



Tutorial

Sun Java™ Studio Mobility 6
2004Q3

Sun Microsystems, Inc.
www.sun.com

Part No. 817-2343-10
July 2004, Revision A

Submit comments about this document at: <http://www.sun.com/hwdocs/feedback>

Copyright © 2004 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

U.S. Government Rights - Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

This distribution may include materials developed by third parties. Sun, Sun Microsystems, the Sun logo and Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon architecture developed by Sun Microsystems, Inc.

UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Products covered by and information contained in this service manual are controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright © 2004 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, Etats-Unis. Tous droits réservés.

Cette distribution peut comprendre des composants développés par des tierces parties. Sun, Sun Microsystems, le logo Sun et Java sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays.

Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Les produits qui font l'objet de ce manuel d'entretien et les informations qu'il contient sont regis par la législation américaine en matière de contrôle des exportations et peuvent être soumis au droit d'autres pays dans le domaine des exportations et importations. Les utilisations finales, ou utilisateurs finaux, pour des armes nucléaires, des missiles, des armes biologiques et chimiques ou du nucléaire maritime, directement ou indirectement, sont strictement interdites. Les exportations ou réexportations vers des pays sous embargo des Etats-Unis, ou vers des entités figurant sur les listes d'exclusion d'exportation américaines, y compris, mais de manière non exclusive, la liste de personnes qui font objet d'un ordre de ne pas participer, d'une façon directe ou indirecte, aux exportations des produits ou des services qui sont régis par la législation américaine en matière de contrôle des exportations et la liste de ressortissants spécifiquement désignés, sont rigoureusement interdites. LA

DOCUMENTATION EST FOURNIE "EN L'ETAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFAÇON.



Adobe PostScript

Contents

Preface	7
1. Getting Started	11
Obtaining and Installing the Required Software	11
Starting the IDE	12
2. Introduction to the Currency Converter	13
Description of the Currency Converter Application	14
Installing the Currency Converter Application	14
Compiling and Running the Application	16
Switching Emulators	18
Summary	20
3. Tutorial Concepts	21
MIDP Application Concepts	21
Java 2, Micro Edition (J2ME)	21
Connected, Limited Device Configuration (CLDC)	22
Mobile Information Device Profile (MIDP)	22
Structure of a MIDP Application	22
MIDlets	22
MIDlet Suites	22

Creating MIDlets in the Java Studio Mobility IDE	23
Mounting a Filesystem	23
Compilation and Preverification	23
Emulators	24
Networked Mobile Data Application	24
4. Creating a MIDlet and MIDlet Suite	25
Initial Setup	25
Creating the MIDlet Suite	26
Coding the MIDlet	28
Creating a MIDP Form	33
Creating a MIDP list	35
Packaging the Currency Converter Application	37
5. Debugging MIDlets and MIDlet Suites	41
MIDP Debugging in Java Studio Mobility	41
Setting a Breakpoint	42
Running a Debugging Session	43
6. The Currency Converter Networked Mobile Data Application	49
Description of the Currency Converter Application	50
Installing the Currency Converter Wireless Application	50
Compiling and Running the Application	52
Summary	54
7. Coding the Currency Converter Wireless Application	55
Initial Setup	55
Creating a Web Module	56
Extending Web Services to a MIDlet.	59
Executing the MIDlet	61

Summary 63

Index 65

Preface

This tutorial creates two applications that conform to the architecture of the Java™ 2 Platform, Micro Edition (J2ME™ platform). The applications are also compliant with the Connected, Limited Device Configuration (CLDC) and Mobile Information Device Profile (MIDP) standards. One of these applications demonstrate how client/server applications can be created using the J2ME Wireless Connection Wizard. By working through tasks that create, develop, and deploy these J2ME compliant applications, you will learn the major features of the Sun™ Java™ Studio Mobility 6 software.

Before You Read This Book

Before starting, you should be familiar with the following subjects:

- Java programming language

A knowledge of MIDP and CLDC concepts is helpful, as described in the following resources:

- CLDC Specification, v. 1.0
<http://java.sun.com/products/cldc/>
- MIDP Specification, v. 1.0
<http://java.sun.com/products/midp/>

Note – Sun is not responsible for the availability of third-party Web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused by or in connection with the use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

How This Book Is Organized

[Chapter 1](#) provides the information necessary to set up the Sun Java Studio Mobility 6.

[Chapter 2](#) explains the Currency Converter application used in this tutorial and shows you how to quickly install, execute, and run the application.

[Chapter 3](#) provides a glossary and explanation of MIDP/CLDC concepts used in this tutorial.

[Chapter 4](#) shows you how to use the Java Studio Mobility to code and compile the Currency Converter application.

[Chapter 5](#) shows you how to use Java Studio Mobility to debug the Currency Converter application.

[Chapter 6](#) explains the Currency Converter Wireless application used in this tutorial and shows you how to quickly install, execute, and run the application.

[Chapter 7](#) shows you how to use Java Studio Mobility to code and compile the Currency Converter Wireless application.

Using UNIX Commands

This document might not contain information on basic UNIX® commands and procedures such as shutting down the system, booting the system, and configuring devices. See the following for this information:

- Software documentation that you received with your system
- Solaris™ operating environment documentation, which is at

<http://docs.sun.com>

Shell Prompts

Shell	Prompt
C shell	<i>machine-name%</i>
C shell superuser	<i>machine-name#</i>
Bourne shell and Korn shell	\$
Bourne shell and Korn shell superuser	#

Typographic Conventions

Typeface*	Meaning	Examples
AaBbCc123	The names of commands, files, and directories; on-screen computer output	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. <code>% You have mail.</code>
AaBbCc123	What you type, when contrasted with on-screen computer output	<code>% su</code> <code>password:</code>
<i>AaBbCc123</i>	Book titles, new words or terms, words to be emphasized. Replace command-line variables with real names or values.	Read Chapter 6 in the <i>User's Guide</i> . These are called <i>class</i> options. You <i>must</i> be superuser to do this. To delete a file, type <code>rm filename</code> .

* The settings on your browser might differ from these settings.

Related Documentation

Application	Title	Part Number
Installation and setup	<i>Sun Java Studio Mobility 6 Getting Started Guide</i>	817-2340-10

Accessing Sun Documentation

You can view, print, or purchase a broad selection of Sun documentation, including localized versions, at:

<http://www.sun.com/documentation>

Contacting Sun Technical Support

If you have technical questions about this product that are not answered in this document, go to:

<http://www.sun.com/service/contacting>

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. You can submit your comments by going to:

<http://www.sun.com/hwdocs/feedback>

Please include the title and part number of your document with your feedback:

Sun Java Studio Mobility 6 2004Q3 Tutorial, part number 817-2343-10

Getting Started

This chapter explains what you must do before starting the Sun Java Studio Mobility 6 tutorial. The topics covered in this section are:

- [“Obtaining and Installing the Required Software” on page 11](#)
- [“Starting the IDE” on page 12](#)

Obtaining and Installing the Required Software

The following items are used to create and run the tutorial:

- The Sun Java Studio Mobility 6 Integrated Development Environment (IDE) which includes the following:
 - J2ME Wireless Toolkit 2.1
 - J2ME Wireless Toolkit 1.0.4_01 (optional)
 - J2ME Wireless Connection wizard (optional at installation, but required for the client/server application)
 - The two Currency Converter applications you will build in this tutorial.

Instructions for obtaining and installing the Java Studio Mobility software are detailed in the *Sun Java Studio Mobility 6 2004Q3 Getting Started Guide*.

- Java 2 Software Development Kit (the J2SE™ SDK), version 1.4.1_02 or later 1.4.x release.

The installer will not run if you do not have the J2SE SDK on your system. If you do have a J2SE SDK on your system, the installer will start and then verify whether the J2SE SDK you are using is the version required by Java Studio

Mobility for your platform. If you do not have the required version, the installer will quit and display a message that you must install the correct version before proceeding.

You can obtain the J2SE SDK from the same locations as the Java Studio Mobility software.

Starting the IDE

There are several ways to start the Java Studio Mobility IDE. Only one is described here. For more options, see the *Sun Java Studio Mobility 6 2004Q3 Getting Started Guide*.

To start the IDE:

- **Start the IDE by running the program executable.**
 - On Microsoft Windows, choose Start → Programs → Sun Microsystems → Sun Java Studio Mobility 6
 - On Solaris, UNIX, and Linux environments, run the `runide.sh` script in a terminal window, as follows:

```
$ sh jstudio-install-directory/bin/runide.sh
```

The `jstudio-install-directory` variable stands for the IDE's home directory, which is by default `$HOME/SUNWjstudio/Mobile04q3` (UNIX standard user) or `/opt/SUNWjstudio/Mobile04q3` (UNIX superuser).

Introduction to the Currency Converter

This tutorial guides you through the construction of two J2ME platform, MIDP/CLDC applications:

- The first is a simple MIDP application, called a *MIDlet*, that illustrates how you can use the Java Studio Mobility IDE to create, debug, and run an application on the device emulators that are included in the IDE.
- The second application recreates the Currency Converter as a client/server application. In this scenario, you will use the J2ME Wireless Connection wizard to extend the Currency Converter service to a MIDP client. The client/server application will be explained more fully in [Chapter 6](#).

This chapter describes the structure and function of the simple application, and shows you how to quickly install, compile, and run the Currency Converter application on a device emulator.

This chapter is organized under the following topics:

- [“Description of the Currency Converter Application” on page 14](#)
- [“Installing the Currency Converter Application” on page 14](#)
- [“Compiling and Running the Application” on page 16](#)
- [“Switching Emulators” on page 18](#)
- [“Summary” on page 20](#)

Description of the Currency Converter Application

The currency converter application, called Currency Converter, converts amounts from one currency to two others. You can choose to display three different currencies, euros, yen, or dollars, and enter a value in one currency to be converted into the other selected currencies. There are three Java source code files for the sample application:

- `ConverterMIDlet.java`. The code for the MIDlet class.
- `Converter.java`. A MIDP form that defines the main screen of the application as it appears on a mobile device.
- `CurrenciesSelector.java`. A MIDP list that maintains the currencies and rates.

These files, which comprise the MIDlet, are packaged with a Java Application Descriptor (JAD) file and Jar Manifest file into a *MIDlet suite*. MIDlet suites, along with other key concepts, are explained in [Chapter 3](#).

Installing the Currency Converter Application

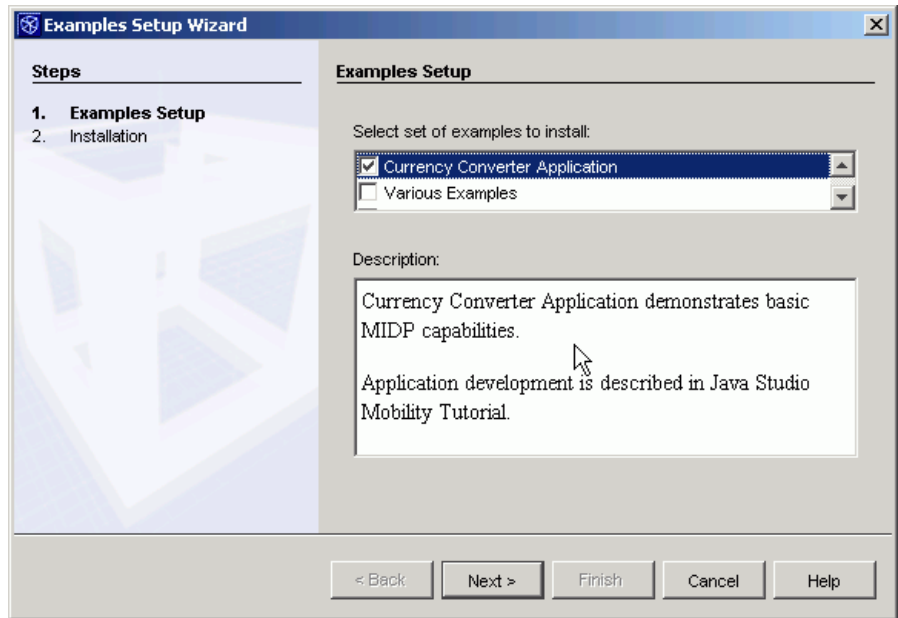
The Java Studio Mobility IDE provides an Examples Setup wizard that installs the complete example in the IDE.

Note – As with all future instructions, it is assumed the Java Studio Mobility IDE is up and running on your desktop.

To install the Currency Converter:

1. **Choose Help → Examples Setup Wizard.**

The Examples Setup wizard opens.

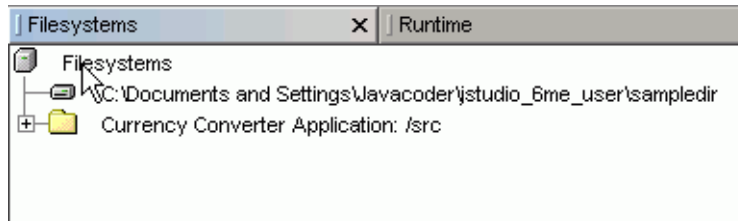


2. On the Examples Setup page, check the Currency Converter Application box. Click Next.

The installation page opens, with a progress bar that indicates when the example is installed.

3. Click Finish to close the Example Setup wizard.

The Currency Converter application is installed and mounted in the default examples filesystem. You can see the directory in the Filesystems window.

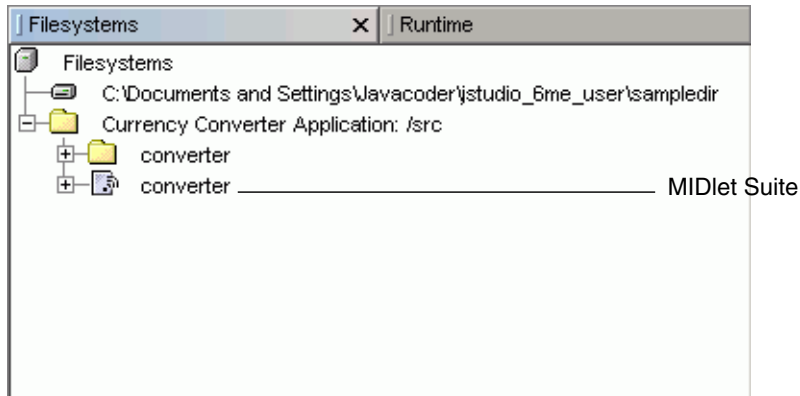


Compiling and Running the Application

Now that you have installed the application, the next section shows you how to compile and run the application in a device emulator.

1. **In the Filesystems window, expand the Currency Converter Application: /src filesystem node.**

A Converter folder and Converter MIDlet suite node are displayed.



2. **Right-click on the MIDlet suite and choose Execute.**

The Execute function compiles the MIDlet application, if necessary, before it executes the application.

Notice that the MIDlet suite's ConverterMIDlet application runs on the emulator device skin that is configured as the default. In this case, the default is the DefaultColorPhone skin within the J2ME Wireless Toolkit 2.1 installation.



Now you're ready to test the application in the device emulator.

3. **Select the currency you want to convert by clicking the up and down arrow keys on the Select button.**

You can select Dollars, Euros, or Yen.

4. **Enter the currency amount to convert by clicking the emulator's numeric keys.**
The application makes the conversion calculations and displays the results.
5. **Click the button underneath the word "Exit" to exit the application.**
6. **Click the red button in the upper right corner to close the emulator.**



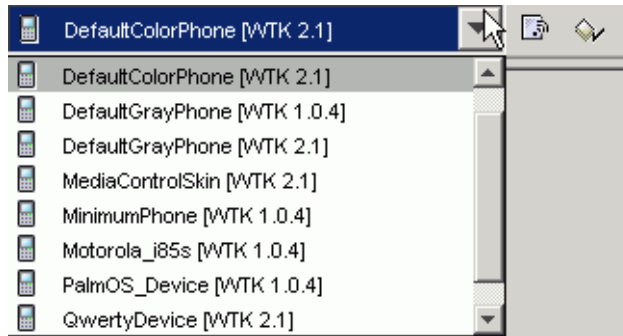
Switching Emulators

You can change the target emulator and device “skin” for a MIDlet suite, to test the performance and appearance of your MIDlet on different devices.

To switch Emulators:

1. Click the dropdown menu in the toolbar.

You can select from any of the currently installed emulators. The menu displays the default devices that are set for each of the installed emulators.



2. Select an alternate device skin. For a dramatic difference, try the RIMJavaHandheld [WTK 1.0.4] device or the QwertyDevice [WTK 2.1] device.

3. Right-click on the Converter MIDlet suite and choose Execute.

Notice that this time the application is executed with a different emulator.

4. Test the application as described in [“Compiling and Running the Application” on page 16](#).

Notice that in the QwertyDevice skin shown below, the emulator has a distinctly different appearance. However, the application performance is exactly the same on both device emulators.



Summary

In this chapter, you went through the few steps it takes to compile and run a simple MIDlet in an emulator device, and how to switch emulator devices.

The next chapter will explain some MIDP concepts and prepare you to write this application using the features of the Java Studio Mobility IDE.

Tutorial Concepts

In the previous chapter, you installed a *MIDP* application, called a *MIDlet*, and executed the packaged application, called a *MIDlet suite*, which was displayed in an *emulator device skin*.

This chapter briefly explains the concepts used in this tutorial, such as MIDlets and MIDlet Suites, and how they are applied in the Java Studio Mobility. These concepts are covered only briefly here, but can serve as a quick reference for you. You can find more detailed information on MIDP, CLDC, and other technologies, at the Wireless Developer web site at <http://developers.sun.com/techttopics/mobility/>.

The topics covered in this chapter are:

- “MIDP Application Concepts” on page 21
- “Structure of a MIDP Application” on page 22
- “Creating MIDlets in the Java Studio Mobility IDE” on page 23

MIDP Application Concepts

This section discusses the J2ME technology platform concepts upon which you create mobile applications.

Java 2, Micro Edition (J2ME)

Java 2, Micro Edition is a group of specifications and technologies that pertain to Java on small devices. The J2ME specification covers a wide range of devices, from pagers and mobile telephones through set-top boxes and car navigation systems. The J2ME platform is divided into configurations and profiles, specifications that describe a Java environment for a specific class of device.

Connected, Limited Device Configuration (CLDC)

The fundamental branches of the J2ME platform are *configurations*. A configuration is a specification that describes a Java Virtual Machine and some set of APIs that are targeted at a specific class of device.

The Connected, Limited Device Configuration is one such specification. The CLDC specifies the APIs for devices with less than 512 KB of RAM available for the Java system and an intermittent (limited) network connection. It specifies a stripped-down Java virtual machine, called the KVM, as well as several APIs for fundamental application services. Three packages are minimalist versions of the J2SE `java.lang`, `java.io`, and `java.util` packages. A fourth package, `javax.microedition.io`, implements the Generic Connection Framework, a generalized API for making network connections.

Mobile Information Device Profile (MIDP)

The Mobile Information Device Profile is a specification for a J2ME profile. It is layered on top of CLDC and adds APIs for application life cycle, user interface, networking, and persistent storage.

Structure of a MIDP Application

This section discusses the concepts specific to the MIDP platform.

MIDlets

An application written for MIDP is called a *MIDlet*. MIDlet applications are subclasses of the `javax.microedition.midlet.MIDlet` class that is defined by MIDP.

MIDlet Suites

MIDlets are packaged and distributed as *MIDlet suites*. A MIDlet suite can contain one or more MIDlets. The MIDlet suite consists of two files:

- Java Application Descriptor (`.jad`) file

The Java Application Descriptor file lists the archive file name, the names and class names for each MIDlet in the suite, and other information. This file is used by the mobile device to ensure that device has the minimum requirements to run the application.

- a Java Archive file (.jar) file.

The archive file contains the MIDlet classes and resource files.

Creating MIDlets in the Java Studio Mobility IDE

Java Studio Mobility is an integrated development environment (IDE), based on the NetBeans development platform, that enables you to use J2ME technologies and add special tools that enable you to code and test J2ME applications, such as emulators and obfuscators.

There are certain concepts used within the IDE that you must know to successfully create applications in the Sun Java Studio environment.

Mounting a Filesystem

A *filesystem* is comparable to a *directory* in the operating system. *Mounting a filesystem* displays the filesystem in the Explorer window, and lets you browse and work with filesystem's files from within the Java Studio Mobility IDE. The mounted filesystem is included in the Java classpath, which is necessary for compiling, running, and debugging code.

Compilation and Preverification

When you compile with the Java Studio Mobility, the tool's MIDP compiler combines into one sequence several steps that might otherwise have to be performed individually. The compiler compiles the MIDlet's .java file and produces a binary .class file.

In the J2SE platform, a bytecode verifier checks the class file to ensure that it meets Java standards and will behave well when executed. However, the code that implements bytecode verification is too large to fit on most mobile devices. Because of the limited memory of mobile devices, bytecode verification for J2ME, CLDC-based applications is broken down into two steps.

In the first step, a MIDlet is *preverified*, after compilation, to ensure that the MIDlet meets the requirements for the device it is being compiled for. The Preverifier rearranges the bytecode in the compiled classes to simplify the final stage of byte code verification on the CLDC virtual machine, and also checks for the use of virtual machine features that are not supported by the CLDC.

In the second step, classes are verified as they are loaded on the device.

In Java Studio Mobility, the preverification step is completed transparently when you compile or execute your MIDlet. You might never need to know this is happening. However, you should know that the preverification compiler can be set, if necessary, in the Execution properties for that MIDlet.

Emulators

An emulator lets you simulate the execution of an application on a target device, just as the user might experience it. It gives you the ability to run and debug applications within the IDE. Typically, an emulator includes a sample of devices that it emulates. These sample devices are called *skins*.

The J2ME Wireless Toolkit Emulator is the default emulator for the Java Studio Mobility IDE. It includes a number of example devices or skins, such as a Default Color Phone and a QWERTY Device. You can test your application on any of the Wireless Toolkit Emulator's example devices. This feature also gives you the ability to test the portability of your application across different devices.

Networked Mobile Data Application

A Networked Mobile Data Application is a client/server application in which the client MIDlet, run on a mobile device, extends an application that exists on a server. The second Currency Converter application in this tutorial is an example of such an example.

Creating a MIDlet and MIDlet Suite

This section of the tutorial shows you how to create a currency converter application using the tools available to you through the Java Studio Mobility IDE. The chapter takes you through the tasks necessary to build the J2ME, MIDP/CLDC application, Currency Converter. The sections in this chapter are:

- [“Initial Setup” on page 25](#)
- [“Creating the MIDlet Suite” on page 26](#)
- [“Coding the MIDlet” on page 28](#)
- [“Creating a MIDP Form” on page 33](#)
- [“Creating a MIDP list” on page 35](#)
- [“Packaging the Currency Converter Application” on page 37](#)

As you go through the tutorial, keep in mind that the Java Studio Mobility often has more than one way to perform a particular function. The tutorial illustrates one way to perform a function, but there are often other ways to accomplish the same function. For example, functions from the drop-down menus can usually be accessed by right-clicking on an entity and then selecting from its contextual menu. As you grow more familiar with the tool, you will find the operational mode with which you are most comfortable.

Initial Setup

Before you create the Currency Converter MIDlet, you need to select a directory (filesystem) on your system, and mount it into the IDE. (Mounting is explained in [“Mounting a Filesystem” on page 23](#).) Then you’ll create a package, `myconverter`, to contain the Currency Converter files.

This tutorial will use a directory in the default Java Studio Mobility user directory for a user named JavaCoder on the Windows platform:

```
c:\Documents and Settings\JavaCoder\jstudio_6me_user\examples.
```

If the `examples` filesystem is already mounted, you can skip to [Step 4](#).

To mount a filesystem and create the package:

1. **From the File menu of the IDE, choose Mount Filesystem.**

This opens a wizard from which you choose the template for the filesystem.

2. **Select the Local Directory and click Next.**

3. **Use the wizard to navigate to the `examples` directory. Select this directory and click Finish to complete the mount process.**

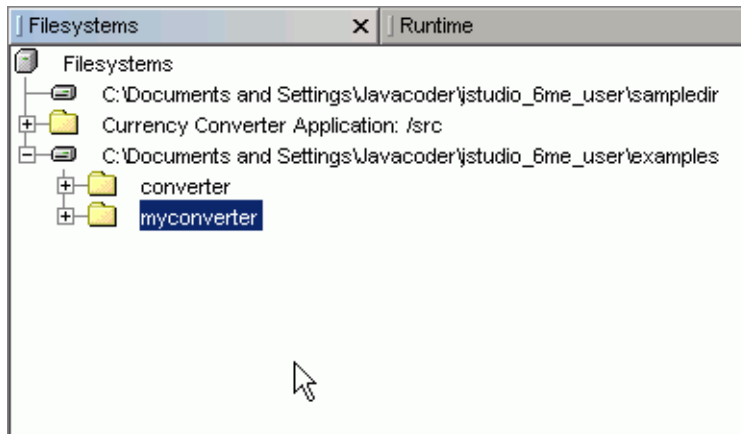
The `examples` filesystem appears in the Filesystems tab of the Explorer pane.

4. **Right-click on the `examples` filesystem, then choose New→Java Package.**

This opens the New Java Package wizard.

5. **Name the package `myconverter`. Click Finish.**

A package `myconverter` is created inside the mounted filesystem `examples`.



Creating the MIDlet Suite

While you can work with individual MIDlets for developing and testing purposes, it is best to create MIDlets within a MIDlet suite. The MIDlet suite helps you to package your MIDlet application and prepare it for deployment.

MIDlet suites give you more control over your MIDP applications. A MIDlet suite organizes the source files and attribute files for a MIDP application. When you build a MIDlet suite, the tool automatically creates the necessary JAR file that contains the application files. The Java Studio Mobility IDE also creates the Java Application Descriptor file, or JAD file, that is required for deployment.

To create a MIDlet Suite:

1. **In the Filesystems window, right-click the `myconverter` package. Choose **New → MIDlet Suite** from the contextual menu.**

The MIDlet Suite wizard takes you through the steps to create a MIDlet suite.

2. **In the MIDlet Suite wizard, type the name for the new MIDlet suite. Then click **Next**.**

Name the new MIDlet suite `converter`.

3. **In the Add MIDlet page, do the following to create a new MIDlet within the suite:**

- a. **Select the **Create New MIDlet** option.**

- b. **Enter `ConverterMIDlet` as the Package and Class Name.**

Notice that you need to capitalize the “C” in the name.

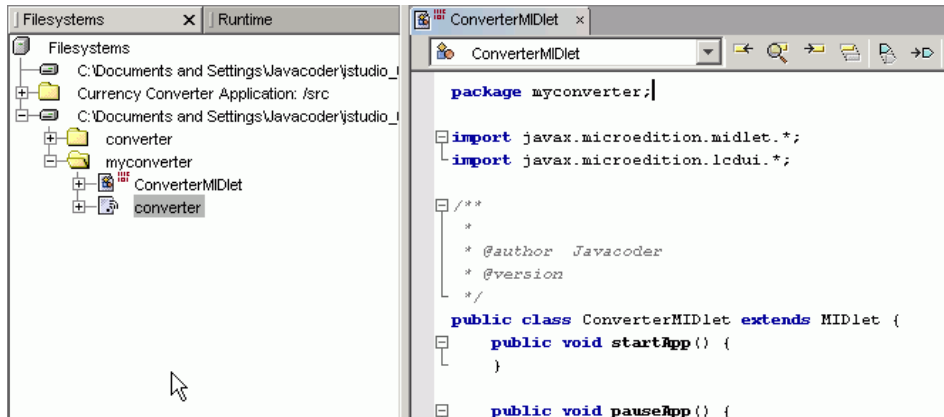
- c. **To select a MIDlet Template, click the arrow in the dropdown list and choose **MIDlet**.**

- d. **Click **Next**.**

4. **In the MIDlet Properties page, change the MIDlet Displayable Name to `Currency Converter`. Click **Finish**.**

The displayable name is the name a mobile device user will see when using the application on a mobile device.

The code for the MIDlet is displayed in the Source Editor window. Notice that the `ConverterMIDlet` icon in the Explorer tab has a set of small red x’s and 0’s next to it. This badge signifies that the MIDlet needs to be compiled.



In the following steps, you will add code to complete the Currency Converter application.

Coding the MIDlet

You can write the code for a MIDlet in one of two ways: either by directly entering code in the Source Editor or by using the tool functions to add methods, fields, constructors, initializers, classes, and interfaces. Typically, you use the tool to add new fields and methods to a class, or modify existing fields and methods, and then later fine-tune the code directly in the Source Editor.

The following procedure shows you how to use the tool and the Source Editor to enter or change code. However, to save time and effort, you can also copy the converter code from the example you installed in [Chapter 2](#).

1. In the Source Editor, add the following import statements to ConverterMIDlet:

```
import java.io.*;
import javax.microedition.rms.*;
```

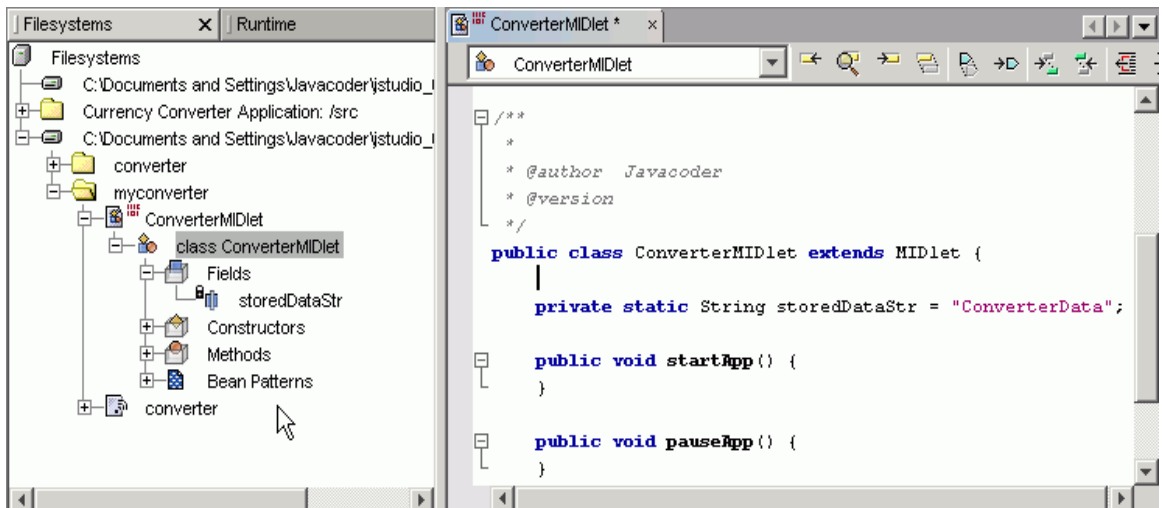
2. In the Filesystem tab window, expand the converterMIDlet node, right-click the ConverterMIDlet class and choose Add→ Field.

This next step will use the Add New Field dialog box to add the field `storedDataStr` to the MIDlet. The `storedDataStr` string contains the name of the RMS stored record.

3. Complete the Add New Field dialog box:

- a. Enter the name of the new field, `storedDataStr`, in the Name box and select its type, `String`, from the Type combo box.
- b. In the Modifiers box, select the type of access for the field, `private`, from the Access combo box.
- c. Check the other modifiers for the field, which in this case is `static`.
- d. Set the initial value for `storedDataStr` to `"ConverterData"`.
- e. Click OK to close the dialog box.

The field is added to the code in the Source Editor window.



4. Add the following fields to the MIDlet code using the Source Editor.

Tip – You can use the Add Field dialog box, copy the text from this page, or from the installed Currency Converter application, and paste it in the Source Editor. Be careful, however, not to change the package name from `myconverter`.

```
public String[] currencies = new String[] { "US $", "Yen \u00a5", "Euro \u20ac"
};
public boolean[] selected = new boolean[] { true, true, true, true };
public long[][] rates = {{ 1000000, 117580000, 911079 },
                        { 8504, 1000000, 7749 },
                        { 1097600, 129056000, 1000000 }};
private RecordStore storedData;
```

5. Add the following code to the method `startApp()`:

```
try {
    storedData = RecordStore.openRecordStore(storedDataStr,
true);
    if (storedData.getNumRecords() > 0) {
        DataInputStream in = new DataInputStream(new
ByteArrayInputStream(storedData.getRecord(1)));
        try {
            int size = in.readInt();
            currencies = new String[size];
            selected = new boolean[size];
            rates = new long[size][];
            for (int i=0; i<size; i++) {
                currencies[i] = in.readUTF();
                selected[i] = in.readBoolean();
                rates[i] = new long[size];
                for (int j=0; j<size; j++) {
                    rates[i][j] = in.readLong();
                }
            }
            in.close();
        } catch (IOException ioe) {
        }
    }
} catch (RecordStoreException e) {
}
notifySettingsChanged();
}
```

This method is called when the application is started. It loads all the data (currencies selected currencies, and exchange rates) from persistent storage and initially displays the Converter form.

6. Add the following code to complete the method `destroyApp()`:

```
try {
    ByteArrayOutputStream bytes = new ByteArrayOutputStream();
    DataOutputStream out = new DataOutputStream(bytes);
    try {
        out.writeInt(currencies.length);
        for (int i=0; i<currencies.length; i++) {
            out.writeUTF(currencies[i]);
            out.writeBoolean(selected[i]);
            for (int j=0; j<currencies.length; j++) {
                out.writeLong(rates[i][j]);
            }
        }
        out.close();
        if (storedData.getNumRecords() > 0)
            storedData.setRecord(1, bytes.toByteArray(), 0, bytes.size());
        else
            storedData.addRecord(bytes.toByteArray(), 0, bytes.size());
    } catch (IOException ioe) {
        ioe.printStackTrace();
    }
} catch (RecordStoreException e) {
    e.printStackTrace();
}
notifyDestroyed();
}
```

The `destroyapp()` method is called when the application is finished, or *destroyed*.

7. Add the following three new methods:

a. showSettings()

```
public void showSettings() {  
    Display.getDisplay(this).setCurrent(new CurrenciesSelector(this));  
}
```

This method creates and displays the CurrenciesSelector list.

b. notifySettingsChanged()

```
public void notifySettingsChanged() {  
    Display.getDisplay(this).setCurrent(new Converter(this));  
}
```

This method displays a new Converter form after the settings are changed.

c. longconvert()

```
public long convert(long frval, int fridx, int toidx) {  
    return (frval * rates[fridx][toidx]) / 1000000;  
}
```

This method performs the currency conversion. The input value, *frval*, is multiplied by the exchange rate stored in the rates table and divided by 1,000,000. The *fridx* and *toidx* values are the indexes of the source and target currencies.

8. Save the ConverterMIDlet by choosing File→ Save.

Creating a MIDP Form

Now that you have completed the code for the MIDlet, you will create the application's graphical interface. A Form is a Java class that can contain an arbitrary mixture of items, including images, read-only and editable text fields, editable date fields, gauges, choice groups, and custom items. The form you create here will specify a text box for each selected currency and specify the `ItemStateListener()` method to monitor and reflect typed values and perform conversions.

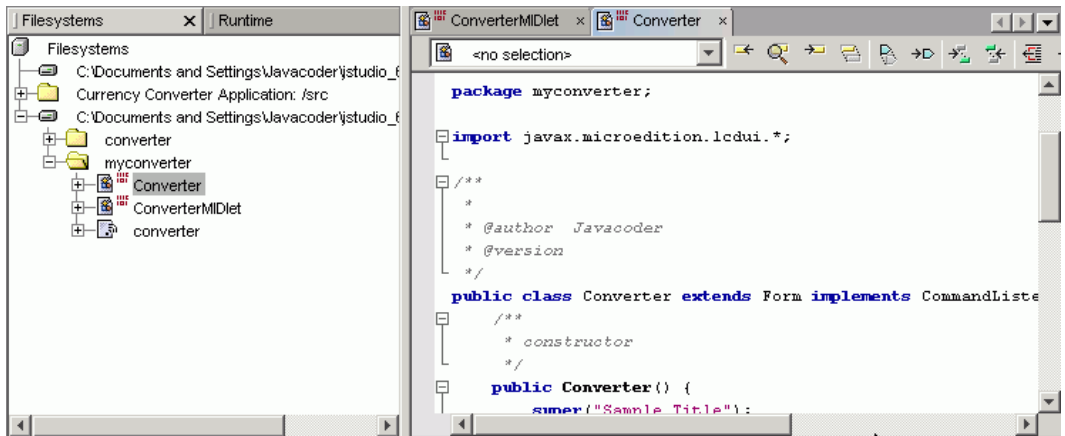
1. In the Filesystems window, right-click the `myconverter` package. Choose **New→All Templates**.

The New Template wizard opens.

2. Expand the MIDP node and select **MIDP Form**. Click **Next**.

3. In the Object name page, Enter `Converter` for the class name. Click **Finish**.

A MIDP form template is created and added to the `myconverter` package.



4. In the Source Editor, add the following fields to the code below the `public class Converter` declaration:

```
private ConverterMIDlet midlet;
private int[] translate;
```

5. Add the following code to replace the constructor:

```
public Converter(ConverterMIDlet midlet) {
    super("Currency Converter");
    this.midlet = midlet;
    this.translate = new int[midlet.currencies.length];
    int current = 0;
    for (int i=0; i<translate.length; i++) {
        if (midlet.selected[i]) {
            translate[current++] = i;
            append(new TextField(midlet.currencies[i], "", 12,
TextField.NUMERIC));
        }
    }
    try {
        // Set up this form to listen to command events
        setCommandListener(this);
        // Set up this form to listen to changes in the internal state of
its interactive items
        setItemStateListener(this);
        // Add the Currencies command
        addCommand(new Command("Currencies", Command.OK, 1));
        // Add the Exit command
        addCommand(new Command("Exit", Command.EXIT, 1));
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

6. Add the following code to complete the method `commandAction()`:

```
if (command.getCommandType() == Command.EXIT) {
    midlet.destroyApp(true);
} else if (command.getCommandType() == Command.OK) {
    midlet.showSettings();
}
```

7. Add the following code to complete the `itemStateChanged()` method:

```
long value = Long.parseLong(((TextField)item).getString());
int from = 0;
while (get(from) != item) from++;
from = translate[from];
for (int i=0; i<size(); i++) {
    int to = translate[i];
    if (from != to) {
        ((TextField)get(i)).setString(String.valueOf(midlet.convert(value,
from, to)));
    }
}
```

This completes the `Converter.java` form file.

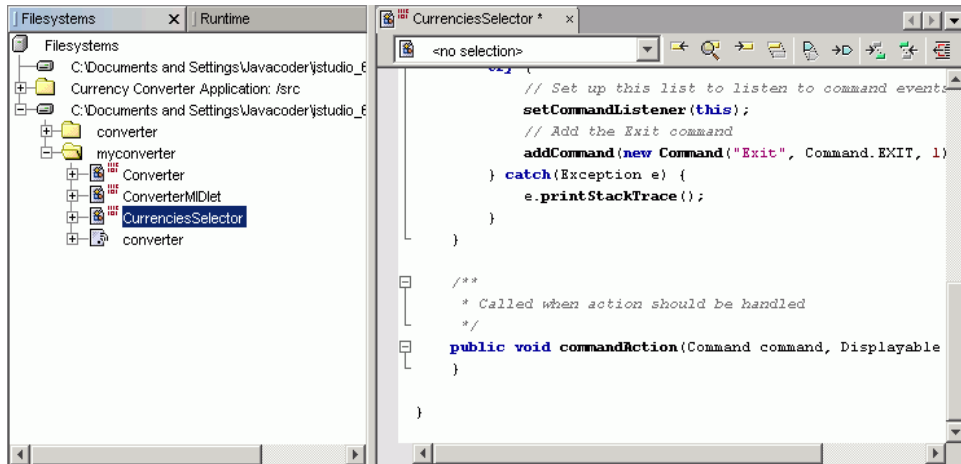
Creating a MIDP list

The final piece of the Currency Converter application is the `CurrenciesSelector.java` list file, which defines the currencies that can be selected for display.

To create the list file:

- 1. In the Filesystems window, right-click the `myconverter` package. Choose `New→All Templates`.**
The New Template wizard opens.
- 2. Expand the MIDP node and select MIDP List. Click Next.**
- 3. In the New Object Name page, Enter `CurrenciesSelector` for the class name. Click Finish.**

A MIDP list template is created and added to the `currencyconverter` filesystem.



4. Declare a field:

```
private ConverterMIDlet midlet;
```

5. Add the following code to replace the constructor

```
public CurrenciesSelector(ConverterMIDlet midlet):
```

```

super("Select Currencies", List.MULTIPLE, midlet.currencies, null);
this.midlet = midlet;
setSelectedFlags(midlet.selected);
try {
    // Set up this list to listen to command events
    setCommandListener(this);
    // Add the Save command
    addCommand(new Command("Save", Command.OK, 1));
} catch(Exception e) {
    e.printStackTrace();
}

```

6. Add the following code to complete the commandAction method:

```

if (command.getCommandType() == Command.OK) {
    setSelectedFlags(midlet.selected);
    midlet.notifySettingsChanged();
}

```

7. To save the list, Select File from the main menu and choose Save.

This completes the `CurrenciesSelector.java` List file.

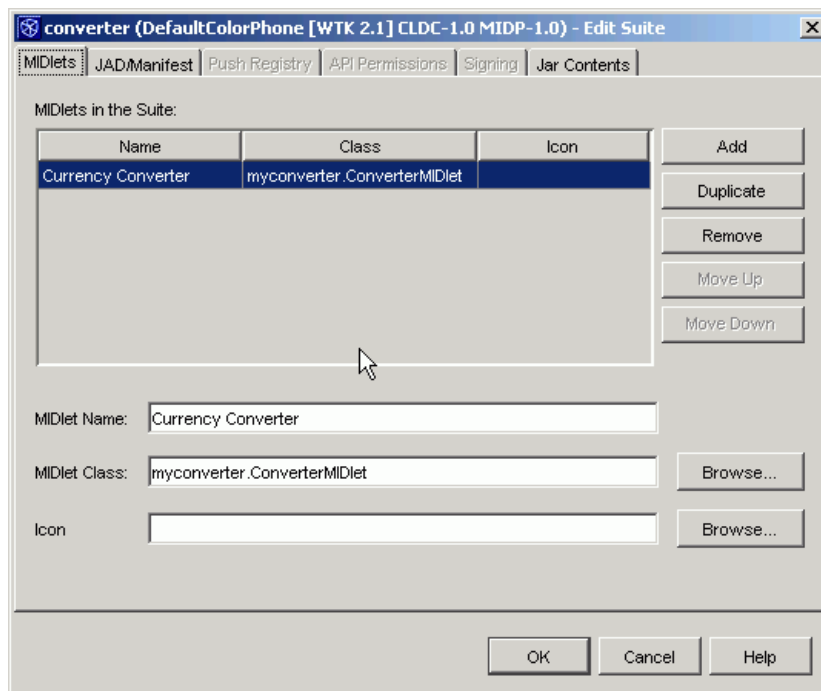
Packaging the Currency Converter Application

When you created the `converter` MIDlet suite, you created the essential package for the Currency Converter application. Now you should check to ensure that the two additional files you created, `Converter.java` and `CurrenciesSelector.java` have been added to the MIDlet suite. To do this, you use the Suite Editor.

To check the contents of the suite:

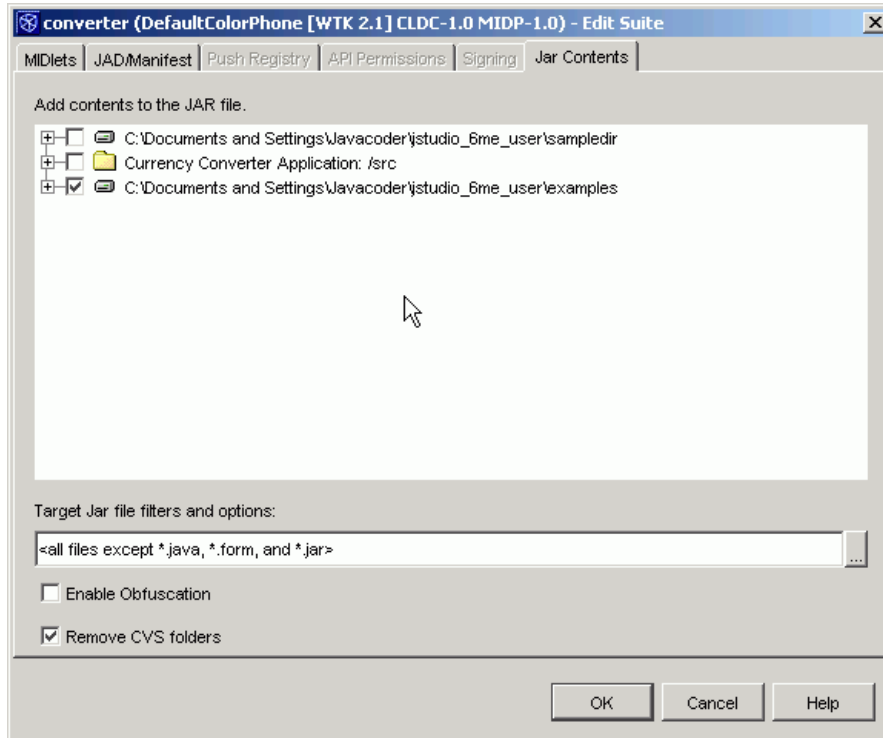
1. Right-click on the `converter` MIDlet suite and choose **Edit Suite** from the contextual menu.

The Suite Editor dialog opens. Notice that the Editor has several tabs: MIDlets, JAD/Manifest, Push Registry, API Permissions, Signing, and Jar Contents.



2. Select the Jar Contents Tab.

The Jar contents tab displays, showing you the mounted file systems available, and the current contents of `converter.jar`. The current contents are marked with a check in the check box.

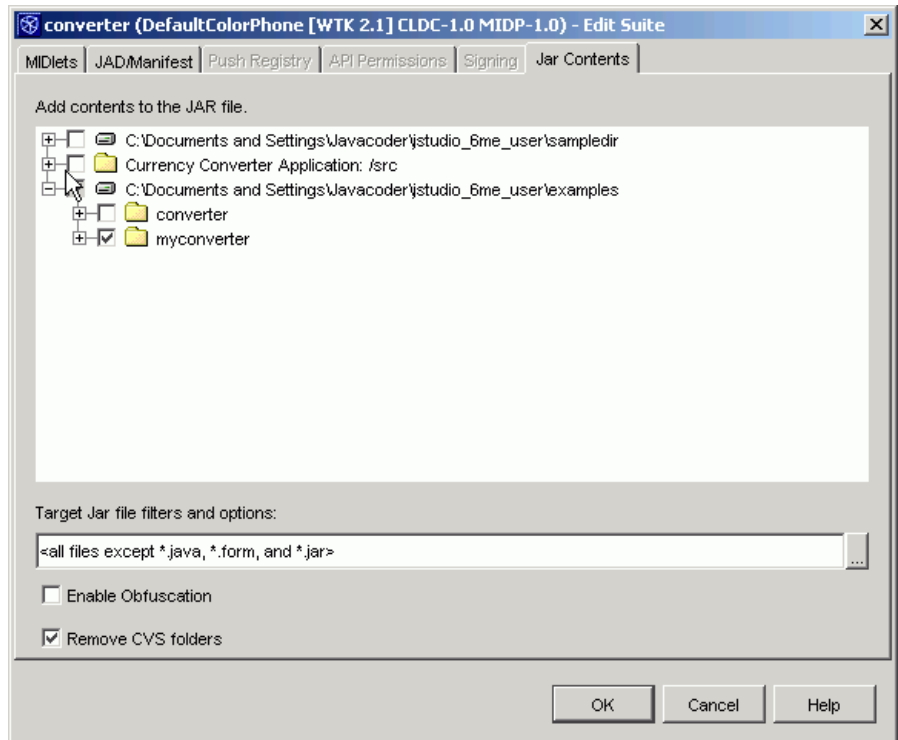


3. Expand the `examples` filesystem node.

4. Expand the `myconverter` folder.

Notice that the MIDlets are already selected. They were automatically added to the MIDlet Suite by the IDE.

If you left the “Include Whole Folder” option checked when you created the `converter` MIDlet in the New MIDlet Suite wizard (see [Step 3 in “Creating the MIDlet Suite” on page 26](#)), the `converter` example application, which also resides in the `examples` folder, is also included in the contents.



5. Make sure the converter folder is unchecked. Click OK to close the dialog.

Now the suite is ready to be compiled and executed in the same way you compiled the MIDlet in [“Compiling and Running the Application”](#) on page 16.

The next chapter will illustrate how to debug a MIDlet suite with the Java Studio Mobility IDE.

Debugging MIDlets and MIDlet Suites

This chapter describes how to debug MIDlets and MIDlet suites in the Java Studio Mobility IDE.

While this tutorial does not describe all the features of the IDE, the tutorial takes you through a short debugging session using the Currency Converter application you installed in [Chapter 2](#). The steps are described in the following sections:

- [“MIDP Debugging in Java Studio Mobility” on page 41](#)
- [“Setting a Breakpoint” on page 42](#)
- [“Running a Debugging Session” on page 43](#)

MIDP Debugging in Java Studio Mobility

If you have ever written a program before, you know that programs do not always perform in the way you expect them to. A program might display an incorrect value, go the wrong screen or page, or fail to display the information you expect to see. The process of finding why these mistakes are occurring is called *debugging*.

In the debugging process, you use breakpoints to stop the execution of the program and to examine its state at the location it has stopped. For example, if the currency conversion is displaying the wrong value, you can set a breakpoint in the code that does the conversion and examine the values of variables used in the computation. If your program is going to the wrong page when a soft button is pressed, you can set a breakpoint in the `commandAction()` method and single-step through it to see why it is displaying the wrong page.

This debugging example, designed to show just a few of the features of the Java Studio Mobility debugging environment, assumes that you want to examine the values used in the conversion. This example uses the Currency Converter application you installed in [Chapter 2](#).

In this debugging session, you will:

- Insert a breakpoint in the method where the conversion starts.
- Identify the method that performs the conversion.
- Examine the values used in the conversion.

Setting a Breakpoint

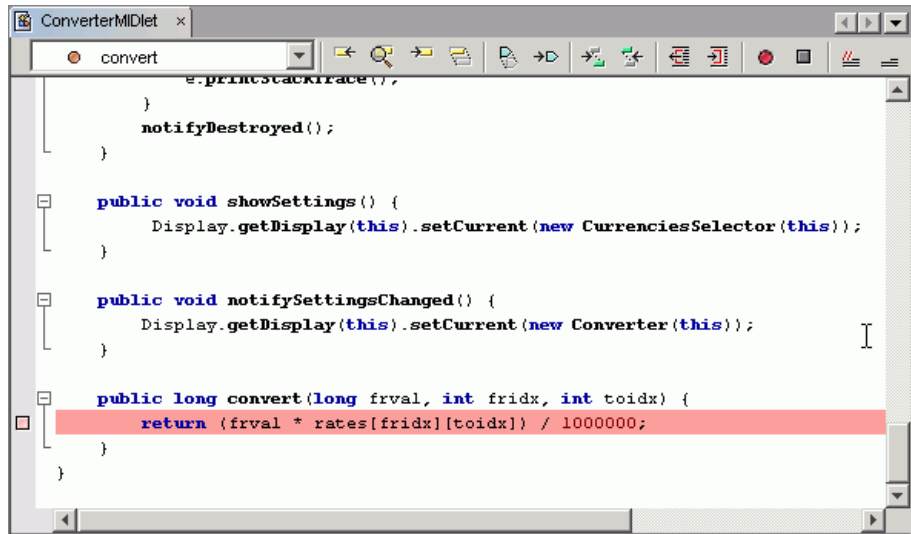
The first step in debugging is to set a breakpoint in the code where you first want to examine the execution of the program. Therefore, you want to set a breakpoint in the conversion code. At this point, however, you are not yet familiar with the code, and don't know where this point is. This is typical when debugging code you are not familiar with so you want to set a breakpoint in the code that initiates the conversion. Conversions are done as each number is entered so you want to set a breakpoint in the `itemStateChanged()` method in the `Converter` class. To set a breakpoint:

1. **Expand the `converter` folder so its contents, the three java source files of the Currency Converter MIDlet, are displayed.**
2. **Double-click on `ConverterMIDlet`.**
The `ConverterMIDlet.java` source file is shown in the Source Editor window.
3. **Scroll down in the MIDlet to the `convert()` method, the last method in the file.**
4. **Set a breakpoint by clicking in the left margin of the code line:**

```
return (frval * rates[fridx][toidx]) / 1000000;
```

The line is highlighted in pink, and is set as a breakpoint.

Note – Clicking the square on the left a second time will remove, or *unset*, the breakpoint setting.



```
e.printStackTrace(),
    }
    notifyDestroyed();
}

public void showSettings() {
    Display.getDisplay(this).setCurrent(new CurrenciesSelector(this));
}

public void notifySettingsChanged() {
    Display.getDisplay(this).setCurrent(new Converter(this));
}

public long convert(long frval, int fridx, int toidx) {
    return (frval * rates[fridx][toidx]) / 1000000;
}
}
```

Running a Debugging Session

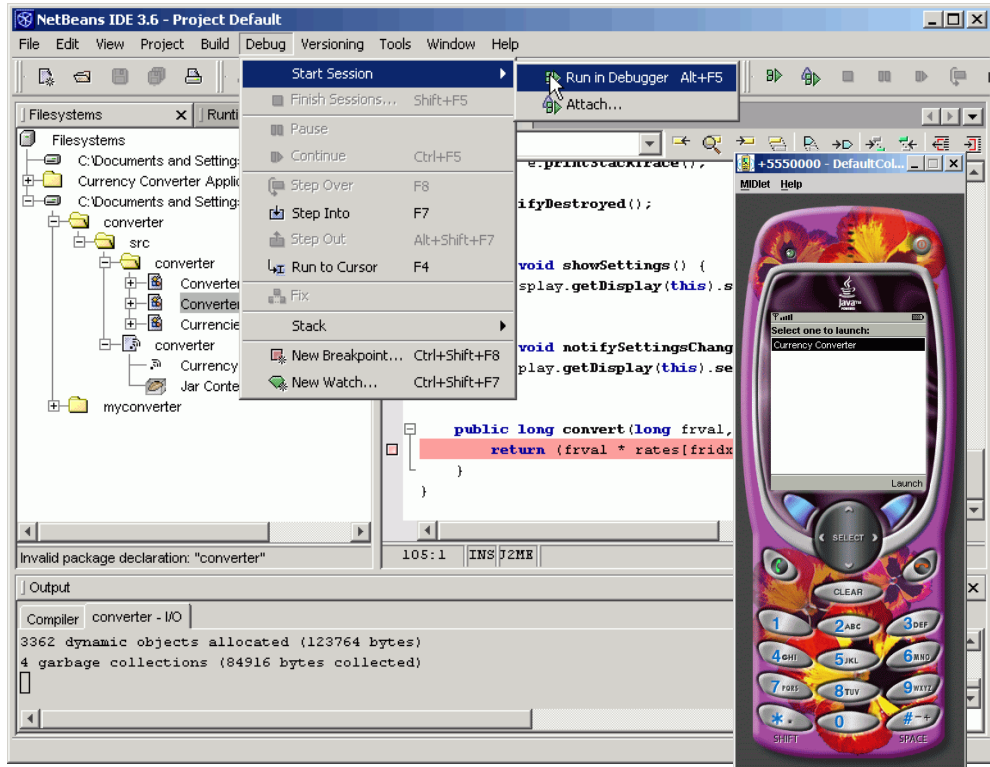
To make use of the breakpoint you just set, you need to run the MIDlet under the control of the debugger. If you execute the MIDlet using the `Execute` command, the IDE will ignore the breakpoint while executing the program.

1. **Start the Debugger by selecting the converter MIDlet Suite in the Filesystems window and choosing `Debug` → `Start Session` → `Run in Debugger`.**

The program runs, opening a device emulator. Notice that a Debugging window opens in the lower right corner. This is called the Debugging workspace. You can change the view using the tabs at the top of the window.

Note – If the emulator does not run as expected, make sure you have selected the converter MIDlet suite and not the Converter MIDlet.

You've now established your debugging environment, so you are ready to debug the application. To find the method you're looking for, you now need to cause a conversion to occur. This is done by interacting with the MIDlet in the device emulator.



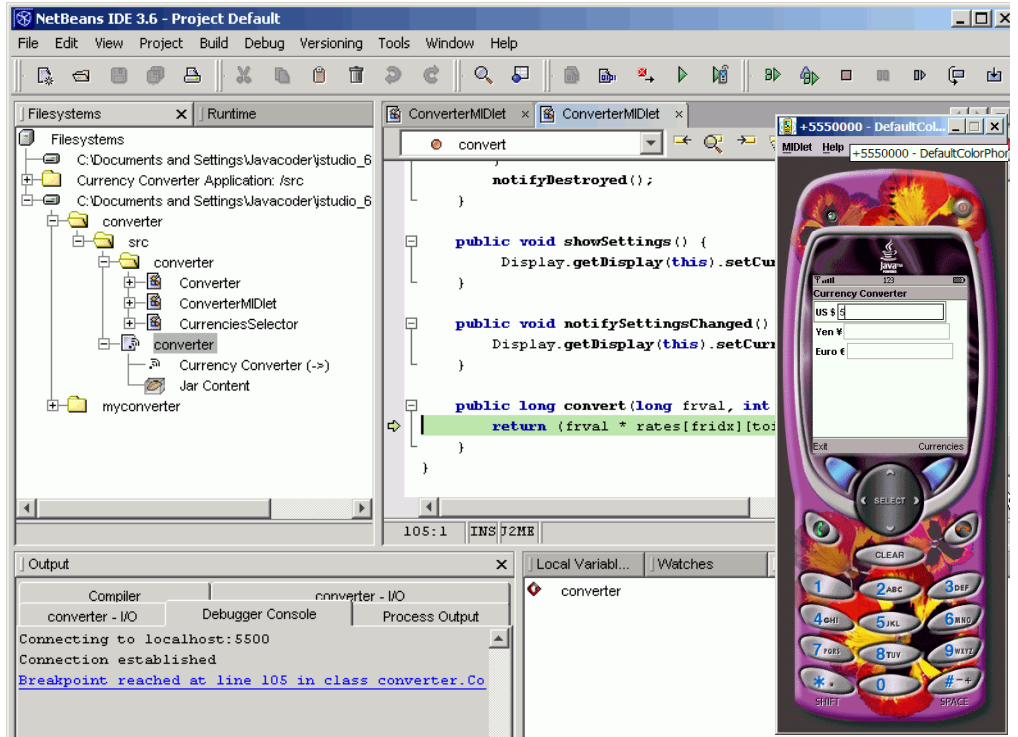
2. In the device emulator, click the Launch button to start the Currency Converter application.

The emulator screen displays the three currency fields: US, Yen and Euro.

3. Click in the device emulator, click the "5" button.

The numeral "5" appears in the US field. The IDE shows that the application has been activated, and the program runs until it hits the breakpoint at the `convert()` method. Notice that the line is now green, and the icon to the left has an arrow.

Now the debugger has control of the application, and you can begin to execute the application one source line at a time, or *step through* the application, until you find the line that will do the actual conversion.



4. Choose **Debug**→**Step Over** from the Main Menu to execute a line.

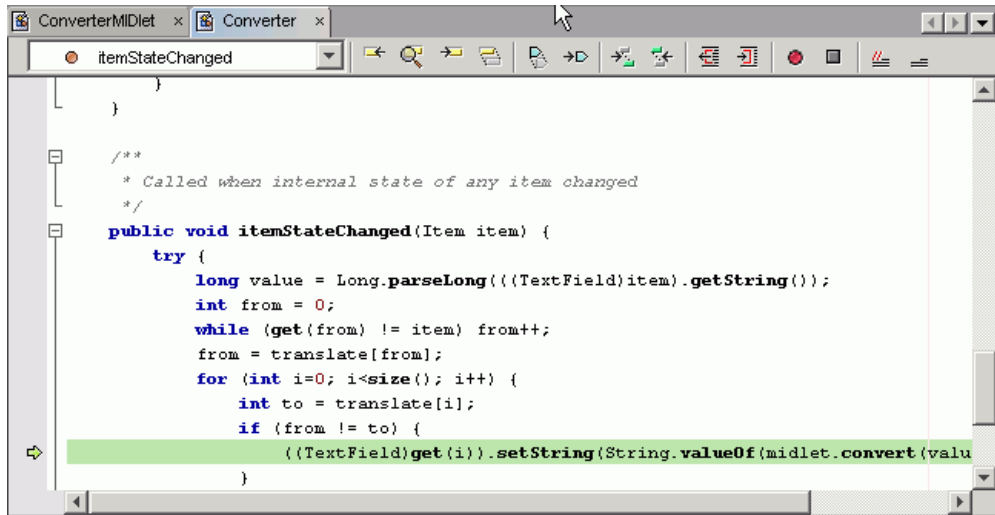
The debugger moves to the next line of code where the conversion occurs.

The function key shortcut for stepping over is the <F8> key. If you are single-stepping through many lines of code, it is often easier to use the shortcut.

5. Use the <F8> key to continue stepping through the code until the green line is on the source code line:

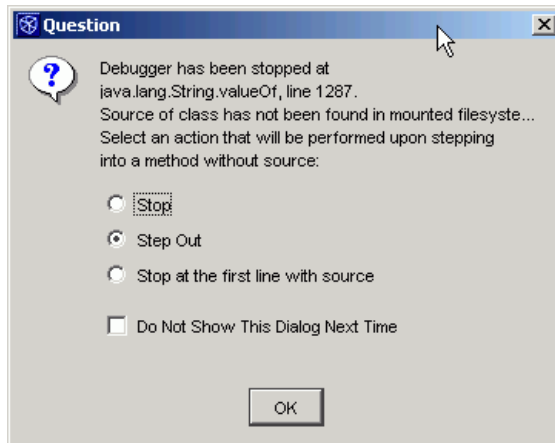
```
((TextField) get(i)).setString(String.valueOf(midlet.convert(value,
from, to)));
```

This is a complex line with several method calls. The `midlet.convert()` call is the one that will do the actual conversion, so that is the call you are interested in. You need to step into this line, because `midlet.convert()` is actually the second method to be executed.



6. Choose Debug→Step Into or press the <F7> key.

A dialog appears that informs you that the source of the `get()` method has not been found in the mounted filesystems. That is because the method invoked is a J2ME method, and is not part of the Currency Converter MIDlet.



7. Ensure that the Step Out radio button is selected and click OK.

8. To continue, choose Debug→Step into, or press the <F7> key.

This takes you past the `get()` method and to the `midlet.convert()` method.

9. Continue pressing the <F7> key.

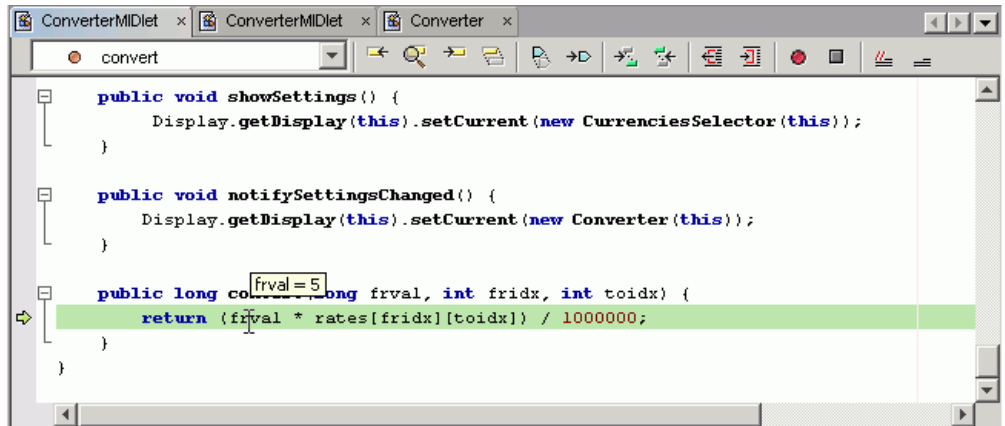
If the dialog appears again, select OK and continue to step into the program until you get to the following line:

```
return (frval * rates[fridx][toidx]) / 1000000;
```

You have now found the line that contains the variables used for conversion. You can examine the variables in the line using a “variable balloon.”

10. Place the mouse cursor over the variable `frval`.

A balloon caption appears indicating that the current value of `frval` is 5, the number you typed in when you began the debugging session.



11. Place the mouse cursor over the values `fridx` and `toidx` to check their values.

In this way, you can verify that the values are correct throughout your code.

12. To end the debugging session, choose Debug→Finish from the main menu.

This concludes this debugging session.

The Currency Converter Networked Mobile Data Application

Now that you have seen how to build the Currency Converter MIDlet, this chapter will show you a modified version of the same application. In this client/server version of the Currency Converter, the application can call the server to check the current rates of exchange before completing currency conversion calculations.

This chapter will guide you through installing and running the Currency Converter Wireless application. The next chapter will show you how to construct both client and server parts of the application, and how to connect them using the J2ME Wireless Connection Wizard.

Note – Because the previous Currency Converter example application uses identical filenames to the application you are about to install, unmount the `Currency Converter /src` filesystem before you continue. Otherwise, the similarity of filenames can cause compilation errors.

This chapter is organized under the following topics:

- [“Description of the Currency Converter Application” on page 50](#)
- [“Installing the Currency Converter Wireless Application” on page 50](#)
- [“Compiling and Running the Application” on page 52](#)
- [“Summary” on page 54](#)

Description of the Currency Converter Application

The following files comprise the client MIDlet Suite, `converter_networked`:

- `ConverterMIDlet.java`. The code for the MIDlet class.
- `Converter.java`. A MIDP form that defines the main screen of the application as it appears on a mobile device.
- `CurrenciesSelector.java`. A MIDP list that maintains the currencies and rates.
- `RatesUpdater.java`. A Java file that updates the currency rate information.

The server side of the application includes the following files:

- `ExchangeRateService.java`
- `RatesUpdaterServlet.java`
- `EndtoEndgateways.java`
- `InvocationAbstraction.java`
- `Utility.java`
- `EndToEndMapping.xml`. An XML mapping file created by the J2ME Wireless Connection wizard.

Installing the Currency Converter Wireless Application

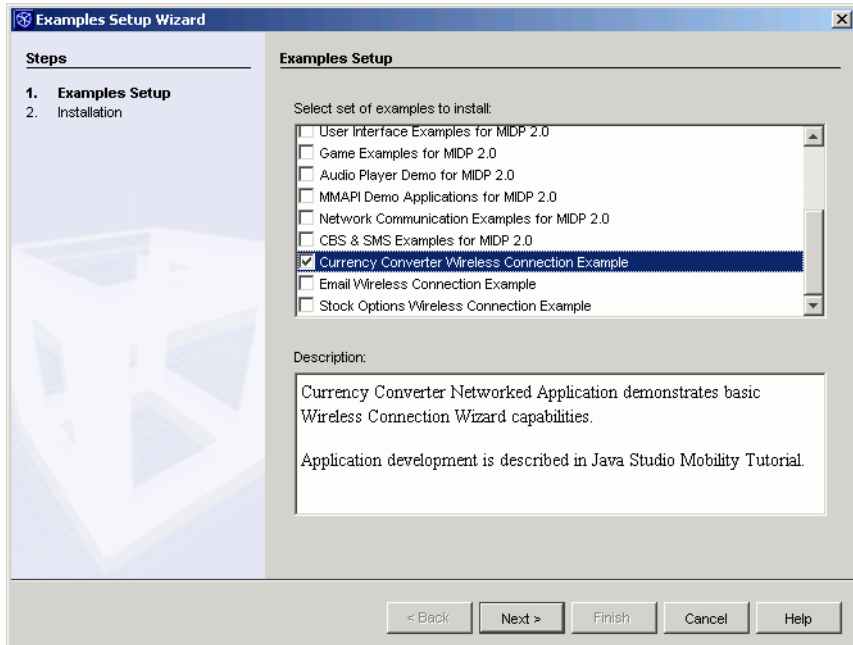
The Java Studio Mobility provides an Examples Setup wizard that installs the complete example in the IDE.

Note – As with all future instructions, it is assumed the Java Studio Mobility IDE is up and running on your desktop.

To install the Currency Converter:

1. **Choose Help → Examples Setup Wizard.**

The Examples Setup wizard opens.

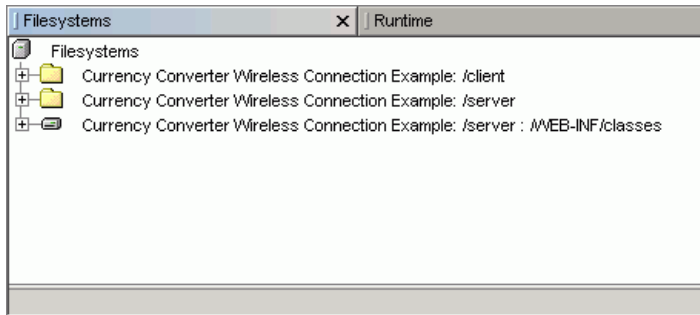


2. On the Examples Setup page, check the Currency Converter Wireless Connection Example box. Click Next.

The installation page opens, with a progress bar that indicates when the example is installed.

3. Click Finish to close the Example Setup wizard.

The Currency Converter Wireless Connection example is installed and mounted in the default examples filesystem. You can see the directory in the Filesystems window.

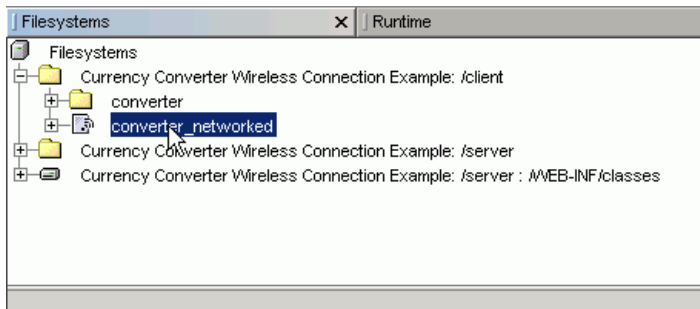


Compiling and Running the Application

Now that you have installed the application, the next section shows you how to compile and run the application in a device emulator.

1. **In the Filesystems window, expand the Currency Converter Wireless Connection Example: /client filesystem node.**

A Converter folder and Converter_networked suite node display.



2. **Right-click on the MIDlet suite and choose Execute.**

The Execute function compiles the MIDlet application, if necessary, before it executes the application.

As with the first Currency Converter application, the MIDlet suite runs on the default DefaultColorPhone skin within the J2ME Wireless Toolkit 2.1 installation.



Click here to select "Launch"

Now you're ready to test the application in the device emulator.

3. Select the currency you want to convert by clicking the up and down arrow keys on the Select button.

You can select Dollars, Euros, or Yen.

4. Enter the currency amount to convert by clicking the emulator's numeric keys. The application makes the conversion calculations and displays the results.
5. Click the button underneath the word "Exit" to exit the application.
6. Click the red button in the upper right corner to close the emulator.



Summary

In this chapter, you went through the few steps it takes to compile and run a simple MIDlet in an emulator device.

In the next chapter, you will learn how to modify the Currency Converter application so that it runs as a client/server application on a network.

Coding the Currency Converter Wireless Application

This section of the tutorial shows you how to modify the Currency Converter application you created in [Chapter 4](#) so that it can operate as client/server application. The chapter takes you through the tasks necessary to build a Web Module, modify the Currency Converter application, create the connecting code using the J2ME Wireless Connection Wizard.

The sections in this chapter are:

- [“Initial Setup” on page 55](#)
- [“Creating a Web Module” on page 56](#)
- [“Extending Web Services to a MIDlet.” on page 59](#)

Initial Setup

For this tutorial, you need to mount the filesystem where your Currency Converter application is stored. If it is not mounted, follow these instructions.

To mount the Currency Converter filesystem:

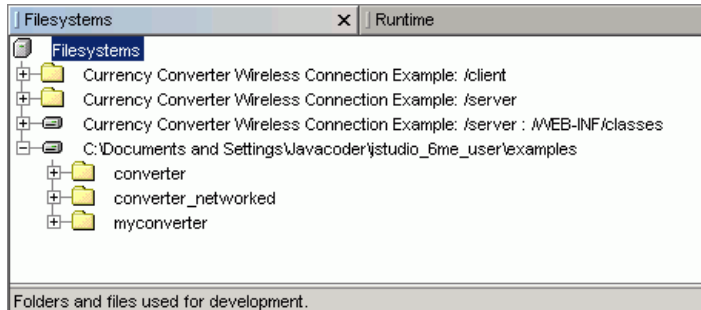
- 1. Choose File→Mount Filesystem.**
This opens the Mounting wizard.
- 2. Choose Local directory and click Next.**
- 3. In the Select Items to Mount page, navigate to the filesystem where you created Currency Converter.**

This tutorial will use a filesystem in the default Java Studio Mobility user directory for a user named JavaCoder on the Windows platform:

```
c:\Documents and Settings\JavaCoder\jstudio_6me_user\examples.
```

4. Click Finish.

The examples filesystem is mounted.

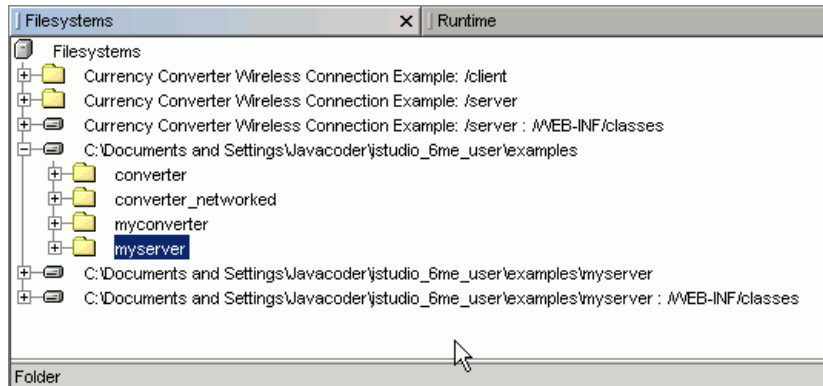


Creating a Web Module

Next, create the Web Module that stores the conversion rates for the Currency Converter.

1. **Right-click the `examples` filesystem and choose `New`→`All Templates`.**
This opens the New wizard.
2. **In the Choose template page, select the Folder template. Click Next.**
3. **Name the new folder `myserver`. Click Finish.**
4. **Right-click on the `myserver` folder and choose `New`→`All Templates`.**
5. **Expand the `JSPs & Servlets` node and select `Web Module`. Click Next.**
6. **In the Choose Target page, make sure the directory path points to the `myserver` folder. Click Finish.**

The Web Module is created and the filesystems `myserver` and `myserver: /WEB-INF/classes` are shown in the Explorer window.



The Web Module requires additional classes to perform the functions it needs to perform.

7. **Right-click the `myserver : /WEB-INF/classes` filesystem and choose `New→All Templates`.**

This opens the New wizard.

8. **In the Choose template page, select the Folder template. Click Next.**

9. **Name the new folder `myserver`. Click Finish.**

10. **Right-click on the folder you just created and choose `New→All Templates`.**

11. **Expand the Java Classes node and select the Java Class template. Click Next.**

12. **Name the class `ExchangeRatesService`. Click Finish.**

The class is shown in the Explorer window as a subnode of `server : WEB-INF/classes`.

13. Double-click on ExchangeRatesService to display the code for the class in the Source Editor window. Replace the code so the final class file looks like the following:

```
package myserver;
public class ExchangeRatesService {
    public String[] currencies = new String[] { "US $", "GB \u00a3", "Yen \
u00a5", "Euro \u20ac", "AU $", "CA $", "CZK" };
    public long[][] rates = {{ 1000000, 631114, 117580000, 911079,
1530220, 1406700, 29270000 },
                             { 1584500, 1000000, 186305000, 1443600,
2424630, 2228920, 47192000 },
                             { 8504, 5368, 1000000, 7749,
13014, 11964, 246858 },
                             { 1097600, 692711, 129056000, 1000000,
1679570, 1543990, 31251600 },
                             { 653501, 412434, 76838600, 595391,
1000000, 1087810, 17172700 },
                             { 710884, 448694, 83585700, 647671,
919280, 1000000, 19090800 },
                             { 34165, 21190, 4050910, 31998,
58232, 52381, 1000000 }};
    /** Creates a new instance of ExchangeRatesService */
    public ExchangeRatesService() {
    }
    public int getCurrenciesNum() {
        return currencies.length;
    }
    public String getCurrencyName(int index) {
        return currencies[index];
    }
    public long getRate(int from, int to) {
        return rates[from][to];
    }
}
```

This file now contains the exchange rates.

The next step is to create the MIDP class that will extend the Web services to the Currency Converter MIDlet. This is where you use the J2ME Wireless Connection wizard.

Extending Web Services to a MIDlet.

1. **Right-click the `myconverter` package and choose `New→J2ME Wireless Connection Mapping`.**

This opens the J2ME Wireless Connection Mapping wizard.

2. **In the `New Object Name` page, accept the default value. Click `Next`.**
3. **In the `Client Target Location` page, enter `RatesUpdater` for the generated J2ME class, and choose the `examples/myconverter` package. Click `Next`.**
4. **In the `Servlet Target Location` page, enter `RatesUpdaterServlet` for the generated servlet class and choose the `myserver: /WEB-INF/classes` filesystem. Click `Next`.**
5. **In the `Code Generation Options`, check the `Generate Stub Methods`, `Enable Multiple Calls per Connection`, and `Supports CLDC 1.1 Data Types` checkboxes. Click `Next`.**
6. **In the `Select Exported Services` page, expand `ExchangeRatesServices` so that you see the three services:**
 - `getCurrenciesNum`
 - `getCurrencyName`
 - `getRate`
7. **Select all three services and click the `Add` button.**

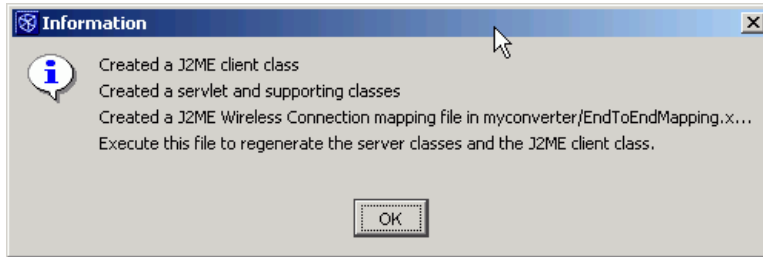
The services are shown in the `Selected Services` pane.

8. **Click `Next`.**

The `Summary` page lists all the changes you have designated. Notice that it will also generate a mapping file named `EndToEndMapping.xml`. Click `Finish`.

A dialog appears that lists the files created. Another dialog appears asking you to confirm changes in the deployment descriptor.

9. **Click `Process All` to confirm the changes, then click `OK` to dismiss the second dialog.**



10. Modify Converter.java:

a. Open Converter.java and add the following two imports:

```
import java.io.IOException;
import java.util.Enumeration;
```

b. Add Runnable to the list of interfaces that the Converter class implements:

```
public class Converter extends Form implements CommandListener,
ItemStateListener, Runnable {
```

c. Add the method new command() into constructor after the super call:

```
addCommand(new Command("Update Data", Command.SCREEN, 1));
```

d. Modify the commandAction() method to handle new command() by inserting the following code:

```
} else if (command.getCommandType() == Command.SCREEN) {
    Display.getDisplay(midlet).setCurrent(new Form("Please
wait..."));
    new Thread(this).start();
}
```

e. Add a new method implementing Runnable:

```
public void run() {
    try {
        RatesUpdater upd = new RatesUpdater();
        int size = upd.myserver_ExchangeRatesService_getCurrenciesNum();
        midlet.currencies = new String[size];
        midlet.selected = new boolean[size];
        midlet.rates = new long[size][];
        for (int i=0; i<size; i++) {
            upd.myserver_ExchangeRatesService_getCurrencyNameGrouped(i);
            for (int j=0; j<size; j++) {
                upd.myserver_ExchangeRatesService_getRateGrouped(i, j);
            }
        }
        Enumeration results = upd.getGroupedResults();
        for (int i=0; i<size; i++) {
            midlet.currencies[i] = ((String)results.nextElement());
            midlet.selected[i] = true;
            midlet.rates[i] = new long[size];
            for (int j=0; j<size; j++) {
                midlet.rates[i][j] = ((Long)results.nextElement()).longValue();
            }
        }
    } catch (IOException ioe) {
        ioe.printStackTrace();
    }
    midlet.notifySettingsChanged();
}
```

This completes the coding for the Currency Converter application.

Executing the MIDlet

Executing the MIDlet is the same procedure as before.

1. Right-click on the converter MIDlet suite and choose Execute.

The Execute function compiles the MIDlet application, if necessary, before it executes the application.

As with the first Currency Converter application, the MIDlet suite runs on the default DefaultColorPhone skin within the J2ME Wireless Toolkit 2.1 installation.



Now you're ready to test the application in the device emulator.

2. **Select the currency you want to convert by clicking the up and down arrow keys on the Select button.**

You can select Dollars, Euros, or Yen.

3. **Enter the currency amount to convert by clicking the emulator's numeric keys.**

The application makes the conversion calculations and displays the results.

4. **Click the button underneath the word "Exit" to exit the application.**



Summary

In this chapter, you used the J2ME Wireless Connector wizard to extend the services Currency Converter MIDlet into a mobile client application that accesses services on a web server.

Index

B

breakpoint, setting, 42

C

coding converterMIDlet, 28

compiling and running

 currency converter, 16

 currency converter wireless connection
 application, 52

Converter.java, 14

ConverterMIDlet.java, 14

creating, 26

CurrenciesSelector.java, 14

currency converter

 compiling and running, 16

 extending to client/server, 49

currency converter client/server application,
description, 13

currency converter networked application
installing, 50

currency converter, description, 13

D

debugging

 running a session, 43

 setting a breakpoint, 42

 step into command, 46

 step out radio button, 46

 step over command, 45

 stepping through, 44

 variable balloon, 47

debugging a MIDlet, 41

DefaultColorPhone device emulator, 16

displayable name, 27

E

emulators

 switching, 18

Examples Setup wizard, 14, 50

extending web services to a MIDlet, 59

I

installation, 11

 currency converter, 14

 Java Studio Mobility, 11

J

J2ME Wireless Connection Mapping, 59

J2ME Wireless Connection wizard, 59

Java package, 26

M

MIDlet

 debugging, 41

MIDlet displayable name, 27

MIDP form, 33

 creating, 33

MIDP list, 35

 creating, 35

mounting a filesystem

 instructions, 26

N

new field
 adding, 28

P

packaging the MIDP application, 37

R

required software, 11
RIMJavaHandheld device emulator, 19
running a debugging session, 43

S

starting the IDE, 12
step through, 44
switching emulators, 18

W

web module, creating, 56
web services, 59