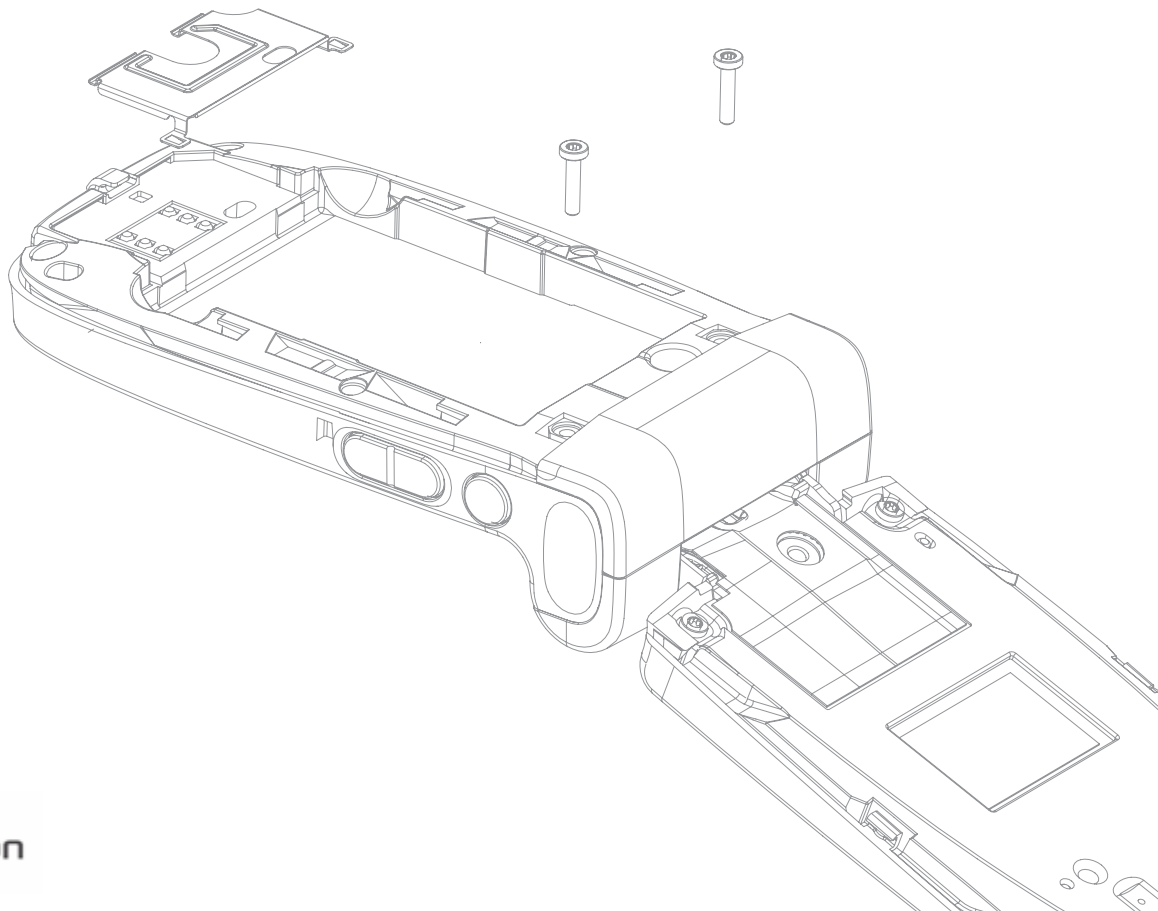
Developers guidelines

DEVELOPER WORLD THE FAST TRACK FROM MIND TO MARKET

November 2006

UIQ 3 C++

for Sony Ericsson UIQ 3 phones



Sony Ericsson

Preface

Purpose of this document

This document describes the **Symbian™ 9.1 / UIQ™ 3 C++** support for the Sony Ericsson P990, M600, W950, and W958 series.

Readers who will benefit from this document include support engineers and software developers.

It is assumed that the reader is familiar with the C++ programming language.

These Developers guidelines are published by:

Sony Ericsson Mobile Communications AB,
SE-221 88 Lund, Sweden

Phone: +46 46 19 40 00
Fax: +46 46 19 41 00
www.sonyericsson.com/

Third edition (November 2006)
Publication number: EN/LZT 108 8155 R3A

Sony Ericsson Developer World

On www.sonyericsson.com/developer, developers will find documentation and tools such as phone White Papers, Developers Guidelines for different technologies, SDKs and relevant APIs. The website also contains discussion forums monitored by the Sony Ericsson Developer Support team, an extensive Knowledge Base, Tips & Tricks, example code and news.

Sony Ericsson also offers technical support services to professional developers. For more information about these professional services, visit the Sony Ericsson Developer World website.

UIQ Developer Program

Developers who register with the new UIQ Developer Program at www.uiq.com/developer will get access to the UIQ 3 forum, getting started tutorial and downloading of the UIQ 3 beta SDK.

UIQ Technology provides support for platform and SDK questions, whereas support for Sony Ericsson UIQ 3 phone specific extensions are to be directed to Sony Ericsson.

Sony Ericsson will continue to provide developer support for previous versions of UIQ (UIQ 2.0 and 2.1 for P800, P900 and P910 series) as long as these products are available on the market.

Document conventions

Products

Sony Ericsson UIQ 3 phones are referred to in this document using generic names as follows:

Generic names Series	**Sony Ericsson mobile phones**
P990	P990i, P990c
M600	M600i, M600c
W950	W950i, W950c
W958	W958c

Abbreviations

API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
CSS	Cascading Style Sheet
DRM	Digital Rights Management
HAL	Hardware Abstraction Layer
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
MMS	Multimedia Message Service
OMA	Open Mobile Alliance
OTA	Over The Air
PDA	Personal Digital Assistant
PIM	Personal Information Manager
RTSP	RealTime Streaming Protocol
SDK	Software Development Kit
SMS	Short Message Service
USB	Universal Serial Bus
WAP	Wireless Application Protocol
XML	eXtensible Markup Language

Typographical conventions

Code is written in Courier font, for example: `TInt CCamera::CamerasAvailable()`

Trademarks and acknowledgements

Document history

Change history		
2005-10-11	Version R1A	Preliminary version published on Developer World.
2006-02-13	Version R2A	Second preliminary version published on Developer World. **Note:** Information in this document is preliminary and in some parts incomplete, and may be changed without any notice until the phones in scope of this document are released to the market.
2006-11-20	Version R3A	Third edition. Updated with new layout and references to W958 series.

Contents

Technical overview

This document is valid for the Sony Ericsson P990, M600, W950, and W958 series of UIQ 3 phones.

The Symbian 9.1/UIQ 3 UIQ 3 phones are versatile application platforms enabling application developers to create applications in a variety of programming languages, including native C++ and Java™. This document provides guidelines for developing C++ applications.

The reader of this document is expected to understand the basics of Symbian OS™, UIQ 3 and the development environment of UIQ 3 SDK. For further reading, please refer to http://www.uiq.com/developer.

Phone features

The Sony Ericsson P990, M600, W950, and W958 mobile phones are based on the Symbian 9.1 / UIQ 3 user interface. UIQ 3 is a pen- and softkey-based user interface platform.

The table below lists general features of the phones.

For more information about the technical features of the P990, M600, W950, and W958, please refer to the product White Papers, available at Sony Ericsson Developer World.

Feature	Support
Colour screen	**P990 Flip open/M600/W950/W958**: portrait mode, full screen 240x320 px portrait mode application area 240x220 px landscape mode, full screen 320x240 px **Note:** Applications for flip open, landscape orientation should always be designed for full screen. P990 Flip closed: portrait mode, full screen 240x256 px portrait mode application area 240x156 px landscape mode, full screen 256x240 **Note:** Applications for flip closed, landscape orientation should always be designed for full screen. 262,144 colours
Screen font	Sans-Serif, Latin
Internal User storage	**P990, M600**: 80 MB **W950, W958**: 4 GB
Additional storage: **P990:** Memory Stick Duo™ **P990:** Memory Stick Duo Pro™ **M600**: Memory Stick Micro™ (M2™)	 supports up to 128 MB external storage. supports up to 8 GB external storage. supports up to 1 GB external storage.

Third party application support	Symbian 9.1 UIQ 3 C++ Java™ ME platform, compliant to the following specifications: Connected Limited Device Configuration, CLDC 1.1, JSR-139 Joint Technology for Wireless Industry R1 (JTWI 1.0), JSR-185 Mobile Media API, MMAPI 1.1 ,JSR-135 Wireless Messaging API, WMA 1.1, JSR-120 Wireless Messaging API, WMA 2.0, JSR-205 Java APIs for Bluetooth, JSR-82 Mobile 3D Graphics API 1.0, JSR-184 PDA Optional Packages, JSR-75, (File and PIM APIs) Web Services API, JSR-172 Nokia UI API 1.1 Connected Device Configuration (CDC) 1.0 Foundation Profile 1.0 Personal Profile 1.0 JDBC for J2ME, JSR-169 PDA Optional Packages, JSR-75
Music player	MP3 ,WAV, AU, AMR, MIDI (G-MIDI with 40 voices polyphony and SP-midi), RMF, iMelody, XMF, DLS, RealAudio®, AAC, AAC+ and eAAC+. W950 and W958 also support WMA
Video recorder (**P990** only)	3GPP and MPEG4 (VGA, QVGA, QQVGA and 24-bit color depth) with AMR-NB or AAC-LC encoded audio (2 quality levels + video without audio).
Video player	3GPP and MPEG4 (VGA, QVGA, QQVGA and 24-bit color depth) with AMR, AAC-LC, AAC+ or enhanced AAC+ encoded audio. RealVideo® decoding
Streaming	RTSP according to 3GPP, RealAudio, RealVideo and FastTrack (PacketVideo proprietary)
Still images	JPEG/EXIF, BMP, WBMP, GIF, PNG

Programming environment

The P990, M600, W950, and W958 are based on Symbian OS 9.1.
Refer to http://www.symbian.com/technology/product_v9.html

The user interface (UI) is based on UIQ 3.
Refer to http://www.uiq.com/documentation

Backwards compatibility

The Symbian OS 9.1 / UIQ 3 environment and its implementation in P990, M600, W950, and W958 series represent a concept with major differences to earlier Symbian/UIQ implementations in Sony Ericsson UIQ 3 phones.

Porting applications

The “Programmers Guide to New Features in UIQ 3”, which can be found in the UIQ 3 SDK documentation, contains extensive tips and code examples for developers wishing to port applications written for earlier versions of the Symbian/UIQ environment.

API overview

The tables below list subsystems and APIs which in one way or other deviates from the standard Symbian/UIQ implementation. Any API or subsystem which are not in the list are fully supported according to specifications by Symbian/UIQ.

For a complete API reference, see the Symbian/UIQ documentation included with the UIQ 3 SDK and documentation found at http://www.symbian.com/technology/product_v9.html and http://www.uiq.com/documentation. **Note:** In the UIQ 3 SDK documentation, API details can be found primarily in the “Symbian OS Guide” and “Symbian OS Reference” sub-sections of the “Symbian OS SDK v9.1” chapter.

Symbian OS subsystems and APIs

Subsystem /API	Support	Comments/references
Base and system libraries		
Base HAL	Partly supported. Includes backlight support.	For more information, see “Base HAL API issues” on page 23
Multimedia		
Multimedia ASR	Not supported	
Multimedia DevASR	Not supported	
Multimedia DevVideo	**Not** recommended, because DevVideo requires MultimediaDD capability in PlatSec. The MMF API should be preferred for multimedia applications.	For more information, see “Audio policies for priorities” on page 36 and “Usage tips for the multimedia APIs” on page 37

Subsystem /API	Support	Comments/references
Multimedia DevSound	**Not** recommended. The MMF API should be preferred for multimedia applications.	For more information, see “Audio policies for priorities” on page 36 and “Usage tips for the multimedia APIs” on page 37
Telephony		
ETEL Core API	Fax and Ownership not supported	For more information, see “Telephony API issues” on page 25
Help		
Help application engine	Not supported	
System agent		
System agent (mail notification service, etc.)	Partly supported. Sysagt has been deprecated in Symbian 9.1 and replaced with a “Publish and Subscribe” (P&S) component. Some properties have changed.	See E32property.h and UIQ 3 SDK for details.
HTTP client		
HTTP framework	User-Agent/x-wap-profile extension implemented.	For more information, see “HTTP framework extension” on page 25

UIQ 3 specific API

API	Support	Comments/references
QimFramework	Not supported	

Other supported APIs

Sony Ericsson specific APIs

API	Description	Comments/references
BT Keyboard	Support for keyboard vendors.	See “Bluetooth keyboard APIs” on page 29

API	Description	Comments/references
Vibration API		See “Vibration API” on page 38
MMS API		See “Sony Ericsson MMS API” on page 39

Open GL-ES API

API	Description	Comments/references
OpenGL-ES extensions		See “OpenGL-ES implementation” on page 44

Application development

This chapter contains general information for the developer of C++ applications for the Sony Ericsson P990, M600, W950, and W958 series of mobile phones.

General development tips

When developing UIQ 3 C++ applications for Sony Ericsson UIQ 3 phones, the following should be considered:

- Always favor CQikXXX classes over CEikXXX classes. Derive your AppUi from CQikAppUi (Qikon) rather than CEikAppUi
- Ensure that your application is "theme aware". Use logical colors to draw application controls and propagate color scheme change messages through your application.
- Make sure that application icons in three different sizes are defined for display on the device.
- Prefer the Symbian exception handling before the standard C++ language exception handling.
- Do not block a `ViewActivatedL` - any potentially long running action chained off a view activation should be done async. Otherwise the system may kill your application for being too sluggish.
- Be conservative in stack usage, including the implementation of recursive algorithms.
- Follow the UIQ 3 style guidelines.
- Have some way to call `Exit` from your application in debug builds. In the emulator, this will cause a panic if there is any memory leakage. It's much better to find memory leaks during development in the emulator, than at the end of development, when testing in the phone is done.
- Do not run timers continuously in the background because it seriously affects battery life.
- Make your code device independent when possible. Do not "hard code" screen sizes, colour depths, font sizes, and so on, instead use Symbian OS APIs to get details of device characteristics and capabilities so that your application will run on the widest possible ranges of devices including any future Symbian/UIQ based Sony Ericsson phones.
- Avoid using local `TBuf` objects. They consume valuable stack space, which is extremely limited on a symbian device, and behave very differently compared to in the emulator. If you need more than a few hundred bytes allocated to a `TBuf`, then use a `HBuf` which is allocated on the Heap instead. Always remember to destroy the `HBuf` after it is finished with. The reasons for stack exhaustion are almost impossible to trace on the real target, they appear as random, unexplained crashes.
- Since applications can be installed either in phone memory or on Memory Stick, avoid using absolute paths for file locations in your code.

The UIQ 3 SDK

A beta version of the UIQ 3 SDK for Symbian OS v9.1 is available as a free download from the UIQ Developer Program Web site, http://www.uiq.com/developer. The SDK is designed for use with Metrowerks™ CodeWarrior™ Development Studio for Symbian OS, and also includes support for Microsoft® Visual Studio® 2003.

Note: An IDE, for example the Metrowerks CodeWarrior Development Studio for Symbian OS, is required if an application needs to be tested in the UIQ 3 SDK emulator. Otherwise it will only be possible to test the application on the target device itself.

Sony Ericsson SDK extension packages

The Sony Ericsson extension Packages will be available at Sony Ericsson Developer World by February 2006. The extension packages include Sony Ericsson specific APIs as well as emulator components such as phone specific fonts and skins.

By April 2006 there will also be extension packages available for development on the Chinese version of the phone.

Sony Ericsson C++ API extensions for the UIQ 3 SDK enable developers to create UIQ 3 applications to access:

- Vibration functions
- Bluetooth keyboard functions
- MMS API functions

Building and installing applications

Generation of project files

A symbian project consists of a number of different files such as bld.inf, mmp, rss, cpp, h and pkg-files. These files are all needed for a complete application. When starting a new project it is possible to create these files manually, for example by copying the “Hello World” example application, shipped with the SDK, and then editing the copied files.

A more convenient way is to use a kind of skeleton generator tool, which can generate the project files automatically.

Building for the device from the command line

It is possible to build a project for the phone using only the command line and the GCC-E compiler without using any IDE.

If there are multiple SDK:s installed on the PC, the correct SDK must be configured using the devices command.

To select the UIQ 3 SDK as the active SDK, the following command is used:

```
$ devices -setdefault @UIQ_30:com.symbian.UIQ
```

To get more information about the devices command:

```
$ devices -help
```

When the active SDK has been set, the project can be built for the handset.This requires that project files such as bld.inf, mmp- and pkg-file are already created, see above.

The build is done using the following commands:

```
$ bldmake bldfiles

$ abld build gcce urel

$ makesis project.pkg
```

This will create an unsigned SIS-file.

If the application use any Platform Security (platsec) capabilities it must also be signed in order to install it on the handset. For this, a developer certificate must be obtained from the Symbian signed web-site: https://www.symbiansigned.com/. For more information, see the section “Signing digital applications” below.

To sign the sis-file, the following command is used:

```
$ signsis -s project.SIS outfile.sis cert.cer key.key password
```

This will create a developer certificate signed SIS-file that may be installed on the device.

Deploying applications

The steps described below are documented in the UIQ 3 SDK. It is important that the developer is familiar with the “MakeSIS” and “SignSIS”, used for preparing applications to execute on the mobile phones.

Each Symbian OS UI variant has a shell program, an Application Launcher, that allows the user to run application programs. For an application to be run from the Application Launcher, the following is recommended:

1. An application file (**.exe**) minimally containing a unique identifier (UID) to identify the program.

2. *Application registration information*. The application icon and caption are specified in a special resource file, the *localizable icon/caption definition file*, or in the application's *UI resource file*. Application properties and other information are defined in another type of resource file, called a *registration file*.
   For more information, refer to the UIQ 3 SDK documentation, *Symbian OS SDK v9.1* / *Symbian OS guide* / *Tools and Utilities* / *Application resource tools guide.*
   Note: In earlier Symbian versions .aif files where used for specification of the application icon, caption and other properties. This method is not supported in Symbian OS 9.

Other files may also be required.

- Create a .pkg file by copying a sample file from the Install directory <SDK_DIR> and modify it to match your application.
- Run the MakeSIS or SignSIS tool from the command line.

"MakeSIS" or "SignSIS" is needed for generating .sis format installation files to ensure availability of a C++ application to end users for installation.

Note: In Symbian OS 9.1, only the **.exe** extension is supported for executable files. The **.app** extension is no longer supported.

.sis packages

The P990, M600, W950, and W958 series handsets are open devices for adding applications. Today, there are Symbian devices on the market based on different user Interfaces (UI), namely UIQ, Series 60, Series 80 and Series 90. Applications for these devices are all installed using an installation package.

The format is developed by Symbian and is called "Symbian Installation System" (SIS).

To ensure that the appropriate **.sis** file is installed on a designated target device, **.sis** packages for the Sony Ericsson UIQ 3 phones need to contain platform-specific package information. If this information is not included, the installation will stop and the user will receive the error "Incompatible installer version" when trying to install the .sis package.

Information that needs to be added to the .sis package files:

Note: X, Y, Z in the ID fields below indicates that the actual values have not been decided when this document is published.

For general UIQ 3.0 compatibility:
(0x101F6300),3, 0, 0, {"UIQ30ProductID"}

For platform identification:
(only needed if the application is platform dependant, required for Symbian Signed applications)
For Sony Ericsson phones, based on Symbian OS 9.1 / UIQ 3 platform:
(0x10274BE7), 1, 0, 0, {"SEMCSEZPlatformID"}

For product (phone model) identification:
(only needed if the application is product dependant)
P990i: *(0x10274BB1), 1, 0, 0, {"SonyEricssonP990PlatformProductID"}.*
P990i Alias*: (0x10274BE8), 1, 0, 0, {"SonyEricssonP990ProductID}.*
P990c: *(0x10274BEE), 1, 0, 0, {"SonyEricssonP990CProductID"}*
M600i: *(0x10274BEA), 1, 0, 0, {"SonyEricsson10274BEAProductID"}*

M600c: *(0x10274BEC), 1, 0, 0, {"SonyEricsson10274BECProductID"}*
W950i: *(0xXXXXXXXX), X, Y, Z, {"SonyEricssonXXXXXXXXProductID"}*
W950c: *(0xXXXXXXXX), X, Y, Z, {"SonyEricssonXXXXXXXXProductID"}*
W958c: *(0xXXXXXXXX), X, Y, Z, {"SonyEricssonXXXXXXXXProductID"}*

For language compatibility:
(optional)
P990i: *(0x10274BE9), 1, 0, 0, {"SonyEricssonP990LanguageID"}.*
P990c: *(0x10274BEF), 1, 0, 0, {"SonyEricssonP990CLanguageID"}*
M600i: *(0x10274BEB), 1, 0, 0, {"SonyEricsson10274BEBLanguageID"}*
M600c: *(0x10274BED), 1, 0, 0, {"SonyEricsson10274BEDLanguageID"}*
W950i: *(0xXXXXXXXX), X, Y, Z, {"SonyEricssonXXXXXXXXLanguageID"}*
W950c: *(0xXXXXXXXX), X, Y, Z, {"SonyEricssonXXXXXXXXLanguageID"}*
W958c: *(0xXXXXXXXX), X, Y, Z, {"SonyEricssonXXXXXXXXLanguageID"}*

Note: When creating a .sis file that is not to be signed, the VID (Vendor ID) must **not** be specified in the mmp file. A .sis file containing a VID without a matching signature will fail to be installed.

Installation and data storage

Applications can be easily downloaded directly to the mobile phone using the browser, or may be installed from a connected PC, using USB, Infrared or Bluetooth. Applications can also be installed from a Memory Stick.

Multimedia content such as images, movie clips, sound clips and general-purpose files such as Word documents can either be stored locally (C: drive) or on a Memory Stick (D: drive). Third party (Java and C++) applications can also use the Memory Stick for both application and data storage. These are all accessed using the viewers provided with the mobile phones.

The built in browser can download .sis files from WML and xHTML pages provided that the server supports the actual file types. P990, M600, W950, and W958 can also download .sis files from ordinary Web servers provided that they support the MIME type for each of the file types.

On-target debugging (ODD)

On-device debugging is addressed through CodeWarrior 3.1, Carbide Developer and Carbide Professional Editions which provision the necessary client for deployment on the phone. The client is signed with sufficent permissions to allow flexible debugging without requiring any specialised re-signing prior to deployment. Once installed, debugging can be invoked in the usual visual fashion through the IDE.

For more information, please refer to http://www.forum.nokia.com/carbide and http://www.forum.nokia.com/main/0,6566,1_74,00.html

Signing digital applications

Symbian Signed

"Symbian Signed" is a program from Symbian that supports **tested and digitally signed applications** on Symbian OS phones. It is supported by all Symbian licensees including Sony Ericsson.

A "signed" application means that the application has passed the test criteria and is digitally signed to **guarantee the source and integrity of the application**. Sony Ericsson encourages all users to use signed applications only.

Developers are allowed to use a Symbian logo called "For Symbian OS" to indicate compliance to the program.

In order to install an application that is "Symbian Signed", a Symbian root certificate must be present in the phone. This certificate is pre-installed in the Sony Ericsson UIQ 3 phones.

For detailed information about the "Symbian Signed" program, please refer to https://www.symbiansigned.com

Unsigned-sandboxed applications

A number of unrestricted APIs may be used for unsigned applications, running in a "sandboxed" environment. Unsigned applications can be deployed with blanket or one-shot permission settings. During installation the user is prompted to select one of these permissions for the application.

Applications using only unrestricted APIs may be tested and assigned a Symbian Signed certificate. This will allow installation with blanket permission, without user interaction during the installation. The signing process requires that the developer owns a valid ACS Publisher ID from Verisign.

Symbian OS v9 security enhancements

Symbian OS v9 introduces enhancements of the security architecture for mobile devices and applications. In short this implies that only trusted applications will be granted access to certain groups of sensitive APIs (capabilities).

An application using restricted APIs may only be Symbian Signed when the application is released (finished). This would make it impossible to test the unfinished application on actual handsets. To take care of that, special Developer Certificates (DevCerts) can be installed on the handsets where the application is to be tested during development. DevCerts are issued by Symbian Signed via a Web portal, https://www.symbiansigned.com/app/page/devcertgeneral.

Testing in an emulator does not require DevCerts, but platform security for the emulator can be turned on and off.

For details on developer certificates and Symbian Signed, please refer to https://www.symbiansigned.com or the DevCert Web portal, https://www.symbiansigned.com/app/page/devcertgeneral.

Also refer to Sony Ericsson Developer World for information about Symbian signing and Channel certification: http://developer.sonyericsson.com/site/global/devservices/certification/symbiancer/dev_certs/p_devcerts.jsp

Programming issues

This chapter contains some programming issues of interest for developers of UIQ 3 C++ applications for the Sony Ericsson P990, M600, W950, and W958 series.

General issues

Power consumption considerations

In Sony Ericsson Mobile devices based on Symbian OS only small parts of the power management framework are implemented. The reason for this is that the ASICs used are very much developed with power management in mind and include mechanisms to implement fast and efficient power management in low level code (for example kernel extensions and device drivers). This pretty much removes the need for the greater part of the framework.

Design guidelines

To get an optimized power management the following should always be kept in mind when developing applications for multitasking environments:

- Timeouts should never be used for control of HW power down.
- Timers should always be preferred before delay loops in the code. The most accurate timers on user side have 1ms granularity and this should be short enough to remove the need for any delay loops.
- Polling should be avoided. If absolutely necessary, a timer callback that initiates the poll should be used. This allows CPU power save during the timer interval.
- Optimal hardware power save is normally done via the hardware device driver.
- Memory handling should be considered. Using large buffers when transferring data and accessing flash drives using large blocks results in more idle time for the CPU and memory HW to be put in low power mode.
- Code efficiency from a CPU load perspective should be considered. The less time code is executing the more time the CPU will be put in low power mode.
- "Focus lost" notification when implementing an app should be respected. When focus is lost there should not be any activity left running, for example, all timers have to be cancelled to avoid stealing CPU usage in the background with less low power mode as a result.

Memory usage considerations

RAM cost in NAND devices

Sony Ericsson UIQ 3 phones are NAND devices, which means that DLL and EXE binary code must be shadowed in RAM before execution. Therefore, the total RAM cost of a running application depends not only on the heap usage, but also on the size of the binary code.

Data files (for example .txt and .ini files) can be accessed as normal files from the file system. The smallest block size that the system reads or writes is 512 bytes, which is therefore the smallest suggested size of data files.

Stack and heap issues

According to the UIQ 3 SDK documentation, processes are by default assigned a stack size of 8K. However, it has been found that some applications run out of stack memory and crash, even if stack objects were handled correctly by the application.

These issues can in many cases be solved by increasing the stack size for the application with the `epocstacksize` directive in the project's mmp file. For example, to increase the stack size to 20K:

```
// set stack to 20K
epocstacksize 0x5000
```

Some general advice to solve stack and heap issues:

- Prefer heap objects before stack objects, by using `HBufC` instead of `TBuf`.
- Spread the stack objects in separated methods. This helps keeping local stack objects smaller.
- Try increasing the stack size with the `epocstacksize` directive in the mmp file.
- Allocate more heap with the `epocheapsize` directive in the mmp file.

Base HAL API issues

The following properties do not fully comply with the SymbianOS standard:

```
EDisplayState
EBacklightState
EDisplayBrightness
EKeyboardBacklightState
```

Unlike the generic code, setting these properties does not directly control the display.

Note: The implementation is not fully decided by the release of this document, but will be specified in later editions.

EDisplayState

`Get`: gets current state of display, 0 if display is off, 1 if display is on

`Set`: makes request to the LCD handler to switch display off (0) or on (1) The result of setting this property depends on what other resources are currently using the LCD.

EBacklightState

`Get`: gets current state of LCD backlight, 0 if display is off, 1 if display is on

`Set`: makes request to LCD handler to turn backlight off (0) or on (1) The result of setting this property depends on what other resources are currently using the LCD.

EDisplayBrightness

`Get`: gets current state of LCD backlight brightness, in the range 0 to the value of property `EDisplay-BrightnessMax`.

`Set`: makes request to LCD handler to change the LCD backlight brightness.

The result of setting this property depends on what other resources are currently using the LCD.

EKeyboardBacklightState

This is read-only. The value is 1 if any part of the keyboard backlight is illuminated, else 0.

EManufacturerHardwareRev

This returns a 4-byte field describing the hardware platform.

Note: The implementation is not fully decided by the release of this document, but will be specified in later editions.

EPowerBatteryStatus

The value for this is derived from the current battery status published by the battery handler, but this is transparent and it behaves as the standard SymbianOS definition.

Retrieving battery status

Battery status is published to the rest of the system using the Symbian Publish and Subscribe (P&S) functionality. Category and property key values are defined in the file `SaCls.h` The battery status is published as an integer value which can have one of three values: `ESABatteryAlmostEmpty`, `ESABatteryLow` or `ESABatteryFull`.

Code example

```
#include <e32property.h>
#include <SaCls.h>

TInt capacity = -1;
TInt r=RProperty::Get(
                      KUidSystemCategory,
                      KUidBatteryStrengthValue,
                       capacity);
```

Retrieving flight mode

Flight mode status is published to the rest of the system using the Symbian Publish and Subscribe (P&S) functionality. Category and property key values are defined in the file `SaCls.h`. The flight mode status is published as an integer value which can have one of two values: `ESAPhoneOff` or `ESAPhoneOn`.

Code example

```
#include <e32property.h>
#include <SaCls.h>
```

```
TInt status = -1;
TInt r=RProperty::Get(
                        KUidSystemCategory,
                        KUidPhonePwrValue,
                         status);
```

Telephony API issues

ETEL Core API

In Sony Sony Ericsson UIQ 3 phones based on Symbian OS 9, Fax and Ownership are not supported. This implies that all RFax methods together with the following methods are **not** supported.

```
void RCall::AcquireOwnership(TRequestStatus& aStatus)
void RCall::AcquireOwnershipCancel()
TInt RCall::TransferOwnership()
TInt RCall::GetOwnershipStatus(TOwnershipStatus& aOwnershipStatus)
TInt RCall::GetFaxSettings(TFaxSessionSettings& aSettings)
TInt RCall::SetFaxSettings(const TFaxSessionSettings& aSettings)
```

HTTP framework extension

Sony Ericsson provides an extension to the HTTP framework, a filter to manage the device User-Agent and x-wap-profile headers.

The filter is automatically loaded into the HTTP session of each client, and modifies the headers of outgoing requests as follows:

- **User-Agent header**: The filter appends the device User-Agent string to any such string supplied by the client.
- **x-wap-profile header**: The filter replaces any existing header with the absolute URI of the device profile. However, there is no support for profile diffs.

UI configuration modes

This section contains information about the different Ui configurations supported in Sony Ericsson UIQ 3 phones.

More information can be found in the “Programmer’s Guide to new features in UIQ 3” section of the UIQ 3 SDK documentation.

Supported UI configurations

The QUiConfigServer in **P990** is configured to support the following UI config modes.

- **Flip Closed (softkey style UI)**
  - `KQikSoftkeyStyleSmallPortrait` (PFC), UIQ supported style
  - `KQikSoftkeyStyleSmallLandscape` (LFC)
  - `KQikSoftkeyStyleSmallLandscape180` (LFC180)
- **Flip Open (pen style UI)**
  - `KQikPenStyleTouchPortrait` (PFO), UIQ reference style
  - `KQikPenStyleTouchLandscape` (LFO), UIQ supported style
  - `KQikPenStyleTouchLandscape180` (LFO180)

In **M600, W950,** and **W958** the QUiConfigServer is configured to support the following UI config modes.

- **Softkey style UI**
  - `KQikSoftkeyStyleTouchPortrait`, UIQ supported style
- **Pen style UI**
  - `KQikPenStyleTouchLandscape`, UIQ supported style

The following UI config modes are available in the UIQ 3 SDK and supported by UIQ, but are **not** supported in Sony Ericsson Symbian OS 9 UIQ 3 phones:

- `KQikSoftkeyStylePortrait`, UIQ reference style
- `KQikSoftkeyStyleTouchPortrait`, UIQ supported style.

Default UIQ 3 flip behavior

When the UI configuration for the device changes the view layout and command list are updated according to the `QIK_VIEW_CONFIGURATIONS` to match the new configuration. All data in the `QIK_VIEW_CONFIGURATIONS` is loaded when the view is created so the controls are always reused which assures no data loss between transitions. As long as the current view is not tasked away from, the window remains active.

Dialogs will remain open when flipping. They are closed automatically by the shutter when tasking away.

When the device goes from flip open to flip closed the standby screen will be activated.

Handling UI config changes

The `HandleUiConfigChangedL` is an interface method used by the `QUiConfigServer` to notify its observers when the UI configuration has changed in the system.

The `CQikViewBase` implements the `HandleUiConfigChangedL` interface that initiates the reconstruction of the view layout and command list etc according to the `QIK_VIEW_CONFIGURATIONS`. This is all done in the framework so the application can depend on always having the layout specified in the resource for the current UI config mode.

`CEikDialog` also implements the `HandleUiConfigChangedL` interface and handles resizing and positioning according to the UI config changes. `CEikDialog` does not have any built in support for changing layouts and commands in the same way as for views though. This means that by default dialogs does not care about which UI configuration they are running in. If a specific dialog shall not be available for a specific UI configuration then it has to be modified to handle this.

Note: `CEikDialog` has been deprecated in UIQ 3. `CQikViewDialog` or `CQikSimpleDialog` should be used instead.

Accessing current UI config mode

In the DLL initialization, the application framework automatically creates a client session, that can be called from anywhere inside the application. The client can be accessed directly by calling `CQUiConfigClient::Static()`. In order to access the client it is only necessary to include the `QUiConfigClient.lib` in the mmp file and include the `QUiConfigClient.h` in the source code. Each environment creates an instance of `CQUiConfigClient` so it is never needed to be instantiated where it is used. The `Static()` does have some overhead so it should only be called once in order to minimize the execution time. If it is used extensively a reference to the client object should be used.

The following are the most important APIs that allows the developer to find out which UI configuration the device is currently in, and to change this configuration:

```
-IMPORT_C TQikUiConfig CurrentConfig() const;

-IMPORT_C TInt SetCurrentConfigL(TInt aConfigMode);
```

It is possible to set any configuration but in some cases there will be a redirection in the server. For example, If the device is in flip closed mode and an attempt is made to change to a flip open UI config mode, the server will redirect to the equivalent UI config mode for flip closed. In other words, a flip switch can never be achieved through the `CQUiConfigClient` API.

UI configurations in the emulator

The emulator does not implement the flip closed and flip open configurations as in the handset. Some of the configurations in the handset has no corresponding configurations in the emulator and some of the configurations in the emulator have no significance in the handset.

The following table lists the correspondence between the default configurations.

Emulator configuration with settings	**P990 configuration**	**M600, W950, and W958 configuration**
`#define KQikSoftkeyStyle-Portrait EQikScreenModePortrait EQikTouchScreenNo EQikUiStyleSoftkey EQikOrientationNormal`	N/A	N/A
`#define KQikSoftkey-StyleSmallPortrait EQikScreenModeSmallPor-trait EQikTouchScreenNo EQikUiStyleSoftkey EQikOrientationNormal`	`#define KQikSoftkey-StyleSmallPortrait EQikScreenModeSmallPor-trait EQikTouchScreenNo EQikUiStyleSoftkey EQikOrientationNormal`	N/A
N/A	`#define KQikSoftkey-StyleSmallLandscape EQikScreenModeSmallLand-scape EQikTouchScreenNo EQikUiStyleSoftkey EQikOrientationNormal`	N/A
N/A	`#define KQikSoftkeyStyleSmallLandscape180 EQikScreenModeSmallLand-scape EQikTouchScreenNo EQikUiStyleSoftkey EQikOrientationInverted`	N/A
`#define KQikSoftkeyStyle-TouchPortrait EQikScreenModePortrait EQikTouchScreenYes EQikUiStyleSoftkey EQikOrientationNormal`	N/A	`#define KQikSoftkeyStyle-TouchPortrait EQikScreenModePortrait EQikTouchScreenYes EQikUiStyleSoftkey EQikOrientationNormal`
`#define KQikPenStyle-TouchPortrait EQikScreenModePortrait EQikTouchScreenYes EQikUiStyleMenubar EQikOrientationNormal`	`#define KQikPenStyle-TouchPortrait EQikScreenModePortrait EQikTouchScreenYes EQikUiStyleMenubar EQikOrientationNormal`	N/A

Emulator configuration with settings	P990 configuration	M600, W950, and W958 configuration
`#define KQikPenStyle-TouchLandscape EQikScreenModeLandscape EQikTouchScreenYes EQikUiStyleMenubar EQikOrientationNormal`	`#define KQikPenStyle-TouchLandscape EQikScreenModeLandscape EQikTouchScreenYes EQikUiStyleMenubar EQikOrientationNormal`	`#define KQikPenStyle-TouchLandscape EQikScreenModeLandscape EQikTouchScreenYes EQikUiStyleMenubar EQikOrientationNormal`
N/A	`#define KQikPenStyleTouchLandscape180 EQikScreenModeLandscape EQikTouchScreenYes EQikUiStyleMenubar EQikOrientationInverted`	N/A

***Note*:** All UI mode definitions in the UIQ 3 SDK can be found in the file `Qikon.hrh`

Bluetooth keyboard APIs

HIDHOST is part of a Bluetooth Human interface device profile (HID). It is a server module, which allows remote devices to establish HID sessions via Bluetooth. The server uses the Symbian OS socket server framework to access the Bluetooth stack.

ECOM is used in order to find and instantiate plug-ins when hid devices connect to the server. Third party developers use the ECOM interface provided by the HIDHOST to develop handlers for their own HID devices.

Interface Design

The services provided by the HIDHOST are:

- Framework for supporting HID devices
- ECOM API/interface for developing HID device handlers.
- Client API with event information about HID device connects/disconnects.
- Client API for communication with plug-in.

Client API

Class RHidHostSession

`RHidHostSession` provides functionality for:

- Starting/stopping the HIDHOST service.
- Connecting to a HID device
- Event information about hid device connects and disconnects
- Information about current loaded plug-in.
- API for communication with the plug-in.

The methods for requesting notifications of events/data:

```
NotifyEvents(TRequestStatus& aRequestStatus, TDes8 & aEvent)
ReceivePluginData(TRequestStatus& aRequestStatus,TInt aOpcode, TDes8& aBuffer)
```

These methods are both “one-shot” and thus the client need to call them again after a request has completed in order to receive more events.

Information about HID device connections/disconnections are delivered in a struct named `TBtHidEvent` which contains the event code (Connect or Disconnect) and the Bluetooth device address of the connecting/disconnecting device.

HIDHOST ECom Plug-in Interface

Third party developers can implement their own BT HID device handlers by implementing an ECOM plug-in which inherits from the class `CBtPluginBase`.

Class CBtPluginBase

The base class for HID HOST ECOM plug-ins is `CBtPluginBase`. Third party plug-ins needs to inherit from this and implement the pure virtual methods declared here.

The pure virtual methods declared here provide interfaces for:

- Receiving data from a connected HID device.
- Receiving data from clients.
- Event notifications about HID device connects/disconnects.

Class MBtHidPluginProxy

The `MBtHidPluginProxy` API provides the following services to the plug-in:

- Sending data on the control and interrupt channels to the hid device.

- Sending data to a connected client.
- Sending key and raw events to the window server.
- Support for specifying which kind of data the plug-in wants to subscribe to. The data can be raw or processed – in the case of processed data the plug-in will receive complete HID boot reports via `ProcessBootReportL`. If the plug-in wants to handle raw data the server will not do anything to the incoming data on the interrupt channel – the data will be sent directly to the plug-in via `HandleIrptL`.

Default_data in the ECOM resource file

In order to develop an ECOM plug-in there must be a special resource file for the project, below an example is found.

In the default_data member in the `IMPLEMENTATION_INFO` struct, third party developers need to specify the device name of the device the plug-in is intended for.

Example:

```
RESOURCE REGISTRY_INFO theInfo
{
    dll_uid = <plug-in uid>;
    interfaces =
    {
        INTERFACE_INFO
        {
            interface_uid = HIDHOST_PLUGIN_INTERFACE_UID;
            implementations =
            {
                IMPLEMENTATION_INFO
                {
                    implementation_uid = <plug-in implementation uid>;
                    version_no = 1;
                    display_name = "3rd party BT HID Plug-in";
                    default_data = "3rd party BT Keyboard";
                    opaque_data = "";
                }
            };
        }
    };
}
```

The default_data is used by the hid host server in order to find an implementation to handle a connecting HID device, see the UIQ 3 SDK documentation for more details.

Camera APIs

For information about the Symbian OS 9.1 camera APIs, please refer to the UIQ SDK documentation, the *Symbian OS SDK v9.1/Symbian OS Reference/Multimedia ECam* section.

General

The following method returns the number of cameras in the device:

```
TInt CCamera::CamerasAvailable()
```

This method can be used for an application to avoid trying to use a non-existing camera.

Also, the `CCamera::NewL()` method will leave with error code `KErrNotSupported` if there is no camera in the device

Compliancy issues

MCameraObserver

From a power management point of view there is no difference between Reserve and Power on. The camera is powered on when it is reserved. Accordingly the camera is powered off when it is released. However, to comply with the Symbian API, the methods `PowerOn()` and `PowerOff()` are implemented, but they have no effect. These methods should still be used by clients to provide compatibility with other implementations.

When using duplicate instances of `CCamera`, `Capture` must be called from the same `CCamera` instance that prepared for it. The last `Prepare` call is the one that is valid at any time.

A service, for example a video viewfinder, must be stopped by the same `CCamera` instance that started it.

When a client receives the event `KUidECamEventCameraNoLongerReserved` it should finish processing of outstanding frames immediately. The client is given a short time to release any outstanding frames and if it has not released all frames when this time has passed, the camera implementation takes ownership of the frames and they will be deallocated. “Release frames” is the only allowed operation in this state and the client will expect no further callbacks.

The rear camera sensor is mounted on the device to be used in landscape mode. This means that if a viewfinder is started in an application which runs in portrait mode the viewfinder is rotated 90 degrees.

Other not supported methods

Method	Comment
`CustomInterface`	
`virtual void StartViewFinderDirectL(RWsSession&aWs,CWsScreenDevice& aScreenDevice,RWindowBase& aWindow,TRect& aScreenRect,TRect& aClipRect)=0;`	To start a viewfinder: `Ccamera::StartViewFinderDirectL(RWsSession& aWs,CWsScreenDevice& aScreenDevice,RWindowBase& aWindow,TRect& aScreenRect)`
`virtual void StartViewFinderBitmapsL(TSize& aSize)=0;`	
`virtual void StartViewFinderBitmapsL(TSize& aSize,TRect& aClipRect)=0;`	
`virtual void StartViewFinderL(TFormat aImageFormat,TSize& aSize)=0;`	
`virtual void StartViewFinderL(TFormat aImageFormat,TSize& aSize,TRect& aClipRect)=0;`	
`virtual void PrepareImageCaptureL(TFormat aImageFormat,TInt aSizeIndex,const TRect& aClipRect)=0;`	
`virtual void PrepareVideoCaptureL(TFormat aFormat,TInt aSizeIndex,TInt aRateIndex,TInt aBuffersToUse, TInt aFramesPerBuffer,const TRect& aClipRect)=0;`	

Camera settings

Clients should check bitfields of `TCameraInfo` and use enumeration methods to find out which services are supported and the exact formats, sizes and parameters that are supported for each service.

Zoom

Digital zoom is supported and the entries in `TCameraInfo` will indicate the ranges and values supported.

Flash

The available flash modes can be retrieved from `TCameraInfo`. The Flash is triggered by the hardware camera button.

Advanced camera settings – Autofocus

By including `ecamadvsettings.h` in the project, an application gets access to the `CCamera::CCamera-aAdvancedSettings interface`.

The following settings are supported in Sony Ericsson UIQ 3 phones, and allows applications to make use of the autofocus functionality of the built-in camera:

```
IMPORT_C TInt SupportedFocusModes() const;
IMPORT_C TFocusMode FocusMode() const;
IMPORT_C void SetFocusMode(TFocusMode aFocusMode);

IMPORT_C TInt SupportedFocusRanges() const;
IMPORT_C TFocusRange FocusRange() const;
IMPORT_C void SetFocusRange(TFocusRange aFocusRange);

IMPORT_C TInt SupportedAutoFocusTypes() const;
IMPORT_C TAutoFocusType AutoFocusType() const;
IMPORT_C void SetAutoFocusType(TAutoFocusType aAutoFocusType);

IMPORT_C TBool AutoFocusLockOn() const;
IMPORT_C void SetAutoFocusLockOn(TBool aState);
```

Note: The rest of the API is **not** supported.

Taking a picture

A camera has to be reserved and powered on to be able to prepare and take a picture.

Before capturing a picture, `PrepareImageCaptureL()` has to be called at least once to set the image format and size. The supported sizes for different formats can be retrieved from `EnumerateCapture-Sizes()`. Image capture is then requested with the method `CaptureImage()` and the camera notifies the client that an image has been captured by call back `MCameraObserver2::ImageBufferReady()`.

It is up to the client to stop the viewfinder and display the captured image on the screen.

Duplicate camera

To allow more than one client reserving the camera at the same time, it is possible to duplicate a created `CCamera` instance.

Solution

A duplicate can only be created from an existing camera object. `CCamera::Handle()` returns a unique handle of this camera object. The client can then create a duplicate camera by calling `NewDuplicateL()` which takes a handle of a `MCameraObserver2` reference. The returned `CCamera` pointer can be used to reserve the camera at the same time the original reserved it.

Code example

```
ipCamera = CCamera::NewL(*this, 0, 0);
CCamera::NewDuplicateL(*this,  ipCamera->Handle());
```

Multimedia issues

Supported MIME types

Audio oriented MIME types

File Extension	MIME types	Description
.3ga, .3gp	audio/3gpp	3GPP Multimedia File
.aac	audio/x-aac	Advanced Audio Coding
.amr	audio/amr	Adaptive Multi-Rate Codec (AMR-NB)
.amr	audio/x-amr	Adaptive Multi-Rate Codec (AMR-NB)
.au	audio/basic	uLaw/AU Audio File
.imy	audio/imelody	iMelody Ringtone Format
.mid, .midi	audio/mid	Musical Instrument Digital Interface MIDI-sequential Sound
.mid, .midi	audio/midi	Musical Instrument Digital Interface MIDI-sequential Sound
.mid, .midi	audio/x-midi	Musical Instrument Digital Interface MIDI-sequential Sound
.mid	audio/sp-midi	Scalable Polyphony MIDI
.mid	audio/vnd.semc.melody	Midi Melody (MusicDJ)
.mmf	audio/smaf	Yamaha SMAF Synthetic music Mobile Application Format
.mmf	audio/x-mmf	Yamaha SMAF Synthetic music Mobile Application Format
.mmf	application/vnd.smaf	SMAF Synthetic music Mobile Application Format
.mp3	audio/mp3	MPEG Audio Stream, Layer III
.mp3	audio/x-mp3	MPEG Audio Stream, Layer III
.mp3	audio/mpeg	MPEG Audio Stream, Layer III

File Extension	MIME types	Description
.mp3	audio/x-mpeg	MPEG Audio Stream, Layer III
.mp3	audio/mpeg3	MPEG Audio Stream, Layer III
.mp3	audio/mpg	MPEG Audio Stream, Layer III
.mp3	audio/x-mpg	MPEG Audio Stream, Layer III
.mp3	audio/mpg3	MPEG Audio Stream, Layer III
.mp4, .m4a	audio/mp4	MPEG-4 Video File
.mp4, .m4a	audio/mp4-latm	MPEG-4 Audio Layer
.mxmf	audio/mobile-xmf	Mobile XMF (eXtensible Music Format)
.rad	audio/fmradio	Sony Ericsson Mobile Communications FM radio alarm file
.ra, .ram	audio/x-pn-realaudio	RealMedia Streaming Media
.rmf	audio/rmf	Beatnik Rich Music Format
.rmf	audio/x-rmf	Beatnik Rich Music Format
.wav	audio/wav	Waveform Audio
.wav	audio/x-wav	Waveform Audio
.xmf	audio/xmf	eXtensible Music Format

Video oriented MIME types

File Extension	MIME Types	Description
.3gp	video/3gpp audio/3gpp	3GPP
.mp4	application/mpeg4, video/mpeg	ISO MPEG-4
.pvx	application/x-pv-p	Packet Video streaming link
.ram	video/x-pn-realaudio	Real Networks Video file
.ram	audio/x-pn-realaudio	Real Networks Audio file
.rm	application/x-pn-realmedia	RealVideo
.sdp	application/sdp	SDP (RFC 2327)

Audio policies for priorities

Audio policies for priorities are handled slightly differently from the description in the Symbian/UIQ SDK documentation.

In the Symbian Multimedia framework (MMF), a client playing audio can set an absolute priority and priority preference. For example, the class `CMdaAudioPlayerUtility` has a method `SetPriority(TInt aPriority, TMdaPriorityPreference aPref)` allowing priority of playback to be set.

In Sony Ericsson UIQ 3 phone implementation the priority and priority preference are ignored. Instead, all audio clients will play with the same priority, and the last started playback will interrupt any ongoing playback. If a client is interrupted and wishes to resume playback, there is a resource notification API provided by, for example `CMdaAudioPlayerUtility::RegisterAudioResourceNotification()`. By using this mechanism, a client will be notified when the audio resource becomes available.

Note: OS level components may override this behaviour. For instance, while a game is playing audio, a ring signal may start and play simultaneously. However, the ring signal will be modified to a "call waiting" beep sequence mixed in with the ongoing game audio.

Usage tips for the multimedia APIs

There are three main groups of interfaces in the Symbian Multimedia framework (MMF) which require some additional explanation beyond that given in the Symbian/UIQ SDK:

- Audio (`CMdaAudioXxxx`)
- Video (`CVideoXxxx`)
- Still Image

The `CMdaAudioXxxx` APIs are intended to be used for file and MIME types that are predominantly audio oriented. They correspond to the table of audio oriented MIME types above. Note that container file formats like mp4 are handled exclusively by the video interface since these files can potentially contain video image data which the audio APIs can not handle.
Note: The only format currently available for audio recording by third party applications is AMR-NB. Additional formats may be available in future versions.

The `CVideoXxxx` APIs are intended to be used for file and MIME types that can be video-only, audio/video or audio-only oriented. They correspond to the table of video oriented MIME types above. Note that there are significant differences of syntax between the audio utility class (`CMdaAudioXxxx`) and the video utility class (`CVideoXxxx`). Certain methods of programming against one of these interfaces may break compatibility with the other interface and vice-versa. The Symbian documentation should be read carefully to find the differences between the interfaces.

The MMF provides several APIs which support still image handling within what is collectively known as the image conversion library (ICL). The classes from earlier Symbian OS versions (`CMdaXxxx`), have been retained for backwards compatibility, but are marked as deprecated. ICL provides two main interfaces for decoding images; `CImageDecoder` and `CImageDisplay`.

The `CImageDecoder` class handles all Symbian supported formats, while `CImageDisplay` handles only the JPG and MNG subset. Both interfaces support JPG. However, the `CImageDecoder` interface provides no direct support for JPG EXIF extensions, which `CImageDisplay` does.

In the future, `CImageDisplay` is likely to include all the other Symbian supported formats, however it is unclear what release of the OS will contain this additional support.

As there is no native Symbian support for the SVG format, this is provided in the form of plugin extensions to the base OS. Animated SVG content can access the appropriate plugin via the Content Handling Framework (CHF). A separate plugin is available to access the first frame of SVG content via the `CImageDecoder` interface.

Handling multiple formats will in many cases require developers to interface to multiple APIs.

Vibration API

Vibration parameters

The method for turning on the vibration uses two parameters to determine the sequence in which the vibration is to be performed. The first parameter, `aIntervalOn`, determines the interval in 0.1 seconds for keeping the vibration on. The second parameter, `aIntervalOff`, determines the interval in 0.1 seconds for keeping the vibration off. The sequence is run until explicitly stopped.

Example:
The call
`CVibration::VibrationOn(10,10)`
runs the sequence "on for one second, off for one second" until explicitly stopped with the `VibrationOff()` call.

API blocking

Sometimes when the vibration is used internally, the public API is intentionally blocked from accessing the vibration. This means that for instance a vibration due to an incoming call can not be stopped via the public API.

Vibration errors and callback functionality

The `CVibration::NewL` and `CVibration::NewLC` methods include the option to pass an `MVibrationObserver` pointer. If this option is used, the observer object (typically the calling object) will receive a `VibrationResponse` callback on all calls made to that `CVibration` instance. If the vibration is created without an observer, no callbacks are made.

It is recommended to make any vibration client an `MVibrationObserver`, since the method provides the error code of commands being passed to the vibration. The proper way for a client to become an `MVibrationObserver` is simply to inherit from that class and implement the `VibrationResponse` method.

Syntax examples

Link against: Vibration.lib

```
#include <Vibration.h>
```

Member functions:

Constructor: Use `SonyEricsson::CVibration::NewL` or `SonyEricsson::CVibration::NewLC` to construct an object.

Turn vibration on:

```
virtual void SonyEricsson::CVibration::VibrationOn (  TUint8  aIntervalOn,
TUint8  aIntervalOff)
```

The parameters decides the requested behaviour. However, the vibration may adjust these internally in order to handle any physical vibration constraints.

Turn vibration off:

```
virtual void SonyEricsson::CVibration::VibrationOff ( )
```

Sony Ericsson MMS API

The Sony Ericsson UIQ 3 phones incorporate their own version of MMS API, different from the MMS API that is found in the standard UIQ 3 SDK.

The Sony Ericsson MMS API is based on a similar MMS API published by Nokia Series 60 Developer Platform 2.0 phones. This will allow easy porting of applications that use MMS between Series 60 and Sony Ericsson UIQ 3 phones. There are some differences, and these are detailed below.

For more information, see the Nokia Series 60 MMS API documentation.

UIQ 3 SDK emulator

When running the UIQ emulator, the UIQ MMS appears as MMS in the service list of the Messaging application. The Sony Ericsson MMS appears as SEMC_MMS in the UIQ emulator's messaging application.

On the Sony Ericsson UIQ 3 phone targets, the MMS service appears in the Messaging application as MMS in English language phones, and appropriate letters for other regions. The UIQ 3 standard MMS service is not supported on Sony Ericsson UIQ 3 phones.

Not supported methods

The following table lists all methods of the Nokia MMS API that are not supported in Sony Ericsson UIQ 3 phones.

Method	Comments

`SetMessageClass()`
`SetDefaultMessageClass()`
`DefaultMessageClass()`
	An outgoing MMS message always has its message class set as 'Personal' by Sony Ericsson UIQ 3 phones. Methods `SetDefaultMessageClass` and `SetMessageClass()` do nothing. Method `DefaultMessageClass` returns 0.

`DefaultSpeaker()`	Sony Ericsson UIQ 3 phones always uses the same speaker for playing MMS audio. This method returns 0.
`SetSecondAccessPoint()` `SecondAccessPoint()`	Only one MMS access point can be set on Sony Ericsson UIQ 3 phones. Method `SetSecondAccessPoint()` does nothing. Method `SecondAccessPoint()` returns 0.
`SetAttachmentCharsetL()`	All text attachments are sent using the UTF-16 encoding of the UCS-2 character set. This method leaves with `KErrNotSupported`.

`SetImageHeight()`
`SetImageWidth()`
`ImageHeight()`
`ImageWidth()`
	The dimensions of an attached image cannot be changed after an image file is attached to a MMS message. Methods `SetImageHeight()` and `SetImageWidth()` do nothing. Methods `ImageHeight()` and `ImageWidth()` return 0.

`SetServiceNameL()` `ServiceNameL( )`	The MMS Service does not have a name on Sony Ericsson UIQ 3 phones. Method `SetServiceNameL()` does nothing. Method `ServiceNameL()` returns an empty descriptor.
`SetSenderL()`	The MMS message sender is inserted automatically. This method leaves with `KErrNotSupported` if called.
`SetExpiryDate()`	Absolute expiry dates are not supported. This method does nothing if called. Use a relative expiry time interval instead.
`SetDeliveryDate()` `DeliveryDate()`	Absolute delivery dates are not supported. Method `SetDeliveryDate()` does nothing if called. Method `DeliveryDate()` returns 0 if called.
`SetDeliveryTimeInterval()` `DeliveryTimeInterval()`	Delivery time is not supported. Method `SetDeliveryTimeInterval()` does nothing. Method `DeliveryTimeInterval()` returns 0.

`SetSendRetryCount()`
`SetSendRetryInterval()`
`SendRetryCount()`
`SendRetryInterval()`
	Retrying to send messages that have failed to send is not supported. Methods `SetSendRetryCount()` and `SetSendRetryInterval()` do nothing if called. Methods `SendRetryCount()` and `SendRetryInterval()` return 0 if called.

`SetMessageFetchState()` `MessageFetchState()`	Message fetching cannot be controlled, and is always on. Method `SetMessageFetchState()` does nothing. Method `MessageFetchState()` returns 0.

Changed method

Method	Comments
`SetExpiryInterval`	The Nokia API uses a `TTimeIntervalSeconds` value to set the expiry interval. The Sony Ericsson MMS instead enumerates expiry intervals to a fixed set of enumerated values. These values are 1 hour, 12 hours, 1 day, 1 week and maximum. When calling `SetExpiryInterval()`, the nearest fixed value to the requested expiry interval is chosen instead.

Accessing the MMS Client MTM

Before using any functionality of the MMS client MTM, it is necessary to obtain a handle to it.

Example:

```
iSession = CMsvSession::OpenSyncL(*this);
iMtmReg = CClientMtmRegistry::NewL(*iSession);
iMmsMtm = (CMmsClientMtm*) iMtmReg->NewMtmL( KUidMsgTypeMMS );
```

Reading and Changing MMS Settings

There are two setting types:

- Settings that apply to the MMS service on the phone, for example `MaximumReceiveSize()`
- Settings that are applied to new MMS messages as default values, but can be changed for individual messages. for example `DefaultSenderVisibility()`.

Both setting types are read and changed the same way. A handle to the MMS Client MTM is required, the context needs to be set, the settings need to be loaded, the appropriate MMS Client member function called, and the settings need to be saved.

Example:

```
aMmsMtm->SwitchCurrentEntryL(aMmsMtm->DefaultSettingsL());
aMmsMtm->LoadMessageL();
aMmsMtm->SetDefaultSenderVisibility(EMmsSenderVisibilitySjow);
aMmsMtm->SaveMessageL();
```

Sending a MMS Message

Creating and sending a MMS message using the Sony Ericsson MMS API is performed in seven steps as follows:

1. Set context to the Draft folder

```
iMmsMtm->SwitchCurrentEntryL( KMsvDraftEntryId );
```

2. Create the message and retrieve its TMsvId

```
iMmsMtm->CreateMessageL( iMmsMtm->DefaultSettingsL() );
TMsvId entryId=iMmsMtm->Entry().EntryId();
```

3. Set message parameters

```
iMmsMtm->SetMessagePriority(EMmsPriorityHigh);
iMmsMtm->SetDeliveryReport(EMmsDeliveryReportNo);
iMmsMtm->SetReadReply(EMmsReadReplyNo);
iMmsMtm->AddAddresseeL( *aRecipient);
```

4. Add attachments and set their content-ids

```
TMsvId attachmentID = KMsvNullIndexEntryId;
TFileName attachmentFile1(_L("c:\\system\\apps\\MmsApiTest\\a1.txt"));
iMmsMtm->CreateAttachment2L( attachmentID, attachmentFile1 );
iMmsMtm->SetAttachmentTypeL(attachmentID, textPlain);
iMmsMtm->SetAttachmentCidL(attachmentID, _L8("A1"));
```

5. Set the message root (indicate which attachment is the SMIL)

```
iMmsMtm->SetMessageRootL(attachmentID);
```

6. Save the message, make it visible and move it to the outbox

```
TMsvEntry ent = iMmsMtm->Entry().Entry();
ent.SetInPreparation(EFalse);
ent.SetVisible(ETrue);
iMmsMtm->Entry().ChangeL(ent);

iMmsMtm->SaveMessageL();
iMmsMtm->Entry().MoveL(entryId,KMsvGlobalOutBoxIndexEntryId);
iMmsMtm->SwitchCurrentEntryL( entryId );
```

7. Send the message in the background

```
CMsvOperationWait* wait = CMsvOperationWait::NewLC();
wait->iStatus = KRequestPending;
```

```
CMsvOperation* op = NULL;
op = iMmsMtm->SendL( wait->iStatus );
wait->Start();
CleanupStack::PushL( op );
CActiveScheduler::Start();

TInt result=op->iStatus.Int();
CleanupStack::PopAndDestroy(2);
```

Error Handling

The Series 60 MMS API defines error codes. These codes are not used by the Sony Ericsson MMS API.

Some standard error detection can be discovered by looking at the result of the background send operation, as shown in the sample code above. However, this error code will not show if the MMS message has failed to send. To detect this it is necessary for the application programmer to monitor the MMS outbox for unsent messages. A MMS message that fails to send will remain in the MMS outbox.

Test Harness

A test harness has been supplied with this SDK extention. It is called UIQ_MmsApiTest. It is built the normal way, either using the CodeWarrior IDE, or from the command line using:

Emulator

```
bldmake bldfiles
abld build winscw udeb
```

Target

```
bldmake bldfiles
abld build GCCE urel
makesis MmsApiTest.pkg
$ signsis -s MmsApiTest.sis MmsApiTest_sign.sis cert.cer key.key password
```

The test harness shows how to initialize the MMS MTM, change settings, create and send a MMS message, and manipulate MMS notifications.

When Send is chosen in the test harness on the emulator, a new MMS message appears in the SEMC_MMS outbox. When the SEMC_MMS outbox has a message in it, it is not possible to switch to this outbox currently, as a panic occurs.

Choosing menu item 'Run Test' in the test harness will change your MMS settings when on target, and the MMS service may not work until you revert your settings to the correct values.

OpenGL-ES implementation

The Sony Ericsson Symbian OS 9.1/UIQ 3 platform has full support for the common and common_lite profiles OpenGL-ES 1.1. This also means that EGL 1.1 is supported. Official 1.1 header and library files can be found in the SDK.

OpenGL-ES is accelerated through dedicated 3D, MBX Lite, and floating-point hardware, VFP9. The geometry stage is not implemented in hardware but has been optimized using the floating point hardware.

All OpenGL-ES windows are double buffered and always support flipping even when not in full screen, which means that rendering to the frame buffer is never performed.

UIQ and OpenGL-ES

There are several ways of creating an OpenGL-ES application:

- Using `QikViewBase`:
  Create OpenGL-ES context and drawable in `ViewConstructL` or `ViewActivatedL` and call `DrawableWindow()` to get the `RWindow` associated with the control. The `RWindow` is submitted to `eglCreateWindowSurface` as the window argument.

  For full screen:
  ```
  TQikViewMode vm;
  vm.SetFullscreen();
  ```

- Using `CCoeControl`
  Create OpenGL-ES context and drawable in `ConstructL` and call `Window()` to get the `RWindow` associated with the control. The `RWindow` is submitted to `eglCreateWindowSurface` as the window argument.

  For full screen:
  ```
  SetExtentToWholeScreen();
  ```

Good practice is to stop the render loop and release all OpenGL-ES resources, especially texture memory and render surfaces, when the application is sent to the background. The behaviour can then be automated by creating all OpenGL-ES objects in `ViewActivatedL` and destroying them in `ViewDeactivated`.

The screen saver can be inactivated by posting `TRawEvent` like below:

```
TRawEvent  RawEvent;
RawEvent.Set(TRawEvent::EActive);
UserSvr::AddEvent(RawEvent);
```

To do this, the application must have the capability WriteDeviceData. The screen saver should not be inactivated for too long since it will disable power saving.

When running in a different orientation than the standard portrait, for example in landscape mode, the OpenGL-ES coordinate system will also be transformed accordingly. In landscape mode a full screen window will be 320 by 240 instead of 240 by 320.

OpenGL-ES development tips

Compiler flags for VFP support

When using `softvfp+vfp` floats are passed in ARM registers instead of VFP registers. Specifying only `vfp`, gives full VFP support, but make sure that the linkage specification is correct when exchanging floats with binaries that are not built for VFP. Exported functions with float arguments should be followed by `__SOFTVFP linkage macro`

Frame buffers

Both 32 and 16 bits per pixel frame buffers are supported and there is no major difference in performance, but memory usage will be twofold when using 32-bit. All 3D operations are performed using 32 bits internally.

Choosing EGLConfig

If 565 is specified for RGB in the attribute list submitted to `eglChooseConfig`, the first `EGLConfig` returned in the list will be an RGBA 8888-config. In order to get a RGB 565-config, you need to retrieve all available configs supporting the attributes and cycle through them until a RGB-565 is found.

Other tips

- `eglCreatePixmapSurface` only supports hardware bitmaps.
- It is always a lot faster to load power of two sized data using `glTexImage2D` or `glTexSubImage2D` than loading non power of two sized data using `glTexSubImage2D`.
- Currently it is not very useful to stream image data greater than 256x256 because of texture load time.
- `eglSwapInterval` is limited to 0 and 1. The default value is 1.
- Memory available for textures, window surfaces and pbuffers is ~5 MB (only applicable for M600 and P990 series). This value is not affected by the application heap.

Links and references

Links

Symbian C++ developer	http://www.symbian.com/developer/index.html (Be aware that this is not Sony Ericsson specific information)
Sony Ericsson Developer World	http://www.sonyericsson.com/developer

Reference documents

Development books

Author	Title	ISBN	Publication date
Dreamtech	Programming for embedded systems: Cracking the code	764549545	July 2002
Harrison	Symbian OS C++ for Mobile Phones	0470856114	04/17/03
Jipping	Symbian OS Communications Programming	0470844302	06/18/02
Mallick	Mobile And Wireless Design Essentials	0471214191	04/25/03
Symbian	Newcomer to Symbian OS Guide		

Index