

Using Symbian OS

J A V A M E O N S Y M B I A N O S



symbian
Press

Secure your route to market

Symbian Signed is an industry-backed scheme that allows Symbian OS software developers to obtain a digital signature for their applications. To be 'Symbian Signed' applications must pass various standardised tests. Some network operators and phone manufacturers (e.g. Orange, Nokia and Sony Ericsson) now require that C++ and AppForge MobileVB/Crossfire applications must be signed if they are to be distributed through their channels.

Applications that pass Symbian Signed can run without the 'application source untrusted' warning and are published in a business to business applications catalog. ISVs with signed applications are also able to use the 'for Symbian OS' logo:



For more information visit www.symbiansigned.com

Using Symbian OS

J A V A M E O N S Y M B I A N O S

from

symbian
Press

Java ME on Symbian OS

part of the Using Symbian OS series

1st edition, 12/06

Published by:
Symbian Software Limited
2-6 Boundary Row
Southwark
London SE1 8HP
UK
www.symbian.com

Trademarks, copyright, disclaimer

Symbian®, 'Symbian OS' and other associated Symbian marks are all trademarks of Symbian Software Ltd. Symbian acknowledges the trademark rights of all third parties referred to in this material. © Copyright Symbian Software Ltd 2006. All rights reserved. No part of this material may be reproduced without the express written permission of Symbian Software Ltd. Symbian Software Ltd makes no warranty or guarantee about the suitability or accuracy of the information contained in this document. The information contained in this document is for general information purposes only and should not be used or relied upon for any other purpose whatsoever.



Compiled by:
Roy Ben Hayun

Managing Editors:
Freddie Gjertsen
Satu McNabb

Design Consultant:
Annabel Cooke

Merchant Adventurer:
James Mentz

Reviewed by:
Neil Broadbent
Alon Or-Bach

This booklet explains the implementation of Java ME on Symbian OS and how to develop your first Java application for a Symbian smartphone.

Contents

Java Technology and Language	1
Java Platforms	2
Java™ Platform Micro Edition (Java ME) platform.....	3
Symbian Java ME platform.....	4
What do I need?.....	5
Where do I find it?	7
HelloWorld example application	8
Build and run MIDP HelloWorld application.....	18
Who should I talk to next?.....	21
Talk about Symbian OS	22

Introduction

Mobile Java is one of the most widely installed runtimes on mobile phones. By enabling rich, advanced data services (for instance, location-based services and wireless commerce) it has gained recognition as a solid revenue stream for operators and content providers.

Java Technology and Language

Java technology is a collection of software products and specifications from Sun Microsystems that provides a system for development and deployment of cross-platform applications.

The Java programming language was introduced by Sun Microsystems in 1995 and is designed for use in the distributed environment of the Internet. It was designed to have familiar C++-like notation, but with greater simplicity and an enforced object-oriented single paradigm.

The Java programming language can be characterized by the following:

- Portability – Java applications are capable of executing on a variety of hardware architectures and operating systems.
- Robust - unlike programs written in C++, Java instructions cannot cause the system to ‘crash’.

- Secure – security features were designed into the language and run-time system.
- Object oriented - with the exception of primitive types, everything in Java is an object in order to support encapsulation and message-passing paradigms of object-based software.
- Multithreaded - Java supports multithreading and has synchronization primitives built into the language.
- Simple and familiar – this is of course relative to learning C++. Don't imagine though that Java can be learned in a day.

In Java, source code is written in .java files which are then compiled into .class files by the Java compiler. A .class file contains platform architecture neutral instructions (bytecode) which are machine language instructions of the Java Virtual Machine (JVM).

Java platforms

A Java platform is a software-only platform that runs on top of a native platform, this being the combination of the operating system and the underlying hardware.

A Java platform is divided into two components:

- The JVM that executes the Java language bytecode – this can be ported to various native platforms.
- The collection of Java APIs - ready-made software libraries that provide programming capabilities.

Together, the JVM and APIs insulate an application from the native platform.

There are three main Java platforms which target different application environments:

- Java Platform Micro Edition (Java ME) — mobile environments with limited resources
- Java Platform Standard Edition (Java SE) — workstation environments
- Java Platform Enterprise Edition (Java EE) — large distributed enterprise environments.

The purpose of this booklet is to take a closer look at Java ME and explain how to develop your first application.

Java Platform Micro Edition (Java ME) platform

Java ME has established itself as the number one wireless gaming platform by volume of sales.

Java ME is the Java platform that allows programmers to use the Java language and tools to develop applications for mobile wireless information devices such as mobile phones and PDAs.

Java ME is subdivided by Configurations, Profiles and optional packages.

A Configuration is the specification of a JVM and a base set of necessary class libraries; commonly based on a subset of the J2SE APIs. Currently, there are two Java ME Configurations:

- Connected Limited Device Configuration (CLDC)
- Connected Device Configuration (CDC).

A Profile sits on top of a configuration to complete the runtime environment by adding more high-level APIs, thereby preparing a device for a specific device category. Currently, there are two common Java ME Profiles:

- Mobile Information Device Profile 1.0 (MIDP1)
- Mobile Information Device Profile 2.0 (MIDP2).

A widely adopted standard is the combination of MIDP/CLDC to provide a complete Java platform for mobile phones and other similar devices.

Optional packages add functionality to the Java ME platform by offering standard APIs for the use of various technologies, such as databases, connectivity, web services, 3D graphics and much more.

The Symbian Java ME platform

Java ME on Symbian OS exposes the strengths and power of Symbian OS to Java developers through standard Java APIs.

A Java implementation is only as good as the underlying platform. The Symbian OS MIDP runtime environment is an integrated component of the operating system. The platform delivered is a Java ME stack incorporating a

high-performance CLDC1.1 virtual machine and a bespoke MIDP2.0 implementation designed to allow Java applications to exploit the advanced multitasking, graphics, and comms capabilities of the underlying OS. The stack includes similarly highly integrated implementations of many optional Java ME API packages, including Bluetooth, 3D graphics, Multimedia and Content Handling.

Symbian OS v9 supports very high Java ME standards:

- CLDC 1.1 - backward-compatible revision of CLDC 1.0 with many additions and enhancements that include support for floating-point math.
- MIDP 2.0 - backward-compatible version of MIDP 1.0 with new features such as multimedia and game functionality, rich user interface, extensive connectivity, over-the-air provisioning (OTA), and robust security model.
- Large collection of optional APIs – such as WMA, File and PIM access, Bluetooth, 3D graphics and advanced multimedia.
- No limits on computing resources – heap, threads, etc.
- On-device debugging.

What do I need?

First of all, you will need development software to help you to write the code and compile it. This is best done

with an Integrated Development Environment (IDE), which contains an editor, compiler and other development tools. NetBeans is an open source Java IDE that provides developers with everything they need to create cross-platform Java applications. When you download the NetBeans IDE, you get a modular, standards-based IDE with all the key functionality.

For Java ME development you need to integrate the NetBeans Mobility Pack into the IDE, which can be used to write, test, and debug applications on MIDP 2.0/CLDC 1.1-enabled mobile devices. It also allows you to integrate third-party SDKs for a robust testing environment for various device vendors - and this is what we will do.

You can download and run the NetBeans IDE 5.0 Installer from www.netbeans.org/community/releases/5.0. When the installation completes, you can download and run the NetBeans Mobility Pack 5.0 Installer from the same URL. You now have an IDE for Java ME application development.

The other software you will need is a MIDP Software Development Kit (SDK), which allows you to write and build programs for your phone. There are a number of different MIDP SDKs available, depending on the phone(s) you want to target – see the following pages for

further details of which MIDP SDK(s) are required.

If you do not have a particular phone or platform in mind and are unsure of which MIDP SDK to select, we would recommend you start with both an S60 and a UIQ SDK. Either opt for both of them for Symbian OS v9 (i.e., UIQ 3 and S60 3rd Edition) if you want to write applications for the latest phones (such as the Nokia E and Nseries, or Sony Ericsson P990i) or go for the previous version of the SDKs (i.e., UIQ 2.1 and S60 2nd Edition FP2).

Where do I find it?

This section specifies where to find the resources discussed in the previous section.

IDEs:

- NetBeans
www.netbeans.org/community/releases/50
- Eclipse
www.eclipse.org/downloads

SDKs:

- S60 Platform SDKs for Symbian OS, for Java
www.forum.nokia.com/main/resources/tools_and_sdks/listings/java_tools.html
- Carbide.j
www.forum.nokia.com/main/resources/tools_and_sdks/carbide/index.html

- UIQ3.0
www.developer.uiq.com/devtools_uiqsdk.html
- SEMC SDK for the Java ME platform
www.developer.sonyericsson.com/site/global/docstools/java/p_java.jsp
- WTK
www.java.sun.com/products/sjwtoolkit

A HelloWorld application explained

The execution unit in MIDP is the MIDlet. This is similar to a web Applet, but one that conforms with CLDC and MIDP and runs on a mobile device.

Below is the canonical HelloWorld example which gives a quick overview of a minimal MIDP application.

```
import javax.microedition.midlet.MIDlet;

import
javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.TextBox;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
/**
 * Hello, World example
 * /
```

```
public class HelloWorld extends MIDlet
implements CommandListener {

    /**
     * Exit command
     */
    private Command iExitCommand;

    /**
     * MIDlet constructor
     */
    public HelloWorld() {
        iExitCommand = new Command("EXIT",
        Command.EXIT, 1);
    }

    /**
     * Signals the MIDlet that it has entered
     the Active state.
     */
    public void startApp() {
        TextBox textBox =
            new TextBox("Hello World", "Hello,
            World!", 256, 0);
        textBox.addCommand(iExitCommand);
        textBox.setCommandListener(this);
        Display.getDisplay(this)
```

```

        .setCurrent(textBox);
    }

    /**
     * Signals the MIDlet to enter the Paused
     * state.
     */
    public void pauseApp() {
    }

    /**
     * Signals the MIDlet to terminate and enter
     * the Destroyed state.
     */
    public void destroyApp
    (boolean aUnconditional) {
    }

    /**
     * Indicates that a command event has
     * occurred on a Displayable.
     */
    public void commandAction(Command aCommand,
    Displayable aScreen) {
        if (aCommand == iExitCommand) {
            notifyDestroyed();
        }
    }
}

```

Now we will explain each section in more detail.

```
import
```

The first six lines import MIDP-specific classes required for the HelloWorld example, which are a part of MIDP along with a modified subset of the Java programming language:

- The `javax.microedition.midlet.MIDlet` class
- The `javax.microedition.lcdui.CommandListener` interface
- User interface (UI) classes.

(There is also always an implicit import to the `java.lang.*` package).

```
public class HelloWorld extends MIDlet
```

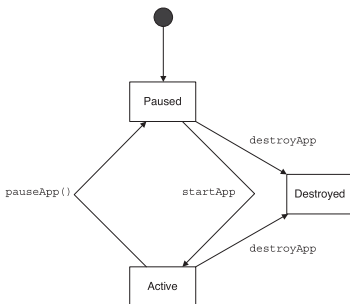
This line declares that our HelloWorld is extending the only class in the `javax.microedition.midlet.*` package, the MIDlet class.

Each MIDP application must provide a derived instance of the abstract MIDlet class, which overrides the following three MIDlet lifecycle methods:

- `MIDlet.startApp()`
- `MIDlet.pauseApp()`
- `MIDlet.destroyApp()`

The MIDlet life cycle is well defined in the MIDP specification below.

1. The Application Management Software (AMS) creates a new instance of a MIDlet and the default constructor for the MIDlet is called.
2. The MIDlet is now in the Paused state.
3. When the AMS has decided that now the MIDlet should run, it calls `startApp()` method so that the MIDlet can acquire any resources it needs and begin to perform its service. The MIDlet is now in the Active state.
4. When the AMS has determined that the MIDlet is no longer needed, it signals the MIDlet to stop performing its service and to release any resources it currently holds by calling the `destroyApp()`. The MIDlet is now in the Destroyed state.
5. During MIDlet execution the AMS might want the MIDlet to significantly reduce the amount of resources it is consuming (this could happen, for example, when a phone call comes in). The AMS will then signal to the MIDlet to reduce its resource consumption by calling the `pauseApp()` method and the MIDlet will then move to be in the Paused state.



```
implements CommandListener
```

This line declares that the HelloWorld MIDlet implements the `CommandListener` interface that is used by applications which need to receive high-level events from the implementation.

The `CommandListener` interface has a single method:
`public void commandAction(Command c, Displayable d)`
 which is invoked by the implementation to indicate that a command event has occurred on `Displayable d`.
 At the bottom of the HelloWorld you will see how the MIDlet implements this callback method.

```
private Command iExitCommand;
```

In this line, the MIDlet declares it has a member variable `iExitCommand` which is a reference to an instance of `javax.microedition.lcdui.Command`.

A Command can be implemented in various user interface forms, which have a semantic of an activation of a single action. This can be a soft button, item in a menu, and possibly even a speech interface may present this Command as a voice tag.

```
public HelloWorld() {  
    iExitCommand = new Command("EXIT",  
        Command.EXIT, 1);  
}
```

As said before, the HelloWorld constructor will be executed before the `startApp()` method is called. A Command contains three pieces of information:

- a label
- a type
- a priority.

In our example, the constructor initializes the `iExitCommand` to display a label “EXIT” and to be of type `Command.EXIT`. It does not mean that when the user invokes this command, the application will exit automatically. Only when the `CommandListener.commandAction()` is called, should the MIDlet decide if it wants to exit.

```
public void startApp()
```

As explained about MIDlet life cycle, when the AMS decides that HelloWorld should run, it calls `startApp()` method.

In this method HelloWorld will acquire any resources it needs and will then move into the Active state.

```
TextBox textBox = new TextBox("Hello World",  
"Hello, World!", 256, 0);
```

This line shows the initialization of the `TextBox` screen. All objects that have the capability of being placed on the display derive from a base class called `Displayable`. MIDP `LcdUI` defines two hierarchies of `Displayables`:

- Higher-level - `Screen` is the common super class of all high-level user interface classes. The contents displayed and their interaction with the user is defined by concrete subclasses.
- Lower-level - Although not in this example, the second hierarchy derives from the `Canvas` base class for handling low-level graphical operations.

The `TextBox` in our example is one of the high-level concrete subclasses of `Screen`, which allows the user to enter and edit text.

It is created with “HelloWorld” as the title string, “Hello, World!” as the initial contents, 256 characters maximum size and no user input constraints.

```
"textBox.addCommand(iExitCommand);"
```

The above line adds the `iExitCommand` to the display of the `TextBox`.

As explained about user interface forms, the implementation may choose to add the `Command` to any of the soft buttons or add it to a menu, depending on the phone UI platform.

```
"textBox.setCommandListener(this);"
```

The application provides the `TextBox` with the instance of `HelloWorld` as the `CommandListener`, to receive high-level events from the implementation.

Any high-level events occurring when this `TextBox` is displayed will be notified to `HelloWorld`.

```
"Display.getDisplay(this).setCurrent(textBox);"
```

Now that the `TextBox` has been created, it can be displayed on the device screen. In this line, the current display is set to be the newly created `TextBox`.

When `HelloWorld` is first started, there is no current `Displayable` yet.

A `MIDlet` can get a reference to a manager of the display (abstracted as the `Display`) and input devices of the system, by calling the `Display.getDisplay()` method. This line ensures that the display manager makes `TextBox` visible and enables it to interact with the user.

```
"public void commandAction(Command aCommand,
    Displayable aScreen)"
```

This method is the implementation of the `CommandListener` interface which the `MIDlet` declared himself as implementing.

When the implementation indicates that the `iExitCommand` command event has occurred, this method will be called.

```
"notifyDestroyed();" "
```

The `MIDlet` notifies the `AMS` that it has entered into the `Destroyed` state.

Please note that in a more complex example a `MIDlet` must perform the same cleanup and releasing of resources it would have done if the `MIDlet.destroyApp()` had been called by the `AMS`.

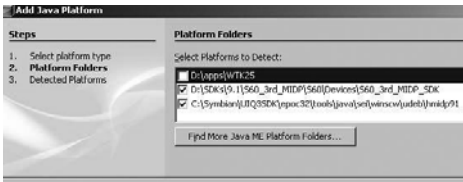
Hopefully, this explanation has succeeded in giving a clear example of a minimal `MIDP` application that displays information and can receive input from the user.

In the next section, we explain how to develop a `MIDP` application for Symbian OS by using industry-standard tools.

Build and run MIDP HelloWorld application

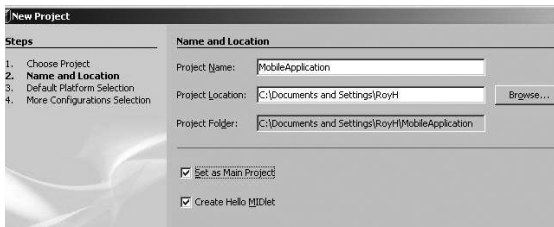
Download and install the S60 Platform SDKs for Symbian OS for MIDP.

Download and install the UIQ3.0 SDK for Symbian OS. Select 'Tools' -> 'Java Platform Manager' -> 'Add Platform...'. You will be presented with 'Add Java Platform' page in which you can tick the "Java Micro Edition Platform Emulator". Hit 'Next' and select 'Find More Java ME Platform Folders...'. Browse to either or both the S60 3rd Edition SDK installation and/or UIQ3.0 SDK installation folder and finish the SDK integration with NetBeans.

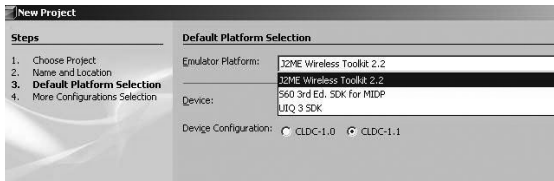


Once the IDE and SDK installation process has completed, your system should be ready for developing your first Symbian OS MIDP application!

Launch NetBeans from the Start menu. Select 'File' -> 'New Project'. A Project Wizard page will be displayed. In the 'Categories' panel, select 'Mobile' and in the 'Projects' panel, select 'Mobile Application', then hit 'Next'. You will be presented with 'Name and Location' page in which you can tick the 'Create Hello MIDlet', which will auto-generate a similar HelloWorld MIDP application.



Hit 'Next' and move to the 'Default Platform Selection' page. As you already integrated the SDK, you can now select either 'S60 3rd Ed. SDK for MIDP' in the 'Emulator Platform' list and 'S60Emulator' as your 'Device', or 'UIQ 3 SDK' and 'win32'.



Now click 'Finish'. The IDE will create a basic 'Hello World' project for you. This should appear in the 'Projects' view of the IDE. If you expand the project folder, you will be able to see the contents. Click on the 'Source' tab and the source file will be displayed in the main editor view of the IDE.

Build the project using the 'Build Main Project' option from the 'Build' menu. Assuming there are no errors (which should be the case as this project has been

generated from a template), you are now ready to run the project in the emulator.

To run the application, go to the 'Run' menu and select the 'Run Main Project' option.

This launches the emulator progress dialog.



Wait for the emulator to come up and see your application running - you have now developed your first application!



To close the application, do not close the emulator but click on 'Abort' on the emulator progress dialog. The emulator will stay running, thereby minimizing start-up time.

To run the application on a real phone it is necessary to copy the Jar and Jad files, which NetBeans already generated when you built the project (they are under the '\dist' folder in your new project location). You should be able to install them onto the phone via Infrared (IR) or Bluetooth and run the application. Beam the application over to the phone and follow the installation process. Once the installation is complete, your application should appear in the list of applications available on the phone.

Who should I talk to next?

SDN (Symbian Developer Network) supports developers with a range of products and services. The SDN website (<http://developer.symbian.com>) contains everything you need to get started on Symbian OS, as well as pointers to community websites like Forum Nokia, UIQ, Sony Ericsson and key network service providers, such as Orange and Vodafone. Sign up to the SDN and you'll have access to a range of services, including technical papers, knowledgebase, system documentation, SDKs and public newsgroups.

Talk about Symbian OS

Share problems and solutions with other developers and interested parties on the public newsgroups. These are accessible via the SDN website and the following free discussion forums are available:

- `discussion.general` – for discussing all things Symbian
- `discussion.symbian.tools` – to discuss the use of the free tools released by Symbian
- `discussion.epoc.java` – to talk about Java on Symbian OS
- `discussion.epoc.connect` – for discussion about the Connectivity Software and Connectivity plug-ins
- `discussion.epoc.hardware` – to discuss licensees' products based on Symbian OS
- `feedback.api` – for feedback about third party API libraries and utilities
- `feedback.docs` – for feedback about the accuracy, quality, content or coverage of the Symbian OS SDK documentation
- `feedback.sdks` – for feedback about Symbian OS SDKs

Developer Resources:

Symbian Developer Network

<http://developer.symbian.com>

Symbian Developer Network Newsletter

<http://developer.symbian.com/register.action>

Symbian OS Tools and SDKs

<http://developer.symbian.com/main/tools>

UIQ Java development forum

<http://developer.uiq.com/forum.html>

S60 forum

www.forum.nokia.com/main/resources/technologies/java

Developing for Sony Ericsson

www.sonyericsson.com/developer/java

Learn about Java ME

<http://java.sun.com/javame>

<http://java.sun.com/products/cldc>

<http://java.sun.com/products/midp>

<http://developers.sun.com/techtopics/mobility/learning/tutorial>

Developer Survey



5 minutes to influence Symbian

Symbian wants to know what you think about our developer support.

We have created a survey which we will use to shape our support programs.

If you want to influence our decisions, or if you just want the opportunity to win the series of Symbian Press books, then visit:

<http://developer.symbian.com>

and complete the questionnaire.

New from

symbian
Press

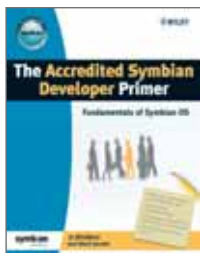


Symbian OS Platform Security



...is a revelatory discourse on the Symbian OS security model and related Symbian initiatives, such as Symbian Signed, that will prove vital to all developer audiences, from licensees to ISVs.

Accredited Symbian Developer Primer



...is the official guide to passing the ASD exam and becoming an Accredited Symbian Developer. This book is authored by industry specialists and has clear and concise explanations.

Symbian Press: www.developer.symbian.com/main/academy/press

Also from

symbian
Press



For all Symbian C++ developers:

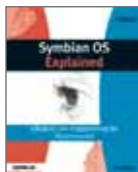
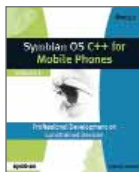
Developing Software for Symbian OS
by Steve Babin

Symbian OS C++ for Mobile Phones – Volume 1
by Richard Harrison

Symbian OS C++ for Mobile Phones – Volume 2
by Richard Harrison

Symbian OS Explained
by Jo Stichbury

Symbian OS Internals
by Jane Sales



Also from

symbian
Press



For enterprise and IT professionals:

Rapid Mobile Enterprise Development for Symbian OS
by Ewan Spence

For Symbian OS project managers:

Symbian for Software Leaders
by David Wood

For connectivity application developers:

Programming PC Connectivity Applications for Symbian OS
by Ian McDowall

For Java developers:

Programming Java 2 Micro Edition for Symbian OS
by Martin de Jode



Also from

symbian
Press



Published Booklets

Coding tips

Signing tips

Performance Tips

Essential UIQ - Getting Started

Getting started

Translated Booklets

新手入门 (Getting Started, Chinese)

编码诀窍 (Coding Tips, Chinese)

UIQ 精要新手入门 (Essential UIQ - Getting Started)

コーディング技法 (Coding Tips, Japanese)

性能に関するヒント (Performance Tips, Japanese)



[illegible]

Notes

Using Symbian OS

J A V A M E O N S Y M B I A N O S



Java ME on Symbian OS explains how to start producing Java ME applications on Symbian OS. Providing valuable tips and practical advice, this booklet is essential reading to anyone wishing to learn more about Java on Symbian OS.

Java ME on Symbian OS is part of the Using Symbian OS series, designed to provide information in a handy format to Symbian OS developers.

Symbian Press

Symbian Press publishes books designed to communicate authoritative, timely, relevant and practical information about Symbian OS and related technologies. Information about the Symbian Press series can be found at

<http://developer.symbian.com/main/academy/press>