J2ME IDE Comparison

prepared by Michael Taylor

29 June 2002
Version 1.1

Developnet Consulting Limited
The Old School House
Grantley
Ripon
North Yorkshire
HG4 3PJ

+44 (0)7711 571 015

j2me@developnet.co.uk

http://wwww.developnet.co.uk/

Copyright

Warranty

Trademarks

Liability

Change History

Issue Date	Issue Number	Description of Changes
27/05/2002	1.0	Initial draft.
29/06/2002	1.1	Various typos fixed. Updated company logo and front page design. Updated “Forte for Java” terminology to “Sun ONE Studio”.

Table of Contents

Table of Figures

1. Preface

This document provides an in-depth review of market-leading IDEs that provide facilities for J2ME development using MIDP/CLDC.

The document does not intend to provide a comprehensive review of all features each IDE offers. Instead, the focus is on features specifically intended for the J2ME developer.

After introducing J2ME and IDE concepts, the document has extensive product-specific sections detailing the J2ME facilities provided by each IDE.

2. Introduction

2.1. J2ME Overview

The first edition for the Java platform was Java 2 Standard Edition (J2SE), which supported the development of desktop applications. The fundamental idea was to provide a language which supported platform-independent code.

Shortly after, a new edition was released, Java 2 Enterprise Edition (J2EE), providing support for large-scale, enterprise-wide applications.

As we enter the post-PC era, the market for network-connected consumer devices – such as smart mobile phones, televisions, VCRs and PDAs – is expected to grow at a phenomenal rate. In order to provide a compelling Java technology solution for these devices, Sun has introduced the latest addition to the Java platform: Java 2 Micro Edition (J2ME).

Within J2ME, there are two broad categories, known as configurations. Connected Device Configuration (CDC) is a set of APIs to support fixed devices such as a television set-top box. Connected Limited Device Configuration (CLDC) is a set of APIs targeted at devices that have limited processing power, display and memory. The majority of these devices will also be mobile.

On top of configurations are profiles. Profiles provide APIs for user interface design, network support, and persistent storage. The Mobile Information Device Profile (MIDP) is a set of Java APIs which, together with the CLDC, provides a complete J2ME application runtime environment targeted at mobile devices, such as mobile phones, pagers and entry level PDAs.

The focus of this document will be the development of J2ME MIDP applications. A MIDP application is known as a MIDlet. A MIDlet suite consists of one or more MIDlets that are packaged together. The MIDlet suite can be deployed to and then executed on the mobile device.

MIDP applications are generally developed using tools on a development workstation. Sun provide a MIDP/CLDC reference implementation or Java Development Kit (JDK). This includes basic command line development tools and a set of generic emulators that allow for testing of the application without actually deploying it to the device. Some of the mobile device manufacturers also provide access to their own MIDP/CLDC JDKs and device-specific emulators.

2.2. J2ME Development Cycle

Broadly, the development cycle for a MIDP application proceeds as follows:

- **Write the Java code.**
- **Compile.**
- **Obfuscate (optional).** Obfuscation removes extraneous class information, such as local variable names. Classes, methods, interfaces, and such are renamed so as to make them ambiguous. An obfuscated package protects the class files from decompilation and reverse engineering. In addition to protecting the source code, the obfuscation process

reduces the size of the class files resulting in smaller JAR files The reduced size is advantageous because of the limited memory available on MIDP devices.

- **Pre-verify.** The class verifier in J2SE takes a minimum of 50 kilobytes, not including heap space requirements and processing time. To reduce the overhead for J2ME, class file verification has been broken into two phases. Pre-verification before deployment augments the class files with additional attributes to speed up runtime verification. The device itself performs a lightweight runtime verification process using the additional attributes generated during pre-verification.
- **Create the JAR file.**
- **Create the JAD file.**
- **Execute in a suitable emulator.**
- **Deploy to the mobile device.**

2.3. J2ME IDEs

An integrated development environment (IDE) aims to improve developer productivity by providing a cohesive set of developer tools through a graphical user interface (GUI).

The canonical Java IDE is expected to contain an editing tool, a project management tool, a compilation environment, and a debugger. However, as the Java IDE market has matured, the leading vendors have introduced innovative new features to differentiate their products from the competition.

Initially, Java IDEs were targeted at J2SE. However, as J2EE gained acceptance, vendors implemented support for developing enterprise applications using J2EE into their IDEs.

The most recent addition to the Java family is J2ME. With many experts predicting enormous growth in the J2ME applications development market, many vendors have implemented extensions to their existing IDE products to support J2ME. In addition, specialist vendors have implemented standalone J2ME IDEs.

A J2ME IDE is expected to provide the following facilities:

- **Project management.** Managing the source files and MIDlets attributes.
- **Editor.** Editing of source code and resources.
- **Build.** Compilation, obfuscation and pre-verification of source code.
- **Package.** Packaging of MIDlets into JAR and JAD files.
- **Emulation.** Execution of MIDlets within an emulator.
- **Debugger.** Debugging of MIDlets.

3. Sun J2ME Wireless Toolkit 1.0.4 Beta

3.1. Introduction

The J2ME Wireless Toolkit (WTK) is a J2ME Java Development Kit (JDK) that provides application developers with the emulation environment, tools, documentation and examples needed to develop MIDP applications.

The WTK is not a complete IDE, as it omits editing and debugging features that are considered mandatory for an IDE. However, KToolBar, included with the WTK is a minimal development environment with a GUI for compiling, packaging, and executing MIDP applications.

The previous version of the WTK, 1.0.3, offered the option of integration with Sun ONE Studio 3.0 (formerly Forte for Java 3.0). However, this option has been dropped from WTK 1.0.4. Instead, Sun has integrated J2ME support into Sun ONE Studio 4.0 Mobile Edition (which is currently only available via the Sun Early Access Program).

The WTK 1.0.4 release also includes an enhanced emulator with new simulation, monitoring and debugging features. In addition, a new mechanism has been added to KToolBar's build process to enable the seamless integration and execution of a Java byte code obfuscator when packaging a MIDlet suite.

3.2. Pre-requisites

The WTK supports the Windows 98 Second Edition, Windows NT 4.0, and Windows 2000 operating systems. Unsupported versions of the WTK for Solaris and Linux are also available.

The WTK has the following hardware requirements on all platforms:

- 166 MHz CPU
- 64 Mb RAM
- 30 Mb hard disk space

The Java 2 SDK, Standard Edition 1.3 or higher must already be installed before installing the WTK.

3.3. Installation

The WTK is distributed as a single executable and can be downloaded from the following URL:

http://developer.java.sun.com/developer/earlyAccess/j2mewtoolkit/

At the start of installation, the installer searches for a suitable Java 2 SDK. The installer prompts with the destination folder of the Java 2 SDK it has found. This can be overridden if required.

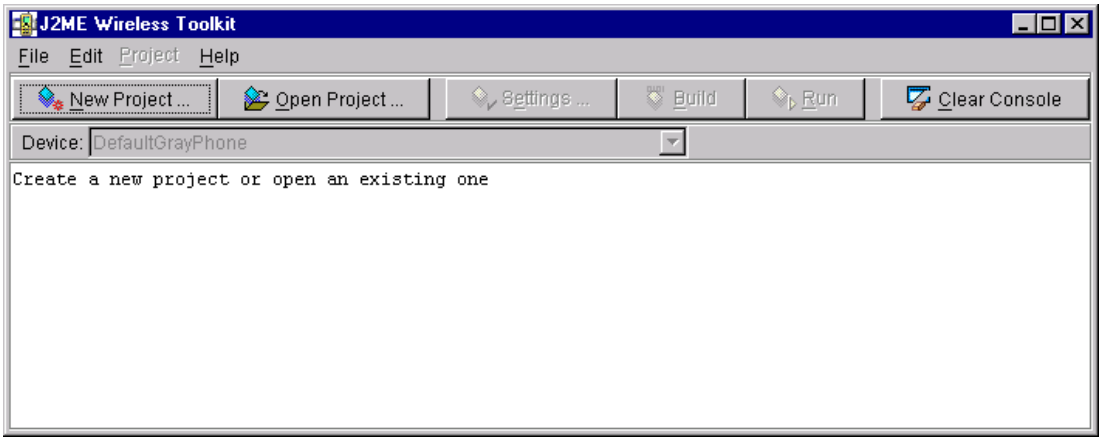
Next, the installer prompts for a destination folder for the WTK itself. This can also be overridden if required.

The WTK is then installed.

3.4. Project Management

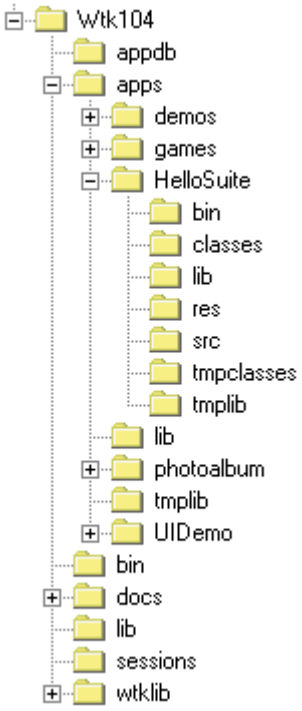
The KToolBar application provides the GUI development environment within the WTK.

Figure 1. WTK KToolBar Application.



A KToolBar project is associated with a MIDlet suite. Within KToolBar the usual facilities are provided for creating new projects and opening existing projects. In addition, MIDlet suite attributes can be modified. These attributes are used during packaging, which results in the creation of JAR and JAD files.

Figure 2. WTK Project Directory Structure.



Each project has its own directory within the `apps` directory. The `apps` directory is located within the main WTK directory.

External class libraries cannot be configured within KToolBar. Instead, class libraries required for a specific project should be manually copied to the `lib` directory within the project directory. If a class library is required for all projects, it can be copied into the `apps/lib` directory.

When a KToolBar project is created, the project directory structure is set up. In addition, a JAD file and manifest file is generated based on the MIDlet suite attributes specified. However, no skeleton source files are generated for the MIDlet. It is up to the developer to write these from scratch themselves.

The KToolBar application does not support multiple projects loaded within the GUI.

3.5. Editor

The WTK does not provide an integrated editor. Instead, a third-party editor should be used.

3.6. Build/Package

The KToolBar application provides integrated facilities to build the project. The project is compiled and pre-verified in one sequence.

A separate facility is provided to create a package of the project files ready for deployment. The package contains a JAR and JAD file for the MIDlet suite.

The WTK contains a plug-in framework for the creation of obfuscated packages via the RetroGuard code obfuscator. The RetroGuard product is not provided as part of the WTK, but can be downloaded, at no charge, from http://www.retrologic.com/. Installation is simple: the `retroguard.jar` file is copied to the `bin` directory within the main WTK directory.

If another code obfuscator is preferred, a plug-in must be implemented. Otherwise, code obfuscation can be performed as a manual process, outside the KToolBar application, before packaging.

3.7. Emulation

The WTK includes an integrated emulator for running MIDP applications.

Figure 3. WTK Emulator.

The emulator uses Sun's MIDP/CLDC reference implementation. Therefore, look-and-feel of the screen UI is based on the reference implementation and not on a specific manufacturers device.

In addition, a sample implementation of an application manager is provided, which supports the installation, maintenance, and removal of applications on a device. However, this sample implementation does not emulate any application manager in particular, as the behaviour of application managers varies from device to device.

The following devices are supported:

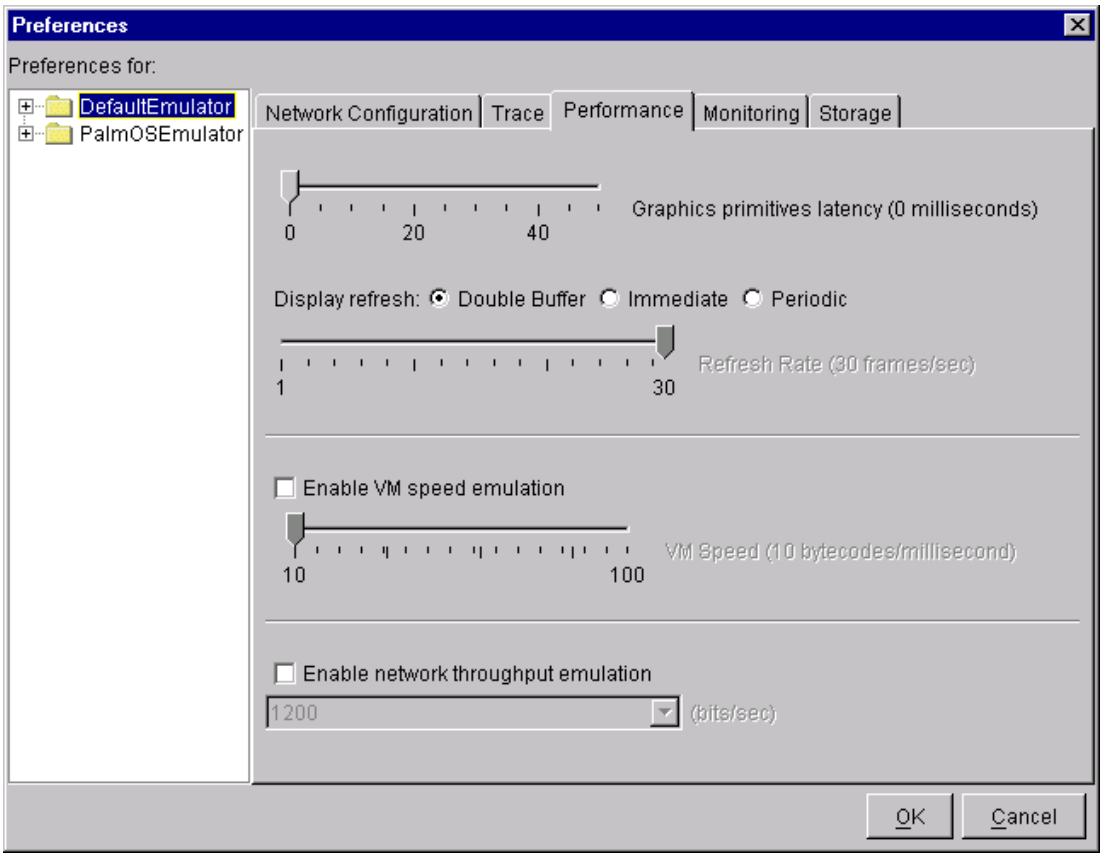
- **DefaultColorPhone.** Generic telephone with a colour display.
- **DefaultGrayPhone.** Generic telephone with a grey-scale display.
- **MinimumPhone.** Generic telephone with minimum display capabilities.
- **RIMJavaHandheld.** RIM device from Research In Motion Ltd.
- **Motorola_i85s.** Motorola i85s telephone from Motorola, Inc.
- **PalmOS_Device.** Palm OS personal digital assistant from Palm, Inc. (The emulation uses the Palm OS Emulator from Palm, Inc.)

It is also possible to customise the emulator for specific devices. The customisations cover the size of the screen that is emulated, the location of image files used to display the device,

and the active areas within these images that are used to represent buttons. However, this is not as useful as it initially may seem, because it is not possible to customise the look-and-feel of the screen UI.

New to WTK 1.0.4 is a speed emulation feature. This allows performance of the emulator to be scaled down to better reflect performance on a real device. Developing the application in a slower performance environment facilitates the monitoring and optimising of the code for low-end devices.

Figure 4. WTK Emulator Performance Configuration.



The performance feature cannot accurately emulate the performance of a specific device. However, it can be useful in providing an adequate approximation.

Broadly, the configurable performance parameters provided control graphics, VM and network speed.

3.8. Debugger

The WTK does not provide an integrated debugger. Instead, a third-party debugger should be used.

However, the WTK does provide limited tracing capabilities. The following events can be traced during emulation:

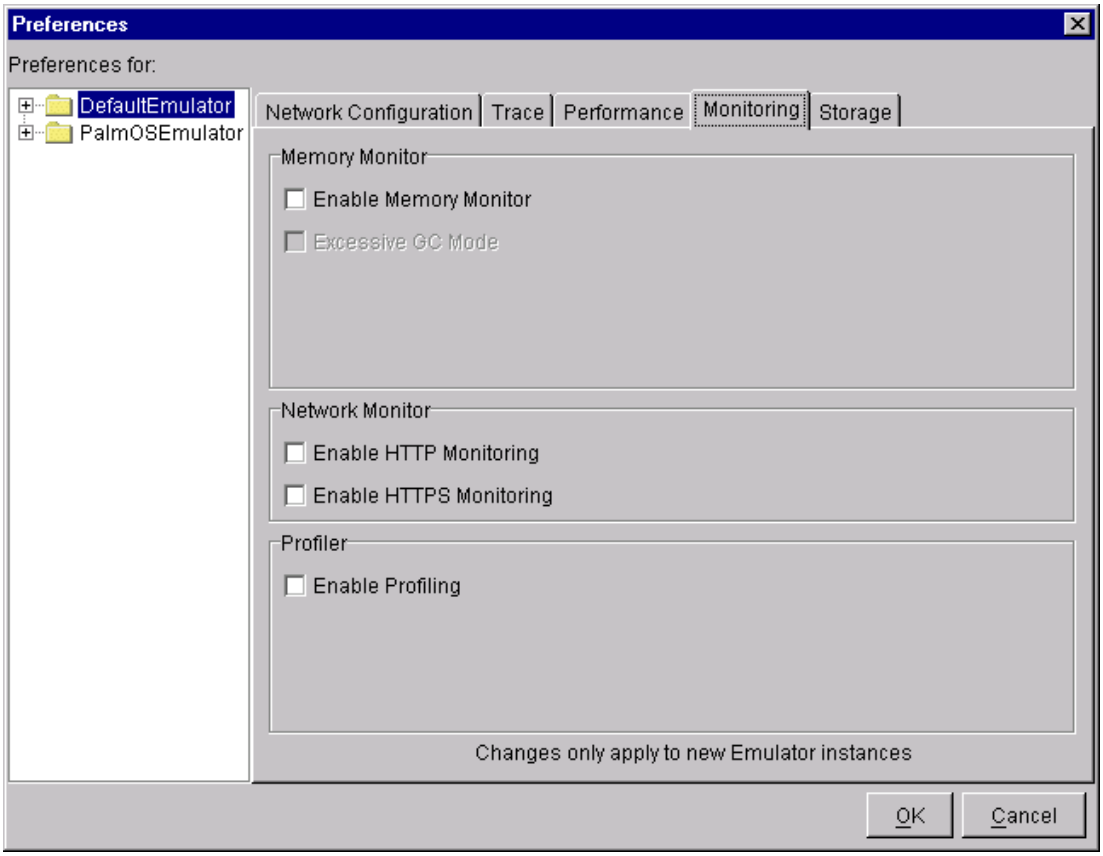
- Garbage collection
- Class loading
- Method calls

- Exceptions

In addition, stdout and stderr are captured to the KToolBar window.

For WTK 1.0.4, new monitoring features have been added which allow detailed analysis of the execution of a MIDlet.

Figure 5. WTK Emulator Monitoring Configuration.



A profiler enables examination of the execution time and the frequency of methods calls in a MIDlet.

A memory monitor enables monitoring of memory usage in a MIDlet.

A network monitor enables monitoring of transmissions between a MIDlet and the network. This is especially useful when a MIDlet is interfacing to a back-end process, such as a servlet, via HTTP. The monitor allows examination of the HTTP requests/responses the MIDlet has handled. The monitor also supports monitoring of the secure HTTPS protocol.

3.9. Conclusion

The WTK provides a competent basic environment for developing MIDlets. Clearly, with its lack of integrated editor and debugger facilities, it cannot (and indeed does not aim to) compete with the commercial IDEs available.

However, the WTK does provide a jumpstart for the beginner J2ME developer. By stipulating a project directory structure, it enables the developer to start building immediately without concerning themselves with the build infrastructure. This was often a complaint of the beginner J2EE developer, and it's encouraging to see Sun have addressed this grievance.

Finally, the WTK is a free and relatively small download. I would recommend all beginner J2ME developers familiarise themselves with the WTK. However, for an experienced J2ME developer involved in a large-scale commercial project, one of the commercial IDEs will provide a far more productive development environment.

4. Borland JBuilder MobileSet 2.0

4.1. Introduction

JBuilder MobileSet 2.0 is an extension to an existing installation of JBuilder 6. It works with any edition of JBuilder 6, although some features are unavailable with JBuilder 6 Personal Edition, such as JDK switching and the Archive Builder.

The JBuilder working environment for MobileSet is the same as that for normal Java development, with the exception of additional tabs and options in existing dialog boxes, and two new wizards: the MIDlet wizard and the MIDP Displayable wizard.

Also, in JBuilder Professional and Enterprise Edition, the Archive Builder can now create a MIDlet suite and its corresponding manifest and JAD files.

MobileSet also installs WTK 1.0.3 which is utilised as the emulation environment. However, instructions are provided for integrating MobileSet with other J2ME JDKs, such as the Nokia Developer's Suite for J2ME and Siemens Mobility Toolkit (SMTK).

In addition to the core JBuilder features, MobileSet adds:

- Code completion for CLDC/MIDP classes
- Class/package browser for CLDC/MIDP classes
- JDK switching
- MIDP wizards
- Visual designer for MIDP UI elements
- Debugging of MIDlets
- JAD and JAR file packaging
- Over The Air (OTA) provisioning

4.2. Pre-requisites

MobileSet supports the Windows NT 4.0 SP6 and Windows 2000 SP2 platforms.

MobileSet has the following hardware requirements:

- Pentium III CPU
- 128 Mb RAM (minimum), 256 Mb RAM (recommended)
- 11-20 Mb hard disk space (depending on features installed)

MobileSet requires an existing installation of JBuilder 6. JBuilder 6 has similar hardware requirements to MobileSet, but requires an additional 500-700 Mb of hard disk space depending on the edition of JBuilder 6 installed.

4.3. Installation

MobileSet is distributed as a ZIP file which can be downloaded from the following URL:

http://www.borland.com/jbuilder/mobileset/

Before installation, the ZIP file should be extracted to a temporary directory.

At the start of installation, the license agreement is presented which must be accepted before continuing. MobileSet is provided as a free extension to an existing, suitably licensed version of JBuilder 6.

Next, the installer prompts for the install set. It is possible to install MobileSet on its own, or MobileSet and the WTK 1.0.3. Unless WTK 1.0.3 is already installed, it is recommended to choose the latter option.

Next, the installer detects the folder where the existing version of JBuilder 6 is installed. This can be overridden if required.

MobileSet is then installed and integrated with the existing version of JBuilder 6.

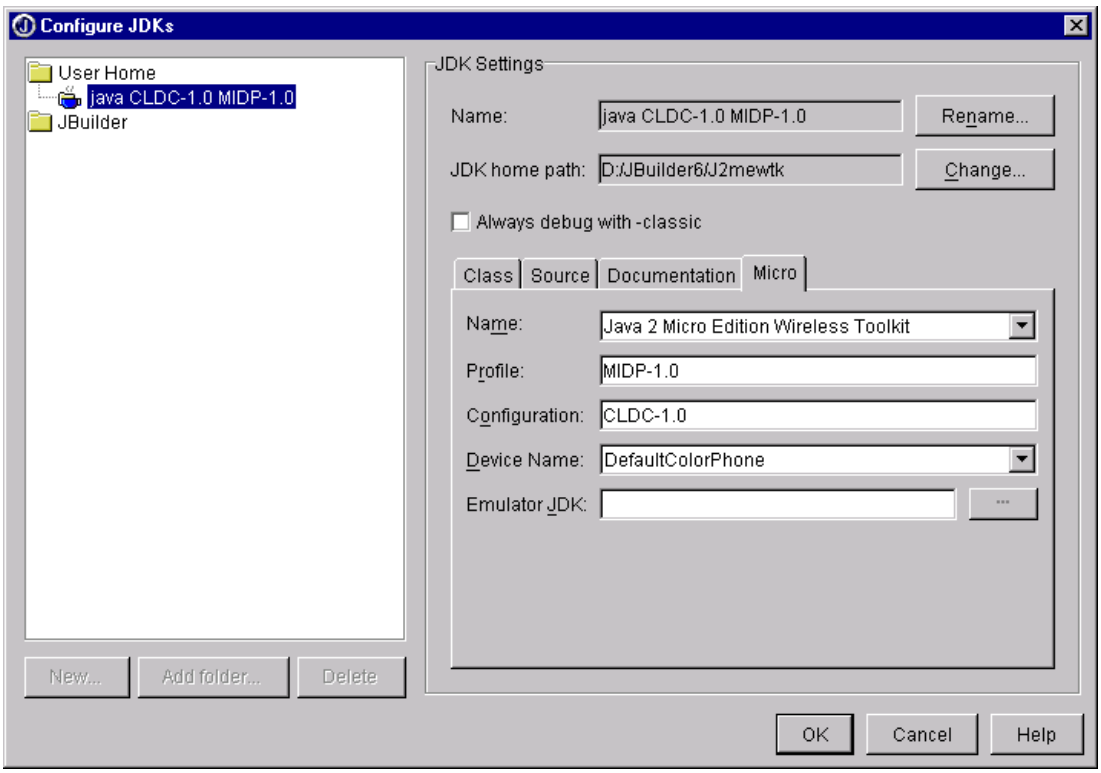
On first starting JBuilder 6 (which is now integrated with MobileSet), an activation key for MobileSet needs to specified. An activation key can be obtained from the following URL:

http://www.borland.com/jbuilder/offers/

By default, JBuilder is configured to use the J2SE JDK. To support J2ME development, JBuilder needs to configured to use a J2ME JDK, such as WTK 1.0.3.

In JBuilder Professional and Enterprise Edition, more than one JDK can be defined and JDKs can be switched on a project by project basis. In the Personal Edition, only a single JDK can specified which applies to all projects.

Figure 6. MobileSet Configuring a J2ME JDK.



4.4. Project Management

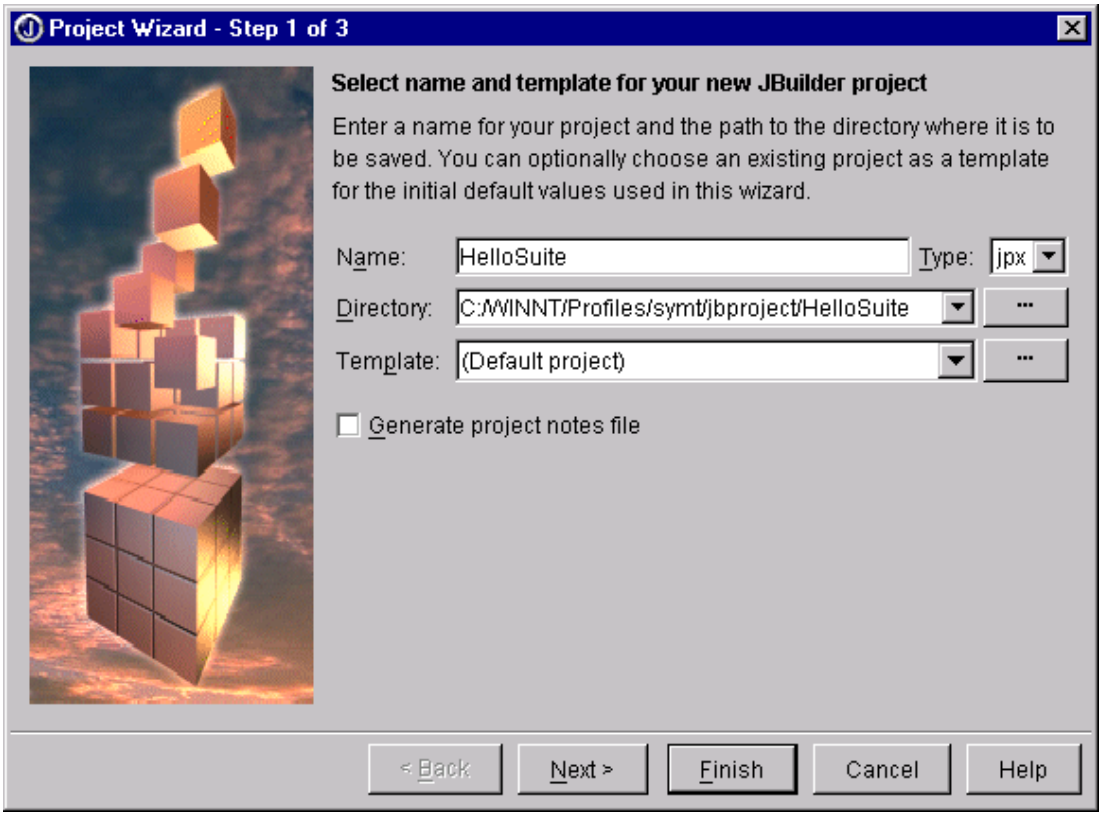
JBuilder development is centred around the concept of a project.

The project file contains a list of files in the project and the project properties. JBuilder uses this information when a project is loaded, saved, built, or run. Project files are modified whenever the JBuilder development environment is used to add or remove files, or set or change project properties.

When developing MIDP applications, the project must be associated with a J2ME JDK before the J2ME features provided by the MobileSet extension can be accessed. The project is in effect the MIDlet suite to which one or more MIDlets can be added.

To create a new project, a wizard is provided to generate the basic framework of files, directories, paths, and preferences. The project is initially empty.

Figure 7. MobileSet Project Wizard.



A wizard allows the additon of MIDlets. The wizard generates skeleton files based on the option chosen:

- **MIDlet.** Generates a class which extends MIDlet and a class which extends a Displayable class (Canvas, Form, List or TextBox). The command handling between the two classes can be configured.
- **MIDP Displayable.** Generates a standalone class which extends a Displayable class.

In most cases, the MIDlet option will be chosen. This invokes the MIDlet wizard which gathers all information required. A separate wizard is invoked for the MIDP Displayable option.

Figure 8. MobileSet MIDlet Wizard Step 1.

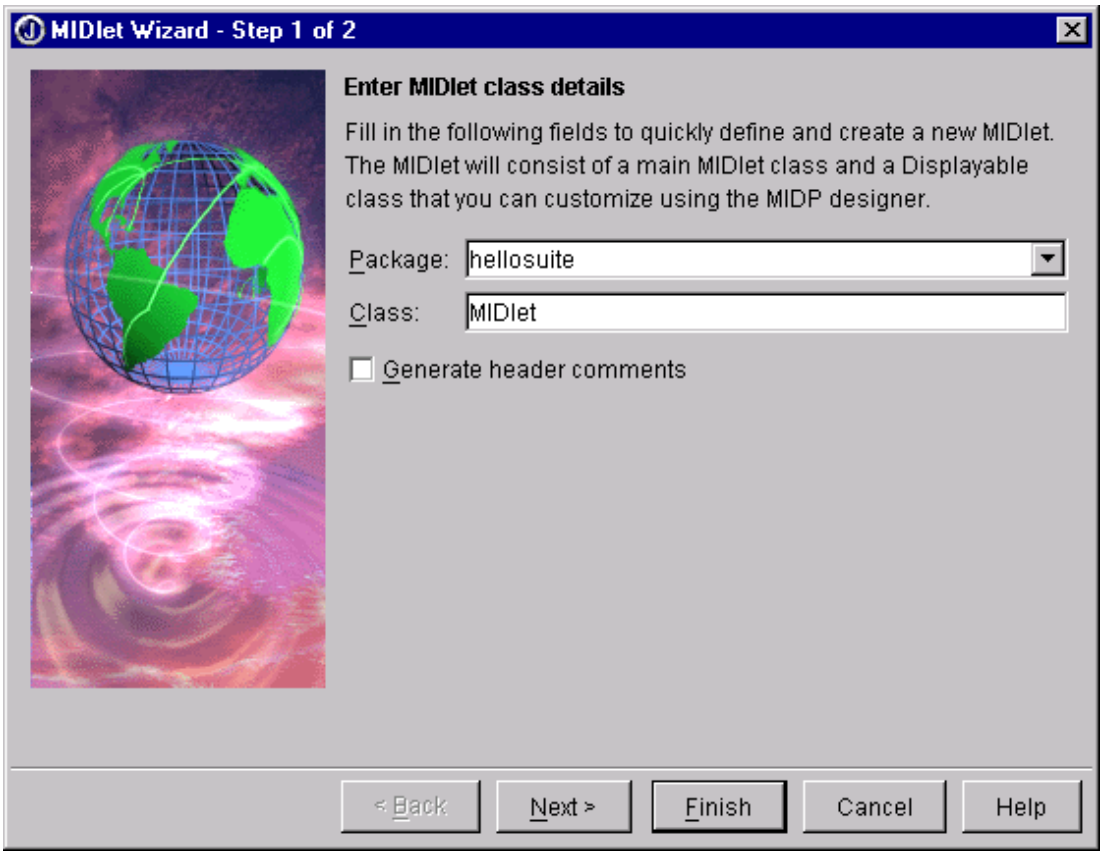
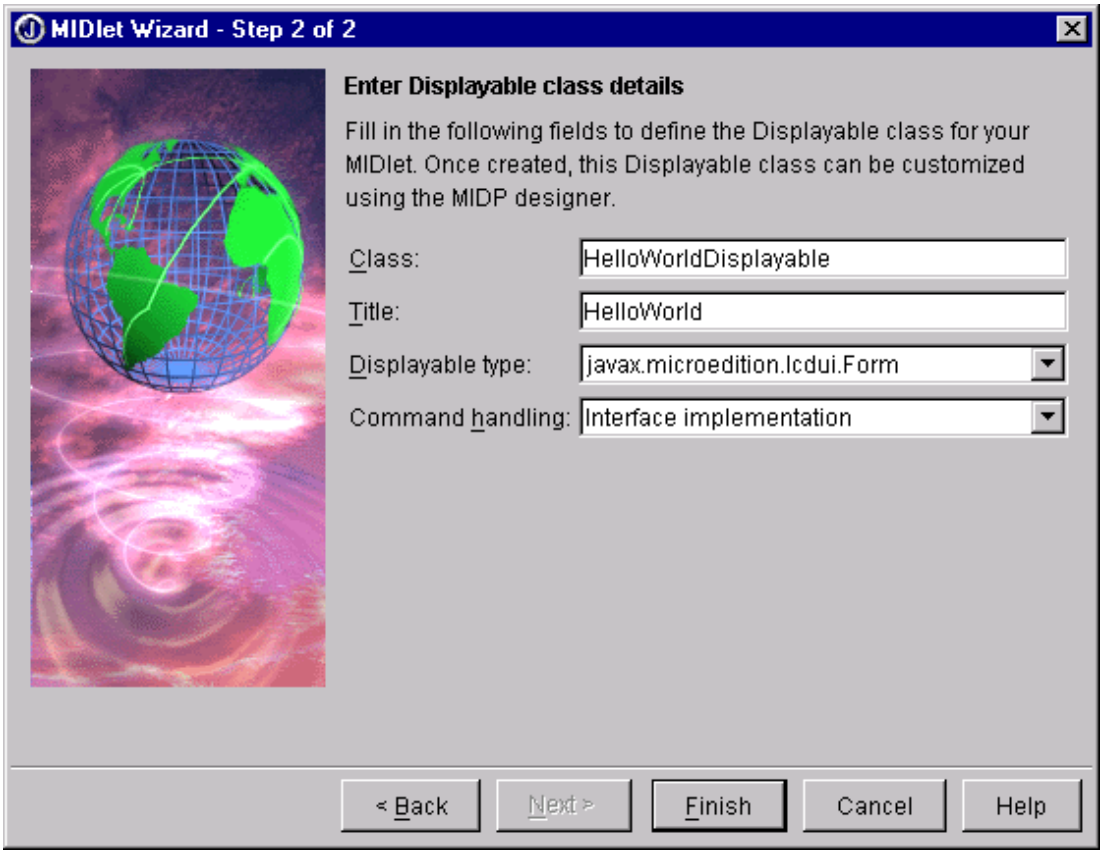


Figure 9. MobileSet MIDlet Wizard Step 2.



The project pane provides a view onto source files within the project.

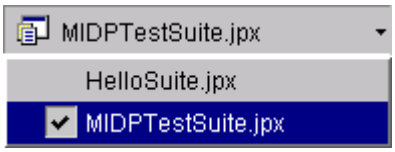
Figure 10. MobileSet Project Pane.

On double-clicking a source file in the project pane, the structure pane provides a detailed view of the classes within that source source file.

Figure 11. MobileSet Structure Pane.

JBuilder supports multiple projects loaded within the IDE. The project selector drop-down box is used to switch between projects.

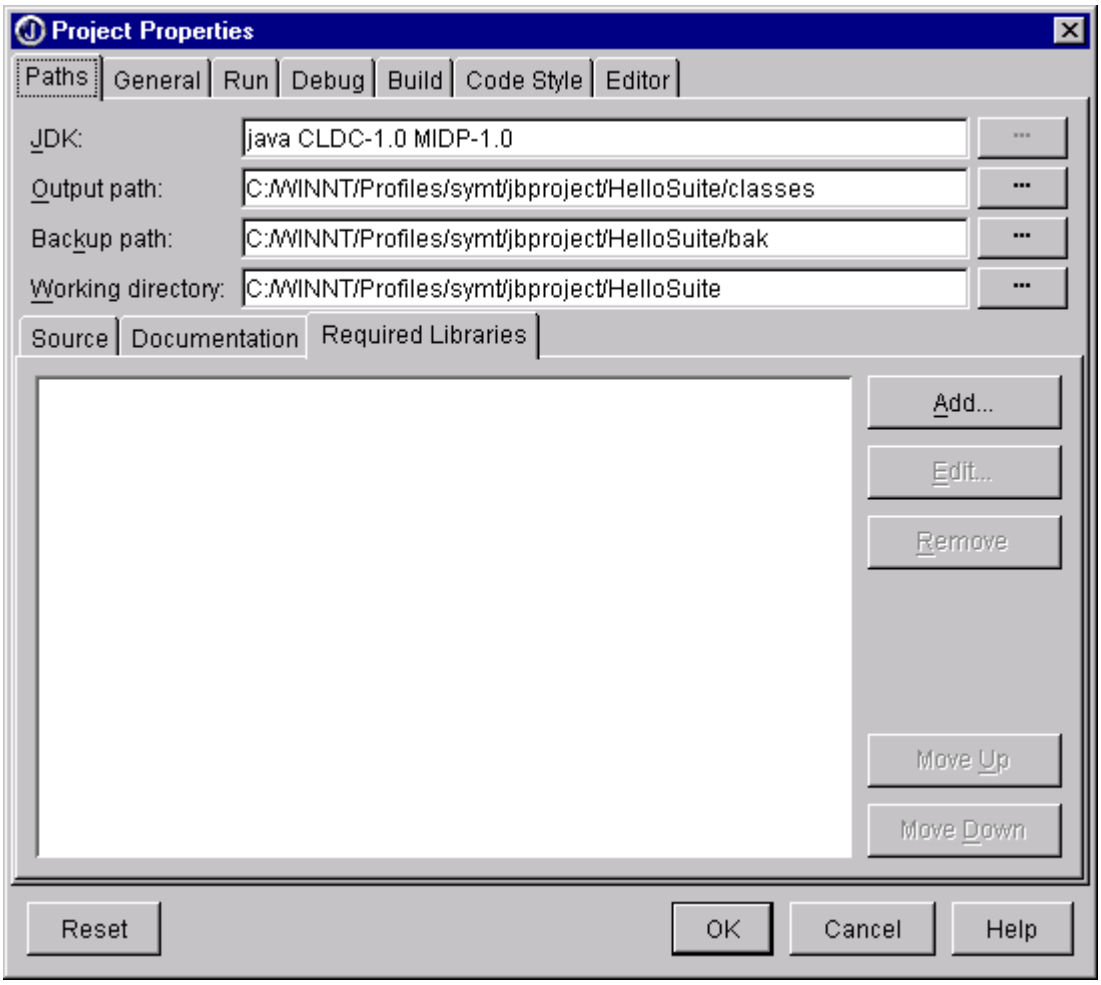
Figure 12. MobileSet Project Selector.



In JBuilder Enterprise Edition, the projects can be associated with any JDK. Therefore, it is possible to have a J2EE project containing a servlet and a J2ME MIDP project acting as a client to the servlet. Both projects can be running within the IDE simultaneously, providing an integrated end-to-end enterprise debugging environment.

External class libraries can be configured via the Required Libraries tab in Project Properties.

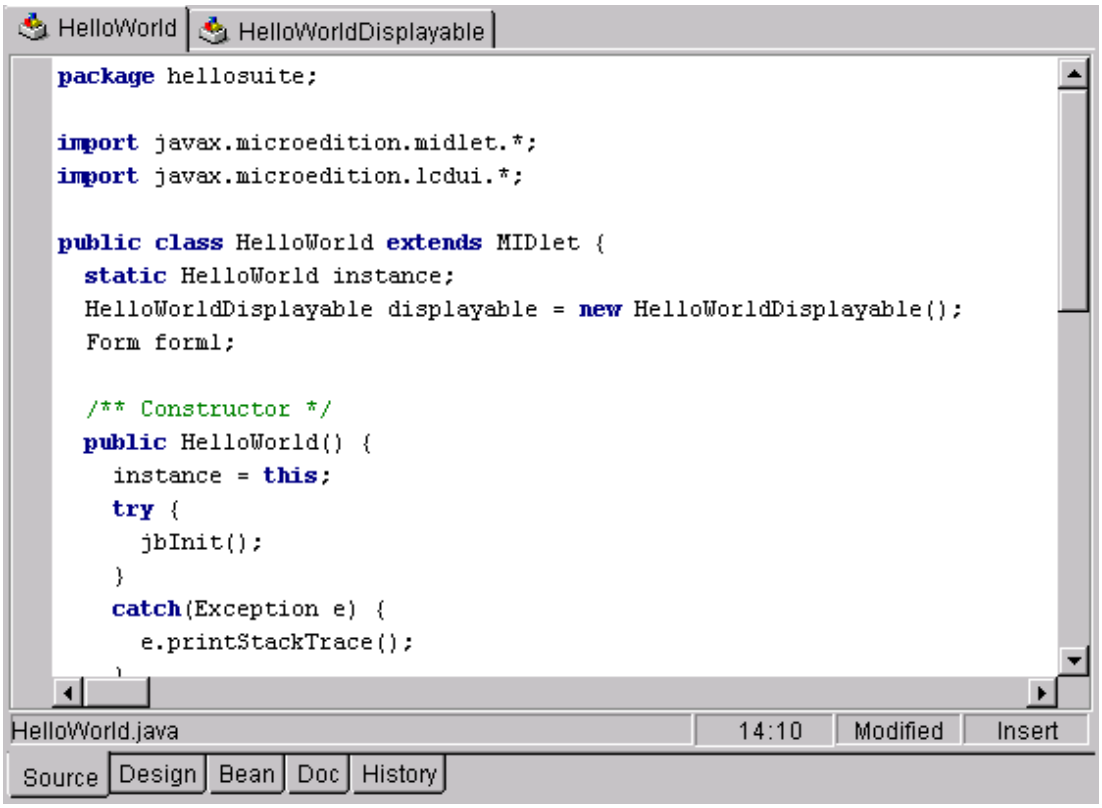
Figure 13. MobileSet Project Properties Required Libraries Tab.



4.5. Editor

MobileSet utilises the JBuilder integrated source code editor.

Figure 14. MobileSet Source Code Editor.

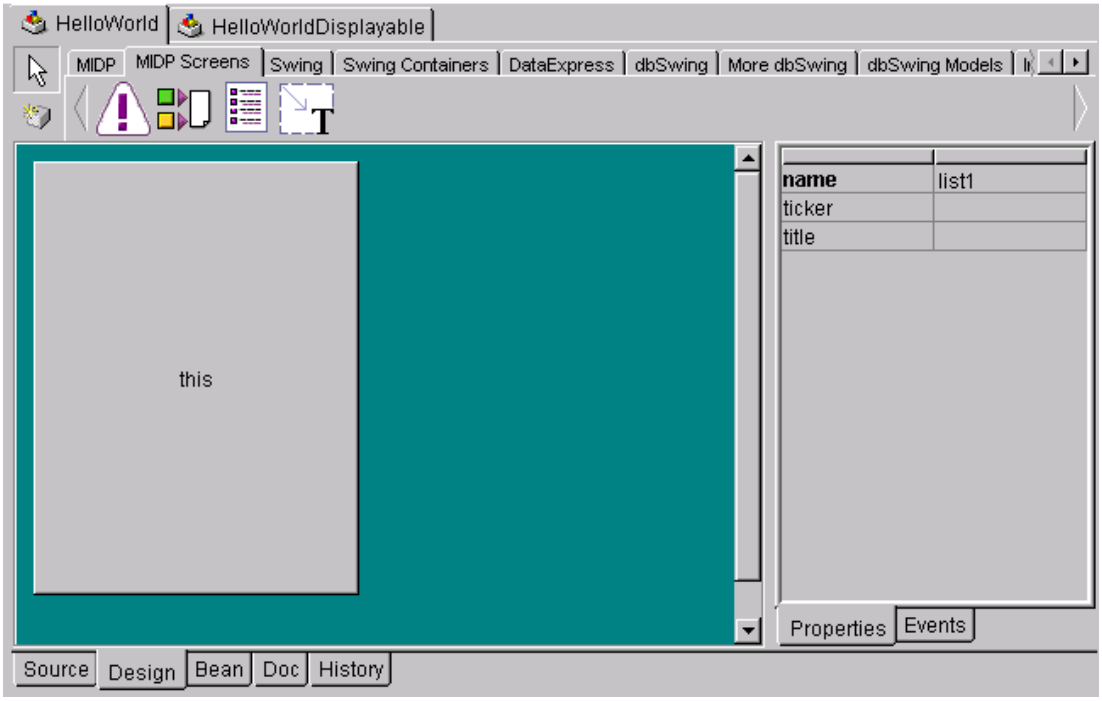


The main features the source editor supports are:

- Brace matching
- Keyboard shortcuts
- Syntax highlighting
- Customisable editor key mappings
- CodeInsight (code completion, parameter lists, and tool tip expression evaluation)
- Code templates
- Multiple file editing via a tabbed interface
- Re-factoring

MobileSet includes a UI designer for MIDP applications. The MIIDP designer is based on the Swing UI designer which has been available in JBuilder for a while.

Figure 15. MobileSet MIDP Designer.



The MIDP designer makes it possible to work visually with the basic MIDP UI components. In addition, generation of skeleton event-handling code is supported.

The following MIDP UI components are supported:

- ChoiceGroup
- DateField
- Gauge
- ImageItem
- StringItem
- TextField
- Ticker

The following MIDP screen components are supported:

- Alert
- Form
- List
- TextBox

When manipulating components with the MIDP designer, MobileSet automatically generates code to implement the UI. The code generated is specific to MobileSet, which means the MIDP designer will not be able to re-work previously handcrafted code.

The MIDP designer is especially useful for rapidly implementing MIDP prototypes and test harnesses.

4.6. Build/Package

MobileSet provides integrated facilities to build the project. When compiling, MobileSet:

- Compiles the MIDlet and produces class files
- Pre-verifies the compiled class files
- Create the JAD file

It is possible to compile the whole project, or an individual MIDlet suite or MIDlet.

JBuilder Professional and Enterprise Edition provide an integrated code obfuscator. This can be activated via the Build tab in Project Properties. In JBuilder Personal Edition, this option is disabled.

However, the implementation of the integrated code obfuscator is widely regarded as weak. This has lead to the availability of Open Tool add-on modules for JBuilder which integrate to third-party code obfuscators. For more information, see the following URL:

http://www.visi.com/~gyles19/fom-serve/cache/132.html

If MobileSet has been integrated with JBuilder Professional or Enterprise Edition, integrated support for packaging is provided via the Archive Builder. In JBuilder Personal Edition, the packaging has to be performed outside the IDE.

The Archive Builder is a wizard that performs the entire packaging process, including creating the JAR and JAD files. Once the wizard is complete, a node is created in the project pane. Whenever the project is rebuilt, the JAR and JAD files are automatically re-created.

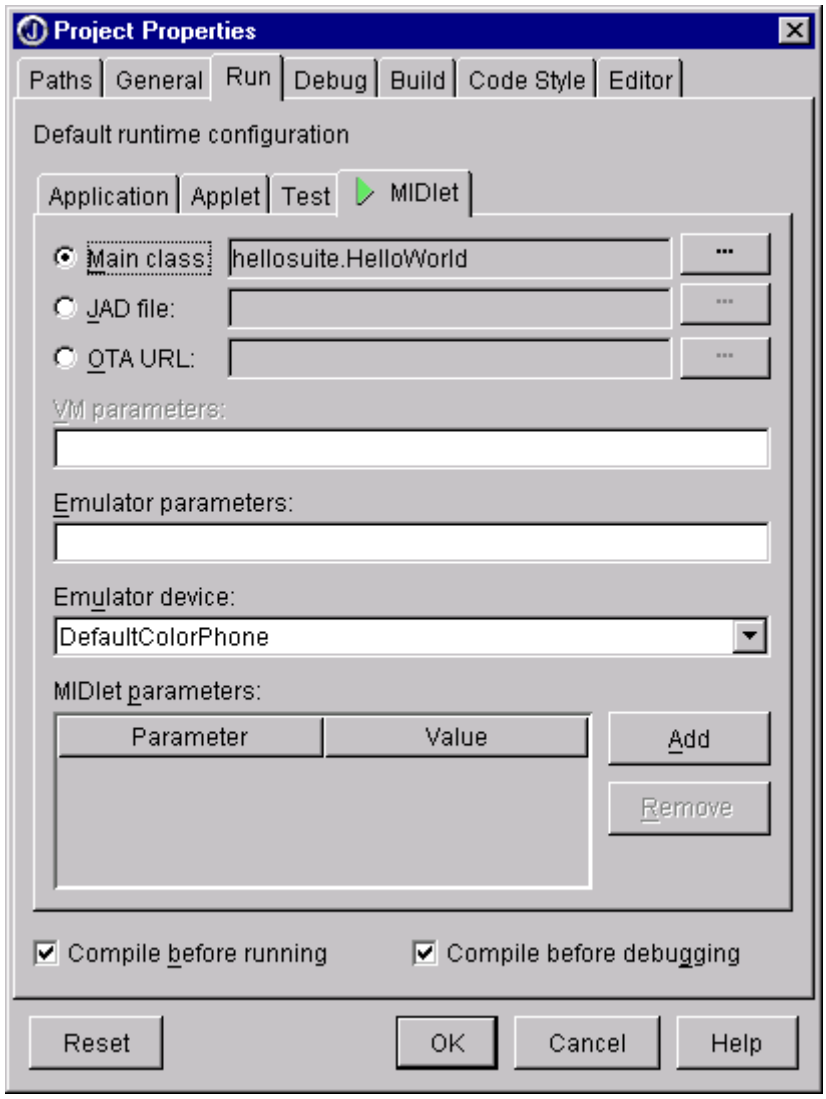
4.7. Emulation

By default, MobileSet utilises the WTK 1.0.3 emulation environment for running MIDP applications. In addition, it is possible to employ third-party emulators by configuring the J2ME JDK which MobileSet uses.

Only third-party emulators which support a J2ME JDK emulation environment can be used with MobileSet. No problems were encountered with MobileSet using the Nokia Developer's Suite for J2ME and Siemens Mobility Toolkit (SMTK).

If the J2ME JDK supports multiple devices, the MIDlet tab in Project Properties can be used to select the emulation device to utilise when running or debugging a MIDP application within the IDE.

Figure 16. MobileSet Project Properties MIDlet Tab.

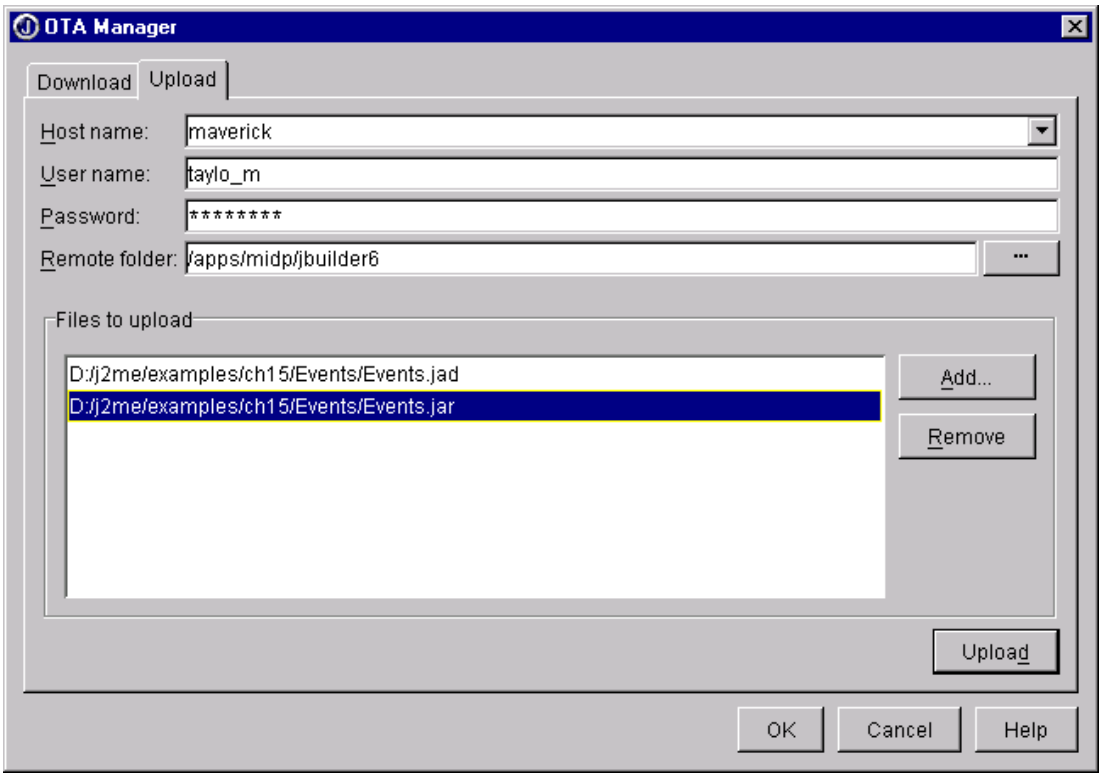


This tab also allows selection of the main MIDlet class. Alternatively, a JAD file or OTA URL can be provided. In addition, emulator and MIDlet parameters can be configured.

A convenient feature of JBuilder is the ability to select multiple MIDlets in the project pane and run the project. The IDE automatically creates a MIDlet suite containing the selected MIDlets and invokes the emulator for this suite.

MobileSet supports Over the Air (OTA) provisioning from within JBuilder.

Figure 17. MobileSet OTA Provisioning.



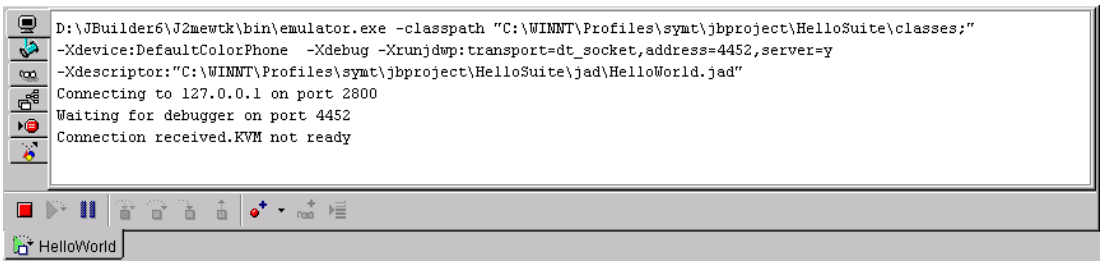
MIDlet suites can be uploaded to an FTP server without leaving JBuilder, then tested in an emulator by downloading and running them using OTA provisioning. It is also possible to discover and run third-party MIDlets on the Internet.

4.8. Debugger

MobileSet utilises the JBuilder integrated debugger. The debugger can be used to debug MIDP applications running in any supported emulator.

When the debugger is started, in addition to the emulator window itself, a horizontal tab representing the debugging session is shown in the IDE. Multiple concurrent debugging sessions are supported and can be switched between using these tabs.

Figure 18. MobileSet Debugging Session.



Vertical tabs on the left represent debugger views of what is going on inside the current debugging session. The following views are provided:

- **Console.** Displays output from the program and errors in the program.

- **Threads, call stacks and data.** Displays the thread groups in the program. Each thread group expands to show its threads, and contains a stack frame trace representing the current call sequence. Each stack frame can be expanded to show available data elements that are in scope.
- **Synchronisation monitors.** Shows synchronisation monitors used by the threads and their current state, which is useful in detecting deadlocked situations.
- **Data watches.** Displays the current values of data members that are being tracked.
- **Loaded classes and static data.** Displays the classes currently loaded by the program. Expanding a class shows static data, if any, for that class. If a package is displayed in the tree, the number of classes loaded for that package is displayed.
- **Data and code breakpoints.** Shows all the breakpoints set in the file and their current state.
- **Classes with tracing disabled.** Displays an alphabetically ordered list of classes and packages not to step into.

The source editor shows the source code at the point where the MIDP application is halted in the current debugging session. Breakpoints are also shown in the source editor.

In JBuilder Professional and Enterprise Edition, variables can be evaluated by moving the mouse pointer over the variable name in the source code. In addition, JBuilder supports ExpressionInsight, where it is possible to drill down into the members of a variable from a pop-up window displayed within the source code.

4.9. Conclusion

MobileSet provides a cohesive extension to JBuilder. Developers familiar with JBuilder for J2SE or J2EE development will find the transition to J2ME straightforward.

The MobileSet MIDP designer is a useful visual tool for constructing MIDP UIs. However, it does not yet support the full set of MIDP UI components and in many cases the code it generates needs handcrafting. We can expect future versions of MobileSet to provide a more flexible and fully featured MIDP designer.

The review version of MobileSet was 2.0 integrated with JBuilder 6 Personal Edition. The Personal Edition is a free download, but is licensed for personal use only.

The Professional and Enterprise Editions are commercial products with unlimited commercial development licenses. These editions also provide a whole host of non-J2ME specific features over the Personal Edition, such as UML code visualisation, re-factoring, integrated unit testing and team development. A full review of these features is outside the scope of this document, but in a large commercial project these features can be expected to increase both individual developer and team productivity.

The Enterprise Edition provides J2EE development facilities, thus allowing development of enterprise-wide J2ME solutions.

JBuilder has been through several releases over the past few years and is one of the most popular commercial Java IDEs available. The Enterprise Edition, in particular, has enjoyed wide acceptance by many blue-chip companies for development of their Java enterprise-wide solutions. The addition of MobileSet integrates J2ME effectively into the existing JBuilder environment.

For the development of large-scale enterprise-wide J2ME solutions, JBuilder together with the MobileSet extension is arguably the most comprehensive commercial IDE available, providing a highly effective development environment.

5. Sun ONE Studio 4.0 EA Mobile Edition

5.1. Introduction

Sun ONE Studio 4.0 EA Mobile Edition is a specially customised version of the Sun ONE Studio IDE to support and facilitate the development of MIDP applications. It combines the technologies of the WTK (Sun ONE Studio ME installs WTK 1.0.3) with the Sun ONE Studio programming environment to provide the following:

- Integrated compilation, pre-verification, and execution of MIDlets and MIDlet suites
- Automatic generation of JAD and JAR files
- Integrated source-level debugging of MIDlets
- Code completion against J2ME APIs
- The ability to preset default compiler/pre-verifier settings on a per project basis
- Integration with third party emulators/JDKs through the Unified Emulator Interface
- Templates for creating MIDlets and MIDlet suites
- An integrated default emulator
- A target emulator that can be switched via the emulator registry for each MIDlet suite
- An AutoUpdate tool that enables modules to be added and ensures existing features of the IDE are kept up to date.

To provide the simplest yet most comprehensive environment possible, Sun has removed some core modules of the Sun ONE Studio IDE. These features are unsupported by the J2ME platform, or are unnecessary for creating mobile device applications.

5.2. Pre-requisites

Sun ONE Studio ME supports the Windows NT 4.0 SP6, Windows 2000 SP2, Solaris and Linux platforms.

Sun ONE Studio ME has the following hardware requirements on all platforms:

- 350 MHz CPU
- 128 Mb RAM
- 100 Mb hard disk space

The Java 2 SDK, Standard Edition 1.4 must already be installed before installing Sun ONE Studio ME.

5.3. Installation

Sun ONE Studio ME is distributed as a single executable and can be downloaded from the following URL:

http://forte.sun.com/eap/home/ffj/

Registration for the Sun Early Access Program (EAP) must be completed and approved before Sun ONE Studio ME can be downloaded.

At the start of installation, the installer searches for a suitable Java 2 SDK. The installer prompts with the destination folder of the Java 2 SDK it has found. This can be overridden if required.

Next, the installer prompts for a destination folder for Sun ONE Studio ME. This can also be overridden if required.

Next, the installer prompts for the set of Sun ONE Studio ME components to install. There appears to a bug in the installer when selecting and de-selecting components. Therefore, it is recommended that the defaults are accepted.

Sun ONE Studio ME is then installed.

After installation, the option is given to associate .java and .nbm files with the IDE.

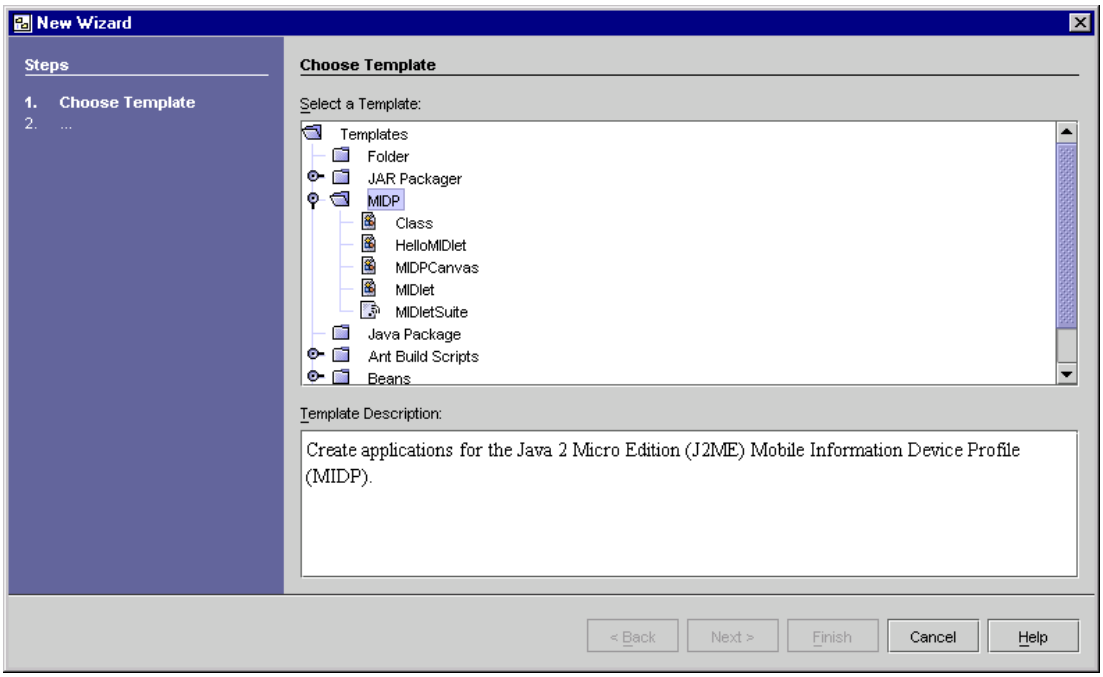
On first starting Sun ONE Studio ME, several wizards are invoked concerning importing existing IDE settings, customising the IDEs appearance and product registration.

5.4. Project Management

A Sun ONE Studio ME project organises the files required to produce MIDlets and/or MIDlet suites.

On creation of a new project, the project is initially empty. A wizard allows the addition of MIDlets and/or MIDlet suites to the project.

Figure 19. Sun ONE Studio ME Add New To Project Wizard.



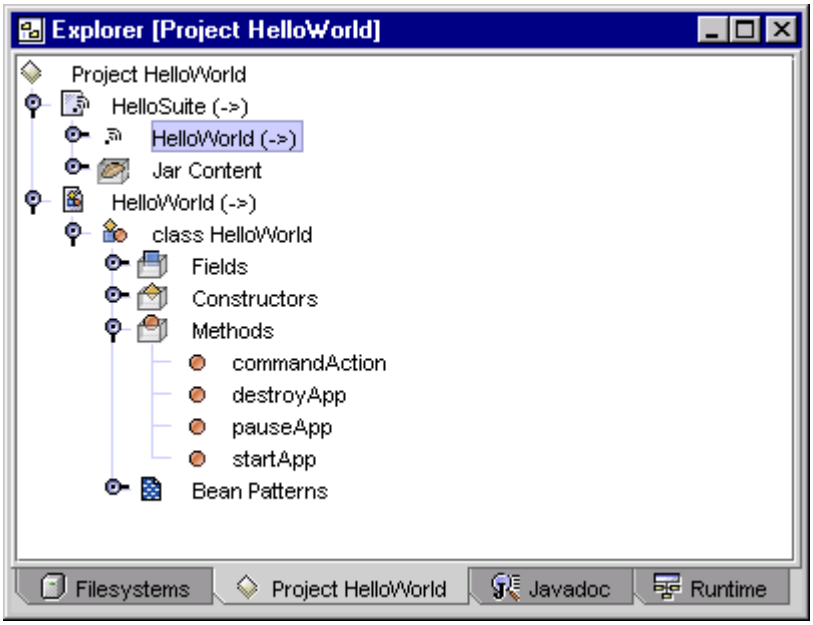
The wizard generates skeleton files based on the type of template selected. The following MIDP templates are available:

- **Class.** Empty Java class which uses the default J2ME compiler, debugger and executor.
- **HelloMIDlet.** Executable class which displays text on the screen of a device.
- **MIDPCanvas.** Creates a subclass of the Canvas class (low-level user interface API). Canvas gives full control over the appearance of the user interface, but may not be portable across devices.
- **MIDlet.** Basic executable class for a wireless device.
- **MIDlet Suite.** Creates a MIDlet suite containing a single skeleton MIDlet.

After selecting a template, a wizard gathers all the information required. Once complete, the skeleton files are generated.

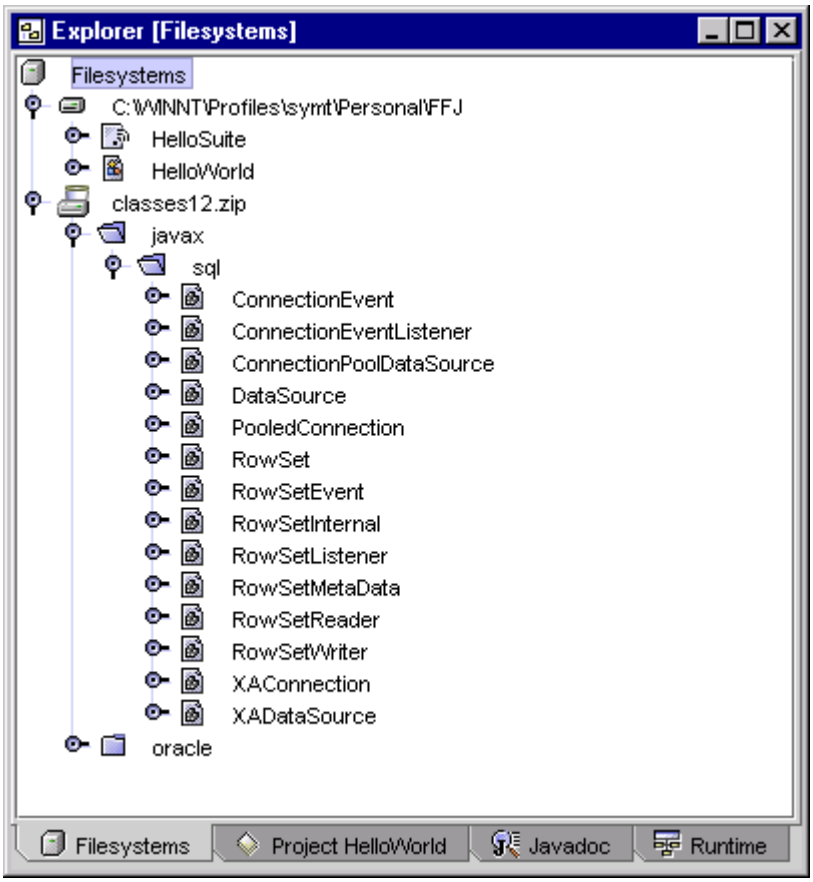
The Project tab on the Explorer window provides a view onto the project. From this window details of the MIDlet suites and MIDlets contained within the project can be browsed.

Figure 20. Sun ONE Studio ME Explorer Project Tab.



External class libraries can be configured in Sun ONE Studio ME using the Filesystems tab on the Explorer window. Once added, an excellent feature is the ability to browse the details of the classes within the library.

Figure 21. Sun ONE Studio ME Explorer Filesystems Tab.

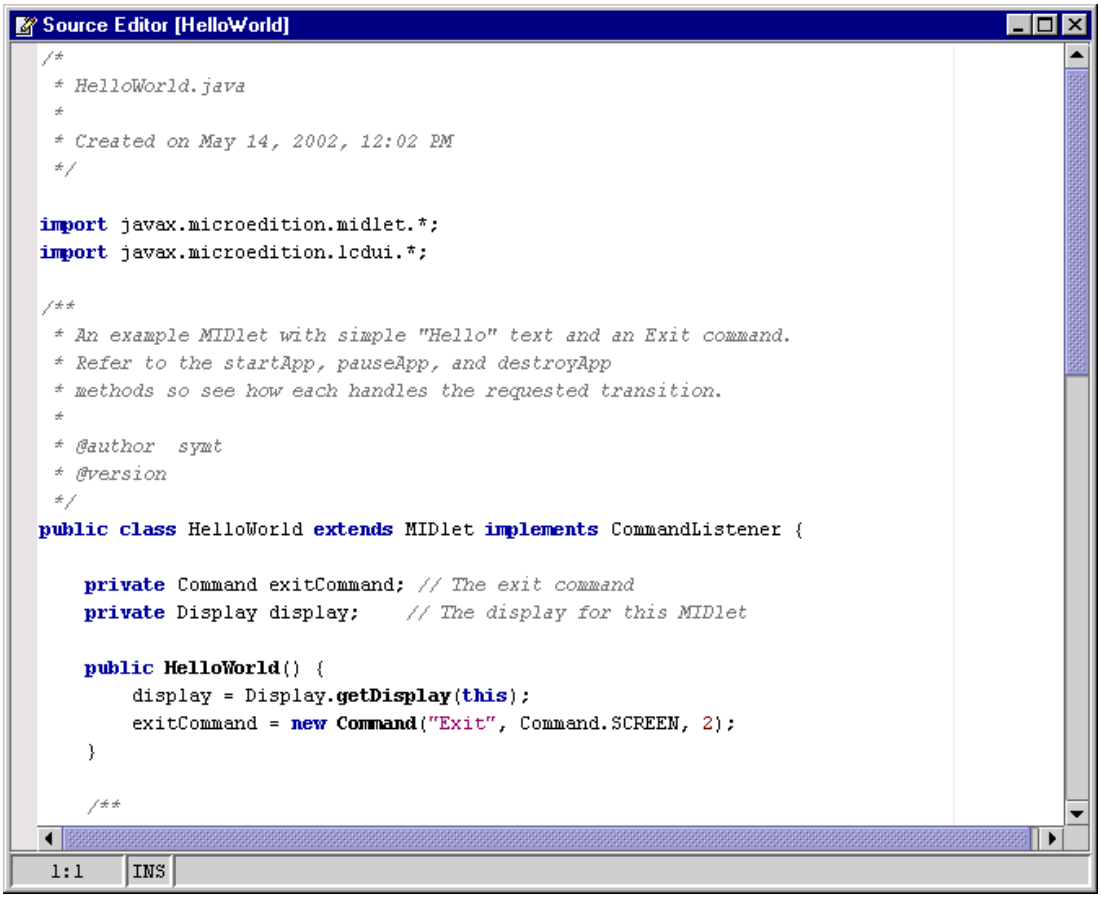


Sun ONE Studio ME does not support multiple projects loaded within the IDE. However, a project can contain multiple MIDlet suites which overcomes this limitation.

5.5. Editor

Sun ONE Studio ME provides an integrated source code editor.

Figure 22. Sun ONE Studio ME Source Editor.



The main features the source editor supports are:

- Configurable syntax highlighting
- Brace matching
- Line numbers
- Right margin indicator
- Multiple file editing via a tabbed interface
- Configurable automatic code formatting
- Code completion
- Abbreviations and macros

Sun ONE Studio ME provides no other resource editors which would be of use to the MIDP application developer. However, the source editor has the ability to display PNG images.

5.6. Build/Package

Sun ONE Studio ME provides integrated facilities to build and package the project. The IDE combines building and packaging into a single step during a compile. When compiling, Sun ONE Studio ME:

- Compiles the MIDlet and produces class files
- Pre-verifies the compiled class files
- Creates the JAR file
- Create the JAD file

It is possible to compile the whole project, or an individual MIDlet suite or MIDlet.

Sun ONE Studio ME provides no integrated support for code obfuscation. However, as Sun ONE Studio ME offers integrated Ant[1] support, it ought to be possible to write an Ant build script which executes the code obfuscator.

5.7. Emulation

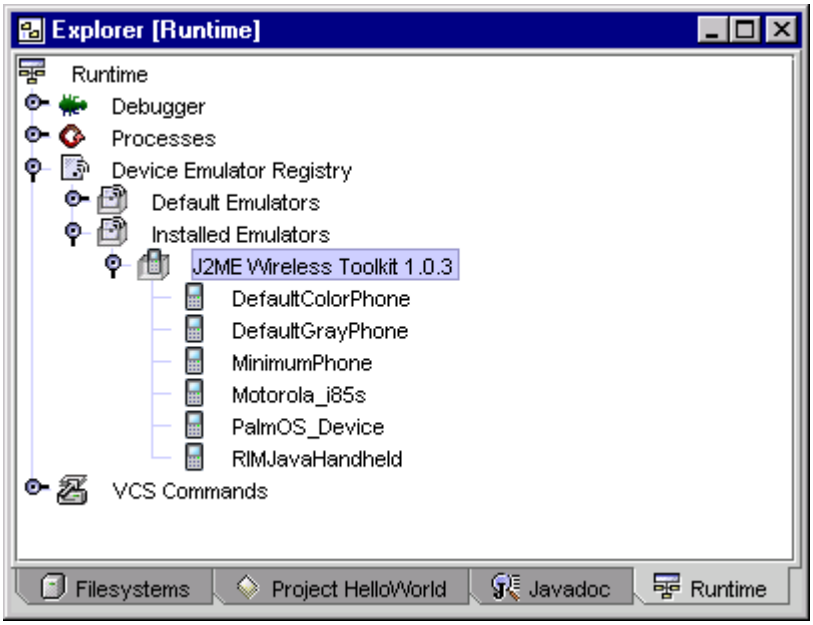
By default, Sun ONE Studio ME utilises the WTK 1.0.3 emulator environment for running MIDP applications. In addition, it is possible to add third-party emulators via the Device Emulator Registry.

Only third-party emulators which support a J2ME JDK emulation environment can be used with Sun ONE Studio ME. No problems were encountered using Nokia Developer's Suite for J2ME and Siemens Mobility Toolkit (SMTK).

The Runtime tab on the Explorer window is used to manipulate the Device Emulator Registry. Emulators instances can be added and deleted. The default device for an emulator instance can be selected.

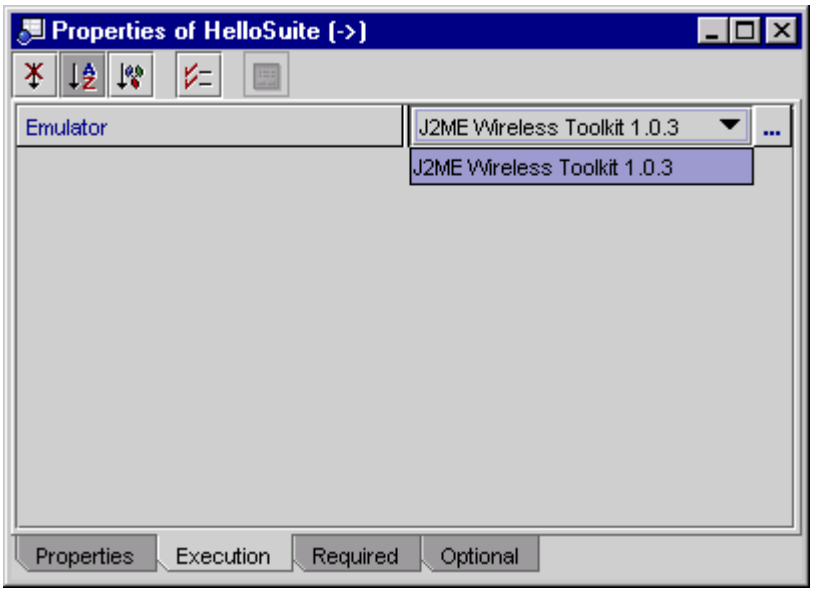
[1] Apache Ant is a Java-based command-line build tool. Ant configuration files and build scripts are XML-based. For more information, see http://jakarta.apache.org/ant/.

Figure 23. Sun ONE Studio ME Device Emulator Registry.



A default emulator instance is assigned to a specific MIDlet suite through the Execution tab in the Properties window.

Figure 24. Sun ONE Studio ME Properties Execution Tab.



5.8. Debugger

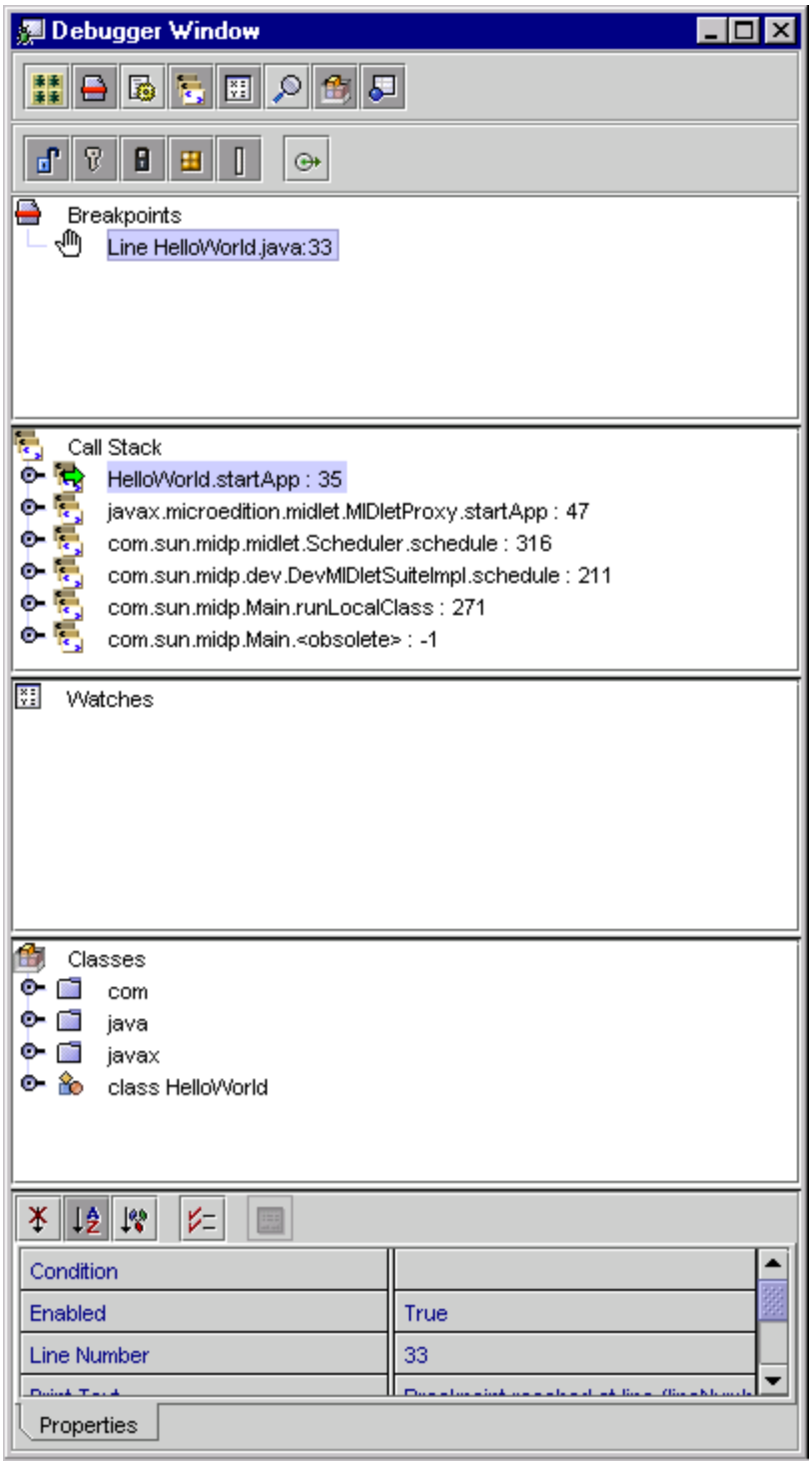
Sun ONE Studio ME provides an integrated debugger which can be used to debug MIDP applications running in any supported emulator.

The debugging workspace is shown when a debugging session is started. This workspace includes three windows: Debugger Window, Output Window and Source Editor.

The Debugger Window contains seven views of what is going on inside the MIDP application:

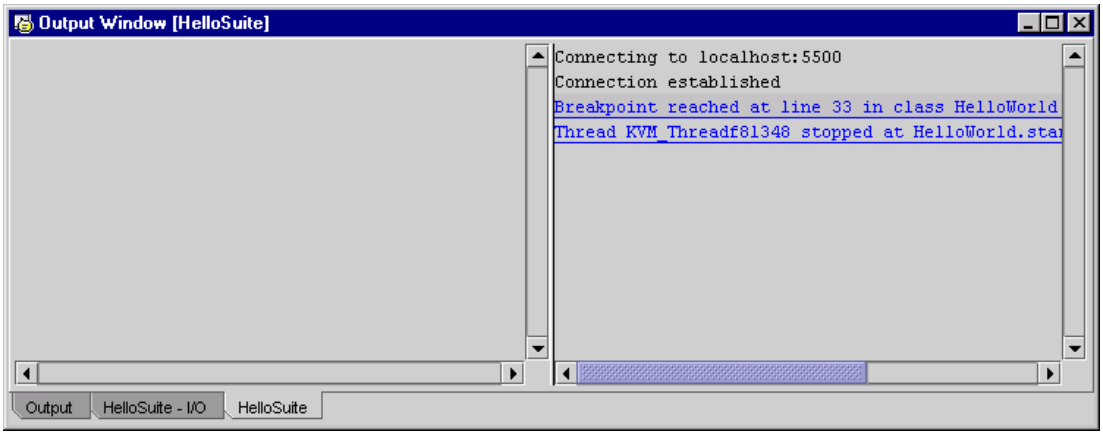
- **Sessions.** Lists the applications which are running. For MIDP applications, there is only ever one session.
- **Breakpoints.** Lists currently set breakpoints. Breakpoints can be set on a specific line, a method name, an exception, a class, a thread or a variable. Conditional breakpoints are supported.
- **Threads.** Lists the threads and thread groups in the current debugging process
- **Call Stack.** Lists the hierarchy of method calls made during execution of the current thread. The method that was executing when the thread stopped is at the top of the stack. The initial method is at the bottom of the stack.
- **Watches.** Enables specification of variables and expressions to watch continuously while debugging the application. Modification of watch variable values is supported.
- **Variables.** Lists the local, instance, and static variables that are within the scope of the method that the application is currently stopped in.
- **Classes.** Display the hierarchy of all classes that have been loaded by the process being debugged. Packages, classes, fields, constructors, static variables and methods can be inspected.

Figure 25. Sun ONE Studio ME Debugger Window During Debugging.



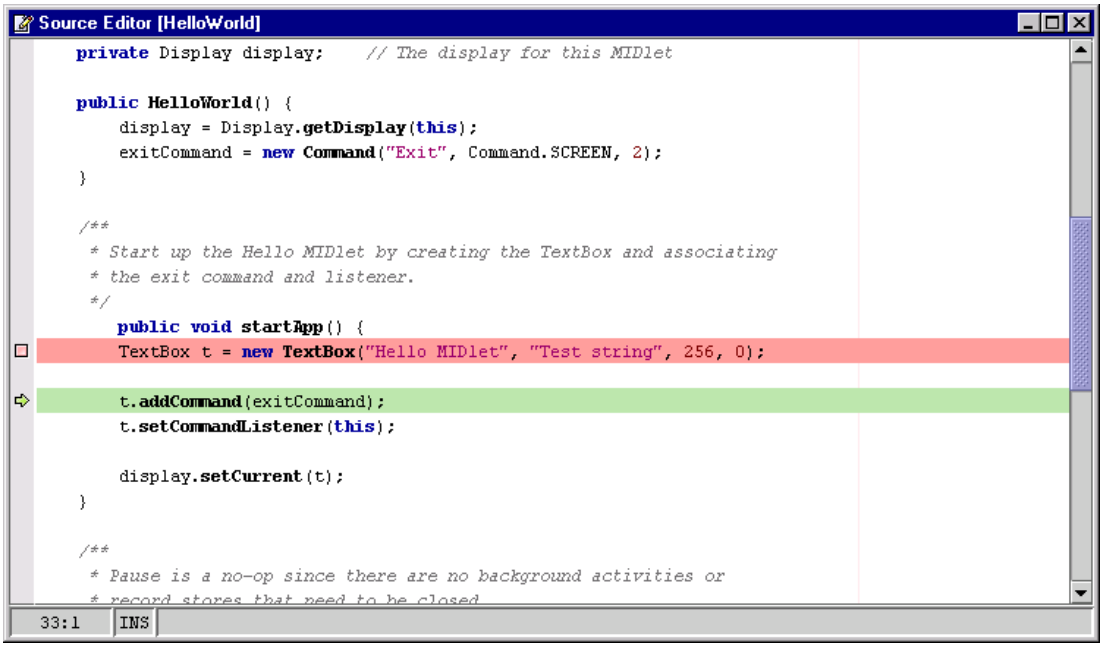
The Output Window contains two panes. The left pane captures stdout and stderr from the MIDP application. The right pane displays details of the MIDP applications threads and breakpoints.

Figure 26. Sun ONE Studio ME Output Window During Debugging.



The Source Editor shows the source code at the point where the MIDP application is currently stopped. Variables can be evaluated directly by moving the mouse pointer over the variable name in the source code.

Figure 27. Sun ONE Studio ME Source Editor During Debugging.



5.9. Conclusion

Sun ONE Studio ME is a good first attempt at a J2ME IDE. All the J2ME features integrate appropriately into the existing Sun ONE Studio programming environment.

Developers who have a basic understanding of MIDP application programming and previous experience of the Sun ONE Studio programming environment should be able use Sun ONE Studio ME effectively within hours.

However, Sun ONE Studio ME lacks extra J2ME-specific features to differentiate it from competitor's IDEs. Most of the features provided are simply leveraging features from the existing Sun ONE Studio programming environment and targeting them at the J2ME developer.

The reviewed version of Sun ONE Studio ME was an early access version, so there may be some additional J2ME-specific features in the final release. During the review, some stability issues were encountered, with the IDE hanging on several occasions. It would be unfair not to assume these stability issues will be fixed in the final release.

Sun ONE Studio ME is a free download. The early access release is licensed for evaluation purposes only. However, the final release is expected to have no restrictions with regard to commercial use. With this in mind, for a “no cost, no frills” J2ME IDE, Sun ONE Studio ME is hard to beat.

