VAS KVM Emulator User's Guide

Document Number: VAS-JAVA-1500
Version: [0.1]
Date: [28/10/2002]
Reference: [BENQ/GSM]

Author: Kuan Hung
Approval: Kuan Hung

BenQ *Enjoyment Matters, Formerly Acer Communications & Multimedia Inc*

DOCUMENT REVISION HISTORY				
ISSUE	**DATE**	**SMSTUS**	**AUTHOR**	**COMMENTS**
0.1	28/10/2002	Create	Kuan Hung	Create

Content

Note

1. Describe the general interface that VAS SW architecture support

1 Glossary

1.1 Abbreviation

VAS	Value Added Service
J2ME	Java 2 Platform, Micro Edition
KVM	Kilobyte Virtual Machine
CLDC	Connected, Limited Device Configuration
MIDP	Mobile Information Device Profile
RI	Reference Implementation
JSR	Java Specification Requests
TCK	Technology Compatibility Kit

2 Objectives

The purpose of this document is to describe the KVM Emulator of BenQ. This document will describe how to setup the emulator, how to use the emulator, and we will also list all the extend APIs which could to been used in BenQ's emulator.

3 Introduction

The emulator is to enables user to develop MIDlet applications which could run at BenQ's cell phone.

4 Setup the Emulator

4.1 System Requirement

The emulator can only run on Windows NT, 2000 or later. That means you need Windows NT, 2000 or later windows platform to run the Emulator.

4.2 Program Installation

Please unzip the package and execute the setup file (setup.exe). The installation wizard will help you complete the entire installation process.

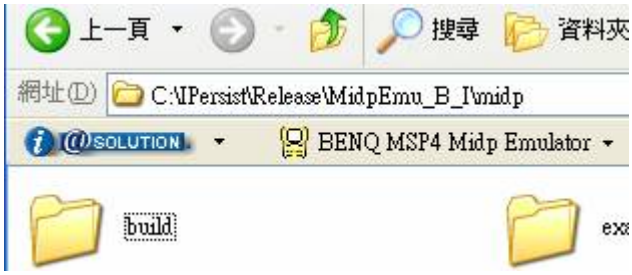
Note:

Please don't install the emulator at the folder with Chinese path name because current emulator cannot support Chinese.

4.3 How to use the emulator

After installation, the JAD file will be associated with this emulator's profile manager (named JadManager). The manager will control the behavior of the emulator. You can use option setup tool to change the settings of the manager. To change the options of emulator, you must open system File Explorer or Internet Explorer and press right key on the toolbar of the explorer

After this, choice “iaSolution CKBand Class”. The JAD manager's toolbar will present. You can change the skins of emulator by pressing the dropped menu. (Default emulator is BENQ MSP4).



5 Running the Emulator

There are two batch files at the following path folder. They are *_run_b_.bat* and *_run_i_b.bat* in <emulator_path>\midp\build\win32. To setup parameters for emulator, you have to modify the bath files. Open the files you may find something like this:

_run_b_.bat

```
@echo off
bin\midp_b %4 %5 -skin skins\%1 -classpath classes;%3 -Xdescriptor %2
```

_run_i_b.bat

```
@echo off
bin\midp_b %3 %4 %5 -skin skins\%1 -classpath classes -transient %2
```

You can modify the batch file as what you want but don't change the batch file parameters variables. The shell supporting module heavily depends on these batch files. For example, you can run the emulator in debug mode by this way.

```
@echo off
bin\midp_b -debugger %4 %5 -skin skins\%1 -classpath classes;%3
-Xdescriptor %2
```

BenQ's emulator skin is "skin05"; you can use the parameter to run the emulator.

For example:

<emulator_path>\midp\build\win32>_run_b_.bat

skin05 ..\..\example\auction.jad ..\..\example\auction.jar

The result is as following:

6 Virtual Machine Trace Options

Following is a list of trace options available in KVM Emulator which can be enabled to assist the debugging session of MIDP application development.

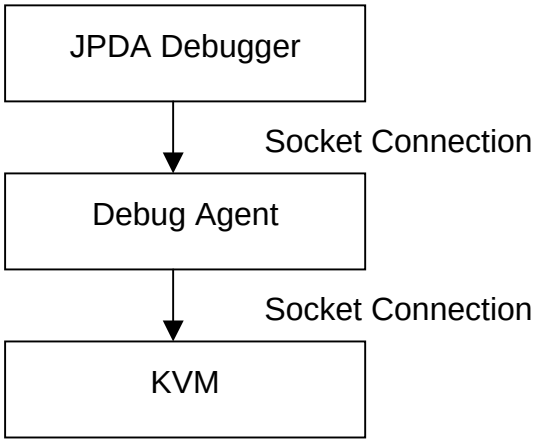
All these options can be configured through the on screen menu bar (as shown in following figure).

Trace option	Meaning
traceallocation	trace memory allocation
tracedebugger	trace the debugging interface
tracegc	trace garbage collection
tracegcverbose	trace garbage collection, more verbose
traceclassloading	trace class loading
traceclassloadingverbose	trace class loading, more verbose
traceverifier	trace class file verifier
tracestackmaps	trace the behavior of stack maps
tracebytecodes	trace bytecode execution
tracemethods	trace method calls
tracemethodsverbose	trace method calls, more verbose
traceframes	trace stack frames
tracestackchunks	trace the allocation of new stack chunks
traceexceptions	trace exception handling
traceevents	trace the behavior of the event system
tracethreading	trace the behavior of the multithreading system
tracemonitors	trace the behavior of monitor objects
tracenetworking	trace the network access
traceall	activates all the tracing options above simultaneously

7 Debugging MIDP Application with Emulator and BORLAND JBuilder

7.1 Concept

KVM implements KDWP(KVM Debug Wire Protocol) for Java-Level debugging support. KDWP is a strict subset of JDWP(Java Debug Wire Protocol) because of the memory constraints of KVM. As a result, a debug agent(debug proxy) is required for acting as a bridge between KVM and the JPDA(Java Platform Debug Architecture) which looks like this:



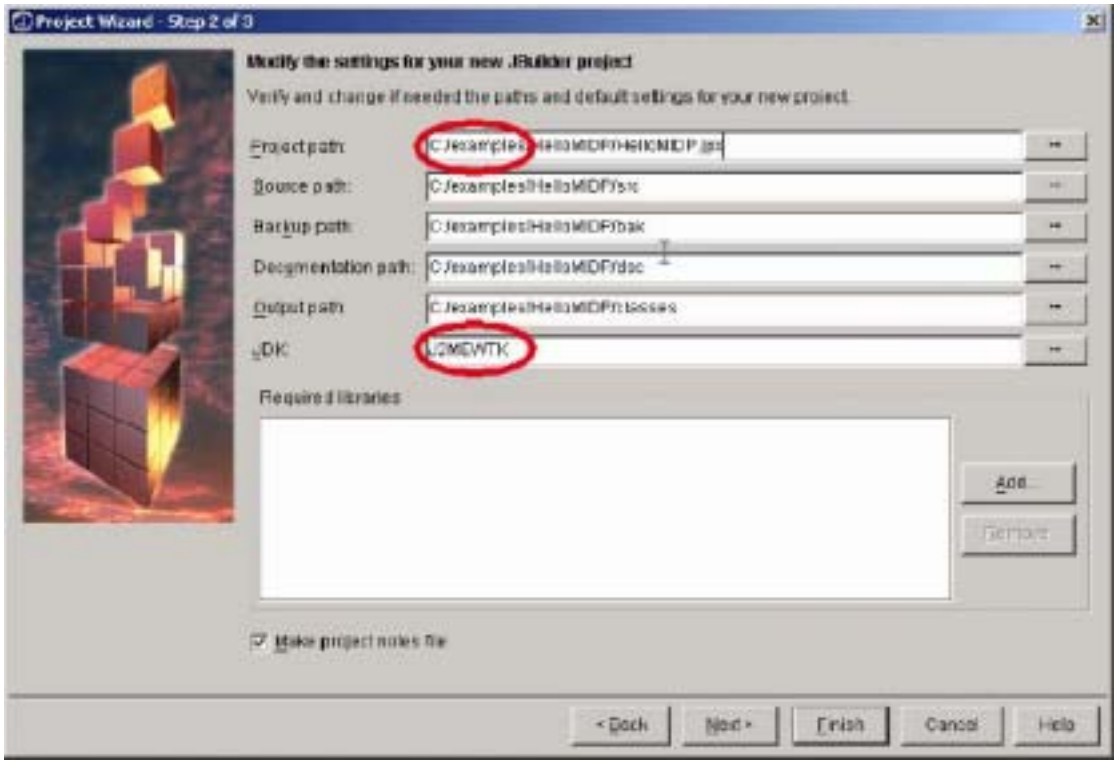
If you have ever used the Borland MobileSet along with Sun Wireless Toolkit 1.0.4, you would find the command line parameters of Sun Wireless Toolkit Emulator(emulator.exe) is different with those of MIDP Reference Implementation Emulator(midp.exe). Consequently, it is impossible to use the MIDP Reference Implementation (RI) Emulator directly with Borland MobileSet without modifying the source code of MIDP RI.

In the rest of the document, we use a step-by-step example to use the Java-Level debugging with JBuilder5 and MDIP RI in an alternative method. KVM Emulator

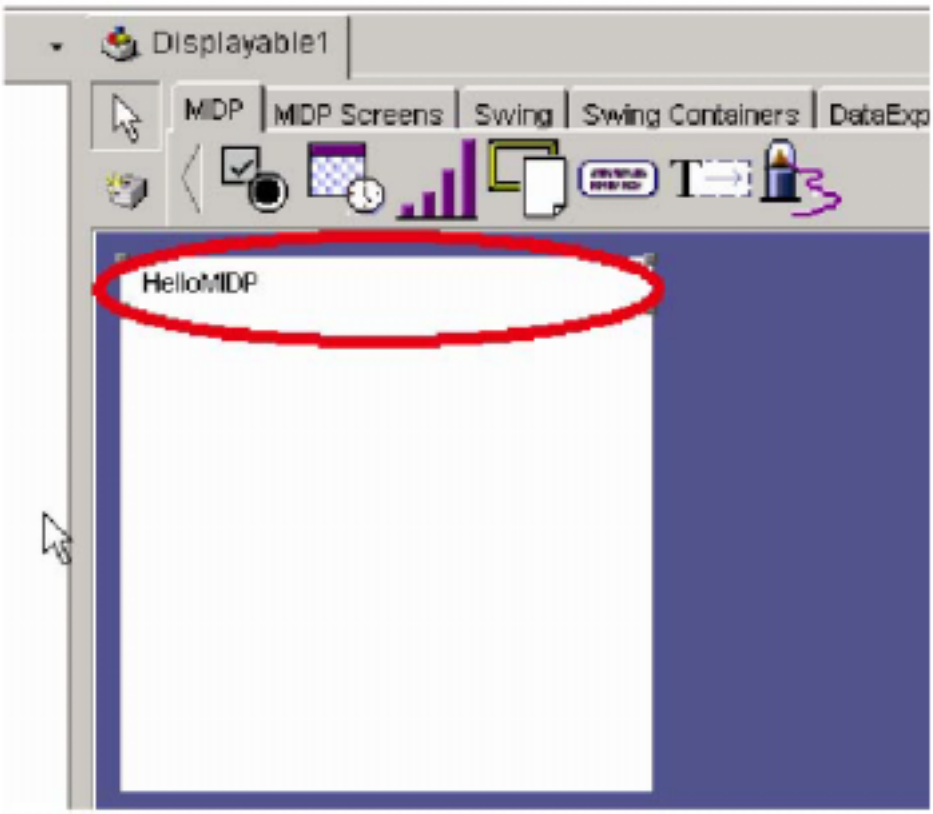
7.2 An example

7.2.1 Creating a HelloMIDP project

In the example, we put the project in ***c:/examples***. And choose the Sun Wireless Toolkit as the Project JDK.



Create a new MIDlet and put a Form on it. Next, we put a StringItem with text "HelloMIDP".



7.2.2 Launching the MIDlet

If we press F9 to run or Shift-F9 to begin debug, it will use the Sun WTK to run the project. But it is not our intention. Instead, we should launch the project manually, and then use the debugger of JBuilder5 to debug. First, we should locate the emulator to use. In the example, the emulator is put on ***c:/midp***.

Before you run the project, remember to build the project with JBuilder5 which will help you to compile and preverify the project and generate a JAD file. The generated file is located on ***[project_directory]/preverified-temp***, in our example the files are put on ***c:/examples/HelloMIDP/preverified-temp***.

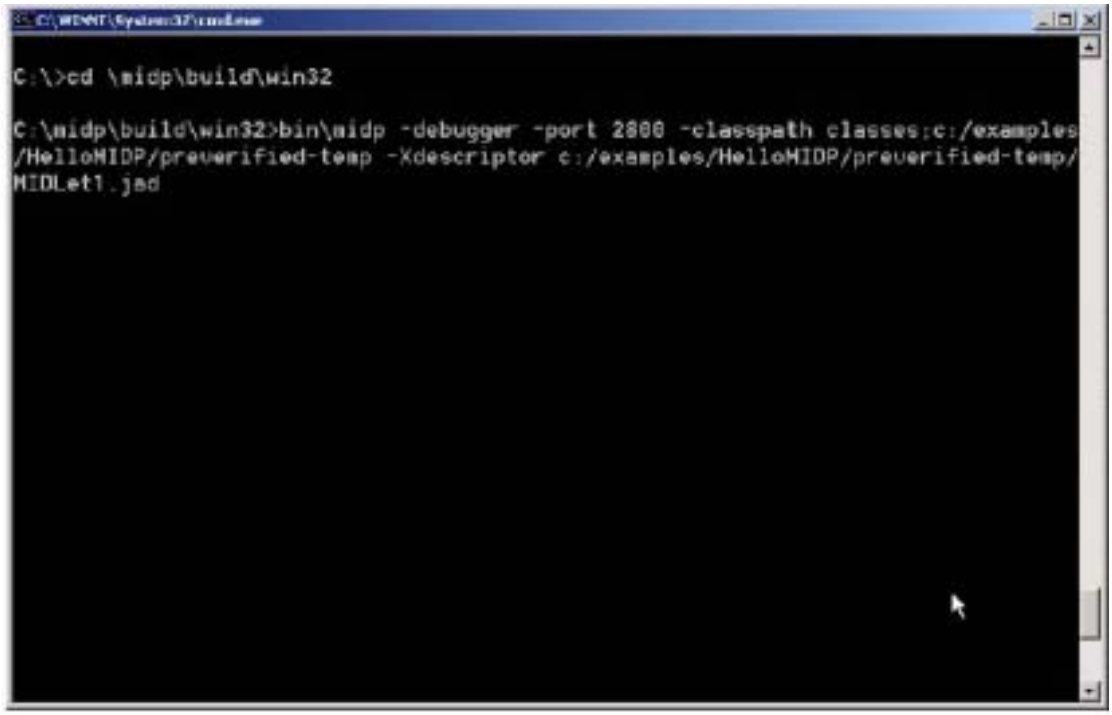
Open a command line console, change the current directory to ***c:/midp/build/win32.***

Finally, Launch the MIDlet by the command:

bin\midp –debugger –port 2800 –classpath classes;c:/examples/HelloMIDP/preverified-temp –Xdescriptor c:/examples/HelloMIDP/preverified-temp/MIDLet1.jad

where the options –**debugger** and –**port 2800** are used to indicate the usage of debugger on port 2800.

To get more information of the options of **midp.exe**, you can type ***midp –help***.



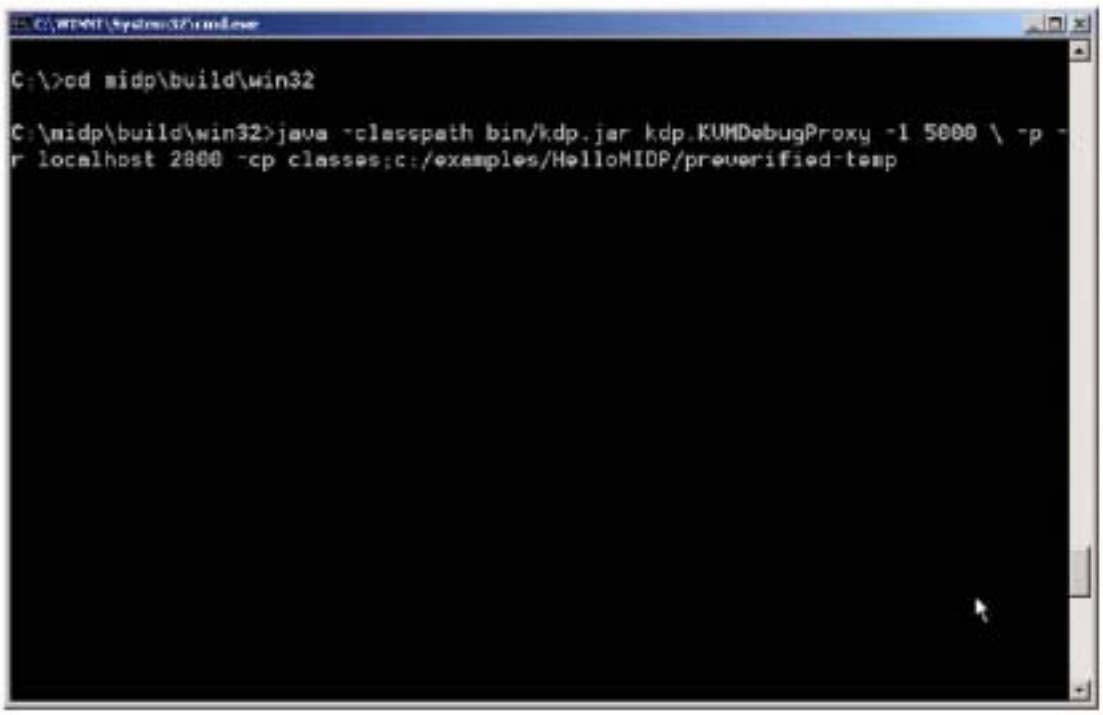
After execute the command the command, you can see a handset emulator has been launched.

7.2.3 Launching KVM debug proxy

Open another command line console, and execute the following command to launch the KVM Debug Proxy:

java –classpath bin/kdp.jar kdp.KVMDebugProxy –l 5000 –p –r localhost 2800 –cp classes;c:/examples/HelloMIDP/preverified-temp

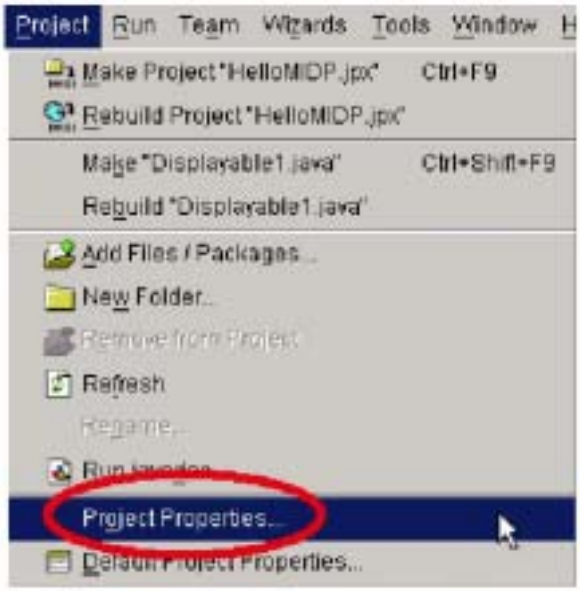
Where the option **–l 5000** means the debug proxy will listen on port 5000 for a JPDA debugger, the option **–r localhost 2800** indicates the debug proxy will connect to localhost on port 2800 to connect to the Java VM running the application being debugged.



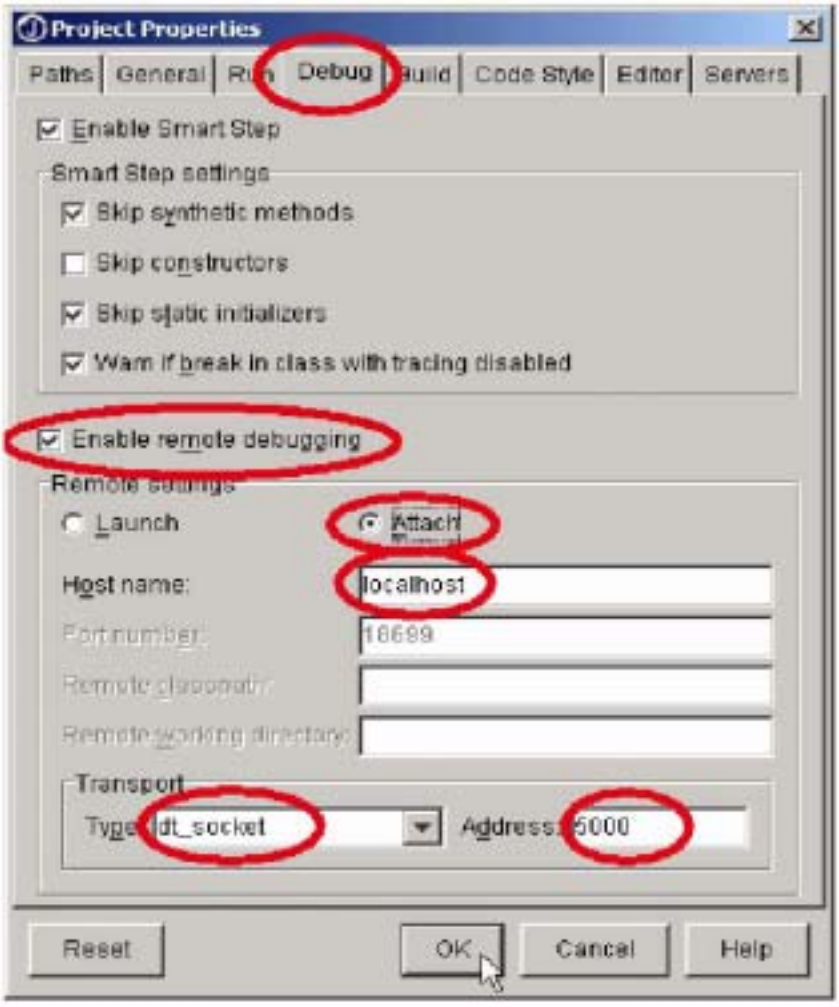
To get more information about the options of KVMDebugProxy, you can type: **java –classpath bin/kdp.jar kdp.KVMDebugProxy** (without any other parameters).

7.2.4 Setting up JBuilder5

Now, you will need to set the project properties in JBuilder5. In **[Project, ProjectProperties…]**, you can modify the project properties.



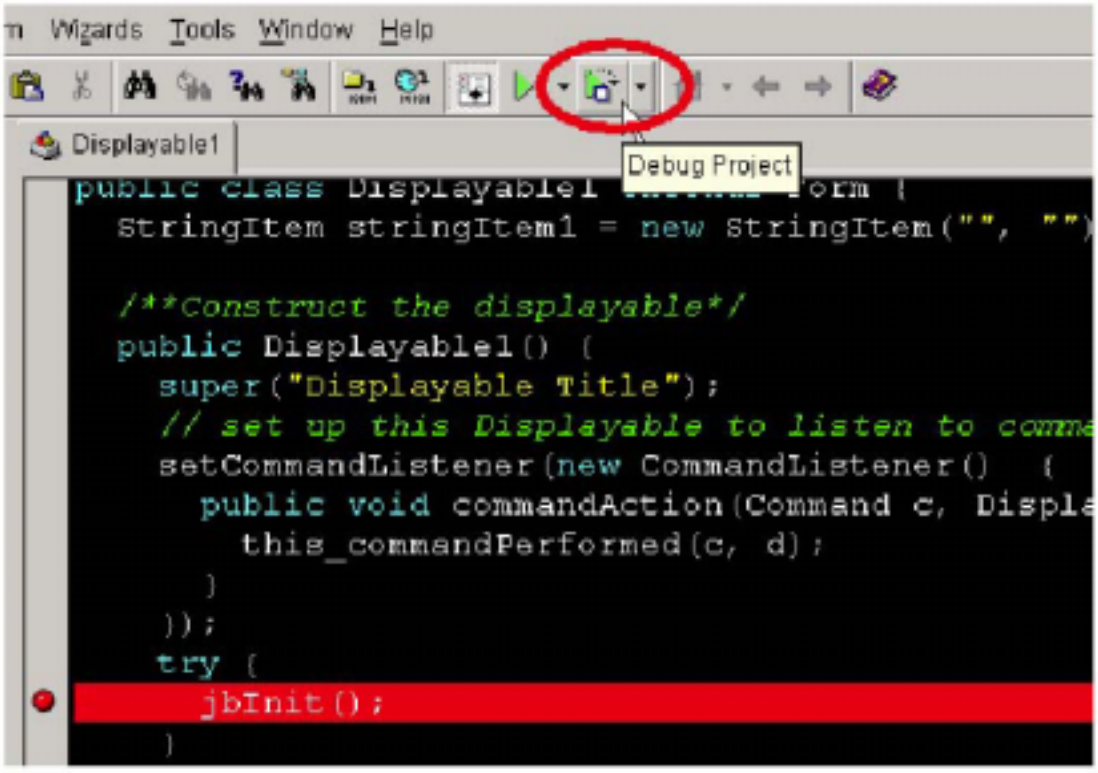
Switch to the **Debug** tab, and check the **Enable remote debugging** options. Then, in **Remote Settings** group, select the approach **Attach**, set the Hostname field to be ***localhost***, the **Transport Type** to be **dt_socket**, and the **Address 5000**.

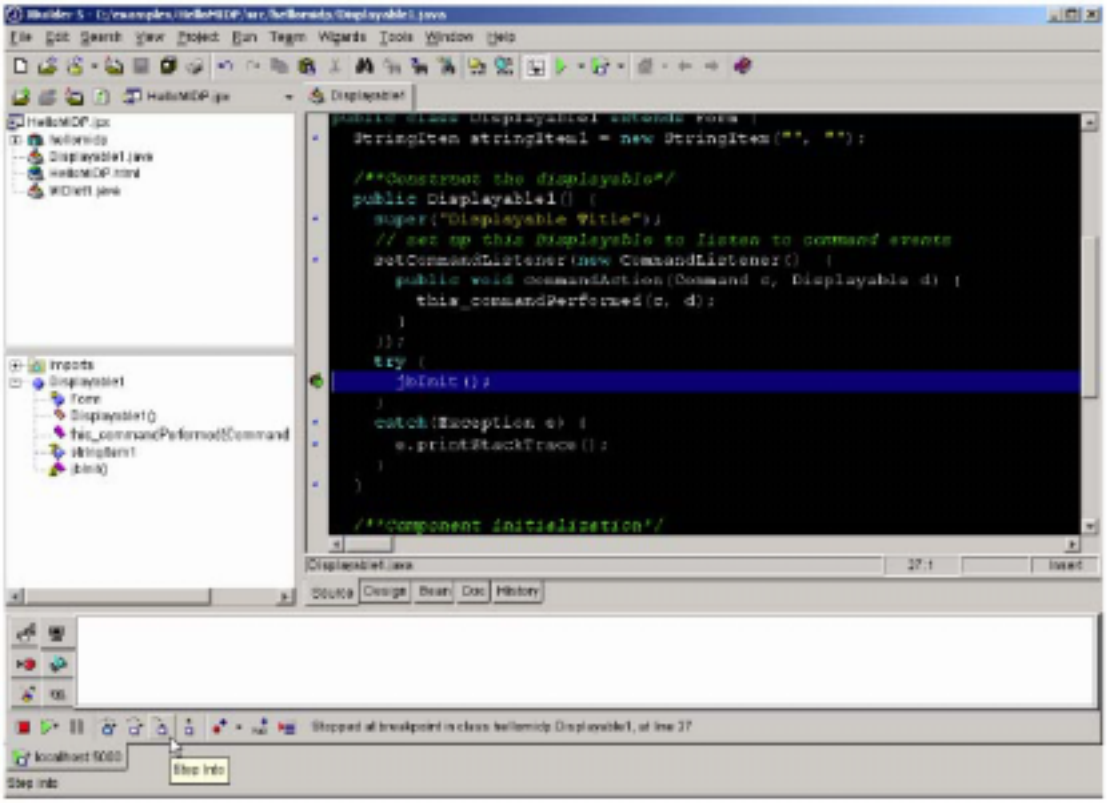


Press OK to apply the new configuration.

7.2.5 Using JBuilder5 to debug

Finally, press the Debug Project button. Then, you can begin to debug your program.





8 Extend API

8.1 JSR 118 Mobile Information Device Profile 2.0

- Network connectivity via sockets and datagrams

HTTP protocol is supported in current implementation to enable both Java OTA Provisioning and application level network access.

- Formal inclusion of OTA Provisioning (i.e., Recommended Practice 1 for MIDP 1.0)

Current implementation of Java Application Manager follows the protocol defined in above document.

- User Interface - extensions to low-level LCDUI to allow greater game functionality and layout control for larger screen sizes

This is the so-called GameAPI which brings the low-level access to MIDlets. Current implementation covers these packages.

- Base sound API

Basic sound support will be supported in current KVM Development Service. Following types of audio playback can be supported: Single Tone, Tone Sequence, Wave Audio playback.

8.2 JSR 120 Wireless Messaging API

Package javax.wireless.messaging

Only the SMS sending classes and methods are supported for now. Due to insufficient information about MT SMS handling, it is not feasible to implement the MT SMS in Java environment for now. Hence we supports only the MO SMS mechanism in Wireless Messaging package currently. Following is a list of supported classes,

javax.wireless.messaging.Message
- getAddress()
- getTimeStamp()
- setAddress()

javax.wireless.messaging.MessageConnection
- newMessage(java.lang.String type)
- (NOTE: ONLY TEXT_MESSAGE SUPPORTED)
- newMessage(java.lang.String type, java.lang.String address)
- numberOfSegments(Message)
- send(Message msg)

For other not supported classes, the class body and method signature will be supported for extension in the future.

8.3 JSR 135 Mobile Media API

JSR 118 and JSR 135 are interoperable APIs which enable seamless sound and multimedia content creation across J2ME range of devices using the same API principles.

As mentioned above in JSR 118 section, audio playback logic are supported including Single Tone, Tone Sequence, Wave Audio playback.

SINGLE TONE AND TONE SEQUENCE

Must provide interface for tone generation and related controls.

WAVE AUDIO

Several requirements should be met to implement wave audio playback capability. Please refer to the specific document for detailed interface required.

9 References

[VAS-JAVA-1300.pdf]
KVM emulator user manual
[VAS-JAVA-0600.pdf]
Debugging MIDP Application With KVM Emulator and BORLAND JBuilder
[VAS-JAVA-xxxx.pdf]
K JAVA Environment Extension Feature Description

[J2ME(TM)-Java 2 Platform, Micro Edition]
http://java.sun.com/j2me/
[Mobile Information Device Profile (MIDP)]
http://java.sun.com/products/midp/
[JSR 118 - Mobile Information Device Profile 2.0]
http://jcp.org/jsr/detail/118.jsp
[JSR 120 - Wireless Messaging API]
http://jcp.org/jsr/detail/120.jsp
[JSR 135 - Mobile Media API]
http://jcp.org/jsr/detail/135.jsp

[Java(TM) 2 Platform Micro Edition, Wireless Toolkit]
http://java.sun.com/products/j2mewtoolkit/