handling writing systems that require special line breaking behavior. For example, some Asian languages are written without spaces between words, and line breaking can occur either after any character (Chinese, Japanese, Korean), or after logical word boundaries (Thai).

Qt Extended supports so-called bi-directional writing systems (Arabic, Hebrew), where the language is written from right to left except for numbers and embedded English strings, which are written left to right. The exact behavior is defined in Unicode Standard Annex #9.

Typographical features, such as the use of non-spacing or diacritical marks (accents or umlauts in European languages) are correctly handled by Qt Extended. This is important for languages that make extensive use of these marks, such as Vietnamese, in which multiple marks are used to clarify pronunciation.

Qt Extended also supports use of ligatures. In certain contexts, special pairs of characters can be replaced by a combined glyph forming a ligature. Common examples are the “fl” and “fi” ligatures used in typesetting US and European books.

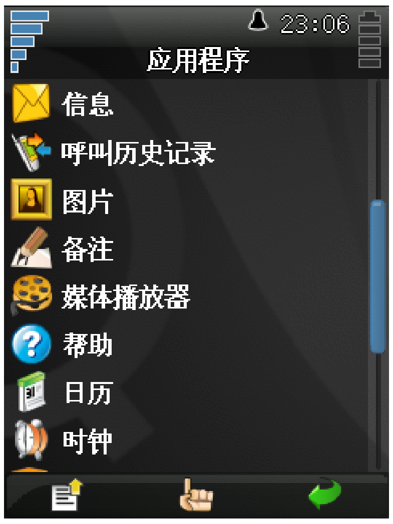


**Figure 4:** Internationalization in Qt Extended

Applications developed for Qt Extended are written using a translation scheme that allows translatable text to be maintained separately from source code. Translators can use Qt Linguist (see The Qt Extended Development Environment on page 24) to work on multiple translations while an application is developed.

Translations are stored in small binary files and components to provide a default configuration for a particular group of users or, since applications do not need to be recompiled to use a different translation, they can be supplied separately and loaded by applications when necessary.

Additionally, widget placement can be reversed to provide a more natural appearance for users who work with right-to-left writing systems.

Translation packages for the Qt Extended GUI and applications are available on request.

## 5.4. 2D and 3D Graphics

Scalable Vector Graphics (SVG) is an XML-based file format and language for describing graphical elements that is commonly associated with Web-based graphics. SVG support in Qt Extended is based on the SVG 1.1 standard, a World Wide Web Consortium (W3C®) Recommendation, and provides additional features to support the Tiny profiles of SVG 1.1 and 1.2.

Since Qt Extended uses a consistent painting API to display and animate all kinds of graphics, SVG drawings can be rasterized, rendered using OpenGL, and used in place of bitmap images.

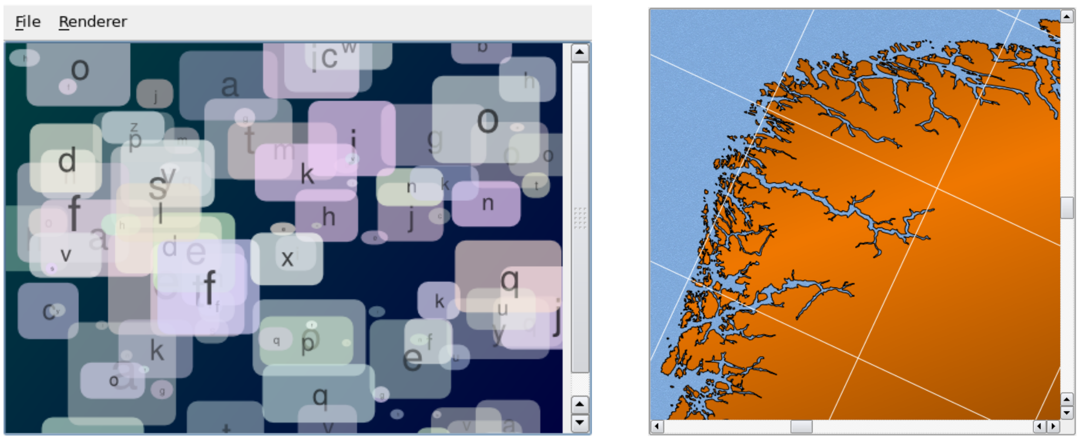


**Figure 5:** Qt Extended provides support for SVG files, OpenGL integration facilities, and common image file formats.

Qt Extended applications can take advantage of the Graphics View framework for interactive graphical applications. Graphics View provides a canvas that is used to manage and display large numbers of items in a two-dimensional scene. Views onto the scene can zoom, pan and rotate the visible part of the scene.

OpenGL is a standard API for rendering 3D graphics. Developers use OpenGL components to provide a context within the user interface in which to call OpenGL functions provided by their system's OpenGL implementation. Integration with OpenGL in Qt Extended is designed to be familiar to developers who are already familiar with OpenGL programming.

Qt Extended can specifically target OpenGL ES – a subset of OpenGL for use on embedded devices – on devices that are shipped with compliant rendering libraries. Typically, since manufacturers include OpenGL ES to demonstrate support for hardware acceleration, Qt Extended applications can take advantage of improved graphics performance on such devices.

Developers can also take advantage of a specialized paint engine to automatically accelerate 2D graphics with OpenGL. Applications can use this feature to improve 2D rendering performance on appropriate hardware or overlay controls and decorations onto 3D scenes.

# 6. Application Life-Cycle Management

*The Qt Extended architecture enables vendors to supply a wide range of applications and services. These can be pre-installed as part of a device package that is initially supplied to end-users, or they can supplied as standalone packages for later installation. The Qt Extended comprehensive package management and Safe Execution Environment makes it convenient and safe for both native code and Java® applications to be deployed on devices.*

End-user applications in the device environment are launched and managed by the Qt Extended Server. Applications use the server to request services and to communicate with other applications running on the device (see Inter-Process Communication on page 6).

The Qt Extended Safe Execution Environment (SXE) provides a platform-level security solution which allows end-users to run native code applications in a sandboxed environment on the device. SXE allows the device owner to be confident that running applications will not result in significant risk of compromising the operation of the embedded device, or its data.

Access to third-party content is carefully controlled by a package management system that can be configured to allow end-users to safely deploy applications and install media content on devices. The package manager works with the SXE and content management system (see Content Management on page 17) to provide an integrated approach to security of the device platform.

Java Virtual Machines for Qt Extended can be obtained from Nokia partners. Java applications integrate seamlessly with the device environment, providing Java content with a native Qt Extended look and feel.

## 6.1. Qt Extended Server

The Qt Extended Server is responsible for managing the execution of applications and services in the environment. It performs functions ranging from controlling access to low-level systems, such as devices or operating system services, to launching applications and orchestrating inter-process communication (see Inter-Process Communication on page 6).

### Structure and Operation

The Qt Extended Server is structured as a collection of largely independent server tasks. Each server task is responsible for providing a small, well-defined portion of functionality, which is often the backend work for some other system functionality. For example, the network management APIs ultimately communicate with the Network Server task running in the background. The set of server tasks can be thought of as the set of building blocks from which the Qt Extended Server is built by system integrators.

In addition to task management, the server manages a collection of independent daemon processes. Some applications, such as the VoIP agent and the media server, are run as daemons. These applications provide services or functionality to the system but do not have a user interface. Other applications, such as the synchronization agent (see Personal Information Management (PIM) on page 19), do have a user interface, but also provide an important service to the system that requires them to continue running in the background, even when the UI is not shown.

#### Life-Cycle Management

The Qt Extended Server is also responsible for managing the complete life-cycle of each native Qt Extended application, from launch to shutdown and even after shutdown. Efficient application life-cycle management is extremely important in embedded devices, where RAM and swap space are both limited. The server provides for both quick launch and lazy shutdown of applications to minimize application startup times.

The server's quick launcher performs the time consuming startup tasks of C++ symbol relocation and application construction ahead of time, while the user is selecting an application or entering data. At system startup time, the Qt Extended server launches the quick launcher stub application. While the user is busy selecting an application to run, the quick launcher stub process performs all the common application initialization work. By the time the user has selected an application, the quick launcher process is ready and waiting in the background. It loads the user's application into the quick launcher address space and runs it immediately, rather than running the application from scratch as a new, separate process. Once used, the quick launcher process is started anew to wait for the next user application.

After the user terminates an application, rather than shutting it down completely, the system leaves it running in a hidden state, called the lazy shutdown state. When the user restarts that same application later, the still running but hidden process only has to perform the comparatively quick operation of showing its UI to be usable again. The lazy shutdown almost eliminates startup time for frequently used applications.

#### Integration with Other Components

Since the Qt Extended environment is document-centric rather than file-centric, the Qt Extended server uses services provided by the content management system (see Content Management on page 17) to match document content types to the applications that are able to handle them, taking into account DRM restrictions on content use, and in turn ensuring that executable content and applications are run within SXE.

The server can be extended to provide additional services through the use of plugins. These enable systems integrators to extend the default set of plugins to support specific hardware features of a device, or to roll out extensions to a pre-installed environment once a device is in use.

## 6.2. Safe Execution Environment (SXE)

There is a growing demand among end-users for devices that allow the use of third-party applications and packages. This demand is driven in part by the availability of games and other applications specially written for reduced functionality Java environments available for mobile hardware. Such reliance on a particular programming language and environment for security features is a limitation that SXE is designed to overcome.

SXE provides a platform-level security solution based on the use of policy-based countermeasures that ensure that applications only access appropriate resources. SXE uses features provided by Linux to restrict access to operating system resources and devices while also limiting access to features provided by the Qt Extended Server.

This approach extends beyond single-language solutions for sandboxing applications, enabling applications compiled to high-performance native code to be deployed safely on users' devices. This makes it possible for the user to download and deploy third-party applications while making the deployment process more predictable and less risky for vendors and systems integrators.

The platform-level security approach used by SXE can be contrasted with two other commonly used security approaches: perimeter and trigger-based solutions. While easier to implement, these alternatives have well-understood disadvantages:

- Perimeter solutions, such as firewalls or systems relying on trust, do not provide continuous security. Once a malicious program is through the perimeter gateway it is free to perform any action.
- Signature or trigger-based solutions, such as anti-virus or change detection systems, can only react to signs of malicious activity rather than prevent it.

Platform security can be combined with these perimeter and trigger-based security mechanisms to provide multiple layers of defence. However, platform security is a desirable minimum requirement for a system which allows untrusted downloaded software to be run. Additionally, the use of a sandbox for untrusted applications allows the security policy to be tailored to the file-systems and application stack, providing stronger and more robust security at the operating system level.

SXE is also able to co-exist with certificate-based approaches, enhancing security by providing a complementary set of features.

## 6.3. Package Management

Qt Extended provides a comprehensive, network-aware package management system that works with SXE to present a consistent view of the packages available for the end-user's device.

The package manager obtains information about the currently available packages by using the Hypertext Transfer Protocol (HTTP) to fetch a list of packages from a remote server. Only packages which are suitable for the device are presented to the end-user.

Installation of packages is performed in a multi-stage process that takes into account the SXE policies and DRM configuration – see Digital Rights Management (DRM) on page 18 – to ensure that only authorized packages are installed and that applications are set up to run with the correct privileges. Untrusted third-party applications are controlled by SXE, which means that only specific files and device resources may be accessed.

The package manager keeps track of the resources provided in each package, making uninstallation of unused or unwanted packages a straightforward process.

As well as simplifying the vendor's task of deploying updates and extras to end-users, SXE and the package management system enable third party developers to create software for devices and form development communities around them.

# 7. Content Management

*Qt Extended provides a comprehensive set of features to make managing data a consistent and intuitive experience for developers and users. Documents are catalogued and managed using a SQL database-driven management system that enables data to be located efficiently, and Personal Information Management features provide a solid foundation on which to base end-user productivity applications. A plugin-based multimedia framework allows different multimedia engines and codecs to be supplied.*

The content management system focuses on ease of use for developers, a lightweight storage architecture, and extensibility.

Support for Personal Information Management (PIM) is a central part of the content management infrastructure (see Personal Information Management (PIM) on page 19). A rich set of classes encapsulate aspects of personal and productivity features, such as addresses, contacts, events and tasks.

Developers do not need to worry about implementing support for the SXE or DRM in order to access data. Qt Extended provides a convenient API that applications use to locate, read, modify and write data – the data management system handles interaction with the SXE and DRM systems

Data may be held in internal storage, or on removable media. Cooperation between the Qt Extended Server and the content management system ensures that removable media automatically become available when they are connected to a device.

As embedded hardware becomes more powerful, end-users increasingly demand access to multimedia content on mobile devices. Qt Extended provides a modular multimedia framework (see Multimedia Framework on the next page) that works within the document management system to deliver both unrestricted and DRM content.

## 7.1. Media Management

Qt Extended provides a unified document management system to facilitate access to user-visible documents such as images and music files.

Qt Extended contains a document management application that efficiently handles large numbers of documents. A media management system is also provided, which includes media-specific features, such as support for playlists, MP3 ID3 tags, as well as allowing documents to be grouped and complex queries to be used to access data.

The document system maintains content information in an SQL database (see page 19) to ensure that information about the available documents can be accessed efficiently, and enables queries to be made with minimal file system interaction.

Files and content subject to DRM controls are transparently browsed, launched, moved, deleted and examined for their properties. The document system provides powerful mechanisms to allow content to be searched for by file type, category and location. File operations such as copy, move and rename are also handled and can follow OMA DRM standards for DRM controlled files.

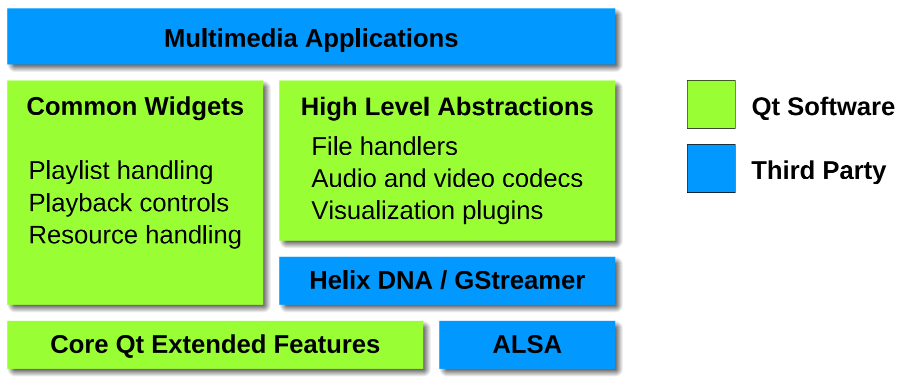


**Figure 6:** Qt Extended Multimedia Framework

## 7.2. Multimedia Framework

Qt Extended includes a multimedia framework for applications that play music and videos. Features include simple, abstracted multimedia playback APIs and reusable control widgets. At the lowest level, the framework interacts with system libraries for audio and video display, including the Advanced Linux Sound Architecture (ALSA) and Video For Linux (V4L) libraries. Graphics are implemented by the underlying Qt libraries. Abstractions are provided to allow the integration of media engines that provide for audio and video codecs. At the highest level, the framework provides a common API for playing media, which includes widgets for managing media resources, visual controls to play videos, and provide feedback on audio and video playback.

A reference integration of the RealNetworks Helix DNA Client library is provided that supports most consumer-level multimedia codecs, including MP3, WAV, 3GP, AMR-NB, and H.263. Convenient APIs provide simple playback interfaces that avoid the requirement to call into the Helix DNA media engine directly.

Because the Qt Extended multimedia framework is modular, other 3rd party implementations can be used as plugins to replace or augment the reference integration. A plugin is provided that uses the GStreamer media engine to provide audio and video playback.

## 7.3. Digital Rights Management (DRM)

DRM protected content in Qt Extended is presented alongside regular content and is browsed, launched, moved, and deleted in the same manner. When presented in a document selector or launcher view, protected content can be identified by a superimposed key icon, which may be barred indicating the content has no valid rights associated with it.

Qt Extended applications access DRM content through the use of plugins to the document system. A complete DRM agent integration is typically composed of three parts, a plugin to the content system, a license and settings management application, and a Qt Extended service. The plugin interface for the DRM agent allows all three components to be delivered in a single plugin library. Qt Extended includes a reference integration of a DRM agent, which implements the OMA DRM version 1.0 and version 2.0 standards.

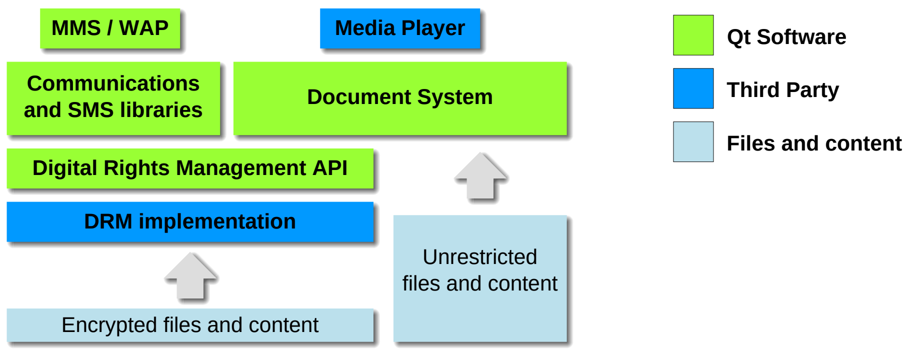

MMS / WAP
Media Player
Communications and SMS libraries
Document System
Digital Rights Management API
DRM implementation
Encrypted files and content
Unrestricted files and content
Qt Software
Third Party
Files and content

**Figure 7:** Qt Extended DRM Architecture

## 7.4. SQL Database Management

Underlying the document management system is a SQL database that is used to store meta-data about files on media supplied with the device. By default, Qt Extended is supplied with a reference implementation that uses the lightweight SQLite database, but this can be replaced by any SQL database for which there is a Qt database driver available.

Content stored in the database can be accessed using standard SQL queries, but Qt Extended also provides high-level APIs to simplify and streamline the process of accessing and managing content.

## 7.5. Personal Information Management (PIM)

Qt Extended provides a library of classes to encapsulate most common types of personal information; a set of applications use these classes to bring “enterprise grade” PIM features to mobile devices.

The PIM Library provides a unified API for accessing personal information, making it possible for many applications to access data concurrently, and allowing high level integration between PIM applications; for example, a contact application can share information about birthdays with a calendar application so that those days are specially marked.

Underlying this API is a lightweight SQL database. Information about events, tasks and contacts is stored and retrieved using the API, but it is possible to access this data directly via SQL queries if required.

The library provides facilities to allow applications to search for specific information from the database. Results can be filtered and sorted based on the relevant criteria for each data type.

Synchronization support is an integral part of the library’s design. The library handles communication with the Sync Agent, a tool for managing the synchronization process between Qt Extended devices and standard desktop productivity applications.

Simple controls for common features are also provided with the aim of simplifying the development of PIM applications. Reference implementations of the PIM applications are provided.

# 8. Connectivity

*Qt Extended supports all the common hardware types and network protocols used in mobile devices, including many that are specific to cellular telephony. In addition to standard cellular telephony, Qt Extended provides support for integrated VoIP telephony. Qt Extended devices can communicate using non-cellular bearers, including Bluetooth, 802.11, and infrared connections.*

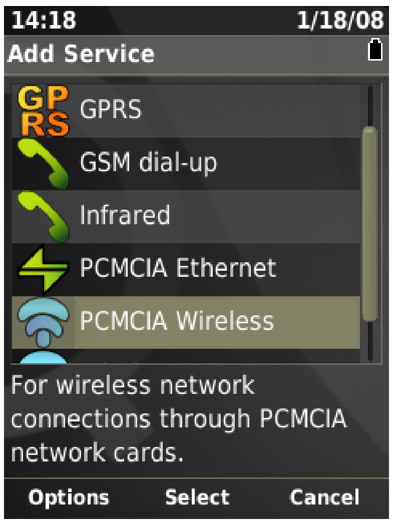
The Qt Extended Telephony Library provides an abstraction layer between specific telephony implementations, presenting a unified API for developers to use that takes care of implementation-specific details, such as AT commands for GSM telephony, thus making it easier for application developers to present a consistent user interface for end-users.

Qt Extended includes support for both standard GSM/3G telephony with GPRS features and VoIP telephony.

Qt Extended supports conventional cellular SMS and MMS messaging in addition to e-mail.

Qt Extended also enables wireless networked communication between WLAN devices, supporting IEEE 802.11 wireless networking, (including encryption that is provided using WPA2 and WPA-EAP), and Bluetooth. Access to the Internet can be enabled by using plugins to provide Wi-Fi or GPRS connectivity. Support for communication over USB connections is also provided.

A comprehensive set of networking classes are exposed to applications. As a result, standard Internet applications, such as Web browsers, e-mail clients and Instant Messaging applications, can be deployed on devices. The extent to which telephony and network communication can be performed is limited only by the restrictions imposed by running inside SXE (see page 15).



For wireless network connections through PCMCIA network cards.

**Figure 8:** Adding a new connection service to a device.

## 8.1. VoIP Telephony

VoIP is integrated into Qt Extended at the Telephony Library level. The intuitive user interface integration, together with a generic VoIP framework, facilitate the use of 3rd party VoIP stacks. The VoIP user interface functions the same way as traditional GSM telephony, that is, a VoIP call can be made by simply referencing a VoIP identifier in the same way as a phone number, and a contact can have a corresponding VoIP identifier. As with GSM calls, the corresponding contact is shown when making and receiving a VoIP call, and VoIP calls are recorded in the Call History maintained by the Telephony Library.

Qt Extended VoIP features include:

- Complete integration with the dialer user interface and call management features.
- VoIP ID is included in the contacts database.
- VoIP ID is accessible from the Contacts application.
- Presence of contacts is detected and shown in the Contacts application.
- VoIP functionality and status information are available to applications.
- VoIP is included or excluded at system build time.
- Call management functionality is provided for VoIP calls.
- Session Initiation Protocol (SIP) is supported.

The reference VoIP implementation provided with Qt Extended is written to support the Session Initiation Protocol (SIP). It supports the Real-Time Transport Protocol (RTP) for voice media traffic, provides basic call management functionality for initiating outgoing calls and terminating incoming calls, and it provides simple support for two presence states: available and unavailable.

## 8.2. Messaging

Qt Extended supports many different kinds of message types and messaging protocols over cellular and Internet communications channels. Access to messages services is handled by low level classes specific to each type of message, while classes in the Messaging Library provide a unified API for message composition and archival.

The Messaging Library exposes a "Collective API" that provides integration Telepathy-based services. This currently uses Gabble, the Jabber/XMPP connection manager for Telepathy, but may be extended to support other Telepathy services such as Sophia SIP in future releases.

The basic components of messages for each of the most common types of messages are well supported by Qt Extended. Classes are provided to facilitate MIME type handling, message encoding and decoding, time stamp management and image handling. A comprehensive set of text codecs ensure that the contents of messages are preserved correctly.

A reference implementation for a messaging application is provided with Qt Extended. The Messages application provides facilities for reading, composing and sending e-mail, SMS and MMS messages. It also handles attachments and provides search and querying mechanisms for the message archive. Multiple messaging accounts can be created and configured via the user interface.

## 8.3. Bluetooth

Bluetooth connectivity is implemented as a series of layers above BlueZ, the standard Linux Bluetooth stack. Qt Extended manages devices using the D-Bus system bus, exposing low-level functionality via a collection of wrapper classes. These can be used by applications to perform operations such as device pairing, changing the visibility of a device, or simply switching on and off the Bluetooth hardware. These classes can additionally be used to notify other components of changes to the status of Bluetooth communications via Qt's "signals and slots" mechanism (see Inter-Process Communication on page 6).

Higher level communication is performed using Bluetooth and OBEX classes. OBEX classes are based on the open source OpenOBEX library's implementation of the OBEX protocol. Qt Extended provides convenience classes to perform common tasks based on specialized OBEX services, such as Object Push Profile (OPP) (used to send and receive business cards and appointments). Where applicable, these classes closely resemble other networking classes in Qt Extended, providing signals and slots to facilitate the production of modular, reusable software.

In addition to the OPP OBEX service, Qt Extended also supports the Service Discovery Protocol (SDAP), Basic Brinting Profile (BPP), Dial Up Networking (DUN), File Transfer Profile (FTP), Headset Profile (HSP), Handsfree Profile (HFP) and File Browsing services. Additional profiles can be implemented by subclassing standard client and server classes which take care of most of the OBEX-specific implementation details.

## 8.4. Infrared

Infrared connectivity is provided by a set of classes that are analogous to the classes provided for Bluetooth communication and device handling. Although many of the services and operations traditionally performed using infrared communication, such as synchronization, are now available via Bluetooth, infrared communication still serves a need for point-to-point communication.

Classes are used to represent local and remote devices, provide access to the device's Information Access Service (IAS) database, and monitor the availability of remote devices. Low level classes for server and socket-level operations are supplied, enabling device-specific infrared features and services to be written.

## 8.5. Internet

Qt Extended provides an Internet Settings application allowing the user to configure network connections for the device. It allows the user to perform the following functions:

- Start and stop existing network interfaces.
- Add, edit and delete network configurations.

- Add, edit and delete WAP/MMS accounts.
- Access network specific functionality (e.g., WLAN scanner).
- Add, edit and delete VPN connections.
- Configure the use of encryption for wireless connections (WPA/WPA2/WEP).

The user can have several Internet connections open at the same time. A use case may be a user who is browsing on the Internet via GPRS and enters a public WLAN zone. The Internet application allows the user to change the routing table in order to make maximum use of the available data connections, by choosing between the fastest connection or cheapest medium, for example.

Qt Extended includes a session manager for Internet connections. This is used to ensure that a network interface is shutdown when there is no need for it. If an application starts a network interface the interface will remain active for the lifetime of the application. This means that network interfaces are stopped if an application crashes or quits. If two applications were to request the same interface the interface will stay active until the last application closes.

# 9. The Qt Extended Development Environment

*Qt Extended includes a software development kit that simplifies the process of developing applications and components. A suite of tools enables most aspects of the development process to be performed from within a desktop environment. A "phone simulator" allows the device modem to be simulated on the developer's workstation.*

Qt Extended is supplied with a set of tools to enable software to be developed independently of the hardware used for devices. The consistency of the underlying APIs used for GUI development on desktop and embedded systems makes it possible for developers to design and prototype applications for devices on workstations using a suite of tools.

- **Qt Virtual Framebuffer (QVFb)** (see The Virtual Framebuffer on the next page) is used to simulate the use of mobile and embedded applications on a workstation.
- **Qt Designer** (see Qt Designer on page 26) is a visual tool for building graphical user interfaces interactively. It enables designers to produce prototypes of working applications without any programming knowledge; these designs can later be turned into code by a set of supporting tools that are integrated with the Qt Extended build system.
- **Qt Linguist** (see Qt Linguist on page 27) facilitates the translation of user-visible text into languages suitable for different regions and groups of users. Supporting tools streamline the process of making applications translation-ready.
- **Qt Assistant** (see Qt Assistant on page 28) is a documentation viewer that developers can use to view the API documentation and high level overviews for the release of Qt Extended that they are working with.
- **QBuild** and **QtUiTest** (see Building and Testing Tools on page 29) are tools to help with configuration, building and testing of software for devices.

Since Qt Extended is built on the solid foundations provided by Embedded Linux, the task of developing for Qt Extended devices is simpler and easier than with other embedded operating systems, requiring only simulation of a device's behavior rather than a complete system emulation.

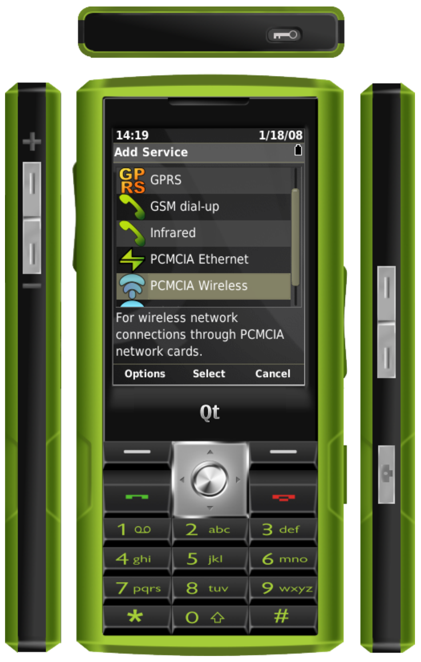


**Figure 9:** An example QVFb session using the Greenphone skin supplied with Qt Extended, showing the buttons on each side of the device.

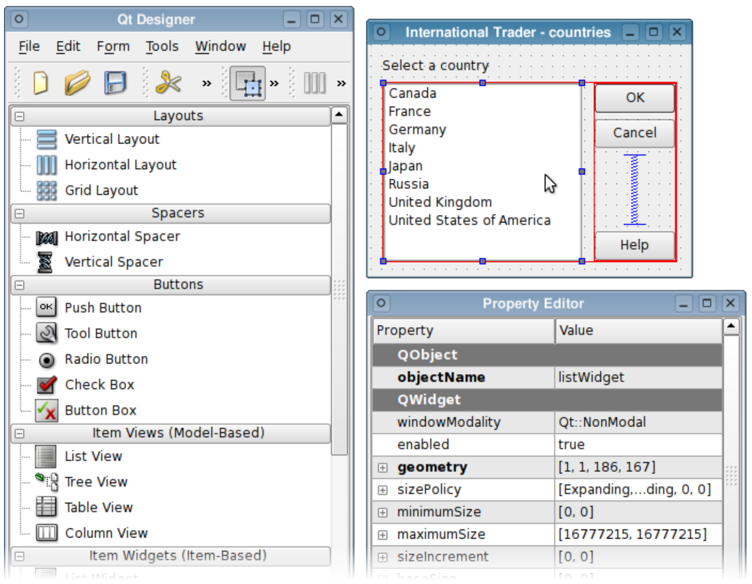
## 9.1. The Virtual Framebuffer

The Qt Extended environment can be simulated on a desktop display using QVFb, an application that provides a pixel-perfect representation of a device's screen. The virtual framebuffer emulates the hardware framebuffer using a shared memory region in order to display the framebuffer in a desktop window.

The tool also supports a feature known as the "custom skin", which can be used to change the look and feel of the display and the simulated device. A skin is a set of text and image files that tell the virtual framebuffer what it should look like and how it should behave. The skin changes the desktop display into an accurate representation that looks and behaves like the target device. This approach enables the teams responsible for application development and usability to work without actual hardware.

Note that a skin can have buttons, which, when clicked, send signals to the Qt Extended application running inside the virtual framebuffer, just as would happen on the target device.

The QVFb skins provided include generic clamshell, touchscreen and smartphone designs in addition to a specific skin for the Greenphone.



| Property | Value |
|---|---|
| QObject | |
| **objectName** | listWidget |
| QWidget | |
| windowModality | Qt::NonModal |
| enabled | true |
| **geometry** | [1, 1, 186, 167] |
| sizePolicy | [Expanding,...ding, 0, 0] |
| minimumSize | [0, 0] |
| maximumSize | [16777215, 16777215] |
| sizeIncrement | [0, 0] |

**Figure 10:** Editing a user interface component in *Qt Designer*.

## 9.2. Qt Designer

Qt Designer is the Qt tool for designing and building graphical user interfaces (GUIs). With Qt Designer, developers can quickly design and build user interface components using the same widgets that will be used in the application. Components created with Qt Designer automatically communicate using the underlying "signals and slots" inter-object communication mechanism. All components can be previewed to ensure they have the intended look and feel.

Designing a form with *Qt Designer* is a simple process. Developers drag widgets from a toolbox onto a form, and use standard editing tools to select, cut, paste, and resize them. Each widget's properties can then be changed using the property editor. The precise positions and sizes of the widgets do not matter. Developers select widgets and apply layouts to them (see Widgets, Layouts and User Input on page 9). This approach makes design very fast, and the finished forms will scale properly to fit the screen size, font size, and language texts on the target device.

*Qt Designer* eliminates the time-consuming "compile, link, and run" cycle for user interface design. This makes it easy to correct or change designs. *Qt Designer*'s preview options let developers see their forms in other styles, decoupling even further the style of an application from its design.

Several form templates are supplied, and developers can create their own templates to ensure consistency across an application or family of applications. Programmers can create their own custom widgets that can easily be integrated with *Qt Designer*.

Form designs are stored in XML User Interface Compiler (UIC) files, and converted into C++ header and source files by the UIC – this is automatically handled by the build tools. Alternatively, applications can load the contents of UIC files at run-time and dynamically construct their user interfaces, making it possible for the look and feel of applications to be configured by system integrators and third parties.

## 9.3. Qt Linguist

Qt Linguist enables translation of a user interface into multiple languages. Most of the text that must be translated for an application's user interface consists of either single words or short phrases. These typically appear as window titles, menu items, pop-up help text, and labels on buttons, check boxes and radio buttons.

Translators and programmers must address a number of subtleties and complexities of human language:

- A single phrase may require translation into several different forms depending on context; e.g., *open* in English might become *öffnen* or *aufbauen* in German for *open file* or *open Internet connection* respectively.
- Phrases that contain variables need to be reworded in a language where the word order and therefore the placement of the variables may be different.

The Qt Extended internationalization system (see Internationalization on page 10) and translation tools provide clear and simple solutions to these and other issues.

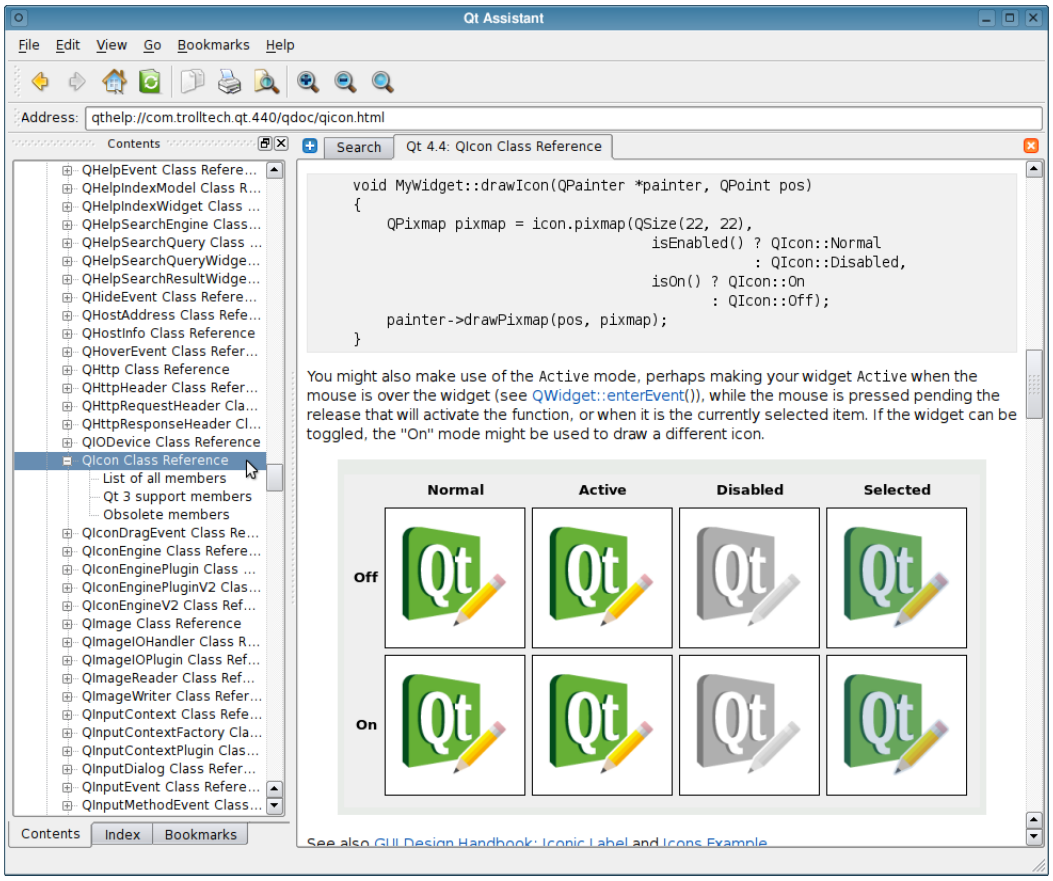
### Workflow

The translation tools are designed to be used in repeated cycles as applications change and evolve, preserving existing translations and making it easy to identify which new translations are required. A typical workflow proceeds in the following way:

1. The programmers enter words and phrases into the application source code in their native language using a simple syntax that identifies phrases requiring translation.
2. The release manager generates a set of translation files that are produced automatically from the source files and passes these to the translator.
3. The translator opens the translation files using Qt Linguist, enters the translations for the identified words and phrases, and saves the results back into the translation files, which are then returned to the release manager.
4. The release manager is able to use the tools provided with Qt Extended to generate compact versions of these translation files ready for use by the running application.

Qt Linguist also provides a phrase book facility to help ensure consistent translations across multiple applications and projects.

The format used by Qt Linguist is a straightforward XML file that can be imported into other professional translation tools. Qt Linguist is also able to import and export industry standard XLIFF format files.


Qt Assistant
File Edit View Go Bookmarks Help
Address: qthelp://com.trolltech.qt.440/qdoc/qicon.html
Contents
Search
Qt 4.4: QIcon Class Reference
QHelpEvent Class Refere...
QHelpIndexModel Class R...
QHelpIndexWidget Class ...
QHelpSearchEngine Class...
QHelpSearchQuery Class ...
QHelpSearchQueryWidge...
QHelpSearchResultWidge...
QHideEvent Class Refere...
QHostAddress Class Refe...
QHostInfo Class Reference
QHoverEvent Class Refer...
QHttp Class Reference
QHttpHeader Class Refer...
QHttpRequestHeader Cla...
QHttpResponseHeader Cl...
QIODevice Class Reference
QIcon Class Reference
List of all members
Qt 3 support members
Obsolete members
QIconDragEvent Class Re...
QIconEngine Class Refere...
QIconEnginePlugin Class ...
QIconEnginePluginV2 Clas...
QIconEngineV2 Class Ref...
QImage Class Reference
QImageIOHandler Class R...
QImageIOPlugin Class Ref...
QImageReader Class Ref...
QImageWriter Class Refer...
QInputContext Class Refe...
QInputContextFactory Cla...
QInputContextPlugin Clas...
QInputDialog Class Refer...
QInputEvent Class Refere...
QInputMethodEvent Class...
Contents Index Bookmarks
Normal
Active
Disabled
Selected
Off
On

```
void MyWidget::drawIcon(QPainter *painter, QPoint pos)
{
    QPixmap pixmap = icon.pixmap(QSize(22, 22),
                                   isEnabled() ? QIcon::Normal
                                               : QIcon::Disabled,
                                   isOn() ? QIcon::On
                                          : QIcon::Off);
    painter->drawPixmap(pos, pixmap);
}
```

You might also make use of the Active mode, perhaps making your widget Active when the mouse is over the widget (see QWidget::enterEvent()), while the mouse is pressed pending the release that will activate the function, or when it is the currently selected item. If the widget can be toggled, the "On" mode might be used to draw a different icon.

See also GUI Design Handbook: Iconic Label and Icons Example

**Figure 11:** *Qt Assistant* displaying a page from the Qt 4 documentation.

## 9.4. Qt Assistant

Qt Assistant is the Qt tool for browsing and searching online documentation. The documentation for Qt Extended is written directly in the C++ source files by Qt developers. The Qt documentation team continually revises the inline documentation to ensure that it is up-to-date, accurate, and readable for each release. The documentation team also writes the larger texts, such as "how to" documents and complete, working examples, as well as high-level class descriptions that introduce the concepts and designs used to implement classes.

Extensive cross-referencing in the Qt Extended documentation makes navigation easy with Qt Assistant. Even snippets of example code can contain clickable links. For example, class declarations in code examples are typically marked up as clickable links to the appropriate class documentation.

Documentation written for Qt Assistant uses a simple syntax, which developers can use to document their own Qt Extended applications. Qt Assistant can then be deployed as the help browser for the embedded system.

## 9.5. Building and Testing Tools

Qt Extended is supplied with a build system that aims to make it easier to configure, build and test software, and to deploy it on devices. The *QBuild* tool is used for Qt Extended itself, but it is designed to be used for the whole development process, from configuration of the device platform to third party application development.

*QBuild* is able to automate the following tasks:

- Monitoring and update of translation files used with Qt Linguist.
- Handling of cross-compilers and custom compilers, such as lexers and parser generators.
- Configuration of services and plugins.
- Creation of binary images and packages ready for installation on devices.
- Creation of binary SDKs for deployment to third party developers.

Applications can be tested using *QtUiTest*, a framework for automated system testing of GUI applications that is designed to be extended by developers and testers. *QtUiTest* tests are typically written in Qt Script, an ECMAScript-based scripting language, but the framework itself is language-neutral.

Other key features include support for script recording, abstraction of input devices, testing on real and virtual embedded devices, and a focus on performance testing.

More information about *QtUiTest* is included in the *QtUiTest Whitepaper*, available from the relevant section of the Qt website.

## About Qt Software:

Qt Software (formerly Trolltech) is a global leader in cross-platform application frameworks. Nokia acquired Trolltech in June 2008, renamed it to Qt Software as a group within Nokia. Qt allows open source and commercial customers to code less, create more and deploy everywhere. Qt enables developers to build innovative services and applications once and then extend the innovation across all major desktop, mobile and other embedded platforms without rewriting the code. Qt is also used by multiple leading consumer electronics vendors to create advanced user interfaces for Linux devices. Qt underpins Nokia strategy to develop a wide range of products and experiences that people can fall in love with.

## 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.

Oslo, Norway +47 21 60 48 00 ▪ Redwood City, CA, USA +1 650-551-1676 ▪ Beijing, China +86 10 6588 8836 ▪ qt-info@nokia.com www.trolltech.com