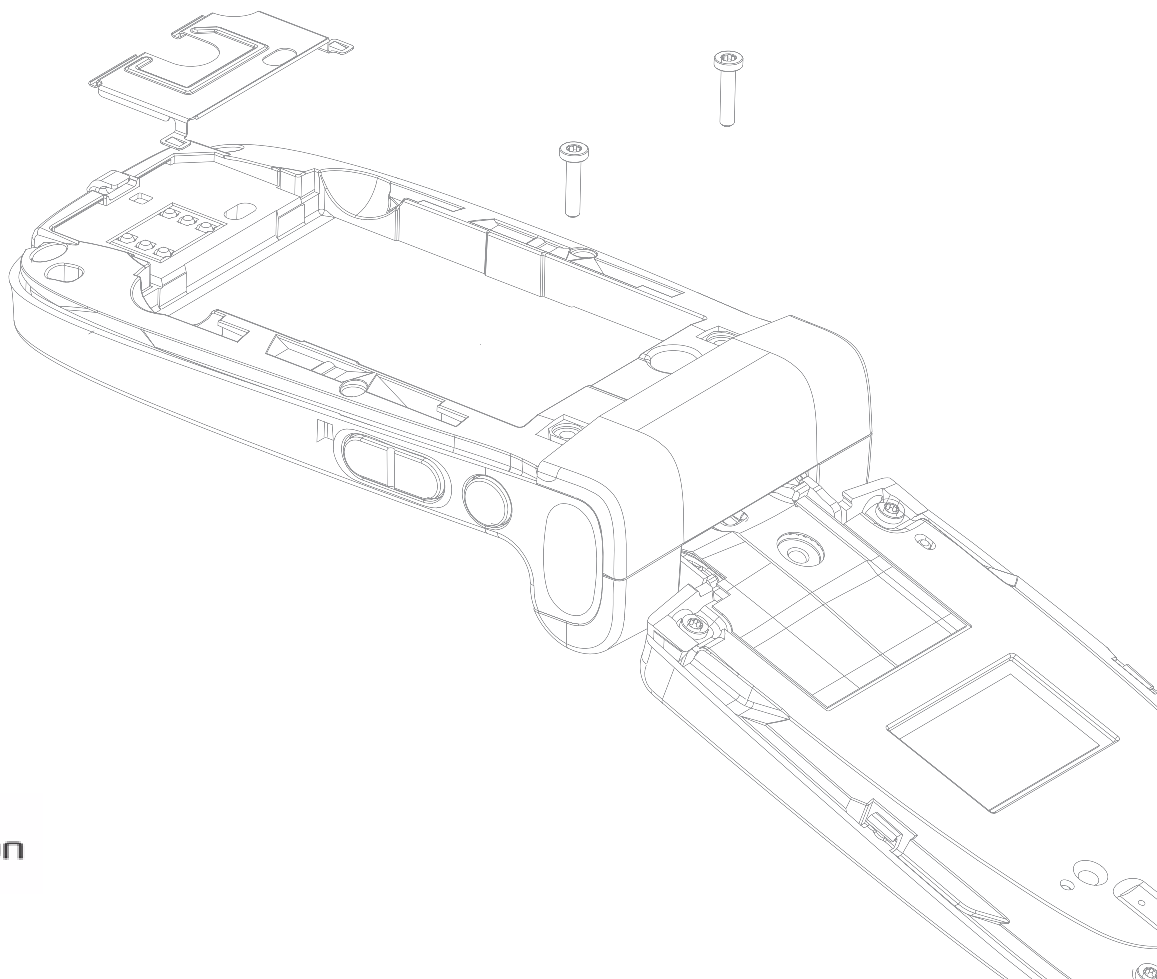
Tutorial

DEVELOPER WORLD THE FAST TRACK FROM MIND TO MARKET

September 2006

Getting started with on-device debugging for NetBeans

with Sony Ericsson phones



Sony Ericsson

Preface

About this tutorial

This tutorial has been authored by Thomas Bailey, a Product Tools Manager within the Sony Ericsson Developer Program.

The document describes how to get started with on-device debugging with NetBeans IDE 5.0, NetBeans Mobility Pack 5.0 and a Sony Ericsson K800. It is intended for developers of Java™ applications who want a background to on-device debugging technology and how to use it.

It is assumed that the reader is familiar with the basics of the Java Platform, Micro Edition, CLDC.

This tutorial is 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/

First edition (September 2006)
Publication number: EN/LZT 108 9075 R1A

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

Terminology

JPDA	**Java Platform Debugger Architecture**. The Java Platform Debugger Architecture provides the infrastructure to build end-user debugger applications for the Java Platform, Standard Edition (Java SE).
JDI	**Java Debug Interface**. An interface for JPDA designed for the Java platform.
JDWP	**Java Debug Wire Protocol**. A protocol that defines the format and requests between a debugger and a Java SE virtual machine.
KDWP	**KVM Debug Wire Protocol**. Subset of JDWP used in a resource constrained environment.
JVM	**Java Virtual Machine**. A software model of a virtual machine that executes Java byte code.
KVM	**K-Virtual Machine**. A virtual Machine designed for use in resource constrained environment, created by Sun Microsystems.
SDK	**Software Development Kit**. Typically a set of development tools that allows a software engineer to create applications for a certain software package, software framework, hardware platform, computer system, operating system or similar.

IDE	**Integrated Development Environment**. Suite of tools, usually related to programming, brought together into a visual environment.
ODD	**On-device debugging**. Debugging a phone VM.

Typographical conventions

Code is written in Courier font:

```
public StringItem getHelloStringItem ()
```

Trademarks and acknowledgements

Document history

Change history		
2002-09-15	Version R1A	Document published on Developer World

Contents

Introduction

On-device debugging refers to the ability to debug a MIDP application while it is executing on a real device, typically a phone. This tutorial covers the KDWP technology behind on-device debugging and presents its usage together with the NetBeans 5.0 IDE and Mobility Pack 5.0.

For initial development and prototyping, an emulator, such as the Sun Java Wireless Toolkit (WTK) device profiles or the Sony Ericsson SDK for the Java ME platform, provides a sufficient and viable environment for development. However, as complexity increases, particularly when working with 3D or network intensive applications, development on the phone becomes an increasingly attractive proposition to reduce the number of development cycles by identifying and addressing issues immediately.

A phone is typically regarded as a "black box", where error messages are usually user-orientated and provide little clue as to a point of failure. On-device debugging addresses this by exposing the virtual machine allowing Java SE debugging tools to be connected to it, allowing for breakpoints, stepping and variable watches.

The NetBeans debugger, through the use of the underlying KDWP technology and Mobility Pack, can be readily used towards a mobile phone VM without requiring any vendor specific plug-ins. The same debugging methodology applied to Java SE applications and can be re-applied when debugging a MIDlet on-device.

T610 (2003) W850 (2006)

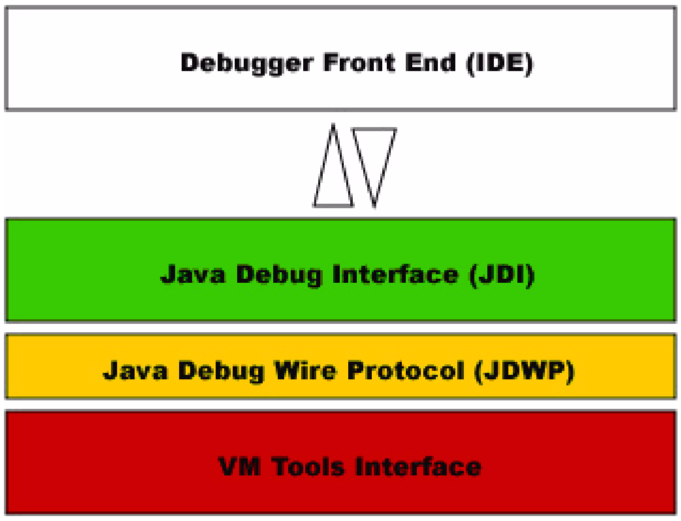
On-device debugging is found in all Sony Ericsson feature phones going as far back as the T610 in 2003.

How does on-device debugging work?

Underlying on-device debugging is KVM debug wire protocol (KDWP), a subset of the more familiar Java debug wire protocol (JDWP).

JDWP is the protocol used between a debugger and virtual machine designed for usage in a Java SE environment. It defines the format of the information and a command set used to request and control the debugging behaviour of the target VM.

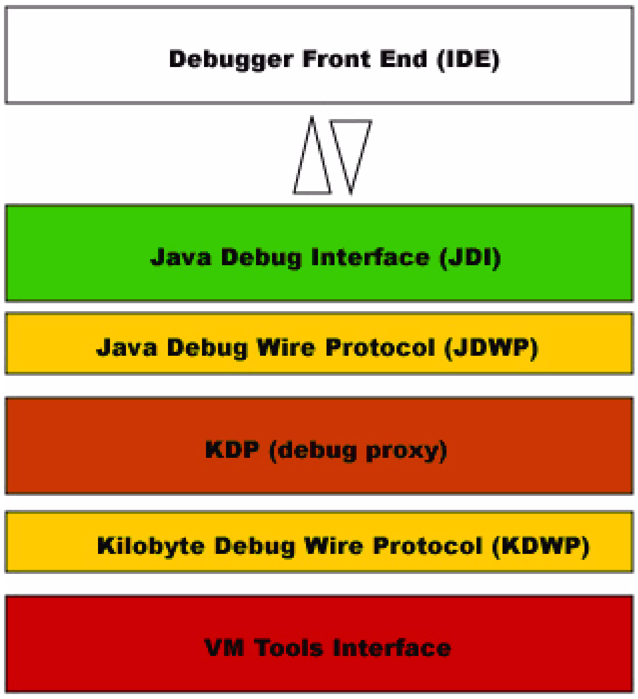
JDWP is one of the layers in the Java Platform Debugger Architecture (JPDA).



In the JPDA model, the debugger front end would usually implement the Java Debug Interface (JDI), or use JDWP directly to communicate with the VM tools interface.

However, the JPDA model can not be directly applied to a small footprint, resource constrained VM found in a mobile phone. To enable debugging on a mobile VM, KDWP is used in place of JDWP. KDWP implements only a subset of JDWP therefore reducing the requirements on the phone VM.

Together with KDWP, a debug agent or debug proxy (KDP) is introduced. It resides on the host and acts as a broker between JDWP and KDWP protocols. It serves to move logic from the phone VM to the host and takes over some of the responsibility of the VM, such as extracting and parsing debugging information from the class files and handling JDWP commands not implemented in KDWP.


Debugger Front End (IDE)
Java Debug Interface (JDI)
Java Debug Wire Protocol (JDWP)
KDP (debug proxy)
Kilobyte Debug Wire Protocol (KDWP)
VM Tools Interface

The use of the agent allows any JPDA debugger, such as the NetBeans debugger, to be used towards a phone VM – the debugger is “unaware” that it is debugging a phone VM.

Why use on-device debugging?

The benefits of on-device debugging are particularly useful when considering the variety and number of phones available on the market today.

Typically Java implementations come from a multitude of different vendors. Proprietary API, such as the Mascot Capsule 3D engine, may be available. The development kits and emulators, such as Sun Java Wireless Toolkit (WTK), may provide a useful prototyping environment but can not not always provide an identical environment to that of the phone.

On-device debugging allows you to quickly locate and resolve an issue in its native environment.

Getting started

In order to illustrate the process of setting up and debugging an application we will use the following tools:

- NetBeans IDE 5.0
  http://www.netbeans.org/products/ide/
  NetBeans IDE is a free Java IDE built on the NetBeans platform.

- NetBeans Mobility Pack 5.0
  http://www.netbeans.org/kb/50/mobility.html
  The NetBeans Mobility Pack 5.0 provides CLDC and MIDP awareness to the NetBeans 5.0 IDE

- A Sony Ericsson phone
  http://www.sonyericsson.com
  All Sony Ericsson feature phones support on-device debugging and require no software modification.

- Sony Ericsson SDK for the Java ME Platform
  http://developer.sonyericsson.com/getDocument.do?docId=65255
  The SDK provides the necessary libraries and emulation for developing towards Sony Ericsson phones.

- Sony Ericsson Developers Guidelines for Java ME, CLDC
  http://developer.sonyericsson.com/getDocument.do?docId=65067
  The Developer Guidelines are updated frequently and are intended to cover every aspect of Java ME CLDC development that is Sony Ericsson specific.

- Microsoft® Windows® XP, Service Pack 2
  Service Pack 2 provision native support for Bluetooth™ wireless technology which will be used to connect to the phone.

Setting up the tools

Having installed NetBeans and Mobility Pack together with the Sony Ericsson SDK for the Java ME Platform, the available device profiles must be made available to NetBeans using the Java Platform Manager.

The Sony Ericsson SDK for the Java Platform consists of two main branches, **emulator profiles** and **on-device profiles** respectively. This is reflected in the directory structure:

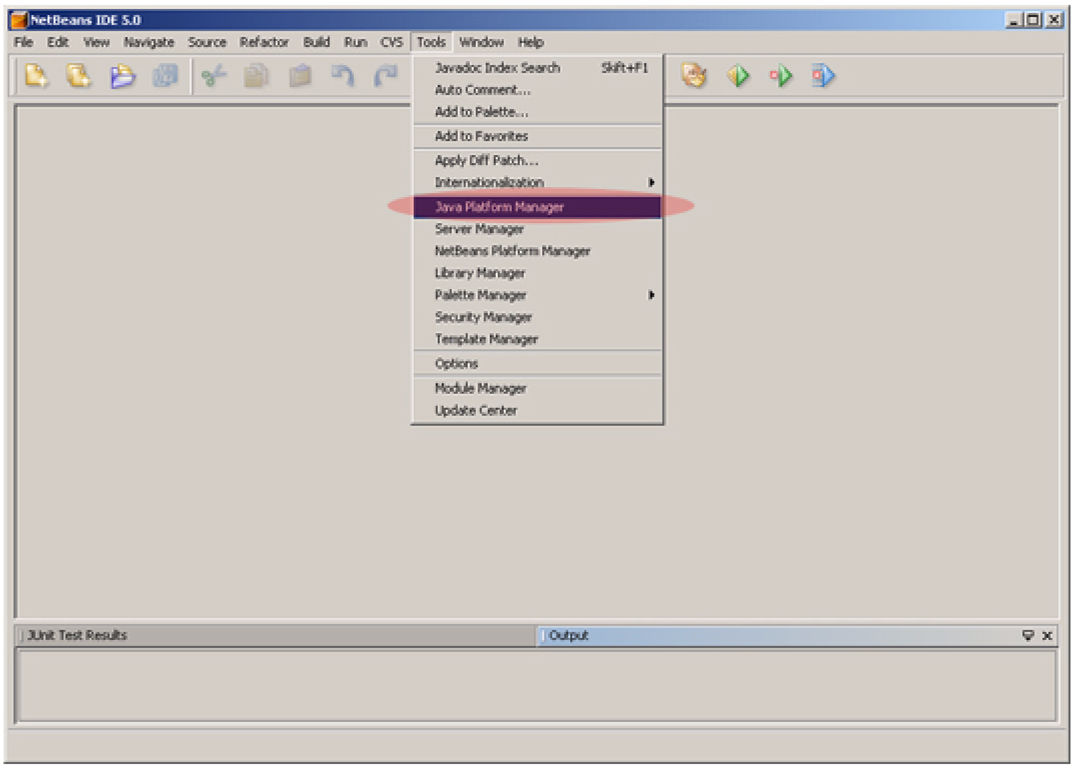
```
<your installation path>JavaME_SDK_CLDC
-- OnDeviceDebug
----bin
----lib
----docs
-- PC_Emulation
----WTK 1
-------bin
-------lib
-------docs
----WTK 2
-------bin
-------lib
-------docs
```

This structure follows the UEI (Unified Emulator Interface) specification. The specification intends to ensure interoperability between IDE and SDK vendors, for example between NetBeans and Sony Ericsson.

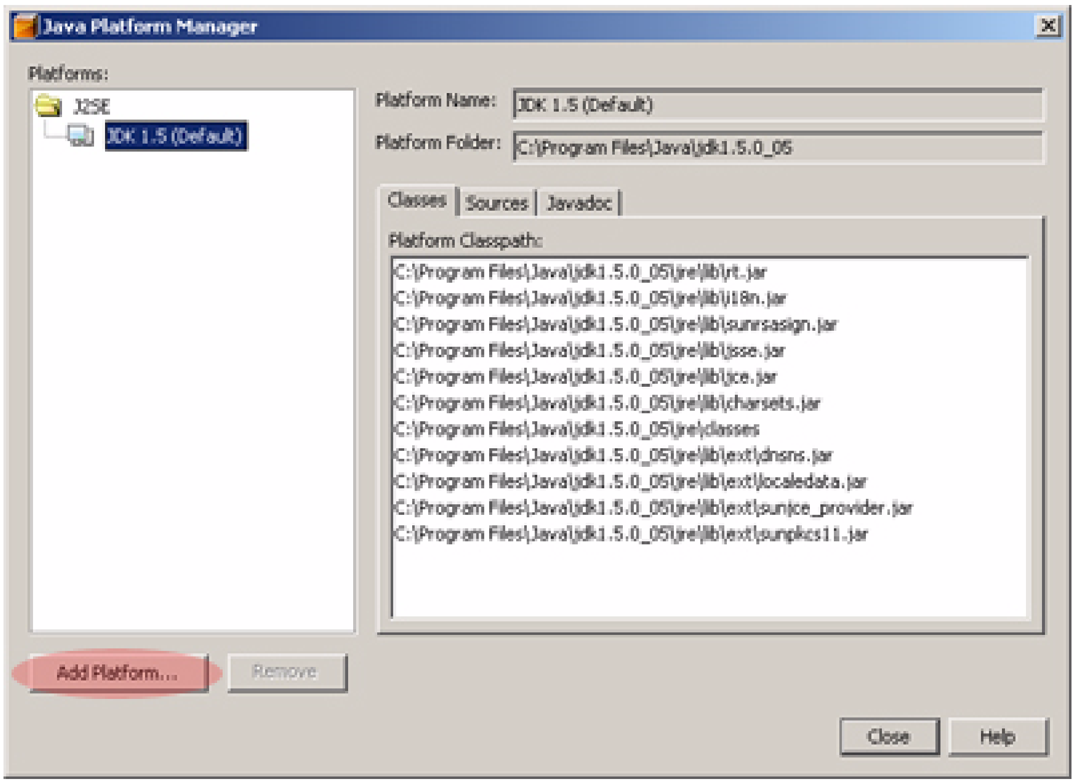
By following the UEI specification in the context of on-device debugging, the IDE regards the on-device profiles as emulators, removing the need for additional proprietary plug-ins.

Platform setup in NetBeans

The Java Platform Manager resides under the Tools menu:



By default, the platform list is empty. We need to point NetBeans at the Sony Ericsson SDK for the Java ME Platform. Click the "Add Platform..." button:

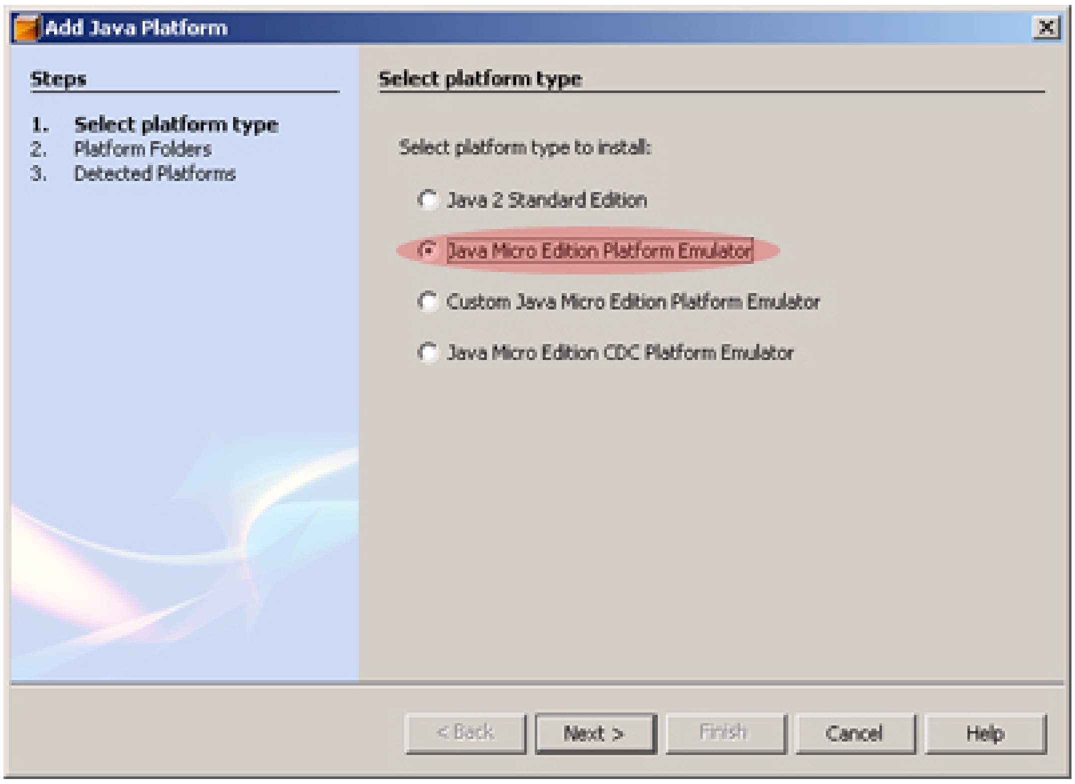


Platform Name: JDK 1.5 (Default)
Platform Folder: C:\Program Files\Java\jdk1.5.0_05

Platform Classpath:

```
C:\Program Files\Java\jdk1.5.0_05\jre\lib\rt.jar
C:\Program Files\Java\jdk1.5.0_05\jre\lib\i18n.jar
C:\Program Files\Java\jdk1.5.0_05\jre\lib\sunrsasign.jar
C:\Program Files\Java\jdk1.5.0_05\jre\lib\jsse.jar
C:\Program Files\Java\jdk1.5.0_05\jre\lib\jce.jar
C:\Program Files\Java\jdk1.5.0_05\jre\lib\charsets.jar
C:\Program Files\Java\jdk1.5.0_05\jre\classes
C:\Program Files\Java\jdk1.5.0_05\jre\lib\ext\dnsns.jar
C:\Program Files\Java\jdk1.5.0_05\jre\lib\ext\localedata.jar
C:\Program Files\Java\jdk1.5.0_05\jre\lib\ext\sunjce_provider.jar
C:\Program Files\Java\jdk1.5.0_05\jre\lib\ext\sunpkcs11.jar
```

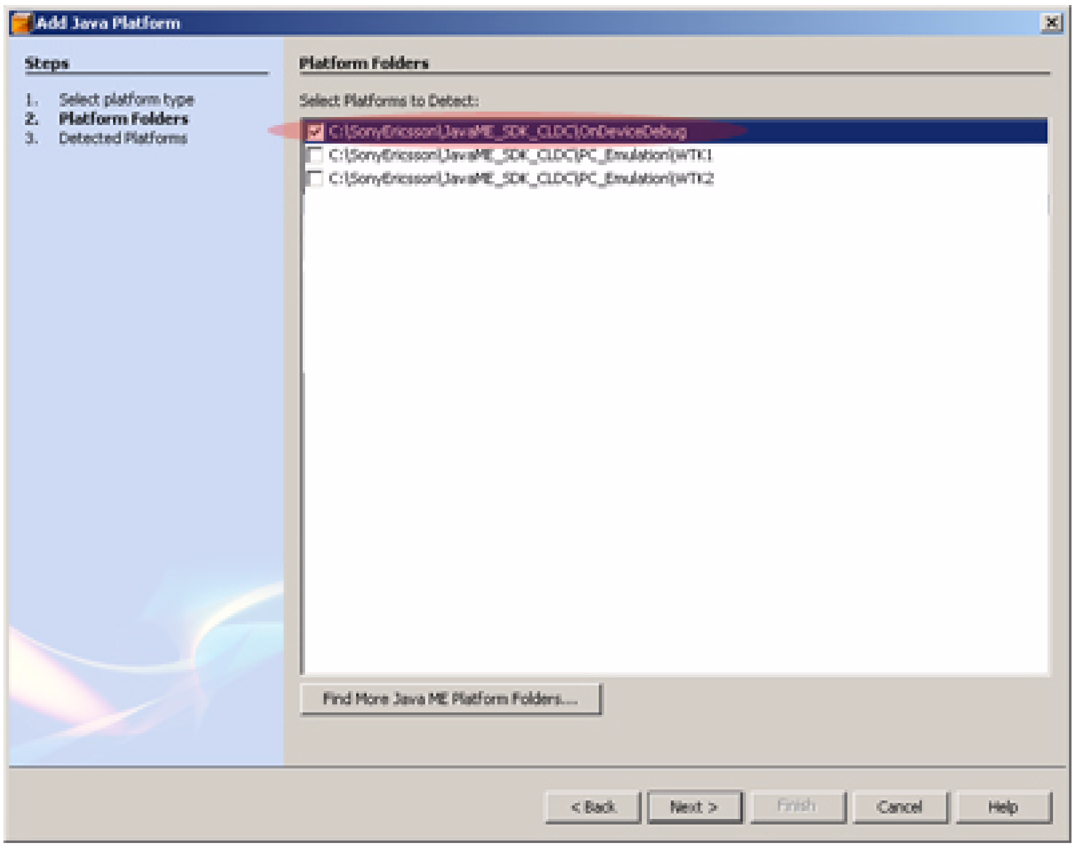
Select the "Java Micro Edition Platform Emulator" option and click "Next >". NetBeans will scan common directories and attempt to automatically find any relevant SDKs:



Select platform type to install:

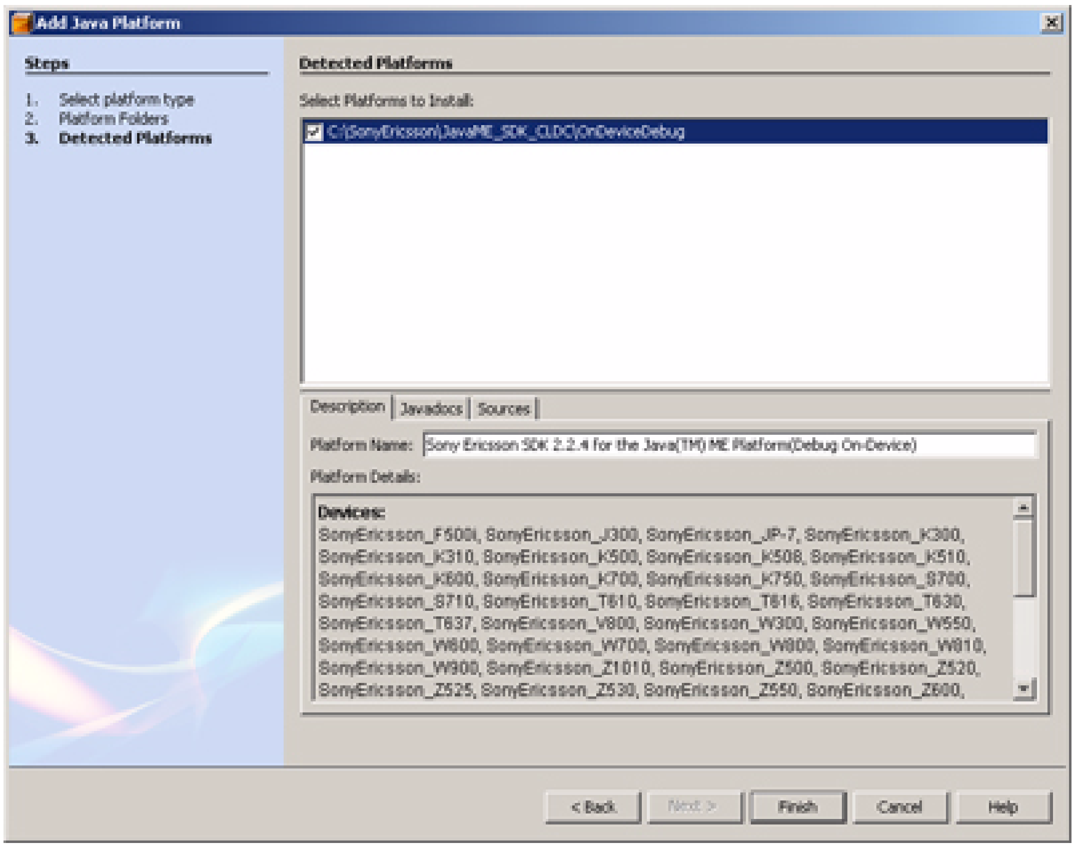
- ( ) Java 2 Standard Edition
- (•) Java Micro Edition Platform Emulator
- ( ) Custom Java Micro Edition Platform Emulator
- ( ) Java Micro Edition CDC Platform Emulator

Note that, using the UEI convention, it has automatically recognized the two branches, on-device debugging and emulation. Since we are interested in on-device debugging, check only the “<your installation path>\JavaME_SDK_CLDC\OnDeviceDebug” entry, then click “Next >”:



[✓] C:\SonyEricsson\JavaME_SDK_CLDC\OnDeviceDebug
[ ] C:\SonyEricsson\JavaME_SDK_CLDC\PC_Emulation\WTK1
[ ] C:\SonyEricsson\JavaME_SDK_CLDC\PC_Emulation\WTK2

NetBeans proceeds to query all the device profiles available in the platform selection. Click "Finish" to conclude the platform setup:



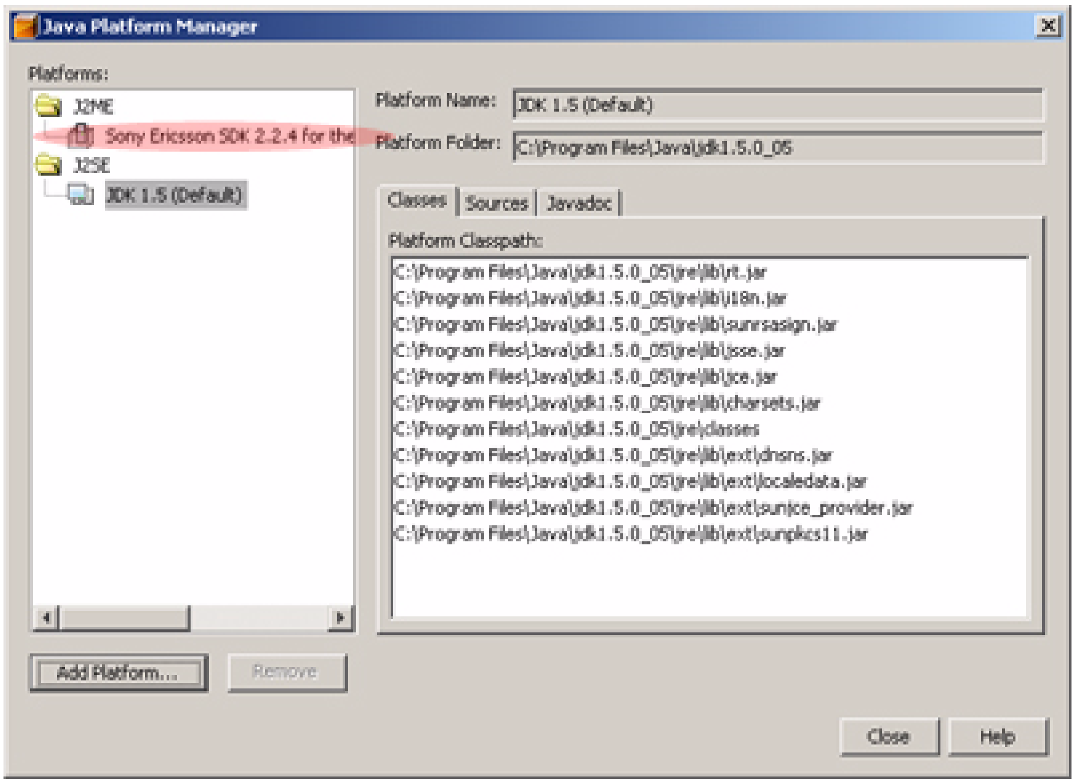
[✓] C:\SonyEricsson\JavaME_SDK_CLDC\OnDeviceDebug

Platform Name: Sony Ericsson SDK 2.2.4 for the Java(TM) ME Platform(Debug On-Device)

Platform Details:

Devices:
SonyEricsson_F500i, SonyEricsson_J300, SonyEricsson_JP-7, SonyEricsson_K300, SonyEricsson_K310, SonyEricsson_K500, SonyEricsson_K508, SonyEricsson_K510, SonyEricsson_K600, SonyEricsson_K700, SonyEricsson_K750, SonyEricsson_S700, SonyEricsson_S710, SonyEricsson_T610, SonyEricsson_T616, SonyEricsson_T630, SonyEricsson_T637, SonyEricsson_V800, SonyEricsson_W300, SonyEricsson_W550, SonyEricsson_W600, SonyEricsson_W700, SonyEricsson_W800, SonyEricsson_W810, SonyEricsson_W900, SonyEricsson_Z1010, SonyEricsson_Z500, SonyEricsson_Z520, SonyEricsson_Z525, SonyEricsson_Z530, SonyEricsson_Z550, SonyEricsson_Z600,

The platform is now ready to be used for development and debugging:



Platform Name: JDK 1.5 (Default)

Platform Folder: C:\Program Files\Java\jdk1.5.0_05

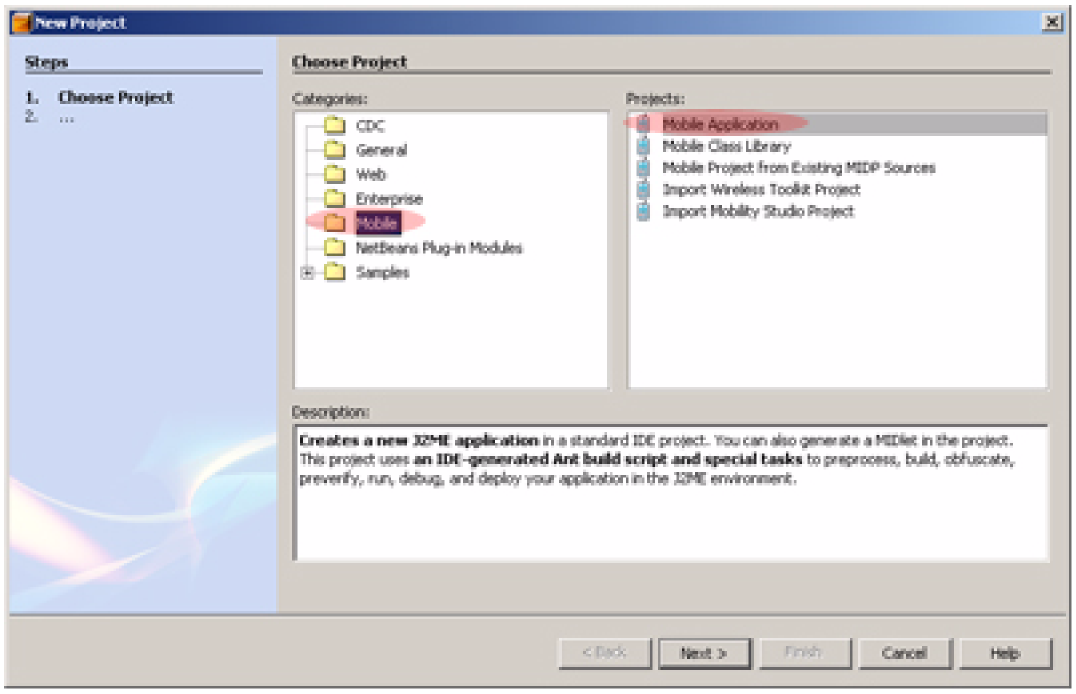
Platform Classpath:

```
C:\Program Files\Java\jdk1.5.0_05\jre\lib\rt.jar
C:\Program Files\Java\jdk1.5.0_05\jre\lib\i18n.jar
C:\Program Files\Java\jdk1.5.0_05\jre\lib\sunrsasign.jar
C:\Program Files\Java\jdk1.5.0_05\jre\lib\jsse.jar
C:\Program Files\Java\jdk1.5.0_05\jre\lib\jce.jar
C:\Program Files\Java\jdk1.5.0_05\jre\lib\charsets.jar
C:\Program Files\Java\jdk1.5.0_05\jre\classes
C:\Program Files\Java\jdk1.5.0_05\jre\lib\ext\dnsns.jar
C:\Program Files\Java\jdk1.5.0_05\jre\lib\ext\localedata.jar
C:\Program Files\Java\jdk1.5.0_05\jre\lib\ext\sunjce_provider.jar
C:\Program Files\Java\jdk1.5.0_05\jre\lib\ext\sunpkcs11.jar
```

Debugging a project

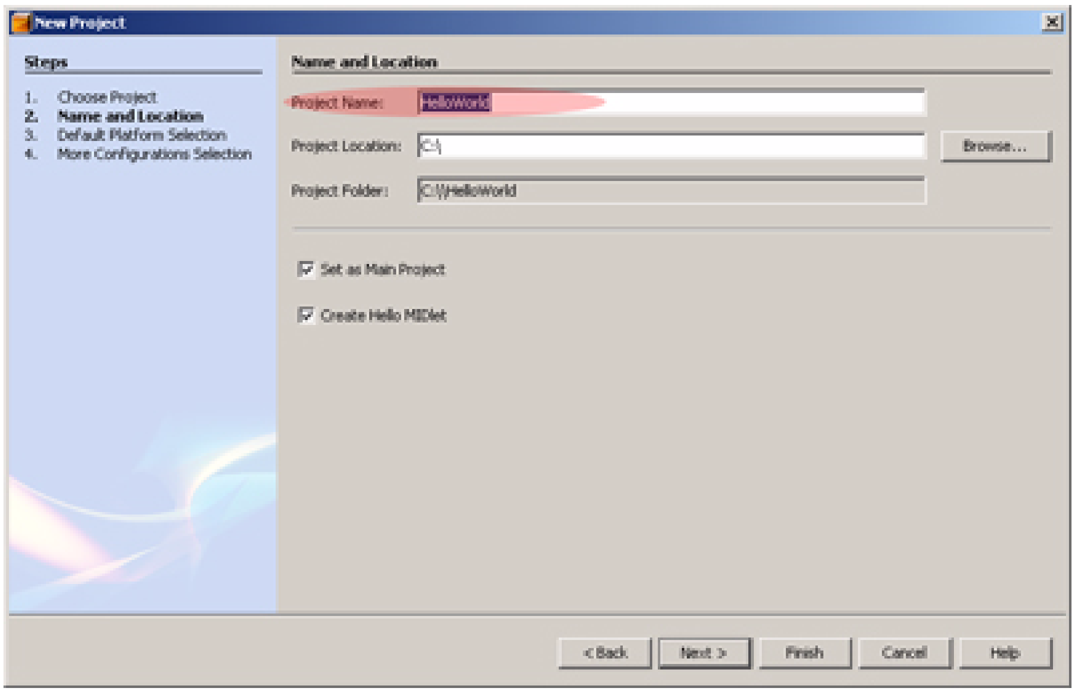
Creating a test project

In order to highlight on-device debugging we create a simple “Hello World” project. Use the Project Wizard to create a new project, selecting the "Mobile" category and "Mobile Application" project type:



Creates a new J2ME application in a standard IDE project. You can also generate a MIDlet in the project. This project uses **an IDE-generated Ant build script and special tasks** to preprocess, build, obfuscate, preverify, run, debug, and deploy your application in the J2ME environment.

Enter "HelloWorld" as a project name and click "Next >":



Project Name: HelloWorld

Project Location: C:\

Project Folder: C:\\HelloWorld

☑ Set as Main Project

☑ Create Hello MIDlet

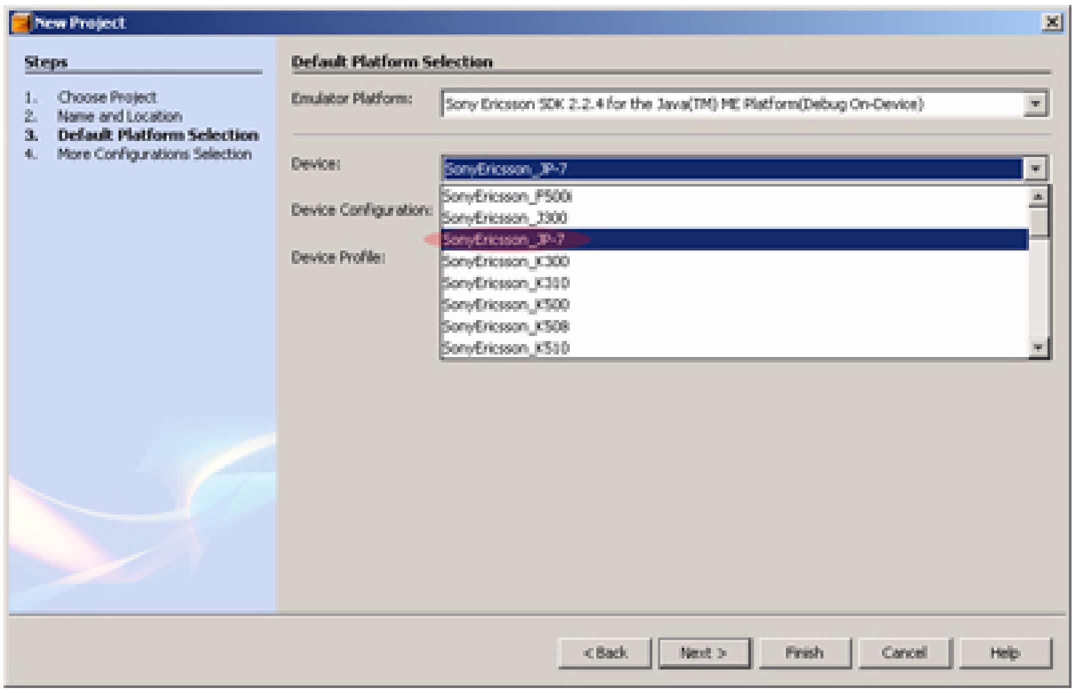
In order to use the correct device profile, we need to select a suitable emulator platform.

Sony Ericsson uses a platformized approach to implementing and supporting Java technology. With the announcement of Sony Ericsson Java Platform 7 (JP-7), the SDK device profiles are based on the Java platform, as opposed to specific phone models as for the earlier Java platforms. For example, a generic SonyEricsson_JP-7 profile can be used for all JP-7 phone models.

The SDK provides a serial proxy, unrelated to the debug agent, allowing communication between the host and the phone. When using phones prior to JP-7, the correct phone-specific profile must be selected to allow the serial proxy to successfully communicate with the phone.

The grouping of phone models on different Java platforms is described in the Developer Guidelines.

In our example we use a K800, which is JP-7 phone, and we select the generic SonyEricsson_JP-7 profile for the emulation:


New Project
Steps
1. Choose Project
2. Name and Location
3. Default Platform Selection
4. More Configurations Selection
Default Platform Selection
< Back
Next >
Finish
Cancel
Help

Emulator Platform: Sony Ericsson SDK 2.2.4 for the Java(TM) ME Platform(Debug On-Device)

Device: SonyEricsson_JP-7

Device Configuration:

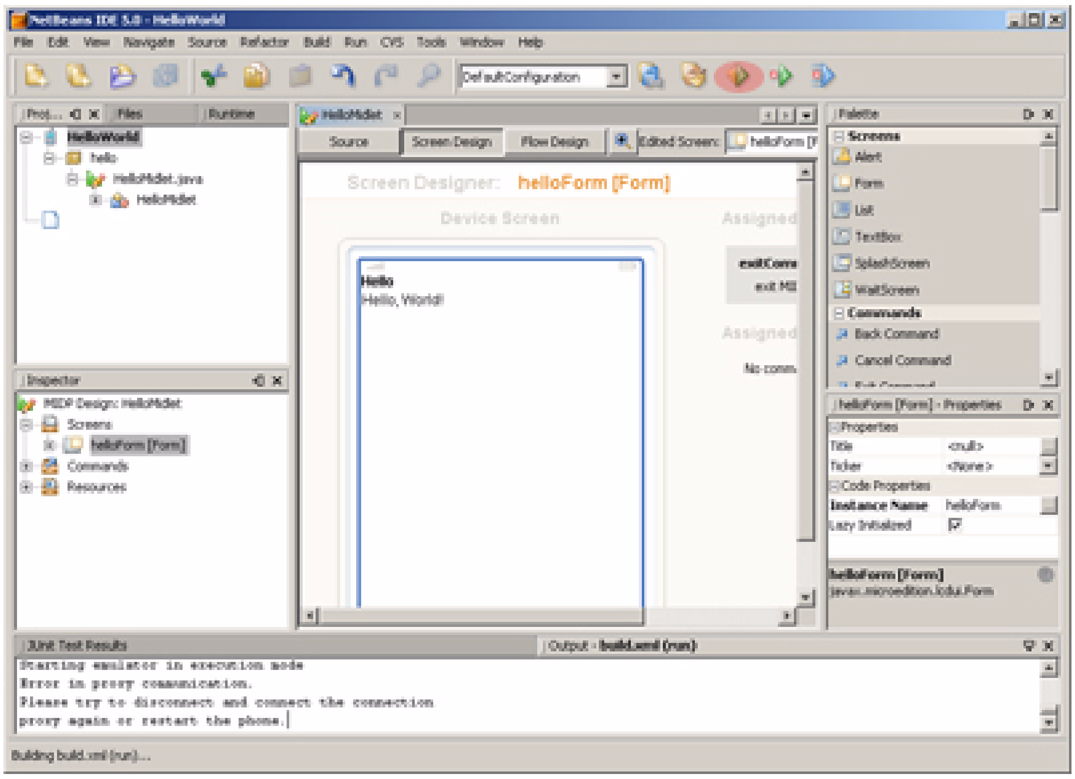
Device Profile:

SonyEricsson_P500i
SonyEricsson_J300
SonyEricsson_JP-7
SonyEricsson_K300
SonyEricsson_K310
SonyEricsson_K500
SonyEricsson_K508
SonyEricsson_K510

Click the "Finish" button and the new “HelloWorld” MIDlet is created.

Testing the connection

Initially we can start by ensuring that the phone is correctly connected and available. Click the "Run Main Project" button in the toolbar:



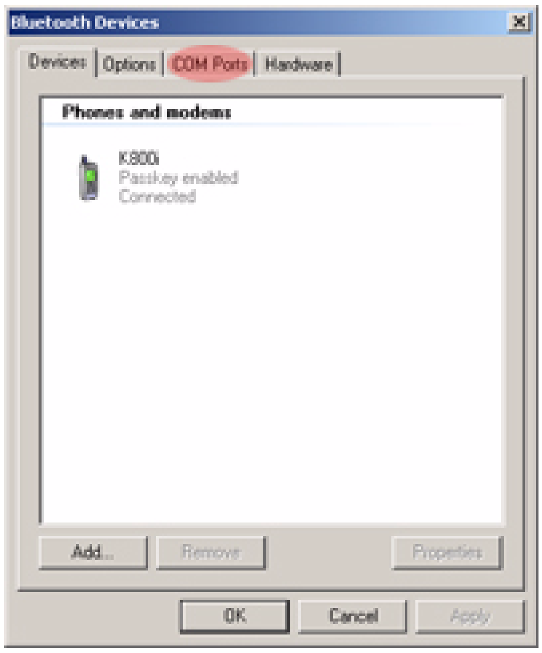
```
Starting emulator in execution mode
Error in proxy communication.
Please try to disconnect and connect the connection
proxy again or restart the phone.
```

NetBeans uses UEI to launch what it believes to be an emulator. The Sony Ericsson SDK attempts to launch the connection proxy, connect to a phone and deploy the MIDlet. Since no phone is currently connected, the launch fails and the connection proxy remains in the disconnected state. We need to tweak the settings to reflect the COM port the phone is connected to.

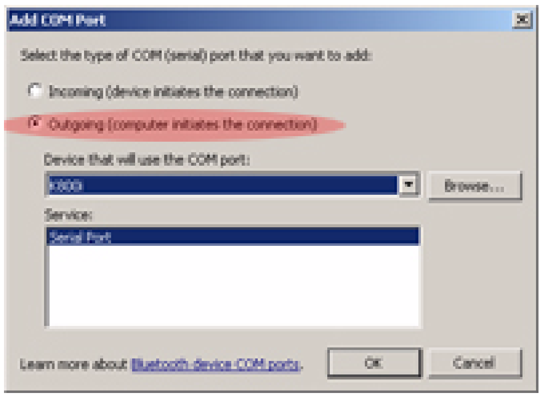
In this example we will use a serial connection over Bluetooth to connect the phone.

Note: The Serial Port Profile (SPP), a Bluetooth profile specification, is very convenient since it allows working with the phone without needing to install any drivers.

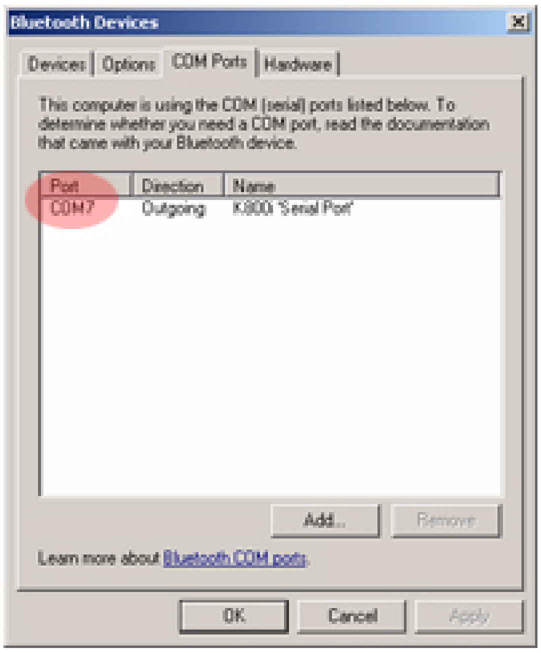
Whilst your Bluetooth stack software usually provides its own UI, we use the default Windows support to illustrate creating a serial connection. From the Control Panel, use the "Bluetooth Devices" applet. Assuming you have paired the device, the “Devices” field should contain your phone:



Under the COM ports tab, click the "Add" button and create a new outgoing connection:


Add COM Port
Select the type of COM (serial) port that you want to add:
Incoming (device initiates the connection)
Outgoing (computer initiates the connection)
Device that will use the COM port:
K800i
Browse...
Service:
Serial Port
Learn more about Bluetooth device COM ports.
OK
Cancel

A new COM port will be assigned:


Bluetooth Devices

Devices | Options | COM Ports | Hardware

This computer is using the COM (serial) ports listed below. To determine whether you need a COM port, read the documentation that came with your Bluetooth device.

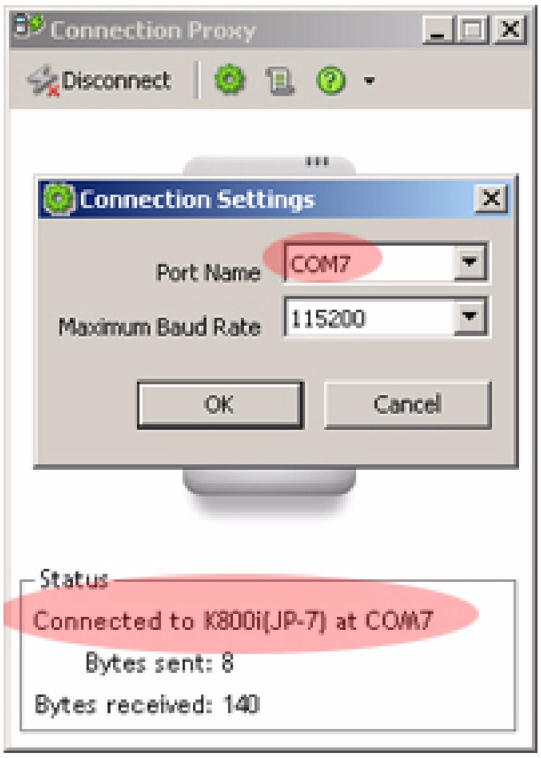
Port	Direction	Name
COM7	Outgoing	K800i 'Serial Port'

Add... Remove

Learn more about Bluetooth COM ports.

OK Cancel Apply

We proceed to use this new COM port in the connection proxy. Click the “Settings” button in the toolbar to select the freshly assigned COM port, and then click “Connect”. After connecting, the Connection Settings appear like this:


Connection Proxy

Disconnect

Connection Settings

Port Name: COM7

Maximum Baud Rate: 115200

OK Cancel

Status

Connected to K800i(JP-7) at COM7

Bytes sent: 8

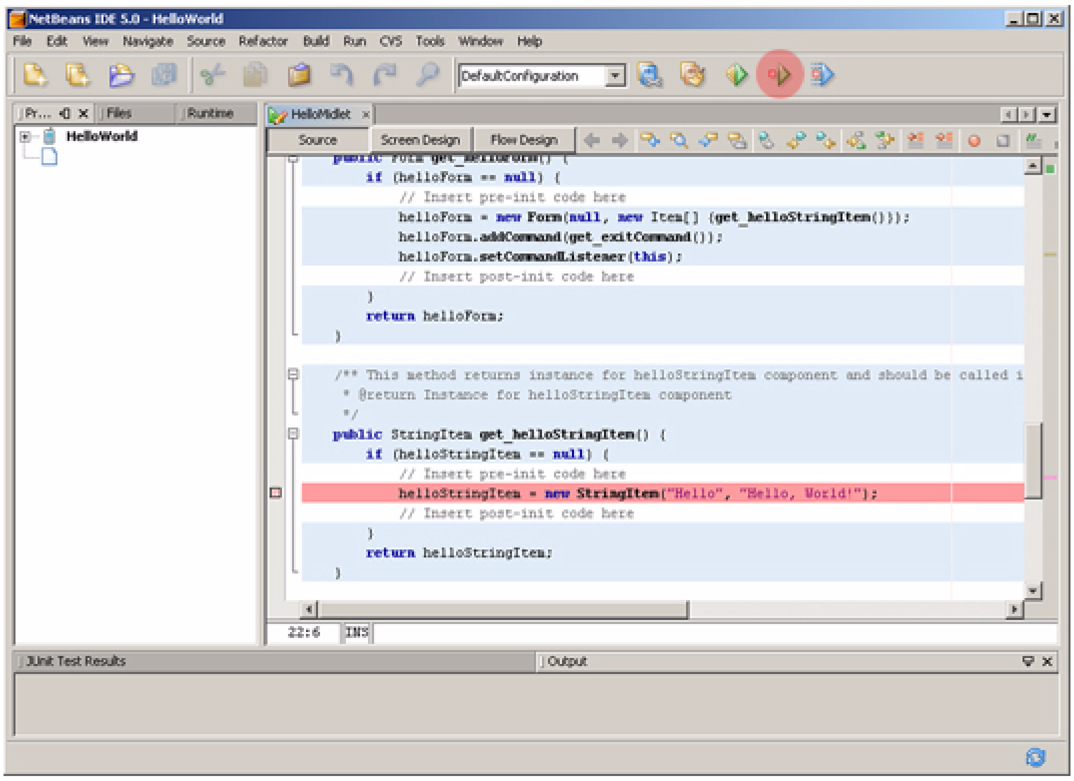
Bytes received: 140

Setting a breakpoint

A breakpoint allows setting a dynamic point in the program where execution can be paused or cancelled and the MIDlet behaviour examined.

Using the example created by NetBeans, we set a breakpoint on the creation of a new `helloStringItem` instance. We set a breakpoint by pressing CTRL-F8.

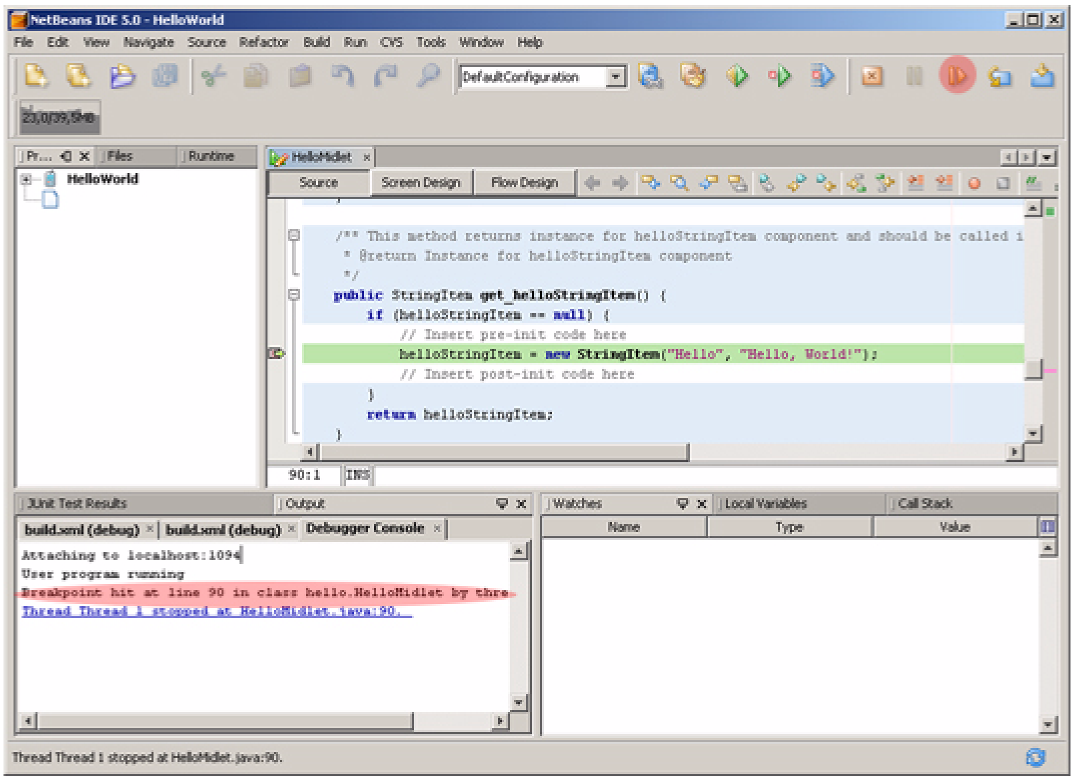
The MIDlet can be debugged on the phone by clicking the "Debug Main Project" button in the toolbar:


NetBeans IDE 5.0 - HelloWorld
File Edit View Navigate Source Refactor Build Run CVS Tools Window Help
DefaultConfiguration
Pr... Files Runtime
HelloWorld
HelloMidlet
Source Screen Design Flow Design
22:6 INS
JUnit Test Results
Output

```
    public Form get_helloForm() {
        if (helloForm == null) {
            // Insert pre-init code here
            helloForm = new Form(null, new Item[] {get_helloStringItem()});
            helloForm.addCommand(get_exitCommand());
            helloForm.setCommandListener(this);
            // Insert post-init code here
        }
        return helloForm;
    }

    /** This method returns instance for helloStringItem component and should be called i
     * @return Instance for helloStringItem component
     */
    public StringItem get_helloStringItem() {
        if (helloStringItem == null) {
            // Insert pre-init code here
            helloStringItem = new StringItem("Hello", "Hello, World!");
            // Insert post-init code here
        }
        return helloStringItem;
    }
```

The MIDlet is compiled and deployed to the phone. Upon execution we quickly hit our breakpoint and execution is suspended:



```
    /** This method returns instance for helloStringItem component and should be called i
     * @return Instance for helloStringItem component
     */
    public StringItem get_helloStringItem() {
        if (helloStringItem == null) {
            // Insert pre-init code here
            helloStringItem = new StringItem("Hello", "Hello, World!");
            // Insert post-init code here
        }
        return helloStringItem;
    }
```

```
Attaching to localhost:1094
User program running
Breakpoint hit at line 90 in class hello.HelloMidlet by thre
Thread Thread 1 stopped at HelloMidlet.java:90.
```

Thread Thread 1 stopped at HelloMidlet.java:90.

Pressing CTRL-F5, "Continue button", resumes execution.
Alternatively, we can step over or into breakpoints using F8 or F7 buttons respectively.

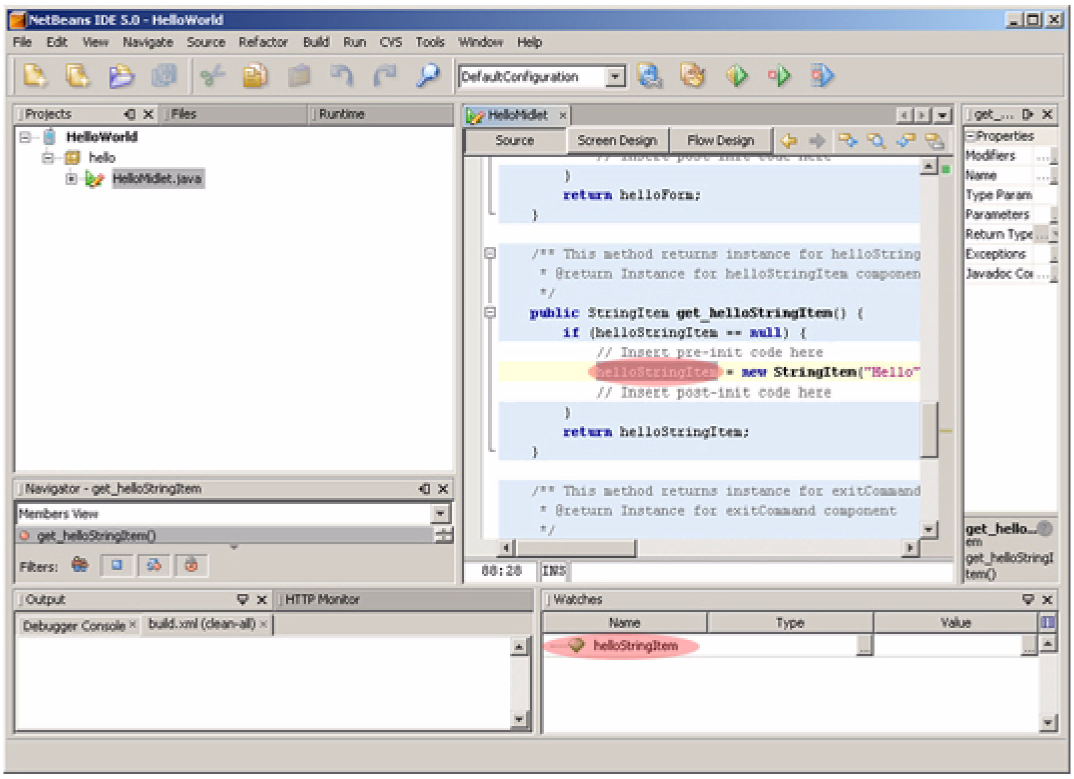
Note: Typically an initially re-occurring message of "99%" or "VM not ready" appears when launching a debugging session. The phone VM runs in a highly optimized state and when debugging, dynamically reconfigures itself to allow for debugging leading to a delay in initiation of the session.

Setting a watch

A watch refers to a variable or expression that is continuously monitored throughout execution.

We reuse the HelloWorld application to create a trivial watch. Select the `helloStringItem` variable and assign a watch to it by pressing CTRL-Shift-F7. The variable appears in the “Watches” window where its state can be monitored.

To allow us to see the contents of the variable before execution proceeds, we need to set a breakpoint on the return of the method:

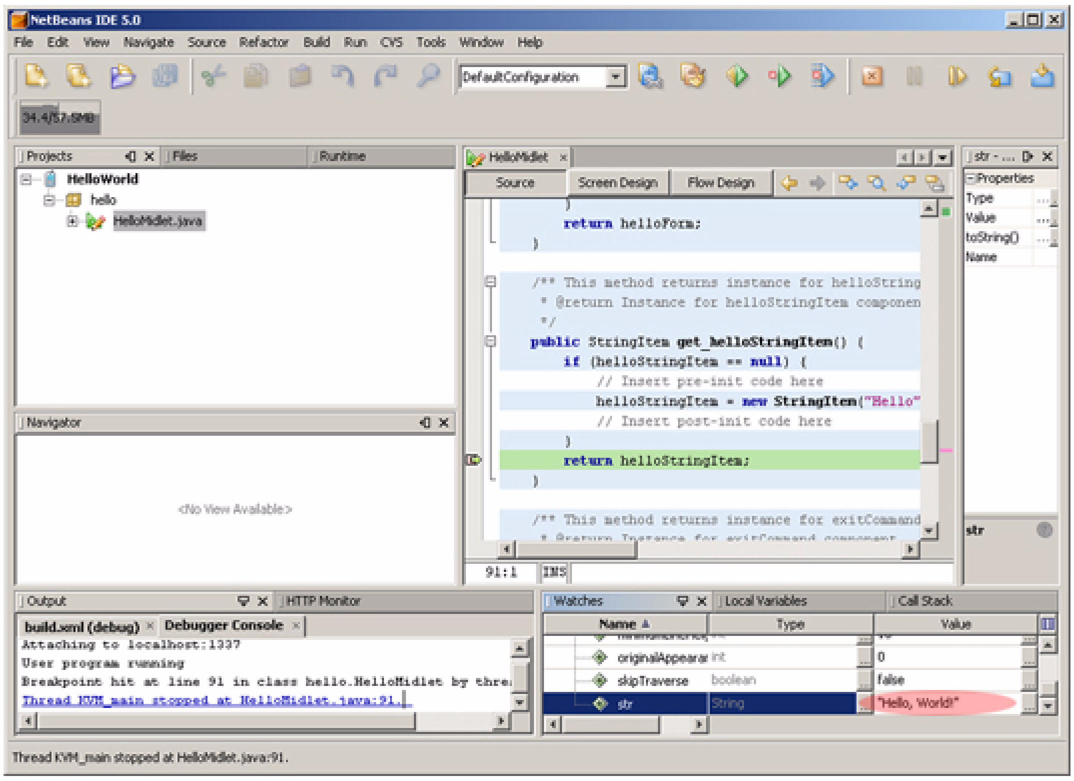


```
        }
        return helloForm;
    }

    /** This method returns instance for helloString
     * @return Instance for helloStringItem componen
     */
    public StringItem get_helloStringItem() {
        if (helloStringItem == null) {
            // Insert pre-init code here
            helloStringItem = new StringItem("Hello"
            // Insert post-init code here
        }
        return helloStringItem;
    }

    /** This method returns instance for exitCommand
     * @return Instance for exitCommand component
     */
```

When debugging the application execution will halt at the return of `helloStringItem`. The value of the watch will change in the "Watches" window. The variable can be expanded to show the value assigned and other attributes:



```
        }
        return helloForm;
    }

    /** This method returns instance for helloString
     * @return Instance for helloStringItem componen
     */
    public StringItem get_helloStringItem() {
        if (helloStringItem == null) {
            // Insert pre-init code here
            helloStringItem = new StringItem("Hello"
            // Insert post-init code here
        }
        return helloStringItem;
    }

    /** This method returns instance for exitCommand
```

```
Attaching to localhost:1337
User program running
Breakpoint hit at line 91 in class hello.HelloMidlet by threa
Thread KVM_main stopped at HelloMidlet.java:91.
```

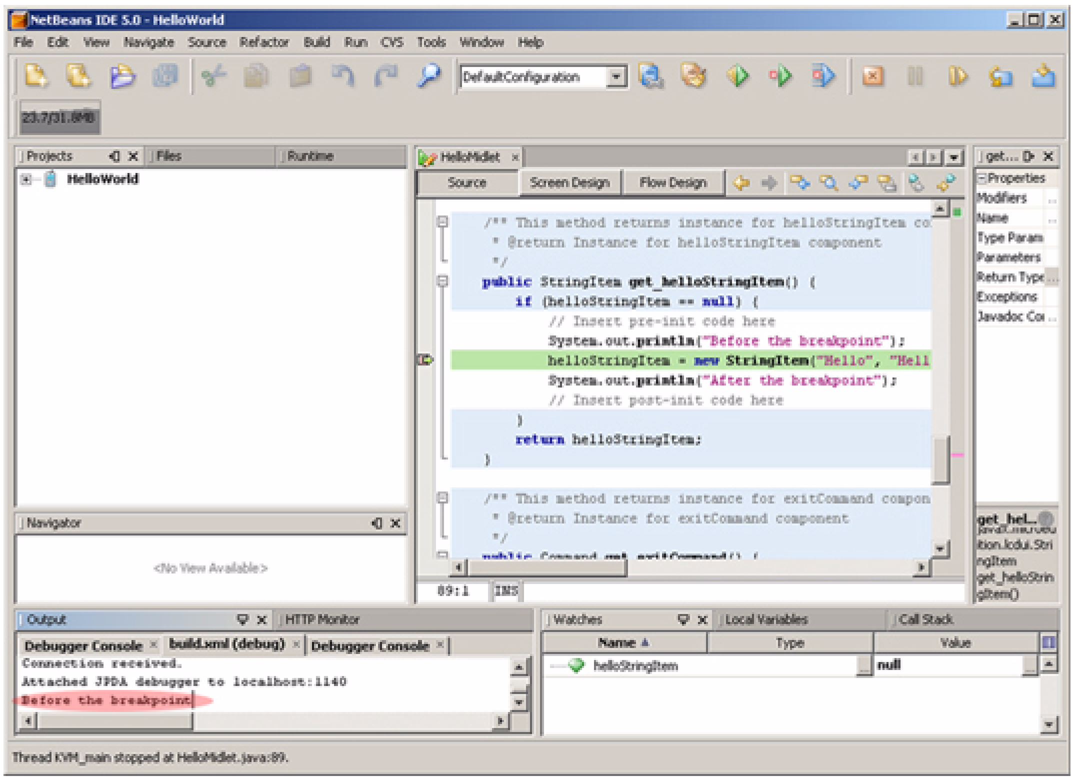
Name	Type	Value
originalAppearar	int	0
skipTraverse	boolean	false
str	String	"Hello, World!"

Thread KVM_main stopped at HelloMidlet.java:91.

Redirection of output streams

Though not specifically related to on-device debugging, a convenient feature that is also exposed, is the redirection of standard output and error output streams.

Typically if an exception occurs while running on the phone VM, determining the cause usually involves resorting to use alert dialogs or screens. By redirecting output streams, errors are immediately apparent:


NetBeans IDE 5.0 - HelloWorld
File Edit View Navigate Source Refactor Build Run CVS Tools Window Help
DefaultConfiguration
Projects Files Runtime
HelloWorld
Navigator
<No View Available>
HelloMidlet
Source Screen Design Flow Design
Properties
Output HTTP Monitor
Debugger Console build.xml (debug) Debugger Console
Watches Local Variables Call Stack

```
    /** This method returns instance for helloStringItem co
     * @return Instance for helloStringItem component
     */
    public StringItem get_helloStringItem() {
        if (helloStringItem == null) {
            // Insert pre-init code here
            System.out.println("Before the breakpoint");
            helloStringItem = new StringItem("Hello", "Hell
            System.out.println("After the breakpoint");
            // Insert post-init code here
        }
        return helloStringItem;
    }

    /** This method returns instance for exitCommand compon
     * @return Instance for exitCommand component
     */
```

```
Connection received.
Attached JPDA debugger to localhost:1140
Before the breakpoint
```

Name	Type	Value
helloStringItem		null

Thread KVM_main stopped at HelloMidlet.java:89.

Conclusion

In many instances, simply having a redirected output to the host is invaluable and is a proven and familiar way of bug hunting. However, as the number of phones and possible implementations increases, together with the evolution of the Java ME platform, on-device debugging becomes an increasingly useful tool to be familiar with. Together with NetBeans 5.0 and Mobility Pack 5.0, it is easily accessible to beginners and experts alike.

Further reading

- Home of the Java Platform Debugger Architecture (JPDA)
  http://java.sun.com/products/jpda

- The Unified Emulator Interface (UEI) board and latest specifications and discussions
  https://uei.dev.java.net/.

- Sony Ericsson SDK 2.2.4 for the Java ME Platform
  http://developer.sonyericsson.com/site/global/docstools/java/p_java.jsp

- Home of NetBeans 5.0 with Mobility Pack 5.0
  http://www.netbeans.org/

- Sony Ericsson and NetBeans presented on-device debugging at JavaOne 2006. The session Power-Point is available online (Tools, TS-5454)
  http://java.sun.com/javaone/sf/sessions/main_session_pdfs.jsp

- Sony Ericsson Developer guidelines cover everything that is Sony Ericsson specific and is invaluable before and during development
  http://developer.sonyericsson.com/site/global/docstools/java/p_java.jsp

- Overview of the Sony Ericsson Java Platform strategy
  http://developer.sonyericsson.com/site/global/newsandevents/latestnews/newsmar05/p_platform_strategy.jsp