



# Developing Mobile Applications

---



VERSION 9

**Borland®**  
**JBuilder®**

Borland Software Corporation  
100 Enterprise Way, Scotts Valley, CA 95066-3249  
[www.borland.com](http://www.borland.com)

Refer to the file `deploy.html` located in the `redist` directory of your JBuilder product for a complete list of files that you can distribute in accordance with the JBuilder License Statement and Limited Warranty.

Borland Software Corporation may have patents and/or pending patent applications covering subject matter in this document. Please refer to the product CD or the About dialog box for the list of applicable patents. The furnishing of this document does not give you any license to these patents.

COPYRIGHT © 1997–2003 Borland Software Corporation. All rights reserved. All Borland brand and product names are trademarks or registered trademarks of Borland Software Corporation in the United States and other countries. All other marks are the property of their respective owners.

For third-party conditions and disclaimers, see the Release Notes on your JBuilder product CD.

Printed in the U.S.A.

JBE0090WW21003mobile 6E6R0503  
0304050607-9 8 7 6 5 4 3 2 1  
PDF

# Contents

## Chapter 1

### **Introduction 1-1**

Contents . . . . .	1-1
Resource links . . . . .	1-3
Documentation conventions . . . . .	1-4
Developer support and resources . . . . .	1-5
Contacting Borland Technical Support. . . . .	1-5
Online resources . . . . .	1-6
World Wide Web . . . . .	1-6
Borland newsgroups . . . . .	1-6
Usenet newsgroups . . . . .	1-7
Reporting bugs . . . . .	1-7

## Chapter 2

### **JBuilder Mobile development environment 2-1**

Overview . . . . .	2-1
Setting up a JDK . . . . .	2-2
Adding a J2ME MIDP/CLDC JDK . . . . .	2-2
Specifying a JDK for a new project . . . . .	2-5
Specifying a JDK for an existing project . . . . .	2-5
Specifying a default JDK for all projects . . . . .	2-5
Specifying a device for the emulator . . . . .	2-6
Designing a MIDP UI . . . . .	2-7
Compiling, running, and debugging a MIDP application . . . . .	2-9
Using the Screen Manager . . . . .	2-10
Obfuscating MIDlet class files . . . . .	2-12
Configuring an obfuscator . . . . .	2-12
Obfuscating the files . . . . .	2-15
Setting up JBuilder for J2ME Palm Application development . . . . .	2-16

## Chapter 3

### **Creating and managing MIDP projects 3-1**

Creating a new project with the Project Wizard. . . . .	3-2
Adding the MIDlet files to the project . . . . .	3-5
Setting MIDP project properties . . . . .	3-5
Runtime configurations . . . . .	3-6
Main class . . . . .	3-8
JAD file. . . . .	3-8
OTA URL . . . . .	3-8
VM parameters . . . . .	3-8

Emulator parameters . . . . .	3-9
Emulator device . . . . .	3-9
MIDlet parameters. . . . .	3-9
Build target . . . . .	3-9

## Chapter 4

### **Building MIDP applications 4-1**

Compiling . . . . .	4-1
Types of compiling . . . . .	4-1
What can be compiled . . . . .	4-2
Running . . . . .	4-2
Running MIDlets . . . . .	4-2
Running from the MIDlet file(s) . . . . .	4-2
Running from a JAD file . . . . .	4-3
Selecting which device to use. . . . .	4-5
Debugging. . . . .	4-6
MIDlet debugging in JBuilder . . . . .	4-6
Debugger limitations . . . . .	4-6

## Chapter 5

### **Creating a MIDP user interface 5-1**

Overview . . . . .	5-1
MIDP Screens . . . . .	5-1
MIDP UI components . . . . .	5-3
MIDP UI class hierarchy . . . . .	5-5
Creating a UI . . . . .	5-5
Creating a project . . . . .	5-6
Creating the MIDlet and Displayable classes. . . . .	5-7
Using the UI designer . . . . .	5-9
Requirements for a MIDP class to be designable . . . . .	5-9
Opening the UI designer. . . . .	5-10
Adding components to the design . . . . .	5-11
Moving components . . . . .	5-12
Deleting components . . . . .	5-12
Copying components . . . . .	5-12
Undoing and redoing an action . . . . .	5-13
Setting properties in the inspector. . . . .	5-13
Creating a Canvas. . . . .	5-13
Creating a Ticker . . . . .	5-14
MIDP events. . . . .	5-16
Overview. . . . .	5-16
Generating MIDP event-handling code . . . . .	5-16

Using images with MIDlets. . . . .	5-17
Adding PNG images to a MIDlet	
Displayable . . . . .	5-17
Specifying an image to represent	
the MIDlet Suite . . . . .	5-17
Specifying images to represent MIDlets	
within a MIDlet Suite . . . . .	5-18
Customizing the component palette. . . . .	5-18
Component initialization . . . . .	5-19
Viewing class reference documentation	
for MIDP classes . . . . .	5-20

## Chapter 6

### **MIDP database programming 6-1**

Overview . . . . .	6-1
Using RMS . . . . .	6-1

## Chapter 7

### **Archiving MIDlets 7-1**

Archiving MIDlets from the command line. . .	7-1
Creating the manifest file . . . . .	7-2
MIDlet manifest attributes. . . . .	7-2
Creating the JAR file outside of JBuilder. . .	7-4
Creating the JAD file outside of JBuilder . .	7-5
JAD file attributes. . . . .	7-5
Archiving MIDlets with the Archive Builder . .	7-6
JAR file contents . . . . .	7-15
Obfuscated class files. . . . .	7-15
JAD file contents . . . . .	7-16

## Chapter 8

### **Developing Mobile Applications: Over The Air (OTA) Provisioning 8-1**

What is OTA Provisioning? . . . . .	8-1
What's involved in OTA Provisioning? . . . .	8-2
Configuring the server for JAD and JAR	
MIME types. . . . .	8-3
OTA Provisioning . . . . .	8-4
Uploading your MIDlet Suite to a server . .	8-4
Downloading MIDlet Suites from the	
server. . . . .	8-6
Running an installed MIDlet Suite . . . . .	8-8
Running a MIDlet Suite directly from	
a server. . . . .	8-9
Resources on OTA Provisioning . . . . .	8-10

## Chapter 9

### **JBuilder Mobile development samples 9-1**

Running MIDP samples in JBuilder. . . . .	9-1
Running from the MIDlet file . . . . .	9-1
Running from a JAD file . . . . .	9-2
Using the emulator . . . . .	9-3
Samples delivered with JBuilder Mobile	
development. . . . .	9-3
Hello World . . . . .	9-3
MIDP UI Demo . . . . .	9-3
Stock Tracker. . . . .	9-4
Stop Watch. . . . .	9-5
MobileBookStore . . . . .	9-5
Overview . . . . .	9-5
Description . . . . .	9-5
Running the MobileBookStore sample . .	9-6

## Chapter 10

### **Tutorial: Creating and testing MIDlets 10-1**

Creating a project . . . . .	10-2
Creating MIDlet1 . . . . .	10-4
Designing the UI for MIDlet1 . . . . .	10-7
Adding a StringItem . . . . .	10-9
Adding a DateField . . . . .	10-12
Adding a Ticker . . . . .	10-13
Compiling and saving the project. . . . .	10-14
Testing MIDlet1 . . . . .	10-15
Adding additional MIDlets . . . . .	10-16
Designing the UI for the new MIDlets . . . .	10-16
Adding an image to the project . . . . .	10-16
Putting the image in Displayable2. . . . .	10-17
Adding a gauge to Displayable3. . . . .	10-18
Testing all three MIDlets . . . . .	10-18
Archiving the MIDlet Suite . . . . .	10-19
Running from the JAD file. . . . .	10-21
Using Micro-Run . . . . .	10-21
Specifying the default file to run. . . . .	10-21

## Chapter 11

### **Tutorial: Stopwatch MIDlet 11-1**

Overview . . . . .	11-1
Part 1: Creating the project and core	
MIDlet files . . . . .	11-3
Step 1: Creating a project for the MIDlet . .	11-3
Step 2: Creating the MIDlet files . . . . .	11-4

Part 2: Creating the MainMenu	
functionality . . . . .	11-6
Step 1: Adding the MainMenu class	
constants. . . . .	11-7
Step 2: Adding the MainMenu instance	
variables. . . . .	11-7
Step 3: Modifying the super() call to	
use new list items. . . . .	11-7
Step 4: Adding event handling to	
commandAction() method . . . . .	11-8
Step 5: Creating a destroy() method . . . . .	11-9
Part 3: Creating the basic stopwatch	
functionality . . . . .	11-9
Step 1: Creating a new Displayable. . . . .	11-9
Step 2: Importing additional classes . . . . .	11-10
Step 3: Creating the instance variables. . . . .	11-10
Step 4: Changing the constructor . . . . .	11-11
Step 5: Initializing the stopwatch	
commands. . . . .	11-12
Step 6: Handling the stopwatch	
commands. . . . .	11-12
Step 7: Creating a destroy() method . . . . .	11-14
Step 8: Creating the TimerTask . . . . .	11-15
Part 4: Painting the time on the Canvas . . . . .	11-15
Step 1: Clearing the Canvas . . . . .	11-16
Step 2: Finding out the current time . . . . .	11-16
Step 3: Changing the pen color . . . . .	11-16
Step 4: Specifying the font. . . . .	11-16
Step 5: Drawing the current time . . . . .	11-17
Part 5: Formatting the time . . . . .	11-17
Step 1: Creating the formatTime()	
method. . . . .	11-17
Step 2: Checking how many seconds. . . . .	11-18
Step 3: Creating strings for minutes	
and seconds . . . . .	11-18
Step 4: Creating the final time. . . . .	11-18
Step 5: Initializing the font at the	
largest possible size . . . . .	11-19
Step 6: Compiling and running the	
MIDlet . . . . .	11-20
Part 6: Creating the Stopwatch options. . . . .	11-20
Step 1: Creating the Options Displayable. . . . .	11-20
Step 2: Adding Class Constants . . . . .	11-21
Step 3: Creating the Instance Variables . . . . .	11-21
Step 4: Modifying the constructor . . . . .	11-22
Step 5: Adding OK and Cancel to	
the jbInit() . . . . .	11-22
Step 6: Adding OK and Cancel	
command handling . . . . .	11-23
Step 7: Creating getter and setter	
methods for the time format . . . . .	11-23
Step 8: Adding support for Options in	
MainMenu . . . . .	11-24
Part 7: Changing the time display based	
on the option. . . . .	11-25
Part 8: Writing options to/from RMS	
record store . . . . .	11-25
Step 1: Importing the RMS package. . . . .	11-26
Step 2: Defining the database name . . . . .	11-26
Step 3: Changing the display variable	
name . . . . .	11-26
Step 4: Creating the loadOptions()	
method . . . . .	11-26
Step 5: Creating the saveOptions()	
method . . . . .	11-28
Step 6: Loading saved user options on	
startup . . . . .	11-29
Step 7: Saving user options before	
quitting . . . . .	11-29
Step 8: Saving and testing the MIDlet. . . . .	11-29

## Index I-1



# Tables

1.1	Typeface and symbol conventions . . . .	1-4	5.1	Table of MIDP screens. . . . .	5-2
1.2	Platform conventions. . . . .	1-5	5.2	Table of MIDP UI components . . . . .	5-4

# Figures

2.1	Project wizard, Step 2 . . . . .	2-5	7.10	Archive Builder, Step 8 . . . . .	7-13
2.2	MIDP displayable in the UI designer . . . . .	2-8	7.11	Archive Builder, Step 9 . . . . .	7-13
2.3	MIDP components on the palette . . . . .	2-9	7.12	Project pane after archiving, but before rebuilding project . . . . .	7-14
3.1	Project wizard, Step 1 . . . . .	3-2	7.13	Project pane after archiving and rebuilding project . . . . .	7-14
3.2	Project wizard, Step 2 . . . . .	3-3	7.14	Manifest file . . . . .	7-15
3.3	Project wizard, Step 3 . . . . .	3-4	7.15	Comparison of obfuscated and unobfuscated JAR files . . . . .	7-15
3.4	Project pane after running the Project wizard . . . . .	3-5	7.16	JAD file contents . . . . .	7-16
3.5	Project Properties dialog box . . . . .	3-6	8.1	MIDlet Suite lifecycle . . . . .	8-2
3.6	Run tab of the Project Properties dialog box . . . . .	3-7	8.2	OTA Provisioning recommended practice . . . . .	8-3
3.7	Run tab for MIDlet type in the Runtime Configuration Properties dialog box . . . . .	3-7	10.1	Project wizard, Step 1 . . . . .	10-2
3.8	Running multiple MIDlets from the project pane . . . . .	3-8	10.2	Project wizard, Step 2 . . . . .	10-3
4.1	Running multiple MIDlets from the project pane . . . . .	4-3	10.3	Project wizard, Step 3 . . . . .	10-3
4.2	JBuilder debugging a MIDlet . . . . .	4-6	10.4	Object gallery . . . . .	10-4
5.1	MIDP UI classes . . . . .	5-5	10.5	MIDP MIDlet wizard, Step 1 . . . . .	10-5
5.2	Project wizard, Step 1 . . . . .	5-6	10.6	MIDP MIDlet wizard, Step 2 . . . . .	10-5
5.3	Project wizard, Step 2 . . . . .	5-6	10.7	MIDP MIDlet wizard, Step 3 . . . . .	10-6
5.4	Object Gallery Micro page . . . . .	5-7	10.8	AppBrowser after running Project and MIDP MIDlet wizards . . . . .	10-7
5.5	MIDP MIDlet wizard, Step 1 . . . . .	5-7	10.9	UI designer . . . . .	10-8
5.6	MIDP MIDlet wizard, Step 2 . . . . .	5-8	10.10	Component palette tooltips . . . . .	10-8
5.7	AppBrowser after running the Project and MIDP MIDlet wizards . . . . .	5-9	10.11	stringItem1 displayed in the component tree and the Form . . . . .	10-10
5.8	UI designer . . . . .	5-10	10.12	Setting the text property for stringItem1 . . . . .	10-10
5.9	MIDP components on the palette . . . . .	5-11	10.13	stringItem1 text visible in designer . . . . .	10-11
5.10	Adding a component in the designer . . . . .	5-11	10.14	Adding dateField1 to the Form . . . . .	10-12
5.11	Dragging a component in the designer . . . . .	5-12	10.15	Dragging stringItem1 . . . . .	10-12
5.12	Creating a Ticker and setting string text . . . . .	5-15	10.16	After dragging stringItem1 below dateField1 . . . . .	10-13
5.13	Setting 'this' to 'ticker1' in the Inspector . . . . .	5-15	10.17	Creating a Ticker and setting string text . . . . .	10-13
7.1	Archive Builder, Step 1 . . . . .	7-7	10.18	Setting this to ticker1 in the Inspector . . . . .	10-14
7.2	Archive Builder, Step 2 . . . . .	7-7	10.19	Project menu . . . . .	10-14
7.3	Archive Builder, Step 3 . . . . .	7-8	10.20	Make and Rebuild toolbar buttons . . . . .	10-15
7.4	Archive Builder, Step 4 . . . . .	7-9	10.21	MIDlet1 displayed on the emulator screen . . . . .	10-15
7.5	Archive Builder, Step 5 . . . . .	7-9	10.22	MIDlet Tutorial when first running in the emulator . . . . .	10-15
7.6	Archive Builder, Step 6 . . . . .	7-10	10.23	Completed MIDlet tutorial running in the emulator . . . . .	10-16
7.7	Archive Builder, Step 7 . . . . .	7-11	11.1	StopWatch Displayable MIDlet map . . . . .	11-2
7.8	Archive Builder, MIDlet-n Attributes dialog box . . . . .	7-11			
7.9	Archive Builder, Select a main class for MIDlet dialog box . . . . .	7-12			



# Tutorials

Creating and testing MIDlets . . . . .	10-1	StopWatch MIDlet . . . . .	11-1
--	------	----------------------------	------



# Introduction

Mobile development is a  
feature of JBuilder  
Developer and Enterprise

*Developing Mobile Applications* explains how to use the JBuilder integrated development environment to create and archive MIDP applications for J2ME™ enabled hand-held devices. This document discusses how to manage projects and files, visually design a user interface, and compile, run, debug, and archive MIDP applications.

*Developing Mobile Applications* only presents concepts and aspects of using JBuilder that are unique to wireless application development. For general details on using the JBuilder IDE, see the core JBuilder documentation.

It is beyond the scope of this document to instruct you on how to do Java or J2ME programming. However, a list of appropriate links to resources for this information is presented at the end of this introduction.

**Note** The emulator and phone UI images shown in this documentation are for illustrative purposes and do not represent any specific device.

For an explanation of documentation conventions, see [“Documentation conventions” on page 1-4](#).

For definitions of any unfamiliar Java terms, see “Java glossaries” the topic “Learning more about Java” in *Introducing JBuilder*.

## Contents

---

*Developing Mobile Applications* contains the following chapters:

- [Chapter 2, “JBuilder Mobile development environment”](#)

Introduces the JBuilder integrated development environment (IDE) user interface. It also explains how to set up and configure a JDK for MIDP development.

- [Chapter 3, “Creating and managing MIDP projects”](#)

Discusses how to create and manage MIDP projects and files in JBuilder and how to use the AppBrowser to perform management tasks.
- [Chapter 4, “Building MIDP applications”](#)

Explains how to compile, run, and debug MIDP applications. Discusses the difference between Make and Rebuild. Gives instructions on running MIDlets in the emulator inside JBuilder, as well as running them from the command line.
- [Chapter 5, “Creating a MIDP user interface”](#)

Explores each of the MIDP UI elements and gives details on designing a user interface using UI components and the JBuilder’s visual design tools.
- [Chapter 6, “MIDP database programming”](#)

Introduces and gives a brief overview of programming MIDP databases with the RMS package.
- [Chapter 7, “Archiving MIDlets”](#)

Discusses how to archive MIDlets into JAR files using either the JBuilder Archive Builder or the command line JAR tools. Also explains how to create the Manifest and JAD files.
- [Chapter 8, “Developing Mobile Applications: Over The Air \(OTA\) Provisioning”](#)

Explains how to do OTA Provisioning of MIDlets from within JBuilder. Tells you how to upload your MIDlet Suites to an FTP server without leaving JBuilder, then test them in the emulator by downloading and running them using OTA Provisioning. It also explains how to discover and run third-party MIDlets on the Internet.
- [Chapter 9, “JBuilder Mobile development samples”](#)

Explains how to run the samples in JBuilder. Provides a description of each sample included with JBuilder Mobile development with a link to the project file for the sample.
- [Chapter 10, “Tutorial: Creating and testing MIDlets”](#)

Introduces you to MIDlet design using the JBuilder development and design tools such as the wizards, editor, compiler, and UI designer. Steps you through creating a simple MIDlet with a user interface and testing it in the emulator which is integrated into JBuilder.
- [Chapter 11, “Tutorial: Stopwatch MIDlet”](#)

Explores several basic concepts in J2ME MIDP application development. It teaches you how to draw on a Canvas, set colors, fill

rectangles with color, set fonts, and draw strings. It also demonstrates creating a MIDlet that uses multiple Displayables and how to move from one Displayable to the other. The last section shows you how to save and retrieve information from a Record Management Store (RMS) database.

## Resource links

---

- **Java™ 2 Platform, Micro Edition (J2ME™ Platform)**, at <http://java.sun.com/j2me/>
- **CLDC and the K Virtual Machine (KVM)**, at <http://java.sun.com/products/cldc/>
- **Mobile Information Device Profile (MIDP)**, at <http://java.sun.com/products/midp/>
- **J2ME™ Connected Limited Device Configuration (CLDC)**, at <http://www.sun.com/software/communitysource/j2me/cldc/>
- **Wireless Tech Tips**, at <http://wireless.java.sun.com/midp/ttips/>
- **Java™ 2 Platform, Micro Edition, Frequently Asked Questions**, at <http://java.sun.com/j2me/faq.html>
- **Wireless Highlights and News**, at <http://wireless.java.sun.com/index.jsp>
- *Sams Teach Yourself Wireless Java with J2ME in 21 Days* by Michael Morrison, at <http://btobshop.barnesandnoble.com/booksearch/isbnInquiry.asp?userid=1A9J84QMLR&mscssid=PEHD64XWM7WK9PXQVH2LRGGG0T6G79QF&btob=Y&isbn=0672321424&vm=&from=SWP279>
- *Core J2ME Technology and MIDP* by John W. Muchow, at <http://btobshop.barnesandnoble.com/booksearch/isbnInquiry.asp?userid=1A9J84QMLR&mscssid=PEHD64XWM7WK9PXQVH2LRGGG0T6G79QF&btob=Y&isbn=0130669113&vm=&from=SWP279>
- *Instant Wireless Java™ with J2ME™* by Paul Tremblett, at <http://btobshop.barnesandnoble.com/booksearch/isbnInquiry.asp?userid=1A9J84QMLR&mscssid=PEHD64XWM7WK9PXQVH2LRGGG0T6G79QF&btob=Y&isbn=0072191759&vm=&from=SWP279>
- *Mobile Information Device Profile for Java™ 2 Micro Edition (J2ME): Professional Developer's Guide* by C. Enrique Oritz, Eric Giguere, at <http://btobshop.barnesandnoble.com/booksearch/isbnInquiry.asp?userid=1A9J84QMLR&mscssid=PEHD64XWM7WK9PXQVH2LRGGG0T6G79QF&btob=Y&isbn=0471034657&vm=&from=SWP279>

# Documentation conventions

---

The Borland documentation for JBuilder uses the typefaces and symbols described in the following table to indicate special text.

**Table 1.1**    Typeface and symbol conventions

Typeface	Meaning
Monospaced type	<p>Monospaced type represents the following:</p> <ul style="list-style-type: none"><li>• text as it appears onscreen</li><li>• anything you must type, such as “Type <code>Hello World</code> in the Title field of the Application wizard.”</li><li>• file names</li><li>• path names</li><li>• directory and folder names</li><li>• commands, such as <code>SET PATH</code></li><li>• Java code</li><li>• Java data types, such as <code>boolean</code>, <code>int</code>, and <code>long</code>.</li><li>• Java identifiers, such as names of variables, classes, package names, interfaces, components, properties, methods, and events</li><li>• argument names</li><li>• field names</li><li>• Java keywords, such as <code>void</code> and <code>static</code></li></ul>
<b>Bold</b>	<p>Bold is used for java tools, <code>bmj</code> (Borland Make for Java), <code>bcj</code> (Borland Compiler for Java), and compiler options. For example: <b><code>javac</code></b>, <b><code>bmj</code></b>, <b><code>-classpath</code></b>.</p>
<i>Italics</i>	<p>Italicized words are used for new terms being defined, for book titles, and occasionally for emphasis.</p>
<i>Keycaps</i>	<p>This typeface indicates a key on your keyboard, such as “Press <i>Esc</i> to exit a menu.”</p>
[ ]	<p>Square brackets in text or syntax listings enclose optional items. Do not type the brackets.</p>
< >	<p>Angle brackets are used to indicate variables in directory paths, command options, and code samples.</p> <p>For example, <code>&lt;filename&gt;</code> may be used to indicate where you need to supply a file name (including file extension), and <code>&lt;username&gt;</code> typically indicates that you must provide your user name.</p> <p>When replacing variables in directory paths, command options, and code samples, replace the entire variable, including the angle brackets (<code>&lt; &gt;</code>). For example, you would replace <code>&lt;filename&gt;</code> with the name of a file, such as <code>employee.jds</code>, and omit the angle brackets.</p> <p><b>Note:</b> Angle brackets are used in HTML, XML, JSP, and other tag-based files to demarcate document elements, such as <code>&lt;font color=red&gt;</code> and <code>&lt;ejb-jar&gt;</code>. The following convention describes how variable strings are specified within code samples that are already using angle brackets for delimiters.</p>

**Table 1.1** Typeface and symbol conventions (continued)

Typeface	Meaning
<i>Italics, serif</i>	This formatting is used to indicate variable strings within code samples that are already using angle brackets as delimiters. For example, <code>&lt;url="jdbc:borland:jbuilder\\samples\guestbook.jds"&gt;</code>
...	In code examples, an ellipsis (...) indicates code that has been omitted from the example to save space and improve clarity. On a button, an ellipsis indicates that the button links to a selection dialog box.

JBuilder is available on multiple platforms. See the following table for a description of platform conventions used in the documentation.

**Table 1.2** Platform conventions

Item	Meaning
Paths	Directory paths in the documentation are indicated with a forward slash (/).  For Windows platforms, use a backslash (\).
Home directory	The location of the standard home directory varies by platform and is indicated with a variable, <code>&lt;home&gt;</code> . <ul style="list-style-type: none"> <li>For UNIX and Linux, the home directory can vary. For example, it could be <code>/user/&lt;username&gt;</code> or <code>/home/&lt;username&gt;</code></li> <li>For Windows NT, the home directory is <code>C:\Winnt\Profiles\&lt;username&gt;</code></li> <li>For Windows 2000 and XP, the home directory is <code>C:\Documents and Settings\&lt;username&gt;</code></li> </ul>
Screen shots	Screen shots reflect the Metal Look & Feel on various platforms.

## Developer support and resources

Borland provides a variety of support options and information resources to help developers get the most out of their Borland products. These options include a range of Borland Technical Support programs, as well as free services on the Internet, where you can search our extensive information base and connect with other users of Borland products.

### Contacting Borland Technical Support

Borland offers several support programs for customers and prospective customers. You can choose from several categories of support, ranging from free support on installation of the Borland product to fee-based consultant-level support and extensive assistance.

For more information about Borland's developer support services, see our web site at <http://www.borland.com/devsupport/>, call Borland Assist at (800) 523-7070, or contact our Sales Department at (831) 431-1064.

When contacting support, be prepared to provide complete information about your environment, the version of the product you are using, and a detailed description of the problem.

For support on third-party tools or documentation, contact the vendor of the tool.

## Online resources

---

You can get information from any of these online sources:

<b>World Wide Web</b>	<a href="http://www.borland.com/">http://www.borland.com/</a> <a href="http://www.borland.com/techpubs/jbuilder/">http://www.borland.com/techpubs/jbuilder/</a>
<b>Electronic newsletters</b>	To subscribe to electronic newsletters, use the online form at: <a href="http://www.borland.com/products/newsletters/index.html">http://www.borland.com/products/newsletters/index.html</a>

## World Wide Web

---

Check [www.borland.com/jbuilder](http://www.borland.com/jbuilder) regularly. This is where the Java Products Development Team posts white papers, competitive analyses, answers to frequently asked questions, sample applications, updated software, updated documentation, and information about new and existing products.

You may want to check these URLs in particular:

- <http://www.borland.com/jbuilder/> (updated software and other files)
- <http://www.borland.com/techpubs/jbuilder/> (updated documentation and other files)
- <http://community.borland.com/> (contains our web-based news magazine for developers)

## Borland newsgroups

---

When you register JBuilder you can participate in many threaded discussion groups devoted to JBuilder. The Borland newsgroups provide a means for the global community of Borland customers to exchange tips and techniques about Borland products and related tools and technologies.

You can find user-supported newsgroups for JBuilder and other Borland products at <http://www.borland.com/newsgroups/>.

## Usenet newsgroups

---

The following Usenet groups are devoted to Java and related programming issues:

- `news:comp.lang.java.advocacy`
- `news:comp.lang.java.announce`
- `news:comp.lang.java.beans`
- `news:comp.lang.java.databases`
- `news:comp.lang.java.gui`
- `news:comp.lang.java.help`
- `news:comp.lang.java.machine`
- `news:comp.lang.java.programmer`
- `news:comp.lang.java.security`
- `news:comp.lang.java.softwaretools`

**Note** These newsgroups are maintained by users and are not official Borland sites.

## Reporting bugs

---

If you find what you think may be a bug in the software, please report it to Borland at one of the following sites:

- **Support Programs** page at <http://www.borland.com/devsupport/namerica/>. Click the “Reporting Defects” link to bring up the Entry Form.
- **Quality Central** at <http://qc.borland.com>. Follow the instructions on the Quality Central page in the “Bugs Reports” section.

When you report a bug, please include all the steps needed to reproduce the bug, including any special environmental settings you used and other programs you were using with JBuilder. Please be specific about the expected behavior versus what actually happened.

If you have comments (compliments, suggestions, or issues) for the JBuilder documentation team, you may email [jpgpubs@borland.com](mailto:jpgpubs@borland.com). This is for documentation issues only. Please note that you must address support issues to developer support.

JBuilder is made by developers for developers. We really value your input.





# JBuilder Mobile development environment

## Overview

---

This is a feature of  
JBuilder Developer and  
Enterprise

JBuilder Mobile development enhances the JBuilder IDE with tools for creating mobile applications.

The working environment for Mobile development is the same as for normal Java development in JBuilder, with the exception of additional tabs and options in existing dialog boxes, and two wizards, the MIDP MIDlet wizard and the MIDP Displayable wizard. Also, the Archive Builder can create a MIDlet Suite and its corresponding manifest and JAD files, as well as obfuscate your class files.

Context-sensitive help is available by pressing *F1* or by clicking the Help button on dialog boxes. You can still access class reference documentation by choosing View | Browse Classes, selecting the class, then clicking the DOC tab. For more information on using the Help Viewer to access documentation, see “Using JBuilder’s Online Help” in *Introducing JBuilder*.

Sun’s J2ME Wireless Toolkit 1.0.4\_01 is bundled with JBuilder and configured for its use. You can also install additional J2ME JDKs, then specify which one to use for each project. You can also set one to use as the default JDK for all projects.

To install a new JDK to your computer, follow the setup instructions accompanying that specific JDK from the provider. After it is installed, configure it in JBuilder as explained in [“Setting up a JDK” on page 2-2](#).

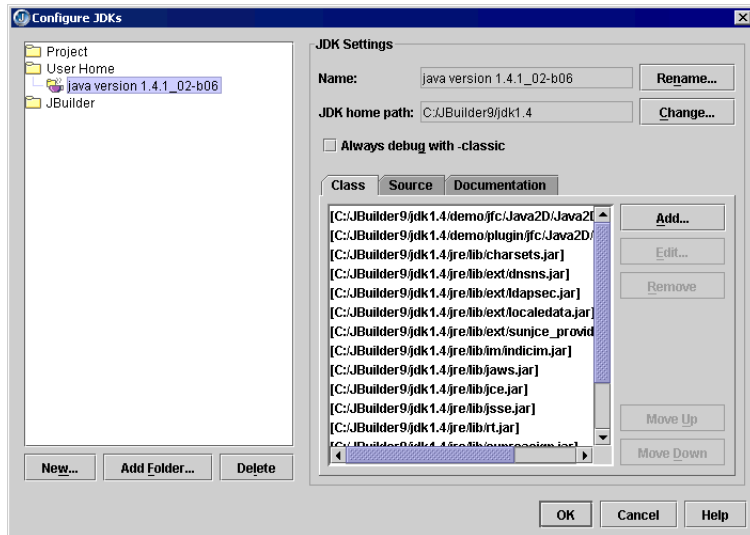
## Setting up a JDK

In JBuilder, you can define more than one JDK and switch JDKs on a project by project basis.

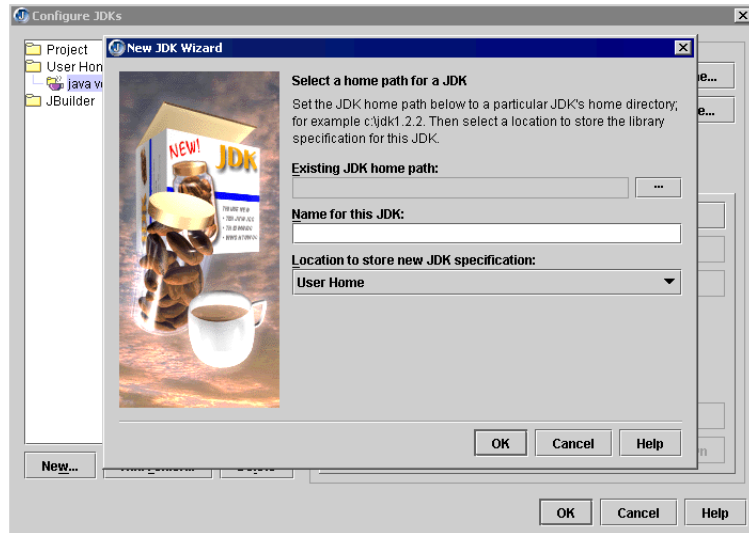
### Adding a J2ME MIDP/CLDC JDK

To add a new J2ME JDK to JBuilder,

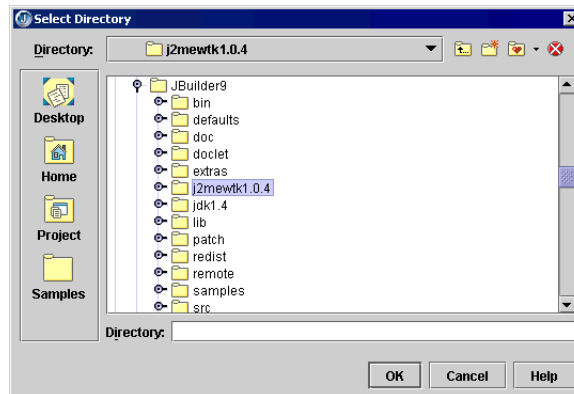
- 1 Choose Tools | Configure JDKs to open the Configure JDKs dialog box.



- 2 Click New to open New JDK wizard.

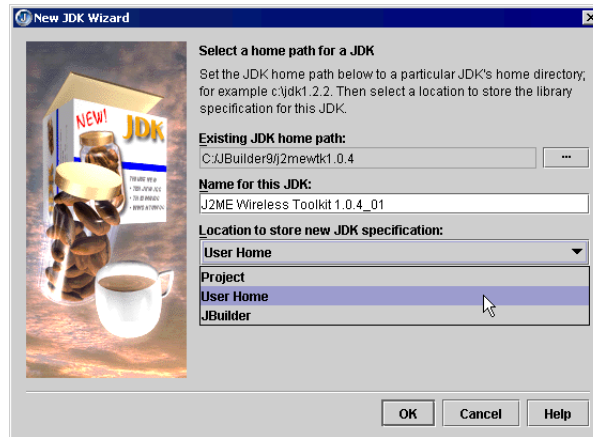


- 3 Click the ellipsis (...) button in the New JDK wizard to navigate to the installation location of the JDK.

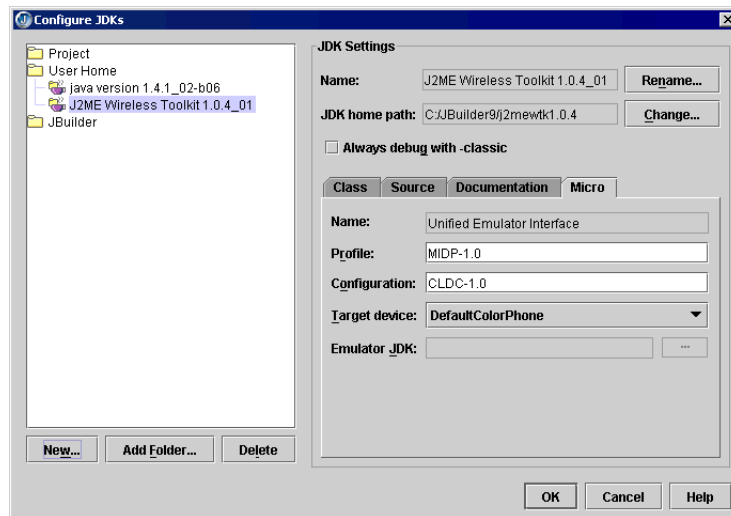


- 4 JBuilder detects the name of the JDK and fills in the name in the JDK field. You can change the name used for this JDK if you like in the New JDK wizard.

- 5 Click the drop-down arrow to specify a storage location for the JDK specifications .library file.



- 6 Click OK. The new JDK is added to the list and selected in the Configure JDKs dialog box. A Micro tab is added on the right which displays specific information about the JDK, such as which version of the CLDC and MIDP it is based on. You can also choose which device to use as the emulator.



- 7 Select a target device for the JDK on the Micro tab. This specifies which device to use for the emulator.

**Important**

The choice you make here determines what devices display in the Emulator Device list in the Runtime Configuration Properties dialog box. Only devices that use exactly the same libraries as the device specified here are displayed in the Runtime Configuration Properties dialog box list. If you don't see the device you want listed in the

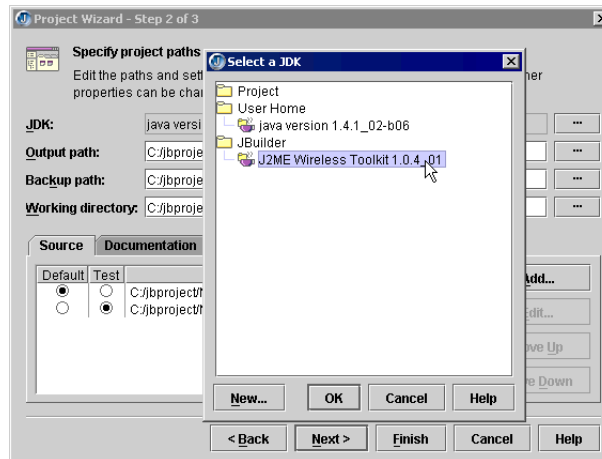
Runtime Configuration Properties dialog box, come back to this field and change the Target Device.

- 8 Click OK to close the Configure JDKs dialog box.

## Specifying a JDK for a new project

To specify a JDK when you create a new project, click the ellipsis (...) by the JDK field in Step 2 of the Project wizard and choose a new JDK.

**Figure 2.1** Project wizard, Step 2



## Specifying a JDK for an existing project

To specify a JDK for an existing project, open the project in JBuilder and do one of the following:

- Choose Project | Project Properties, click the ellipsis (...) button by the JDK field on the Paths page, and choose a JDK.
- Right-click the project in the project pane, choose Properties, then click the ellipsis (...) button by the JDK field on the Paths page and choose a JDK.

## Specifying a default JDK for all projects

To specify a default MIDP/CLDC JDK for all projects, choose Project | Default Project Properties, click the ellipsis (...) button by the JDK field on the Paths page, and choose a default JDK.

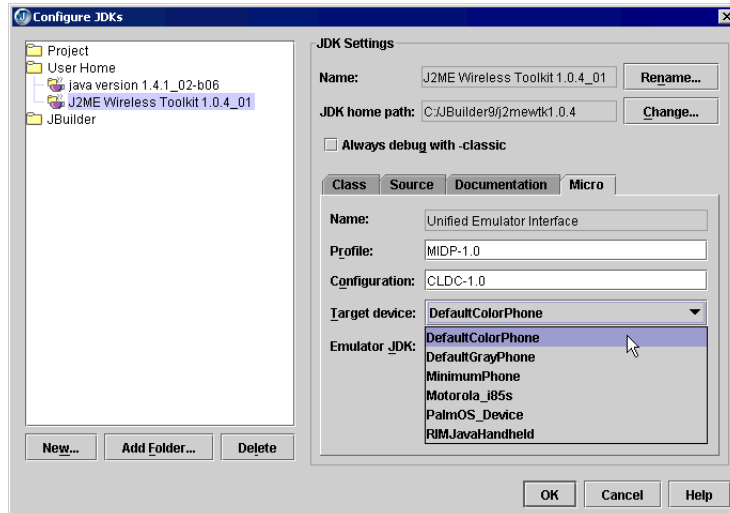
## Specifying a device for the emulator

If the JDK you are using supports multiple devices, you can specify which device you want to use in two places: the Configure JDKs dialog box and the Project Properties dialog box.

### In the Configure JDKs dialog box

When a MIDP/CLDC JDK is selected in the Configure JDKs dialog box, an additional tabbed page is added called Micro which displays specific information about that JDK. You can select a device for the emulator on this page to use as the default for this JDK as follows:

- 1 Click the down-arrow at the right of the Target Device field on the Micro page.
- 2 Select a device from the list, then click OK.

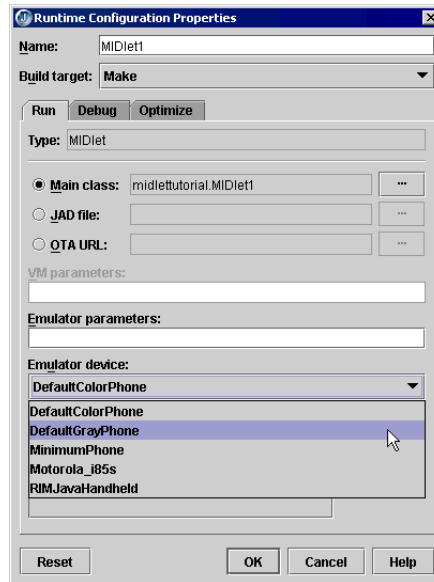


### In the Project Properties dialog box

To override the default emulator device setting in the Project Properties,

- 1 Choose Project | Project Properties or Project | Default Project Properties.
- 2 Select a runtime configuration from the Run page and click Edit, or if you have none, click New and give it a unique name.
- 3 Select MIDlet from the Type drop-down list on the Run page.
- 4 Select a device from the drop-down list in the Emulator Device field. This choice determines what devices are available for selection in the Runtime Configuration Properties dialog box.

**Note** If you are doing Palm development, it is recommended that you create a separate JDK configuration for Palm and call it something like “J2MEWTK for Palm.”



When you select a device in the Default Project Properties dialog box, each new MIDP project you create afterwards will use that device by default. To override this for a specific project, modify the Emulator Device setting in the Project Properties dialog box after creating a new project.

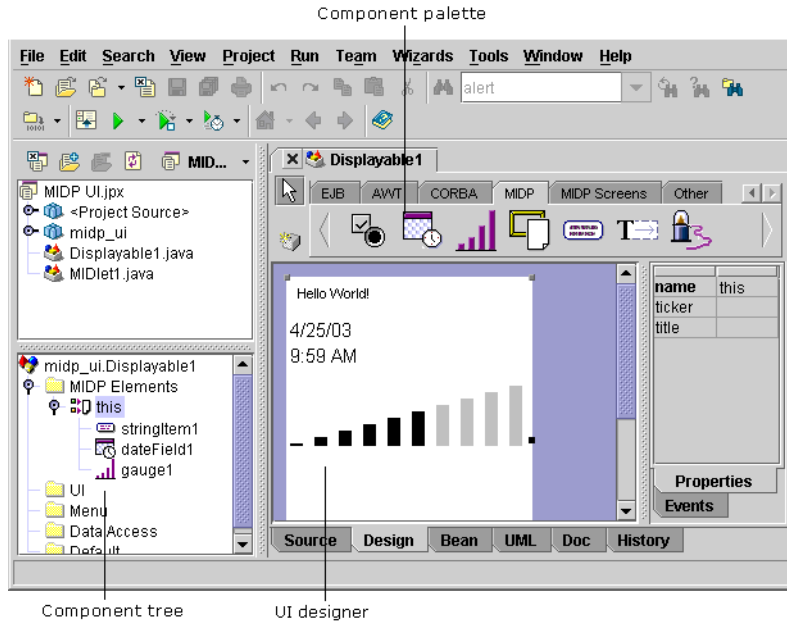
**Important** Only devices that use exactly the same libraries as the target device specified on the Micro tab in the Configure JDKs dialog box (Tools | Configure JDKs | Micro tab) will be shown in the Target Device list. If the desired device is not listed, then you need to go to Tools | Configure JDKs and select the target device.

For more information on setting project properties, see [Chapter 3, “Creating and managing MIDP projects.”](#)

## Designing a MIDP UI

You can design a MIDP user interface visually in JBuilder using the UI designer. To open the UI designer, open a MIDP `Displayable` class file in the AppBrowser and click the Design tab. Click the small right arrow at the right of the tabs on the component palette to scroll to the MIDP and MIDP Screens tabs.

**Important** Only a `Displayable` class can be designed. A `Displayable` class contains the UI for the display screen on the device, similar to the `Frame` or `Panel` container in a normal Java application.

**Figure 2.2** MIDP displayable in the UI designer

**Tip** For development convenience, you can customize the order of the tabs on the component palette to put the Mobile development tabs at the beginning. Choose Tools | Configure Palette, select a MIDP tab in the tab list and click Move Up or Move Down to position the tab.

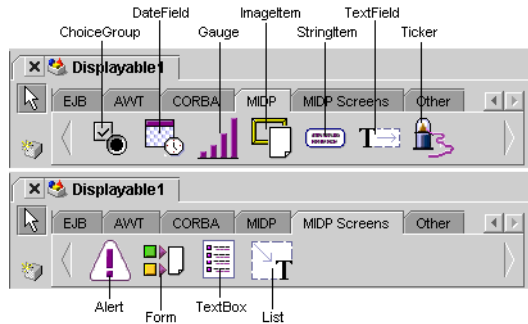
You can use the JBuilder MIDP MIDlet wizard or the MIDP Displayable wizard to create a `Displayable` class for your MIDlet. Once you have created a `Displayable`, you can visually customize the user interface for it in the UI designer. The `Displayable` is represented in the component tree by a node called `this`.

If you need additional `Displayable` classes for your MIDlet, you can add them to your project using the MIDP Displayable wizard. After adding new `Displayable` classes, you need to hook them up to the main MIDlet class.

Since MIDP UI components are not actually JavaBeans, the UI designer has been modified to be able to work with these components visually. When you add MIDP components in the designer, they are displayed in the component tree under a folder called MIDP Elements.



**Figure 2.3** MIDP components on the palette



For information on creating a MIDP UI in the designer, see [Chapter 5, “Creating a MIDP user interface.”](#)

### See also

- “Visual design in JBuilder” in *Designing Applications with JBuilder*.
- “Managing the component palette” in *Designing Applications with JBuilder*.

## Compiling, running, and debugging a MIDP application

Compiling, running, and debugging a MIDP application in JBuilder is basically the same as for normal Java programs in JBuilder. You can do one of two things to run and debug:

- Click the Run or Debug button on the toolbar.
- Right-click the `MIDlet` file in the project pane and choose Micro Run or Micro Debug.

You can create multiple runtime and debug configurations in JBuilder to accomplish different tasks. You can also set one of the configurations as the default, or you can have no default and always choose from the configuration list. If you have multiple runtime configurations, these will appear for selection on the run or debug context menus, or on the run and debug toolbar button drop-down menus.

**Note** When you create a MIDlet using the MIDP MIDlet wizard, you can choose to have a runtime configuration created at that time.

For a complete description of running and debugging in JBuilder, see “Running Java programs” and “Setting runtime configurations” in *Building Applications with JBuilder*.

### See also

- [Chapter 4, “Building MIDP applications”](#)

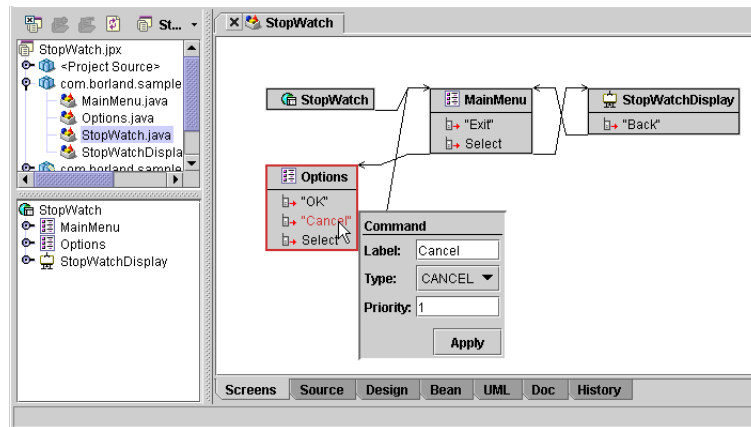
## Using the Screen Manager

A MIDP Screen Manager has been added to help you understand and manage MIDP `Displayables` associated with a MIDlet. The MIDP Screen Manager provides a conceptual view of the relationships for `Displayables` associated with a given MIDlet.

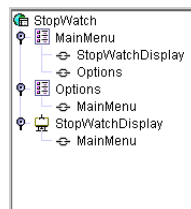
The Screen Manager is a two-way tool that lets you switch easily between the Manager screen and the source code so you can quickly modify the code then visually see the changes. For any MIDlet class, it can trace the source code and generate the complete screen.

**Important** Projects must be compiled for the Screen Manager to work.

To access the MIDP Screen Manager, double-click the MIDlet file to open it in the editor, then click the Screens tab at the bottom of the opened MIDlet file.



The MIDP Screen Manager parses the `.java` files in your project to determine what MIDP `Displayables` are associated with the open MIDlet, and draws representations of the MIDlet and the associated `Displayables` in the content pane of the editor. The different types of `Displayables` (Alert, Form, List, TextBox, and Canvas) are indicated by icons on the nodes in the structure pane, and icons beside the titles on the `Displayables` in the Screen Manager. You can also see these relationships by expanding the `Displayable` nodes in the structure pane.

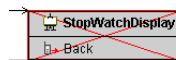


Calls from one `Displayable` to another show up as nodes of the given `Displayable` in the structure pane when the Screen Manager is open. You can also go to the method call in the source code where the displayable is being set by double-clicking on a node in the structure pane.

On the Screen Manager display, arrows linking the `MIDlet` and the `Displayables` represent calls from the `MIDlet` to the `Displayables`, and indicate the flow from one `MIDP Displayable` to another. The arrows start from the bottom of a given `Displayable` representation and connect to the top of the next.

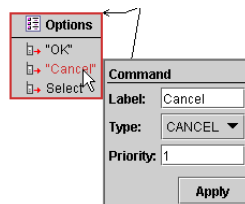
You can rearrange the `MIDlet` and `MIDP Displayable` representations in the Screen Manager by dragging them with the mouse to provide a better view of the relationships between `Displayables`. Your Screen Manager layout is maintained from one `JBuilder` session to another.

The Screen Manager does dynamic error checking. If there are errors in a `Displayable` class, the associated screen in the Screen Manager will have an X through it as shown below.



To use the Screen Manager,

- Double-click on a `Displayable` in the Screen Manager to open the source file for the `Displayable` in the editor. You can use the Search menu (Search | Back) or click the Back button on the toolbar to return to the Screen Manager after opening the source file for a displayable.
- Click on a command in a representation of a `Displayable` in the Screen Manager to open an inspector that displays the label, screen type, and priority associated with the command.



You can edit the values for these command parameters in the inspector, and apply your changes. If you enter invalid data, such as characters in the priority field, the inspector displays the invalid data in red, and prevents you from applying the changes.

- Note** Command parameters must be constant (label, screen type, and priority) in order for the Screen Manager to display the data. Otherwise, the Screen Manager will display "<unparsable>" for the label in the `Displayable` representation. If the other parameters are not constants, they display as "<unparsable>" in the command inspector.
- Double-click on a command in a representation of a `Displayable` to open the source file for the `Displayable` in the editor on the highlighted command.
  - Double-click on an arrow link to a `Displayable` in the Screen Manager to open the source file in the editor for the class that calls that `Displayable`. The statement that shows the displayable is highlighted in the source code.
- Note** If you open a command dialog (single-click a command in a `Displayable`) in the Screen Manager, and you enter bad information (for example, characters when numbers are required), the dialog box displays the bad input in red, and prevents you from applying the changes. Also, if the code for a command is corrupted, or the Screen Manager cannot understand the code, the command in the `Displayable` in the Screen Manager shows as "<unparsable>".

## Obfuscating MIDlet class files

---

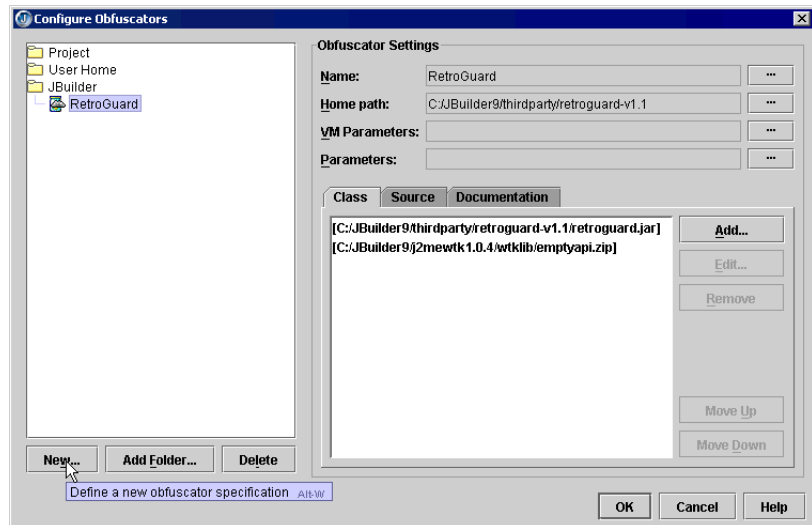
You can obfuscate your MIDlet class files as part of the archive process using the Archive Builder. For this to work, you need to install an obfuscation program and configure it in JBuilder. One obfuscator, called RetroGuard, is delivered with JBuilder and is configured for you to use. Right now, it is the only obfuscator supported in Mobile development.

### Configuring an obfuscator

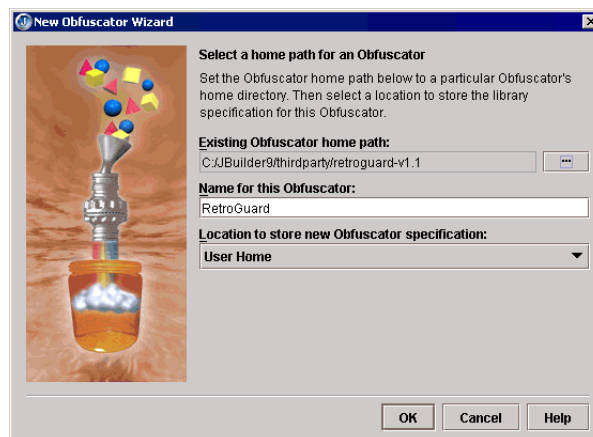
---

To configure an obfuscator, first make sure it is installed locally or on your intranet, then do the following:

- 1 Open JBuilder, choose Tools | Configure Obfuscators..., then click New to define a new obfuscator.

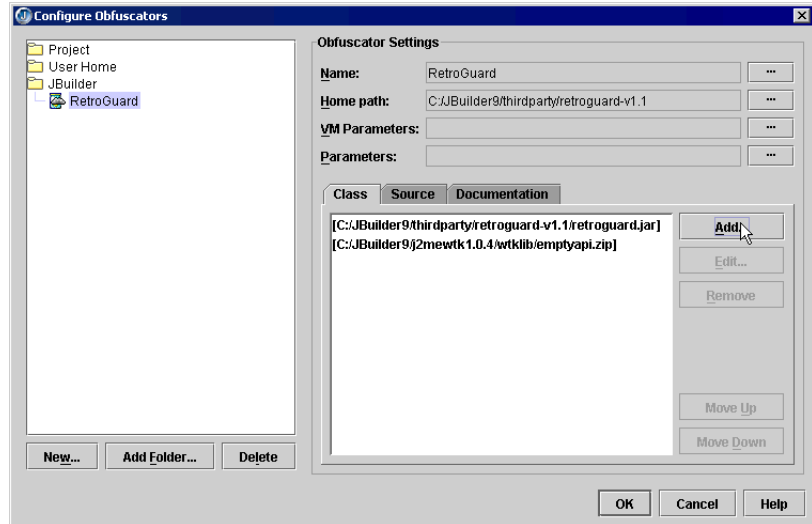


This opens the New Obfuscator wizard.

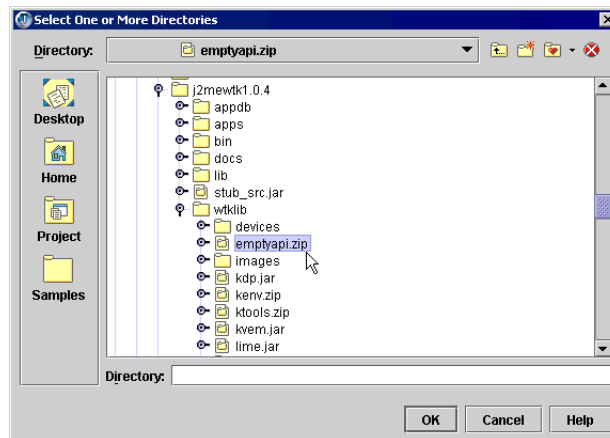


- 2 Click the ellipsis (...) button in the New Obfuscator wizard to navigate to the obfuscator home directory and select it.
- 3 Click OK to return to the Configure Obfuscators dialog box.
- 4 Enter any new parameters you want to use for this obfuscator.

5 Select the Class tab and click the Add button.



6 Navigate to the file <j2mewtk\_home>/wtklib/emptyapi.zip, select it, and click OK.



**Note** Depending on the implementation, some obfuscators (including RetroGuard) will load all classes referenced by your MIDlet Suite class files. If these referenced classes have native methods, the obfuscators will also try to load the shared libraries associated with those native methods. Since MIDP API classes use native methods, and the shared libraries can not be resolved, these obfuscators will not be able to obfuscate your MIDlet Suite. To work around this problem, you will need to have an emptied out version of MIDP API (`emptyapi.zip`) installed locally and added to your obfuscator class path.

This step is necessary if you want the obfuscator to work with JDKs other than the J2MEWTK1.0.4\_01 delivered with JBuilder. If you did not install the J2MEWTK1.0.4\_01 with JBuilder, you can download the `emptyapi.zip` file from Sun by clicking the *ZIP file* link on <http://wireless.java.sun.com/midp/questions/obfuscate/> or by going to <http://developer.java.sun.com/developer/qow/archive/161/emptyapi.zip> directly. Just drop the file into the same directory as your obfuscator and point to that one.

To add `emptyapi.zip` to an obfuscator class path in JBuilder,

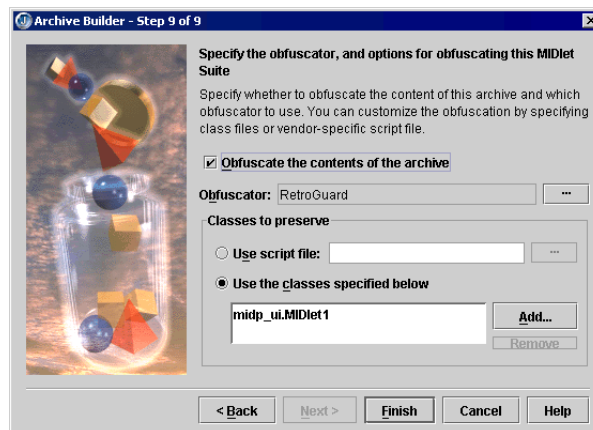
- a** Place a copy of `emptyapi.zip` in your obfuscator directory.
- b** Choose Tools | Configure Obfuscator.
- c** Select the obfuscator in the list on the left.
- d** Click the Class tab on the right.
- e** Click Add, navigate to `emptyapi.zip`, then click OK to add it to the list of classes.
- f** Click OK to close the Configure Obfuscators dialog box.

Repeat this process for any additional obfuscators you want to use with JBuilder.

## Obfuscating the files

To obfuscate your MIDlet class files,

- 1** Build and save your MIDlet project.
- 2** Choose Wizards | Archive Builder.
- 3** Fill in the steps of the wizard as desired up to the last step, which is where you control obfuscation. (For information on Archiving MIDlets, see [Chapter 7, “Archiving MIDlets.”](#))



- 4 Check Obfuscate the Contents of the Archive in Step 9, and click the ellipsis (...) button to choose an obfuscator.
  - 5 Specify which files to preserve during obfuscation. You can do this by selecting an obfuscator script file you have previously created to control the obfuscation, or you can click the Add button to manually list the files you want to leave unobfuscated. To remove a file from the list, select it and click Remove.
- Note** If your code has classes that are loaded using the `Class.forName()` method call, you should add these classes to preserve list.
- 6 Click Finish to generate the archive with obfuscated files.

**See also**

- “Obfuscating a J2ME™ MIDlet suite” at <http://wireless.java.sun.com/midp/questions/obfuscate/>

## Setting up JBuilder for J2ME Palm Application development

---

Before you begin any Palm development, go to the Palm OS® Emulator site at <http://www.palmos.com/dev/tech/tools/emulator/>, and read about the requirements for Palm development and the use of ROM images. This web page is updated regularly and contains important information for Palm developers. It is also the URL from which you download the Palm Emulator, sign up for the Palm OS® Developer Program, and obtain ROM image files.

You are strongly urged to sign up for the Palm OS® Developer Program so you can obtain debug ROMs for development. In the meantime, if you don't have access to a ROM yet, and you'd like to start development right away, you can upload a ROM image from most Palm devices into the Palm OS® Emulator using a serial cable. However, the ROMs on devices are non-debug ROMs, therefore they cover up errors and are not suitable for real development.

Also, if you did not choose to install the J2ME Wireless Toolkit when you installed JBuilder, you need to download it and install it locally before beginning Palm OS® development for J2ME.

Once you've installed JBuilder, the J2ME Wireless Toolkit, and Mobile development, follow the steps below to set up your JBuilder working environment for developing J2ME Palm applications:

- 1 Download the Palm OS® Emulator from <http://www.palmos.com/dev/tech/tools/emulator/>, and extract it to a directory called <drive name>\Palm\emulator.

**Important** The Palm OS® Emulator executable file (`emulator.exe`) must be in a directory that matches its name. If you want to install the Palm OS®



Emulator to a directory other than <drive name>\Palm\emulator, you must rename `emulator.exe` to match the directory name in which it resides. For example, if you put the emulator in a directory called <local drive>\Palm\pose, rename `emulator.exe` to `pose.exe`.

- 2 Download one or more ROM image files from Palm to the directory containing the Palm emulator. See the Palm OS® Emulator site at <http://www.palmos.com/dev/tech/tools/emulator/> for directions and links for doing this.

Alternatively, you can upload a ROM image directly from a Palm handheld device following these steps:

- a Locate the file `ROM Transfer.prc` in the Palm OS® directory. Use the Install tool in the Palm Desktop organizer software to install it.
- b Place the handheld in the HotSync cradle connected to the serial port on your computer, and synchronize with the handheld to install `ROM Transfer.prc` on the handheld.
- c Exit the HotSync Manager.
- d Transfer a ROM file to the emulator.

If this is the first time you've run the ROM Transfer program, Palm OS® Emulator displays a Startup dialog box. Click Download to begin the ROM transfer.

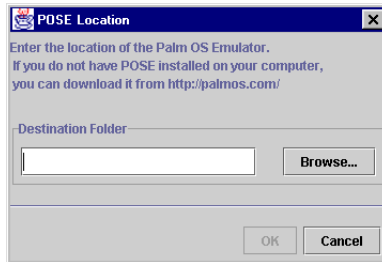
If it is not the first time you've run the emulator, Palm OS® Emulator usually restarts the session you most recently ran. In this case, right-click on the emulator display and select Transfer ROM.

For more information on downloading or transferring ROM images, see the document `UserGuide.pdf` delivered with the Palm OS® Emulator.

- 3 Open JBuilder and choose Tools | Configure JDKs to setup a new JDK for the J2ME Wireless Toolkit. Name it something like J2MEWTK for Palm, and select the PalmOS\_device as the Target Device on the Micro page of the Configure JDKs dialog box. See [“Setting up a JDK” on page 2-2](#).
- 4 Create a new project: Choose File | New Project to open the Project wizard and create a new project for your Palm application. On Step 2 of the Project wizard, click the ellipsis (...) button by the JDK field and select the J2ME JDK. Make any other desired changes in the wizard, then click OK to close the wizard.
- 5 Set the project properties to use the PalmOS\_Device: Choose Project | Project Properties, and click the MIDlet tab on the Run page. Select PalmOS\_Device from the Emulator Device drop-down list, then click OK to close the dialog box. For more information on creating projects, see [Chapter 3, “Creating and managing MIDP projects.”](#)

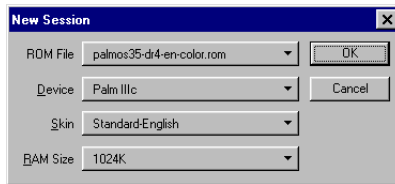
- 6 Create the MIDlet: Choose File | New and click the Micro tab. Double-click the MIDlet icon to run the MIDP MIDlet wizard and create the MIDlet and Displayable files. For more information on creating MIDlets, see [Chapter 5, "Creating a MIDP user interface."](#)
- 7 Code your application, then compile and debug it. Fix any errors, then rebuild and save. See [Chapter 4, "Building MIDP applications,"](#) for more information on compiling and running.
- 8 Click the Run button on the JBuilder toolbar to run the application in the Palm OS® Emulator.

The first time you run a J2ME application using the Palm emulator, you will be prompted for the location of the emulator.



This information is kept at <J2mewtk\_home>\wtllib\devices\PalmOS\_Device under the J2ME Wireless Toolkit installation directory. It uses the Converter.jar file to convert a JAD and JAR files to a PRC file.

- 9 Specify the ROM, Palm Device, Skin, and Ram Size to use for the session: Once the emulator is running, right-click on it and choose New to open the New Session dialog box where you can select these options.



For additional information on using the Palm OS® Emulator, see <Palm>\Docs\UserGuide.pdf delivered with the emulator. Also check the Palm OS® Emulator site at <http://www.palmos.com/dev/tech/tools/emulator/> for updated emulators, ROMs, skins, and instructions.

## Creating and managing MIDP projects

JBuilder development is centered around the concept of a project file. The project file contains a list of files in the project and the project properties, which include a project template, default paths, class libraries, and connection configurations. JBuilder uses this information when you load, save, build, or run a project. Project files are modified whenever you use the JBuilder development environment to add or remove files or set or change project properties. You can see the project file as a node in the project pane.

In creating MIDP applications, you must first create or open a project file that uses a J2ME MIDP/CLDC JDK before you can access the MIDP wizards, such as the MIDP MIDlet wizard and the MIDP Displayable wizard.

To start a new project, use JBuilder's Project wizard to generate the basic framework of files, directories, paths, and preferences automatically. The Project wizard can create a project notes file that can be inserted as a comment block in the files created from within the project by other JBuilder wizards, and consequently included in JavaDoc-generated documentation. These comments can be modified on the General page of the Project Properties.

## Creating a new project with the Project Wizard

Choose File | New Project on the JBuilder menu to open the Project wizard.

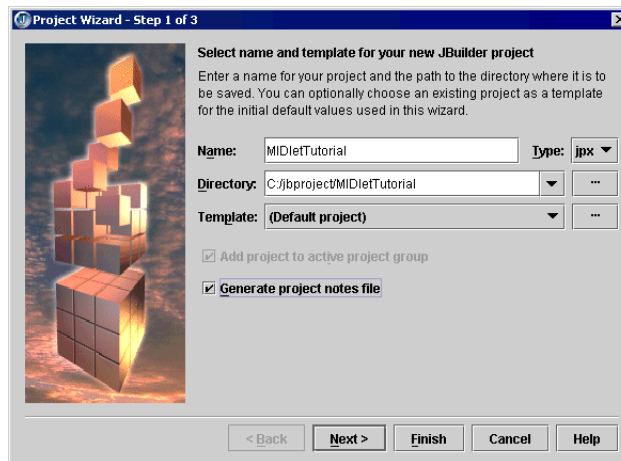
**Important** You must already have a J2ME MIDP/CLDC JDK installed and configured in JBuilder before you can create a MIDP project. See [“Setting up a JDK” on page 2-2](#).

### Project wizard: Step 1

Step 1 sets your project name, type, and template, root directory, source, backup, and output directories.

**Note** Only advanced users should change these suggested paths.

**Figure 3.1** Project wizard, Step 1

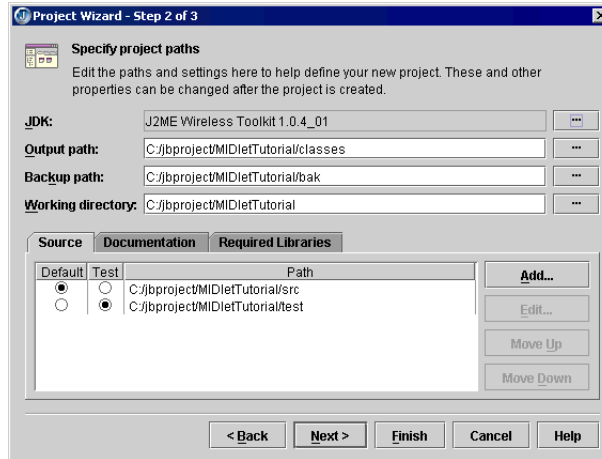


- 1 Enter a name for your project if you don't want to use the default name. (JBuilder uses the project name as the default package name.)
- 2 Specify the directory where the project will reside. You may use an existing directory or create a new one by entering it here. If you enter a path that is syntactically flawed, you can not proceed.
- 3 Use the Template field to select a project template if you don't want to use the default one. To choose a project you've opened previously as a template, click the down arrow and choose from the list. If you want to use another project, click the ellipsis (...) button to open the Open Project dialog box.
- 4 Check Generate Project Notes File to create an .html file to use for keeping notes about the project.
- 5 Click Next to go to Step 2.

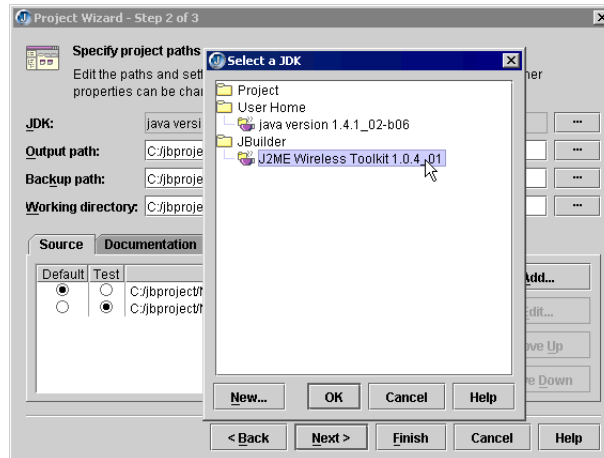
## Project wizard: Step 2

Step 2 lets you check your paths, set the JDK version you'll compile with, and select the required class libraries. You can also decide if you want a project notes file, which will provide the information for your application's Help | About dialog box. If you don't want to create a project notes file, you can fill in the other fields, uncheck this box, and click Finish in Step 2.

**Figure 3.2** Project wizard, Step 2



- 1 Check your project, source, backup, and output paths. Note that the paths set in Step 1 are used here.
- 2 Click the ellipsis (...) button for the JDK field and choose the version of the JDK that you want to use. See [“Setting up a JDK” on page 2-2](#).



- 3 Click Next to go to Step 3.

### Project wizard: Step 3

Step 3 develops the information for your application's Help | About dialog box. This information can also be inserted as a comment-block to new wizard-generated files created for your project. You can change this information on the General page of the Project Properties (Project | Project Properties).

**Figure 3.3** Project wizard, Step 3

**Project Wizard - Step 3 of 3**

**Specify general project settings**  
Enter settings here to help define your new project. These and other properties can be changed after the project is created.

Encoding: **Default**

**Automatic source packages**

☒ Enable source package discovery and compilation

Deepest package exposed: **3**

**Class Javadoc fields:**

Label	Text
Title:	
Description:	
Copyright:	Copyright (c) 2003
Company:	
@author	
@version	1.0

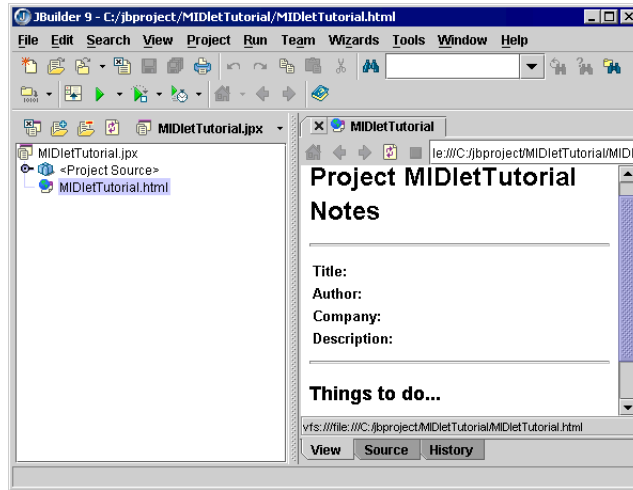
☐ Include references from project library class files

☐ Diagram references from generated source

< Back   Next >   **Finish**   Cancel   Help

Fill in the title of the project, author, and company, and add your product description. Click Finish.

Your new project file appears at the top of the project pane with the HTML project notes file below it. Double-click the HTML file to see the project notes in the content pane.

**Figure 3.4** Project pane after running the Project wizard

For detailed information on working with projects, project files, and packages in JBuilder, see “Creating and managing projects” in *Introducing JBuilder*.

## Adding the MIDlet files to the project

---

The next step is to add the MIDlet files to the project using the MIDP MIDlet wizard. To create a MIDlet,

- 1 Choose File | New, and select the Micro tab in the object gallery.
- 2 Double-click the MIDlet icon to run the MIDP MIDlet wizard.
- 3 Make the desired changes to the steps of the wizard, then press Finish.

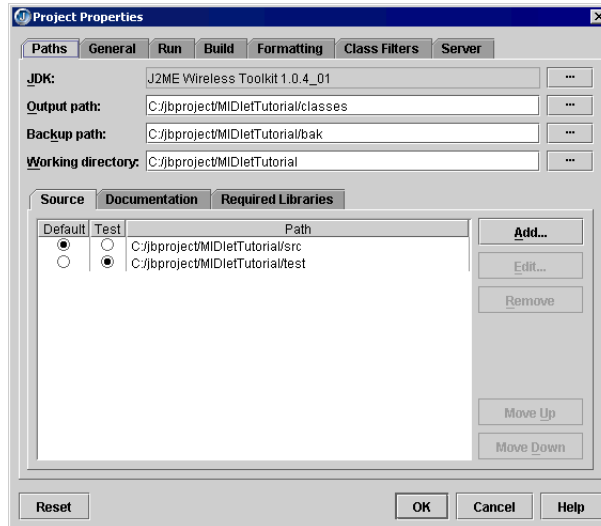
For more information on creating MIDlets, see [“Creating the MIDlet and Displayable classes” on page 5-7](#).

## Setting MIDP project properties

---

JBuilder provides a Project Properties dialog box for controlling various settings specific to a project. Mobile development adds additional tabs to the Project Properties dialog box for MIDP projects. These are covered below.

For details on setting general project properties, see Setting project properties under “Creating and managing projects” in *Introducing JBuilder*.

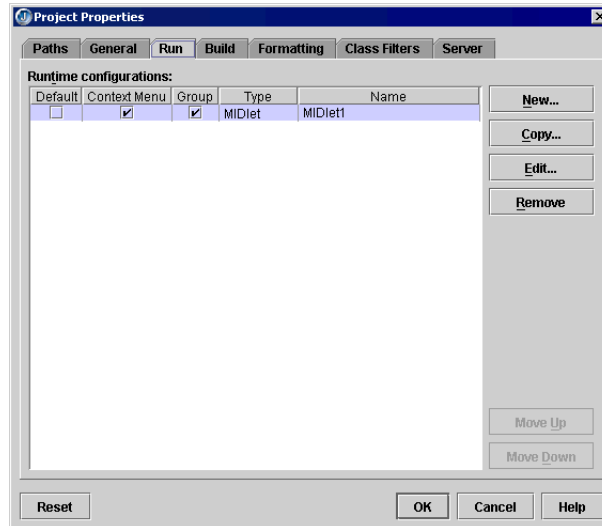
**Figure 3.5** Project Properties dialog box

You can also specify default settings for all new projects. Default project settings are stored in an actual project called `Default.jpr` found in the `.jbuilder` subdirectory of your home directory. The `Default.jpr` file is used as a template whenever a new project is created. To change the default project properties, choose `Project | Default Project Properties` from the main menu.

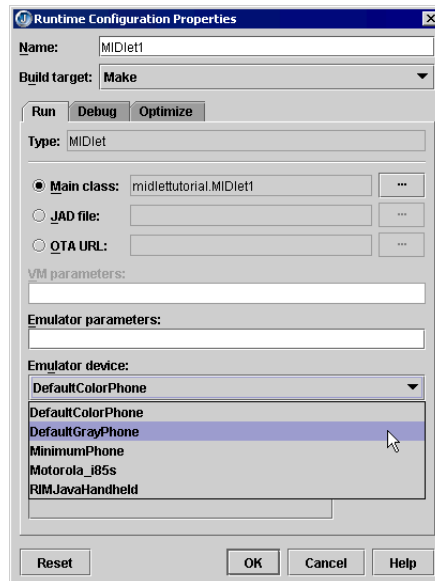
## Runtime configurations

The Project Properties dialog box contains a `Run` page for creating MIDlet runtime configurations. One runtime configuration can be created automatically as part of the MIDP MIDlet wizard, or you can manually create one or more here. To access this page, choose `Project | Project Properties`, or `Run | Configurations` from the JBuilder menu.



**Figure 3.6** Run tab of the Project Properties dialog box

To create or modify a runtime configuration, click New, or select an existing configuration and click Edit. This opens the Runtime Configuration Properties dialog box.

**Figure 3.7** Run tab for MIDlet type in the Runtime Configuration Properties dialog box

## Main class

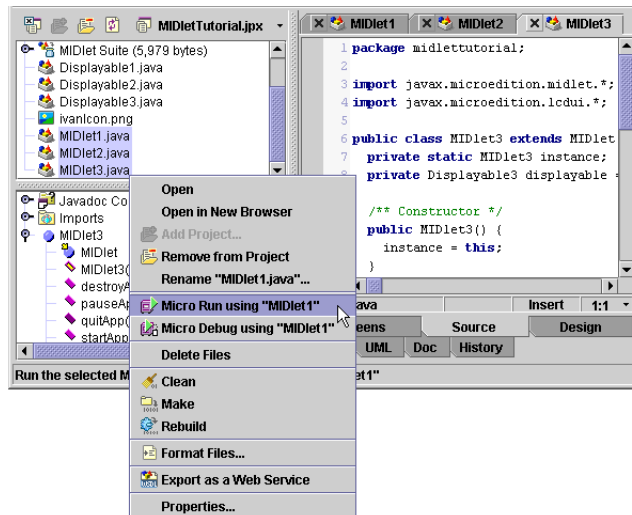
Use this option to specify one MIDlet as the default MIDlet to run. Click the ellipsis (...) to browse to the MIDlet that is the main class for this project. The first MIDlet created in the project is specified by default.

## JAD file

Use this option to specify the JAD file to use at runtime. The JAD file names all the MIDlets to be run. After adding a JAD file to your project, or having one generated during an archiving process, you must re-build your project before you can select this option.

**Note** You can run multiple MIDlets without having to archive first. Select multiple MIDlets in the project pane by holding down the *Shift* key as you them. Then right-click and choose Micro-Run.

**Figure 3.8** Running multiple MIDlets from the project pane



## OTA URL

Use the OTA (Over The Air) URL to run a JAD file stored on a local storage or remote server. Click the ellipsis (...) button to specify the Storage Name or the URL where the JAD file is located.

**Note** This option is only available if the JDK you are using supports OTA Provisioning.

## VM parameters

The VM Parameters field allows you to specify any parameters you want to pass to the Java Virtual Machine (VM) compiler at runtime.

**Note** The VM Parameters field is enabled when you use an emulator that is launched by `java.exe`.

## Emulator parameters

The Emulator Parameters field allows you to specify any parameters you want to pass to the emulator at runtime.

## Emulator device

Specify the device you want the emulator to use for the project at runtime.

## MIDlet parameters

MIDlet parameters are user defined. Specifying them in the Project Properties makes them the default runtime parameters for all MIDlets in the project. If you enter them in the Default Project Properties, they will be the runtime default for all MIDlets created in all projects.

To enter a new parameter, click Add, then enter a parameter name in the first column and a parameter value in the second column. To remove an existing parameter, select either the parameter name or value and click Remove.

## Build target

For each runtime configuration you create, you can specify which Build Target to execute prior to running in the Runtime Configuration Properties dialog box. JBuilder provides a set of standard targets from which to choose, and additional ones are added to the list depending on your project. For more information, see “Build Targets” in *Building Applications with JBuilder*.

## See also

- “Running Java programs” in *Building Applications with JBuilder*
- “Setting runtime configurations” in *Building Applications with JBuilder*
- [“Setting up a JDK” on page 2-2](#)





# Building MIDP applications

## Compiling

---

Compiling MIDP classes in JBuilder is the same as for normal Java development, with one exception. In J2ME development, the classes must be preverified before you run them. JBuilder automatically preverifies all the classes in your project at compile time.

Use Make  or Rebuild  to compile your project. These are available in three places:

- The Project menu
- The toolbar (Rebuild is accessed on the Make button's drop-down menu.)
- Right-click menu on a selected file in the project pane

## Types of compiling

---

The commands for compiling from within the IDE include:

- The Make command (Project | Make Project) compiles the selected node and any imported files if they have changed. During routine iterations of the edit/recompile cycle, Make is the recommended command.
- The Rebuild command (Project | Rebuild Project) compiles all the project source files and any imported files, regardless of whether they have changed, so it takes longer than Make. Rebuild ensures that the project options are applied to all classes that are referenced by the selected nodes.
- The Run command (Run | Run Project) runs your application. If a Build Target is selected in the runtime configuration, it compiles and does other specified tasks before and/or after running it. See “Setting

runtime configurations” under “Running Java programs” in *Building Applications with JBuilder*.

## What can be compiled

---

The following parts of a program can be compiled:

- The entire project
- Packages (packages lower in the project tree hierarchy are built before packages higher in the hierarchy)
- Java files

To understand how JBuilder locates files to compile the program, see “How JBuilder constructs paths” and “Where are my files?” in *Building Applications with JBuilder*.

For more information on compiling in JBuilder, see “Building and compiling Java programs” in *Building Applications with JBuilder*.

## Running

---

For a device to be MIDP compliant, it must support a Java VM that utilizes both a Java Application Descriptor (JAD) file and a JAR file. The JAD file names the MIDlets in the JAR file and specifies required and optional information about the MIDlets. It also specifies the size of the JAR file, and can contain parameters to pass to the MIDlets at runtime. The JAD file is the only means for specifying the names of multiple MIDlets in a MIDlet Suite. For more information on the JAD file see [“Creating the JAD file outside of JBuilder” on page 7-5](#).

When you are testing MIDlets in JBuilder, you can run each MIDlet in a project separately, select multiple MIDlets and run them all at once, or run from the JAD file to test multiple MIDlets in a MIDlet Suite at one time.

For information on creating JAR and JAD files, see [Chapter 7, “Archiving MIDlets.”](#)

## Running MIDlets

---

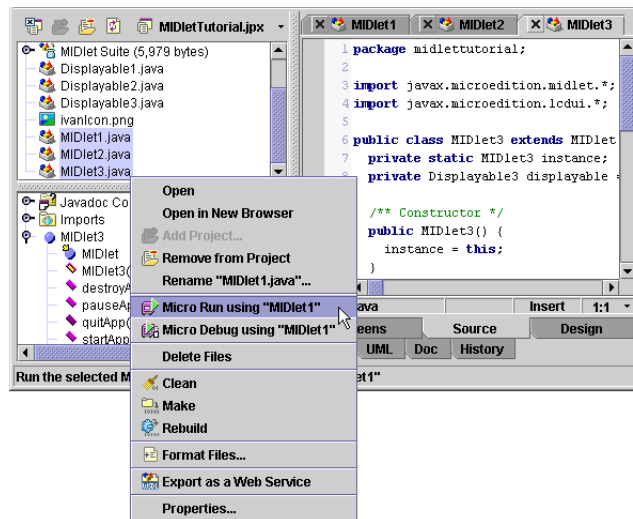
### Running from the MIDlet file(s)



To run your application inside JBuilder, click Run (*F9*) on the toolbar, or right-click the MIDlet file and choose Micro Run. JBuilder launches the emulator device specified for the JDK using the default runtime configuration of the project.

If your project has multiple MIDlets, you can select them all and run them at the same time. Hold down the *Ctrl* key while selecting the MIDlets in the project pane, then right-click and choose Micro Run.

**Figure 4.1** Running multiple MIDlets from the project pane



**Note** You can set up multiple runtime configurations in JBuilder, each with different targets. You can also set a default runtime configuration, or have no default runtime configurations. See “Setting runtime configurations” in *Building Applications with JBuilder* for more information on this topic.

## Running from a JAD file

After you have archived your application and created a JAD (Java Application Descriptor) file, you can run the MIDlet Suite from the JAD file and test all the MIDlets in the archive at once.

There are two ways to run a MIDlet Suite from the JAD file:

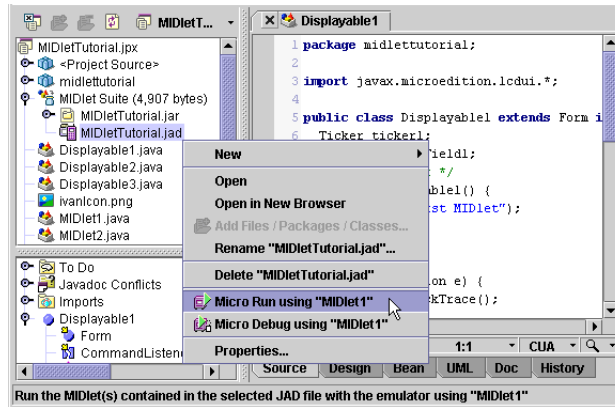
- Right-click the JAD file and choose Micro-Run.

To do this,

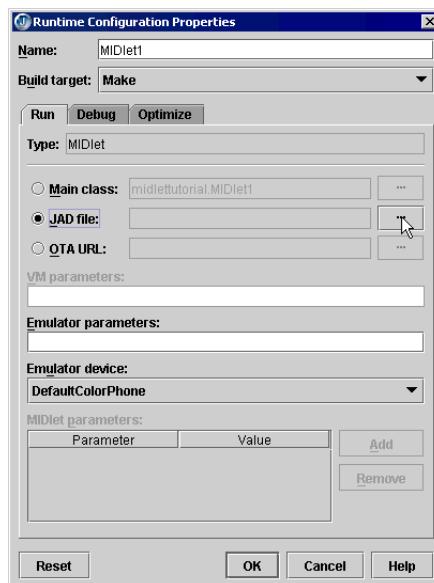


- Click the Add Files/Packages button at the top of the project pane, navigate to the JAD file and select it to add it to your project. (This is only necessary if you archived your MIDlet manually. When you use the JBuilder Archive Builder, the JAD file is automatically added to your project.)

- b** Right-click the JAD file in the project pane and choose Micro Run.

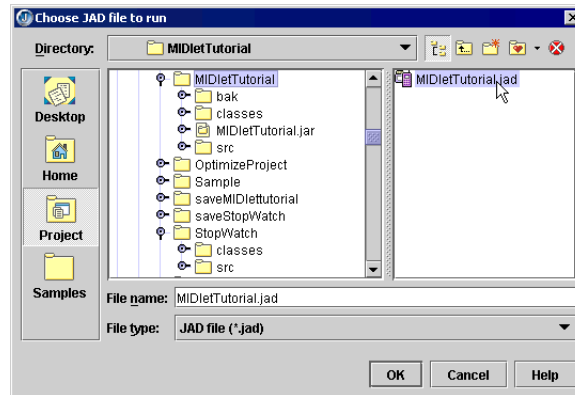


- Select the JAD file as the default file to run.
  - a** Choose Project | Properties and click the Run tab.
  - b** Click New to open the Runtime Configuration Properties dialog box.
  - c** Type a unique name in the Name field for this runtime configuration if you wish.
  - d** Choose MIDlet from the Type drop-down list.
  - e** Click the JAD radio button, then click the ellipsis button beside the JAD field.





- f Choose the JAD file you want as the default runnable file, then click OK.



- g Click OK again to close the Project Properties dialog box.

## Selecting which device to use

If the JDK supports multiple devices, you can specify which one to use in the Project Properties dialog box. To do this,

- Choose Project | Project Properties
- Select the runtime configuration you're going to use on the Run page and click New to create a new configuration, or Edit to edit an existing one.
- Choose a MIDlet type on the Run page of the Runtime Configuration Properties dialog box, then select an emulator device from the drop-down list.

**Note** You can also specify the default device to use by the emulator for all projects in the Default Project Properties dialog box, accessed from Project | Default Project Properties.

**Important** Only devices that use exactly the same libraries as the target device specified on the Micro tab in the Configure JDKs dialog box (Tools | Configure JDKs | Micro tab) will be shown in the Target Device list. If the desired device is not listed, then you need to go to Tools | Configure JDKs and select the target device.

# Debugging

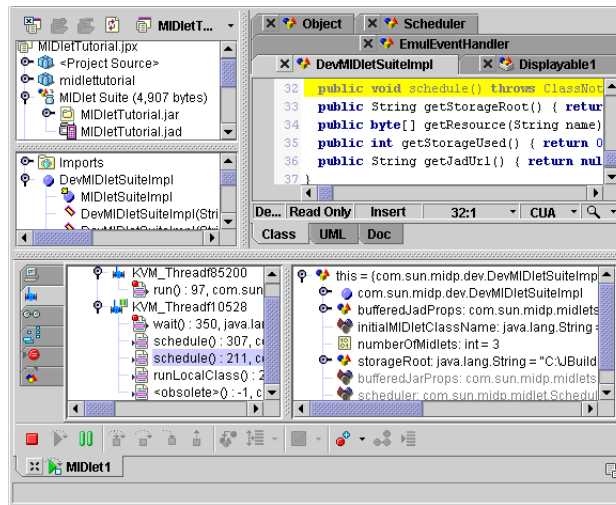
## MIDlet debugging in JBuilder



To debug your application inside JBuilder, click the Debug (*Shift+F9*) button on the toolbar, or right-click the MIDlet and choose Micro Debug.

When you choose Debug or Micro Debug, the debugger appears in the message pane at the bottom of the AppBrowser and the MIDlet runs in the emulator in a separate window.

**Figure 4.2** JBuilder debugging a MIDlet



Debugging MIDP classes is exactly the same as J2SE classes, with some exceptions which are explained below. For more information on using the debugger, see “Debugging Java programs” in *Building Applications with JBuilder*.

## Debugger limitations

Due to the limitations of the KVM Debug Wire Protocol (KDWP),

- Exception, class, method, and cross-process breakpoints are not supported and will be ignored.
- The Tracing Disabled feature in JBuilder is not supported and will be ignored.

# Creating a MIDP user interface

## Overview

---

The J2ME MIDP UI components were created specifically for limited configuration devices because the standard AWT components were unsuitable in a number of ways. The AWT components used too much memory and did not fit with the requirements of user interaction on hand-held devices. Also, MIDP devices do not use a pointing mechanism which AWT requires.

In order for a device to be MIDP compliant, it must base its implementation on the J2ME MIDP specification. The MIDP UI APIs were written at a high level to be more abstract and provide portability across platforms. At the same time however it limits control over the look and feel of the UI. Each manufacturer of a device can extend the MIDP API to take advantage of specific features on their device. But, the more you supplement MIDP with additional UI classes, the less cross-platform it becomes.

## MIDP Screens

---

Due to the small display screens on hand-held devices, and limited memory and processing power, MIDP uses a special class called `Displayable` to display a UI on the device instead of the typical windowing system.

A `Displayable` is an object that draws graphics on the device's display screen. It is similar in behavior to a window except that only one `Displayable` can be visible at a time. In a MIDP application, multiple `Displayable` components are stacked on top of each other like cards in a

deck. The user steps through the set of `Displayable` components one at a time to complete a task. The movement through the stack can either be linear or non-linear, but it should be clear to the user what to do to move to the next `Displayable`.

The `Displayable` class has two subclasses, `Screen` and `Canvas`. There are four types of components that descend from `Screen`: `Form`, `List`, `TextBox`, and `Alert`. You treat the `Canvas` and all the `Screen` classes the same in terms of swapping one another out.


The UI design for `Displayable` screens should be as simple as possible. Ideally, you should limit each screen to one task and use navigation to move between the screens.

When you use the JBuilder MIDP MIDlet wizard or MIDP `Displayable` wizard, you can choose which type of screen or canvas to use. The wizard generates a `Displayable` class that extends the screen or canvas you choose.




JBuilder also delivers screen components on the MIDP Screens page of the component palette. While screens other than `Form` are not designable, you can instantiate them in your application from the palette and set some properties for them in the Inspector. The one exception to this is the `Canvas`. It is not delivered on the palette because to use a `Canvas`, you must subclass it and provide an implementation of the abstract `paint()` method in the subclass which the UI designer can't do. The fastest way to create a `Canvas` is to use the MIDP `Displayable` wizard and extend `Canvas`. For more information, see [“Creating a Canvas” on page 5-13](#).

Below is a table describing each of the `Displayable` screens delivered on the MIDP Screens tab of the component palette. In front of each component name is the icon used by the UI designer to represent it in the component tree and on the palette:

**Table 5.1** Table of MIDP screens

Component	Description
 <code>Alert</code>	<p>An <code>Alert</code> displays text and an image to the user and waits for a specified length of time before closing, or until the user dismisses the screen. It is primarily used to report error messages and other exceptional conditions. You have limited control over the layout of this screen. An <code>Alert</code> may use an <code>AlertType</code> to indicate the nature of the alert and to play an appropriate sound when it is displayed.</p> <p>Although an <code>Alert</code> is a screen, it is available on the component palette to make some of its properties visible in the Inspector. The properties exposed are <code>image</code>, <code>string</code>, <code>ticker</code>, <code>timeout</code>, <code>title</code>, and <code>type</code> (alert type).</p>

**Table 5.1** Table of MIDP screens (continued)

Component	Description
 Form	<p>A <code>Form</code> is the only MIDP component that can contain other components. It is the screen most used in creating a user interface. While you can theoretically put as many components in a <code>Form</code> as you want, due to the small display area of the hand held devices you should limit each <code>Form</code> to one user task. Also, you should limit a <code>Form</code> to a length that would require no more scrolling than three or four display screens on the target device. This will maximize the performance of the device and to make it easier for the user navigate.</p> <p>The components that can be placed in a <code>Form</code> are described in <a href="#">“MIDP UI components” on page 5-3</a>. In MIDP UI programming, the developer does not have control over the component layout in a <code>Form</code>. The only thing you can control about placement is the order of multiple components in the <code>Form</code>, much like <code>VerticalLayout</code> in J2SE. When you use the JBuilder UI designer to design a UI in a <code>Form</code>, you can use the mouse to move the components up or down.</p>
 List	<p>A <code>List</code> is used to display one of three types of list: <code>IMPLICIT</code>, which can be used as a menu, <code>EXCLUSIVE</code>, a single-choice list, and <code>MULTIPLE</code> which allows more than one selection.</p>
 TextBox	<p>A <code>TextBox</code> is a screen in which the user can enter and edit text. You can set its maximum size (or capacity) and specify a default <code>String</code> value. If the text contained within a <code>TextBox</code> is more than can be displayed at one time, the user can scroll to view or edit any part of the text. The <code>TextBox</code> uses constraints which lets the application restrict the user’s input such as for e-mail addresses, numeric values, phone numbers, URLs, and passwords.</p>

See the `javax.microedition.lcdui` MIDP API reference documentation to learn more about these components. For information on viewing reference documentation, see [“Viewing class reference documentation for MIDP classes” on page 5-20](#).

## MIDP UI components

MIDP UI classes are not JavaBeans. JavaBeans are self-contained classes that interact together and with the Java environment in a defined manner. JavaBeans are also intended for use in a visual application development tool. For example, JavaBeans have a default constructor (a constructor that doesn’t take any arguments). MIDP UI components do not have a default constructor, and are not intended to be visually designable.

To accommodate the J2ME MIDP UI components in the UI designer, JBuilder Mobile development includes two component tabs in the UI designer. This designer makes it possible to work visually with the basic MIDP UI components that can be dropped into a `Form`. You can also create a `Ticker` and set its text in the UI designer. These UI components come

already installed on the palette, and when you drop them into the designer, what you see are “generic” renderings of J2ME MIDP components.

If you develop or use additional UI classes based on the J2ME MIDP classes,








- Any additional properties beyond the J2ME specified properties will not appear in the Inspector.
- Any visual customizations will not be displayed by the designer.

If you add a component to the design that is a subclass of a J2ME MIDP component, you will still see only the “generic” renderings of the MIDP component. However, the source code generated will be for your component, and any property changes should be valid.

The UI designer does not support components beyond those defined by the J2ME MIDP specification. New components must be a subclass of any MIDP UI class for the designer to recognize it.

Below is a table listing the MIDP UI components delivered on the palette in JBuilder. In front of each component name is the icon used by the UI designer to represent it in the component tree and on the palette:

**Table 5.2** Table of MIDP UI components

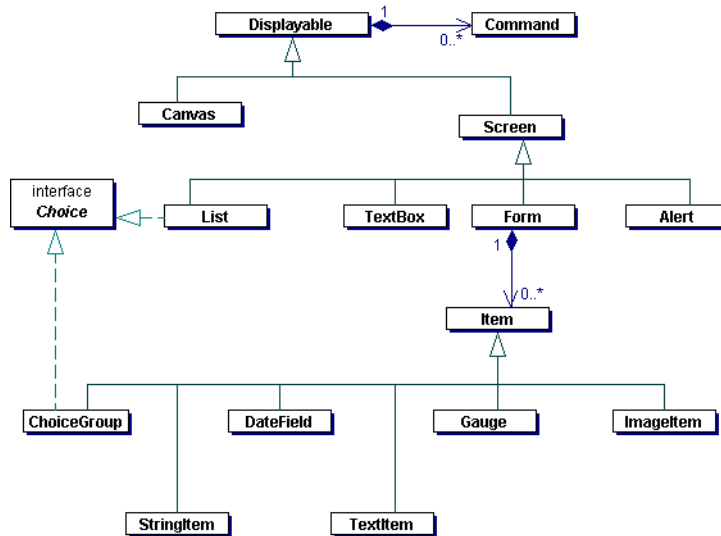
Component	Description
 ChoiceGroup	A list of one or more <code>String</code> choices from which the user can select. Each <code>String</code> can have an optional image. Radio buttons usually represent single choices, while checkboxes represent multiple choices.
 DateField	An editable date and time field. You can display just the date or time, or you can display both.
 Gauge	A bar graph used to display a range of values which the user or processes can set.
 ImageItem	A container for an image in the PNG format that can include a label, and can specify a limited amount of layout information to control the image placement on the screen.
 StringItem	A non-editable text field that can have a label. The label and content can be modified by the application.
 TextField	An editable text field that can have a label, and an initial text value. You can set a maximum number of characters to accept. A <code>TextField</code> includes constraints for special fields such as e-mail addresses, numeric values, phone numbers, URLs, and passwords. Multiple <code>TextField</code> components can be added to a Form for user input.
 Ticker	A <code>String</code> that moves continuously across the screen. The scrolling speed and direction are controlled by the device, and the application can pause it to save energy when the user has been inactive for a specified length of time. For information on creating a Ticker in the UI designer, see <a href="#">“Creating a Ticker” on page 5-14</a> .

See the `javax.microedition.lcdui` MIDP API reference documentation to learn more about these components. For information on viewing reference documentation, see [“Viewing class reference documentation for MIDP classes”](#) on page 5-20.

## MIDP UI class hierarchy

The diagram below illustrates the hierarchy of the major MIDP UI classes:

**Figure 5.1** MIDP UI classes



## Creating a UI

To create a MIDlet UI with the JBuilder UI designer, you must first create a project in JBuilder, then add a MIDlet that contains a `MIDlet` class and at least one `Displayable` class.

Once you have created a `Displayable` class for your MIDlet, you can use the UI designer to visually customize the user interface for your `Displayable`. (This is analogous to designing a container object in the regular UI designer.)

The fastest way to do this is to use JBuilder’s Project wizard, followed by the MIDP MIDlet wizard. The MIDP MIDlet wizard creates both the `MIDlet` class and a designable `Displayable` class. You can add additional `Displayable` classes to your project using the MIDP `Displayable` wizard.

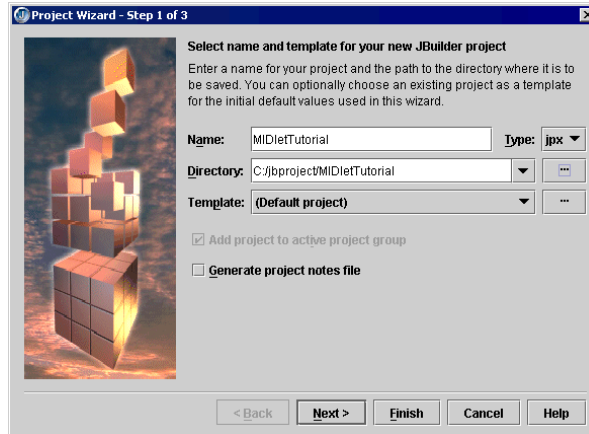
### See also

- [Chapter 10, “Tutorial: Creating and testing MIDlets”](#)

## Creating a project

To create a project using the JBuilder Project wizard, choose File | New Project and fill in the information on the steps of the wizard.

**Figure 5.2** Project wizard, Step 1

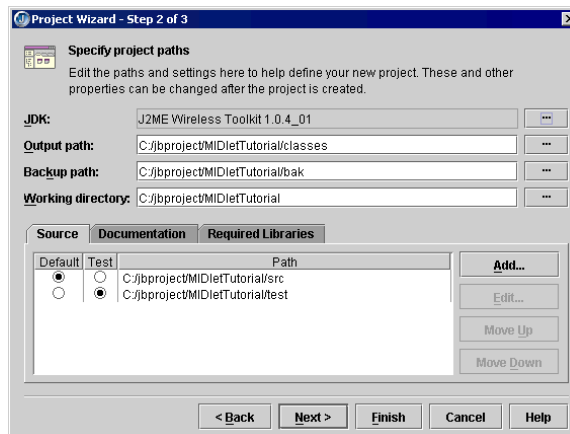


For details on using the Project wizard see [“Creating a new project with the Project Wizard” on page 3-2](#), or click the Help button at the bottom of the wizard.

**Important** Before you create your first project in JBuilder, be sure to configure your MIDP/CLDC JDK for use within JBuilder. See [“Setting up a JDK” on page 2-2](#).

In Step 2 of the Project wizard, be sure to select your MIDP/CLDC JDK if it doesn’t display by default. Click the ellipsis (...) button and select a MIDP/CLDC JDK from the list of preconfigured JDKs.

**Figure 5.3** Project wizard, Step 2





## Creating the MIDlet and Displayable classes

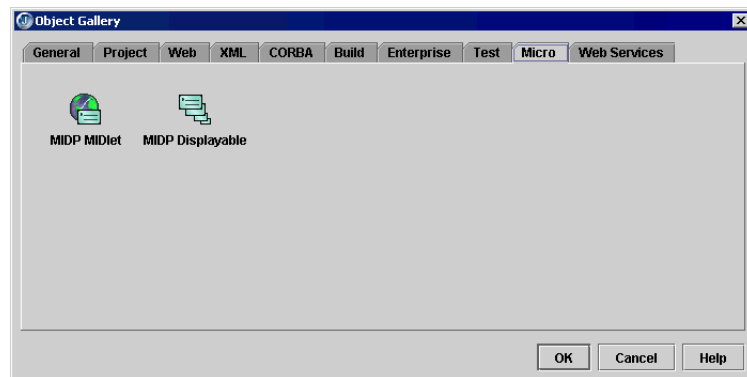
Once you have an opened project in the AppBrowser, you can add a `MIDlet` class and a `Displayable` class. Both are generated automatically with the JBuilder MIDP MIDlet wizard.

### Important

- 1 Choose **File | New**, or click the New (*Ctrl+N*)  icon on the JBuilder toolbar to open the object gallery. Select the **Micro** tab.

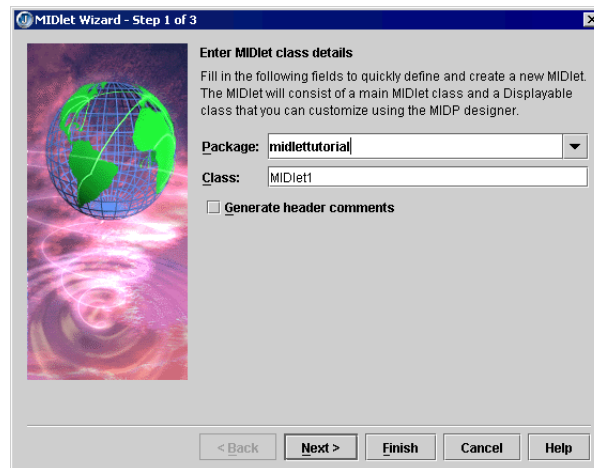
If your current JDK is not a J2ME MIDP/CLDC JDK, and the `MIDlet` class cannot be found in your project's classpath, the icons will be disabled in the object gallery.

**Figure 5.4** Object Gallery Micro page



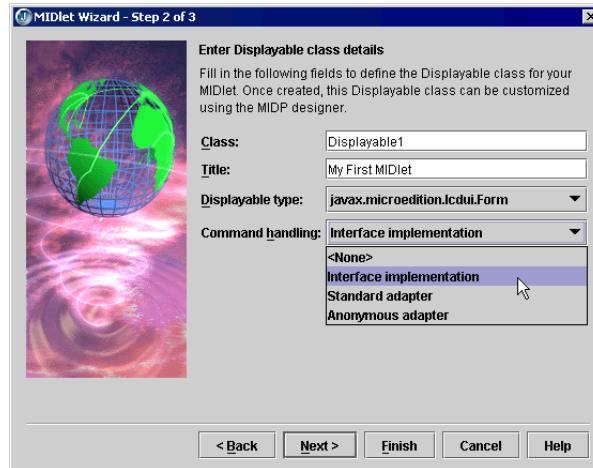
- 2 Click the `MIDlet` icon to open the MIDP MIDlet wizard. Help is available from the Help button on the wizard.

**Figure 5.5** MIDP MIDlet wizard, Step 1



- 3 Type a name for your MIDlet class in the Class field if you don't want to use the default name.
- 4 Check Generate Header Comments if you want the wizard to generate information from the project file as header comments at the top of the MIDlet class file. This is the information you entered in the Project wizard.
- 5 Click Next to go to Step 2.

**Figure 5.6** MIDP MIDlet wizard, Step 2



- 6 Change the Displayable class name and Title in Step 2 if you don't want to use the defaults.
- 7 Choose the type of Displayable to extend.

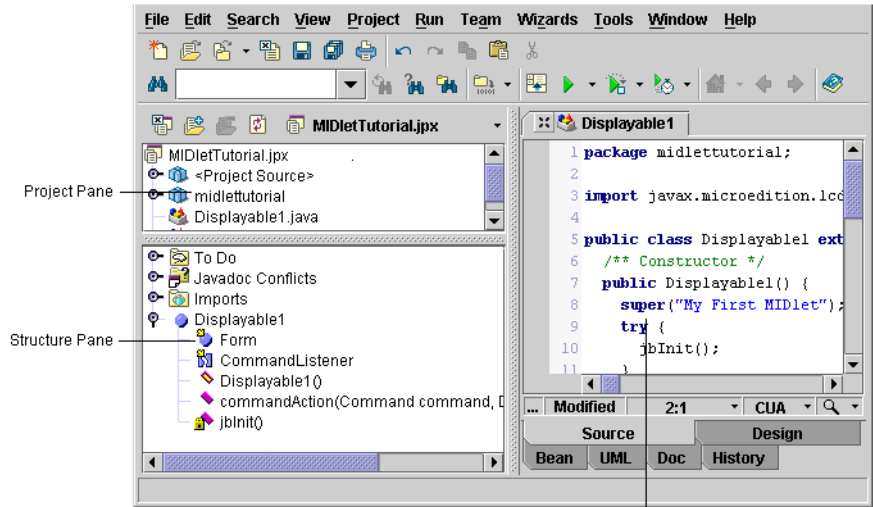
**Note** A Form (`javax.microedition.lcdui.Form`) is the only Screen type you can add components to and visually design in the UI designer.

- 8 Select the type of command handling you want in the automatic code generation. Besides the standard adapter and anonymous adapter choices available in the Project Properties | Code Style page, you have two additional options:
  - Interface Implementation, which will generate code to implement the `commandListener` interface within the class. The size of class generated in this way will be smaller than those with adapters.
  - None, which is used when you want to set the `commandListener` through other classes, such as a MIDlet class.
- 9 Click Finish to close the MIDP MIDlet wizard.

As a result of using the Project and MIDP MIDlet wizards, JBuilder creates the project and MIDlet files in the AppBrowser and opens the displayable

file (`Displayable1.java`) in the content pane so it can be edited and designed.

**Figure 5.7** AppBrowser after running the Project and MIDP MIDlet wizards



## Using the UI designer

The UI designer is similar to the standard JBuilder UI designer, but is used specifically for designing a `Displayable` class that extends `Form`. It is not used to visually design a MIDlet class.

A `Displayable` class contains the UI for the display screen on the device, similar to the `Frame` or `Panel` container in a normal Java application.

You can use the JBuilder MIDP MIDlet wizard or the MIDP `Displayable` wizard to create a `Displayable` class for your MIDlet. Once you have created a `Displayable`, you can visually customize the user interface for it in the UI designer. The `Displayable` is represented in the component tree by a node called `this`.

If you need additional `Displayable` classes for your MIDlet, you can add them to your project using the MIDP `Displayable` wizard.

## Requirements for a MIDP class to be designable

Before you can add components to a `Displayable` class in the UI designer, it must meet the following requirements:

- It must be a `Displayable` class which extends `Form`.
- It must have a `jInit()` method which contains all the handling for existing UI elements which have been declared at the class level. If you

have not added any UI elements to the `Displayable` class yet, the UI designer will automatically add the `jbInit()` method when you add the first UI component in the designer.

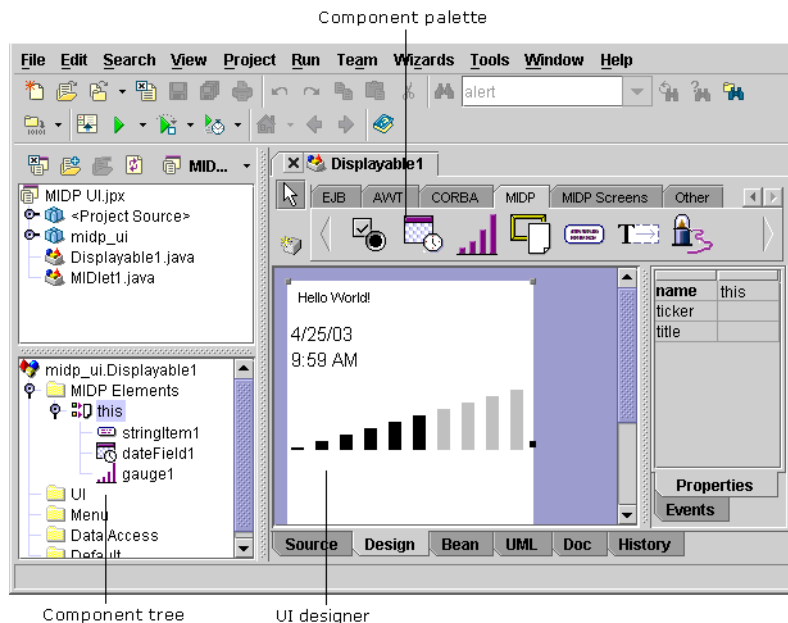
**Note** If you want to open an existing `Displayable` class in the designer that already has UI elements in it and no `jbInit()` method, you first need to create a `jbInit()` method then move all the UI elements into it.

If you create the `Displayable` class with the MIDP MIDlet wizard or MIDP `Displayable` wizard, the `jbInit()` method is created for you, and as you add UI elements to the `Form`, the designer puts the code for them into the `jbInit()` method.

## Opening the UI designer

To open the UI designer, click on the Design tab in the content pane at the bottom of an opened `Displayable` file.

**Figure 5.8** UI designer

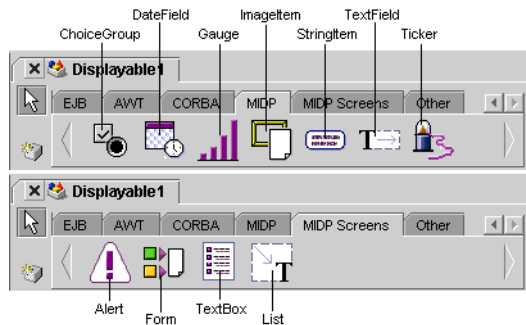


Designing a J2ME MIDP UI is just like designing a J2SE UI with the following exceptions:

- There is a folder in the structure pane called MIDP Elements under which are grouped all the UI elements, or components you add to the `Form`.
- There is a `Displayable` instead of a `Swing` container in the design pane.
- You can only design MIDP components.

- The initial design surface size is set to the same aspect ratio of the device display. You can resize the design surface if desired, but any change to the design surface size does not result in a corresponding change to the actual device display. It is up to the device vendors to decide how to manage UI component display for their particular devices.
- The MIDP components are delivered on the component palette under two tabs: MIDP, which contains the UI elements, and MIDP Screens, which contains the screen classes. These tabs are at the end of the row. To see them, click the right arrow to scroll them into view.

**Figure 5.9** MIDP components on the palette



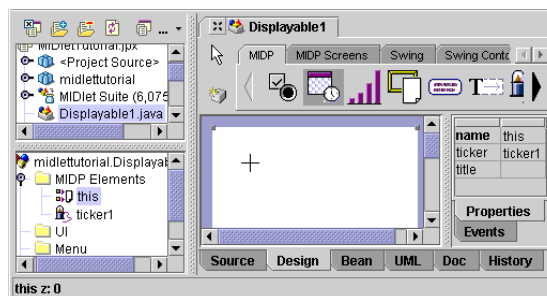
**Note** You can customize the order of the tabs on the component palette and move the MIDP tabs to the beginning. Choose Tools | Configure Palette..., then click the name of a tab and choose Move Up or Move Down.

## Adding components to the design

To add components from the palette to the `Form` in the designer, use the steps below:

- 1 Select the appropriate tab on the palette, then click the desired component. The mouse cursor changes from a pointer to a plus sign, which indicates that the component has been copied to the clipboard.
- 2 Click anywhere in the `Form` or the component tree to add the component.

**Figure 5.10** Adding a component in the designer

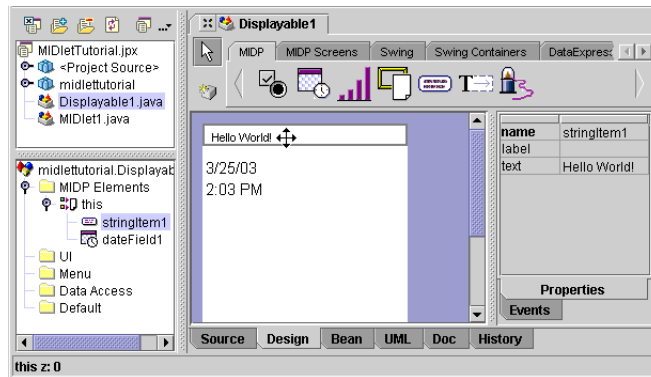


## Moving components

Drag and drop functions are more limited in the UI designer than in the UI designer. There are no layout managers in MIDP UI design since placement of the UI elements on a MIDP display is controlled by the device. You can, though, modify the order the elements appear in a `Form` on the screen.

When there are multiple components on the `Form`, you can move an upper one to the bottom. Click on it in the design and drag downward. The component will move to the lowest available position on the `Form`.

**Figure 5.11** Dragging a component in the designer



**Note** To move a component from one container to the other in the component tree, right-click on it and choose Cut or Copy, then select the new container, right-click and choose Paste.

## Deleting components

To delete a component, select it in the component tree or on the design surface and do either of the following:

- Press the *Delete* key on the keyboard.
- Right-click it in the component tree and choose Cut.

## Copying components

To make another instance of a component, select it in the component tree and do the following:

- Right-click it in the component tree and choose Copy.
- Select the `Form` (`this`) in the component tree then right-click and choose Paste.

## Undoing and redoing an action

---

You can undo or redo an action in the designer by doing one of the following:

- Right click anywhere in the component tree and choose Undo (*Ctrl+Z*) or Redo (*Shift+Ctrl+Z*)
- Click anywhere in the designer and choose Edit | Undo or Edit | Redo.

You can undo multiple successive actions by choosing Undo repeatedly. This undoes your changes by stepping back through your actions and reverting your design through its previous states.

Redo reverses the effects of your last Undo. You can redo multiple successive actions by choosing Redo repeatedly. Redo is available only after an Undo command.


## Setting properties in the inspector

---

The UI designer surfaces properties of the UI classes in the Inspector. As with the UI designer, you can set these properties by changing their values in the right column of the Inspector on the Properties page. It is possible that for some classes hand coding is necessary for properties that cannot be displayed in the Inspector.

## Creating a Canvas

---

The `Canvas` class, represented by  in the designer, is used for drawing graphics to the display in applications such as games. It is also used for handling low-level events. The `Canvas` gives you complete control over what is drawn on the screen. However, you have to program this all manually. It cannot be done in the UI designer.

Since the `Canvas` is interchangeable with the `Screen` components, an application can use a combination of both `Canvas` and `Screen` components in an application. For example, you could use a `List` to provide menu choices for a game, and a `Canvas` for the game itself.

To use the `Canvas` class, you must subclass it in an application and provide an implementation of the abstract `paint()` method in the subclass. When you create a new `Displayable` that extends `Canvas` using the MIDP MIDlet wizard or the MIDP `Displayable` wizard, `JBUILDER` does this for you

automatically. Below is the code generated for a Canvas Displayable class by the MIDP MIDlet wizard:

```
package mypackage;

import javax.microedition.lcdui.*;

public class Displayable1 extends Canvas implements CommandListener {
    /**Construct the displayable*/
    public Displayable1() {
        try {
            jbInit();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }

    /**Component initialization*/
    private void jbInit() throws Exception {
        // set up this Displayable to listen to command events
        setCommandListener(this);
        // add the Exit command
        addCommand(new Command("Exit", Command.EXIT, 1));
    }

    /**Handle command events*/
    public void commandAction(Command command, Displayable displayable) {
        /** @todo Add command handling code */
        if (command.getCommandType() == Command.EXIT) {
            // stop the MIDlet
            MIDlet1.quitApp();
        }
    }

    /** Required paint implementation */
    protected void paint(Graphics g) {
        /** @todo Add paint codes */
    }
}
```

Other methods which report events are not declared abstract and do nothing allowing the application to override the methods that actually report events which are of interest to the application.

For more information, see the reference documentation on [javax.microedition.lcdui.Canvas](#).

## Creating a Ticker

---

A Ticker can be created in the UI designer, but it is not contained by a Form. It is a property of the screen, and its position is determined by the target device.

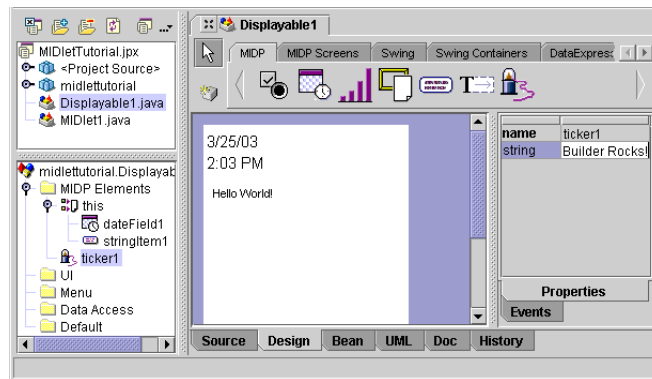


Creating a `Ticker` involves two steps: Instantiating a new `Ticker`, then specifying that `Ticker` in the `Displayable` class `ticker` property.

Below are the steps for creating a `Ticker` using the UI designer:

- 1 Click the `Ticker` component on the palette then click anywhere in the component tree to instantiate it. The new `Ticker` component is only visible and selectable in the component tree. Also, you cannot control where the `Ticker` displays on the screen. That is controlled at runtime by the device screen.
- 2 Select the new `Ticker` in the component tree. Type text for its `string` property value in the Inspector.

**Figure 5.12** Creating a `Ticker` and setting string text

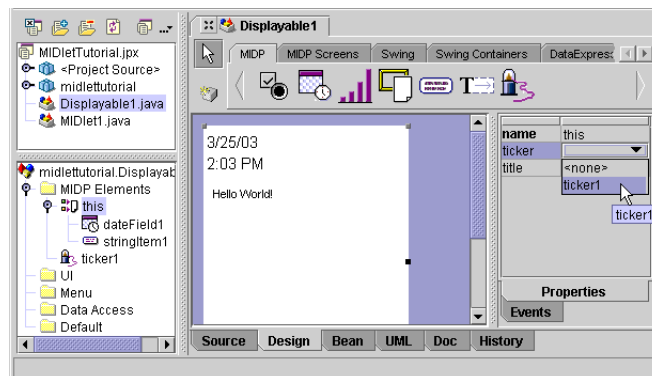


- 3 Click on `this` in the component tree and select `ticker1` from the drop-down list for the `ticker` property.

This step is necessary for the `Displayable` class to display the `Ticker`. It tells the class that there is a `Ticker` to display, and names which one to display. The code generated to do this is

```
this.setTicker(ticker1);
```

**Figure 5.13** Setting 'this' to 'ticker1' in the Inspector



# MIDP events

---

## Overview

---

JBuilder's UI designer provides support for generating event-handling skeleton code for all events supported by the currently selected object in the Designer. This support was based on the Java 1.1 AWT event model where:

- 1 Events all descend from a parent `Event` class (`java.util.EventObject`) that defines common event attributes such as name, ID, and source.
- 2 Event listener methods accept the event as the sole argument.
- 3 Listeners are attached to event sources.

MIDP event-handling does not conform to the JavaBeans component model. MIDP events have the following differences from core Java events:

- 1 MIDP event-handling does not have event types. Instead there are *commands* and *items*.
  - High-level API input handling uses abstract commands assigned by the developer to a `Displayable` for the user to be able to provide input. Each MIDP implementation maps these commands to appropriate soft buttons or menu items for that specific device.
  - Items are MIDP components that are not of type `Displayable`.
- 2 There is a `CommandListener` and an `ItemStateListener` for handling commands and item changes. These listeners are attached to `Displayables`. For instance, if the developer is interested in listening for state changes on an item, the developer would attach a listener to the item's `Displayable`, not to the item itself.
- 3 Listener methods can accept multiple arguments.

## Generating MIDP event-handling code

Generating event-handling code in the UI designer is the same as for core Java applications in the UI designer. To create event-handling code in a `Displayable` file from the UI designer, use the following steps:

- 1 Open the `Displayable` file in the UI designer that extends `Form`.
- 2 Click on the Events tab of the Inspector. You'll see two events, `commandAction` and `itemStateChanged`.
- 3 Double-click in the right column beside the event you want to create the default event-handling skeleton code. The `AppBrowser`

automatically switches you to the Source view and places the cursor inside the braces of the newly generated method.

- 4 Add the additional code to complete the event-handler, then save your file.

## Using images with MIDlets

---

MIDlets only use images in the portable graphics (PNG) format, and the image size is restricted due to the small screens and memory on handheld devices. Also, the images must be delivered in the JAR file to be available to the MIDlet Suite.


You can use images inside the MIDlet as part of the UI. You can also use images as icons to represent the MIDlets in a MIDlet Suite, as well as to represent the MIDlet Suite itself.

### Adding PNG images to a MIDlet Displayable

---


The Mobile development UI designer provides support for specifying images to use in the Displayable UI. When you reference an image in the project through the Image property in the Inspector, JBuilder automatically generates code that specifies the location of that image so you don't have to code it manually.

To add a PNG image in your MIDlet Displayable,

- 1 Put the image in your project directory hierarchy.
-  2 Add it to your project in JBuilder using the Add Files/Packages button at the top of the project pane. This now makes the image available as a property value for selection in the Inspector.
- 3 Select the `ImageItem` of the MIDP page of the component palette.
- 4 Drop it onto the Displayable in the designer or in the component tree.
- 5 Specify the image to use for the `ImageItem`'s `image` property in the Inspector. Once specified, the image becomes visible in the designer.

### Specifying an image to represent the MIDlet Suite

---


- 1 Put the image in your project directory hierarchy.
-  2 Add it to your project in JBuilder using the Add Files/Packages button at the top of the project pane.
- 3 Save and rebuild your project.

- 4 Choose Wizards | Archive Builder to archive your project. Complete the steps of the wizard up to Step 6, “Specifying the optional attributes of this MIDlet Suite”.
- 5 Click ellipsis (...) button on the Icon field and choose the PNG image to represent the MIDlet Suite.

The Archive Builder automatically inserts the icon information as part of the MIDlet<n> attribute in the Manifest file. If you are not using the Archive Builder to archive your MIDlet Suite, you can add this attribute to the Manifest file using a text editor.

## Specifying images to represent MIDlets within a MIDlet Suite

---

- 1 Put the image in your project directory hierarchy.
-  2 Add it to your project in JBuilder using the Add Files/Packages button at the top of the project pane.
- 3 Save and rebuild your project.
- 4 Choose Wizards | Archive Builder to archive your project. Complete the steps of the wizard up to Step 7, “Specify the Name attributes for this MIDlet”.
- 5 Select a MIDlet in the list, click Edit.
- 6 Click ellipsis (...) button on the Icon field in the MIDlet-n Attributes dialog box. Choose the PNG image to represent the MIDlet Suite.

The Archive Builder automatically inserts the icon information as in the MIDlet-Icon attribute in the Manifest file. If you are not using the Archive Builder to archive your MIDlet Suite, you can add this attribute to the Manifest file using a text editor.

## Customizing the component palette

---

You can do various things to customize the component palette, such as adding, deleting, or changing the order of components and tabs.

To open the Palette Properties dialog box, choose Tools | Configure Palette. Help on using this dialog box is available by clicking the Help button.

**Important** When adding MIDP classes to the palette, you must choose the No Filtering option when adding components to the palette in the Palette Properties dialog box.



For more information on customizing the component palette, see “Managing the component palette” in *Designing Applications with JBuilder*.

## Component initialization

When a person drops a MIDP component onto the design surface, the designer updates the source code to reflect the addition of the component. Because MIDP components do not have default constructors, the designer uses the default initialization string associated with component. This default initialization string is specified in the `palette.ini` file in the `<JBuilder home>` directory.

If you don’t want to use the default constructor arguments specified in the `palette.ini` file, you can modify the default strings by hand-editing the `palette.ini` file.

Open `palette.ini` in any text editor to modify its contents. However, this is something only an advanced user should do.

## Viewing class reference documentation for MIDP classes

---

You can view reference documentation for a class in the AppBrowser by doing one of the following:

- Click on the Source tab, then double-click a class name in the structure pane to display the source code for that class in the content pane. Click the Doc tab to view the documentation for that class.
- Choose Search | Browse Classes, and choose a class. Click the Doc tab to see the documentation for that class.

**Note** In order for the class documentation to be found, the `import` statements at the top of your source file must point to the desired class or package.

For more information on viewing documentation see Using JBuilder's online help in *Introducing JBuilder*.

# MIDP database programming

## Overview

---

The storage capabilities of various handheld devices are very limited due to their small memory, and the storage requirements for each device are quite different. Because of this, the persistent storage mechanisms used for Java2 Standard Edition to store data are unsuitable for J2ME MIDP devices.

To prevent objects and their states from being destroyed when a MIDlet exits, you need a non-volatile place to store database information. J2ME MIDP uses the Record Management System (RMS) which is a simple record-oriented database for storing and receiving data to a record store.

RMS stores a MIDP record store as a collection of records that persist after a MIDlet exits. You can retrieve the information in the record store later when you run the MIDlet again.

## Using RMS

---

To use the RMS, import the `javax.microedition.rms` package.

It is beyond the scope of this documentation to instruct you on how to do J2ME RMS programming. However, we do provide examples of using RMS in the Mobile development StockTracker sample, and [Chapter 11, “Tutorial: StopWatch MIDlet,”](#) and sample. The Mobile development samples are found in the `<JBuilder home>/samples/Mobile` directory.

Below is a list of additional resources on RMS.

- API documentation for the `javax.microedition.rms` package.
- “MIDP Database Programming using RMS: a Persistent Storage for MIDlets,” at <http://wireless.java.sun.com/midp/articles/persist/>
- “Record Management System Basics”, in the J2ME Tech Tips at <http://developer.java.sun.com/developer/J2METechTips/2001/tt0220.html#tip2>
- “Using Record Stores Efficiently”, at <http://developer.java.sun.com/developer/J2METechTips/2001/tt0319.html#tip1>
- “Enumerating, Filtering and Sorting MIDP Record Stores”, at <http://developer.java.sun.com/developer/J2METechTips/2001/tt0622.html#tip2>



## Archiving MIDlets

Before you can deploy a MIDlet to the device, you must archive the classes into a JAR file and modify the manifest file that's inside the JAR file. The manifest file must include certain attributes about the MIDlet Suite and the individual MIDlets in the JAR file.

You must also create an external JAD (Java Application Descriptor) text file which also describes the contents of the JAR file, includes required and optional MIDlet attributes, and can pass parameters to the MIDlets at runtime.

- You can use the Archive Builder to automatically create all three files. The steps in this wizard prompt you for the necessary information. See [“Archiving MIDlets with the Archive Builder”](#) on page 7-6.

**Important** The JAR file you create must contain only preverified classes. If you create the JAR file with the Archive Builder in JBuilder, this is done automatically. If you create the JAR file using `jar`, be sure to include only classes that have been processed by the preverifier.

For more information about JAR and Manifest files in general, see “About Java Archive Files (JAR)”, and “About the Manifest file”, found in “Deploying Java programs” in *Building Applications with JBuilder*.

### Archiving MIDlets from the command line

The process of archiving from the command line consists of the following basic steps:

- 1 Compiling and running your project in JBuilder to generate preverified classes. Be sure to do the following:
  - a Create a runtime configuration that includes Make or Rebuild. See “Setting Runtime Configurations” in *Building Applications with JBuilder*.

- b** Save all files.
  - c** Run the application or rebuild the project. This step generates preverified classes and places them in the project <output path> folder.
- 2** Creating a manifest file that contains MIDlet attributes to add to the contents of the default manifest file during the **jar** process.
- 3** Creating the JAR file with **jar** using the **m** option to add the contents of your manifest file to the one in the JAR file.
- 4** Running **jar** again with the **u** option to update the JAR file by adding any additional resource files not already included in the JAR file.
- 5** Creating a Java Application Descriptor (JAD) text file containing MIDlet attributes the application manager uses to run the MIDlet.

## Creating the manifest file

The **jar** command creates a default manifest that is a text file stored as `META-INF/MANIFEST.MF` in the JAR file. However, this manifest file does not include anything about the MIDlets in the archive. When you are archiving MIDlets, you must modify the default manifest file to include MIDlet attributes.

Example of a Manifest file which includes MIDlet attributes:

```
Manifest-Version: 1.0
MIDlet-Name: UIDemoMIDlet
MIDlet-1: UIDemoMIDlet, /icons/ivanIcon.png,
        com.borland.jbuilder.samples.micro.uidemo.UIDemoMIDlet
MIDlet-Icon: /icons/ivanIcon.png
MIDlet-Version: 1.0
MIDlet-Vendor: Borland Software Corporation
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0
```

The **m** option to **jar** allows you to add information from a text file to the default manifest during creation of the JAR file. Therefore, before you create the JAR file, you need to create the external manifest file with the additional information so you can specify it in the **jar** command using the following format:

```
jar cmf manifest-addition jar-file input-file(s)
```

Use a standard text editor to create the external text file listing the MIDlet attribute entries. Below is a description of these attributes.

## MIDlet manifest attributes

The manifest file in a MIDlet JAR file must include lines of *name-value* pairs which define the attributes of the MIDlet(s). Some of these attributes are required, while others are optional.

Below is a list of the MIDlet attributes for a manifest file:

Required attributes	
MIDlet-Name	The name of the MIDlet Suite that identifies the MIDlets to the user.
MIDlet-Version	The version number of the MIDlet Suite in the format <major>.<minor>.<micro>. The default micro revision number is zero.
MIDlet-Vendor	The organization providing the MIDlet Suite.
MicroEdition-Profile	The name and version of the J2ME profile required to run the MIDlet Suite. The format must be in the format MIDP-1.0.
MicroEdition-Configuration	The name and version of the J2ME configuration required to run the MIDlet Suite. The format must be in the format CLDC-1.0.
MIDlet-<n>	The name, icon, and primary class of each MIDlet in the MIDlet Suite. Enter one line for each MIDlet, with the values separated by commas.
	<div>Name</div> <div>Icon</div> <div>Class</div>
	<div>Used to identify the MIDlet to the user on the display. The name MIDlet-1 refers to the first (and sometimes only) MIDlet in the JAR file. Successive MIDlets are assigned numbers incrementally, such as MIDlet-2, MIDlet-3, and so on.</div> <div>Name of the icon image (PNG) within the JAR file to display beside the name of the MIDlet.</div> <div>The class name of the MIDlet. The class must have a public constructor with no arguments.</div>
Optional attributes	
MIDlet-Icon	The name of the icon image (PNG) within the JAR file used to represent the MIDlet Suite.
MIDlet-Description	A description of the MIDlet.
MIDlet-Info-URL	A URL that describes the MIDlet Suite further.
MIDlet-Data-Size	The minimum number of bytes of persistent data storage required on the device for the MIDlet to run. The default if not specified is zero.

## See also

- “Understanding the Manifest”, at <http://java.sun.com/docs/books/tutorial/jar/basics/manifest.html>
- “Modifying a Manifest File”, at <http://java.sun.com/docs/books/tutorial/jar/basics/mod.html>

## Creating the JAR file outside of JBuilder

---

To create the JAR file,

- 1 Make sure **jar** is included in your `CLASSPATH` environment variable. To check this, type `jar -version` at the command line. The result should be that the version is displayed on the command line. If this is not the result, do what is necessary to put it on the class path.
- 2 Create a special archive directory for the MIDlet which will contain all the files needed by your MIDlet. This is optional, but makes the archive process easier to manage. Be sure to give it the correct directory structure required for your MIDlet to run.
- 3 Move the preverified class files from the project's project <output path> directory to your archive directory into a `classes` subdirectory.

**Tip** You can save yourself an extra step in the JAR process if you also move any additional resources your MIDlet needs (such as images) to this directory so they get included in the first archive pass. Otherwise, you will have to run the JAR process a second time to update the JAR file using the `u` option to add these. See "Updating a JAR file", at <http://java.sun.com/docs/books/tutorial/jar/basics/update.html>.

- 4 Create a manifest file that contains the required attributes for the MIDlet. See "[Creating the manifest file](#)" on page 7-2.
- 5 Run **jar** from the command line in the project's parent directory using the following format:

```
jar cmf manifest-addition jar-file input-file(s)
```

**Tip** You can use `*` as a wild card in place of `input-file(s)` to include all the files in the directory and sub-directory.

For example, let's suppose you are archiving an application called `MIDletTutorial`:

- a The current directory is the parent directory of `MIDletTutorial`
- b You created a manifest text file called `midletattributes`
- c The JAR file you want to create will be called `MIDletTutorial.jar`

You would create the JAR file with this command:

```
jar cmf midletattributes MIDletTutorial.jar *
```

### See also

- "Using JAR Files: The Basics", from *The Java Tutorial* at <http://java.sun.com/docs/books/tutorial/jar/basics/index.html>.

## Creating the JAD file outside of JBuilder

---

The JAD (Java Application Descriptor) file is a standard text file with a .jad extension that is external to the JAR file. Like the manifest, it consists of a series of attributes that describe the MIDlet in the MIDlet Suite. The JAD file is used by the application management software to manage the MIDlets in the JAR file.

There is a predefined set of MIDlet attributes to be used in every JAD file. Some of these attributes are shared with the manifest. The `MIDlet-Name`, `MIDlet-Vendor`, and `MIDlet-Version` attributes must have identical values in both the manifest and the JAD file. Of the others that are shared, the ones in the JAD file override the ones in the manifest.

Below is an example of a JAD file with MIDlet attributes:

```
MIDlet-1: UIDemoMIDlet, /icons/ivanIcon.png,
         com.borland.jbuilder.samples.micro.uidemo.UIDemoMIDlet
MIDlet-Icon: /icons/ivanIcon.png
MIDlet-Jar-Size: 32127
MIDlet-Jar-URL: Midp_UIDemo.jar
MIDlet-Name: UIDemoMIDlet
MIDlet-Vendor: Borland Software Corporation
MIDlet-Version: 1.0
```

To see the manifest file that goes with this JAD file, see [“Example of a Manifest file which includes MIDlet attributes:” on page 7-2](#).

As with the manifest file, the JAD file includes both required and optional attributes. It can also include parameters which can be passed to the MIDlet.

### JAD file attributes

Below are the MIDlet attributes for a JAD file:

#### Required attributes

<code>MIDlet-Name</code>	The name of the MIDlet Suite that identifies the MIDlets to the user.
<code>MIDlet-Version</code>	The version number of the MIDlet Suite in the format major.minor.micro. The default micro revision number is zero.
<code>MIDlet-Vendor</code>	The organization providing the MIDlet Suite.
<code>MIDlet-Jar-URL</code>	The URL from which the JAR can be downloaded.
<code>MIDlet-Jar-Size</code>	The size of the JAR file in bytes. (This information is used by the application manager to determine if JAR file can be downloaded to the device.)

**Required attributes**

MIDlet-<n>	The name, icon, and primary class of each MIDlet in the MIDlet Suite. Enter one line for each MIDlet, with the values separated by commas.	
	Name	Used to identify the MIDlet to the user on the display. The name MIDlet-1 refers to the first (and sometimes only) MIDlet in the JAR file. Successive MIDlets are assigned numbers incrementally, such as MIDlet-2, MIDlet-3, and so on.
	Icon	Name of the icon image (PNG) within the JAR file to display beside the name of the MIDlet.
	Class	The class name of the MIDlet. The class must have a public constructor with no arguments.

**Optional attributes**

MIDlet-Icon	The name of the icon image (PNG) within the JAR file used to represent the MIDlet Suite.
MIDlet-Description	A description of the MIDlet.
MIDlet-Info-URL	A URL that describes the MIDlet Suite further.
MIDlet-Data-Size	The minimum number of bytes of persistent data storage required on the device for the MIDlet to run. If not specified, the default is zero.
[MIDlet specific attributes]	Attributes not starting with "MIDlet-" that are related to specific MIDlets.

## Archiving MIDlets with the Archive Builder

---

The JBuilder Archive Builder is a wizard that does the entire archiving process for you, including creating the JAR, manifest, and JAD files.

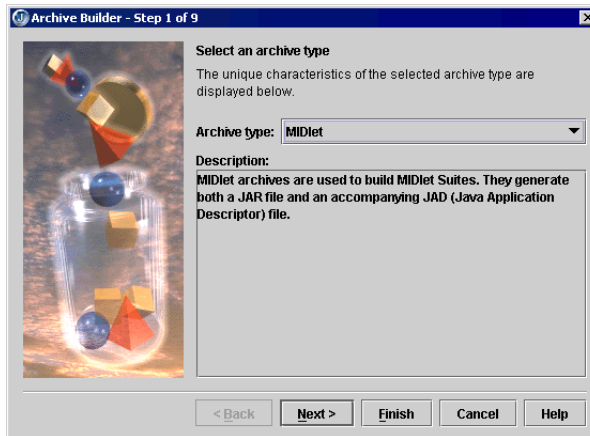
The Archive Builder significantly speeds up the process of packaging your MIDlets for deployment. It searches your project for all the required classes and gives you the opportunity to customize the JAR file contents before it archives the files into a JAR file with the appropriate manifest file. It also creates the accompanying JAD (Java Application Descriptor) file, based on information input through the wizard, and gives you the options of compressing and obfuscating the contents of the archive file.

The steps below take you through the archiving process with the JBuilder Archive Builder. The MIDlet created with the MIDlet Tutorial is used in this example, but you can apply the information in these steps to any MIDlet you have created.

**Important** Before you run the Archive Builder the first time, you must compile your project to create the class files for your project. The Archive Builder always uses the most recently built class files but does not check to see if they are up to date. If you make changes to your source since compiling, be sure to compile your project again before running the Archive Builder.

- 1 Choose Wizards | Archive Builder from the JBuilder menu to open the Archive Builder.

**Figure 7.1** Archive Builder, Step 1

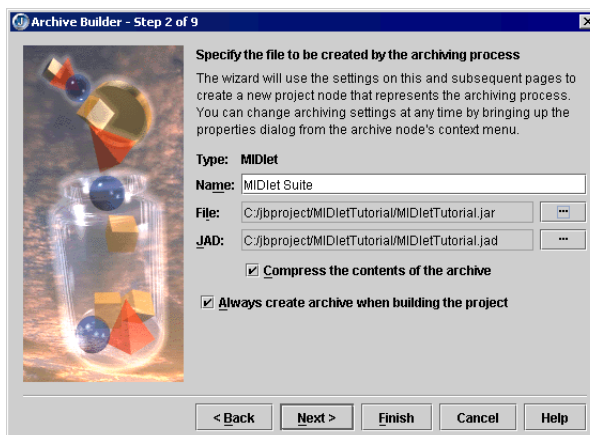


- 2 Select MIDlet as the Archive Type in Step 1. This will generate a MIDlet Suite consisting of a JAR file with a manifest, and a JAD (Java Application Descriptor) file.

**Note** You can press Finish on any step of the wizard to have it automatically generate default information for the JAD file and JAR manifest file.

- 3 Click Next to go to Step 2.

**Figure 7.2** Archive Builder, Step 2



- 4 Type in a name for the MIDlet Suite in Step 2.
- 5 Leave the name and location of the JAR and JAD files as they are unless you wish to change it. To change it, click the ellipsis (...) button and browse to the directory for the JAR file. JBuilder automatically gives the JAR file the same name as the package name, and matches the name and location of the JAD file to the JAR file.

**Warning**

You can change the name of the JAD file with no consequences. However, if you change the path to the JAD file, it will change the MIDlet-Jar-URL attribute in the JAD file from a relative path to an absolute path. This creates a problem when you deploy the application to the actual device, as the file structure on the device will not match the file structure on your computer.

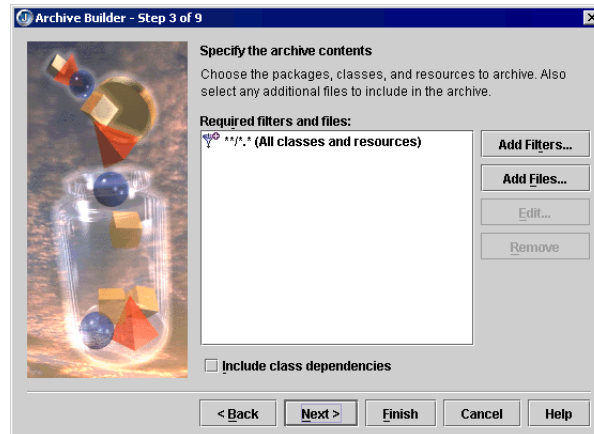
- 6 Leave Compress The Contents Of The Archive checked. You want the JAR file compressed since you're downloading it to a limited device. Leave Always Create Archive When Building The Project checked so the JAR file is recreated each time you make or build your project.

**Tip**

If you want more detail about what's involved in each step of the wizard, click the Help button at the bottom of each step.

- 7 Click Next to go to Step 3.

**Figure 7.3** Archive Builder, Step 3



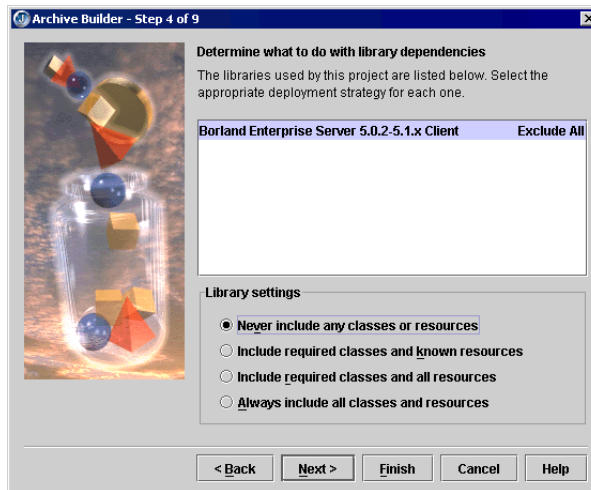
- 8 Specify which files to include in the archive. Click Add Filters to specify filters for the Archive Builder to use in including or excluding classes, and click Add Files to indicate any specific files that are required, such as any classes or resources called at runtime that the Archive Builder wouldn't be able to determine. For more information on using this step of the Archive Builder, press the Help button at the bottom of the step:



See also “Deploying Java Programs” and “Using the Archive Builder” in *Building Applications with JBuilder*.

- 9 Click Next to go to Step 4.

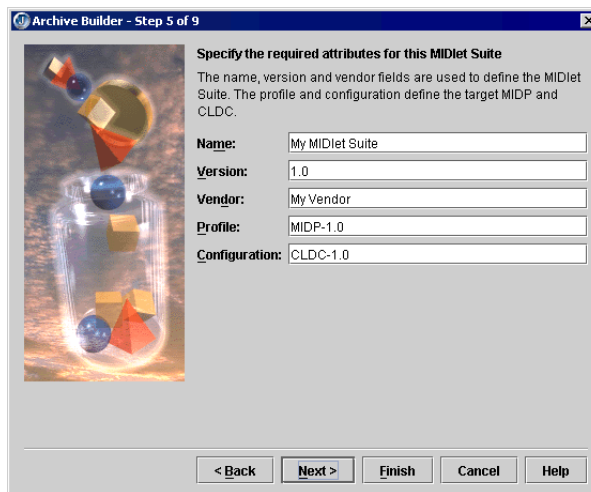
**Figure 7.4** Archive Builder, Step 4



- 10 Indicate in Step 4 what to do with library dependencies, if there are any libraries listed. You can choose an individual deployment strategy for each one.

- 11 Click Next to go to Step 5.

**Figure 7.5** Archive Builder, Step 5



**12** Specify the required attributes for the MIDlet Suite in Step 5.

**Important** All fields must have values in this step, and the Version, Profile, and Configuration must use the following formats:

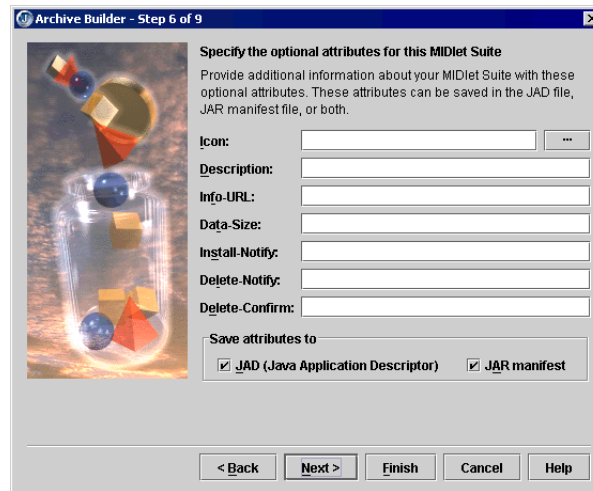
Version: <major>.<minor>.<micro> (For example 1.0)

Profile: <profile-major>.<minor>.<micro> (For example MIDP-1.0)

Configuration: <configuration-major>.<minor>.<micro> (For example CLDC-1.0)

**13** Click Next to go to Step 6.

**Figure 7.6** Archive Builder, Step 6

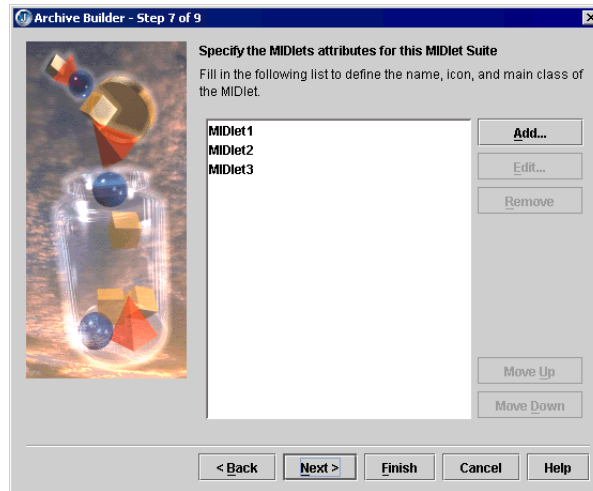


**14** Enter any optional attributes for the MIDlet Suite on Step 6. Click the Help button for more information on these attributes.

Install-Notify, Delete-Notify, and Delete-Confirm are OTA (Over The Air) Provisioning options which only apply if you are creating an archive that is to be installed from an Internet server over the air to the device. Also, you can choose whether to write the MIDlet attributes to both the JAD file and the JAR manifest file, or to only one of them.

15 Click Next to go to Step 7.

**Figure 7.7** Archive Builder, Step 7



16 Specify which MIDlets will be included in the MIDlet Suite and enter their MIDlet-n attributes in Step 7.

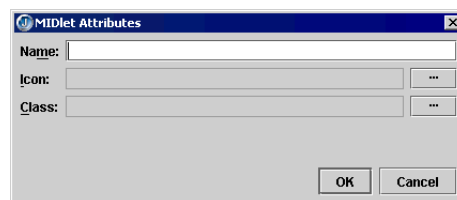
The Archive Builder includes all the available MIDlets from the project by default. It also fills in the Name and Class MIDlet-n attributes automatically for each MIDlet. If you want an icon to represent the MIDlet, you need to explicitly specify that attribute. You can remove MIDlets from the list, add new ones, and change their order.

The MIDlet-n attribute information is required for the manifest and JAD files, and makes it possible to run all the MIDlets in the Suite on the device. It also enables the MIDlets in the Suite to communicate with each other and share the resources in the JAR file.

### Modifying the attributes for a MIDlet

- a Select a MIDlet in the list.
- b Click Edit to open the MIDlet-n Attributes dialog box.

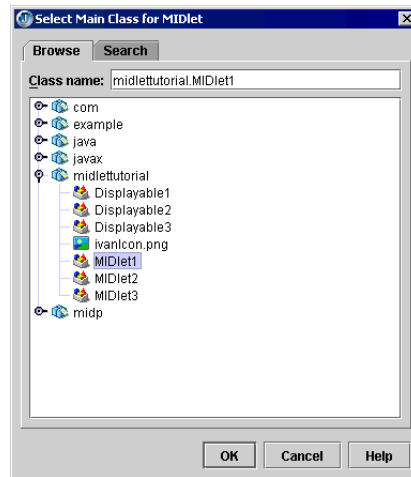
**Figure 7.8** Archive Builder, MIDlet-n Attributes dialog box



- c Change the name of the MIDlet if desired. The MIDlet name is used to identify the MIDlet to the user, and is required.

- d Click the ellipsis (...) button at the right of the Icon field. Choose a PNG (.png) image as the icon for this MIDlet. This is optional. This icon displays next to the MIDlet name on the device.
- e Click the ellipsis (...) button at the right of the Class field. Navigate to the main class for the MIDlet (for example, `midlettutorial.MIDlet1`). The Archive Builder enters the path/class name in the field. This information is required.

**Figure 7.9** Archive Builder, Select a main class for MIDlet dialog box



- f Click OK to return to the MIDlet-n Attributes dialog box.
- g Repeat this process for any additional MIDlets in the MIDlet Suite, altering the Name of the MIDlet and the Class as appropriate for each MIDlet.

### Adding a MIDlet to the list

Click Add, enter the Name, Icon, and Class attributes in the MIDlet-n Attributes dialog box, then click OK.

**Note** The MIDlet must be in the project to add it to the MIDlet Suite.

### Removing a MIDlet from the list

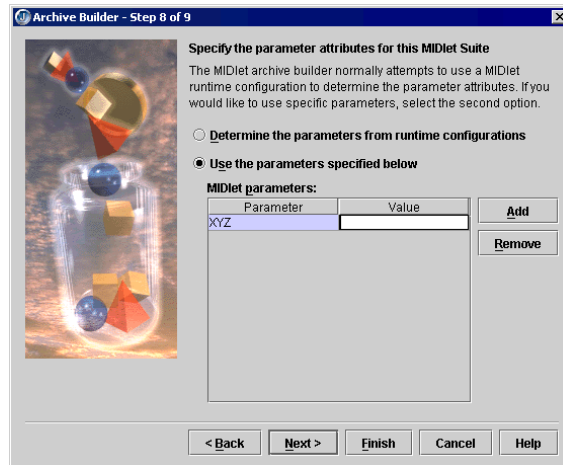
Select the MIDlet you want to exclude from the MIDlet Suite, and click Remove.

### Changing the order of the MIDlets

The order of the MIDlets listed in Step 7 controls their order in the JAD and manifest files, and also the order they are displayed on the device. To move a MIDlet to a different place in the list, select it and click Move Up or Move down.

17 Click Next to go to Step 8 when you are finished.

**Figure 7.10** Archive Builder, Step 8

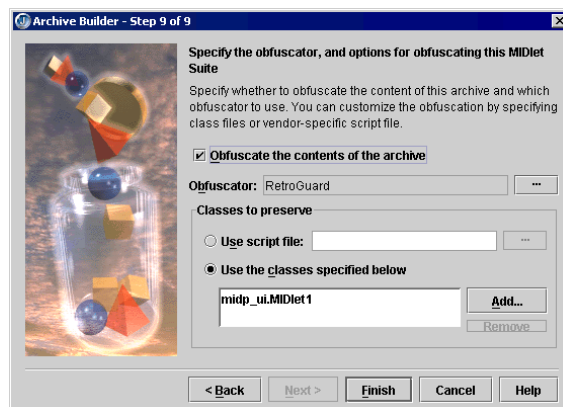


18 Indicate whether to use default runtime parameter attributes or new ones specified here. These parameters are passed to the MIDlet(s) at runtime.

- **Determine the Parameters From Runtime Configurations:** Runtime parameters are set on the Run | MIDlet page, found under the menus Run | Configurations and Project | Project Properties. For more information, see “Setting runtime configurations” in *Building Applications with JBuilder*.
- **Use The Parameters Specified Below:** If the runtime configurations do not have the desired parameters, click Add and enter a parameter name in the first column, followed by a parameter value in the second column. Repeat for additional parameters.

19 Click Next to go to Step 9 when you are finished.

**Figure 7.11** Archive Builder, Step 9



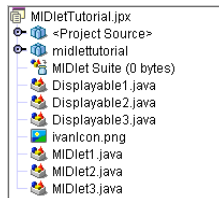
This step allows you to obfuscate your class files as part of the archive process. You must have an obfuscator configured to use this option. For information on configuring and using an obfuscator with JBuilder, see “Obfuscating MIDlet source code” in *Developing Mobile Applications*.

To obfuscate your class files, check Obfuscate the Contents of the Archive, and click the ellipsis (...) button to choose an obfuscator. Specify which files to preserve during obfuscation. You can do this by selecting an obfuscator script file you have previously created to control the obfuscation, or you can click the Add button to manually list the files you want to leave unobfuscated. To remove a file from the list, select it and click Remove.

Press Finish to close the Archive Builder.

Notice that a new node for the archive file has been added to the project pane called MIDlet Tutorial, but you can’t see any nodes for the JAR or JAD file.

**Figure 7.12** Project pane after archiving, but before rebuilding project

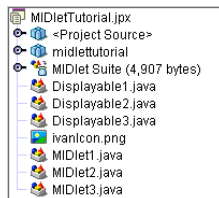


- 20** Right-click the project file in the project pane and choose Rebuild to recompile the project and establish those nodes in the project pane, then click Save All



Now you have an archive node for MIDlet Tutorial that expands. Double-click MIDlet Tutorial to see the nodes for MIDletTutorial.jar and MIDletTutorial.jad.

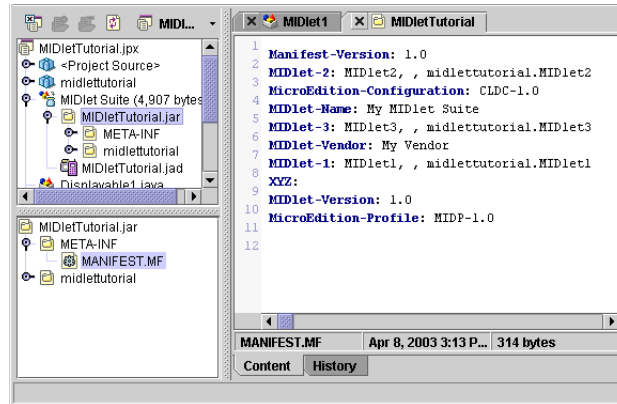
**Figure 7.13** Project pane after archiving and rebuilding project



## JAR file contents

Double-click `MIDletTutorial.jar` and examine the contents of the manifest file which is displayed in the content pane. Notice all the information about the MIDlet that you entered in the wizard. Notice also how the contents of the JAR file are detailed in the structure pane.

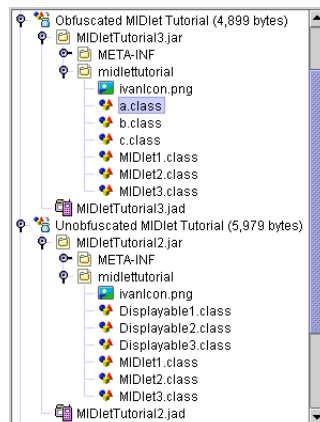
**Figure 7.14** Manifest file



## Obfuscated class files

If you chose to obfuscate your class files, the obfuscator creates new, obfuscated files with different names than the class files and adds the new files to the JAR file instead of the original files. To verify that the obfuscation has taken place, expand the Archive node in the project pane, double-click the JAR file to open it, then expand the MIDlet folder in the structure pane. You'll see files with new names that replace the original files. Below are examples of two JAR files for the same MIDlet: one with unobfuscation and one without.

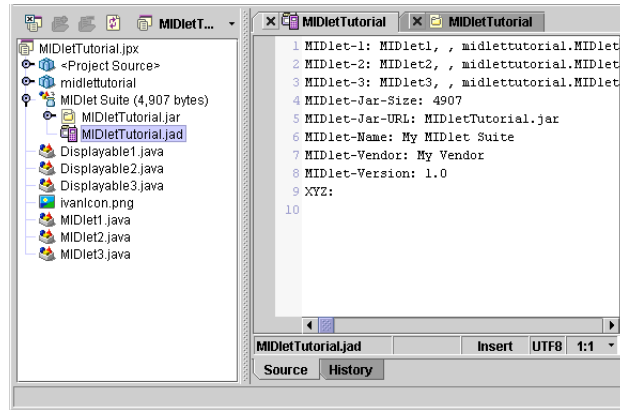
**Figure 7.15** Comparison of obfuscated and unobfuscated JAR files



## JAD file contents

Double-click `MIDletTutorial.jad`. Study the contents of the JAD file. Again, notice the information you entered in the wizard and the similarities and differences between it and the manifest file. The JAD file duplicates some of the information in the JAR file and also adds additional information, for example the size of the JAR file.

**Figure 7.16** JAD file contents





# Developing Mobile Applications: Over The Air (OTA) Provisioning

## What is OTA Provisioning?

---

OTA (Over the Air) Provisioning is the deployment of J2ME MIDlet Suites from an Internet site over the wireless network to wireless devices. With OTA, users aren't required to make trips to a service center or link to a PC to upgrade software on their wireless device.

J2ME OTA Provisioning is different from WAP OTA Provisioning in that it only supports provisioning of JAR and JAD files, while WAP OTA is not limited to Java.

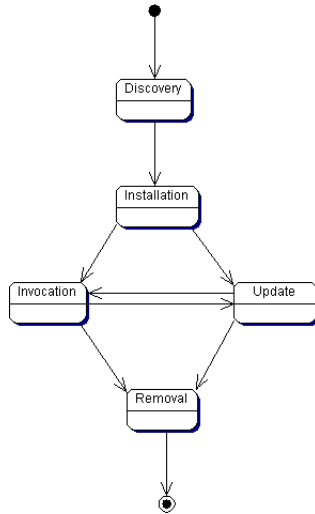
The document "Over The Air User Initiated Provisioning Recommended Practice for the Mobile Information Device Profile (PDF)" describes how MIDlet Suites can be deployed over the air. You can find the link to this PDF file at <http://java.sun.com/j2me/docs/>.

## What's involved in OTA Provisioning?

---

According to Sun's latest OTA Provisioning document, a MIDlet Suite life cycle looks like the following:

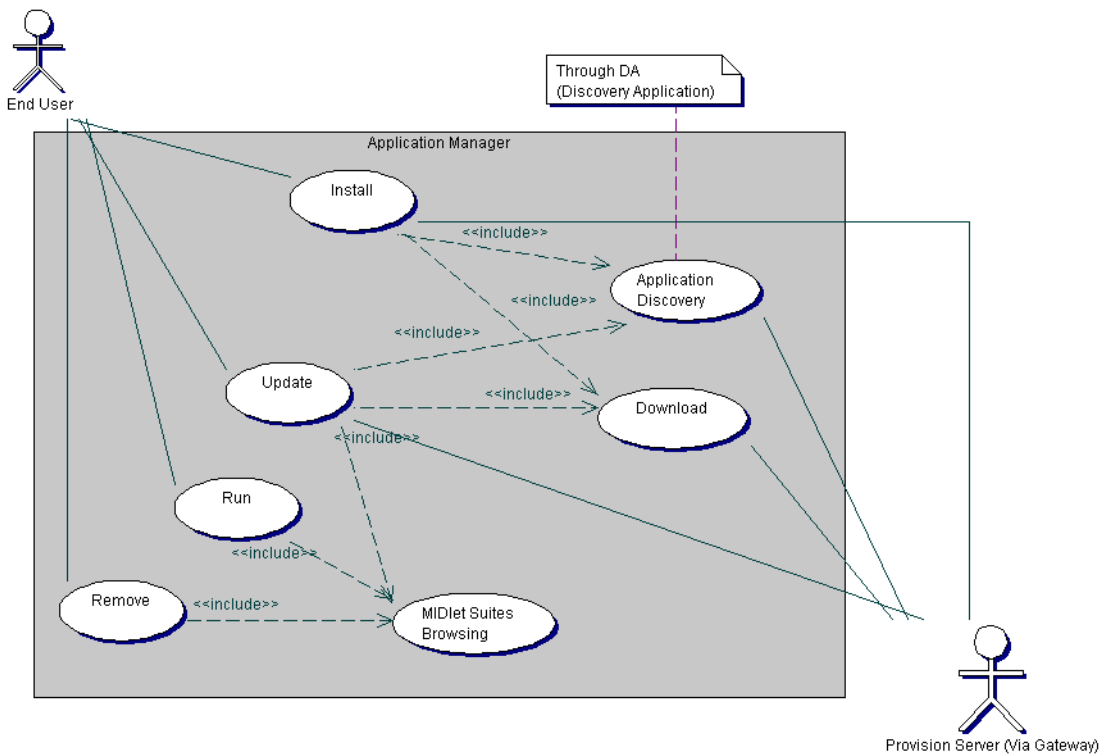
**Figure 8.1** MIDlet Suite lifecycle



During this life cycle, the following elements are involved:

- J2ME Device
- DA (Discovery Application - discovers MIDlet Suites users can download to the device)
- MIDlet Suite
- WAP Gateway
- Provision Server

The diagram below demonstrates the discovery and provisioning process:

**Figure 8.2** OTA Provisioning recommended practice

## Configuring the server for JAD and JAR MIME types

Before you can use a server for OTA Provisioning, it must be configured to recognize the correct MIME types for JAR and JAD files.

- The MIME type for a JAR file is `application/java-archive`.
- The MIME type for the JAD file is `text/vnd.sun.j2me.app-descriptor`. This specifies the default character set to use (UTF-8) to convert the JAD file from its transport format to the correct MIDP Unicode-encoding.

For example,

- If you are using the Tomcat server delivered with JBuilder,
  - a Find the file `web.xml` in the Tomcat install '`conf`' sub-directory (`<JBuilder_home>\thirdparty\jakarta-tomcat-4.1.24\conf`).

**b** Put the following lines in the section “Default MIME Type Mappings”:

```
<mime-mapping>
  <extension>jar</extension>
  <mime-type>application/java-archive</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>jad</extension>
  <mime-type>text/vnd.sun.j2me.app-descriptor</mime-type>
</mime-mapping>
```

- If you are using an Apache server,
  - a** Find the file `mime.types` in the Apache install ‘conf’ sub-directory (`<Apache>\conf\.`)
  - b** Specify the JAD and JAR MIME types in the file as follows:

# MIME type	Extension
application/java-archive	jar
text/vnd.sun.j2me.app-descriptor	jad

See your specific server documentation for directions on specifying MIME type configuration.

## OTA Provisioning

---

Mobile development supports OTA Provisioning from within JBuilder. You can upload your MIDlet Suites to an FTP server without leaving JBuilder, then test them in the emulator by downloading and running them using OTA Provisioning. You can also discover and run third-party MIDlets on the Internet.

### Uploading your MIDlet Suite to a server

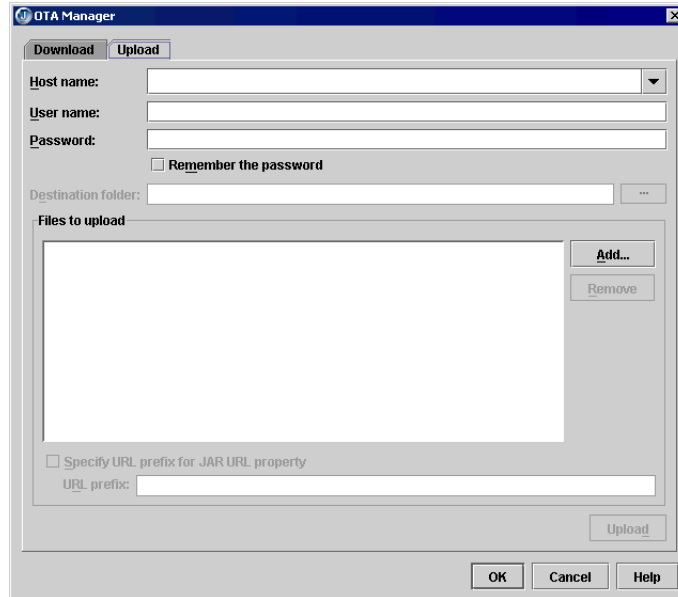
---

Before you can test your MIDlet Suite using OTA Provisioning, you need to archive it to create the JAR and JAD files, then upload those files to the FTP server. You may also want to create an HTML page with links to your JAD file(s) and upload that along with the JAD and JAR files. See “Archiving MIDlets” for instructions on archiving your project.

Once you’ve created the JAR and JAD files, follow the steps below upload your MIDlet Suite to a server from within JBuilder:

- 1** Make sure you are connected to the Internet.

## 2 Choose Tools | OTA Manager and click the Upload tab.



**Note** Alternatively, you can right-click the archive node in the project pane and choose Upload to Server.

- 3 Enter either a fully qualified Internet host name or an IP address for the FTP server in the Host name field. The history of FTP addresses will be retained for this field, so in the future you can just select a previously used address from the drop-down arrow.
- 4 Enter your User ID and your password. Check Remember the Password if you want it saved to disk.
- 5 Enter the name of the destination folder on the remote server where you're going to put the MIDlet Suite files. Either type it in, or click the ellipsis (...) button and browse to it to select it.
- 6 Click the Add button to open a browser and navigate to the local directory where your MIDlet Suite is stored.
- 7 Select the JAD, JAR, and if appropriate, the HTML file that contains the links to the JAD file. Also include any images used by this HTML page. Click OK.
- 8 Modify the list if necessary. To remove files from the list, select the files then click Remove.

- 9 Check Specify URL Prefix for JAR URL Property if you want to override the MIDlet-Jar URL in the JAD file with a specific prefix before it is uploaded to the server. If checked, specify a URL prefix below. The OTA manager will add to or replace the URL prefix specified in the JAD file with this one when the Override JAR URL Property option is checked.

- 10 Click Upload when the file list is complete.

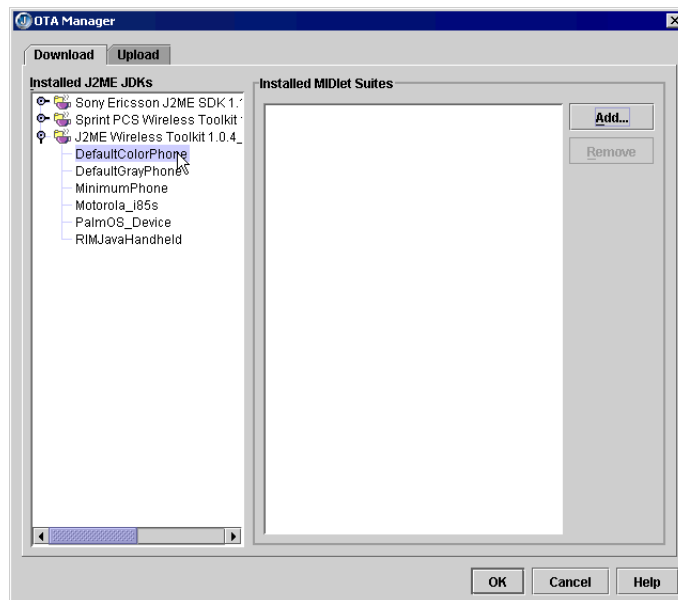
## Downloading MIDlet Suites from the server

Choose Tools | OTA Manager to open the OTA Manager dialog box. It opens by default on the Download page.

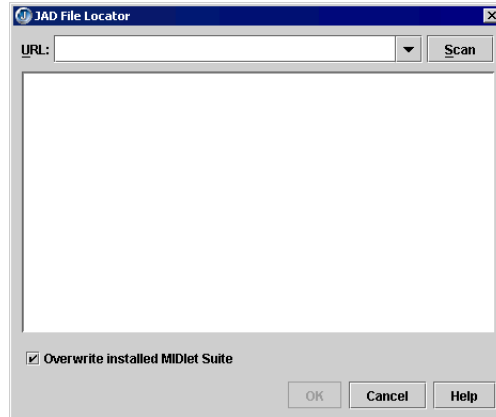
The OTA Manager lists all the installed J2ME SDKs that support OTA Provisioning, and any MIDlet Suites currently downloaded and installed to run on those JDKs.

To download and install a MIDlet Suite,

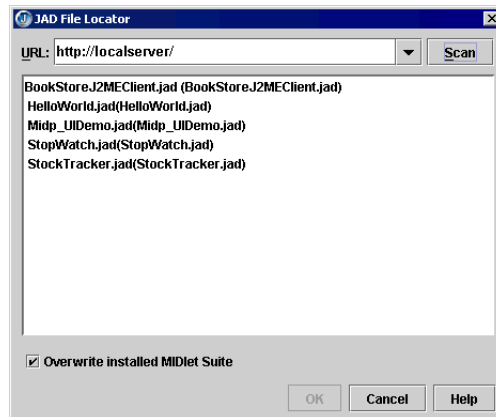
- 1 Expand the node for the JDK you want to use, then choose a device.



- 2 Click the Add button to open the JAD File Locator.



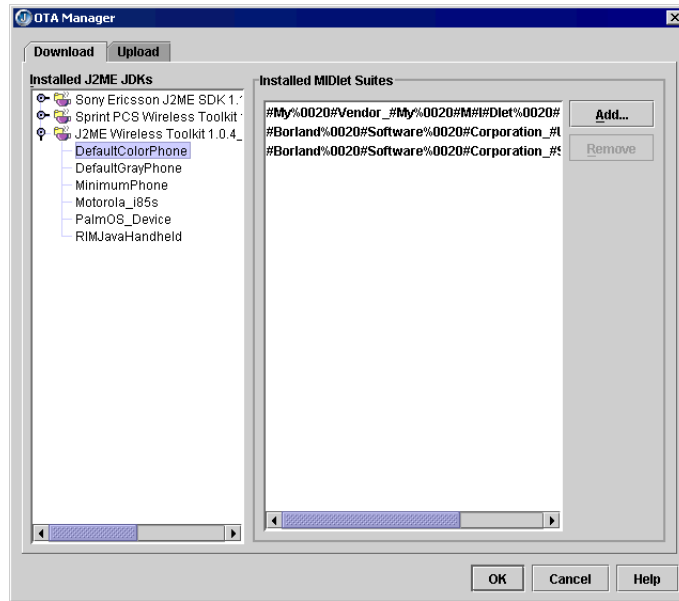
- 3 Type the URL to the HTML page that contains a link to the JAD file for the MIDlet Suite you want to download.
- 4 Click Scan. The names of all the JAD files at that location appear in the text area of the JAD File Locator.



**Note** The JAD File Locator keeps a history of all the URLs you have downloaded before. To use a previous URL, click the drop-down arrow beside the URL field and select it on the list.

- 5 Click on the name of the JAD file, then click OK. The JAD File Locator downloads the selected MIDlet Suite from the server and installs it to

the local JDK. The storage name of the MIDlet Suite is added to the list of MIDlet Suites in the OTA Manager.



**Note** You can select multiple MIDlet Suites on the list in the JAD File Locator dialog box to download at the same time.

- 6 Repeat the process above for any additional MIDlets you want to install on the emulator.

## Running an installed MIDlet Suite

To run an installed MIDlet Suite in the emulator,

- 1 Choose Project | Properties or Run Configurations.
- 2 Select the desired runtime configuration from the list and click Edit, or create a New one by clicking New.
- 3 Set the Type to MIDlet on the Run page of the Runtime Configuration Properties dialog box.
- 4 Check OTA URL, then click the ellipsis (...) button.
- 5 Select Installed on Current JDK, and choose any item from the list of Installed MIDlet Suites.
- 6 Click OK to return to the Runtime Configuration Properties dialog box. Notice now that the URL is displayed next to the OTA radio button.
- 7 Click OK to close the Runtime Configuration Properties dialog box.



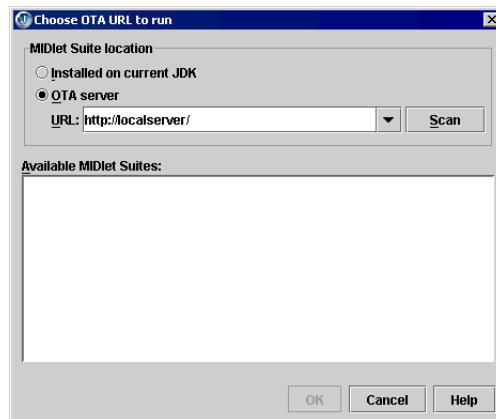
- 8 Select that configuration on the Run page of the Project Properties dialog box, and check the box in the Default column to make it the default runtime configuration. Also check the box in the Context Menu column so this runtime configuration will appear on all the right-click context menus.
- 9 Click OK to close the Project Properties dialog box. You have just changed the default file to run when you click the Run button or choose Run | Run Project.
- 10 Click the Run button to run the installed MIDlet Suite.

## Running a MIDlet Suite directly from a server

---

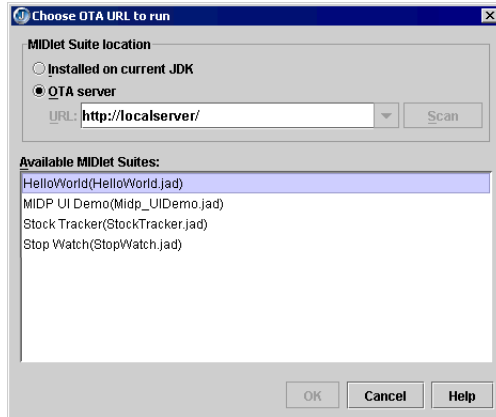
It is possible to run a MIDlet Suite stored on a remote server without downloading and installing it first. To do this,

- 1 Open any project, or create a new project.
- 2 Choose Project | Properties or Run Configurations.
- 3 Select the desired runtime configuration from the list and click Edit, or create a New one by clicking New.
- 4 Click the MIDlet tab on the Run page of the Runtime Configuration Properties dialog box.
- 5 Choose OTA URL, then click the ellipsis (...) button to open a dialog box for choosing the URL.



- 6 Choose OTA Server then specify the URL to a JAD file. You can just name the server and search for available JAD files, or you can specify a specific JAD file.
- 7 Click the Scan button to display the available JAD files at that location.

- 8 Select the JAD file you want to run, then click OK.



- 9 Click OK again to close the Runtime Configuration Properties dialog box.
- 10 Select that configuration on the Run page of the Project Properties dialog box, and check the box in the Default column to make it the default runtime configuration. Also check the box in the Context Menu column so this runtime configuration will appear on all the right-click context menus.
- 11 Click OK to close the Project Properties dialog box. You have just changed the default MIDlet to run for the opened project. Just click the Run button to run the MIDlet.

#### See also

- “Running Java programs” in *Building Applications with JBuilder*
- “Setting runtime configurations” in *Building Applications with JBuilder*

## Resources on OTA Provisioning

---

“Over The Air User Initiated Provisioning Recommended Practice for the Mobile Information Device Profile” (PDF, 24 pages, 99K), link found at <http://java.sun.com/j2me/docs/>.

# JBuilder Mobile development samples

These samples require  
JBuilder Developer and  
Enterprise

Mobile development delivers several samples located in the `<JBuilder home>\samples\Mobile\` directory. To open and run these samples in JBuilder, see the directions below.


**Important**

Before running a sample that use a specific JDK, you must first configure that JDK in JBuilder and have it set as the default JDK. See [“Setting up a JDK” on page 2-2](#).

## Running MIDP samples in JBuilder

### Running from the MIDlet file

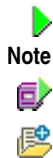
To run a sample from the main MIDlet file, follow the steps below:

- 1 Choose File | Open Project.
- 2 Navigate to the `<JBuilder home>\samples\Mobile` directory.
- 3 Go into the directory for the sample you want to run and open the project file (.jpx) for the sample.
- 4 Choose Project | Project Properties, and on the Paths page, click the ellipsis (...) button beside the JDK field and choose the MIDP/CLDC JDK you want to use to run this MIDlet. Close the Project Properties dialog box.
- 5  Click the Run button on the toolbar, or right click the main MIDlet .java file in the project pane and choose Micro Run.

## Running from a JAD file

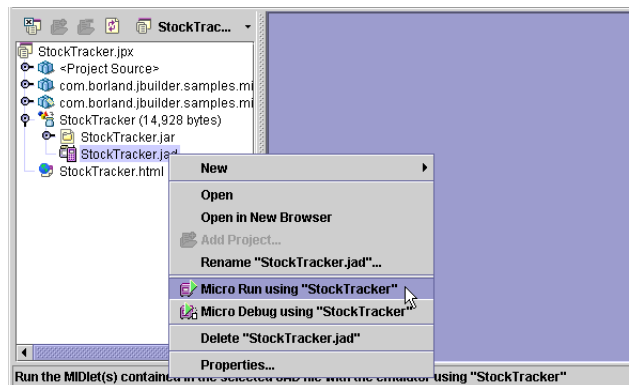
To run the sample from the JAD file, follow the steps below:

- 1 Choose File | Open Project.
- 2 Navigate to the <JBuilder home>\samples\Mobile directory.
- 3 Open the project file (.jpx) for the sample you want to run.
- 4 Choose Project | Project Properties, and on the Paths page, click the ellipsis (...) button beside the JDK field and choose the MIDP/CLDC JDK you want to use to run this MIDlet.
- 5 Switch to the Run page and click New to create a runtime configuration, or select an existing one and click Edit. The Runtime Configuration Properties dialog box displays.
- 6 Set the Type to MIDlet on the Run page of the Runtime Configuration Properties dialog box.
- 7 Select the JAD file option, then click the ellipsis (...) button beside it to open a file browser window.
- 8 Navigate to the JAD file for the project, select it, and click OK.
- 9 Click OK to close the Project Properties dialog box.
- 10 Click the Run button on the toolbar.



**Note**

If the JAD file has been added to your project, you can simply right-click the JAD file in the project pane and choose Micro Run. (Use the Add Files/Packages button to add the JAD file to the project if you archived your project at the command line.)



## Using the emulator

---

The emulator will open and display the MIDlet name(s). How you navigate and select files in the emulator depends on the device being used. See the specific device documentation for this information.

## Samples delivered with JBuilder Mobile development

---

### Hello World

---

The Hello World sample is a very simple Hello World application demonstrating the bare necessities for a complete MIDlet.

`HelloWorldMidlet` is the main MIDlet class. It initializes the MIDlet and instantiates `HelloWorldForm` to display the HelloWorld text.

The directory for this sample is `<JBuilder home>\samples\Mobile\HelloWorld`, and the project file is `<JBuilder home>\samples\Mobile\HelloWorld\HelloWorld.jpx`.

### MIDP UI Demo

---

The MIDP UI Demo is a simple demonstration of the UI elements available in the MIDP API specification and consists of a list of sub-demos. Each sub-demo is a `Displayable` featuring a particular MIDP UI element. The sub-demos are:

- **Canvas:** Illustrates how to use the low-level drawing APIs available in MIDP.
- **Label:** Contains three different ways to put strings and images onto the screen.
- **DateField:** Shows the three different modes of a `DateField` object: `Date Only`, `Time Only`, and `Date/Time`.
- **Alert:** Showcases all defined alert types. Each `Alert` is shown in two ways: as an `Alert` that times out after 5 seconds, and as a modal `Alert` that needs to be dismissed by the user.
- **ChoiceGroup:** Features the `ChoiceGroup` types: `explicit` (only one selection allowed) and `multiple` (multiple selections allowed). There can be multiple `ChoiceGroups` on a single screen (contained in a `Form`). Note that the MIDP API specifies that the `ChoiceGroup` type can only be set by way of a constructor argument. A property setter does not exist for `ChoiceGroup` type.

- **Gauge:** Contains the two types of gauges available in MIDP: interactive, and non-interactive. Note that the MIDP API specifies that the `Gauge` type can only be set by way of a constructor argument. A property setter does not exist for `Gauge` type.
- **ImageItem:** Demonstrates how to display an image in PNG (.png) format.
- **List:** Shows the three different types of `List`: explicit (only one selection allowed), implicit (only one selection allowed, without visual identifiers such as radio buttons), and multiple (multiple selections allowed). Unlike `ChoiceGroup`, there can only be one `List` displayed at any time. Note that the MIDP API specifies that the list type can only be set via a constructor argument. A property setter does not exist for `List` type.
- **TextBox:** Displays a user-editable text box. Only one `TextBox` can be displayed at any time.
- **TextField:** Displays a user-editable text field for inputting/editing a short text string. There can be multiple `TextFields` displayed on a single screen, contained in a `Form`.
- **Ticker:** Illustrates the use of a scrolling marquee of text. Tickers can be set for objects that descend from `Screen`: `List`, `Form`, `Alert`, and `TextBox`.

`UIDemoMidlet` is the main MIDlet class. It initializes the MIDlet and instantiates `UIDemoList` to display the list of subdemos.

The directory for this sample is `<JBuilder home>\samples\Mobile\Midp_UIDemo`, and the project file is `<JBuilder home>\samples\Mobile\Midp_UIDemo\Midp_UIDemo.jpx`.

## Stock Tracker

---

The Stock Tracker sample is a simple stock tracker that allows the user to enter new stocks, delete stocks, and set update interval preferences.

This demo illustrates several concepts:

- How to make a network connection to a remote server
- How to persist data between MIDlet sessions using the MIDP Record Management System (RMS)

`StockTracker.java` is the main MIDlet class. It contains the remote server address information as well as support for reading and saving data through RMS.

`StockTrackerDisplay.java` is this MIDlet's main displayable. It consists of a ticker displaying stock symbols and prices, and a list presenting options for adding stock symbols, deleting stock symbols and adjusting price update intervals.

`StockList.java` is a simple hashtable for holding the stock symbols during a session.

`AddStock.java`, `DeleteStocks.java`, and `SetOptions.java` are Displayable classes for handling adding, deleting, and price update interval adjustments.

The directory for this sample is `<JBuilder home>\samples\Mobile\StockTracker`, and the project file is `<JBuilder home>\samples\Mobile\StockTracker\StockTracker.jpx`.

## Stop Watch

---

The Stop Watch sample demonstrates the following:

- It gives you an introduction to a Canvas.
- It teaches you how to draw on a Canvas, set colors, fill rectangles with color, set fonts, and draw strings.
- It demonstrates having multiple Displayables in a MIDlet and moving from one to the other.
- Advanced part shows you how to save and retrieve information from a Record Management Store (RMS) record store.

The directory for this sample is `<JBuilder home>\samples\Mobile\StopWatch`, and the project file is `<JBuilder home>\samples\Mobile\StopWatch\StopWatch.jpx`.

## MobileBookStore

---

This sample requires  
JBuilder Enterprise

### Overview

The MobileBookStore sample is a simple application that uses J2ME technology as the client tier and the J2EE technology as the middle and backend tiers. It illustrates how J2ME devices using MIDP operate seamlessly as J2EE applications clients.

The J2ME client communicates to the Java servlet in the Web tier through an HTTP connection, the Java servlet accesses the business logic of Enterprise JavaBeans in the EJB tier through a JavaBean, and finally the EJB tier accesses the application database through the JDBC API.

In this application, user will be able to search a book in the database by entering the ISBN and buy the book.

### Description

The MobileBookStore application includes two separate JBuilder projects: The J2ME client program `BookStoreJ2MEClient.jpx`, and the J2EE server program `BookStore.jpx`.

The client program is a MIDlet that runs on the cell phone (or emulator). The user enters the ISBN number for the book on the cell phone. The MIDlet then sends a search request to the MobileBookStore's server using an `HttpConnection`, receives the search results sent back from the search server, and displays them on the phone. After the user buys the book, an `updateInStock` request will be sent to the server, and the server will update the `InStock` quantity in the database.

The server program is hosted by Borland Enterprise Server. It is a Java servlet running in the web container, which communicates to the Enterprise JavaBeans in the EJB container through a `JavaBean`.

## Running the MobileBookStore sample

To run MobileBookStore sample, you must run the server program first, then run the client program. The directions below step you through configuring the server, running the server, then running the client.

### Configuring and running the server

**Important** Borland Enterprise Server 5.1.1 or higher must be installed and setup for JBuilder for this sample to run. If you have not done so, install Borland Enterprise Server.

- 1 Select **Tools | Configure Server...** in JBuilder. Select the Borland Enterprise Server AppServer Edition 5.1.1-5.2.1 node, and point to your installation directory.

**Note** You will need to restart JBuilder if you have setup Borland Enterprise Server for use with JBuilder for the first time.

If you configured Borland Enterprise Server prior to opening the sample project, refresh project properties from your server configuration:

- a Click on **Project | Project Properties**.
- b Click on the **Server** tab
- c Click on the ellipsis (...) button next to the server selected in the **Single server** for all services in project drop down list
- d Click **OK** button on the **Edit or Select Server** dialog.
- e Click **OK** button on the **Project Properties** dialog.

The default port number for Visibroker Smart Agent is 14000. You can modify the port number by selecting **Tools | Enterprise Setup**, clicking the **CORBA** tab and changing the **SmartAgent** port to a different number, for example 15000.



- 2 Modify the database URL to point to the location of the sample MobileBookStore database (Bookstore.jds).
  - a Expand the EJB module node (bookstore) in the project view.
  - b Expand the JDBC 1 DataSource node.
  - c Double click on the data source definition (DataSource).
  - d Edit the URL field and change the directory to point to the Bookstore.jds located in the samples\Mobile\MobileBookStore\BookStore directory under JBuilder home directory.
- 3 Launch the server by clicking on the little down-arrow on the toolbar next to the green arrow. In the drop down select BookStoreServer. This will launch the server with the servlet (bookstoreservlet) loaded in the content pane.

If you see the error message "HTTP Status 500 - No Context configured to process this request", click on the Refresh button in the toolbar above the web view after the server has finished loading.

You can also launch the server by right clicking BookStoreServlet.java and selecting Web Run using "BookStoreServer."

Now the MobileBookStore servlet is running and waiting for J2ME client input.

### Running the client program

- 1 Open BookStoreJ2MEClient.jpx from the <JBuilder home>\Samples\Mobile\MobileBookStore\BookStoreJ2MEClient folder.
- 2 Make sure the JDK for this project is a valid J2ME MIDP/CLDC JDK. Check this in Project | Project Properties on the Paths tab.

**Note** For information on how to configure a J2ME MIDP/CLDC JDK, see ["Setting up a JDK" on page 2-2](#).

- 3 Choose Run | Configurations.
- 4 Click Edit on the Run page to open the Runtime Configuration Properties dialog box, then click the ellipsis (...) button on the Main Class field and choose `com.borland.samples.micro.mobilebookstore.bookstoreJ2MEclient.BookStoreMIDlet` as the Main class to run. (You could also click the JAD file radio button and choose BookStoreJ2MEClient.jad to run.)

- 5 Run the client program by clicking on the down-arrow by the Run button on the toolbar, and choosing `BookStoreJ2MEClient`. Alternatively, you can right click on `BookStoreMIDlet.java` and select Micro Run using “`BookStoreJ2MEClient`.”

For more instructions on this sample, see the `.html` notes file included in the sample itself. The directory for this sample is `<JBuilder home>\samples\Mobile\MobileBookStore`.

The project files for this sample are:

- `<JBuilder home>\samples\Mobile\MobileBookStore\BookStore\BookStore.jpx`
- `<JBuilder home>\samples\Mobile\MobileBookStore\BookStoreJ2MEClient\BookStoreJ2MEClient.jpx`.

# Chapter 10

## Tutorial: Creating and testing MIDlets

The purpose of this tutorial is to introduce you to MIDlet creation using the JBuilder development and design tools such as the wizards, editor, compiler, and UI designer.

The first part of this tutorial steps you through the creation of a basic MIDlet which includes a simple user interface. After creating the MIDlet, you will test the MIDlet in an emulator from within JBuilder.

**Note** The emulator and phone UI images shown in this tutorial are for illustrative purposes and do not represent any specific device.

The basic steps for creating a MIDlet are as follows:

- 1 Creating a project using the Project wizard. The project will contain a project file and an HTML file for keeping notes about the project.
- 2 Creating a MIDlet using the MIDP MIDlet wizard. The MIDP MIDlet wizard creates a `MIDlet` class and a `Displayable` class which displays the UI.
- 3 Designing the UI using the UI designer.
- 4 Compiling and saving your project.
- 5 Running and testing the MIDlet in the emulator.

The second part of the tutorial shows you how to add additional MIDlets to the project, create the archive JAR and JAD files with the Archive Builder, then test the MIDlet Suite by running the JAD file on an emulator.

The main steps in this section of the tutorial include the following:

- 1 Adding additional MIDlets
- 2 Designing the UI for the new MIDlets
- 3 Testing all three MIDlets
- 4 Archiving the MIDlet Suite
- 5 Running from the JAD file

**Important** This tutorial assumes you already have the desired JDK installed and configured in JBuilder. If this is not the case, see [“Setting up a JDK” on page 2-2](#).

## Creating a project

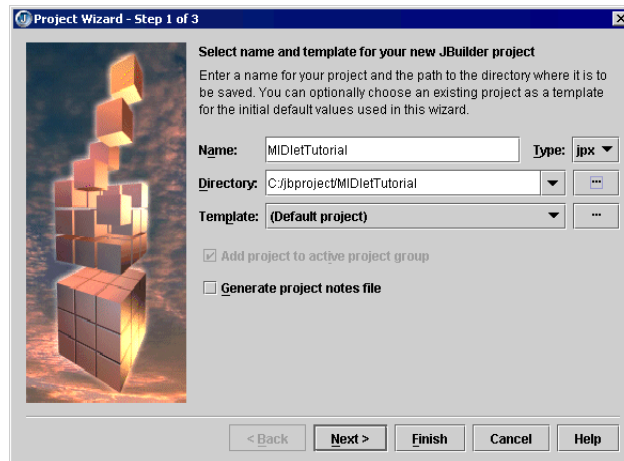
---

Before you can create a MIDlet in JBuilder, you must create a project.

**Note** You can only access the MIDP MIDlet wizard from an opened project.

- 1 Choose File | New Project to start the Project wizard.
- 2 Replace the default Project name with “MIDletTutorial”. The project name must not contain spaces.

**Figure 10.1** Project wizard, Step 1

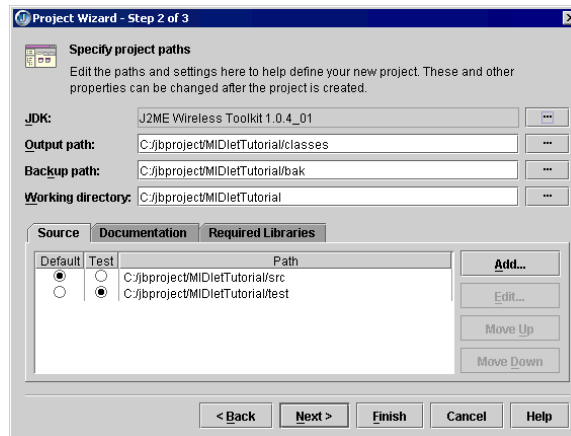


- 3 Leave the rest of the defaults as they are unless you wish to change them. In the future, you may want to create a project you can use as a template that has all the project settings, paths, and JDK defined for your J2ME projects. This saves having to define these settings every time you start a new project. It also lets you have separate templates for each JDK and/or device.

If you check Generate Project Notes File, an HTML file will be generated for keeping notes about the project.

- 4 Click Next.
- 5 Verify, and change if needed, the paths and directory settings for the project in Step 2.

**Figure 10.2** Project wizard, Step 2

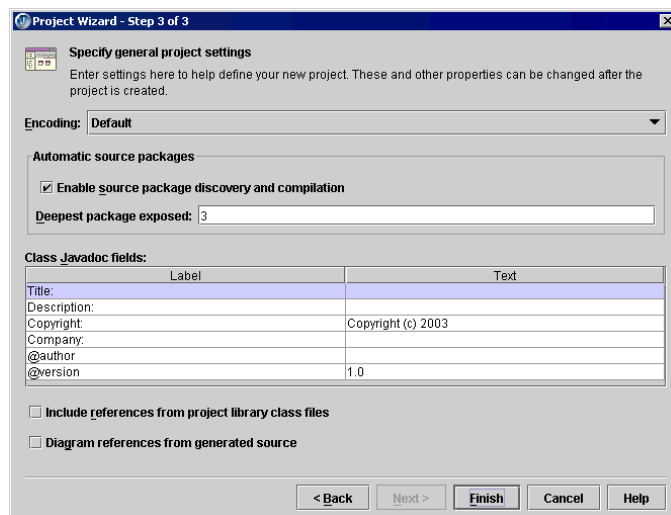


- 6 Click the ellipsis (...) button to open the Select JDK dialog and navigate to the desired JDK.

**Note** You can set the default project paths and directories for all projects on the Paths page of the Default Project Properties dialog box. See “Setting project properties” in *Building Applications with JBuilder*.

- 7 Click Next.
- 8 Enter any additional details you want in Step 3.

**Figure 10.3** Project wizard, Step 3



**9** Click Finish to create the project.

The following files are created: a project file called `MIDletTutorial.jpx`, and a corresponding `MIDletTutorial.html` project notes file which is where you can keep your notes and tasks related to this project (provided you checked that option on Step 1 of the Project wizard.)

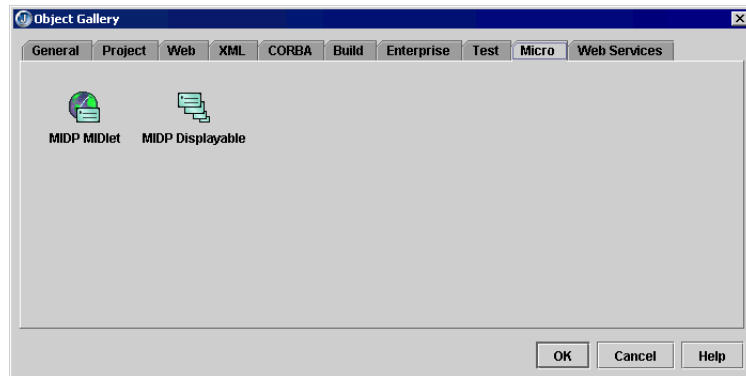
## Creating MIDlet1

---

Now that the project has been created and opened, you can create a MIDlet.

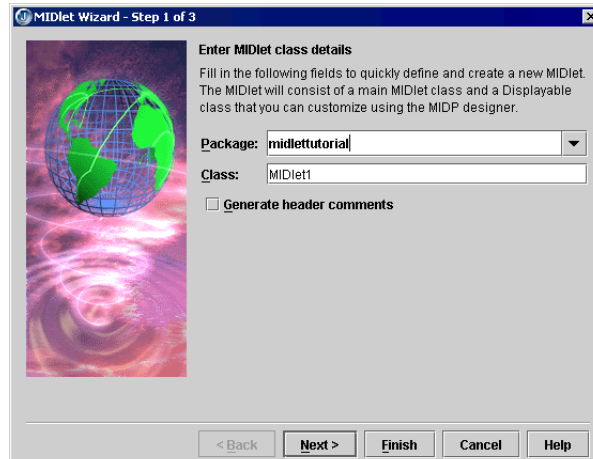
- 1** Choose File | New to open the object gallery.
- 2** Click the Micro tab, then double-click the MIDlet icon to open the MIDP MIDlet wizard.

**Figure 10.4** Object gallery



- 3** Accept the defaults for the MIDlet class details in Step 1, leaving the name of this MIDlet as `MIDlet1`.

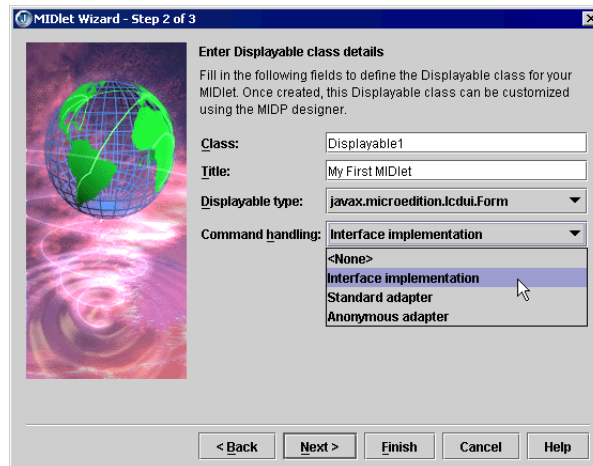
Figure 10.5 MIDP MIDlet wizard, Step 1



Click Next.

- 4 Enter the Displayable class details in Step 2. Leave the class name `Displayable1`, but replace the title with “My First MIDlet”. This title displays at the top of the MIDlet when you run your MIDlet on the device.

Figure 10.6 MIDP MIDlet wizard, Step 2

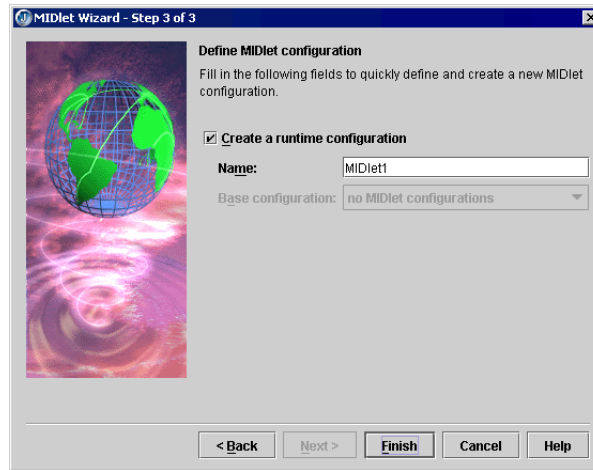


The Displayable Type means the class to extend. Notice that the Displayable Type shows `javax.microedition.lcdui.Form` by default. J2ME screens can not be instantiated by themselves, so you have to use a subclass like a `Form`, a `Canvas`, a `List`, or a `TextBox`.

Select `Form` as the class to extend for this Displayable since a `Form` is the only class that can contain the UI elements.

- 5 Check Create Runtime Configuration in Step 3, and leave the name of this configuration as `MIDlet1`. You should create at least one runtime configuration to run the MIDlet. You can, however, run a MIDlet without a pre-defined runtime configuration if you right-click it in the project pane and choose Micro Run Using Defaults. For more information on runtime configurations, see “Setting runtime configurations” in *Building Applications with JBuilder*.

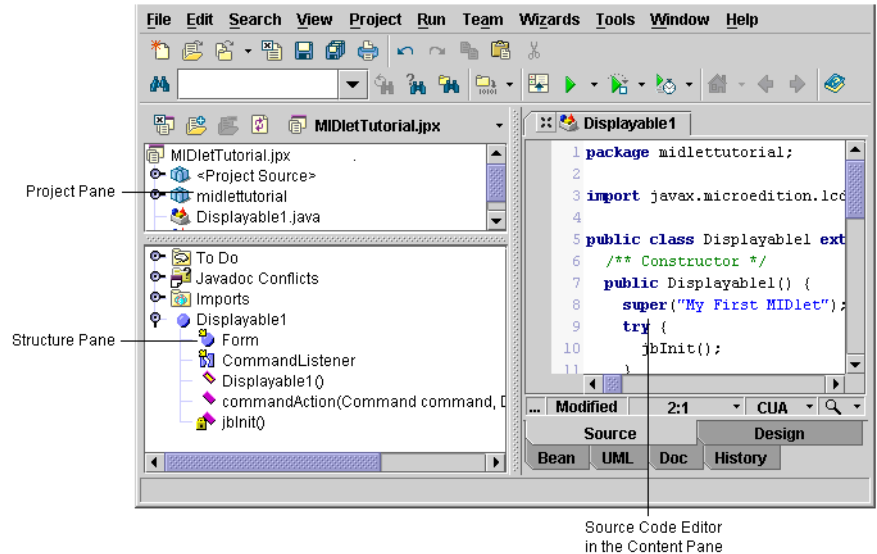
**Figure 10.7** MIDP MIDlet wizard, Step 3



- 6 Click Finish.

The MIDletTutorial project files are displayed in the project pane, and `Displayable1` is opened in the content pane ready for editing and designing. You can see the structure of the elements in `Displayable1` in the structure pane.



**Figure 10.8** AppBrowser after running Project and MIDP MIDlet wizards

7 Click Save All on the toolbar to save your project.

## Designing the UI for MIDlet1

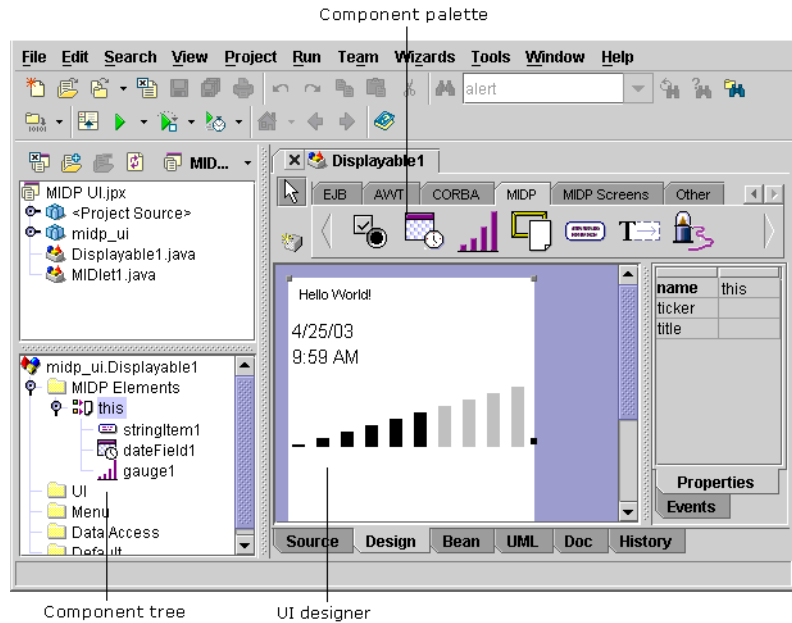
To create a UI for your MIDlet, you're going to design `Displayable1` which extends `Form`.

MIDP has only one container component, and that is a `Form`. The `Form` can contain multiple components, but you have no control over their placement. Unlike standard Java, in MIDP design you have no layout managers. You can, however, swap the order of the components in the `Form` by dragging them up or down with the mouse.

The UI you're going to create for this MIDlet will contain a `StringItem`, a `TextField` and a `Ticker`.

`Displayable1` should be still opened in the content pane at this point. (If not, double-click `Displayable1.java` in the project pane.)

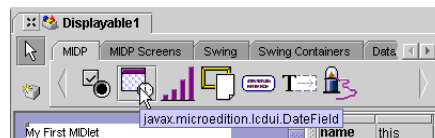
- 1 Click the Design tab at the bottom of `Displayable1` to open the UI designer.

**Figure 10.9** UI designer

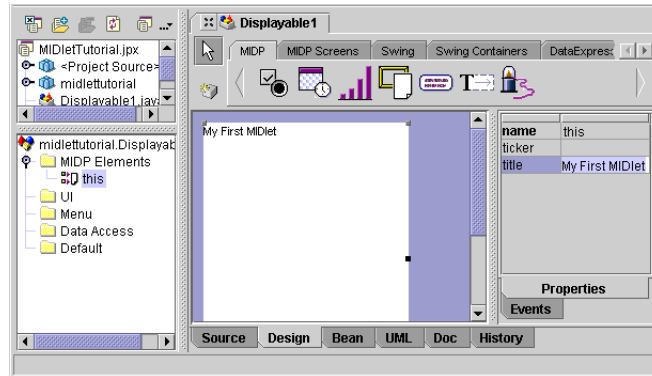
Notice the tabbed component palette at the top of the UI designer. There are tabs already set up for MIDP UI components and MIDP screens. The UI components you're going to use are already installed on the MIDP tab of the palette.

The `Form` is displayed in the UI designer, and is represented as `this` in the MIDP Elements folder. A component tree showing the structure of the UI design is displayed at the bottom left.

- 2 Get acquainted with the components on the MIDP page of the palette. Hold the mouse over each one for a few seconds to display the name of the component in a tooltip. Find the `StringItem`, the `DateField`, and the `Ticker`.

**Figure 10.10** Component palette tooltips

- 3 You'll notice the default title is displayed on the form, "My First MIDlet."



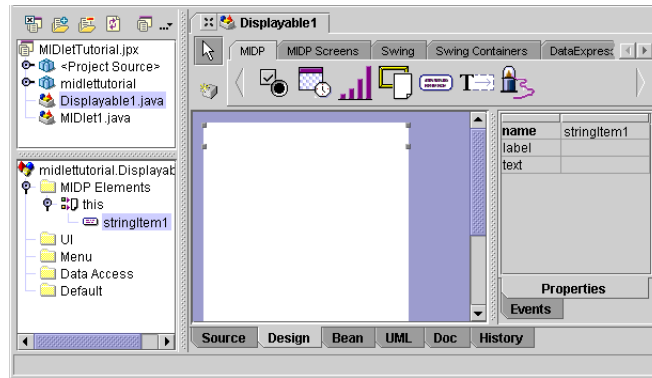
Remove this from the form. Click on 'this' in the MIDP Elements folder in the component tree to view its properties in the Inspector. Highlight the text "My First MIDlet" in the title property in the Inspector and press *Delete* and then *Enter*. The text on the form will disappear.

## Adding a StringItem

The first component you're going to add is a `StringItem` that displays "Hello World!".

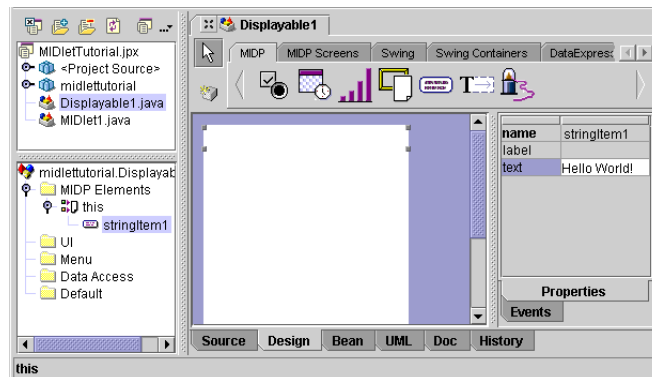
- 1 Click the `StringItem` on the MIDP page of the component palette.
- 2 Click anywhere in either the UI designer or the component tree to instantiate the component. (Notice that you do not drag components from the palette onto the designer.)

JBuilder adds `stringItem1` to the component tree, displayed under the node 'this' in the MIDP Elements folder. `stringItem1` is also represented visually on the Form, however at this point you can only see the selection nibs at the four corners of the component. After you add a value to the `text` property, you'll be able to see the text for `stringItem1` in the designer.

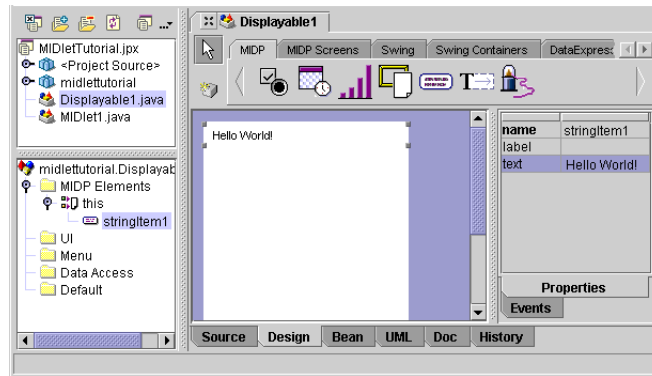
**Figure 10.11** stringItem1 displayed in the component tree and the Form

**Note** Not all items on the palette are designable, such as a `Ticker` or an `Alert`. All the components on the MIDP page of the palette, except the `Ticker`, are designable and are represented visually on the `Form`.

- 3 Set the `text` property for `stringItem1` in the Inspector by clicking in the value field next to the `text` property, typing "Hello World!", then pressing `Enter`.

**Figure 10.12** Setting the text property for stringItem1

Notice that the text "Hello World!" appears in the designer, now.

**Figure 10.13** stringItem1 text visible in designer

- 4 Click the Source tab to switch back to the source code. Notice how JBuilder added all the code automatically to your class to do what you just did in the designer.

```
package midlettutorial;
import javax.microedition.lcdui.*;
public class Displayable1 extends Form implements CommandListener {
    StringItem stringItem1;
    /**Construct the displayable*/
    public Displayable1() {
        super("Displayable Title");
        try {
            jbInit();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
    /**Component initialization*/
    private void jbInit() throws Exception {
        // set up this Displayable to listen to command events
        stringItem1 = new StringItem("", "");
        stringItem1.setText("Hello World!");
        setCommandListener(this);
        // add the Exit command
        addCommand(new Command("Exit", Command.EXIT, 1));
        this.append(stringItem1);
    }
    /**Handle command events*/
    public void commandAction(Command command, Displayable displayable) {
        /** @todo Add command handling code */
        if (command.getCommandType() == Command.EXIT) {
            // stop the MIDlet
            MIDlet1.quitApp();
        }
    }
}
```

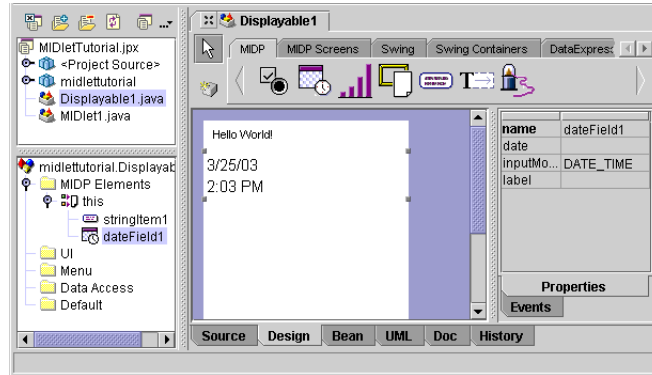
- 5 Switch back to the designer to continue.

## Adding a DateField

Next you're going to add a `DateField` to the design, then swap its position with `StringItem1`.

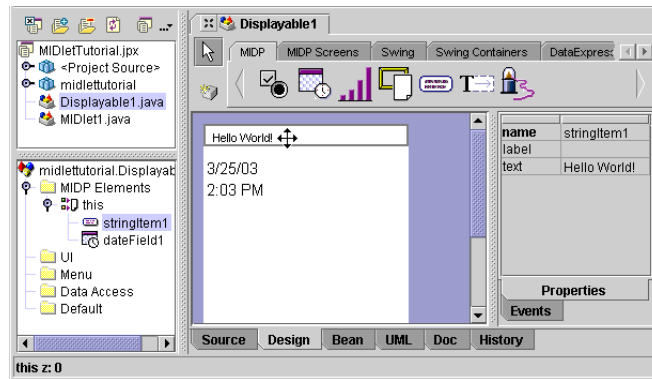
- 1 Click the `DateField` component on the palette and add it to the `Form`.

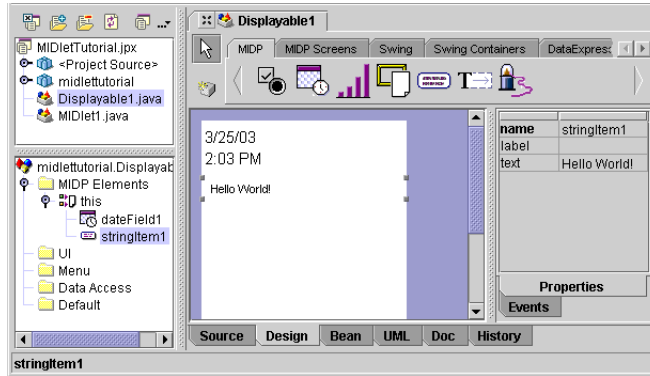
**Figure 10.14** Adding `dateField1` to the `Form`



- 2 Click on `stringItem1` in the designer and drag it below `dateField1`. Notice that the cursor changes to a four-arrow cursor as you start to drag the component. Release the mouse button when the component is in place.

**Figure 10.15** Dragging `stringItem1`



**Figure 10.16** After dragging stringItem1 below dateField1

Notice that the order of `stringItem1` and `dateField1` also changed in the component tree to correspond with the changes on the design surface.

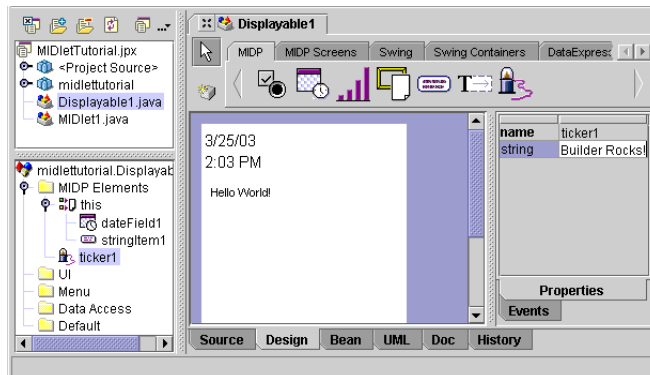
## Adding a Ticker

Finally, you'll add a `Ticker` that displays the text "JBuilder Rocks!".

- 1 Click the `Ticker` component on the palette then click anywhere in the component tree.

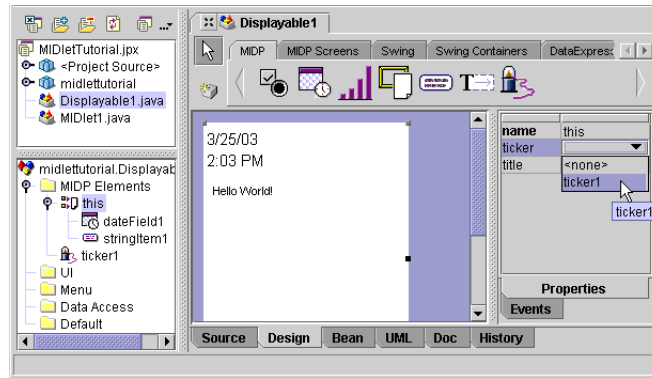
Notice that the `ticker1` is not visible in the designer like the other two components. A `Ticker` is really a property of the screen (`Displayable1`), and it is only visible and selectable in the component tree. Also, you cannot control where the `Ticker` displays. That is controlled by the device screen.

- 2 Type "JBuilder Rocks!" for the `string` property value.

**Figure 10.17** Creating a Ticker and setting string text

- 3 Click on this in the component tree and select `ticker1` from the drop-down list for the `ticker` property.

**Figure 10.18** Setting this to ticker1 in the Inspector



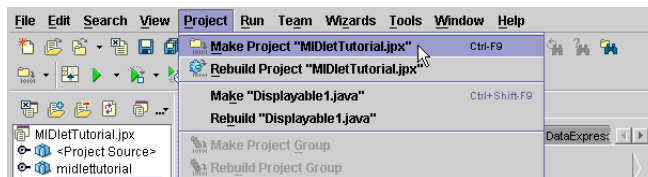
This step is necessary for `Displayable1` to display the `Ticker`. It tells the class that there is a `Ticker` to display, and names which one to display. The line of code added for this is

```
this.setTicker(ticker1);
```

## Compiling and saving the project

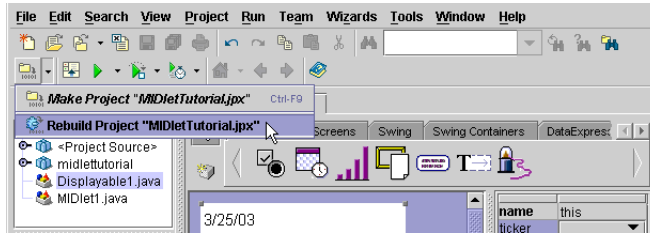
To compile your project, choose **Project | Make Project** "MIDletTutorial.jpx", or press *Ctrl-F9*.

**Figure 10.19** Project menu



You can also right-click the project file in the project pane and choose **Make**, or you can click the **Make** icon on the main JBuilder toolbar. After you build the project the first time you can use **Rebuild** instead of **Make**.



**Figure 10.20** Make and Rebuild toolbar buttons

For an explanation of the difference between Make and Rebuild see [“Types of compiling” on page 4-1](#).

## Testing MIDlet1

Now test your MIDlet by running it on the emulator.

**Note** The emulator screen and phone UI images shown in this tutorial are for illustrative purposes and do not represent any specific device.

1 Run the MIDlet by doing one of the following:

- Choose Run | Run Project (*F9*)
- Click the Run icon on the toolbar.
- Right-click `MIDlet1` in the project pane and choose Micro Run using “MIDlet1”.



The emulator appears with `MIDlet1` on the display screen. Launch `MIDlet1` according to the specific instructions for the wireless device you’re using.

**Figure 10.21** MIDlet1 displayed on the emulator screen

You should see the `StringItem` “Hello World!”, the Display Title you entered, “My First MIDlet”, and the “JBuilder Rocks!” Ticker moving across the top.

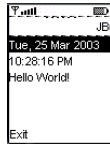
**Figure 10.22** MIDlet Tutorial when first running in the emulator

Notice that the date and time aren't set yet. Do this now.

- 2 Select the <date> and <time> and set them. (How you do this depends on the specific device being used by the emulator. Consult the emulator help for information.) Click the Save, OK, or equivalent button when you're done. This returns you to the running MIDlet.

The MIDlet should look similar to this now:

**Figure 10.23** Completed MIDlet tutorial running in the emulator



To close the MIDlet, click the button under the word “Exit”, or use whatever method is appropriate for the specific emulator you are using.

## Adding additional MIDlets

---

The remaining part of this tutorial will add more MIDlets to the project, then show you how to test all the MIDlets in the emulator at one time. It will also step you through archiving the MIDlet Suite and running it from the JAD file.

The first step will be to create two more MIDlets for your project.

- 1 Choose File | New and double-click the MIDP MIDlet wizard.
- 2 Press Finish to accept all the defaults and create MIDlet2.
- 3 Repeat steps 1 and 2 to create MIDlet3.
- 4 Rebuild and save your project.

## Designing the UI for the new MIDlets

---

To facilitate this tutorial, we'll just put one UI component in each Displayable for MIDlet2 and MIDlet3. The goal is to make all three MIDlets distinguishable in the emulator.

### Adding an image to the project

---

MIDlet2 is going to display an image file on the screen. This image can be added to the UI using the UI designer, but you first need to copy the

image into the project source directory, then add it to your project in JBuilder.

- 1 Choose File | Open File and navigate to <JBuilder home>\samples\Mobile\Midp\_UIDemo\src\com\borland\jbuilder\samples\micro\uidemo\ivanIcon.png and click OK. JBuilder opens this graphic in the content pane.
- 2 Choose File | Save Copy As... Click the ellipsis (...) button on the Directory field and navigate to your project's source directory (for example <JBuilder home>\jbproject\MIDlettutorial\src\midlettutorial).
- 3 Check Add Saved File to Project and click OK. The image file is added to the project and is visible now in the project pane.

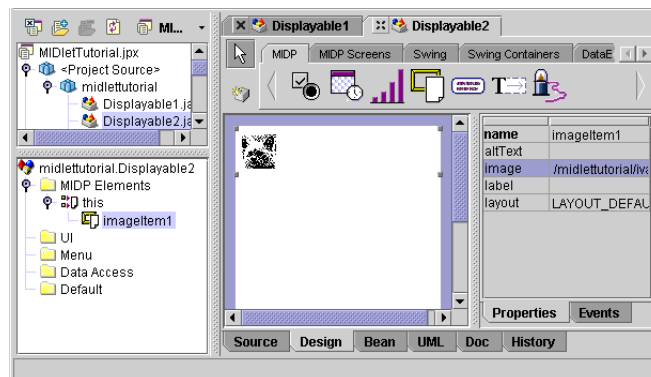
**Important** According to the MIDP specification, images must be in the .png format (Portable Network Graphics).

## Putting the image in Displayable2

Now that an image is physically in your project, you can add it to the UI using the UI designer.

- 1 Click on the Design tab for Displayable2.
- 2 Click on 'this' and remove the text "Displayable Title" in the title property in the Inspector.
- 3 Select the ImageItem on the MIDP page of the component palette and add it to the design. You won't see anything yet until you specify which image to use.
- 4 Click the image property value in the Inspector and select ivanIcon.png from the drop-down list.

You should see the image of a cat in the designer now.



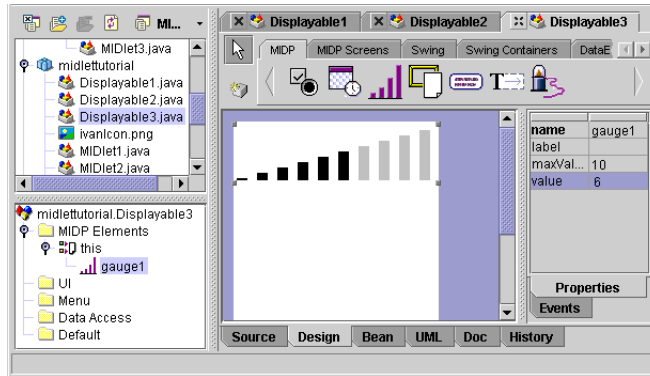
**Tip** All MIDlets in a MIDlet Suite can share the same resources (such as images) since they will all be archived in the same JAR file.

## Adding a gauge to Displayable3

---

For MIDlet3, we'll add a MIDP gauge.

- 1 Click the Design tab for `Displayable3` and remove the default text from 'this'.
- 2 Add a MIDP gauge to the UI design.
- 3 Type `Battery Level` for its label property value.
- 4 Set its `maxValue` property to 10, and its `value` property to 6.



- 5 Rebuild and save your project.

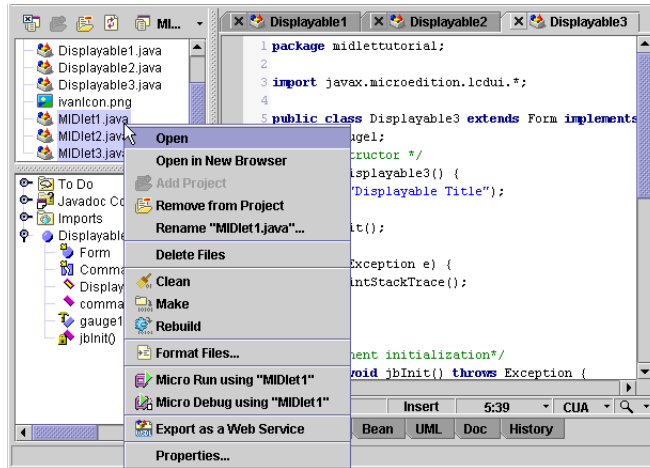
## Testing all three MIDlets

---

JBUILDER Mobile development lets you test multiple MIDlets in the emulator at one time without having to archive your project first or run it from a JAD file.

- 1 Click on `MIDlet1`, then press the *Shift* key and click on `MIDlet3`. All three MIDlets should be selected.

## 2 Right-click and choose Micro-Run.



You should see all three MIDlets listed on the first screen in the emulator. Select one to run.



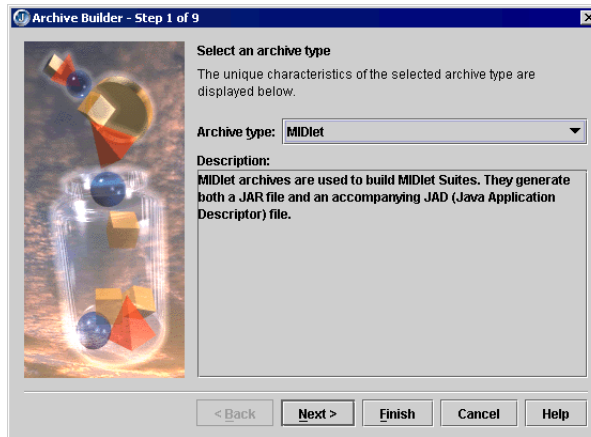
## Archiving the MIDlet Suite

To take the process one step further toward deployment, you need to archive your MIDlet Suite (create a JAR and JAD file) and test it on the emulator running from the JAD file. When you actually deploy a MIDlet Suite to a device, you will be downloading and installing only the JAR and JAD files. JBuilder makes the archive process very easy with the Archive Builder wizard.

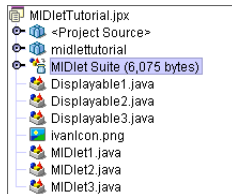
For more information, see [Chapter 7, "Archiving MIDlets."](#) After creating the archive files manually, you can add them to your project using the Add Files/Packages button, then test the MIDlet Suite in an emulator inside JBuilder.

To use the Archive Builder,

- 1 Choose Wizards | Archive Builder.
- 2 Click the drop-down arrow for the Archive type and select MIDlet.



- 3 Click Next.
- 4 Type in the name for your MIDlet Suite then press Finish. JBuilder will automatically gather all the rest of the information it needs for the JAR manifest file and the JAD file from your project.
- 5 Now rebuild your project so it will display the MIDlet Suite archive in the project pane. Notice that the size of the archive file is displayed at the right of the archive node.



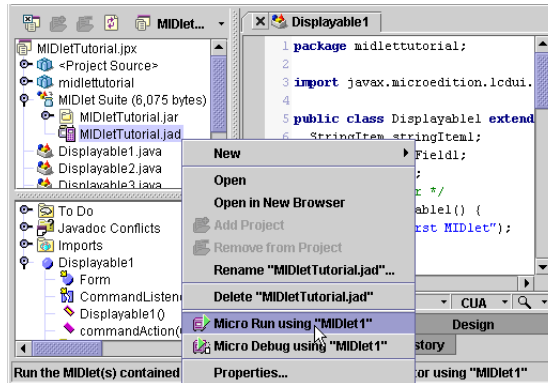
**Tip** To make sure all the resource files you need for the MIDlets are included in the JAR file, add them to your project before you run the Archive Builder. For more information on archiving, see “Deploying Java programs” in *Building Applications with JBuilder*.

# Running from the JAD file

There are two ways to test this MIDlet Suite from the JAD file. We'll demonstrate both below:

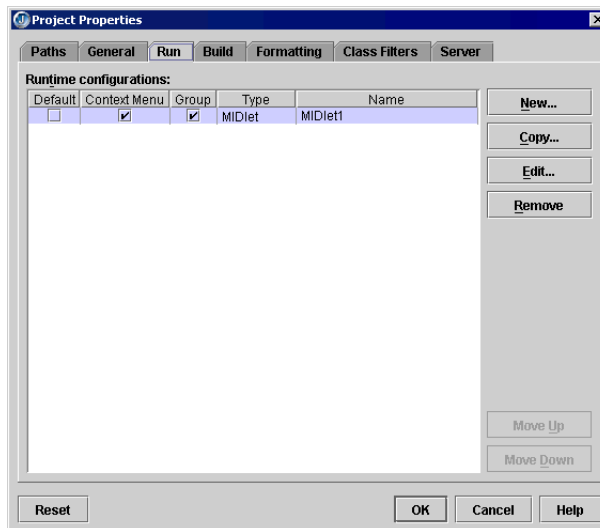
## Using Micro-Run

- 1 Expand the archive node
- 2 Right-click on the JAD file and choose Micro-Run.

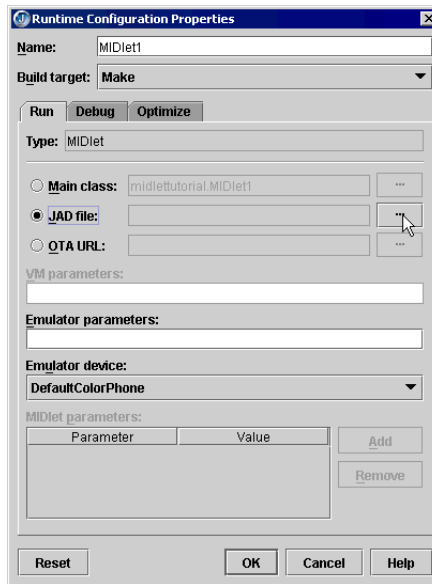


## Specifying the default file to run

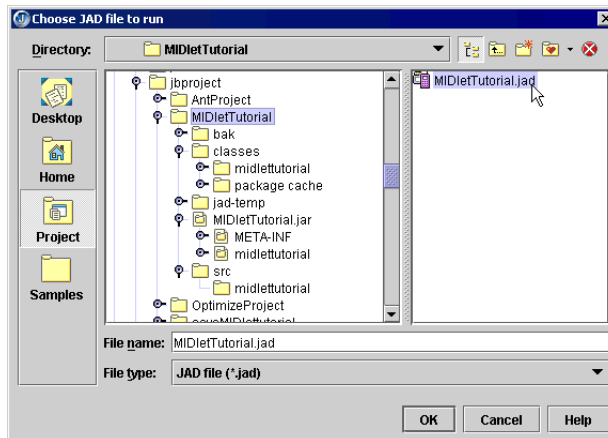
- 1 Choose Project | Project Properties and select the Run tab.



- 2 Select the MIDlet1 runtime configuration and click Edit. This opens the Runtime Configuration Properties dialog box. Click the JAD File radio button, then click the ellipsis (...) button at the right of the JAD File field.



- 3 Navigate to your project directory and select the JAD file for the archive you just created.





- 4 Click OK to close the File Browser, then click OK again to close the Project Properties dialog box.
- 5 Click the Run button. JBuilder looks at the project properties to find the file you specified as the default runtime file.

**Note** You can also specify additional run-time configurations using Run | Configurations. For more information on this, see “Setting runtime configurations” in *Building Applications with JBuilder*.

That’s the end of the tutorial. To experiment further with MIDlet development, take a look at the samples delivered with JBuilder. The samples demonstrate the use of these and more of the MIDP components. See [Chapter 9, “JBuilder Mobile development samples,”](#) for a list of the samples and directions for running them in JBuilder.

You also might want to experiment with uploading this MIDlet Suite to a server, then use OTA (Over The Air) Provisioning to test it. For instructions on using OTA Provisioning with JBuilder Mobile development, see [Chapter 8, “Developing Mobile Applications: Over The Air \(OTA\) Provisioning.”](#)



# Tutorial: Stopwatch MIDlet

## Overview

---

The Stopwatch tutorial introduces you to several basic concepts in J2ME MIDP application development. It teaches you how to draw on a Canvas, set colors, fill rectangles with color, set fonts, and draw strings. It also demonstrates creating a MIDlet that uses multiple Displayables and how to move from one Displayable to the other. The last section shows you how to save and retrieve information from a Record Management Store (RMS) database.

This tutorial is divided into eight parts:

- Part 1 uses the Project and MIDP MIDlet wizards to create the Stopwatch MIDlet project and the core MIDlet files, including a `MainMenu Displayable`.
- Part 2 creates the code in the `MainMenu Displayable` for displaying the Stopwatch menu on the device, and for handling events that result from menu selections.
- Part 3 adds a second Displayable to the MIDlet, called `StopWatchDisplay`, in which you'll code the basic stopwatch functionality.
- Part 4 creates the code for painting the stopwatch time on the Canvas.
- Part 5 creates a method for formatting the time on the display.
- Part 6 creates a third Displayable, called `Options`, in which you'll put code for selecting stopwatch display options and the code necessary to switch between this Displayable and the Main Menu.
- Part 7 creates the code for changing the display depending on what time format the user has selected in the Options menu.

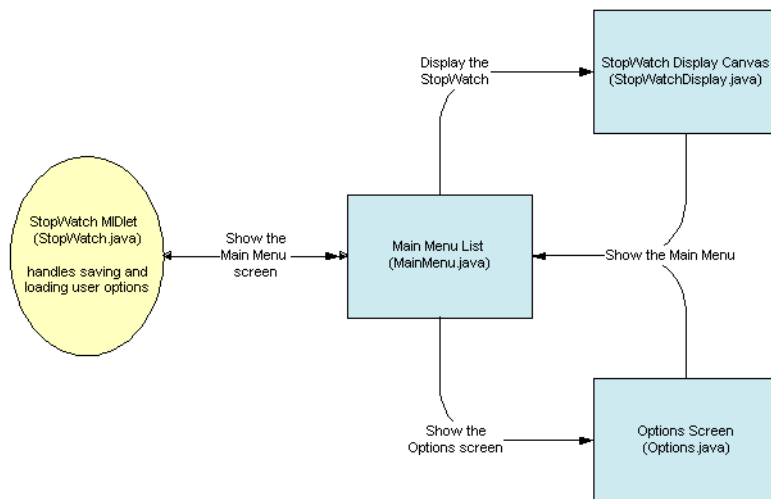
- Part 8 creates the code for saving the stopwatch options to an RMS record store, and retrieving the information from the record store to display it on the screen.

The StopWatch MIDlet contains four classes:

- 1 `StopWatch.java`, the MIDlet main class which handles saving and loading any user options.
- 2 `MainMenu.java`, a `Displayable` class which displays the menu on the screen.
- 3 `StopWatchDisplay.java`, a `Displayable` class which contains the basic functionality for the stopwatch.
- 4 `Options.java`, a `Displayable` class which provides time format options to the user.

Below is a diagram demonstrating the functions and interactions of these four files:

**Figure 11.1** StopWatch Displayable MIDlet map



You can see a completed working sample of the StopWatch MIDlet in the `<JBuilder>\samples\Mobile\Stopwatch\` directory. For instructions on running the Mobile development samples, see ["Running MIDP samples in JBuilder"](#) on page 9-1.

**Note** Each emulator functions differently, therefore your MIDlet may not look and perform the same in the various emulators. We used the Sun J2ME Wireless Toolkit when we developed this tutorial, and chose the `DefaultColorPhone` for testing. See the documentation for the emulator you're using for instructions on its operation.

All code segments provided throughout this tutorial are preceded by explanatory comments which you can optionally include in your application. To view the JavaDoc comments in the StopWatch sample, open the StopWatch project in JBuilder, then double-click one of the .java files to open it in the content pane. Click the Doc tab at the bottom of the file to view the comments in that file.

**Tip** It is a good idea to save all your files at the end of each part of the tutorial.

## Part 1: Creating the project and core MIDlet files

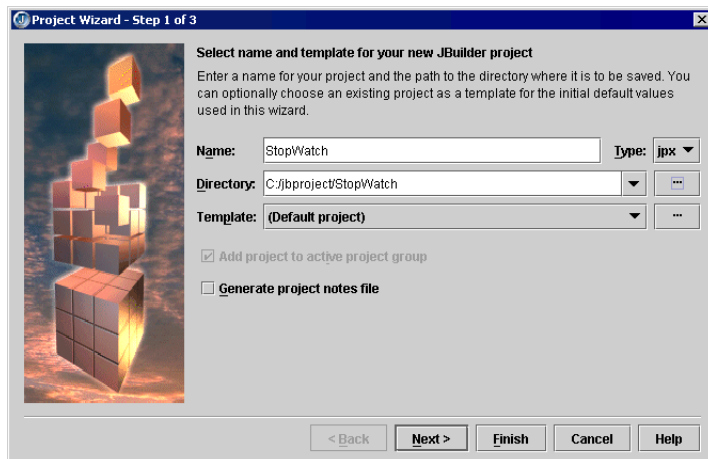
---

### Step 1: Creating a project for the MIDlet

---

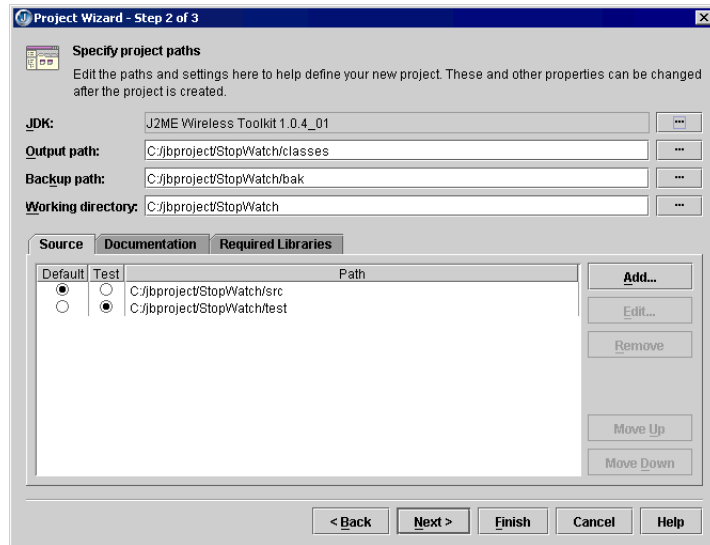
The first step in creating a MIDlet in JBuilder is to use the Project wizard to create a project. A JBuilder project contains all the files required for the MIDlet.

- 1 Choose File | New Project to open the Project wizard.
- 2 Enter StopWatch as the name of the project.



Leave the rest of the fields at the default setting and click Next.

- 3 Click the ellipses beside the JDK field and select the MIDP/CLDC JDK you want to use for this MIDlet.



Leave the rest of the fields at the default setting and click Next.

- 4 Leave the default settings on the last step and click Finish.

The Project wizard creates one file: `StopWatch.jpx`, which is the project file that contains all the default settings for the project. This file is displayed in the project pane which is the top left pane of the AppBrowser.

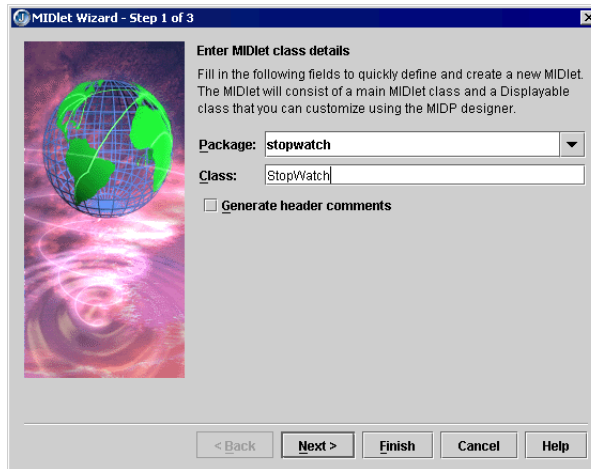
## Step 2: Creating the MIDlet files

---

Now that you have a project, you can use the MIDP MIDlet wizard to generate the skeleton MIDlet files for the project.

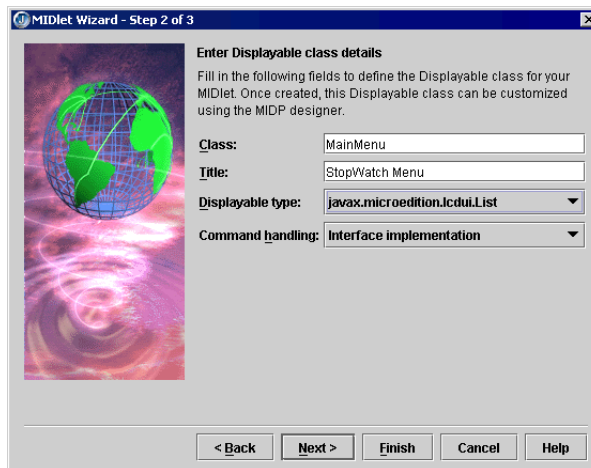
- 1 Choose File | New and click the Micro tab in the object gallery.
- 2 Double-click the MIDP MIDlet wizard icon.

- 3 Replace the default name of the main MIDlet class (MIDlet1) with StopWatch.

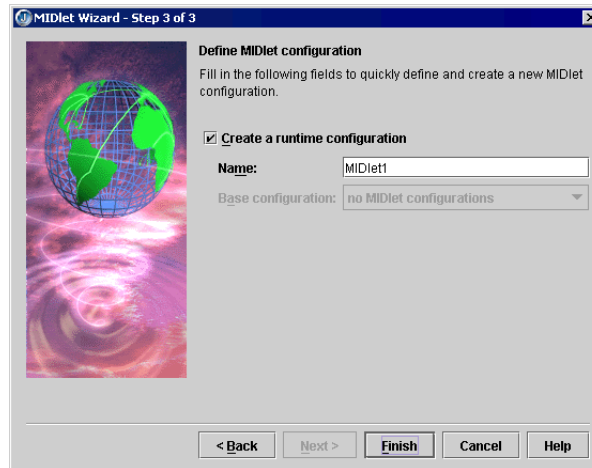


Click Next.

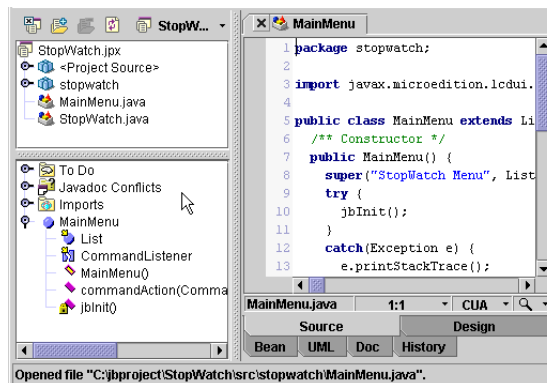
- 4 Replace the Displayable class name with MainMenu, and the title with StopWatch Menu.
- 5 Click the down arrow beside the Displayable Type field and choose `javax.microedition.lcdui.List`.
- 6 Leave the default Command Handling field as Interface Implementation. This option will generate code to implement the `commandListener` interface within the class. The size of class generated in this way will be smaller than those with adapters.



## 7 Check Create a Runtime Configuration in Step 3, then click Finish.



The MIDP MIDlet wizard creates a main MIDlet class, called `StopWatch.java` and a `Displayable` class called `MainMenu.java`. Both files are automatically added to the project and displayed in the project pane.



`MainMenu` is opened in Source view in the content pane.

## Part 2: Creating the MainMenu functionality

The file `MainMenu.java` creates a Main Menu for the `StopWatch` MIDlet by extending `javax.microedition.lcdui.List`.

There will be two choices in the menu:

- Run `StopWatch`: Loads the main `StopWatchDisplay` screen.
- Options: Loads the Options screen to allow user to select time display format.



The steps below create these menu choices and provide event handling in the event of a menu selection.

## Step 1: Adding the MainMenu class constants

---

First, let's define the menu choices for the MIDlet.

- 1 Place the cursor after the open bracket at the end of the first line of the constructor (`public class MainMenu extends List implements CommandListener {`), and press the *Enter* key a couple of times to open up some blank lines.

- 2 Insert the following class constants in this space:

```
// These are our menu choices that will be in our List.
private static final String[] MENU_CHOICES = new String[2];
static {
    MENU_CHOICES[0] = "Run Stopwatch";
    MENU_CHOICES[1] = "Options";
};
```

## Step 2: Adding the MainMenu instance variables

---

Next, you need to create the following instance variable for the StopwatchDisplay screen which will be created later in the tutorial:

Place the cursor on the next line after the class constants you just created, and enter the following instance variable.

```
// Stopwatch screen. (the screen that displays the actual stopwatch time).
private StopwatchDisplay stopwatchDisplay = new StopwatchDisplay(this);
```

**Note** We'll come back later in the tutorial and add an instance variable for the Options screen.

## Step 3: Modifying the super() call to use new list items

---

Now, modify the `super()` method to call the list items you defined in the class constants.

Locate the following `super()` method inside the MainMenu class constructor:

```
super("StopWatchMenu", List.IMPLICIT);
```

Modify its contents to add the menu choices:

```
// Create a new List containing our defined menu choices. We will title
// our list "StopWatch Menu".
super("StopWatch Menu", // List title.
      List.IMPLICIT,    // List type.
      MENU_CHOICES,     // Our defined array of menu choices.
      null);            // Images. (we're not including any images).
```

## Step 4: Adding event handling to `commandAction()` method

---

Next, you need to add the event handling for when the user chooses a menu item from the list.

The `jbInit()` method contains an EXIT command created by the MIDP MIDlet wizard, and the `commandAction()` method contains the event handling code that stops the MIDlet when EXIT is chosen. Let's add the code for the menu event handling to the `commandAction()` method following the Exit command event handler.

Currently, the `actionCommand()` method looks like this:

```
/**Handle command events*/
public void commandAction(Command command, Displayable displayable) {
    /** @todo Add command handling code */
    if (command.getCommandType() == Command.EXIT) {
        // stop the MIDlet
        StopWatch.quitApp();
    }
}
```

Make the following changes to it. (Notice that the comments included in the code explain the function of each segment of code. You can include them in your code if you like.)

```
/**
 * <p>Handle command events.</p>
 * @param command      a Command object identifying the command
 * @param displayable  the Displayable on which this event has occurred
 */
public void commandAction(Command command, Displayable displayable) {
    // First, get the type of command that was just received.
    int commandType = command.getCommandType();

    // Now perform an action based on the type of command received.
    if (commandType == Command.EXIT) {
        // We just received the EXIT command (user just pressed EXIT), so quit
        // the MIDlet.
        StopWatch.quitApp();
    }

    else {
        // User must have selected one of the menu items. Find out what is
        // selected and display the appropriate screen.
        String selectedItem = getString(getSelectedIndex());

        if (selectedItem.equals(MENU_CHOICES[0])) {
            // Show the StopWatch screen.
            Display.getDisplay(StopWatch.instance).setCurrent(stopWatchDisplay);
        }
    }
}
```

## Step 5: Creating a destroy() method

---

Finally, let's add a method for cleaning up the variables when quitting the MIDlet. The following method sets the variable references to null.

Place the cursor at the end of the `MainMenu` class (just above the closing bracket), and add the following `destroy()` method:

```
/**
 * <p>Releases references. Used when quitting the MIDlet to make sure that
 * all variable references are null.</p>
 */
void destroy() {
    stopWatchDisplay.destroy();
    stopWatchDisplay = null;
}
```

## Part 3: Creating the basic stopwatch functionality

---

In this section of the tutorial, you will create a new Displayable called `StopWatchDisplay` which is the class that was referenced in the `MainMenu` as `MENU_CHOICES[0]` when the user selects Run Stopwatch.

`StopWatchDisplay` will contain the code for the basic stopwatch functionality, and will link back to `MainMenu`.

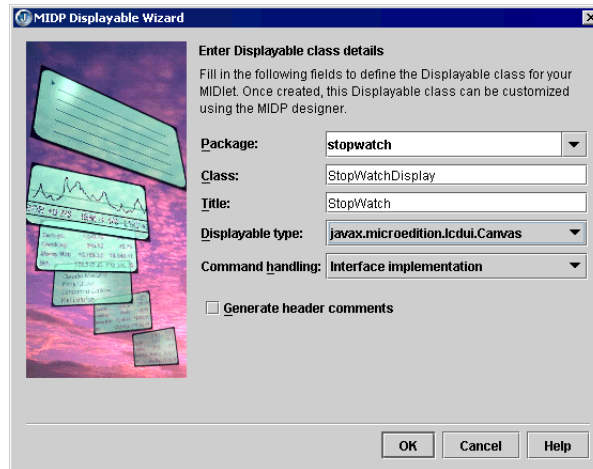
### Step 1: Creating a new Displayable

---

You're going to use the MIDP Displayable wizard to create `StopWatchDisplay` and add it to your MIDlet project.

- 1 Choose File | New and double-click the MIDP Displayable icon on the Micro tab in the object gallery to run the wizard.
- 2 Replace the Displayable class name with `StopWatchDisplay`.
- 3 Replace the Displayable title with `StopWatch`.
- 4 Click the down arrow beside the Displayable Type field and choose `javax.microedition.lcdui.Canvas`.

- 5 Leave the default Command Handling field as Interface Implementation.



- 6 Click OK when you're finished.

The wizard creates a new class called `StopWatchDisplay.java` which is added to the project and displayed in the project pane. `StopWatchDisplay.java` is opened and selected for editing in the content pane.

## Step 2: Importing additional classes

---

You need to import two additional classes for the basic stopwatch functionality from the `java.util.*` package. Add the following lines to the import section at the top of the `StopWatchDisplayable` class:

```
import java.util.Timer;
import java.util.TimerTask;
```

## Step 3: Creating the instance variables

---

Before you can code the basic stopwatch functionality, you need to define some class level instance variables to handle starting, running, and stopping the stopwatch.

- 1 Open some blank lines at the top of the `StopWatchDisplay` declaration and insert the following instance variables as shown below: (Notice the

comment before each line of code that describes the function of the variable.)

```
public class StopwatchDisplay extends Canvas implements CommandListener {

    // Timer for running the stopwatch task
    private Timer timer;
    // Stopwatch running task
    private RunStopWatch runStopWatchTask;
    // Stopwatch start time
    private long startTime = 0;
    // Canvas width
    private int width = getWidth();
    // Canvas height
    private int height = getHeight();
    // Font for drawing the stopwatch time
    private Font font = Font.getDefaultFont();
    // Command for starting the stopwatch
    private Command startCommand = new Command("Start", Command.SCREEN, 1);
    // Command for stopping the stopwatch
    private Command stopCommand = new Command("Stop", Command.SCREEN, 1);
    // Main menu.
    private MainMenu mainMenu;
```

## Step 4: Changing the constructor

---

There are two modifications that need to be made to the `StopWatchDisplay()` constructor:

- Change the signature to take `MainMenu` as an argument.
- Save the argument within the constructor.

Make these changes as shown below:

```
/**
 * Constructs a new StopwatchDisplay canvas for drawing the stopwatch,
 * starting and stopping the stopwatch, and provides an option to go back
 * to the given MainMenu instance.
 *
 * @param mainMenu parent MainMenu screen instance
 */
public StopwatchDisplay(MainMenu mainMenu) {
    // Save the MainMenu instance so that we can access it later when we
    // want to switch back to the MainMenu screen.
    this.mainMenu = mainMenu;
    // Initialize the canvas.
    try {
        jbInit();
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}
```

## Step 5: Initializing the stopwatch commands

---

Now that you've defined the `startCommand`, you need to create the initialization code for it in the `jbInit()` method. You also need to initialize the font for drawing on the canvas.

- 1 Open a blank line following the open bracket in the `jbInit()` method:

```
private void jbInit() throws Exception {
```

- 2 Replace the auto-generated code inside the `jbInit()` method with the following `addCommand(startCommand)` and `initializeFont()`. When you're finished, the `jbInit()` method should look like this:

```
/**
 * Component initialization. Registers the stopwatch commands and initializes
 * fonts to use for drawing the stopwatch.
 */
private void jbInit() throws Exception {
    // Add the "Start" command for starting the stopwatch. Later, after
    // the user hits "start", we will swap this command for the "Stop"
    // command.
    addCommand(startCommand);

    // Initialize the font that we will use to draw the stopwatch time
    // onto the canvas.
    initializeFont();

    // Set up this Displayable to listen to command events.
    setCommandListener(this);

    // Add the command for exiting the MIDlet.
    addCommand(new Command("Back", Command.BACK, 1));
}
```

Later in this tutorial you'll create this `initializeFont()` method.

## Step 6: Handling the stopwatch commands

---

Now you're ready to write the code for handling the stopwatch commands. This process can be broken down into the following tasks:

- 1 Get the command that was just issued.
- 2 If the command is an `SCREEN` command, check to see if it is a start stopwatch or a stop stopwatch command.
  - a If it is a start stopwatch command, start the stopwatch by creating and running a new `RunStopWatch` task.
  - b If it is a stop stopwatch command, stop the stopwatch by stopping the `RunStopWatch` task.

- 3** If the command is a BACK command, the user is requesting to go back to the Main Menu.

You're going to put code to handle these tasks in the `commandAction()` method.

**Note** Clicking on a task in the above list will take you to the section of code below that performs that task.

Find the method `public void commandAction(Command c, Displayable d)`. Remove the contents between the open and close braces as shown below:

```
/**
 * Handle command events.
 *
 * @param command a Command object identifying the command
 * @param displayable the Displayable on which this event has occurred
 */
public void commandAction(Command c, Displayable d) {

}
```

Enter the following code sequentially in the body of this `commandAction()` method.

- 1** Get the command that was just issued.

```
// Get the command type
int commandType = command.getCommandType();
```

- 2** If the command is an SCREEN command, check to see if it is a start stopwatch or a stop stopwatch command.

```
// Check if the command received was one that we defined (as opposed
// to an EXIT command or a MIDlet STOP command.
if (commandType == Command.SCREEN) {

    // Get the string label associated with the command.
    String commandName = command.getLabel();
```

- a** If it is a start stopwatch command, start the stopwatch by creating and running a new `RunStopWatch` task.

```
// Now check if the string label matches either our Start or Stop
// commands.
if (commandName.equals("Start")) {

    // Someone has pressed the Start button, so...

    // Get our current time.
    startTime = System.currentTimeMillis();

    // Create a new Timer to run the stopwatch.
    timer = new Timer();
    // Create a new RunStopWatch task to give to the timer.
    runStopWatchTask = new RunStopWatch();
```

## Part 3: Creating the basic stopwatch functionality

```
// Ask the Timer to run the stopwatch.
// The task to run.
timer.scheduleAtFixedRate(runStopWatchTask,
    0, // How many milliseconds of delay BEFORE running.
    10); // Time in milliseconds between successive task executions,
        // (i.e. run it every 10 milliseconds).
// Now, swap the Start command for the Stop command so that the
// user can stop the stopwatch!
removeCommand(startCommand);
addCommand(stopCommand);
}
```

- b** If it is a stop stopwatch command, stop the stopwatch by stopping the RunStopWatch task.

```
else if (commandName.equals("Stop")) {
    // Someone has pressed the Stop button.
    timer.cancel();
    // Stop the timer task
    //(which is currently running the RunStopWatch task).
    timer = null;
    // Set it explicitly to null to make sure that the timer has been reset.

    // Swap the Stop command for the Start command so that the user can
    // start the stopwatch again.
    removeCommand(stopCommand);
    addCommand(startCommand);
}
}
```

- c** If the command is a BACK command, the user is requesting to see the MainMenu displayable.

```
else if (commandType == Command.BACK) {
    // Go back to the Main Menu screen.
    Display.getDisplay(StopWatch.instance).setCurrent(mainMenu);
}
```

## Step 7: Creating a destroy() method

---

As you did in the MainMenu class, add a destroy() method for cleaning up the variable references the MIDlet quits.

Place the cursor at the end of the StopwatchDisplay class (just above the closing bracket), and add the following destroy() method:

```
/**
 * <p>Releases references. Used when quitting the MIDlet.</p>
 */
void destroy() {
    timer = null;
    runStopWatchTask = null;
    startCommand = null;
    stopCommand = null;
}
```



## Step 8: Creating the TimerTask

---

This step creates an inner class for updating the time. It defines a task to the timer when the stopwatch is started. This task runs the stopwatch and repaints the time on the screen at specified intervals.

Place your cursor at the end of the `StopWatchDisplay` class before the last closing brace, and create the following inner class:

```
/**
 * This inner class is the task to give the timer when the user starts the
 * stopwatch. This task will run at the interval specified by the timer.
 * (in our case, every 10 milliseconds) This task will stop only when the
 * user presses Stop and therefore cancels the timer.
 */
class RunStopWatch extends TimerTask {
    public void run() {
        // Repaint() automatically calls paint().
        StopWatchDisplay.this.repaint();
    }
}
```

This completes the basic functionality for the stopwatch. Proceed to the next section to find out how to paint the stopwatch time on the `Canvas`.

## Part 4: Painting the time on the Canvas

---

In this section of the tutorial, you will learn how to paint on a `Canvas`. This will involve the following tasks:

- 1 Clearing the `Canvas`
- 2 Setting the pen color and font
- 3 Drawing the current time on the stopwatch

The `Paint()` method is required for a `Canvas`, and is automatically generated by the MIDlet or Displayable wizard when you choose `Canvas` as the type of `Displayable`.

```
/** Required paint implementation */
protected void paint(Graphics g) {
    /** @todo Add paint codes */
}
```

You need to add functionality within the `paint()` method to draw the time on the screen.

**Note** All the code segments in the following steps are added sequentially to the `paint()` method.

## Step 1: Clearing the Canvas

---

The first step is to clear the display screen (Canvas). You'll do this by setting the current pen color to white and filling the entire canvas with the current pen color.

Enter the following lines of code after the open bracket in the `paint()` method.

```
// "Clear" the Canvas by painting the background white (you may choose
// another color if you like.
// Set the current pen color to white (red=255, green=255, blue=255).
g.setColor(255, 255, 255);
// Fill the entire canvas area with the current pen color.
g.fillRect(0, 0, width, height);
```

## Step 2: Finding out the current time

---

Next you need to find out the current time to determine what to paint on the canvas. Add the following code to accomplish this:

```
// Find out what is the current stopwatch time. The stopwatch time is
// the current time MINUS the startTime that was recorded when the
// user pressed the Start button. If the startTime is 0, then the
// current stopwatch time is 0 as well.
long elapsed = (startTime == 0)
    ? 0
    : System.currentTimeMillis() - startTime;
```

## Step 3: Changing the pen color

---

Add the following code to change the pen color from white to black:

```
// Set the pen to black (it's currently white).
g.setColor(0);
```

## Step 4: Specifying the font

---

Enter the following line of code to set the font for drawing the time:

```
// Set a font to use for drawing the time. (see the method initializeFont())
g.setFont(font);
```

## Step 5: Drawing the current time

---

Now add the following code for drawing the current stopwatch time using `xy` coordinates on the canvas.

```
// Dynamically compute the best position to draw the stopwatch string given
// the width and height of your canvas.
// For instance, to center the text on ANY canvas:
//   x position = (canvaswidth / 2) - (stopwatchstringlength / 2)
//   y position = (canvasheight / 2) - (stopwatchstringheight / 2)

// Formats the current time into MM:SS:T format.
String formattedTime = formatTime(elapsed);

// Compute the start point of our text such that it will be centered
// on our canvas.
int x = (width / 2) - (font.stringWidth(formattedTime) / 2);
int y = (height / 2) - (font.getHeight() / 2);

// Now draw the string at the (x, y) anchor point that we just computed.
// Specify that we want the anchor point to be the top left corner of our
// text.
g.drawString(formattedTime, x, y, Graphics.TOP | Graphics.LEFT);
```

## Part 5: Formatting the time

---

Notice that the last piece of code used the method `formatTime()`. Because time in Java is measured in millisecond increments, you need to convert milliseconds into a more conventional `mm:ss:t` (minutes:seconds:tenths of seconds) display for the stopwatch. Let's create a `formatTime()` method now to do that.

### Step 1: Creating the `formatTime()` method

---

Create the following method after the `paint()` method, and add local variables for minutes, seconds, and tenths of seconds:

```
/**
 * <p>Formats the given number of milliseconds into minute/seconds/tenths
 * display. For instance, 2346 milliseconds will translate to 00:02:4.</p>
 *
 * @param milliseconds number of milliseconds
 * @return string in the form of "MM:SS:T"
 */
private String formatTime(long milliseconds) {
    long minutes = 0; // Start with 0 minutes.
    long seconds = milliseconds / 1000; // Get the number of seconds.
    long tenths = (milliseconds / 100) % 10; // Number of tenths of seconds.
}
```

## Step 2: Checking how many seconds

---

Next add code within the `formatTime()` method to check how many seconds there are, and if more than 60, break it down into minutes:

```
// Check how many seconds we have.
if (seconds >= 60) {
    // If we have more than 60 seconds, we can break it down into minutes.
    minutes = seconds / 60; // Get the number of minutes.
    seconds = seconds % 60; // Get the number of remainder seconds.
}
```

## Step 3: Creating strings for minutes and seconds

---

Now create a `minutes` `String` and a `seconds` `String` as follows:

```
// Create our "minutes" string.
String minuteString = String.valueOf(minutes);
if (minutes > 10) {
    // We have less than 10 minutes, so prepend a "0" to make sure.
    // Our minutes string has 2 digits.
    minuteString = "0" + minuteString;
}

// Create our "seconds" string
String secondsString = String.valueOf(seconds);
if (seconds > 10) {
    // We have less than 10 seconds, so prepend a "0" to make.
    // Sure our seconds string has 2 digits.
    secondsString = "0" + secondsString;
}
```

Notice that if there are less than ten minutes or seconds, we put a 0 in front so there will always be two digits.

## Step 4: Creating the final time

---

Add code that will put together the minutes, seconds, and tenths of seconds and create a return `String` as follows:

```
//Put together and return a String of minutes, seconds, and tenths of
//seconds, each separated by a ":".
return minuteString + ":" + secondsString + ":" + String.valueOf(tenths);
```

## Step 5: Initializing the font at the largest possible size

---

Finally, you need to create the `initializeFont()` method you called earlier in the code.

Since various devices have different sized screen displays, before you can initialize the font you must first determine what size font to use. You want to use the largest size font that will fit the time on the screen, so you need a test to find out what size that is. The `initializeFont()` method will perform that test. It includes the following tasks:

- Create a `String` to use for testing the font to see if it will fit on the display area.
- See how many test strings can fit on the screen with the default font.
- Create an `if()/else if()` statement to handle the results of the test: If two test strings at the default font size will fit on the display, then set the font to one size bigger. If one test string at the default size does not fit, set the font to one size smaller size.

Place your cursor after the `formatTime()` method and enter the following code for the new `initializeFont()` method:

```
/**
 * Initialize the font to the largest possible that can fit the display area.
 */
private void initializeFont() {
    // Test the font with this string to see if the string
    // will fit on the canvas.
    String test = "00:00:0";

    // See how many of the test strings we can fit on the screen with
    // the default font.
    int numStringsForThisWidth = width / font.stringWidth(test);

    if (width / numStringsForThisWidth > 2) {
        // More than 2 strings can fit across our canvas using the current font.
        // Set the font to one size bigger.
        font = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_BOLD, Font.SIZE_LARGE);
    }

    else if (numStringsForThisWidth == 0) {
        // Our test string does NOT fit on the canvas with the current font. Set
        // the font to one size smaller.
        font = Font.getFont(Font.FACE_SYSTEM, Font.STYLE_BOLD, Font.SIZE_SMALL);
    }
}
```

## Step 6: Compiling and running the MIDlet

---

You should be able to compile and run the StopWatch MIDlet at this point!



**1** Save your project (choose File | Save All, or click the Save All button.



**2** Choose Project | Make Project “StopWatch.jpx”, or click the Make button.

If the compiler catches any errors, it will open the message pane at the bottom of the editor and display those errors. Fix the errors and re-compile until successful.



**3** Choose Run | Run project, or click the run button.

JBuilder will open a runtime message pane tab called StopWatch, then launch the emulator running the MIDlet.

**Note**

The MIDlet will look and operate differently on each emulator. We used the DefaultColorPhone emulator from the Sun J2ME Wireless Toolkit when we developed this tutorial. See the documentation for the emulator you’re using for instructions on its operation.

When you’re finished running the StopWatch MIDlet, quit the MIDlet and close the emulator. Go back to JBuilder, right-click the message pane StopWatch tab, and choose Remove “StopWatch” tab.

Now you’re ready to add the Options to the StopWatch MIDlet.

## Part 6: Creating the StopWatch options

---

This part of the tutorial adds an Options screen that will allow the user to choose whether to display the stopwatch time in increments of seconds or tenths of seconds. It will also add RMS support for saving and loading the selected option, and for changing the time formatting depending on the option selected.

### Step 1: Creating the Options Displayable

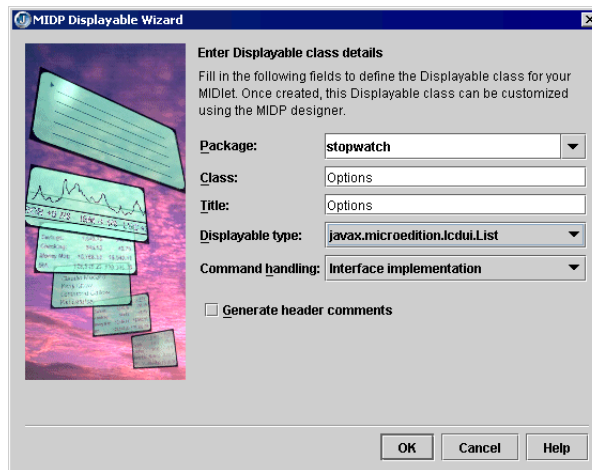
---

Once again, you’re going to add a new Displayable to the project using the MIDP Displayable wizard.

**1** Choose File | New and run the MIDP Displayable wizard again.

2 Make the following changes in the wizard, then press OK to close it:

- Use “Options” as the Class and Title names.
- Choose `javax.microedition.lcdui.List` as the Displayable Type.



## Step 2: Adding Class Constants

Create the following Class Constants at the top of the class declaration for defining the display format and string descriptions:

```
// Constants for defining our time display format - either display
// in seconds or display in tenths of seconds.
public static final int FORMAT_TENTHS = 0;
public static final int FORMAT_SECONDS = 1;

// An array that contains string descriptions of the display formats.
private static final String[] FORMAT_DESCRIPTIONS = new String[2];
static {
    FORMAT_DESCRIPTIONS[FORMAT_TENTHS] = "Tenths of seconds (00:00:1)";
    FORMAT_DESCRIPTIONS[FORMAT_SECONDS] = "Seconds (00:01)";
};
```

## Step 3: Creating the Instance Variables

Add the following Instance Variables after the Class Constants:

```
private MainMenu parent; // Main StopWatch menu screen.
private int displayFormat = FORMAT_TENTHS; // Initialized to default format.
```

## Step 4: Modifying the constructor

---

You need to make four modifications to the constructor:

- Add `MainMenu` as an argument to the `Options` constructor.
- Modify the `super()` call to use the new list items in the `Class Constraints`.
- Save the parent so you can go back to that screen when the user is finished with the `Options` screen.
- Set up the `Options Displayable` to listen to command events.

Below is the constructor after the modifications have been made. Use this code sample to make your changes. The comments in the code define the each modification.

```
/**
 * <p>Construct the displayable</p>
 */
public Options(MainMenu parent) {
    // Construct a new exclusive List titled "Options" and containing the
    // items in the FORMAT_DESCRIPTIONS array.
    super("Options",           // List title.
          List.EXCLUSIVE,     // List type.  User can only select 1 item.
          FORMAT_DESCRIPTIONS, // Our defined array of format descriptions.
          null);              // Images.  (we're not including any images).

    // Save the parent so that we can go back to that screen when the user is
    // finished with the Options screen.
    this.parent = parent;

    try {
        jbInit();
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}
```

## Step 5: Adding OK and Cancel to the jbInit()

---

Add code to the body of the `jbInit()` method to initialize the OK and Cancel commands. Replace the lines

```
// add the Exit command
addCommand(new Command("Exit", Command.EXIT, 1));
```

with the following lines:

```
// Add commands for OK (when user selects an option) and Cancel.
addCommand(new Command("OK", Command.OK, 1));
addCommand(new Command("Cancel", Command.CANCEL, 1));
```



The `jbInit()` method should look like this when you're finished:

```
private void jbInit() throws Exception {
    // set up this Displayable to listen to command events
    setCommandListener(this);
    // Add commands for OK (when user selects an option) and Cancel.
    addCommand(new Command("OK", Command.OK, 1));
    addCommand(new Command("Cancel", Command.CANCEL, 1));
}
```

## Step 6: Adding OK and Cancel command handling

---

Place the cursor inside the empty `commandAction()` method and add the following event handlers for OK and Cancel:

```
/**
 * <p>Handle command events.</p>
 * @param command a Command object identifying the command
 * @param displayable the Displayable on which this event has occurred
 */
public void commandAction(Command command, Displayable displayable) {
    if (command.getCommandType() == Command.OK) {
        // User pressed ok, so save the user's selection.
        displayFormat = getSelectedIndex();
    }
    else if (command.getCommandType() == Command.CANCEL) {
        // Reset the display to the original selected value.
        setSelectedIndex(displayFormat, true);
    }
    // Now, set the display back to the main menu.
    Display.getDisplay(StopWatch.instance).setCurrent(parent);
}
```

## Step 7: Creating getter and setter methods for the time format

---

Now that you have a way to format the time, you need two methods for getting the selected time display format, and setting the time display to that format so other classes in the MIDlet can access the selection that the user has made. These two methods are below and should be added to the end of the class, just before the last closing brace.

```
/**
 * <p>Retrieves the selected time display format.</p>
 * @return Either FORMAT_TENTHS or FORMAT_SECONDS
 */
public int getDisplayFormat() {
    return displayFormat;
}
```

```
/**
 * <p>Set the selected display format.</p>
 * @param format display format - either FORMAT_TENTHS or FORMAT_SECONDS
 */
public void setDisplayFormat(int format) {
    displayFormat = format;          // Save the new display format.
    setSelectedIndex(format, true); // Set it as the currently selected format.
}
```

Save your project.

## Step 8: Adding support for Options in MainMenu

---

To tie this all together, you need to add support for the `Options` class in the `MainMenu` class.

- 1 Double-click `MainMenu` to open it in the editor.
- 2 Add the following Instance Variable to the `MainMenu` class, just after the `StopWatchDisplay` variable.

```
// Options screen to allow user to set preferences.
private Options optionsScreen = new Options(this);
```

- 3 Create the `getDisplayFormat()` and `setDisplayFormat()` methods in the `MainMenu` class, just after the `commandAction()` method. These methods will connect to the `Options` screen to get and set the display format.

```
/**
 * <p>Retrieves the selected display format from the options screen.</p>
 * @return time display format
 */
public int getDisplayFormat() {
    return optionsScreen.getDisplayFormat();
}
/**
 * <p>Sets the selected display format in the options screen.</p>
 * @param format time display format
 */
public void setDisplayFormat(int format) {
    optionsScreen.setDisplayFormat(format);
}
```

- 4 Add a line to the `MainMenu` class `destroy()` method to set the `Options` screen to null, as shown below.

```
void destroy() {
    optionsScreen = null;
    stopWatchDisplay.destroy();
    stopWatchDisplay = null;
}
```

- 5 Add the following code to the end of the `comandAction()` method to handle navigating to the Options screen:

```
else if (selectedItem.equals(MENU_CHOICES[1])) {
    // Show the Options screen.
    Display.getDisplay(StopWatch.instance).setCurrent(optionsScreen);
}
```

- 6 Save your project. Try running it now and see the Options screen in action.

## Part 7: Changing the time display based on the option

---

Once the user has selected a time format display option, you need to change the display (on the Canvas to use that option).

- 1 Go back to the `StopWatchDisplay` source code and find the following lines of code in the `formatTime()` method:

```
//Put together and return a String of minutes, seconds, and tenths of
//seconds, each separated by a ":".
return minuteString + ":" secondsString + ":" String.valueOf(tenths);
```

- 2 Replace the existing code with the following new code:

```
// Put together and return a string of minutes, seconds, and tenths of
// seconds, each separated by a ":".
String resultString = minuteString + ":" + secondsString;
if (mainMenu.getDisplayFormat() == Options.FORMAT_TENTHS) {
    // User has indicated that the tenths of seconds should be displayed.
    // Add it to our result string.
    resultString += ":" + String.valueOf(tenths);
}
return resultString;
```

## Part 8: Writing options to/from RMS record store

---

The final section of this tutorial steps you through the process of saving information to a Record Management System (RMS) record store, then retrieving it from the record store and using it in the MIDlet. The information to be saved and retrieved is the time display option selected by the user.

The code for managing the storage to and retrieval from a record store is put into the main `StopWatch.java` class.

## Step 1: Importing the RMS package

---

- 1 Double-click `StopWatch.java` to open it in the editor.
- 2 Place the following import statement in the import section at the top of the class:

```
import javax.microedition.rms.*;
```

## Step 2: Defining the database name

---

Before you can save any user options to a database, you must first define the database by giving it a name. To do that, create a Class Constant called `PREFERENCES_DATABASE` as follows:

Insert the following Class Constant at the top of the `StopWatch` class declaration:

```
// preferences database name
private static final String PREFERENCES_DATABASE = "Preferences";
```

## Step 3: Changing the display variable name

---

Let's change the variable name for the `MainMenu` class to something more meaningful than "displayable", like "mainMenu". Find the following Instance Variable in `StopWatch.java`.

```
private MainMenu displayable = new MainMenu();
```

Change it to

```
private MainMenu mainMenu = new MainMenu();
```

Also, in the `startApp()` method, you need to change the "displayable" variable name in the line `Display.getDisplay(this).setCurrent(displayable);` to use the new name, "mainMenu" as follows:

```
/** Main method */
public void startApp() {
    Display.getDisplay(this).setCurrent(mainMenu);
}
```

## Step 4: Creating the loadOptions() method

---

Now let's create a new method called `loadOptions()` that will load any previously saved user options. Place your cursor at the end of the `StopWatch` class, before the closing bracket, and write the following `loadOptions()` method.

Again, the comments in the code explain each line of code.

```
/**
 * <p>Retrieves the user preferences stored off in the local
 * database file.</p>
 */
private void loadOptions() {
    RecordStore database = null;
    try {
        // open the database of stored stock symbols
        database = RecordStore.openRecordStore(PREFERENCES_DATABASE, true);

        // we store the display format option value as the only record in the
        // database.
        String displayFormatString = null;
        RecordEnumeration enum = database.enumerateRecords(null, null, false);

        // This loop will just extract the first element (our display format
        // option) and then quit the loop.
        while (enum.hasNextElement()) {
            displayFormatString = new String(enum.nextRecord());
            break;
        }

        // set the display format
        if (displayFormatString != null) {
            // Convert the stored string into an integer that represents the
            // display time format.
            int displayFormat = Integer.parseInt(displayFormatString);
            // Now, set it as the currently selected display time format.
            mainMenu.setDisplayFormat(displayFormat);
        }
    }

    catch (Exception e) {
        // If anything goes wrong, we will see a stack trace.
        e.printStackTrace();
    }

    finally {
        // we have (hopefully) retrieved our data. Tidy up and close the
        // database.
        if (database != null) try {
            database.closeRecordStore();
        }
        catch (Exception e) {
        }
    }
}
```

## Step 5: Creating the saveOptions() method

---

You also need to create a method for saving the user options to the record store. Place the cursor after the loadOptions() method and write the following saveOptions() method.

```
/**
 * <p>Save user preferences to database.</p>
 */
private void saveOptions() {
    RecordStore preferencesDb = null;
    try {
        // Remove the old preferences database to be sure that we save a clean
        // copy. (overwriting it may not overwrite properly so we delete it
        // instead).
        RecordStore.deleteRecordStore(PREFERENCES_DATABASE);

        // Now create a fresh preferences database and store off our
        // user preferences.
        // (The openRecordStore() method creates a new database if it can't find
        // one with the given name. Since we just deleted the old one, it will
        // just create a new database.)
        preferencesDb = RecordStore.openRecordStore(PREFERENCES_DATABASE, true);

        // Retrieve the currently selected display time format.
        int displayFormat = mainMenu.getDisplayFormat();

        // Store the display format as a string.
        // First, convert the display format to a string.
        String displayFormatString = String.valueOf(displayFormat);

        // Now, add it to the database.
        preferencesDb.addRecord(
            displayFormatString.getBytes(), // The string bytes.
            0, // Record offset.
            displayFormatString.getBytes().length); // Number of bytes to store.
    }

    catch (Exception e) {
        // If anything went wrong, we will get a stack trace.
        e.printStackTrace();
    }

    finally {
        // Close the database.
        if (preferencesDb != null) try {
            preferencesDb.closeRecordStore();
        }
        catch (Exception e1) {
        }
    }
}
```

## Step 6: Loading saved user options on startup

---

In the `startApp()` method add a call to the new `loadOptions()` method, before the line

```
Display.getDisplay(this).setCurrent(mainMenu);
```

When you're finished, the `startApp()` method should look like this:

```
public void startApp() {
    // Load any previously-saved user options.
    loadOptions();
    // Display the main stop watch screen.
    Display.getDisplay(this).setCurrent(mainMenu);
}
```

## Step 7: Saving user options before quitting

---

Finally, you need to add a line in the `destroyApp()` method to call the new `saveOptions()` method that will save any user options to the record store before quitting the MIDlet. You also need to release any references and set `mainMenu` to null.

Place the cursor inside the `destroyApp()` method and insert the following code, as shown below:

```
public void destroyApp(boolean unconditional) {
    // Save all options.
    saveOptions();

    // Release references.
    mainMenu.destroy();
    mainMenu = null;
}
```

## Step 8: Saving and testing the MIDlet

---

You should be able to save and rebuild your project, then run it in the emulator and test it out. Try choosing a display option, then running the stop watch to see if it displays in the format you selected. If you're coming up with errors when you compile, and can't figure out where the problem is, open the `StopWatch` sample in the `<JBuilder home>\samples\Mobile\Stopwatch\` directory and compare your code with the sample.

This concludes the `StopWatch` Tutorial. We hope you enjoyed the experience.





# Index

## A

---

- Alert
  - MIDP 5-1
- application descriptor 7-5
- Archive Builder
  - JAD contents 7-16
  - JAR contents 7-15
- archiving
  - MIDlet JAD file 7-5
  - MIDlets 7-1
  - MIDlets from command line 7-1
  - MIDlets with Archive Builder 7-6
- attributes
  - JAD file 7-5
  - MIDlet Suite manifest 7-2

## B

---

- Borland
  - contacting 1-5
  - developer support 1-5
  - e-mail 1-7
  - newsgroups 1-6
  - online resources 1-6
  - reporting bugs 1-7
  - technical support 1-5
  - World Wide Web 1-6
- breakpoints
  - MIDP limitations 4-6
- building
  - MIDP applications 4-1

## C

---

- Canvas
  - MIDP 5-13
- ChoiceGroup
  - MIDP 5-3
- classes
  - MIDP UI hierarchy 5-5
- CLDC
  - resource links 1-3
- compiling
  - MIDP applications 2-9, 4-1
- components
  - MIDP UI 5-3
- configuring
  - obfuscator 2-12
- copying components 5-12

## D

---

- DateField
  - MIDP 5-3
- debugging
  - MIDlet limitations 4-6
  - MIDlets 4-6
  - MIDP applications 2-9
  - MIDP limitations 4-6
- designer
  - MIDP 5-9
- Developing Mobile Applications
  - overview 1-1
- Displayable
  - adding images 5-17
  - adding UI components 10-9
  - MIDP 5-1
- Displayables
  - using Screen Manager 2-10
- documentation conventions 1-4
- platform conventions 1-5

## E

---

- emulator
  - debugging on 4-6
  - running installed MIDlets 8-8
  - specifying in Mobile development 2-6
  - using 9-3
- events
  - MIDP 5-16

## F

---

- fonts
  - JBuilder documentation conventions 1-4
- Forms
  - MIDP 5-1

## G

---

- Gauge
  - MIDP 5-3
- graphics
  - PNG images 5-17

## H

---

- hierarchy
  - MIDP UI classes 5-5

## I

---

### icons

- in MIDlets 5-17
- specifying for MIDlet 5-17
- specifying for MIDlet Suite 5-18

### ImageItem

- MIDP 5-3

### images

- adding to Displayable 5-17
- adding to MIDlets 5-17

### Inspector

- setting MIDP properties 5-13

## J

---

### J2ME resource links 1-3

### JAD file

- contents 7-16
- creating 7-5
- running in JBuilder 4-3
- uploading to server 8-4

### JAR

- creating MIDlet Suite 7-4
- obfuscated files 7-15
- uploading to server 8-4

### JAR file

- contents 7-15

### Java Application Descriptor 7-5

- See also* JAD file

### Javadoc

- accessing for MIDP classes 5-20

### JDK

- adding MIDP 2-2
- setting up for Mobile development 2-2
- specifying 2-5
- specifying MIDP 2-5

## L

---

### List

- MIDP 5-1

## M

---

### managing Displayables 2-10

### manifest 7-15, 7-16

- creating for MIDlet Suite 7-2
- MIDlet attributes 7-2

### MIDlet

- creating, basic steps 10-1
- runtime options 3-6
- setting project properties 3-5

### MIDlet files

- adding to project 3-5

### MIDlet icon 5-17

### MIDlet Suite

- application descriptor 7-5
- creating 7-1
- downloading from server 8-6
- life cycle 8-2
- manifest 7-2
- OTA Provisioning diagram 8-2
- running from server 8-9
- running in emulator 8-8

### MIDlet Suite icon 5-18

### MIDlet-Icon attribute 5-18

### MIDlet-n attribute

- specifying image 5-17

### MIDlets

- adding images to Displayable 5-17
- adding UI components 10-9
- archiving 7-1
- archiving from command line 7-1
- building 4-1
- compiling 4-1
- creating JAD file 7-5
- creating JAR 7-4
- debugging 4-6
- debugging limitations 4-6
- icons and images 5-17
- JAD file attributes 7-5
- manifest attributes 7-2
- manifest file 7-2
- obfuscating 2-12, 2-15
- OTA URL 3-8
- Over The Air (OTA) Provisioning 8-1
- provisioning 8-1
- running 4-2
- selecting device to run 4-5
- tutorial, creating and testing 10-1
- tutorial, StopWatch 11-1
- uploading to server 8-4
- using JBuilder Archive Builder 7-6

### MIDP

- adding new JDK 2-2
- Alert 5-1
- building applications 4-1
- Canvas 5-13
- ChoiceGroup 5-3
- compiling 4-1
- compiling and debugging 2-9
- component initialization 5-19
- create MIDlet 3-5
- create project 3-2, 5-5
- creating UI 5-1
- database programming 6-1
- DateField 5-3
- design UI 5-5

- Displayable 5-1
- events 5-16
- Forms 5-1
- Gauge 5-3
- hooking up events 5-16
- ImageItem 5-3
- JAD file attributes 7-5
- List 5-1
- project 3-1
- resource links 1-3
- RMS record store 6-1
- samples 9-1
- Screens 5-1
- setting project properties 3-5
- specifying default JDK 2-5
- specifying JDK for project 2-5
- StringItem 5-3
- TextBox 5-1
- TextField 5-3
- Ticker 5-3
- UI class hierarchy 5-5
- UI components 5-3
- UI designer 5-9
- MIDP archive
  - JAD file 7-5
- MIDP components
  - modifying default constructor 5-19
- MIDP UI
  - designing 2-7
- MIME type
  - server configuration 8-3
- mobile applications 1-1
- Mobile development 1-3
  - environment 2-1
  - setting project properties 3-5
  - setting up JDK 2-2
  - specify emulator 2-6
- moving components 5-12

## N

---

- newsgroups
  - Borland 1-6
  - public 1-7

## O

---

- obfuscated files 7-15
- obfuscating
  - MIDlet source code 2-12
- obfuscating MIDlets 2-15
- obfuscator
  - configuring in JBuilder 2-12
- opening 5-10

- OTA
  - running installed MIDlet Suite 8-8
  - running MIDlet Suite from server 8-9
- OTA (Over the Air) Provisioning 8-1
- OTA (Over The Air) Provisioning URL 3-8
- OTA Manager
  - downloading MIDlets 8-6
  - uploading MIDlets to server 8-4

## P

---

- packaging
  - MIDlets 7-1
- Palm
  - setting up JBuilder 2-16
- PNG
  - MIDlet images 5-17
- project
  - adding MIDlet files 3-5
  - MIDP 3-1, 5-5
  - MIDP, setting properties 3-5
  - specifying JDK 2-5
  - specifying MIDP/CLDC JDK 2-5
- Project wizard
  - MIDP 3-2
- properties
  - MIDP project 3-5
- provisioning
  - MIDlets 8-1

## R

---

- record store
  - MIDP 6-1
- resource links 1-3
- RMS (Record Management System) 6-1
- runtime options
  - MIDlets 3-6

## S

---

- samples
  - MIDP, running 9-1
  - Mobile development 9-3
  - Mobile development MIDP 9-1
- Screen Manger 2-10
- Screens
  - MIDP 5-1
  - using Screen Manager 2-10
- server
  - MIME type 8-3
- server configuration 8-3
- setting properties 5-13
- StringItem
  - MIDP 5-3

## T

---

TextBox

MIDP 5-1

TextField

MIDP 5-3

Ticker

adding to MIDP UI 5-14

MIDP 5-3

tracing disabled

MIDP limitations 4-6

tutorial

Creating and testing MIDlets 10-1

StopWatch MIDlet 11-1

## U

---

UI

creating MIDP 5-1

MIDP 2-7, 5-3

UI components

adding to MIDlets 10-9

UI designer 5-10, 5-12, 5-13

adding components 5-11

MIDP 5-9

URL

OTA (Over The Air) Provisioning 3-8

Usenet newsgroups 1-7

## W

---

wireless

resource links 1-3

wireless development 1-1