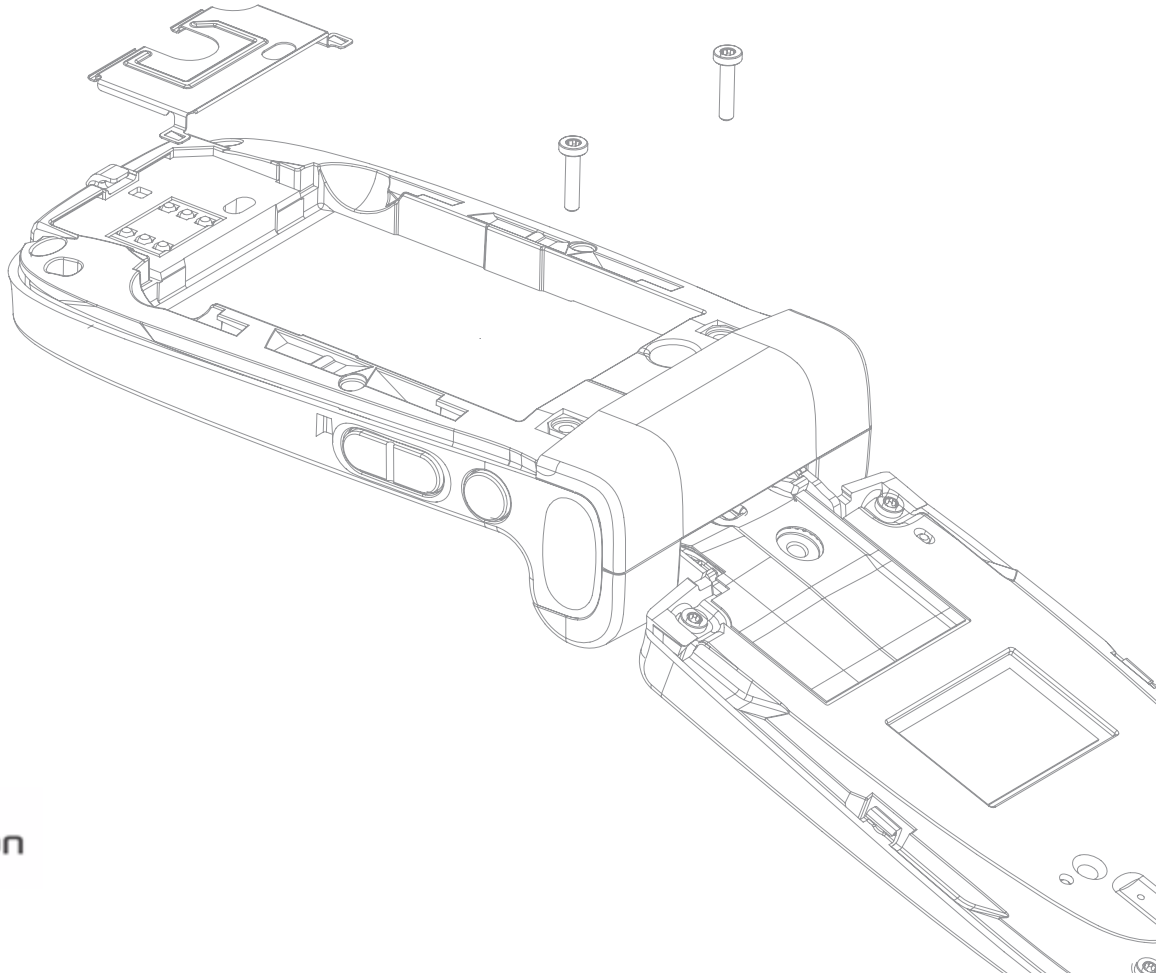
Developers guidelines

DEVELOPER WORLD THE FAST TRACK FROM MIND TO MARKET

February 2007

Java™ Platform, Micro Edition, CLDC – MIDP 2.0

for Sony Ericsson feature and entry level phones



Sony Ericsson

Preface

Purpose of this document

This document describes the Java™ ME platform support for Sony Ericsson Java platforms JP-2, JP-3, JP-4, JP-5, JP-6, and JP-7. Corresponding Developers guidelines for the Sony Ericsson Symbian Java platforms SJP-1 – SJP-3 (P900/ P910 series and P990/M600/W950 series of phones) can be found on Sony Ericsson Developer World.

Readers who will benefit from this document include:

- Software developers
- Corporate buyers
- IT professionals.
- Support engineers
- Business decision makers

It is assumed that the reader is familiar with Java.

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/

18th edition (February 2007)
Publication number: EN/LZT 108 7584, R18A

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 (Software Development Kits) and relevant APIs (Application Programming Interfaces). The Web site also contains discussion forums monitored by the Sony Ericsson Developer Support team, an extensive Knowledge base, Tips and 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 Web site.

Document conventions

Products

Sony Ericsson phones are referred to in this document by generic names (for information about Sony Ericsson Java platforms, JP-2, JP-3, and so on, see “Sony Ericsson Java platforms” on page 14):

Generic names Series	**Sony Ericsson phones**
JP-2 phone:	
Z1010	Z1010
JP-3 phones:	
F500	F500i
J300	J300i, J300c, J300a
K300	K300i, K300c, K300a
K500	K500i, K506c, K508i, K508c
K700	K700i, K700c
S700	S700i, S700c, S710a
Z500	Z500a
JP-4 phones:	
V800	V800, Vodafone 802SE
Z800	Z800i

Generic names Series	**Sony Ericsson phones**
JP-5 phones:	
K600	K600i, K608i
K750	K750i, K750c, D750i
V600	V600i
W700	W700i, W700c
W800	W800i, W800c
Z520	Z520i, Z520c, Z520a
Z525	Z525a
JP-6 phones:	
K310	K310i, K310c, K310a
K320	K320i, K320c
K510	K510i, K510c
W200	W200i, W200c
W300	W300i, W300c
W550	W550i, W550c
W600	W600i
W810	W810i, W810c, W810a
W900	W900i
Z530	Z530i, Z530c
Z550	Z550i, Z550c, Z550a
Z558	Z558i, Z558c

Generic names Series	**Sony Ericsson phones**
JP-7 phones:	
K550	K550i, K550c
K610	K610i, K610c, K618i
K790	K790i, K790c, K790a
K800	K800i, K800c
K810	K810i, K818c
W610	W610i, W610c
W710	W710i, W710c
W830	W830i, W830c
W850	W850i, W850c
W880	W880i, W888c
Z310	Z310i, Z310a
Z610	Z610i
Z710	Z710i, Z710c

Terminology and abbreviations

API
Application Programming Interface

CLDC
Connected Limited Device Configuration. A Java ME platform configuration for mobile phones

DRM
Digital Rights Management

GSM
Global System for Mobile Communications. GSM is the world's most widely used digital mobile phone system, operating in over 100 countries around the world, particularly in Europe and Asia-Pacific

HTTP
HyperText Transfer Protocol

IDE
Integrated Development Environment

Java SE
Java platform, Standard Edition

JSR
Java Specification Request

Mascot Capsule®
Mascot Capsule Micro 3D Engine is software that renders 3D objects in real-time on a display screen of an embedded device, portable game unit or mobile phone

MIDP
Mobile Information Device Profile. A Java ME platform profile connected to the CLDC for mobile phones

MMAPI
Mobile Media Application Programming Interface

OMA
Open Mobile Alliance

SDK
Software Development Kit. A collection of tools used to develop application

SMS
Short Message Service. Allows messages of up to 160 characters to be sent and received in a phone via the network operator's message centre

URI
Uniform Resource Identifier.
URIs are short strings that identify online resources: documents, images, downloadable files, services, and electronic mailboxes, for example. URIs use a variety of naming schemes and access methods, such as http, ftp, mailto and telnet, to make resources available

URL
Uniform Resource Locator. See URI

WAP
Wireless Application Protocol

WMA
Wireless Messaging API

WTK
Wireless Toolkit

Trademarks and acknowledgements

Java and all Java-based marks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

End-user license agreement for Sun Java platform, Micro Edition.

1 Restrictions: Software is confidential copyrighted information of Sun and title to all copies is retained by Sun and/or its licensors. Customer shall not modify, decompile, disassemble, decrypt, extract, or otherwise reverse engineer Software. Software may not be leased, assigned, or sublicensed, in whole or in part.
2 Export Regulations: Software including technical data, is subject to U.S. export control laws, including the U.S. Export Administration Act and its associated regulations, and may be subject to export or import regulations in other countries. Customer agrees to comply strictly with all such regulations and acknowledges that it has the responsibility to obtain licenses to export, re-export, or import Software. Software may not be downloaded, or otherwise exported or re-exported (i) into, or to a national or resident of, Cuba, Iraq, Iran, North Korea, Libya, Sudan, Syria (as such listing may be revised from time to time) or any country to which the U.S. has embargoed goods; or (ii) to anyone on the U.S. Treasury Department's list of Specially Designated Nations or the U.S. Commerce Department's Table of Denial Orders.
3 Restricted Rights: Use, duplication or disclosure by the United States government is subject to the restrictions as set forth in the Rights in Technical Data and Computer Software Clauses in DFARS 252.227-7013(c) (1) and FAR 52.227-19(c) (2) as applicable.

Borland, the Borland Logo and JBuilder are trademarks or registered trademarks of Borland Software Corporation in the United States and other countries.

NetBeans is a trademark or registered trademark of Sun Microsystems, Inc. in the U.S. and other countries.

Bluetooth is a trademark or registered trademark of Bluetooth SIG Inc.

Nokia is a registered trademark of Nokia Corporation.

Mascot Capsule is a registered trademark of HI Corporation.

ARM and Jazelle are registered trademarks of ARM Limited.

RealAudio and RealVideo are trademarks or registered trademarks of RealNetworks, Inc.

Other product and company names mentioned herein may be the trademarks of their respective owners.

Java Verified™ program for Java ME platform

The Java Verified™ Program uses the results of the Unified Testing Initiative launched by the leading mobile phone manufacturers and Sun Microsystems.

The Java Verified program gives developers a direct way to application testing and to the market. Testing Providers all over the world, covering different regions, languages and price structures, are authorized by the Java Verified program to undertake testing on behalf of the program. A developer selects one of these providers to complete the testing of their application.

An application that successfully meets both the program guidelines and passes the testing process is permitted to use the Java Powered logo. The logo is provided by Sun Microsystems, at its own discretion, on a non-exclusive license basis. An application that passes the Java Verified program testing is digitally signed so that potential distributors can be assured of its integrity and authenticity.

Once the the developer has successfully passed the testing process, their application has the opportunity to be promoted and showcased in the on-line catalogs of all Java Verified Member Companies, as well as the commercial catalogs of participating operators.

More information on the Java Verified Program is available at the Java Verified Web site (www.javaverified.com).

Document history

Change history		
2003-12-05	Version R1A	First edition
2003-12-11	Version R1B	Minor updates in technical specifications
2004-03-30	Version R2A	Document updated to comply with the latest software version of the Z1010 phone. Information about the K700 series and Z500 series added. 3D API information added
2004-07-05	Version R3A	Document updated and supplied with complementary information. Information about the S700 series and F500 added
2004-07-19	Version R4A	Information about the K500 series added
2004-11-23	Version R5A	Information about the V800 series added
2005-03-09	Version R6A	Revised edition. Information about the K300, J300, K750, W800, Z800 and K600 series added
2005-04-15	Version R6B	Revised edition. Updated specifications. Added programming tips. Sony Ericsson Java platform concept implemented in documentation
2005-04-25	Version R6C	Revised edition. Minor editorial changes
2005-08-01	Version R7A	Revised edition. Information about the V600, S600, W600 and Z520 series added
2005-08-08	Version R7B	Revised edition. S600 changed to W550
2005-09-30	Version R8A	Revised edition, adapted to the Sony Ericsson SDK for the Java™ ME Platform, ver 2.2.3
2005-10-21	Version R9A	Revised edition. Information about the W900 series added
2005-11-07	Version R9B	Revised edition. Minor editorial changes
2005-12-12	Version R9C	Revised edition. Minor changes
2006-01-04	Version R10A	Revised edition. Information about the W810 series added
2006-02-13	Version R11A	Revised edition. Information about the K610 series added
2006-02-28	Version R12A	Revised edition. Information about the K800, K790, Z530, W300, K510 and K310 series added

2006-04-12	Version R13A	13th edition. Information about the W700 and Z525 series added
2006-05-19	Version R14A	14th edition. Information about the Z550, W850, Z710 and W710 series added
2006-06-26	Version R14B	Revised edition
2006-07-07	Version R14D	Revised edition. Information about Mobile JUnit added
2006-08-22	Version R15A	15th edition. Information about the K618i and Z610 series added
2006-09-28	Version R16A	16th edition. Information about W830, K320 and Z558 series and the Z550a model added
2006-11-15	Version R16B	16th revised edition. Minor changes
2007-01-08	Version R17A	17th edition. Information about Z310 and W200 series added
2007-02-06	Version R18A	18th edition. Information about W880, K550, W610 and K810 series added

Contents

The Java ME platform

The phones covered in this document support the MIDP 2.0 and CLDC 1.1 specifications. They also have Java support for sending and receiving SMS (via WMA 1.1), playing audio and video, as well as taking snapshots from built-in cameras (via MMAPI 1.1). The basic MIDP 2.0 features, such as life cycle, memory handling etc, are the same as for the MIDP 1.0 environment. More information about MIDP 1.0 in Sony Ericsson phones is available at Sony Ericsson Developer World, (www.sonyericsson.com/developer/java). MIDP 1.0 applications developed for the T61x, T628/T630, and Z60x phones should also execute on Sony Ericsson MIDP 2.0 supported phones.

Sony Ericsson Java platforms

Sony Ericsson uses a platform approach to Java implementation allowing developers to focus on a platform rather than on a variety of different product names. Two platform branches exist, supporting Symbian (SJP) and non-Symbian (JP) based phones respectively. The platforms are implemented through an evolutionary approach in order to ensure forwards compatibility between platform versions. Normally each platform version is used in several phone models.

A list of Sony Ericsson Java platform versions for the phones in this document can be found below. Some platform features are optional, that is, configurable. For example, the Bluetooth™ APIs (JSR 82) are only enabled for phones who actually support Bluetooth wireless technology.

JP = Sony Ericsson Java platform

	Features	**Phones**
JP-2	CLDC 1.1, MIDP 2.0, JTWI (JSR 185), JSR 120, JSR 135, Nokia UI API 1.1	Z1010
JP-3	CLDC 1.1, MIDP 2.0, JTWI (JSR 185), JSR 120, JSR 135, Nokia UI API 1.1, JSR 184, Mascot Capsule Ver. 3	F500, J300, K300, K500, K700, S700 and Z500 series
JP-4	CLDC 1.1, MIDP 2.0, JTWI (JSR 185), JSR 120, JSR 135, Nokia UI API 1.1, JSR 184, Mascot Capsule Ver. 3	V800 and Z800 series
	Optional: VSCL 2.0	V800 series
JP-5	CLDC 1.1, MIDP 2.0, JTWI (JSR 185), JSR 120, JSR 135, Nokia UI API 1.1, JSR 184, Mascot Capsule Ver. 3, JSR 75	K600, K750, V600, W700, W800, Z520 and Z525 series
	Optional: JSR 82	K600, K750, V600, W700, W800, Z520 and Z525 series
	Optional: VSCL 2.0	V600 series
JP-6	CLDC 1.1, MIDP 2.0, JTWI (JSR 185), JSR 120, JSR 135, Nokia UI API 1.1, JSR 184, Mascot Capsule Ver. 3, JSR 75, JSR 172, JSR 205	K310, K320, K510, W200, W300, W550, W600, W810, W900, Z530, Z550 and Z558 series
	Optional: JSR 82	K320, K510, W300, W550, W600, W810, W900, Z530, Z550 and Z558 series
JP-7	CLDC 1.1, MIDP 2.0, JTWI (JSR 185), JSR 120, JSR 135, Nokia UI API 1.1, JSR 184, Mascot Capsule Ver. 3, JSR 75, JSR 172, JSR 205, JSR 234 (camera capabilities)	K550, K610, K790, K800, K810, W610, W710, W830, W850, W880, Z310, Z610 and Z710 series. **Note:** JSR 184, Mascot Capsule Ver. 3 and JSR 234 are **not** enabled in Z310 series
	Optional: JSR 82	K550, K610, K790, K800, K810, W610, W710, W830, W850, W880, Z610 and Z710 series

MIDP 2.0 support

The phones covered in this document are MIDP 2.0 and JTWI 1.0 compliant.
For a list of protocols, formats, memory size, display size etc. supported by the MIDP 2.0 implementation in the phones, see “Appendix A Phone specifications” on page 29, which contains technical specifications for each phone.

The MIDP 2.0 specification contains a number of optional features of which the following are supported:

- PushRegistry Alarm and PushRegistry SMS. In *JP-4*, *JP-5*, *JP-6*, and *JP-7* PushRegistry CBS is also supported.
- Signed MIDlets as specified in JTWI 1.0.
- TCP and UDP server sockets as specified in MIDP 2.0.
- `PlatformRequest` supports the tel, http and https schemes.
  When the method is invoked with the tel scheme, the native phone application is accessed and the user can initiate a voice or video call, or send a message to the given phone number.
  A `PlatformRequest` invocation for http/https initiates downloading of the given URI, for example, a Java application, image etc. For http/https URIs referencing WAP pages, the Web browser is invoked. The Java application is then left in the background until the phone call/download/Web session is completed, after which it is resumed.
- `GameCanvas.getKeyStatus()` supports the detection of several simultaneous keys. See also “Simultaneous key presses” on page 22.
- `TextBox` and `TextField` with input constraints ANY, EMAILADDR and URL support the character set specified in JTWI 1.0.
- PNG images with colour depth of 1, 2, 4, 8, 16, 24 and 32 bits per pixel are supported.
- The maximum number of application-created threads is limited only by the amount of available memory.
- A `TextBox` or `TextField` object with input constraint `TextField.PHONENUMBER` allows the user to select a phone number from the phonebook, as specified in JTWI.
- In *JP-7* `CommConnection` is implemented, but requires an AT command, AT*SEJCOMM, to open a port before `CommConnection` can be used by a MIDlet.
- The Z558 series features a touchscreen, with support for writing recognition (in Chinese and English). The standard pointer control methods of the MIDP `Canvas` class are supported for this series of phones.

WMA (JSR 120)

The Wireless Messaging API v 1.1 (JSR 120) is supported. GSM SMS is supported in all phones covered in this document, while GSM Cell Broadcast (CBS) is only supported in *JP-4*, *JP-5*, *JP-6* and *JP-7*.
MIDP 2.0 security has been added to the Open connection, Send and Receive functions, as specified in WMA 1.1, http://www.jcp.org/en/jsr/detail?id=120.

The Sony Ericsson SDK for the Java™ ME platform provides support for developing WMA MIDlets. This includes API documentation, support for compiling WMA MIDlets and debugging these MIDlets using any of the phones covered in this document.

Per Appendix A, “GSM SMS Adapter”, of the WMA specification, implementations of the GSM SMS adapter must support at least three concatenated short message segments. The phones covered in this document exceed this minimum requirement, allowing MIDlets to send and receive SMS messages of up to ten segments in length.

The 3GPP specification for SMS specifies the port numbers 16000-16999 as available for applications. It is recommended that Java developers use non-reserved port numbers within this range. WMA has a system list of restricted port numbers which may not be used by Java applications. In addition to the port numbers restricted in the WMA specification, the phones covered in this document also reserve the ports listed in the table below. If a Java application attempts to use any of the restricted and/or reserved ports, an exception will be thrown.

Port number	Description
0	Internal system use
650	General obex
2948-2949	WAP
5505	PM ringtone (Nokia Smart Messaging)
5506	PM Logo (Nokia Smart Messaging)
5507	PM Icon (Nokia Smart Messaging)
5514	Picture message (Nokia Smart Messaging)
9200-9207	WAP
16733	Calendar
16987	Email notification
16988	Email account setting
49996-49997	WAP provisioning
49999	WAP provisioning

WMA 2.0 (JSR 205)

Note: The JSR 205 API is only supported on the *JP-6* and *JP-7* platforms.

The Wireless Messaging API 2.0 is an extension and enhancement of WMA (JSR 120). GSM SMS, GSM Cell Broadcast (CBS), and MMS are supported.

WMA 2.0 is based on the Generic Connection Framework (GCF), which is defined in the Connected Limited Device Configuration (CLDC) 1.0 specification. The package `javax.microedition.io` defines the framework and supports input/output and networking functionality in Java ME profiles.

The JSR 205 specification can be downloaded from http://www.jcp.org/en/jsr/detail?id=205.

Note: Sending DRM protected files as message parts is **not** supported in the Sony Ericsson implementation of the JSR 205 API.

MMAPI (JSR 135)

The MMAPI support in the phones in this document provides access to audio and video playback, as well as image capture with the phone camera. For a list of supported data formats for each phone, see “Java specifications” on page 32. The JSR 135 specification can be downloaded from http://www.jcp.org/en/jsr/detail?id=135.

Players can be created from:

- Java streams
- DataSources
- URIs with “http://”, “https://” or “capture://video”

Audio support

See also “Java specifications” on page 32.

In *JP-2* and *JP-3*, a maximum of 16 audio players can exist at the same time in Started state at the Java level. The number of players that can produce audio in parallel is limited by the phone hardware. Simple tones can be generated in parallel to any of the supported audio formats, but no other parallel audio playback is supported.

In *JP-4*, *JP-5*, *JP-6* and *JP-7* the number of simultaneously started players is limited only by available memory. These phones also support more advanced mixing. In *JP-4* and *JP-5*, one player can play a waveform audio file (.WAV or ADPCM) with a sample rate of 8 or 16 kHz in parallel to another player playing a MIDI file. In *JP-7* up to 4 parallel players are supported, either 4 waveform players (AMR, .WAV or ADPCM) or 3 waveform players and 1 player playing a MIDI file.

The MidiControl, supported in *JP-5*, *JP-6* and *JP-7*, allows control of a maximum of 16 MIDI channels when playing MIDI.

The following controls are implemented:

- VolumeControl
- ToneControl
- StopControl
- MetaDataControl (from *JP-6*)
- MIDIControl (from *JP-5*)
- PitchControl (from *JP-6*)
- TempoControl (*JP-7*)
- RateControl (*JP-7*).

JP-6 and *JP-7* phones (except Z310) support audio capture in the MMAPI.

Video support

See also “Java specifications” on page 32.

Note: Video playback is **not** supported in *JP-2* phones and in Z310 series.

Only one video player can exist at a particular time. The video player can display its contents in a `Canvas` or in an `Item` on a `Form`.

The snapshot functionality is only supported for taking a picture with the built-in camera of the phone. Access to the camera snapshot functionality follows the security policy specified in JTWI.

Video recordings from the built-in camera is supported in *JP-7* phones except Z310.

When a phone call is received while running a Java application that uses the native camera, the Java reference to the native camera is released. Once the phone call is terminated, the application will regain focus and an END_OF_MEDIA_EVENT is sent to the application. It is then up to the application whether to restart the camera or not.

The following controls are implemented:

- VideoControl
- FramePositionControl (*JP-7* only) allows precise positioning of a video frame for the player.

Advanced Multimedia Supplements (JSR 234)

Note: Only *JP-7* phones, except Z310, support the JSR 234 API.

Advanced Multimedia Supplements (AMMS) builds on the framework already established in the Mobile Media API (MMAPI) (JSR 135). AMMS adds many new controls and extensions to the MMAPI framework.

The full JSR 234 specification can be downloaded from http://www.jcp.org/en/jsr/detail?id=234

In Sony Ericsson *JP-7* phones, the following classes/interfaces for extended camera and image handling functionality have been implemented:

- `javax.microedition.amms.control.FormatControl`
- `javax.microedition.amms.control.ImageFormatControl`
- `javax.microedition.amms.control.camera.CameraControl`
  All methods supported, **except** `getCameraRotation()`
- `javax.microedition.amms.control.camera.ExposureControl`
  Not supported: `setExposureTime()`, `getExposureTime()`, `getExposureValue()`, `getFStop` and `setFStop()`

- `javax.microedition.amms.control.camera.FocusControl`.
  Only supported in K800 and K790.
  No support for “servo focus”, that is, focus does not change automatically when the motive changes.
  Macro focusing is supported.
  Manual focusing is not supported.
  To utilize auto focusing:
  To focus, call `FocusControl.setFocus(FocusControl.AUTO)`, and the camera focuses on the object currently in the viewfinder. An ongoing focusing procedure can be interrupted by calling `FocusControl.setFocus(FocusControl.AUTO_LOCK)`, however this call has no effect if the camera has already locked focus on the motive

- `javax.microedition.amms.control.camera.SnapshotControl`

- `javax.microedition.amms.control.camera.ZoomControl`
  Only supported in K800 and K790

- No support for *optical* zoom features. All *digital* zoom features are supported. `getMinFocalLength()` is not supported

- `javax.microedition.amms.GlobalManager`

3D APIs

Note: *JP-2* phones and Z310 series do **not** support the 3D APIs.

JP-3, *JP-4*, *JP-5*, *JP-6* and *JP-7* (except Z310) support real-time 3D graphics rendering. These platforms support two different 3D graphics APIs, Mascot Capsule Micro3D version 3 and Mobile 3D Graphics API for J2ME (JSR 184). For more information on the implementation of the 3D APIs, see the Developers guidelines “3D graphics with Java ME”, available at www.sonyericsson.com/developer/java.

PDA optional packages (JSR 75)

Note: Only *JP-5*, *JP-6* and *JP-7* support the JSR 75 API.

The PDA optional packages for Java ME (JSR 75) consist of two separate APIs, one for accessing PIM data and one for file system access.

PIM optional package

The PIM (Personal Information Management) API is standardized in the JSR 75 specification, which can be downloaded from http://www.jcp.org/en/jsr/detail?id=75. The following describes shortly the implementation in the Sony Ericsson phones where the API is supported.

In Sony Ericsson phones the PIM API handles:

- Contacts (ContactList)
- Calendar (EventList)
- Tasks (ToDoList).

For more details on the Sony Ericsson implementation of the PIM package, see “Appendix B Java programming issues” on page 41.

File Connection optional package

The File Connection API is standardized in the JSR 75 specification, which can be downloaded from http://www.jcp.org/en/jsr/detail?id=75. The following describes shortly the implementation in the Sony Ericsson phones where the API is supported.

In general, Java applications can access the same folders, subfolders and files as the built-in File manager application, both in phone internal memory and on an inserted memory card. The following folders and all contained subfolders and files are accessible via the API:

- <file:///c:/> (internal memory file root)
- <file:///c:/other/>
- <file:///c:/pictures/>
- <file:///c:/sounds/>
- <file:///c:/videos/>
- <file:///e:/> (memory card file root)
- <file:///e:/dcim/> (camera pictures folder on memory card).

Note: The folders *Games*, *Themes*, *Applications* and *Webpage* are **not** available via the File Connection API.

Note: Which folders are accessible via the File Connection API differ between phone models. For example, in JP-7 phones, there is a *Camera* folder in phone internal memory, <file:///c:/camera>, and the *Themes* and *Webpage* folders on memory card are accessible.
The PDAPDemo application supplied with the Sony Ericsson SDK for the Java ME platform is recommended to find out exactly which folders are accessible in internal memory and installed memory card of a specific phone.

For more details on the Sony Ericsson implementation of the package, see Appendix B, “JSR 75 implementation” on page 51.

Bluetooth API (JSR 82)

See also “Java specifications” on page 32.

JP-5, *JP-6* and *JP-7* phones, support JSR 82, the standard Java API for Bluetooth, as an optional feature. It provides the means for developers to create Bluetooth games and other applications as well as implement new Bluetooth profiles.

For example, the Bluetooth API offers developers the ability to:

- Create multiplayer games
- Connect to PCs from Java applications.

The complete JSR 82 specification can be downloaded from the Java Community Pages, http://www.jcp.org/en/jsr/detail?id=82

Note: JSR 82 v1.1 is supported from JP-7.4 and onwards, **except** for the Push Registry features. JP-5, JP-6 and JP-7.0 – JP-7.3 phones support JSR 82 v1.0a.

Java ME Web Services 1.0 (JSR 172)

Note: The JSR 172 APIs are only supported on the *JP-6* and *JP-7* platforms.

The JSR 172 contains two independent, optional packages, both supported on the *JP-6* and *JP-7* platforms:

- Java ME XML Parser
- Java ME RPC, which facilitates access to XML based Web services from CDC and CLDC based profiles.

The complete JSR 172 specification can be downloaded from the Java Community Pages, http://www.jcp.org/en/jsr/detail?id=172

Memory

The phones covered in this document utilize a number of different memory areas for user interface features, and for images in particular. The total amount of memory available varies depending on how much of this memory other native phone applications have currently allocated. If needed, and if memory is available, the Java heap grows dynamically up to about 1.5 MB. In general, a Java application with around 500 kB of image data is executable.

For more information about memory in different phones, see “Java specifications” on page 32. More information about memory allocation can be found in Appendix B, “Memory usage” on page 43.

The navigation key

The phones covered in this document detect navigation key actions in the following manner:

- Two adjacent directions are simultaneously detected. If the navigation key is pressed in one of the four main directions, up, down, left or right, one event is delivered to the application. If the navigation key is pressed in a diagonal direction, two events are delivered to the application, for example, one for “up” and one for “right”
- Navigational changes are detected directly without having to go back to neutral position.

Simultaneous key presses

Support for simultaneous key presses enhances gaming experience. For example, a user playing a game can move around on the screen and shoot at the same time.

Most Sony Ericsson phones support simultaneous key presses. However, a MIDP developer can not take for granted that a certain phone model supports simultaneous key presses in all possible combinations. Games and other applications should always be tested with the actual targeted hardware. Hardware emulators does not necessarily emulate simultaneous key presses properly.

In general, when two keys are pressed at the same time, the proper events are delivered to the application. When three or more keys are pressed in some combinations, extra key presses may be detected. In other combinations, a third key is not detected at all. The general approach when more than two keys need to be detected at the same time, is to map the game keys (Fire, game A, game B, and so on) to actions that might occur at the same time as two or more other key presses.

More specific information for the simultaneous key press functions can be found in Appendix A, “Key mapping” on page 38.

Command types

The MIDP commands defined by an application are displayed on either the left selection key, the right selection key or they are placed in the “More” menu which is associated with the right selection key. Command type BACK is always mapped to the back button of the phone. Command type OK is generally mapped to the left selection key. It is recommended to always include BACK and OK commands. The command types are prioritized in the following order (from higher to lower):

- OK
- ITEM
- SCREEN
- BACK
- CANCEL
- EXIT
- STOP
- HELP

If a command of type BACK exists it will be mapped to the back key of the phone. It is also recommended to use the BACK command instead of the EXIT command for exiting the application, allowing the user to press the back key of the phone to exit. This conforms to the normal behaviour of Sony Ericsson phone applications.

Of all the remaining commands (excluding the one mapped to BACK) the one with the highest priority is mapped to the left selection key. All other commands are mapped to the right selection key. If more than one command is to be mapped to the right selection key, a “More” option is displayed and a list of the commands appears when the user presses the right selection key.

Error messages

Java exceptions that are not handled by the active MIDlet are dealt with by the Java environment. The following error messages are displayed to the user:

Java exception	**Error message displayed to user**
java.io.IOException	Network failure
javax.microedition.io.ConnectionNotFoundException	Network failure
java.lang.ClassNotFoundException	Invalid application
java.lang.OutOfMemoryError	The application consumes too much memory
java.io.EOFException, java.io.UnsupportedEncodingException, java.io.UTFDataFormatException	Network data error
javax.microedition.rms.RecordStoreFullException	Application memory full
All other Exceptions or Errors	Application error

Sony Ericsson SDK for the Java™ ME Platform

Development of Java applications for the phones covered in this document is supported by the Sony Ericsson SDK for the Java™ ME Platform. This includes PC emulation based on a customized version of the Wireless Toolkit from Sun. For example, screen size, colour depth and key inputs of the phone are emulated.

The latest version of the Sony Ericsson SDK is available for download at www.sonyericsson.com/developer/java.

For more information about the SDK, see “Appendix C Sony Ericsson SDK for the Java™ ME Platform” on page 66.

Security policy for Sony Ericsson phones

All of the phones described in this document comply with the JSR185 Java Technology for the Wireless Industry (JTWI) specification and MIDP 2.0 recommended security policy. For a detailed description of the installation and security rules, see Chapter 7, *Security Policy for GSM/UMTS Compliant Devices* of the JSR 185 specification (http://www.jcp.org/en/jsr/detail?id=185). A number of APIs are categorized as “restricted”. Usage can result in costs for the user (traffic charges), inappropriate use may potentially affect the user data integrity or cause disturbance to other parties. The following tables describe the specific security configuration implementation on Java platforms *JP-2*, *JP-3*, *JP-4*, *JP-5*, *JP-6* and *JP-7*.

Permission settings

The following table lists definitions of permission settings:

Permission setting	Screen description	Definition	Standard description
1	Yes, Ask Once per Session	Ask the first time an application requests this function, then retain this setting for the remainder of the session	Session (Yes)
2	Yes, Always Ask	Ask every time the application requests access to this function	One-shot (Yes)
3	No, Ask Later (Runtime only)	Do not accept the request to the function at this time, however next time this function is needed during this session please ask again	One-shot (No)
4	No Access No, Never Grant (Runtime Only)	Do not grant permission to this function during this session and do not ask again	Session (No)
5	Not Allowed	This function is not granted access by the operator. The Application is not allowed access to this function and the user is unable to modify this setting	No
6	Allowed	This function is always granted access by the operator	Blanket
7	Yes, Never Ask Again Blanket Permission	User Defined - never ask again for permission, permission always granted	Blanket (Yes)

Security Configuration

This table lists permission settings per functionality and security domain:

Functionality/Domain	**Untrusted**	**Trusted 3rd party**
Data Network		
javax.microedition.io.HttpConnection	1, 2, 4	1, 7, 4
javax.microedition.io.HttpsConnection	1, 2, 4	1, 7, 4
javax.microedition.io.Connector.datagram	1, 2, 4	1, 7, 4
javax.microedition.io.Connector.datagramreceiver datagram server (w/o host)	1, 2, 4	1, 7, 4
javax.microedition.io.Connector.socket	1, 2, 4	1, 7, 4
javax.microedition.io.Connector.serversocket server socket (w/o host)	1, 2, 4	1, 7, 4
javax.microedition.io.Connector.ssl ssl	1, 2, 4	1, 7, 4
All App Auto-Start		
javax.microedition.io.PushRegistry	1, 2, 4	2, 7, 4
Messaging- Wireless Messaging API - JSR 120		
javax.wireless.messaging.sms.send	2, 4	2, 4
javax.wireless.messaging.sms.receive	2, 4	2, 4
javax.microedition.io.Connector.sms	2, 4	2, 4
javax.wireless.messaging.cbs.receive (*JP-4*, *JP-5*, *JP-6* and *JP-7* **only**)	2, 4	2, 4
javax.microedition.io.Connector.cbs (*JP-4*, *JP-5*, *JP-6* and *JP-7* **only**)	2, 4	2, 4
PIM and File Connection APIs (JSR 75, *JP-5*, *JP-6* and *JP-7 only*)		
javax.microedition.pim.ContactList.read	1, 2, 4	2, 7, 4
javax.microedition.pim.ContactList.write	1, 2, 4	2, 7, 4
javax.microedition.pim.EventList.read	1, 2, 4	2, 7, 4
javax.microedition.pim.EventList.write	1, 2, 4	2, 7, 4
javax.microedition.pim.ToDoList.read	1, 2, 4	2, 7, 4
javax.microedition.pim.ToDoList.write	1, 2, 4	2, 7, 4
javax.microedition.io.Connector.file.read	2, 4 (from JP-7.4: 1, 4)	2, 7, 4 (from JP-7.4: 1, 7, 4)
javax.microedition.io.Connector.file.write	2, 4 (from JP-7.4: 1, 4)	2, 7, 4 (from JP-7.4: 1, 7, 4)

Functionality/Domain	Untrusted	Trusted 3rd party
Vodafone Specific Class Library (VSCL 2.0 in Vodafone customized phones. Parts of VSCL 2.1 in JP-5, JP-6 and JP-7 phones customized for Vodafone)		
com.vodafone.midlet	4	2

Note: Cingular have their own Java signing and permission process, which should be considered before purchasing a signing certificate for applications to be installed on Sony Ericsson phones customized for Cingular. For more information, see the Cingular developer site, http://developer.cingular.com/developer/technologies/java/signing.jhtml (registration required).

Note: *Unsigned* MIDlets are not allowed to:

- open datagram connections on ports 9200, 9201 or 9203
- open socket connections on ports 80, 443 or 8080
- open SSL connections on port 443.

The security domain is determined at installation as follows:

- If the midlet suite is unsigned, then it will be installed in the “Untrusted” domain
- If the midlet suite was signed using a certificate granted by a trusted third party such as Verisign or Thawte, then it will be installed in the "Trusted 3rd party" domain. Operators maintain control of their certification process
- A signed midlet suite is not installed if certificate verification fails, for example, when a midlet suite, signed by one operator, is attempted to install on a phone issued by another operator. In other words, operator signatures are not generic, but are specific to phones provided by each individual operator

The digital certificate embedded in the JAD and the signed JAR file are verified for authenticity and date validity at install time according to chapter 4 of the JSR 118 specification (http://www.jcp.org/en/jsr/detail?id=118). This assures data integrity and vendor identity.

Certificates in Sony Ericsson phones

The table below lists “factory installed” root certificates in Sony Ericsson phone models/series. The table is valid for the first released version of the different phones, later releases may in some cases contain more certificates.

Phone model/series	Certificates		
	UTI from GeoTrust (Java Verified)	**Verisign**	**Thawte**
Z1010			
F500			
J300, K300	•		•
K500			
K700			
S700i, S700c			

Phone model/series	Certificates		
	UTI from GeoTrust (Java Verified)	Verisign	Thawte
S710a	•	•	
Z500	•	•	
V800	•		•
K310, K320, K510, K550, K600, K610, K750, K790, K800, K810, V600, W200, W300, W550, W600, W610, W700, W710, W800, W810, W830, W850, W880, W900, Z310, Z520, Z525, Z530, Z550, Z558, Z610, Z710, Z800	•	•	•

Download and installation

The typical distribution mechanism for MIDP applications is over the air (OTA) via WAP or HTTP. The JAD and or JAR file(s) are accessible via the Internet and users may access either file. After downloading via the phones Browser application, installation is automatic.

Note: When JAR files are downloaded OTA via a WAP gateway, the file size may be limited by the network operator.

In the case of a signed midlet, the user must access the JAD file, because the signature is in it. The JAD file is read and the URL property in the file is used to access the JAR file. Transferring the JAR file via Bluetooth, IR or serial/USB cable do not work, since these methods only work with unsigned midlets.

Signed midlets may also be installed using the correct JAD file with the Sony Ericsson SDK for the Java™ ME via the DeviceExplorer, ejava.exe command line tool or by right-clicking the file and selecting the “Install on Device” option.

A signed midlet can be installed on a phone with no UTI root certificate, by removing the following JAD/ Manifest attributes and the corresponding values before installation:

- MIDlet-Certificate-1-1:
- MIDlet-Jar-RSA-SHA1:

A list of JAD attributes supported in MIDP2 compliant Sony Ericsson phones can be found in Appendix B, see “JAD/manifest attributes” on page 48.

Java applications can be installed on the memory card as well as in internal memory in *JP-4*, *JP-5*, *JP-6* and *JP-7* phones with memory cards. **Note:** This is not possible in the S700 series.

To install a MIDlet on the memory card:

- Transfer the application files (JAR/JAD) to the directory *mssemc**media files**other* in the phone file system.
- From the phone main menu, select File manager (Data folder) and browse to the application in the *other* directory. Select **Install**.

Appendix A
Phone specifications

In this appendix the technical specifications are listed for the phones covered in this Developers guideline.

Note: market/customer variations in the specifications may exist.

Screen and memory specifications

Screen sizes are specified as Width x Height (pixels).

Specification/Phone	Z1010	K700	S700	F500, K500, Z500, Z520, Z525	V800, Z800
Screen					
Screen size	176x220	176x220	240x320	128x160	176x220
Fullscreen canvas size	176x220	176x220	240x320	128x160	176x220
Non fullscreen canvas size	176x182	176x176	240x266	128x128	176x182
Pixel ratio (H:W)	1:1	1:1	1:1	1:1	1:1
Colour depth	65,536 (16-bit)	65,536 (16-bit)	262,144 (18-bit) [a]	65,536 (16-bit)	262,144 (18-bit) [a]
Transparency	Full (8-bit) alpha blending				
Memory					
Max. RMS size	Limited only by the amount of available free storage.				
Memory, storage	8 MB	40 MB	32 MB	F500 10 MB K500 10 MB Z500 6 MB Z520 16 MB Z525 16 MB	V800 8 MB [b] Z800 5 MB [b]
	Note: The amount of memory available for Java applications depends on the free amount of internal memory in the phone. Other contents, such as pictures, video clips and themes, use the same memory pool				
Java heap size	512 kB - 1.5 MB (dynamic, depending on available memory)				
Max. JAR size	Unlimited, but depending on available storage.				
Native video RAM available to Java	Approx. max 500 kB				

a) In Java only 65,536 colours (16-bit colour depth) can be used.

b) Java applications can be installed on the memory card as well as in internal memory.
To install a MIDlet on the memory card:

- Copy the application files (JAD/JAR) to the directory *\mssemc\media files\other* in the phone file system.
- From the phone main menu, select Data folder and browse to the application in the *other* directory. Select **Install**.

Screen and memory specifications - continued

Specification/Phone	J300, K300	K550, K600, K610, K750, V600, W550, W600, W610, W700, W710, W800, W810, Z550, Z558, Z610, Z710	K790, K800, K810, W830, W850, W880, W900	K310, K320, W200, Z310, Z530	K510, W300
Screen					
Screen size	128x128	176x220	240x320	128x160	128x160
Fullscreen canvas size	128x128	176x220	240x320	128x160	128x160
Non fullscreen canvas size	128x110	176x176	240x266	128x128	128x128
Pixel ratio (H:W)	1:1	1:1	1:1	1:1	1:1
Colour depth	65,536 (16-bit)	262,144 (18-bit) [a]	262,144 (18-bit) [a]	65,536 (16-bit)	262,144 (18-bit) [a]
Transparency	Full (8-bit) alpha blending				
Memory					
Max. RMS size	Limited only by the amount of available free storage.				
Memory, storage	8 MB	K550 64 MB[b] K600 37 MB K610 64 MB[b] K750 34 MB[b] V600 32 MB W550 256 MB W600 256 MB W610 64 MB[b] W700 34 MB[b] W710 10 MB[b] W800 32 MB[b] W810 21 MB[b] Z550 20 MB[b] Z558 18 MB[b] Z610 64 MB[b] Z710 10 MB[b]	K790 64 MB[b] K800 64 MB[b] K810 64 MB[b] W830 64 MB[b] W850 64 MB[b] W880 16 MB[b] W900 470 MB[b]	K310 15 MB K320 15 MB W200 20 MB Z310 14 MB Z530 28 MB[b]	K510 28 MB[b] W300 20 MB[b]
	Note: The amount of memory available for Java applications depends on the free amount of internal memory in the phone. Other contents, such as pictures, video clips and themes, use the same memory pool				
Java heap size	**JP-3, JP-5:** 512 kB - 1.5 MB (dynamic, depending on available memory) **JP-6**: 1.0 - 1.5 MB (dynamic, depending on available memory) **JP-7**: dynamic, depending on available memory				

Specification/Phone	J300, K300	K550, K600, K610, K750, V600, W550, W600, W610, W700, W710, W800, W810, Z550, Z558, Z610, Z710	K790, K800, K810, W830, W850, W880, W900	K310, K320, W200, Z310, Z530	K510, W300
Max. JAR size	Unlimited, but depending on available storage.				
Native video RAM available to Java	Approx. max 500 kB				

a) In Java only 65,536 colours (16-bit) can be used.

b) Java applications can be installed on the memory card as well as in internal memory.
To install a MIDlet on the memory card:

- Copy the application files (JAD/JAR) to the directory *\mssemc\media files\other* in the phone file system.
- From the phone main menu, select the Data folder and browse to the application in the *other* directory. Select **Install**.

Java specifications

The table lists the Java characteristics of the phones covered in this document.

Characteristic	Support	Comments
CLDC version	1.1	
MIDP version	2.0 *Supported image formats*: GIF87a, GIF89a, PNG v 1.0 (colour depth 1, 2, 4, 8, 16 bits per pixel), BMP v 3.x, WBMP level 0 *Networking:* secure sockets, http 1.1, https. TLS 1.0 is also supported *Serial communication*: JP-7 phones implement the `CommConnection` interface via the AT command port. The AT command port must be set to transparent mode with the AT*SEJCOMM AT command before serial communication can proceed.	See also “MIDP 2.0 support” on page 15 See also “Serial Port Communications (JP-7 only)” on page 49

Characteristic	Support	Comments
JTWI (JSR 185) compliant	Yes, Release 1	
MMAPI (JSR 135)	1.1 *Supported Audio Content types*: • audio/midi - MIDI (GM, GML and SP-MIDI) • audio/x-wav - WAV (PCM) • audio/x-tone-seq - JSR 135 tone sequence • audio/mpeg - MP3 (MPEG-1 layer 3, MPEG-2 layer 3, MPEG 2.5 layer 3) • audio/imelody - iMelody • audio/emelody - eMelody • audio/amr - AMR • audio/mp4a-latm - 3GP (MPEG-4 AAC LC) • audio/x-pn-realaudio (.ra) - RealAudio®, ver. 8 • audio/x-ms-wma – Windows media audio. *Supported Video Content types*: • video/mp4v-es - 3GP (MPEG-4 Visual Simple Profile Level 0) • video/h263-2000 - 3GP (H.263 Baseline Profile 0 Level 10) • video/x-pn-realvideo (.rm) - RealVideo®, ver. 8 • GIF89a animations are supported from *JP-6* • video/x-ms-wmv – Windows media video. *Supported Image (Camera) Content types*: • image/jpeg - JPEG	See also “MMAPI (JSR 135)” on page 17. **Note:** Not all content types are supported in all phones **Note:** Video playback is **not** supported in *JP-2* and Z310 series **Note:** Not all content types are supported in all phones See also “Camera specifications” on page 35 **Note:** The camera in Z310 series is not accessible from Java.
AMMS (JSR 234)	Extended camera and image handling functionality.	**Note:** Only supported in *JP-7* **except** Z310. See also “Advanced Multimedia Supplements (JSR 234)” on page 18
WMA (JSR 120)	1.1 - GSM SMS	See also “WMA (JSR 120)” on page 15
WMA 2.0 (JSR 205)	GSM SMS GSM CBS MMS	**Note:** Only supported in *JP-6* and *JP-7*

Characteristic	Support	Comments
PDA optional packages for Java ME (JSR 75)	Version 1.0 *PIM API, supported package*: • Javax.microedition.pim *PIM API, supported classes/interfaces*: • Contact • Event • ToDo • Serialization methods on PIM items • Serialization of PIM items according to vCard 2.1/vCalendar 1.0. *File connection API, supported package*: • javax.microedition.io.file.	**Note:** Only supported in *JP-5*, *JP-6* and *JP-7* See also "PDA optional packages (JSR 75)" on page 19 and "JSR 75 implementation" on page 51
Java Bluetooth API (JSR 82)	Version 1.0a, from **JP-7.4** version 1.1 *Supported packages*: • javax.bluetooth • javax.obex. *Supported connections*: • L2Cap (btl2cap://) • Serial Port Profile (btspp://) • Generic Object Exchange Profile (btgoep://) • irdaobex (irdaobex://). *Not supported:* Push Registry	**Note:** Only supported in *JP-5*, *JP-6* and *JP-7* See also "Bluetooth API (JSR 82)" on page 20
Java ME Web Services (JSR 172)	Version 1.0 *Supported packages*: • XML parsing • XML Web services	**Note:** Only supported in *JP-6 and JP-7*
Java IR APIs	No	
Java Serial APIs	No	See also "Serial Port Communications (JP-7 only)" on page 49
OTA Recommended Practice	Yes, MIDP 2.0 compliant	
Debug interface	KDWP	
Numeric keys	Yes (0-9, *, #)	
8-way directional key with select	Yes (navigation key)	
Signed MIDlets	Yes	
TCP Sockets	Yes	
UDP Sockets	Yes	

Characteristic	Support	Comments
Java 3D	Mascot Capsule Micro3D version 3 Mobile 3D Graphics API for J2ME (JSR 184)	See also “3D APIs” on page 19 **Note: Not** supported in *JP-2* and Z310
NokiaUI API	Version 1.1	
ARM® Jazelle® technology support	Yes	*JP-2*: No
Multitasking VM	*JP-7*: Yes	

Camera specifications

Note: In Z310, the camera is not available to Java APIs.

In the K550, K600, K610, K750, K790, K800, K810, S700, W610, W700, W800, W810, W830, W850, W880 and W900 the native camera application is designed for taking pictures with the phone in horizontal position. When a snapshot is taken in a Java application, the image is automatically rotated to match the image seen in the Java viewfinder.

Note: In the K600 and V600 series, only the video call camera is available for Java applications via `Manager.createPlayer("capture://video")`

Supported image types for a phone are obtained by calling `System.getProperty("video.snapshot.encodings")`

The supported image types can be used in conjunction with the image sizes defined in the table below by specifying a snapshot parameter string. For example:
```
videoControl.getSnapshot("encoding=jpeg&width=640&height=480");
```

Java applications are restricted to use the values listed in the table below, even if the camera itself supports other image sizes.

Phone/series	Image size (pixels)										
	60 x 100	120 x 160	240 x 320	352 x 288	480 x 640	960 x 1280	1024 x 1280	1224 x 1632	1200 x 1600	1500 x 2000	1536 x 2048
Z1010, K700, K500, Z500, F500, K300		•									
S700			•								
V800, Z800	•	•			•		• [a]				
K750, W800, W700		•			•	• [a]		• [a]			

Phone/series	Image size (pixels)										
	60 x 100	120 x 160	240 x 320	352 x 288	480 x 640	960 x 1280	1024 x 1280	1224 x 1632	1200 x 1600	1500 x 2000	1536 x 2048
K600				•							
V600				•							
Z520, Z525		•	•		•						
W600		•			•		• a				
W550		•			•		• a				
W900		•			•			• a			
W810		•	•		•		• a				
K790, K800, K810					•b	• a		• a		• a	• a
Z530		•	•		•						
W300		•	•		•						
K510		•	•		•	•a					
K310, K320, W200		•	•		•						
Z550, Z558		•			•		•a				
K610, W830, W850, W880					•b	•a			•a		
W710, Z710					•b	•a			•a		
K550, W610, Z610					•b	•a			•a		

a High resolution snapshots may not be possible to view in the Java application that took the picture, because of limited memory. However, it is possible to take a snapshot in the application and then process the created image object, for example, save it as a file in the file system of the phone.

b Resolution reported by `System.getProperty("video.snapshot.encoding")` in JP-7 phones. This is the only resolution that can be used in `VideoControl.getSnapshot()`. The other resolutions supported in these phones are available only through the JSR 234 interface `javax.microedition.amms.control.camera.CameraControl`.

Font sizes

A font is specified by requesting a style, size and face. In Sony Ericsson phones, the style and size attributes are supported, while the face attribute is ignored.

Note: Font attributes are only available for Java in the low-level UI (`Canvas` or `GameCanvas` objects).

Due to space restrictions, all styles are not supported for Chinese characters. Also, the SIZE_LARGE attribute gives the same size as SIZE_MEDIUM.

Font heights in pixels (including line space) are listed below:

MIDP values	**Z1010**	**K700**	**S700**	**F500 K310 K320 K500 K510 W200 W300 Z310 Z500 Z520 Z525 Z530**	**J300 K300**	**K550 K600 K610 K750 V600 V800 W550 W600 W610 W700 W710 W800 W810 Z550 Z558 Z610 Z710 Z800**	**W900**	**K790 K800 K810 W830 W850 W880**
Western characters								
SIZE_LARGE	22 px	22 px	26 px	20 px	15 px	22 px	26 px	26 px
SIZE_MEDIUM	18 px	18 px	22 px	15 px	13 px	18 px	22 px	22 px
SIZE_SMALL	15 px	15 px	18 px	13 px	9 px	15 px	18 px	18 px
Chinese characters								
SIZE_LARGE	18 px	22px	26 px	15 px	15 px	22 px	26 px	26 px
SIZE_MEDIUM	18 px	22 px	26 px	15 px	15 px	22 px	26 px	22 px
SIZE_SMALL	15 px	18 px	22 px	13 px	13 px	18 px	22 px	18 px

Note: In the SDK emulator, only Western font sizes are rendered correctly. Due to variations in phone software, for example, different language packs installed, texts may be displayed differently in the emulator than in the phone.

Key mapping

Sony Ericsson phones support the `keyPressed( )`, `keyReleased( )`, and `keyRepeated( )` event delivery methods in class `Canvas`.

Key	Constant value	MIDP key code	Game action
4-way select up	-1		UP
4-way select down	-2		DOWN
4-way select left	-3		LEFT
4-way select right	-4		RIGHT
4-way select press	-5		FIRE
*	42	KEY_STAR	GAME_C
#	35	KEY_POUND	GAME_D
0	48	KEY_NUM0	
1	49	KEY_NUM1	
2	50	KEY_NUM2	UP
3	51	KEY_NUM3	
4	52	KEY_NUM4	LEFT
5	53	KEY_NUM5	FIRE
6	54	KEY_NUM6	RIGHT
7	55	KEY_NUM7	GAME_A
8	56	KEY_NUM8	DOWN
9	57	KEY_NUM9	GAME_B
Left Selection key (Soft key). Only available in Fullscreen Canvas mode	-6		
Right Selection key (Soft key). Only available in Fullscreen Canvas mode	-7		
C key (Clear)	-8		
Back key	-11		
Special keys The following keys are special keys. For some of them only `keyPressed` is called, not `keyReleased` and `keyRepeated`. **Note** that some of these special keys might not be possible to use, since they may have some native functionality, for example, starting another application.			
Power On/Off key (JP-6 and higher **only**)	-12		

Key	Constant value	MIDP key code	Game action
Special gaming key A (W600, W550, K800, K790, W850, W830 and K810 series **only**)	-13		
Special gaming key B (W600, W550, K800, K790, W850, W830 and K810 series **only**)	-14		
Operator key [a]	**JP-2 – JP-5:** -10 **JP-6 – JP-7:** -20		
Video call key [a] (JP-6 and higher **only**)	-21		
Media player (WALKMAN™) key [a] (JP-6 and higher **only**)	-22		
Play (media) button[a] (JP-6 and higher **only**)	-23		
Camera key [a] (JP-6 and higher **only**)	-24		
Camera focus key [a] (JP-6 and higher **only**)	-25		
Camera, capture key [a] (JP-6 and higher **only**)	-26		
Lamp key [a] (JP-6 and higher **only**)	-27		
PTT (Push-To-Talk) key [a] (JP-6 and higher **only**)	-28		
Camera lense cover open [a] (JP-6 and higher **only**)	-34		
Camera lense cover close [a] (JP-6 and higher **only**)	-35		
Volume+ (Zoom+) key [a] (JP-6 and higher **only**)	-36		
Volume– (Zoom–)key [a] (JP-6 and higher **only**)	-37		

a This key is not present in all phone models and may be referred to with different names, depending on what function it is used for in the UI of the phone. The name and function of the key may also be customized for different operators.

Swivel position detection

On *JP-6* and *JP-7* jack knife phones the `Canvas.keyPressed()` event delivery method can also be used to detect changes of the swivel position:

- keyCode -32 is returned when the swivel is opened
- keyCode -33 is returned when the swivel is closed

Keycodes generated when opening or closing a clam-shell phone

On *JP-6* and *JP-7* clam-shell phones the `Canvas.keyPressed()` event delivery method can also be used to detect when the phone flip is opened or closed:

- keyCode -30 is returned when the phone flip is opened
- keyCode -31 is returned when the phone flip is closed

Simultaneous keypress support (S700, Z500, Z1010)

Two keys pressed at the same time are properly detected. More than two simultaneous key presses can in some combinations generate extra key press events. In some three key sequences, the third key is not detected.

Simultaneous keypress support (F500, J300, K300, K500, K700)

Two keys pressed at the same time are properly detected. More than two simultaneous key presses can in some combinations generate extra key press events. Pressing the navigation key in different directions and pressing other keys at the same time works with key 5 (Fire), key 7 (Game A) and key 9 (Game B). Using other keys together with the navigation key can generate false detections.

Simultaneous keypress support (*JP-4*, *JP-5*, *JP-6* and *JP-7* except Z310)

Two keys pressed at the same time are properly detected. More than two keys pressed simultaneously works in most cases but some combinations of keys may generate extra key press events. Pressing the navigation key and other keys at the same time works with key 0, key 1, key 3, key 7 (GAME A), key 9 (GAME B), key *(GAME C) and key # (GAME D).

Simultaneous keypress support (Z310)

Combinations of the 4-way select keys “up”, “down”, “left”, “right” and the “5” key (FIRE) are prioritized in the key detection mechanisms of the Z310. Pressing one or two of the 4-way select keys, alone or together with the “5” key, will always be properly detected, and return proper key codes with the triggered key events.

All other key combinations with two or more simultaneously pressed keys can not be properly detected by Java in the Z310.

Appendix B
Java programming issues

This appendix contains some programming issues of interest for developers of Java MIDlets/applications for Sony Ericsson phones.

Hints for developing MIDlets

Information specific for developing Java MIDlets for wireless devices may be found in *Applications for Mobile Information Devices*, a Sun white paper with helpful hints for application developers and user interface designers using the MIDP (http://java.sun.com/j2me/docs/pdf/midpwp.pdf). The book *MIDP 2.0 Style Guide* (Wagner, Bloch - Addison Wesley, 2003) contains practical guidelines for utilizing the features of MIDP2.0. In addition, see the FAQ section of the Sony Ericsson SDK for the Java™ ME Platform release notes for more information about application development.

Writing efficient applications

Java MIDlets run on phones with limited screen sizes, memory and processing power. Reducing the number of created and destroyed objects will reduce memory usage and at the same time improve performance by reducing the time spent by the JVM for intitialization and garbage collection of these objects.

Some recommendations for writing efficient applications:

- Make good use of static variables and avoid operations on `String` objects.
- Use the `StringBuffer` class for efficient manipulation of strings.
- Limit the use of inner classes and use an obfuscator to reduce class file size.
- Set object references to null as soon as they are no longer needed.
- Avoid unnecessary re-initialization of variables that are automatically set to 0 or null by the VM.
- Use synchronization sparingly. It is costly and is only needed in multi-threaded applications.
- Avoid loading the same image into memory more than once, since memory is consumed for each duplicate.
- Close network streams when finished with, in order to preserve resources.

Low-level MIDP user interface

An application using the low-level MIDP user interface will always have a `Canvas` object. The `Canvas` is implemented with double buffering to eliminate display flicker. The buffer is flushed when the `paint()` method returns.

Some recommendations for writing low-level UI applications:

- Only repaint the part of the `Canvas` to be changed, but always remember to paint what is requested of you to paint.

- In JP-2 – JP-5, the method `startApp()` is not only called when the midlet starts, but also when resuming after calling `pauseApp()`, for example, after the user has answered an incoming phone call. This behaviour has been changed from JP-6 and onwards, where `pauseApp()/startApp()` are no longer called in this situation. However, when the application uses a `Canvas`, `hideNotify/showNotify` are triggered on these occasions.

Memory usage

Java MIDlets/applications allocates memory in several different memory areas. Memory problems most often occur with allocation of memory for images. In this section some issues concerning memory usage are covered.

Java heap

Java applications use two kinds of heap memory, plain Java heap and LAM (Large Array Memory). The LAM is shared with other processes on the phone. Standard Java objects and vectors of Java objects are always located on the Java heap. Arrays of primitive types (`byte[] int[], float[]`) however may be put in the LAM if the plain Java heap is low on memory. Small arrays have a greater chance of ending up in the plain Java heap, while large arrays more often are stored in the LAM. Images are also sometimes placed in LAM.

The size and configuration of the Plain heap size and the LAM size varies between phone models.

The size of LAM is not included in the values reported by `Runtime.freeMemory` and `Runtime.totalMemory`.

Some simple rules to make the most of phone memory:

1. Always release memory before reallocating it:
   ```
   char [] v = new char[100];
   ... do stuff ...
   v = null; // by setting v to null the allocation below can re-use the memory.
   v = new char[200];
   ```

   The same schema goes for pictures, resources, and so on. For the phone to be able to re-use an image vector the image must first be released:
   ```
   Object o = allocateMyResource(size);
   ... do stuff ...
   o = null; // Remove the reference to the resource so that it can be resued in
   the allocation below
   o = allocateMyResource(someOtherSize);
   ```

2. Allocate objects first, then primitive arrays and images.

Note: The Java VM in *JP-7* phones supports multitasking. Even MIDlets running “in the background” may have some heap memory allocated, which in turn may influence the available amount of heap memory for MIDlets starting and running in other threads.

Video RAM areas

To assure that Java MIDlets will not run out of memory due to use of graphics, the Sony Ericsson phones covered in this document implement several memory areas for graphics. Graphics memory areas are used in the following order. If one area is full or an image to large to fit in the free space of one area, the next one is used instead.

1. One area of fast video RAM dedicated for graphics storage.
2. Another video RAM area, with somewhat slower access.
3. The general heap area of the phone is used for images when it is not possible to use the two video RAM areas.
4. Swapping of images to the phone flash memory is supported.

Hints for using video memory

The developer should always try to fit commonly used images into the fastest RAM area and use the slower areas for more seldomly used images. This is done by making the MIDlet fetch the commonly used images first and make sure that they fit into the 80 kb of fast video RAM.

To increase the chances that an image is actually loaded into fast memory, another image in that area, with at least the same size should be freed. Before allocating the new image into memory, garbage collection (`System.gc()`) should be called.

Another issue to take into consideration when designing applications to use the fastest possible video memory is fragmentation of memory. When an image, allocated between two other images in memory, is freed, only images smaller than the free area can be allocated in that area. Thus, even if the system reports enough free memory for allocation of an image, this may fail, because the free memory consists of several areas, each to small for the image.

When using very large images, another problem can arise. If an image is to large to fit into the memory dedicated for images, the `Image.createImage()` method may still succeed, because the image is stored in flash memory. However when the image is to be displayed, it does not fit in the available video memory, and can not be shown on the screen. The solution is to always estimate the image size in memory before trying to use it in a midlet. All images are stored in phone memory in a 16-bit per pixel RGB format, possibly with a 1-bit or 8-bit per pixel alpha-channel. Make sure to save all opaque images with 1-bit alpha, as they are drawn much faster on the screen.

Retrieving the IMEI number

The following command retrieves the IMEI (International Mobile Equipment Identity*)* number from Sony Ericsson phones:

```
System.getProperty("com.sonyericsson.imei")
```

This returns a string which uniquely identifies a phone, for example: “IMEI 004601-01-501762-8-01” (the exact format of the returned string may differ from the example). Each GSM phone is assigned a unique IMEI code when it is produced. See the following link for further information about IMEI: http://www.numberingplans.com/index.php?goto=guide&topic=imei.

Note: `"imei"` in the attribute must be written with lowercase letters when the command is used for Sony Ericsson phones, except for the P910 series where uppercase letters must be used instead (`"IMEI"`).

Minimizing and maximizing MIDlets

A MIDlet can request to get minimized by calling `setCurrent(null)`. A MIDlet can request to get maximized by calling `setCurrent(x)` with `x != null`.

A request to get maximized will only be granted if the previous `setCurrent()` call was a request to get minimized. Therefore a MIDlet that was minimized via the “long back” dialog can only get maximized by first calling `setCurrent(null)` and then `setCurrent(x)` with `x != null`.

When the MIDlet is minimized, a `Canvas.hideNotify()` event is raised, and when it is maximized, `Canvas.showNotify()` is raised. The `Canvas.isShown()` function can be used to query if the MIDlet is currently in maximized or minimized state.

Multitasking MIDlets

Multitasking Java™ ME was introduced with Sony Ericsson Java Platform 7 (JP-7) and allows multiple Java™ applications (MIDlets) to run concurrently within the same Virtual Machine. The implementation is backwards compatible with previous Java platforms so that all existing MIDlets work on the new platform without adjustments. The implementation is fully compliant with MIDP2/JTWI specifications and does not require any additional JAD properties or proprietary APIs. On earlier Java Platforms, only one Java application was allowed running together with other phone applications.

The resource contention strategy for the multitasking environment is simple. Prioritization in most cases follows the pattern of "first come - first served". For example, Bluetooth connections, sockets, memory resources, and so on, are taken by the application/thread doing the first allocation. Exceptions to this pattern are sounds, screen and user input through keyboard. The rules applied are basically the same as when MIDlets compete with native phone applications for resources in a traditional single tasking Java platform, or when threads within the same MIDlet compete for resources.

To programmatically control MIDlets running in the multitasking environment, the `setCurrent()`, `hideNotify()`, `showNotify()` and `isShown()` methods may be used as outlined above.

The phone user can also select which application to run in the foreground and which to run in the background via the phone MMI:

- Pressing the back button for ~1 second (referred to as "long back") and then selecting "Minimize" in the popup window, puts the foreground application into background

- Pressing the "Activity Menu" button (if available) and then selecting an application running in the background, or starting a new application through the menu system, puts the foreground application into background, and the selected application into foreground.

Standby MIDlets

JP-7 phones have a feature allowing developers to enable a MIDlet as a standby application. Just as the end user can assign a picture as wallpaper, it is also possible to select a Java application for this purpose. A standby MIDlet is handled by the application manager, and is started when the phone enters standby mode. It is stopped when the user selects another wallpaper, theme or picture. A MIDlet is designated as a standby application via a JAD attribute setting.

Note: Standby MIDlets are not supported in early K610, K790 and K800 phones.

For details about creating standby MIDlets and some practical advice on how to design them, see http://developer.sonyericsson.com/site/global/techsupport/tipstrickscode/java/p_standby_midlet_jp7phones.jsp.

Autostarting MIDlets

JP-7 phones, except early K610, K790 and K800 phones, support autostarting MIDlets.

The autostart feature uses the MIDP push registry as its driver. To register an application for autostart, simply do a push registration, either static or dynamic, using the push URI "autostart://:". The application will then start automatically the next time the phone boots.

For more information about how to create autostarting MIDlets and some code samples, see the article on the subject in the Tips, Tricks & Code section on Sony Ericsson Developer World.

Network APIs

Sony Ericsson phones support several network connections:

- HTTP connection
- HTTPS connection (TLS 1.0 is also supported)
  Note: HTTPS connections via Proxy are **only** supported in *JP-5*, *JP-6* and *JP-7*
- Push Registry
- TLS 1.0/SSL 3.0 connections
- Socket connections
- UDP connections (datagram).

The following table lists the Network API features and classes of the `javax.microedition.io` package, and their MIDP 2.0 support in Sony Ericsson phones.

Feature/Class	Supported
`Connector` class	Yes
All Fields, methods, and inherited methods for the `Connector` class	Yes
Mode parameter for the `Connector.open()` method	No
The timeouts parameter for the `Connector.open()` method	No
`HttpConnection` interface	Yes
`HttpsConnection` interface	Yes
`SecureConnection` interface	Yes
`SecurityInfo` interface	Yes
`ServerSocketConnection` interface	Yes
`UDPDatagramConnection` interface	Yes
`PushRegistry` class	Yes
`CommConnection` interface	**JP-2 – JP-6:** No **JP-7:** Yes. See also “Serial Port Communications (JP-7 only)” on page 49
Dynamic DNS allocation through DHCP	Yes

Secure sockets and HTTPS connections

HTTPS is supported only for certificates installed on the phone. The following X.509 root certificates for TSL/SSL server authentication are provided by default. However, operators can change which of them are installed and also add other certificates. Local market variations may also exist.

Certificate issuer	Label
Verisign	Verisign Class 3 CA
Baltimore	GTE Cyber Trust Root
Entrust	Entrust.net Root Certificate
GlobalSign	GlobalSign Root CA
Thawte	Thawte Server CA
RSA data Security	-

When initiating a connection and the certificate can not be validated in JP-2 to JP-5 phones, the connection fails and an exception is thrown. From JP-6, the user is prompted whether to accept the connection or not. However, the behaviour in JP-2 to JP-5 phones can be avoided by installing a certificate granting secure connections on the phone (a self-signed certificate can be used).

JAD/manifest attributes

The application descriptor **must** contain the following attributes:

- MIDlet-Name
- MIDlet-Version
- MIDlet-Vendor
- MIDlet-Jar-URL
- MIDlet-Jar-Size.

The application descriptor **may** contain:

- MIDlet-<n> for each MIDlet
- MIDlet-icon (for the ideal look and feel icon size **16x16 pixels is recommended**) This attribute is only supported in *JP-4*, *JP-5*, *JP-6* and *JP-7*
- MicroEdition-Profile (**recommended**)
- MicroEdition-Configuration (**recommended**)
- MIDlet-Description
- MIDlet-Data-Size
- MIDlet-Permissions (**recommended**)
- MIDlet-Permissions-Opt
- MIDlet-Push-<n>
- MIDlet-Install-Notify
- MIDlet-Delete-Notify
- MIDlet-Delete-Confirm

- MIDlet-Certificate-<X>-<Y>
- MIDlet-Jar-RSA-SHA1
- Any application-specific attributes that do not begin with MIDlet- or MicroEdition-.

Vodafone JAD attributes

Sony Ericsson phones manufactured for Vodafone, support the following additional attributes in the JAD/ manifest:

Attribute	Comments
MIDxlet-Resident	Supported values: *Y = Resident MIDlet*, *N = Not resident MIDlet*. The attribute value *S = Stay resident* is **not** supported
MIDxlet-ScreenSize/ MIDxlet-Application-Range	Values: *W,H* or *Wmin-Wmax,Hmin-Hmax*. Screen size or minimum/maximum width and height expected by the application. Both attributes are supported, but MIDxlet-ScreenSize is recommended. If both attributes are present in a JAD file, MIDxlet-ScreenSize attribute has precedence

Serial Port Communications (JP-7 only)

Java Platform 7 (JP-7) introduces support for serial port communication, defined as optional within the MIDP 2.0 specification.

The interface `CommConnection` extends `StreamConnection` to provide a means to access a serial port.

The port is accessed using a Generic Connection Framework string:

```
comm:<port identifier>[<optional parameters>]
```

The port identifier is one of the exposed serial ports which can be queried through the `microedition.commport` system property. A comma separated list of ports is returned.

```
String port1;
     String ports = System.getProperty("microedition.commports");
     int comma = ports.indexOf(',');
     if (comma > 0) {
          // Parse the first port from the available ports list.
          port1 = ports.substring(0, comma);
     } else {
          // Only one serial port available.
          port1 =ports;
     }
```

To use serial communications, the relevant AT command must first be issued from the host. Both Bluetooth and USB connection mechanisms are supported.

Once the phone is connected to the host, a COM port is assigned on the host side. Characters sent via this connection are, by default, interpreted as AT commands by the phone.

The AT*SEJCOMM command can be used to create a virtual port accessible from the Java platform.

Command	**Responses**
AT*SEJCOMM=<port>[,<persistent>]	CONNECT OK
	ERROR +CME ERROR <err>

The AT*SEJCOMM command puts the port into a so called “transparent mode” where the AT channel stops intercepting input, and subsequent characters appear as input on the serial port. The command expects <port> and optional <persistent> parameters.

The <port> parameter is used to specify a virtual port number which creates a binding to the physically connected port. For example, if the phone has been connected to the host and is using COM 4, the command AT*SEJCOMM=1 will instruct the phone to create a virtual port called “AT1” and connect it to COM 4. If the command is successful, "CONNECT" is returned and the AT channel enters transparent mode.

Depending on the <persistent> parameter, once the MIDlet closes or is terminated, the AT channel leaves transparent mode and the virtual port is destroyed. If the persistent flag is set with the value 1, the port remains until the bearer (for example, a USB cable) is disconnected.

The virtual ports are accessible to the Java platform in the form of “AT<port>”.

```
CommConnection cc = (CommConnection)
               Connector.open("comm:AT1");
     int baudrate = cc.getBaudRate();
     InputStream is  = cc.openInputStream();
     OutputStream os = cc.openOutputStream();
     int ch = 0;
     while(ch != 'Z') {
          os.write(ch);
          ch = is.read();
          ch++;
     }
     is.close();
     os.close();
     cc.close();
```

JSR 75 implementation

JSR 75 is only implemented in *JP-5*, *JP-6* and *JP-7*. This section specifies which features are supported in these phones.

PIM API

The PIM API supports the following PIM lists:

- Contacts (ContactList)
- Calendar (EventList)
- Tasks (ToDoList).

Actual names of lists and other labels depend on locale.

Contacts

Supported Java PIM fields (native/GUI field names in parenthesis):

- UID (LUID)
- NAME (LastName/Name). Supported array elements:
  - NAME_FAMILY
  - NAME_GIVEN
- ADDR (HomeAddress). Only one address, always ATTR_HOME. Supported array elements:
  - ADDR_STREET (Street)
  - ADDR_LOCALITY (City)
  - ADDR_REGION (State)
  - ADDR_POSTALCODE (Zip code)
  - ADDR_COUNTRY (Country)
- TITLE
- ORG (Company)
- EMAIL
- URL
- NOTE (Freetext)
- TEL, supported attributes (max one + ATTR_PREFERRED):
  - ATTR_HOME (HomeNumber)
  - ATTR_WORK (WorkNumber)
  - ATTR_MOBILE (CellNumber/Mobile number)
  - ATTR_FAX (FaxNumber)
  - ATTR_OTHER (OtherNumber)
  - ATTR_PREFERRED (DefaultNbr), on one number only
  - Numbers are sent to different database containers based on attributes. If two numbers have the same attribute only one is stored. One number with multiple attributes creates copies in different containers (not combined on retrieval). No attribute is treated like ATTR_OTHER. As a consequence of all this, field value indexes are not preserved on retrieval
  - Supported char-set: '0'-'9','*','#','?','+' and 'p'
- PHOTO_URL
  - Files that have no Java mapping are not returned on read, for example, predefined images that link to system directories
  - Only local URLs ('file:///') that refer to existing files can be persisted

- FORMATTED_NAME (LastName/Name)
  - Avoid using this field, since it competes with NAME for the same DB container
  - To commit, first delete NAME.

Unsupported Java standard fields

- BIRTHDAY
- CLASS
- FORMATTED_ADDR
- NICKNAME (Requires vCard 3.0)
- PHOTO
- PUBLIC_KEY
- PUBLIC_KEY_STRING
- REVISION.

Unsupported native fields

- Birthday
- ChangeCounter
- ContactPosition
- WVID (Presence ID)
- NameVoiceTag (Voice Commands)
- PersonalRing (Ringtone)
- JapaneseReading (Furigana).

Restrictions

- All fields except TEL can have one value only
- Categories are not supported
- A maximum of 1000 contacts (2500 phone numbers) can be saved in the phone.

Calendar

Supported Java PIM fields (native/GUI field names in parenthesis):

- UID (LUID)
- SUMMARY (Summary/Description)
- LOCATION (Location)
- NOTE (Description)
- END (EndDateAndTime)
  Default: current time + 1 second
- START (StartDateAndTime)
  Default: current time
- ALARM (ReminderDateAndTime)
  Must be positive, that is, before start
- CLASS (Class)
- REVISION (LastModified).

Unsupported native fields

- TimeZone
- DaylightSaving
- AllDayEvent.

Restrictions

- Database must have: ALARM <= START <= END (defaults set on commit)
- RepeatRules are not supported (Recurrence). Only the first item in a recurrence series is retrieved
- Categories are not supported

- A maximum of 300 calendar events can be saved in the phone.

Tasks

Supported Java PIM fields (native/GUI field names in parenthesis):

- UID (LUID)
- SUMMARY (Summary/Description)
- NOTE (Description)
- DUE (RemainderDateAndTime/Reminder)
- COMPLETION_DATE (CompletedDateAndTime)
- COMPLETED (Status). Native system currently uses:
  - CAL_STATUS_NOT_STARTED_VALUE (0), mapped to false
  - CAL_STATUS_IN_PROGRESS_VALUE (1), equated to not started
  - CAL_STATUS_COMPLETED_VALUE (2), mapped to true
- PRIORITY (Priority). Native system currently uses:
  - CAL_PRIORITY_HIGH_VALUE (1)
  - CAL_PRIORITY_NORMAL_VALUE (2)
  - CAL_PRIORITY_LOW_VALUE (3)
  - Only 8 bits are persisted (not sign extended on retrieval)
- CLASS (Class)
- REVISION (LastModified).

Unsupported native fields

- TimeZone
- DaylightSaving
- DueDateAndTime.

Restrictions

- Categories are not supported
- A maximum of 80 tasks can be saved in the phone.

Serialization

Serialization includes converting vCards and vCalendar events/todos in serial (text) form into PIM items (FromSerial), and back again (ToSerial).

There are two parsers, one for vCards and one for vCalendar Events/ToDos. As required by the standard, the parsers support vCard 2.1 and vCalendar 1.0, with Quoted-Printable and BASE64 encoding formats. The character encoding must be UTF-8, which means that normal 7-bit ASCII is also allowed (since it is a subset of UTF-8).

Only values/properties supported by the databases are copied to the PIM item.

If there are too many values for a particular field, the implementation will favour those with attributes that the field supports.

File Connection API

This section specifies the File Connection API support in *JP-5*, *JP-6* and *JP-7*.

The folders listed below and their content (including sub-folders), which are available via the File Manager application in the phone, are available via the FileConnection API as directories (folders) and files. This includes also access to the whole file system on an external memory (removable media), if present.

- <file:///c:/>
- <file:///c:/other>
- <file:///c:/pictures/>
- <file:///c:/sounds/>
- <file:///c:/videos/>
- <file:///e:/> (memory card)
- <file:///e:/dcim/> (camera pictures folder on memory card).

Note: The folders *Games*, *Themes*, *Applications* and *Webpage* are not available via the Java File Connection API.

Note: Which folders are accessible via the File Connection API may differ between different phone models. For example, in JP-7 phones, there is a *Camera* folder in phone internal memory, <file:///c:/camera>, and the *Themes* and *Webpage* folders on memory card are accessible.
The PDAPDemo application supplied with the Sony Ericsson SDK for the Java ME platform is recommended to find out exactly which folders are accessible in internal memory and installed memory card of a specific phone.

To query the location of, for example, the default camera folder, the recommended approach is to use the system property "`fileconn.dir.photos`". See "JSR 75 system properties" on page 60 for details on how to query default locations of folders in the file system.

Attempts to access other file areas than the ones specified above, result in a `java.lang.SecurityException` being thrown to the Java application.

The File Connection API supports the same file/dir attributes as are supported by the built-in File manager application. File and directory names accessed via the File Connection API are case-insensitive.

The length of a file path is limited by the native file system (including the memory card file structure).

Note: The Java path is mapped to a native path. The maximum native path is 120 characters.

Restricted file/directory operations

The following operations fail if they are performed on any of the built-in roots:

- Create new file in the root directory
- Create new directory in the root directory
- Change attributes
- Delete root or built in directory
- Rename root or built in directory
- Request of last modification date returns 0.

Rules for operations on DRM protected files

The following operations are supported on DRM protected files:

- Open connection
- List (DRM protected files appear in directory lists)
- Request file size
- Request attributes and last modification date
- Delete
- Exists

- Is directory.

The following operations are not supported on DRM protected files:

- Create file
- Change attributes
- Rename file
- Truncate file
- Open input stream
- Open output stream.

Rules for operations on Sony Ericsson encrypted files

Sony Ericsson encrypted files are files that are encrypted and stored in phone memory or on the memory card. These files are not accessible for the user.

The following operations are supported on encrypted files:

- Open connection
- list (encrypted files will appear in directory lists)
- exist
- file size
- can read
- can write
- is hidden
- lastModified
- dirSize (encrypted files are counted).

The following operations are **not** supported on encrypted files:

- create
- setReadable
- setWritable
- setHidden
- delete
- rename
- truncate
- openInputStream
- openOutputstream
- read / write.

Playing media files with MMAPI using progressive download

The File Connection API implementation on Java platforms JP-6 and JP-7 allows progressive download of media files to be played via the MMAPI. This allows the player to start playing the media file before the whole file actually has been loaded into memory.

To make use of progressive download in a player application, the `createPlayer` method must be invoked with a file scheme locator string as parameter, for example:

```
Manager.createPlayer(file:///c:/sounds/song.mp3);
```

Note: This functionality is **not** implemented on Java platform JP-5, where the MMAPI implementation does not support the file scheme in the `createPlayer` method. In JP-5 phones, playing of media is invoked via `createPlayer(InputStream stream, String type)` which does not take advantage of progressive download. The consequence is that the whole media file must be loaded into memory before the player starts playing. This can in some cases take quite long time with large media files.

Note: In *JP-7* phones, progressive download according to 3GPP TS 26.234 45.3.0 is supported both for http download and `InputStream` via JSR 135.

Video overlay

On Java Platform JP-7 it is easy to create an overlay over a video clip shown in a MMAPI video player instance. All pixels drawn with `Canvas.paint` are overlayed over the video, canvas areas where nothing has been drawn remain transparent. Note that if a filled shape is drawn, for example, a rectangle with a background color, the video will be completely hidden behind the shape.

Before using the overlay technique, a `javax.microedition.media.control.VideoControl` has to be initiated on the current `javax.microedition.media.Player`, for example:

```
VideoControl videoControl = (VideoControl)player.getControl("VideoControl");
videoControl.initDisplayMode(VideoControl.USE_DIRECT_VIDEO | (overlay << 8),
canvas);
```

where `overlay` is set to 1 for overlay mode, 0 for no overlay.

Video rotation/mirroring

Note: The rotation/mirroring mode described here is not applicable for GIF animations.

A `VideoControl` can be used to mirror and/or rotate the video on the display. A `javax.microedition.media.control.VideoControl` is initiated on the current `javax.microedition.media.Player`, for example:

```
VideoControl videoControl = (VideoControl)player.getControl("VideoControl");
videoControl.initDisplayMode(VideoControl.USE_DIRECT_VIDEO | (orientation <<
4), canvas);
```

`Orientation` is set to one of the following integer constants in the `javax.microedition.lcdui.Sprite` class:

Orientation constant	Effect
`TRANS_MIRROR`	Causes the sprite to appear reflected about its vertical center
`TRANS_MIRROR_ROT180`	Causes the Sprite to appear reflected about its vertical center and then rotated clockwise by 180 degrees
`TRANS_MIRROR_ROT270`	Causes the Sprite to appear reflected about its vertical center and then rotated clockwise by 270 degrees

Orientation constant	Effect
TRANS_MIRROR_ROT90	Causes the Sprite to appear reflected about its vertical center and then rotated clockwise by 90 degrees
TRANS_NONE	No transform is applied to the Sprite
TRANS_ROT180	Causes the Sprite to appear rotated clockwise by 180 degrees
TRANS_ROT270	Causes the Sprite to appear rotated clockwise by 270 degrees
TRANS_ROT90	Causes the Sprite to appear rotated clockwise by 90 degrees

Rotation can only be used in `VideoControl.USE_DIRECT_VIDEO` mode, using `VideoControl.USE_GUI_PRIMITIVE` throws an exception.

Note that `VideoControl.getSourceWidth()` returns the width of the non-rotated object, it is up to the MIDlet to keep track on what is height and what is width of the rotated video. Furthermore, the coordinates (0,0) always refers to the top left corner of the video, regardless of rotation.

Note: The rotation/mirroring modes described here are not applicable for GIF animations.

Tips for using the JSR 82

Local device

To find out what is supported by the phone, use `LocalDevice.getProperty()`. See JavaDoc for valid properties.

Device discovery

Tip 1

Filter found RemoteDevices immediately by using `DeviceClass`. By doing this unnessecary actions can be avoided, for example, doing a service search on a discovered PC when running a Java ME game.

Tip 2

If retrieving cached remote devices via JSR-82 API, then the information about remote device class is not available. It might be better to implement cache with filtered devices from the initial device discovery.

Tip 3

Only ask for friendly names for the devices displayed in GUI, and save time.

`remoteDevice.getFriendlyName(true)` is supported in *JP-5*, *JP-6* and *JP-7*.

Tip 4

To gain better user experience, present discovered remote devices directly when found. Do not wait until the inquiry is completed.

More device discovery tips

On Sony Ericsson Developer world, an article named “Bluetooth probe for mobile phones supporting JSR 82” gives some useful tips and code examples for device discovery: http://developer.sonyericsson.com/site/global/techsupport/tipstrickscode/java/p_bluetooth_probe_jsr82.jsp

Games

Use `ByteArrayOutputStream/ByteArrayInputStream` buffer for RFCOMM.

The transfer rate may be increased by using fixed size byte array, that is, by not having to send the buffer length before sending the actual byte buffer.

Managing connections between Bluetooth SDP records and a game server

On Sony Ericsson Developer world, an article can be found, giving tips and code samples on how to handle SDP records correctly, avoiding problems with rejected Bluetooth connections, for example, when clients in one or more phones tries to connect to a game server in another phone. For details, see http://developer.sonyericsson.com/site/global/techsupport/tipstrickscode/java/p_advice_bluetooth_sdp_game+server.jsp

Querying system properties

Calls to the Java platform to find out which system properties are supported in a phone can be made on different levels, for example, what classes are supported in the phone or what properties are supported by a specific class.

Supported classes

To check if a phone supports a specific class, the `Class.forName()` function can be used.

```
try{
     Class.forName("...");
}
catch(Exception ex){
```

```
    System.out.println("No support for .....")
}
```

Examples:

```
Class.forName("javax.microedition.media.Manager"); //JSR135
Class.forName("com.nokia.mid.ui.DeviceControl");  // Nokia UI extension
Class.forName("javax.bluetooth.LocalDevice"); //JSR82
Class.forName("javax.wireless.messaging.MessageConnection" );//JSR120
Class.forName("javax.microedition.pim.PIM"); //JSR75
Class.forName("javax.microedition.m3g.Graphics3D"); //JSR184
Class.forName("com.mascotcapsule.micro3d.v3.Graphics3D"); //Mascotcapsule
```

System.getProperty(String Key) calls

`Java.lang.System.getProperty(String Key)` calls are used to find out what is supported in the phone.

Example:

```
import    java.lang.*;

String    value;
String    key       = "microedition.pim.version";

value     = System.getProperty( key );
...
```

Standard system properties

The following are examples of standard properties that can be retrieved with the `System.getProperty()` call:

```
microedition.configuration
microedition.profiles
microedition.encoding
microedition.locale
microedition.platform
microedition.jtwi.version //JSR 185
```

Sony Ericsson specific system properties

```
com.sonyericsson.imei
com.sonyericsson.jackknifeopen
com.sonyericsson.flipopen
com.sonyericsson.java.platform
```

`System.getProperty("com.sonyericsson.jackknifeopen")` is only supported for phones on *JP-6 and JP-7*, and returns one of the following values:
0 = swivel closed

1 = swivel open
-1 = the phone does not have jack knife form factor.
Null is returned for phones on platforms *JP-2* - *JP-5*.

`System.getProperty("com.sonyericsson.flipopen")` is only supported for clam-shell phones on *JP-6 and JP-7*, and returns one of the following values:
0 = flip closed
1 = flip open
-1 = the phone does not have clam-shell form factor.
Null is returned for phones on platforms *JP-2* - *JP-5*.

`System.getProperty("com.sonyericsson.java.platform")` is supported on *JP-7* and returns the java platform for the phone, for example, “JP-7.1” is returned for a W850 phone.

JSR 120 system properties

To find out if the API is implemented:

```
System.getProperty("wireless.messaging.sms.smsc")
```

JSR 75 system properties

To find out what versions of the JSR 75 APIs are implemented in the phone:

```
System.getProperty("microedition.io.file.FileConnection.version")
System.getProperty("microedition.pim.version")
```

The following file connection API properties are URLs of default storage directories in the phone, retrieved with the `System.getProperty()` call:

```
fileconn.dir.photos
fileconn.dir.videos
fileconn.dir.graphics
fileconn.dir.tones
fileconn.dir.music
fileconn.dir.recordings
fileconn.dir.private
```

Localized names of directories corresponding to the default URLs above are found in the following properties:

```
fileconn.dir.photos.name
fileconn.dir.videos.name
fileconn.dir.graphics.name
fileconn.dir.tones.name
fileconn.dir.music.name
fileconn.dir.recordings.name
fileconn.dir.private.name
```

The following call returns localized names to the roots returned by the `FileSystemRegistry.listRoots()` method. The returned names are listed in the same order as returned by this method and are separated by semicolon (;):

```
System.getProperty("fileconn.dir.roots.names")
```

Note: Property retrieval behaviour differs slightly between some early JP-6 phone models and other phones, due to changes in the fileconn property syntax. Null may be returned when using the above syntax with some early JP-6 phones. The following code could be used to provide a generic means to address this behaviour difference:

```
public String getProperty(String param)
{
     int index =  param.indexOf(".");
     String extension = param.substring(index,param.length());
     String value = System.getProperty("fileconn" + extension);
     return value != null ?  value : System.getProperty("filconn" + extension);
}
```

MMAPI system properties

The following properties can be retrieved from the MMAPI using the `System.getProperty()` call:

```
microedition.media.version
supports.mixing
supports.audio.capture
supports.video.capture
supports.recording
audio.encodings
video.encodings
video.snapshot.encodings
streamable.contents
```

To find out which protocols and content types are supported, the following calls can be made from a `Manager` class object:

```
static java.lang.String[] getSupportedContentTypes(java.lang.string protocol)
//lists supported content types for a given protocol
static java.lang.String[] getSupportedProtocols(java.lang.string content_type)
//lists supported protocols for a given content type
```

From a `Player` class object, a specific `Control` or the `Controls` collection supported by the player can be retrieved:

```
Control getControl(java.lang.String ControlType)
Control[] getControls()
```

Bluetooth Local device properties (JSR 82)

To find out what Bluetooth API properties are supported in the local device, the `LocalDevice.getProperty("...")` can be called with the following parameters:

```
bluetooth.api.version
bluetooth.master.switch
bluetooth.sd.attr.retrievable.max
bluetooth.connected.devices.max
bluetooth.l2cap.receiveMTU.max
bluetooth.sd.trans.max
bluetooth.connected.inquiry.scan
bluetooth.connected.page.scan
bluetooth.connected.inquiry
bluetooth.connected.page
```

Implementation specific properties in JSR 184

The version of the JSR 184 API is retrieved with:

```
System.getProperty("microedition.m3g.version")
```

Other JSR 184 properties can be retrieved through the `Graphics3D.getProperties("...")` with the following keys:

```
supportAntialiasing
supportTrueColor
supportDithering
supportMipmapping
supportPerspectiveCorrection
supportLocalCameraLighting
maxLights
maxViewportDimension
maxTextureDimension
maxSpriteCropDimension
maxTransformsPerVertex
maxTextureUnits
```

Knowledge base

Q: How can I detect the swivel position on jack knife phones?

A: For *JP-6* and *JP-7* phones the current swivel position is detected by calling
`System.getProperty("com.sonyericsson.jackknifeopen")`
Return values:
0 = swivel closed
1 = swivel open
-1 = the phone doesn't have jack knife form factor.
Null is returned for phones on platforms *JP-2* - *JP-5*.

Changes of the swivel position generates keyCodes and are detected by the `Canvas.Keypressed()` event method.

- keyCode -32 is returned when the swivel is opened
- keyCode -33 is returned when the swivel is closed

Q: How can I detect if a clam-shell phone is open or closed

A: For *JP-6* and *JP-7* phones the current “flip” position is detected by calling
`System.getProperty("com.sonyericsson.flipopen")`
Return values:
0 = flip closed
1 = flip open
-1 = the phone does not have clam-shell form factor.
Null is returned for phones on platforms *JP-2* - *JP-5*.

When the phone is opened or closed, keyCodes are generated and can be detected by the `Canvas.Keypressed()` event method.

- keyCode -30 is returned when the phone is opened
- keyCode -31 is returned when the phone is closed

Q: Is it possible to tell if the user is currently using landscape or portrait playing style?

A: There is no programmatic way to detect if a phone is used in portrait or landscape playing style. Since the JVM is not aware of any screen configuration changes, the following should be considered:

- The canvas `getWidth()` and `getHeight()` methods return the **same values** regardless what screen configuration is being used.
- The canvas `sizeChanged()` is **not invoked** when the playing style changes.
- Key codes remain **constant** and are not automatically inverted.
- Soft key menus **remain unchanged**, there is no way to programmatically change their orientation.

It is the responsibility of the MIDlet to supply a means to allow a user to set their screen configuration preference. The user can change between playing styles, either landscape or portrait, while the MIDlet is active. Therefore the ability to change between playing styles after initalization should be a design consideration.

Q: Are existing portrait mode MIDlets compatible with phones offering landscape playing style?

A: In keeping the Sony Ericsson Java platform strategy, phones supporting landscape playing style remain backward compatible with phones offering the same Java platform but supporting only portrait screen configuration.

Q: How can I develop for a screen in landscape playing style?

A: There are a number of different approaches to consider when working towards landscape playing style:

- **Design for square**
  This is the simplest approach and is concerned with targeting the shortest width offered by a given screen size. By assuming a common screen size of 176 x 220 pixels but designing towards176x176, a virtual square can be created that will cater for both landscape and portrait playing style and be highly portable. A drawback to this approach is the inherent "dead space". A popular solution is to mask the dead space with suitable user interface additions.

- **Generic Layout**.
  By not deciding a layout strategy and instead using the MIDP high level APIs, the application can be allowed to decide how best to cater for the screen dimensions and layout. This is generally only practical for business applications since much of fine grained control required for game oriented content is lost and high level cannot be concurrently mixed with low level.

- **Fit to window**
  By using `Canvas` to query the screen dimensions appropriately, MIDlets can scale effectively regardless of what screen dimensions are available.

  ```
  // assumes a fixed size
  g.fillRect(4,8, 248, 220);
  g.drawString("this isn't generic",10,20,0);

  // makes a decision depending on dimensisions
  g.fillRect(0, 0, getWidth() / 2, getHeight() / 2);
  g.drawString("this is generic",getWidth() / 2, 0,g.TOP | g.HCENTER);
  ```

  This is considered "best practice" but relies heavily on other artifacts positioning to work effectively, and can lead to increased development time. It can also inherently lead to a stretched appearance though this can have relatively little impact depending on application type and genre. A popular compromise is to combine the design for square approach with "fit to window" by centering the virtual square in a generic way.

- **Fit Content**
  This approach requires reimplementation to take advantage of the extra screen width and usually also involves changing the dynamics of the application substantially, such as:

  - Increased number of artifacts.
    The new space needs to be filled in a natural way – a common answer to this is to increase the number of on screen artifacts.

  - Increased horizontal clipping distance.
    The clipping area should also be adjusted to cater for the different screen configurations.

  Fitting the content to the new dimensions provides the most optimal use of the screen space. However, it requires a significant amount of additional development.

Q: How do I use the extra A and B gaming buttons?

A: Regardless of which design strategy is used, the K790, K800, K810, W550, W600, W830 and W850 series are equipped with two additional A and B buttons for improvement of the landscape gaming experience.

You can use the extra buttons via the MIDP 2 `GameCanvas GAME_A` and `GAME_B` constants or by key-codes. The key codes for the extra A and B **gaming buttons** are defined as:

- ButtonA: key code = -13
- ButtonB: key code = -14

They are used in the usual form:

```
protected void keyPressed(int keyCode) {
switch (keyCode) {
case ButtonA:
…
case ButtonB:
…
}
}
```

Q: How can I rotate my graphics to match changing playing styles?

A: The MIDP 2.0 `Sprite` class provides a convenient `setTransform` method to allow images to be easily rotated using predefined constants.

```
sprite.setTransform(Sprite.TRANS_MIRROR_ROT90);
```

Alternatively you can simply use the `Graphics drawRegion` method with the same constants, often in place of `drawImage`:

```
// the original image
g.drawImage(image, x, y, Graphics.TOP | Graphics.LEFT);
// identical to drawImage but with the image rotated.
g.drawRegion(this.image, 0, 0, image.getWidth(), image.getHeight(),
Sprite.TRANS_ROT90, x, y, g.LEFT|g.TOP);
```

When moving back and forth between landscape and portrait playing styles you will commonly use the `Sprite.TRANS_ROT90` and `Sprite.TRANS_NONE` constants.

Appendix C
Sony Ericsson SDK for the Java™ ME Platform

This appendix contains information about the Sony Ericsson SDK for the Java™ ME platform and its integration in different developer tools.

Features

The Sony Ericsson SDK for the Java™ ME Platform supports full 3D emulation, Mascot Capsule Ver. 3 and Ver. 4 (JSR 184) are supported. The developer can also take advantage of many WTK features. The SDK includes detailed documentation (JavaDoc) of supported Java APIs (CLDC, MIDP, MMAPI, WMAPI JSRs).

The SDK also supports On-Device source-level Debugging (ODD). The Sony Ericsson SDK for Java™ ME can be integrated with any UEI compliant Java IDE.

Additionally, the SDK includes useful utilities such as the Device Explorer and the ejava.exe command line tool. These provide an interface for manipulating the phone application manager. The developer can install, remove, start, stop, pause, and resume Java applications. The Device Explorer also provides an interface for displaying heap and file system statistics, requesting garbage collection to run, enabling KVM trace messages, and enabling serial network emulation.

Installing and updating the SDK

Installing

Before installing the Sony Ericsson SDK for the Java™ ME Platform, the SDK Java SE Development Kit (version 1.4 is recommended) need to be installed. If you wish to install an IDE, you may do so either before or after installing the Sony Ericsson SDK for the Java™ ME Platform. Note that an IDE is not required but is highly recommended.

Updating

The latest version of the Sony Ericsson SDK for the Java™ ME Platform is available for download at www.sonyericsson.com/developer/java. The Sony Ericsson SDK for the Java™ ME platform is required for ODD.

After updating the SDK to a newer version, it is recommended to clear all platforms in the IDE and then reselect all profiles from the latest SDK version. It has been noted that, if for example, emulator profiles are selected from version 2.2.2 and device debug profiles are selected from version 2.2.3, the following error occurs when starting device debugging:

```
org.xml.sax.g: Unexpected end of Tag
```

Selecting devices for ODD

Regardless of which IDE is used, the On Device Debugging functionality requires settings adapted to the actual phone or platform to test on. When connecting to a phone via the Connection Proxy interface, the identification for the phone is displayed in the Connection proxy window.

Phones on Java platforms up to and including JP-6 are identified by phone model, for example, “Sony Ericsson K750”.

From Java platform JP-7, phones are identified by platform, for example, “Sony Ericsson_JP-7.1”.

Integrating the Sony Ericsson SDK for the Java™ ME Platform in JBuilder 2005

In the instructions below, <SDK_Path> stands for the path where the Sony Ericsson SDK for the Java™ ME Platform is installed.

To add Sony Ericsson Emulators:

1. In the Jbuilder window, select the menu *Tools/Configure/JDKs…*
2. Choose New.
3. Click Existing JDK home path and browse to <SDK_Path>/PC_Emulation/WTK1.
4. Click OK to add the Sony Ericsson WTK1 emulators.
5. Repeat steps 2 to 4 and add the WTK2 and the On-Device Debug emulators. The paths to use in step 3 are:

   - <SDK_Path>/PC_Emulation/WTK2
   - <SDK_Path>/PC_Emulation/OnDeviceDebug.

To set which phone will be used for a Specific JDK:

1. In the Jbuilder window, select the menu Tools/Configure/JDKs…
2. Chose one of the JDK:s added above.
3. Click the Micro tab.
4. Choose the preferred phone in the Target Device drop-down list.
   Note: From Java Platform JP-7, phones are named by Java platform, for example, “SonyEricsson_JP-7.1”, instead of by phone model.

To use Sony Ericsson emulators in a project:

1. Right-click the project icon and chose Properties... or choose the menu item Project/Project Properties.
2. Select Paths in the upper left panel.
3. Click the “…” button at JDK: and select the emulator you want to use.

To use Sony Ericsson On Device Debug in a project:

1. Make sure the project has been built so that there is a valid JAD file
2. Follow the steps in **To use Sony Ericsson Emulators in a Project** above and choose the On Device Debug emulators.
3. Choose *Run/Configurations…*
4. Chose Edit on the selected configuration.
5. Select the option button JAD file, browse and select the project JAD file.

6. Select a specific phone name (JP-2 to JP-6) or a Java platform (from JP-7) for “Emulator device”.

7. Click OK.

Limitations in the JBuilder 2005

The Project Properties dialog box in JBuilder 2005 behaves in an inconvenient way when switching between different UEI compliant toolkits or between different emulated phones in the same UEI compliant toolkit.

Switching between different UEI compliant toolkits

Description: The list of emulated phones is not updated when JDK for a project is changed.
Example: When switching between “Sony Ericsson SDK 2.2.3 Java ME (Emulation)” and “Sony Ericsson SDK 2.2.3 Java ME (On-Device-Debug)” or vice versa, the *Emulated device* drop-down list is not updated.

Workaround: Close the *Project Properties* dialog, and open it again. The list of emulated phones now corresponds to the selected JDK.

Switching between different emulated phones in the same toolkit

Description: After switching, the list of emulated phones is not correct. Only phones using exactly the same set of “jar” libraries for compilation are listed.
Example: If “SonyEricsson_W550” is chosen in *Target device*, *Project Properties* shows only “SonyEricsson_W550” and “SonyEricsson_W600”.

Workaround (partial): Close the *Project Properties* dialog, open *Configure JDKs* and select the desired emulated phone in *Target Device*. Save settings and open the *Project Properties* dialog. Now the chosen phone is in the list of emulated phones.

The workaround is only partial, since it is still impossible to have two projects for phones using different sets of “jar” libraries open at the same time. For example, projects for “SonyEricsson_W550” and “SonyEricsson_K750” can not be open at the same time.

Integrating the Sony Ericsson SDK for the Java™ ME Platform in NetBeans 4

The Sony Ericsson SDK version 2.2.4 (and later versions) has been tested and found working also with **NetBeans 5.0**.

In the instructions below, <SDK_Path> stands for the path where the Sony Ericsson SDK for the Java™ ME Platform is installed.

Note: Integration of the Sony Ericsson SDK for the Java™ ME Platform in Netbeans requires NetBeans Mobility Pack add-on.

To add Sony Ericsson emulators:

1. In the Netbeans window, select the menu *Tools/Java Platform Manager* (or *File/<project name> properties...* and click *Manage Emulators...*).

2. Select *J2ME* and click *Add Platform...*

3. Browse to <SDK_Path>/PC_Emulation/WTK1.

4. Click *Finish* to add the Sony Ericsson WTK1 emulators.

5. Repeat steps 2 to 4 and add the WTK2 and the On Device Debug emulators. The paths to use in step 3 are:

   - <SDK_Path>/PC_Emulation/WTK2
   - <SDK_Path>/PC_Emulation/OnDeviceDebug.

To set the platform and phone that will be used for emulation:

1. In the Netbeans window, select the menu *File/<project name> properties...* and select *Platform* in the table to the left.

2. Use the drop-down list *Project Configuration* to choose one of the platforms added above.

3. Use the drop-down list *Device* to choose one of the emulators available on the chosen platform.

4. Click *OK.*

Note: From Java Platform JP-7, the phone names for ODD refer to Java Platforms rather than to phone models, for example, “SonyEricsson_JP-7.1”.

Note: If NetBeans complains about the missing file "zayit.dll" when trying to run your project with a Sony Ericsson emulator, you should reboot the computer and try again.

To use Sony Ericsson On Device Debug in a project:

In the Netbeans window, select the menu *Run/Debug Main Project*, or press *F5*. Note that you must choose *Debug Main Project*, **not** *Run Main Project*.

Integrating the Sony Ericsson SDK for the Java™ ME Platform in Eclipse

The Device Explorer plugin elaborates upon the existing Device Explorer tool by allowing it to be integrated with any Eclipse derived IDE. All features found in the standalone Device Explorer tool are encapsulated in a new view plane.

The Device Explorer concept provides the means to interact with the application manager found on all Java enabled phones.

The current list of features offered by Device Explorer plugin include:

- Listing of all installed MIDlets
- Starting, stopping, pausing and resuming execution of a MIDlet
- Deleting installed MIDlets
- Manually invoking garbage collection
- Previewing MIDlet information
- Manually deleting RMS entries
- File system browsing and MIDlet installation.

The Device Explorer is entirely connection independent, allowing you to use Bluetooth, infrared or any other means of phone to PC connection.

By using the Device Explorer plugin in conjunction with the EclipseME plugin suite, a powerful and convenient platform for Java ME based development can be built. **EclipseME v1.1.0** is recommended for use with the Sony Ericsson toolkits. **EclipseME 1.5.0** has been tested and works properly with the Sony Ericsson SDK version 2.2.4 (and later) toolkits.

More information about EclipseME can be found at http://eclipseme.org/

An extensive guide for installation and tuning of EclipseME can be found at http://eclipseme.org/docs/configuring.html.

Prerequisites

The Device Explorer plugin is tested and offered for use with **Eclipse 3.0.x** and **Eclipse 3.1.x**. Also the release candidate version **3.2RC7** has been tested and works with the Device Explorer.

You can obtain Eclipse from http://www.eclipse.org/downloads/index.php.

Acquiring the software

The Device Explorer plugin is offered in two packaged formats - it is currently not offered via the Eclipse automated package manager service.

The latest version of the plugin can be downloaded from Sony Ericsson Developer World. The plugin also comes prebundled in the Sony Ericsson SDK for the Java™ ME Platform and can be found within your installation directory, for example, *<drive>/J2ME_SDK/OnDeviceDebug/lib/devexp/plugins/com.sonyericsson.sdkme.deviceexplorer_<your SDK version>.* However, the prebundled version may be outdated, why it is highly recommended to look for updates on Developer World.

Installation

Before installing the Device Explorer you must stop any running instances of your Eclipse derived IDE. If you have acquired the plugin from the Sony Ericsson Developer World site you should proceed to extract the Zip file into your Eclipse plugin directory (*<drive>/eclipse/plugins*). If you wish to use the version supplied with the Sony Ericsson SDK you can proceed to simply copy the *com.SonyEricsson.sdkme.deviceexplorer_<your SDK version>* directory into your Eclipse plugin folder.

Using the Device Explorer plugin

Having installed the plugin, the new Device Explorer view should be available for immediate usage. Launch the Device Explorer via menu selection *Window – Show View – Other… – Sony Ericsson Device Explorer*.

When the Device Explorer launches you will see a new "Connection Proxy" window. This is a key component of the Device Explorer plugin that facilitates the connection to the Sony Ericsson phone. As with the Device Explorer tool, it is offered in a standalone form as part of the SDK. You only require one instance of the Connection Proxy to be active and it is included with the Device Explorer plugin as a convenience.

You should leave the Connection Proxy open at all times when using the Device Explorer.

Upon a successful connection, the Connection Proxy will display an image of the relevant phone and a new Device Explorer panel will be presented in Eclipse. Java applications present in the phone are listed in the panel. The relevant Device Explorer functions (Play, Stop, Pause, and so on) are available both via the toolbar buttons and by right-clicking a MIDlet in the list.

The Favorites Folder Explorer

In addition to the Device Explorer, the Favorites Folder Explorer allows you to browse the local file system and install MIDlets to the phone.

Launch the Favorites Folder Explorer via menu selection *Window – Show View – Other… – Sony Ericsson Favorites Explorer*.

When the Favorites Folder Explorer launches you will see a new "Sony Ericsson MIDlet favorites" tab, listing all local root drives.

You can browse the file system for MIDlets to install to the phone. When a JAD or JAR file is selected, the phone transfer icon is enabled allowing you to transfer the content.

Adding the PC emulator and On Device Debugger to the Eclipse workspace

Select the menu *Window->Preferences.* The “Preferences” dialog opens.

Select in the left tree: *J2ME/Device Management*

In the right panel, click the *Import* button and add the following two Wireless Toolkits:

- C:\SonyEricsson\JavaME_SDK_CLDC\OnDeviceDebug
- C:\SonyEricsson\JavaME_SDK_CLDC\PC_Emulation\WTK2.

Note: After **updating** the Sony Ericsson SDK for the Java™ ME Platform, the two directories *eclipseme.core* and *eclipseme.ui* residing in *<drive>/Program Files/eclipse/workspace/.metadata/.plugins* have to be deleted before adding the Wireless Toolkits that came with the new SDK version.

Note: When listing available platforms, EclipseME does not include the name of the wireless toolkit. To distinguish between the K700 from OnDeviceDebug and K700 from WTK2 may be difficult. In earlier versions of the SDK, the platform names are only slightly different as shown in the following example:

- In OnDeviceDebug: “Sony Ericsson K700 Platform"
- In WTK2: "SonyEricsson_K700 Platform".

From SDK 2.2.3, the EclipseME platform names are, for example:

- In OnDeviceDebug: "Sony Ericsson K700 Platform"
- In WTK2: "Sony Ericsson K700 (emulator) Platform".

Note: From Java Platform JP-7, the platform names for ODD refer to Java Platforms rather than to phone models, for example, “SonyEricsson_JP-7.1”.

Appendix D
Sony Ericsson Mobile JUnit

This appendix contains information about the Sony Ericsson Mobile JUnit test tool, and how it can be used for MIDlet testing in the emulator or on physical phones.

Mobile JUnit features

Mobile JUnit is a unit testing framework from Sony Ericsson, intended for Java ME CLDC phones, where the results of the test run is communicated to a PC. This enables simple, automated regression testing of JavaME applications. No manual uploading of MIDlets or manually inspecting test results is needed.

Mobile JUnit primarily supports Sony Ericsson phones, but most Wireless Toolkit emulated devices will also be able to run Mobile JUnit.

Prerequisites:

- A wireless toolkit, supporting UEI 1.0.1, that outputs the MIDlet `System.out` to the console. All Sony Ericsson wireless toolkits and SDK versions have this capability.
- JUnit 3.8.1 installed on the computer. The binary is included with the Mobile JUnit installation, and is automatically installed, by default in the top directory of the tool (for example: `C:\SonyEricsson\JavaME_SDK_CLDC\Mobile-JUnit\junit.jar`).
- Sony Ericsson SDK for the Java ME platform is recommended for running tests on physical devices.

Installing Mobile JUnit

Mobile JUnit can be downloaded from Sony Ericsson Developer World.

To install Mobile JUnit, the downloaded `mobile-ju-setup-1.0.exe` is run. A dialog asking for where to install Mobile JUnit appears. The default location is the same as the default location for installation of the Sony Ericsson SDK. If the SDK is installed elsewhere, that location should be entered instead.

It is recommended to install the sample project by marking the “Sample Project” and “Generate Scripts” checkboxes during the installation.

If “Generate Scripts” was selected, a dialog will pop up, asking for the path to a java compiler (`javac.exe`). Browse and select a `javac.exe`. Mobile JUnit requires a java compiler to run, and “Generate Scripts” will generate a script with this particular compiler selected.

Note: The sample project is installed into the selected application directory of the specified WTK derived SDK. For example, `C:\SonyEricsson\JavaME_SDK_CLDC\PC_Emulation\WTK2\apps\mobile-ju-sampleproject`

The default directory was selected during installation, and the following directories have been added:

Mobile JUnit Runtime	`<SDK Installation Path>\JavaME_SDK_CLDC\Mobile_JUnit`
Sample Project	`<SDK Installation Path>\JavaME_SDK_CLDC\PC_Emulation\WTK2\apps\mobile-ju-sampleproject`

The sample project test

Testing with Mobile JUnit works very much the same way as with JUnit. The source code to be tested is in the project `src` directory, and the project binary in the `bin` directory. The file defining the tests to perform is found in a separate `test` directory.

For example, when “Sample Project” was selected during the Mobile JUnit installation, the sample project `mobile-ju-sampleproject` was installed in a folder with subfolders `src`, `bin`, `build` and `test`. In the `test/src` folder, a file named `SampleTest.java` contains the test code. The code is very similar to a “regular” JUnit test:

```
import com.sonyericsson.junit.framework.TestCase;

public class SampleTest extends TestCase {

    public void testCountCharacter() {
          CharacterCounter counter = new CharacterCounter();
          assertEquals(0, counter.count('a', "Hello");
          assertEquals(1, counter.count('a', "a");
          assertEquals(0, counter.count('a', "A");
          assertEquals(2, counter.count('a', "Attackaz!");
     }
}
```

Note: All `void` methods with zero arguments starting with “`test`” are considered tests by the Mobile JUnit framework.

Running the test

To run the tests of the sample project, use one of the command lines below. Normally, Mobile JUnit expects the MIDlet to test to be compiled and present in the `bin` directory. Note the `--compile-midlet` switch, which instructs Mobile JUnit to compile and create a new MIDlet. Without this switch Mobile JUnit will assume that a MIDlet is precompiled and accessible in the bin directory.

Alternative 1 (assuming you selected “Generate Scripts” during the installation):

```
run-mobile-junit --project-dir:<SDK Installation
Path>\JavaME_SDK_CLDC\PC_Emulation\WTK2\apps\mobile-ju-sampleproject --
device:SonyEricsson_W800_Emu --compile-midlet:yes
```

Alternative 2 (replace the `--javac` parameter with the compiler that Mobile JUnit should use):

```
java -classpath mobile-ju-1.0.jar;junit.jar com.sonyericsson.sdkme.junit.OnDe-
viceTest --project-dir:<SDK Installation
Path>\JavaME_SDK_CLDC\PC_Emulation\WTK2\apps\mobile-ju-sampleproject --
device:SonyEricsson_W800_Emu --compile-midlet:yes --javac:<my-installation-of-
jdk>\bin\javac.exe
```

Note: This example assumes a WTK project structure. The tool does however support various configurations. See “Configuring and running mobile tests” on page 80 for more information.

Note: The command lines above should be run from the Mobile JUnit installation directory. For more information regarding the `--javac` parameter see “Configuring and running mobile tests” on page 80.

The tool finds, compiles, and runs tests according to the `SampleTest.java` file. The emulator starts and the tests are run:

The emulator terminates automatically, and the test results are output to the console:

```
Building midlet... 312ms
Building test midlet... 765ms
Uploading and running test midlet... 2s 937ms
..Running with storage root SonyEricsson_W800_Emu
Execution completed.
0 bytecodes executed
0 thread switches
822 classes in the system (including system classes)
0 dynamic objects allocated (0 bytes)
0 garbage collections (0 bytes collected)Done. 6s 109ms

Time: 4,101

OK (2 tests)
```

Test suites

Another concept found in JUnit, and inherited by Mobile JUnit, is test suites. In the sample project directory `Mobile-ju-sampleproject/test/src`, the `MyTestSuite.java` source file can be found:

```
import com.sonyericsson.junit.framework.TestSuite;

public class MyTestSuite extends TestSuite {
    public MyTestSuite() {
        addTestSuite(SampleTest.class);
    }
}
```

Note the following:

- The test suite should inherit from the `TestSuite` class, available in the framework.
- The test suite must have a constructor without any arguments, otherwise the test framework will not be able to create it.
- In this constructor, tests are added using the `addTestSuite` method. In this instance, the `SampleTest` from the previous chapter is added, and provides its class as the argument to `addTestSuite`. Other TestSuites may be added using this method.

To select a specific test suite or individual test, the `--suite:` parameter is used. If no such parameter is provided, the tool will try to find all test cases in the `test` directory and run them.

Alternative 1 (assuming you selected “Generate Scripts” during installation):

```
run-mobile-junit --project-dir:<SDK Installation
Path>\JavaME_SDK_CLDC\PC_Emulation\WTK2\apps\mobile-ju-sampleproject --
device:SonyEricsson_W800_Emu --suite:MyTestSuite --compile-midlet:yes
```

Alternative 2 (replace the `--javac` parameter with the compiler that Mobile JUnit should use):

```
java -classpath mobile-ju-1.0.jar;junit.jar com.sonyericsson.sdkme.junit.OnDe-
viceTest --project-dir:<SDK Installation
Path>\JavaME_SDK_CLDC\PC_Emulation\WTK2\apps\mobile-ju-sampleproject --
device:SonyEricsson_W800_Emu --suite:MyTestSuite --compile-midlet:yes --
javac:<my-installation-of-jdk>\bin\javac.exe
```

The output should be the same as in the previous example.

On-device testing on a Sony Ericsson phone

To run tests on a physical phone, the Wireless Toolkit for testing needs to be changed. If the Sony Ericsson SDK is installed at the default location `C:\SonyEricsson\JavaME_SDK_CLDC\`, the on-device Wireless Toolkit is located at `C:\SonyEricsson\JavaME_SDK_CLDC\OnDeviceDebug`. The `--wtk:` switch is set to this location. The command line is as follows (the `--device` parameter must match the connected phone):

Alternative 1 (assuming you selected “Generate Scripts” during installation):

```
-mobile-junit --project-dir:<SDK Installation
Path>\JavaME_SDK_CLDC\PC_Emulation\WTK2\apps\mobile-ju-sampleproject --
device:SonyEricsson_W800 --suite:MyTestSuite --compile-midlet:yes --wtk:<SDK
Installation Path>\JavaME_SDK_CLDC\OnDeviceDebug
```

Alternative 2 (replace the `--javac` parameter with the compiler that Mobile JUnit should use):

```
java -classpath mobile-ju-1.0.jar;junit.jar com.sonyericsson.sdkme.junit.OnDe-
viceTest --project-dir:<SDK Installation
Path>\JavaME_SDK_CLDC\PC_Emulation\WTK2\apps\mobile-ju-sampleproject --
device:SonyEricsson_W800 --suite:MyTestSuite --compile-midlet:yes --wtk:<SDK
Installation Path>\JavaME_SDK_CLDC\OnDeviceDebug --javac:<my-installation-of-
jdk>\bin\javac.exe
```

Before executing, the Serial Proxy application must be launched with the command:

```
<SDK Installation Path>\JavaME_SDK_CLDC\OnDeviceDebug\bin\serialproxy.exe
```

The Serial Proxy connects to the phone, and a dialog as below appears. If it fails to connect, click the *Settings* button (red circle) and select the serial port to which the phone is connected.

Once connected, the tests can be run by executing the above command line. The test results are output to the phone display.

Configuring and running mobile tests

There are various ways to configure Mobile JUnit. For most situations the default configuration should work. Since there are so many ways to structure a project, the tool will allow configuring most of its tasks and where to put things.

--project-dir

This is used if the project follows the Wireless Toolkit project structure. Source directories, the name of the MIDlet under test, and so on, are set automatically. However, each of these configuration settings may be changed individually.

The `--project-dir` should point at a subdirectory of the `apps` directory in the Wireless Toolkit.

Example:

```
--project-dir:C:\SonyEricsson\JavaME_SDK_CLDC\PC_Emulation\WTK2\apps\Mobile-ju-sampleproject
```

--device

Note: mandatory property.

This is the name of the phone to emulate. Most Java ME enabled IDEs can provide a list of all available phone names. Examples of device names are `SonyEricsson_JP-7` and `SonyEricsson_K750`.

--wtk

The location of the Wireless Toolkit to use for emulation or on-device execution of the tests. The Sony Ericsson SDK provides an on-device Wireless Toolkit.

If no `--wtk` is set, the tool will use the `--project-dir` setting to “guess” it.

Example:

```
--wtk:C:\SonyEricsson\JavaME_SDK_CLDC\OnDeviceDebug
```

--runmode

In some instances it may be useful to only perform the compilation of the test MIDlet, or to only run the test MIDlet without compiling it. The `--runmode` switch enables this.

- `--runmode:COMPILE-ONLY` – the test is only compiled into a test MIDlet
- `--runmode:RUN-ONLY` – the compiled test MIDlet is run (if the other configuration settings are kept from the compilation step)
- `--runmode:COMPILE-AND-RUN` – both steps are performed.

The default value is `COMPILE-AND-RUN`.

--suite

The class name of the test to run, usually a test suite, but it may also be an individual test case.

If no suite is provided, the `--test-source` directory is scanned, and any class that "`extends com.sonyericsson.junit.framework.TestCase`" is executed.

Example:

```
--suite:com.mycompany.myproduct.AllTests
```

--javac

The location of the java compiler executable, `javac.exe`. (Mobile JUnit tries to set this to a default value, but this may not be correct.)

Example:

```
--javac:C:\java\j2sdk1.4.2_08\bin\javac.exe
```

--compile-midlet

If set to `true`, `on` or `yes`, this setting instructs Mobile JUnit to build the MIDlet under test. A particularly nice feature for library development, where no MIDlet is present (this may cause problems with Wireless Toolkit). It also compiles the MIDlet with line number information, which allows for using the line coverage feature.

--name

The name of the test run. This name shows up in the xml report.

--add-line-numbers

This is a special feature for devices that do not provide a stack trace with line number information. If set to `true`, `on` or `yes`, the `--add-line-number` setting adds the line number to every assert in the set of tests.

Note: In the current implementation, the line number will not appear in the stack trace, but in the assert message.

--test-source, --test-resources, --midlet-source, --midlet-resources, --midlet-under-test, --midlet-manifest

These are directories and files the tool uses for compiling the test MIDlet. They can be configured to fit the project structure.

--test-report

This is a file where the XML used internally by the tool is stored. This may be used for reporting. However, it is recommended to wrap the test run in a regular JUnit test run and use tools that are available for JUnit to do this.

--coverage, --coverage-map, --coverage-report, --coverage-xsl

Mobile JUnit provides a simple coverage facility for line (statement) and method coverage.

If set to `L`, Mobile JUnit prepares the test MIDlet for line (statement) coverage.

If set to `M`, Mobile JUnit prepares the test MIDlet for method coverage.

Examples:

```
--coverage:L
```

```
--coverage:M
```

```
--coverage:off
```

There is also an example how to use the coverage functionality in the `build.xml` file of the sample project, target `run-with-coverage`.

`--coverage-map` points to a file that contains an internal representation of the source code.

`--coverage-report` is where the report is output.

`--coverage-xsl` is an XSL file used to produce the HTML report.

Use the `--name` parameter to set the HTML report title.

--progress

Before actually running the tests, Mobile JUnit compiles them and upload them to the (emulated) device. Progress is reported to the user. There are three levels of progress: `TEXT`, `GUI` and `NONE`. The `TEXT` progress reports progress to the console, `GUI` launches a small window to the same effect and `NONE` does not report progress at all.

Examples:

```
--progress:TEXT

--progress:GUI

--progress:NONE
```

Note: The `GUI` option launches an AWT window. Due to AWT threading issues, only a call to `System.exit` actually kills the VM. So, if the on-device tests are wrapped in a JUnit test run (see "Using JUnit to run mobile tests" on page 86), the client code must ensure that the call to `System.exit` is made.

--print-config

Setting this switch to true will output all configuration settings to the console. This may be useful when problems running Mobile JUnit arise.

Example:

```
--print-config:true
```

--config-file

Instructs Mobile JUnit to load a file containing configuration settings. If a particular setting is defined on the command line and in the file, the command line value will take precedence. The file is has the format of a Java properties file.

Example:

```
--config-file:C:/projects/my-project/test.properties
```

Example:

```
wtk = C:\\SonyEricsson\\JavaME_SDK_CLDC\\OnDeviceDebug\\bin
```

Note that parameters have no leading `--`, and that `\` must be escaped in Java properties files.

Default values

Below are default values for each of the above settings and a few more, auxiliary settings. Any one of these settings may be overidden, but it must be made sure that this does not cause some other setting to be undefined or invalid.

The `midlet-name` property is somewhat special, since its default value is set by Mobile JUnit. This is due to the fact that the Wireless Toolkit uses a special naming convention for MIDlet jars.

```
add-line-numbers=Off
clean=Off
compile=yes
compile-midlet=no
generated-tests = ${test-bin}/generated-tests
javac=${java.home}/../bin/javac
jar=${test-bin}/${midlet-name}-test.jar
jad=${test-bin}/${midlet-name}-test.jad
midlet-source=${project-dir}/src
```

```
midlet-resources=${project-dir}/res
midlet-classes=${output-classes}/../midlet-classes
midlet-manifest=${project-dir}/bin/MANIFEST.MF
midlet-under-test=${project-dir}/bin/${midlet-name}.jar
output-classes = ${test-bin}/classes
progress=TEXT
test=${project-dir}/test
test-bin=${test}/bin
test-classes = ${test-bin}/test-classes
test-report=${test}/testreport.xml
test-resources=${test}/res
test-source=${test}/src
coverage=off
coverage-map-file=${test-bin}/coverage.map
coverage-report=${test}/coverage.html
coverage-result=${test-bin}/coverage.xml
wtk=${project-dir}/../..
```

Using ANT to run mobile tests

The following is an example on how to use the ANT Java `task` to run a test:

```
<path id = "test-classpath" >
  <pathelement location="${junit}" />
  <pathelement location="${mobile-junit-jar}" />
</path>

<target name="run-javame-tests" >
  <java
    classname="com.sonyericsson.sdkme.junit.OnDeviceTest"
    fork="true" failonerror="true" >
     <classpath refid="test-classpath" />
     <arg value = "--device:SonyEricsson_K750_Emu"/>
     <arg value = "--compile:true" />
     <arg value = "--compile-midlet:true" />
  </java>
</target>
```

Note the class name `"com.sonyericsson.sdkme.junit.OnDeviceTest"`, and the `failonerror` instruction, which tells ANT to fail unless all tests pass.

In the `mobile-ju-sampleproject/build` directory, an entire ANT `build.xml` is included. It runs the above snippet. If the ANT installation is properly configured, and if “Generate Scripts” was selected at installation, the following command line should be sufficient to run it:

```
ant
```

If “Generate Scripts” was not selected at installation, the `build.properties` file located in the `build` directory may need to be modified.

Compiling a standalone test MIDlet

Normally, the MIDlet terminates after running all tests. In some instances, it might be useful to manually start, stop and restart tests. The command line parameters below will instruct Mobile JUnit to create a standalone MIDlet.

```
--test-runner-class:com.sonyericsson.junit.midletrunner.StandaloneMIDlet --
runmode:COMPILE-ONLY
```

An example can be found in the `build.xml` file (target name `compile-standalone`).

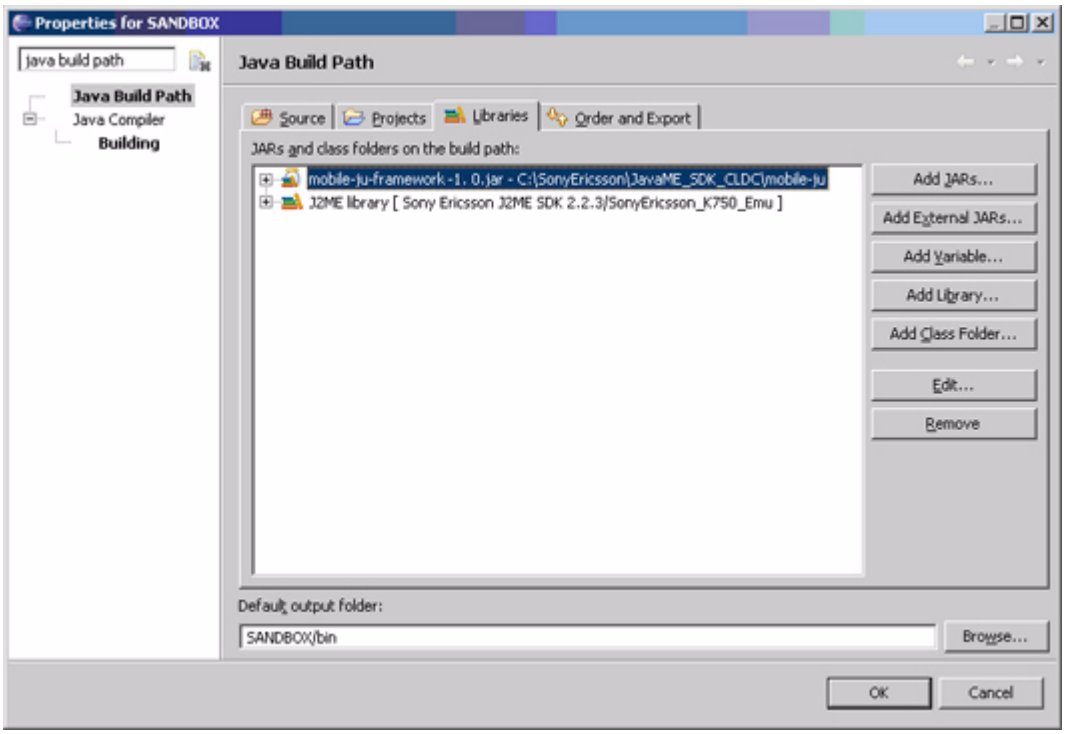
Configuring Eclipse and EclipseME for mobile test development

This section describes how to configure Eclipse version 3.2 and EclipseME version 1.5.0.

The recommended procedure to use Mobile JUnit with EclipseME is as follows:

1. Add the Mobile JUnit framework to the project classpath.

   - In project settings, open the Java Build Path properties page



   - Click *Add External JARs*

- Select the Mobile JUnit framework jar. If installed with default settings, this will be located at C:\SonyEricsson\JavaME_SDK_CLDC\mobile-ju\mobile-ju-framework-1.0.jar.

2. Use an ANT script to run the tests. See “Using ANT to run mobile tests” on page 84.

Using JUnit to run mobile tests

On-device tests can be run as “regular”JUnit tests. The class `com.sonyericsson.sdkme.junit.OnDeviceTest` also implements a JUnit test. The below code snippet illustrates how to run a on-device test as a JUnit test:

```
import junit.framework.Test;
import junit.framework.TestCase;
import junit.textui.TestRunner;

public class WrappedOnDeviceTest extends TestCase {

     public static Test suite() {
         OnDeviceTest test = new OnDeviceTest();
         return test;
     }
}
```

The VM argument mechanism is used to configure Mobile JUnit:

```
java -Dproject-dir=C:\SonyEricsson\JavaME_SDK_CLDC\PC_Emulation\WTK2\apps\Mobile-ju-sampleproject -Ddevice=SonyEricsson_W800_Emu [ ... more parameters ... ] WrappedOnDeviceTest
```

ANT or any IDE can also be used to configure the VM arguments.

For the above command line to work properly, a `main` method is required in `WrappedOnDeviceTest`:

```
     public static void main(String[] args) {
         TestResult result = TestRunner.run(suite());
         int errorsPlusFailures = result.failureCount() + result.errorCount();

         int exitCode = errorsPlusFailures > 0 ? 1 : 0;

         System.exit(exitCode);
     }
```

An `OnDeviceTest` can also be created from a set of command line arguments. `main` is here used in exactly the way as we used the Mobile JUnit tool in the very first example above.

```
     public static void main(String[] args) {
         OnDeviceTest test = new OnDeviceTest(args);
         [...]
     }
```

Links and references

Specifications

CLDC 1.0 (JSR 30)	http://www.jcp.org/en/jsr/detail?id=30
CLDC 1.1 (JSR 139)	http://www.jcp.org/en/jsr/detail?id=139
JTWI R1 (JSR 185)	http://www.jcp.org/en/jsr/detail?id=185
MIDP 1.0 (JSR 37)	http://www.jcp.org/en/jsr/detail?id=37
MIDP 2.0 (JSR 118)	http://www.jcp.org/en/jsr/detail?id=118
MMAPI (JSR 135)	http://www.jcp.org/en/jsr/detail?id=135
AMMS (JSR 234)	http://www.jcp.org/en/jsr/detail?id=234
WMA (JSR 120)	http://www.jcp.org/en/jsr/detail?id=120
WMA 2.0 (JSR 205)	http://www.jcp.org/en/jsr/detail?id=205
3D (JSR 184)	http://www.jcp.org/en/jsr/detail?id=184
Bluetooth (JSR 82)	http://www.jcp.org/en/jsr/detail?id=82
Optional Package (JSR 75)	http://www.jcp.org/en/jsr/detail?id=75
Java ME Web services 1.0 (JSR 172)	http://www.jcp.org/en/jsr/detail?id=172
Mascot Capsule	http://www.mascotcapsule.com

The Java ME platform

Sony Ericsson Developer World	http://www.sonyericsson.com/developer/
J2ME white paper	http://java.sun.com/products/cldc/wp/KVMwp.pdf
OTA Provisioning	http://java.sun.com/products/midp/OTAProvisioning-1.0.pdf
Helpful hints (white paper)	http://java.sun.com/j2me/docs/pdf/midpwp.pdf
Java Developer Connection Web site with Java Technical documentation	http://developer.java.sun.com/developer/infodocs/
Java Consumer Software Documentation Web site	http://java.sun.com/j2me/docs/

3D developer tools/plugins

Mascot Capsule Micro3D Version 3 plug-ins	http://www.mascotcapsule.com/toolkit/sony_ericsson/
Mascot Capsule Micro3D version 4 (JSR 184) plugins	http://www.mascotcapsule.com/M3G/download/e_index.html

Index