Develop a location-based service application using JSR 179

Skill Level: Intermediate

Kevin L Sally (ksally@ca.ibm.com)
Senior IT Architect
IBM

10 Oct 2006

Walk through the steps required to successfully build and test a location-based Java™ Platform, Micro Edition (Java ME) application using the Eclipse SDK, the Java Specification Request (JSR) 179 Location API, and the Sun Java Wireless Toolkit.

Section 1. Before you start

This tutorial takes you through the steps needed to build a location-based services application on a Java ME CDLC 1.1 device using the JSR 179 specification.

The tutorial explains how to install and configure your development environment, then introduces the JSR 179 Location API and takes you through the process of developing a simple location-based application using the Eclipse SDK, the EclipseME plug-in, and the Location API. At the end of the tutorial you learn how to use a mobile phone emulator provided with the Sun Java Wireless Toolkit to test the LBSSample application.

Objectives

The goal of this tutorial is to teach you how to install and configure a Java ME development environment with a mobile phone emulator testing platform. After completing the tutorial, you should know the capabilities of the JSR 179 Location API and how to use it in your application. You should also be capable of acquiring location information and performing calculations with this data, much as you would with any standard GPS device.

Prerequisites

This intermediate-level tutorial is meant for developers with experience developing Java applications using the Eclipse SDK. Because this tutorial is meant to teach you about the JSR 179 Location API, no JSR 179 experience is required.

System requirements

The example application in this tutorial is built and tested using the Eclipse SDK 3.2, the EclipseME plug-in, and the Sun Java Wireless Toolkit 2.5 beta. Together, these technologies comprise the tutorial's integrated development environment, or IDE. You may use your own IDE configuration if you want; just keep in mind that Connected Limited Device Configuration (CLDC) 1.1 and the Mobile Information Device Profile (MIDP) 2.0 are minimum requirements for the JSR 179 Location API. The Eclipse 3.2 SDK also requires that you have installed the Java Platform, Standard Edition (JSE SDK) 1.5.0.

Section 2. Installation and configuration

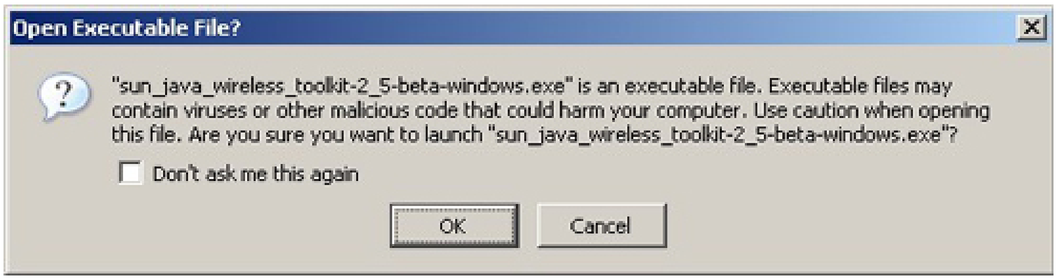
In this section, I'll walk you through installing and configuring the recommended integrated development environment for this tutorial, which consists of the Eclipse SDK 3.2, the Sun Java Wireless Toolkit (WTK) 2.5, and the EclipseME plug-in. I'll also show you how to configure the Eclipse IDE and EclipseME plug-in for Java ME development. Note that a JRE is required to run Eclipse.

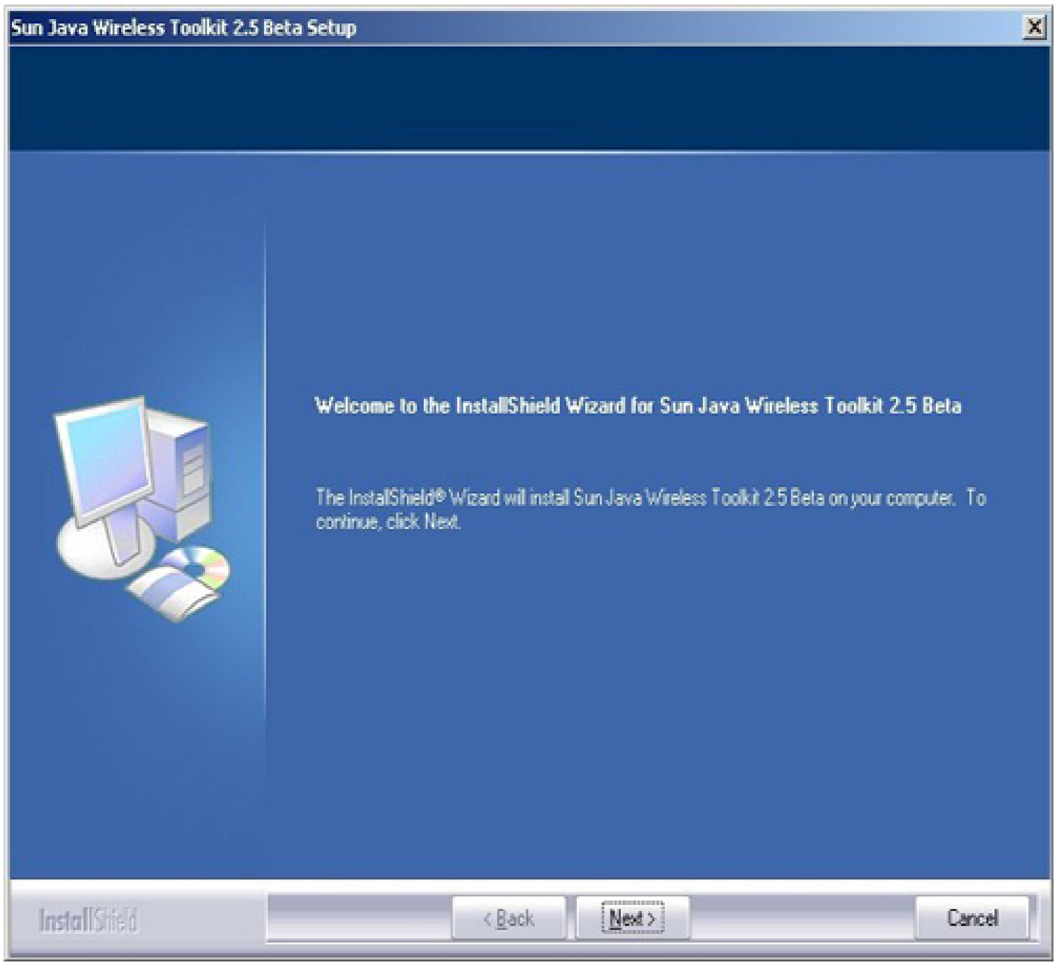
Install the Eclipse SDK 3.2

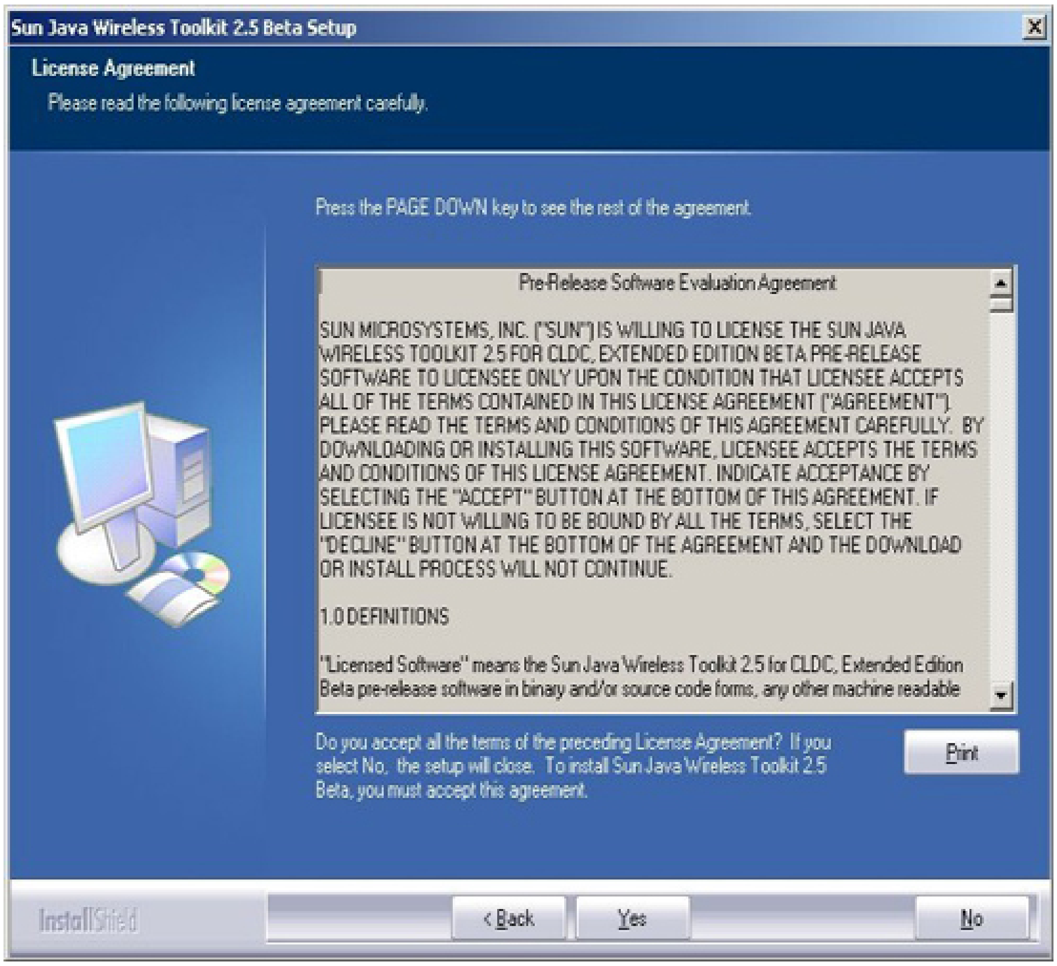
Download and install the Eclipse SDK 3.2 from the Eclipse homepage. Eclipse 3.2 is the latest version at the time of this writing and is the version used for examples in the tutorial. After it's downloaded, extract the contents into a directory of your choice. No further action is required to install Eclipse, though I'll walk you through installing the EclipseME plug-in shortly.

Install the Sun Java Wireless Toolkit

Download and install the Sun Java Wireless Toolkit from the Sun Java Wireless Toolkit homepage. WTK 2.5 is the latest version at the time of this writing and is the version used for examples in the tutorial. You'll need to take these additional steps after downloading the WTK:

1. Run the self-extracting executable and select **OK** if presented with the following warning message:
   Figure 1. WTK self-extracting warning message
   
   Open Executable File?
   "sun_java_wireless_toolkit-2_5-beta-windows.exe" is an executable file. Executable files may contain viruses or other malicious code that could harm your computer. Use caution when opening this file. Are you sure you want to launch "sun_java_wireless_toolkit-2_5-beta-windows.exe"?
   Don't ask me this again

2. Select **Next** from the following screen:
   Figure 2. Wireless Toolkit setup screen
   
   Sun Java Wireless Toolkit 2.5 Beta Setup
   Welcome to the InstallShield Wizard for Sun Java Wireless Toolkit 2.5 Beta
   The InstallShield® Wizard will install Sun Java Wireless Toolkit 2.5 Beta on your computer. To continue, click Next.

3. Read and accept the Sun Java Wireless Toolkit license agreement:
Figure 3. Accept the WTK license agreement


Please read the following license agreement carefully.

Press the PAGE DOWN key to see the rest of the agreement.

Pre-Release Software Evaluation Agreement

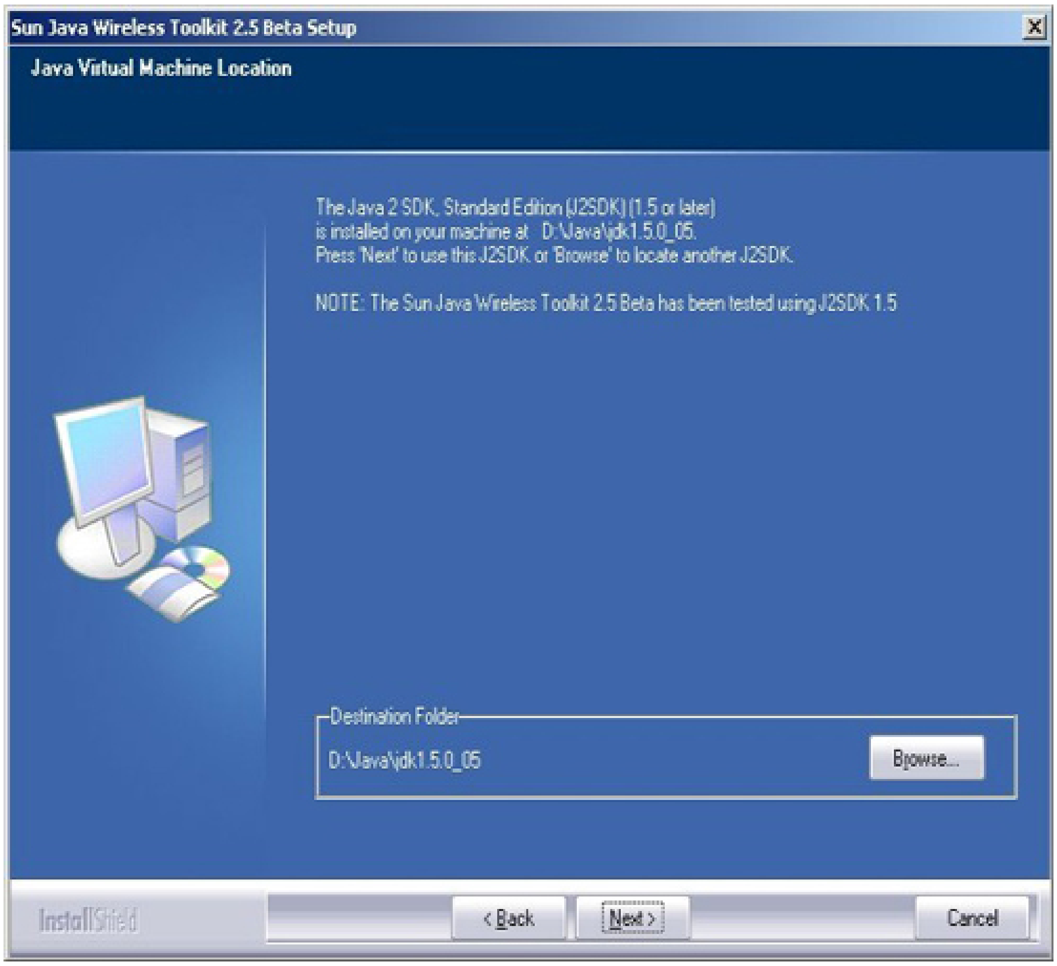
SUN MICROSYSTEMS, INC. ("SUN") IS WILLING TO LICENSE THE SUN JAVA WIRELESS TOOLKIT 2.5 FOR CLDC, EXTENDED EDITION BETA PRE-RELEASE SOFTWARE TO LICENSEE ONLY UPON THE CONDITION THAT LICENSEE ACCEPTS ALL OF THE TERMS CONTAINED IN THIS LICENSE AGREEMENT ("AGREEMENT"). PLEASE READ THE TERMS AND CONDITIONS OF THIS AGREEMENT CAREFULLY. BY DOWNLOADING OR INSTALLING THIS SOFTWARE, LICENSEE ACCEPTS THE TERMS AND CONDITIONS OF THIS LICENSE AGREEMENT. INDICATE ACCEPTANCE BY SELECTING THE "ACCEPT" BUTTON AT THE BOTTOM OF THIS AGREEMENT. IF LICENSEE IS NOT WILLING TO BE BOUND BY ALL THE TERMS, SELECT THE "DECLINE" BUTTON AT THE BOTTOM OF THE AGREEMENT AND THE DOWNLOAD OR INSTALL PROCESS WILL NOT CONTINUE.

1.0 DEFINITIONS

"Licensed Software" means the Sun Java Wireless Toolkit 2.5 for CLDC, Extended Edition Beta pre-release software in binary and/or source code forms, any other machine readable

Do you accept all the terms of the preceding License Agreement? If you select No, the setup will close. To install Sun Java Wireless Toolkit 2.5 Beta, you must accept this agreement.

4. The installation wizard searches for the JSE SDK version 1.5.x directory on your machine. After it finds the installed directory it prompts you with the screen shown in Figure 4. Click **Next** if the destination folder is the correct location of your JSE 1.5 SDK or JRE. Otherwise, click **Browse** and specify the root directory of your JSE 1.5 SDK or JRE.
Figure 4. Establish the location of your JVM


Sun Java Wireless Toolkit 2.5 Beta Setup

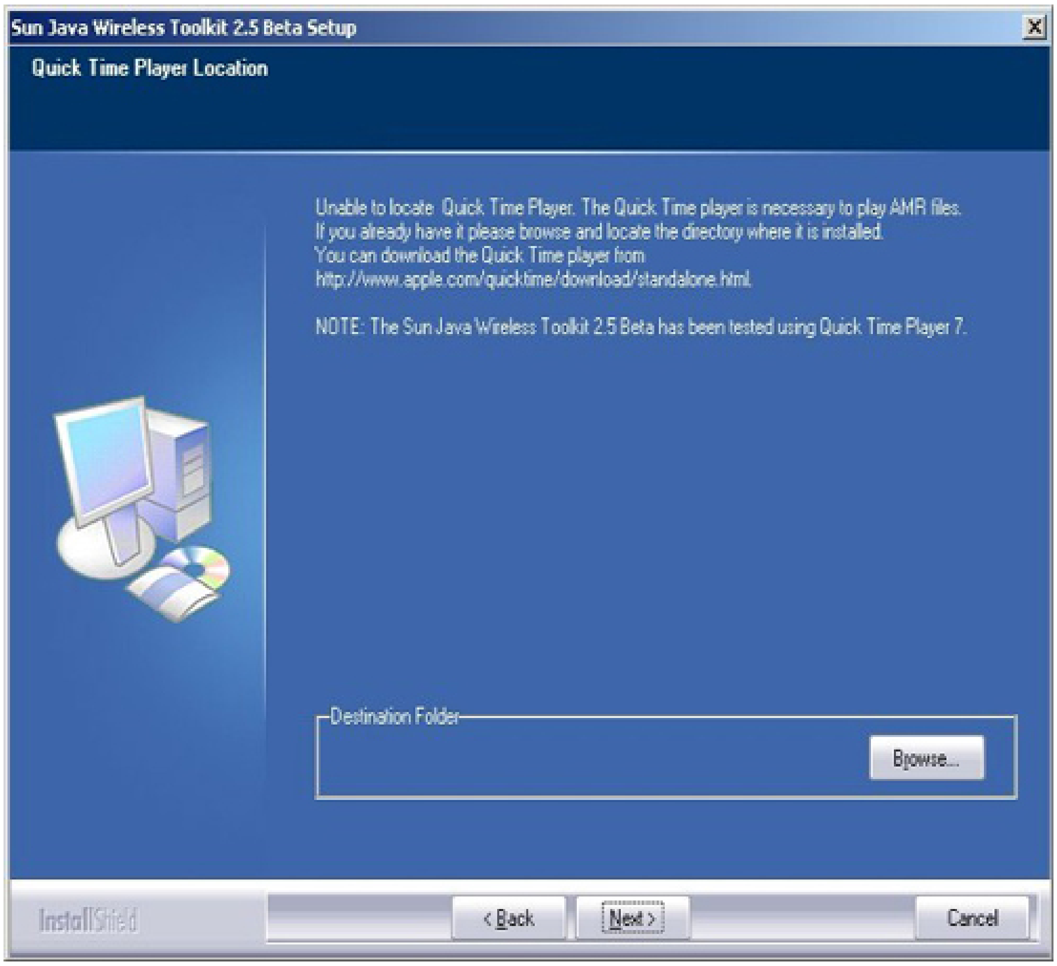
Java Virtual Machine Location

The Java 2 SDK, Standard Edition (J2SDK) (1.5 or later)
is installed on your machine at D:\Java\jdk1.5.0_05.
Press 'Next' to use this J2SDK or 'Browse' to locate another J2SDK.

NOTE: The Sun Java Wireless Toolkit 2.5 Beta has been tested using J2SDK 1.5

Destination Folder
D:\Java\jdk1.5.0_05

5. The WTK uses Quicktime to play AMR files. In this tutorial, you will not use any multimedia AMR files; threfore, you do not need to install Quicktime to run the examples. On encountering the screen in Figure 5, click **Next** even if you do not have Quicktime installed:
Figure 5. Establish the location of Quicktime


Sun Java Wireless Toolkit 2.5 Beta Setup

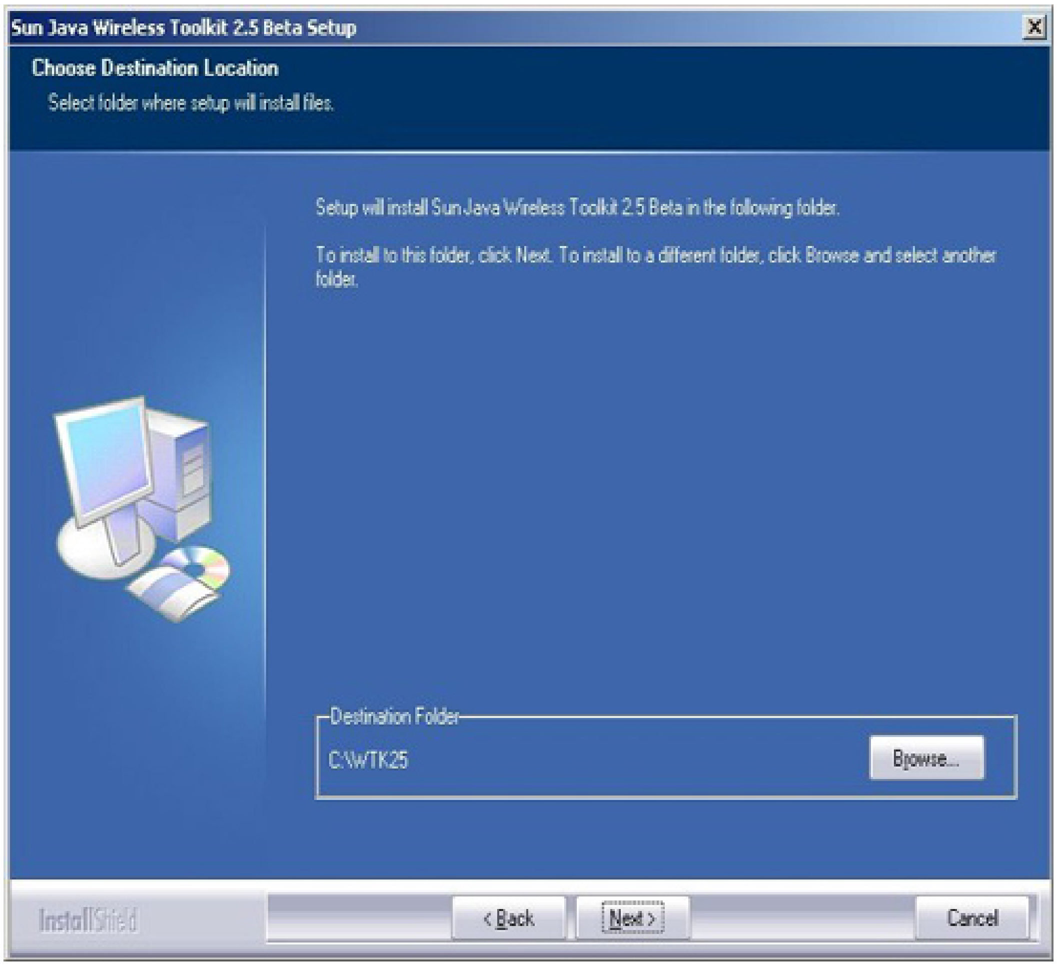
Quick Time Player Location

Unable to locate Quick Time Player. The Quick Time player is necessary to play AMR files. If you already have it please browse and locate the directory where it is installed. You can download the Quick Time player from http://www.apple.com/quicktime/download/standalone.html.

NOTE: The Sun Java Wireless Toolkit 2.5 Beta has been tested using Quick Time Player 7.

Destination Folder

6. Finally, the installation wizard asks you where it should install the Sun Java Wireless Toolkit. You can accept the default location or specify your own location, then click **Next** to install:
Figure 6. Choose the WTK installation location


Sun Java Wireless Toolkit 2.5 Beta Setup

Choose Destination Location

Select folder where setup will install files.

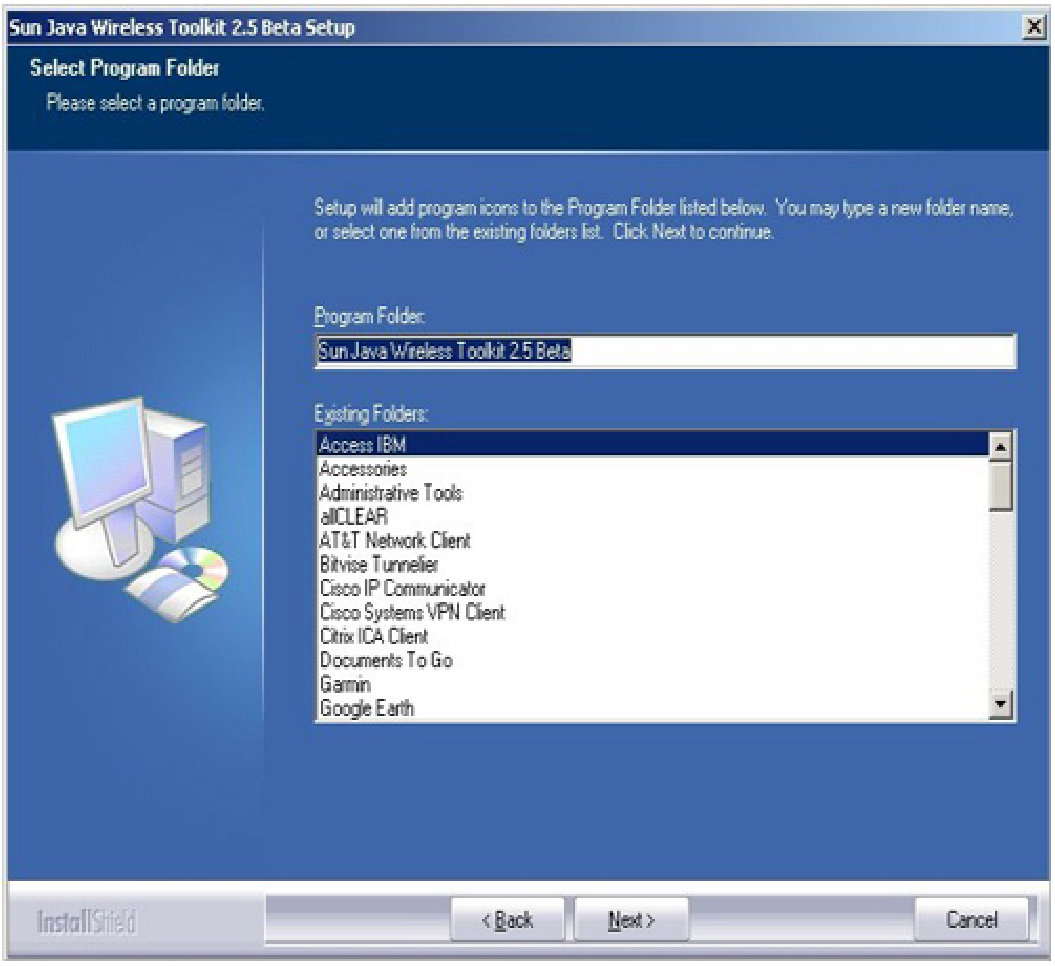
Setup will install Sun Java Wireless Toolkit 2.5 Beta in the following folder.

To install to this folder, click Next. To install to a different folder, click Browse and select another folder.

Destination Folder

C:\WTK25

7. Either specify the program folder or accept the default and click **Next**:
Figure 7. Choose a program folder or accept the default


Sun Java Wireless Toolkit 2.5 Beta Setup

Select Program Folder

Please select a program folder.

Setup will add program icons to the Program Folder listed below. You may type a new folder name, or select one from the existing folders list. Click Next to continue.

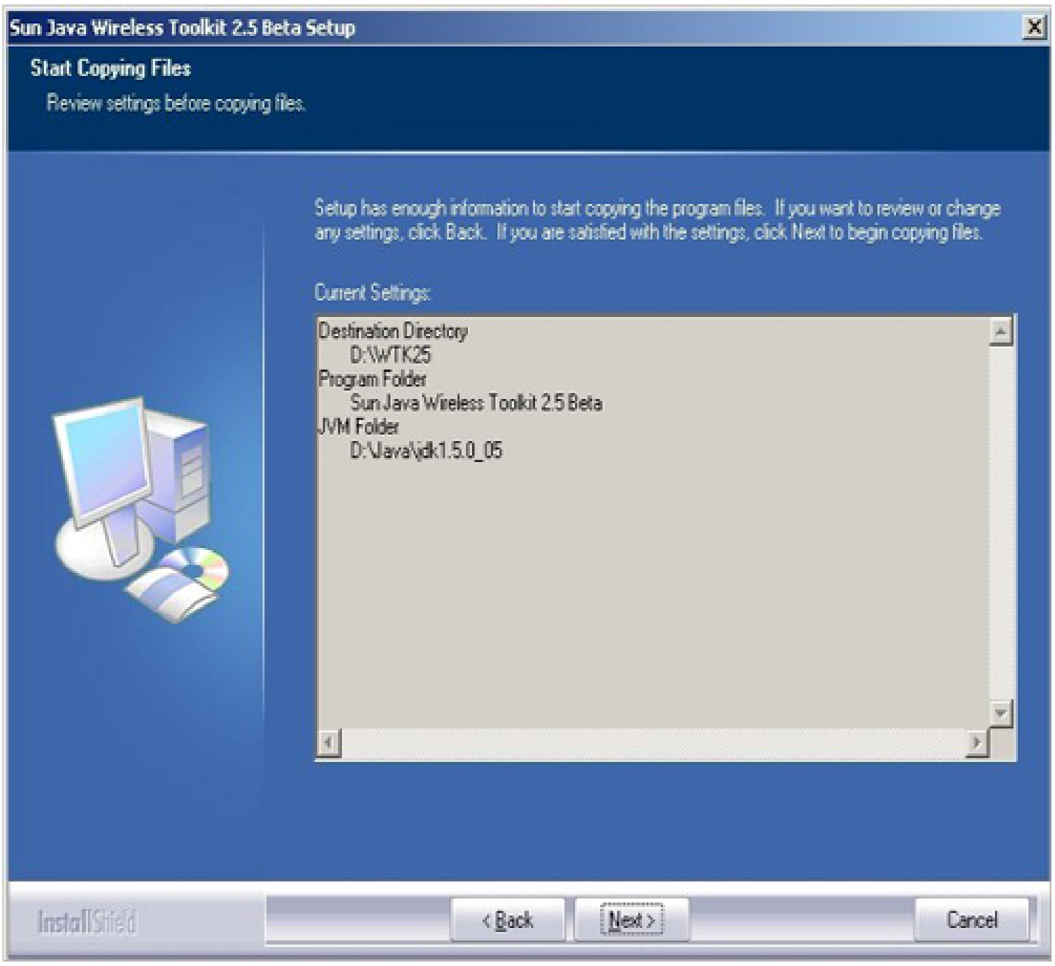
Program Folder:

Sun Java Wireless Toolkit 2.5 Beta

Existing Folders:

Access IBM
Accessories
Administrative Tools
allCLEAR
AT&T Network Client
Bitvise Tunnelier
Cisco IP Communicator
Cisco Systems VPN Client
Citrix ICA Client
Documents To Go
Garmin
Google Earth

8. To confirm your selections, the installation wizard provides you with a list of your choices. Select **Next** if you are happy with your choices.
Figure 8. Installation confirmed

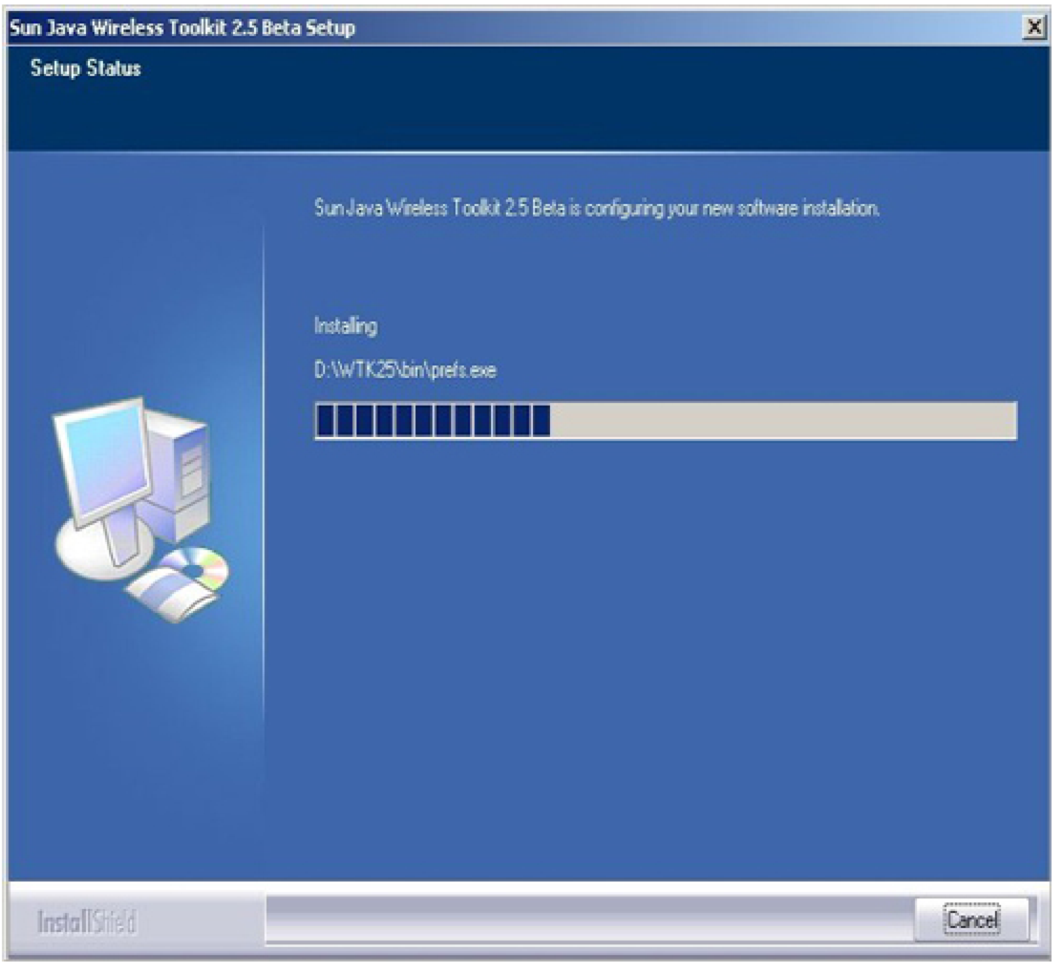

Sun Java Wireless Toolkit 2.5 Beta Setup
Start Copying Files
Review settings before copying files.
< Back
Next >
Cancel
InstallShield

Setup has enough information to start copying the program files. If you want to review or change any settings, click Back. If you are satisfied with the settings, click Next to begin copying files.

Current Settings:

```
Destination Directory
    D:\WTK25
Program Folder
    Sun Java Wireless Toolkit 2.5 Beta
JVM Folder
    D:\Java\jdk1.5.0_05
```

9. A panel showing the progress of copying the WTK files is presented.
Figure 9. WTK file copying in progress

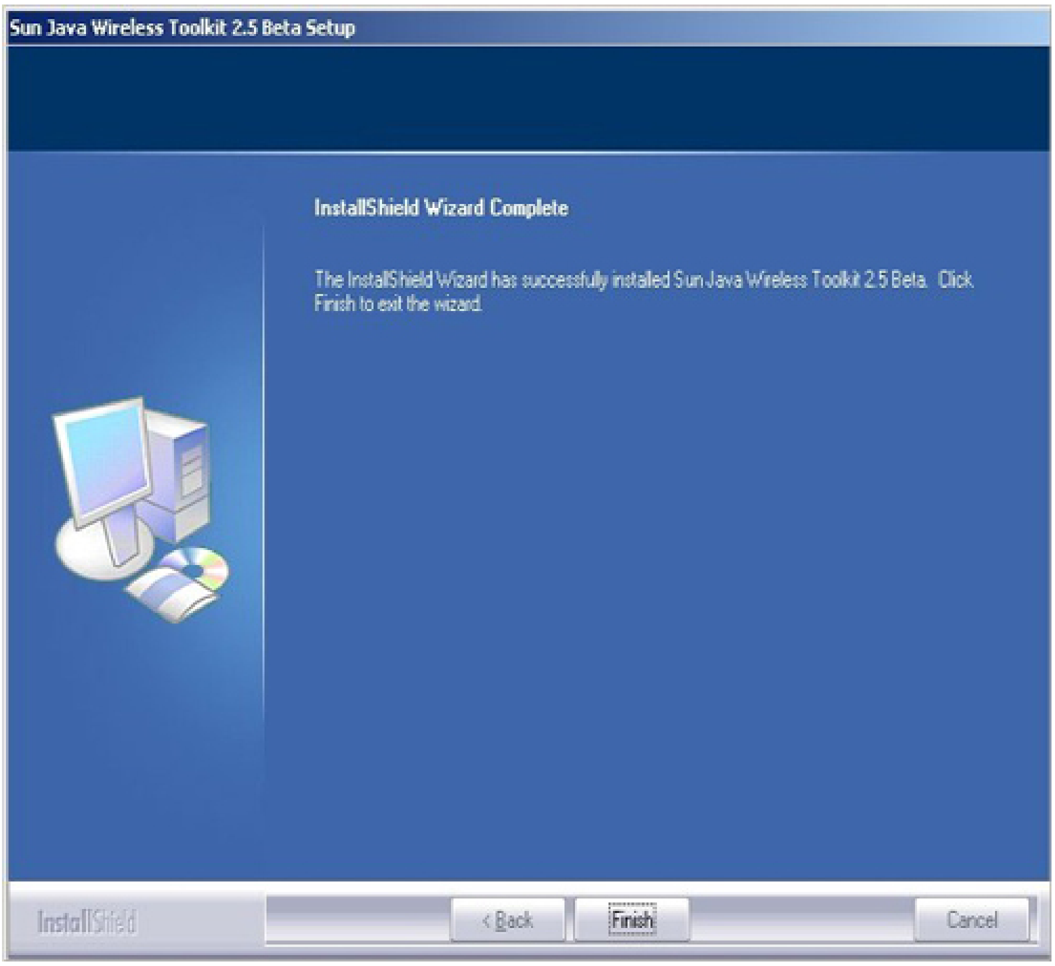


Sun Java Wireless Toolkit 2.5 Beta is configuring your new software installation.

Installing

D:\WTK25\bin\prefs.exe

10. The Sun Java Wireless Toolkit is successfully installed. Click **Finish** to complete.
Figure 10. Installation complete


Sun Java Wireless Toolkit 2.5 Beta Setup

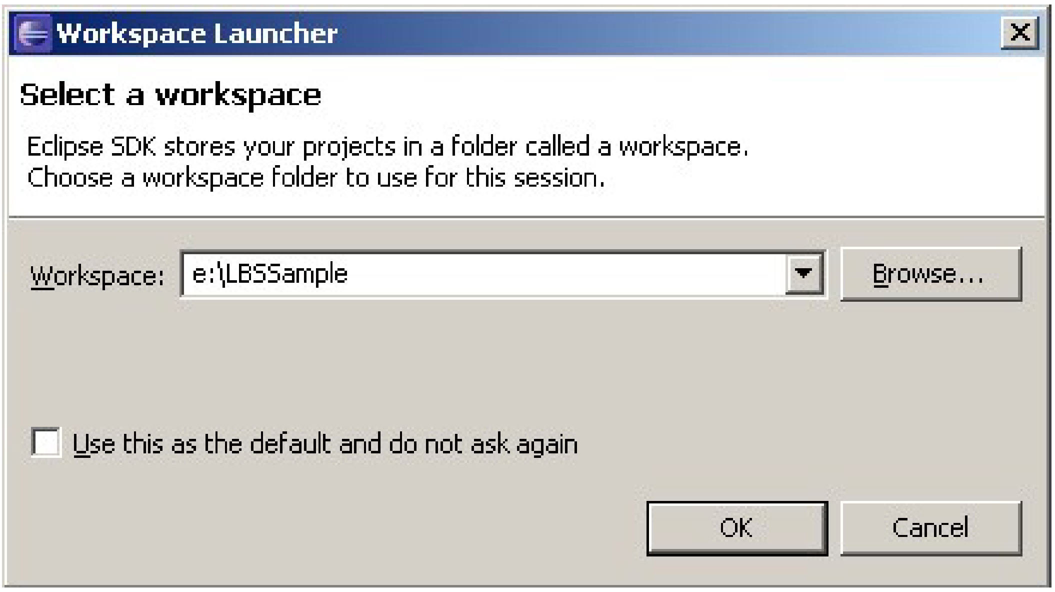
InstallShield Wizard Complete

The InstallShield Wizard has successfully installed Sun Java Wireless Toolkit 2.5 Beta. Click Finish to exit the wizard.

Install the EclipseME plug-in

EclipseME is an Eclipse plug-in that connects wireless toolkits to the Eclipse SDK. Integrating the Sun Java Wireless Toolkit with Eclipse is much easier with the EclipseME plug-in installed. EclipseME 1.5.5 is the most current at the time of this writing. The steps to install the plug-in and add it to your Eclipse IDE are:

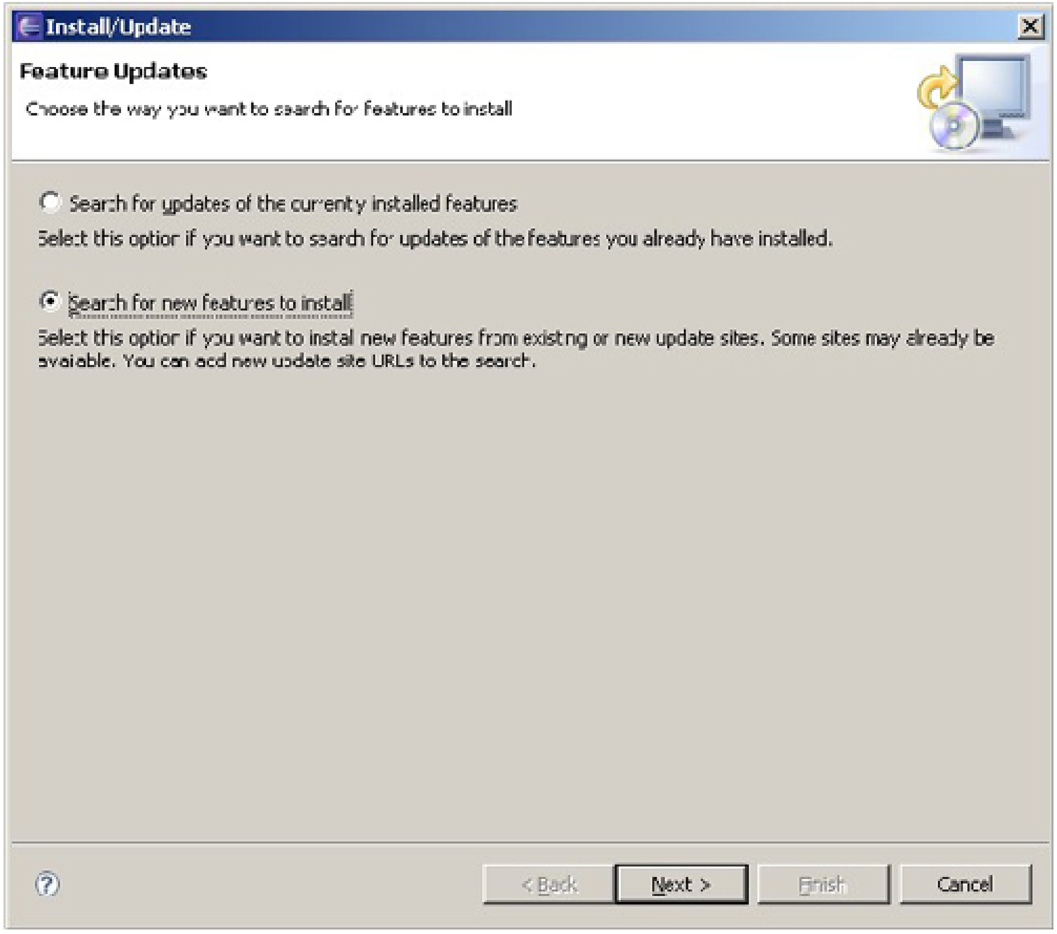
1. Open the Eclipse IDE from the location you extracted its files to. You can specify your own workspace location; I used *e:\LBSSample*:
   Figure 11. Select a workspace



Eclipse SDK stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: e:\LBSSample

2. In Eclipse, navigate to **Help > Software Updates > Find and Install**.
3. From the Install/Update window, select the **Search for new features to install** radio button and click **Next**.
Figure 12. Search for features to install



Feature Updates

Choose the way you want to search for features to install

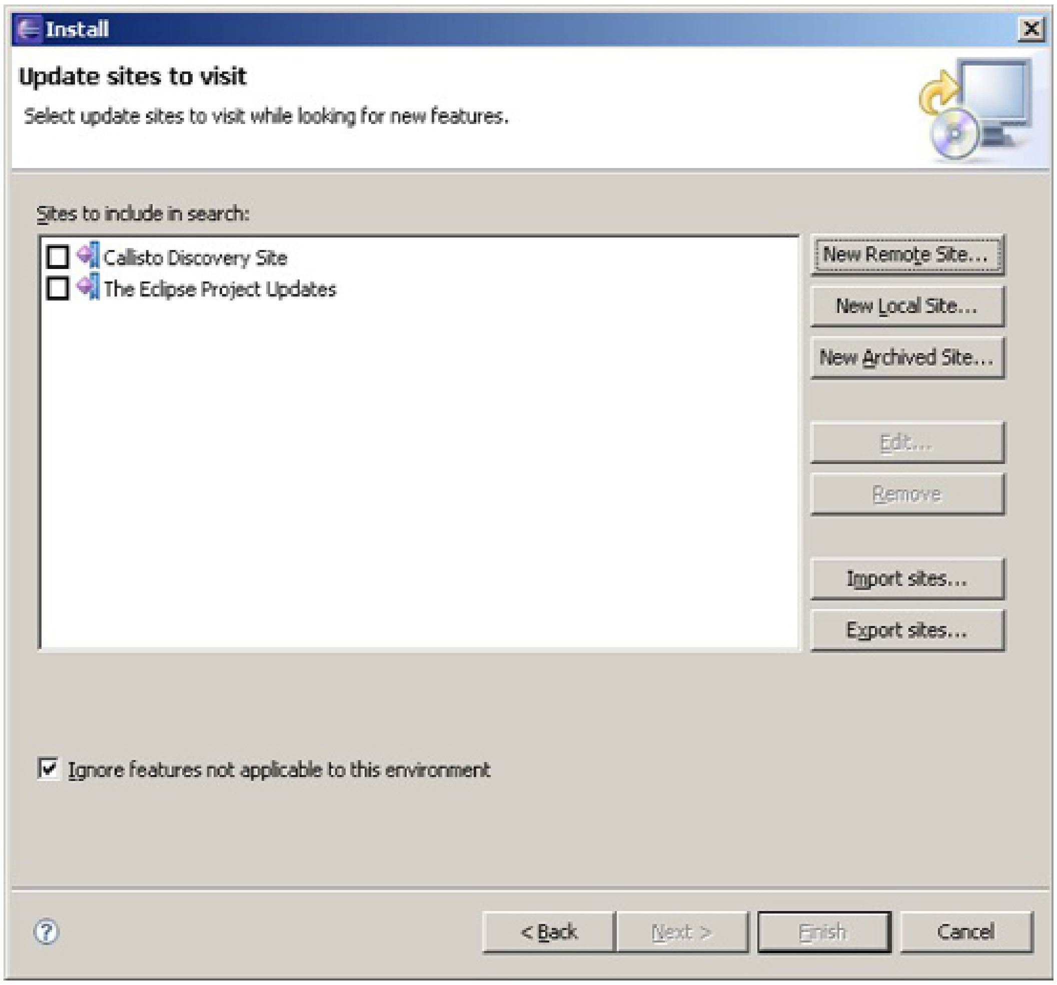
( ) Search for updates of the currently installed features

Select this option if you want to search for updates of the features you already have installed.

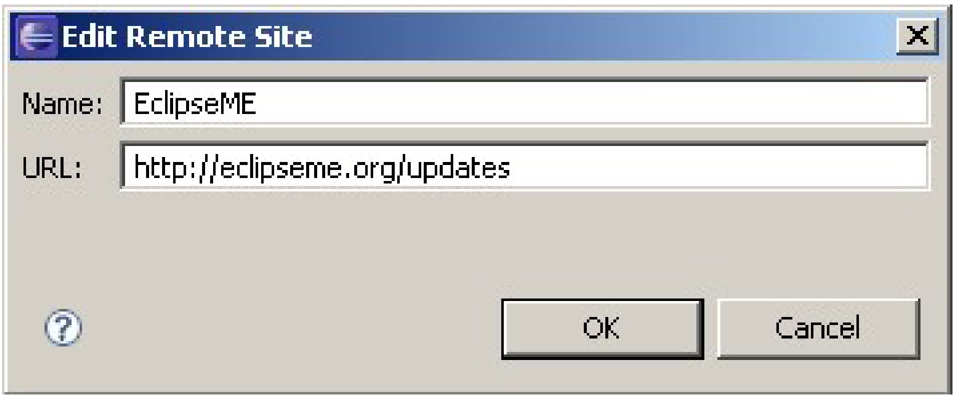
(•) Search for new features to install

Select this option if you want to install new features from existing or new update sites. Some sites may already be available. You can add new update site URLs to the search.

4. Click **New Remote Site** on the Installation screen. You'll see two fields, one for the name of the feature or plug-in you want to install, and one for its URL:
Figure 13. Select the update site



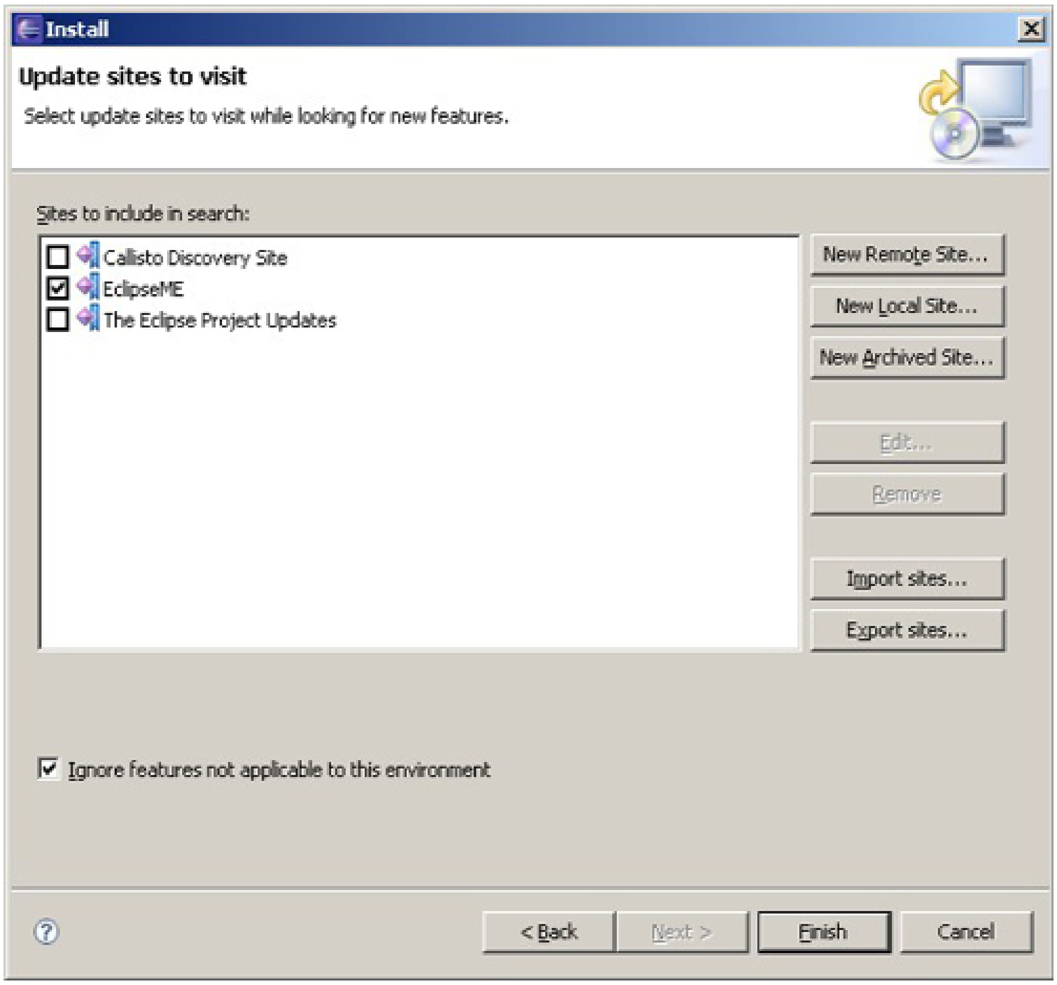
5. Enter `EclipseME` for the name and `http://eclipseme.org/updates` for the URL and click **OK**.
Figure 14. Enter the feature name and URL



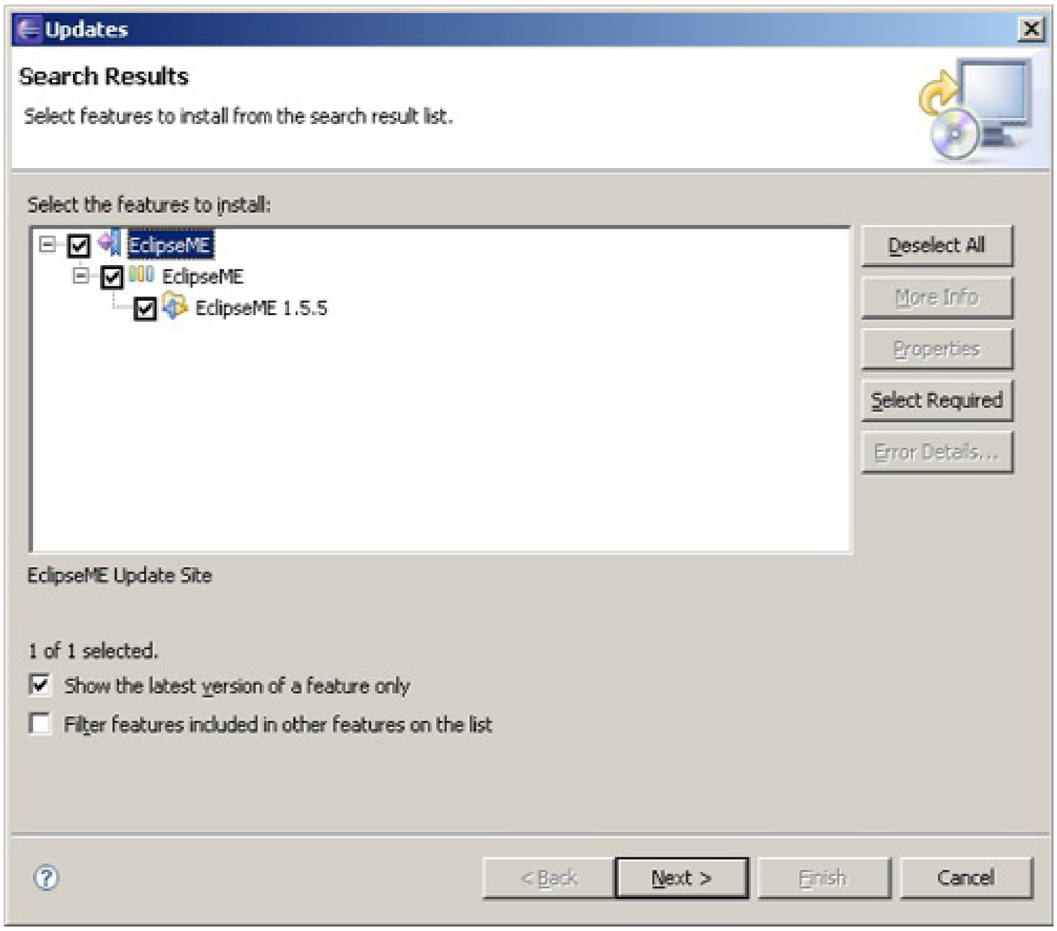
Name: EclipseME
URL: http://eclipseme.org/updates

6. Ensure the check box for "EclipseME" is selected and click **Finish** to search the EclipseME update site for the EclipseME plug-in.

Figure 15. Search for the EclipseME plug-in


Install
Update sites to visit
Select update sites to visit while looking for new features.
Sites to include in search:
☐ Callisto Discovery Site
☑ EclipseME
☐ The Eclipse Project Updates
New Remote Site...
New Local Site...
New Archived Site...
Edit...
Remove
Import sites...
Export sites...
☑ Ignore features not applicable to this environment
< Back
Next >
Finish
Cancel

7. The search results return the EclipseME plug-in. Select the EclipseME 1.5.5 checkbox and click **Next**.
Figure 16. Select the EclipseME plug-in from the search results



EclipseME
EclipseME
EclipseME 1.5.5

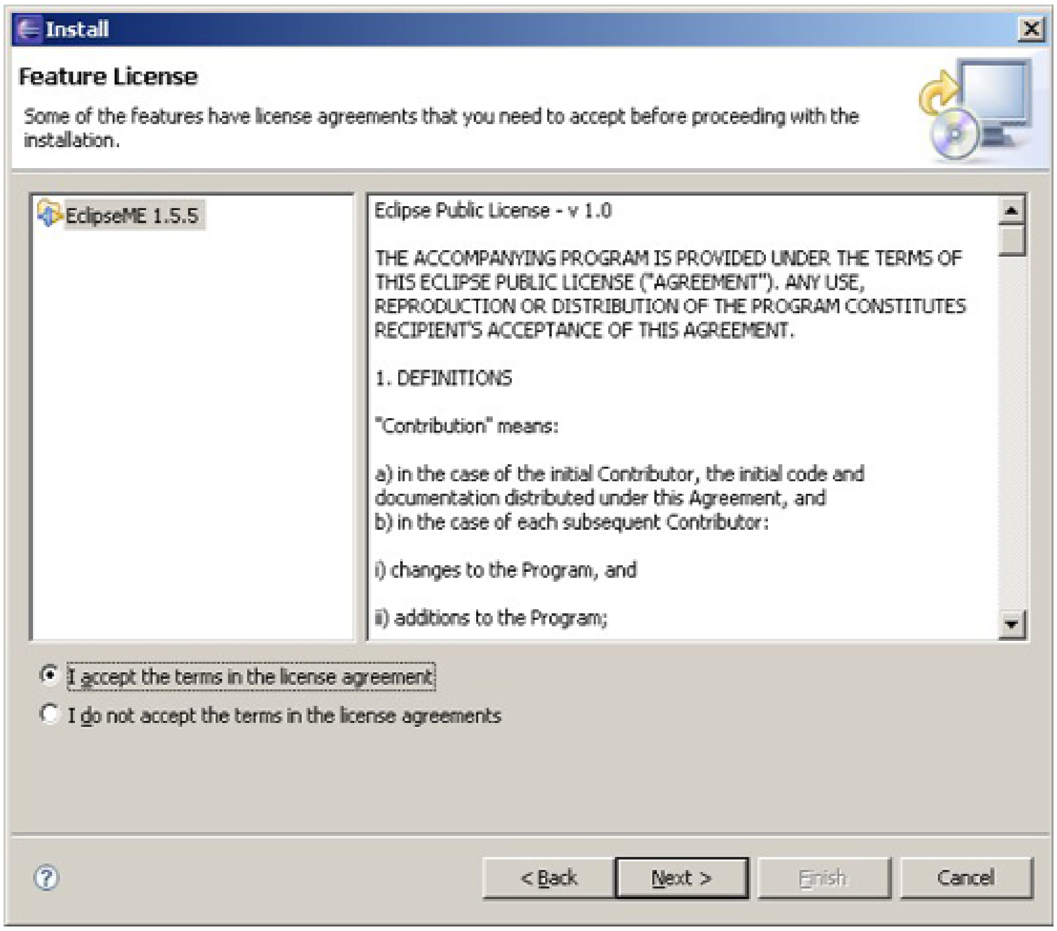
EclipseME Update Site

1 of 1 selected.

☑ Show the latest version of a feature only

☐ Filter features included in other features on the list

8. Read and accept the EclipseME license agreement and click **Next**.
Figure 17. Accept the EclipseME license agreement


Install

Feature License

Some of the features have license agreements that you need to accept before proceeding with the installation.

EclipseME 1.5.5

Eclipse Public License - v 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and
b) in the case of each subsequent Contributor:

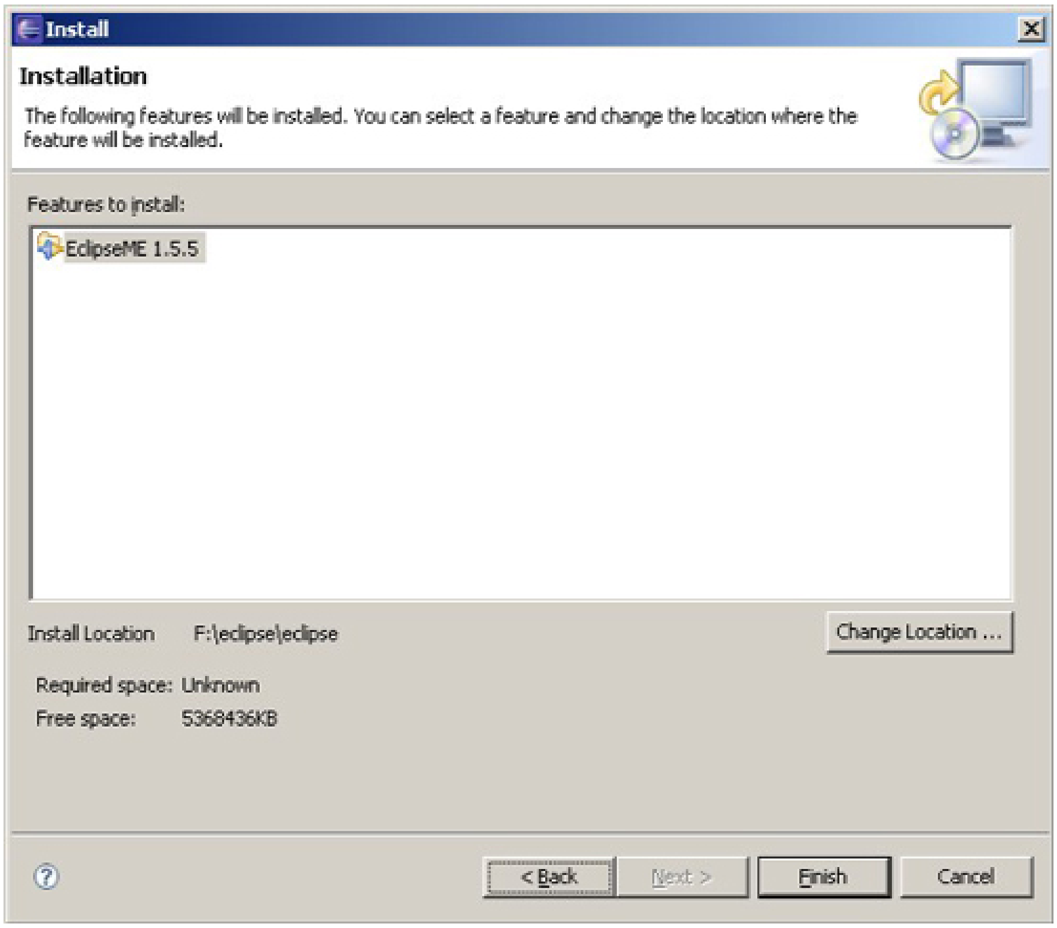
i) changes to the Program, and

ii) additions to the Program;

[✓ I accept the terms in the license agreement]

I do not accept the terms in the license agreements

9. By default the EclipseME feature is installed in the Eclipse SDK root. Click **Finish** to complete the installation.
Figure 18. Complete the installation



The following features will be installed. You can select a feature and change the location where the feature will be installed.

Features to install:

EclipseME 1.5.5

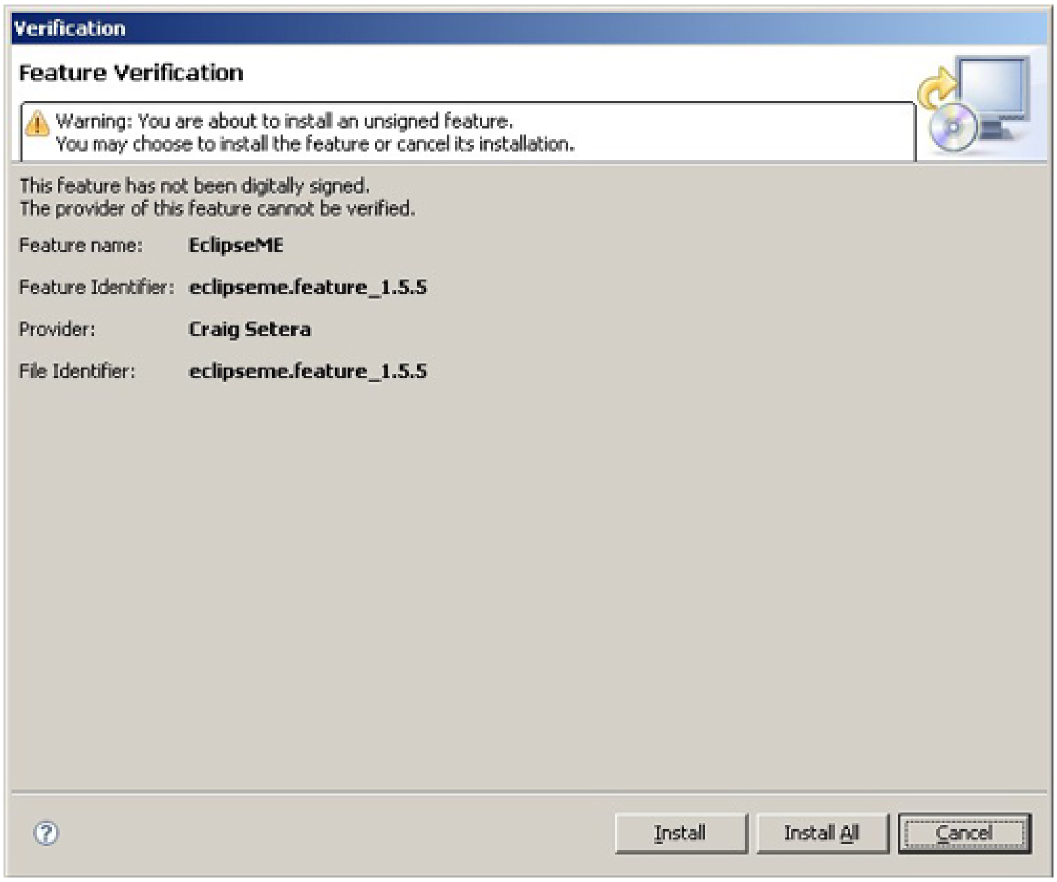
Install Location F:\eclipse\eclipse

Required space: Unknown

Free space: 5368436KB

10. You might get an unsigned warning about the EclipseME feature. Go ahead and click **Install** if your screen matches the one shown in Figure 19.

    Figure 19. An unsigned warning -- click Install



Verification

Feature Verification

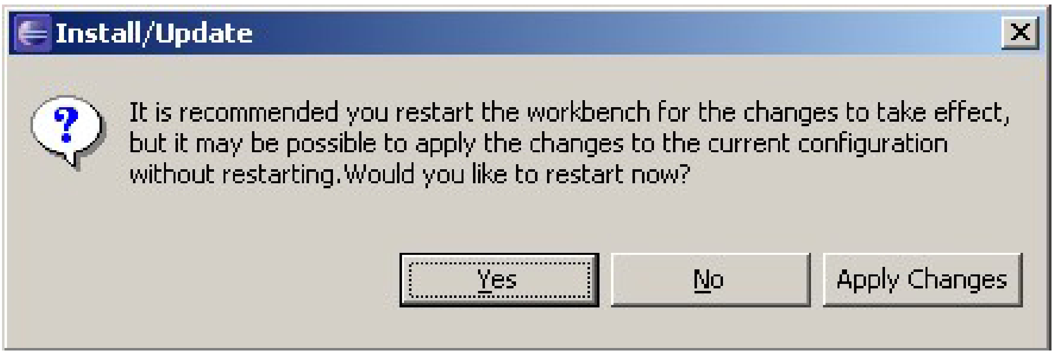
Warning: You are about to install an unsigned feature.
You may choose to install the feature or cancel its installation.

This feature has not been digitally signed.
The provider of this feature cannot be verified.

Feature name:	EclipseME
Feature Identifier:	eclipseme.feature_1.5.5
Provider:	Craig Setera
File Identifier:	eclipseme.feature_1.5.5

11. Installation of the EclipseME plug-in is complete. It is recommended that you click **Yes** to restart Eclipse.

Figure 20. Restart Eclipse



Install/Update

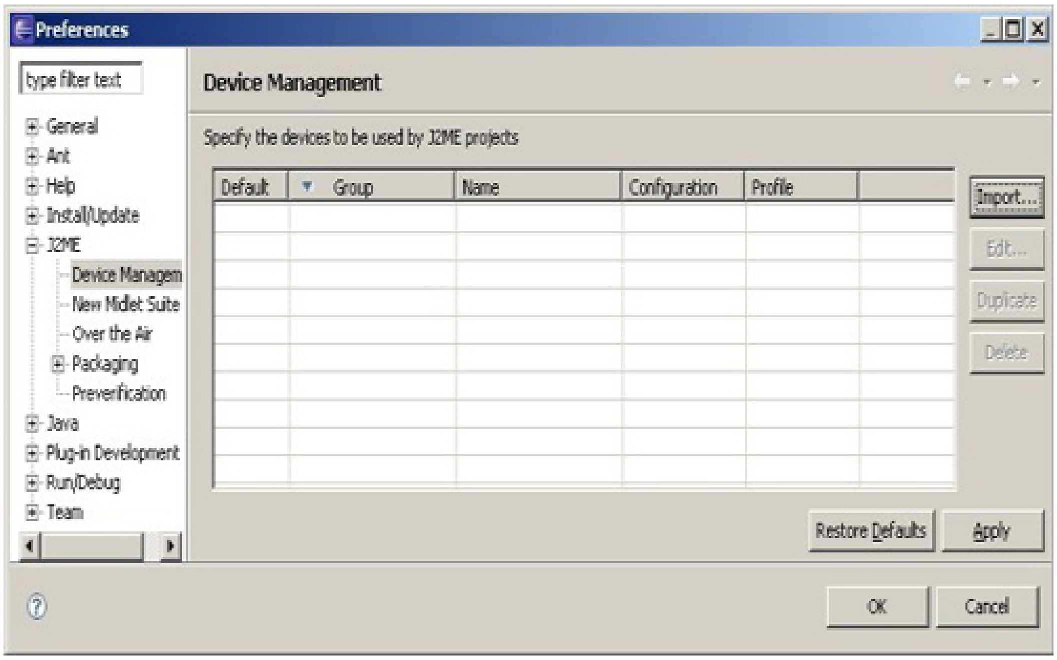
It is recommended you restart the workbench for the changes to take effect, but it may be possible to apply the changes to the current configuration without restarting.Would you like to restart now?

Configure the EclipseME feature and Eclipse SDK

Next, you will configure the EclipseME plug-in and Eclipse SDK for J2ME development. If you haven't already started Eclipse do so now. Then select **Preferences** from the Eclipse Window menu.

1. In the Eclipse Preferences window, expand J2ME and select Device Management, as shown in Figure 21. Click the Import button on the right side of the window.

Figure 21. The Eclipse Preferences window



2. Import the WTK devices:

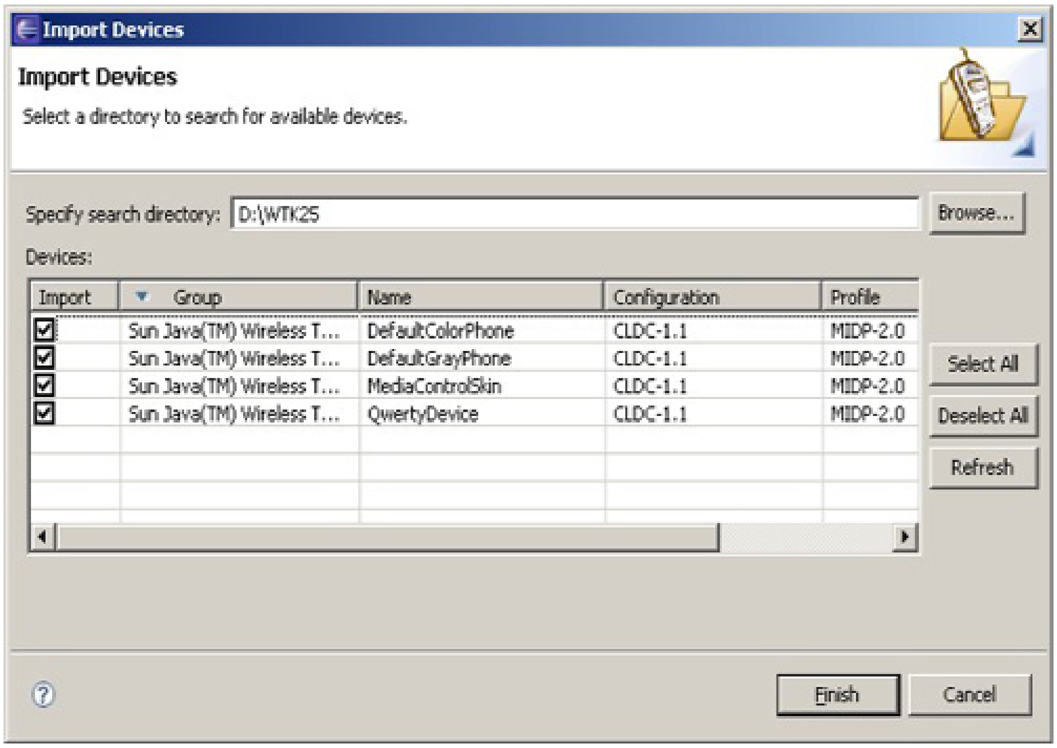
    1. Enter the root directory of the WTK 2.5 install. (I installed my WTK 2.5 into *d:\WTK25*).

    2. Click **Refresh**.

        - Your result should look like what you see in Figure 22.

    3. Click **Finish** to import the WTK devices. You don't need all these devices, but it doesn't hurt to have them.

Figure 22. J2ME device import list



Import Devices

Import Devices

Select a directory to search for available devices.

Specify search directory: D:\WTK25

Devices:

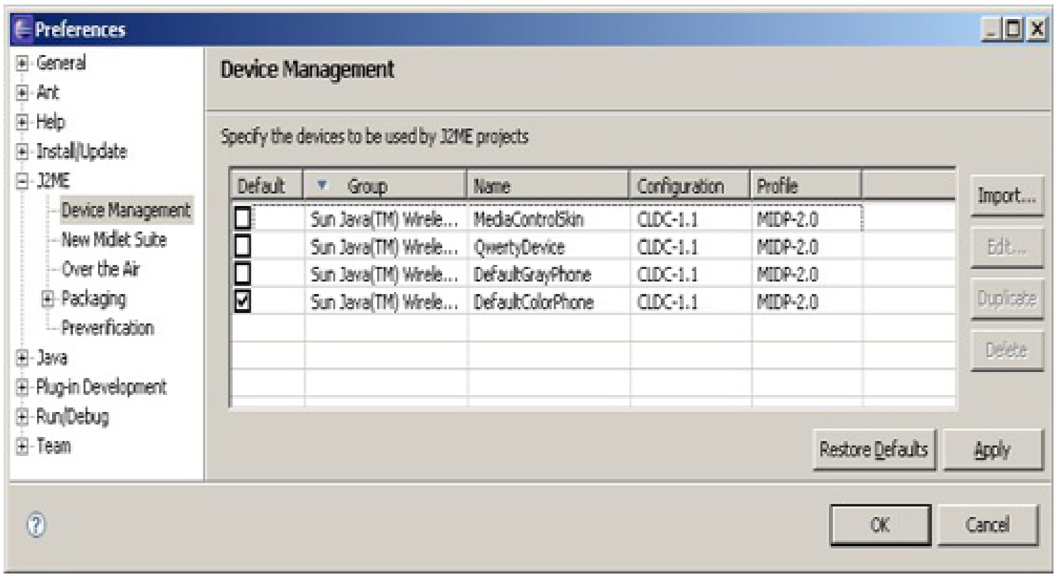
Import	Group	Name	Configuration	Profile
☑	Sun Java(TM) Wireless T...	DefaultColorPhone	CLDC-1.1	MIDP-2.0
☑	Sun Java(TM) Wireless T...	DefaultGrayPhone	CLDC-1.1	MIDP-2.0
☑	Sun Java(TM) Wireless T...	MediaControlSkin	CLDC-1.1	MIDP-2.0
☑	Sun Java(TM) Wireless T...	QwertyDevice	CLDC-1.1	MIDP-2.0

3. Configure the default device and apply the changes as follows:

    1. Select the DefaultColorPhone default check box as shown in Figure 23 below.

    2. Click **Apply** in the Device Management window to complete the WTK 2.5 device import.

Figure 23. Specify the devices to be used



Preferences

Device Management

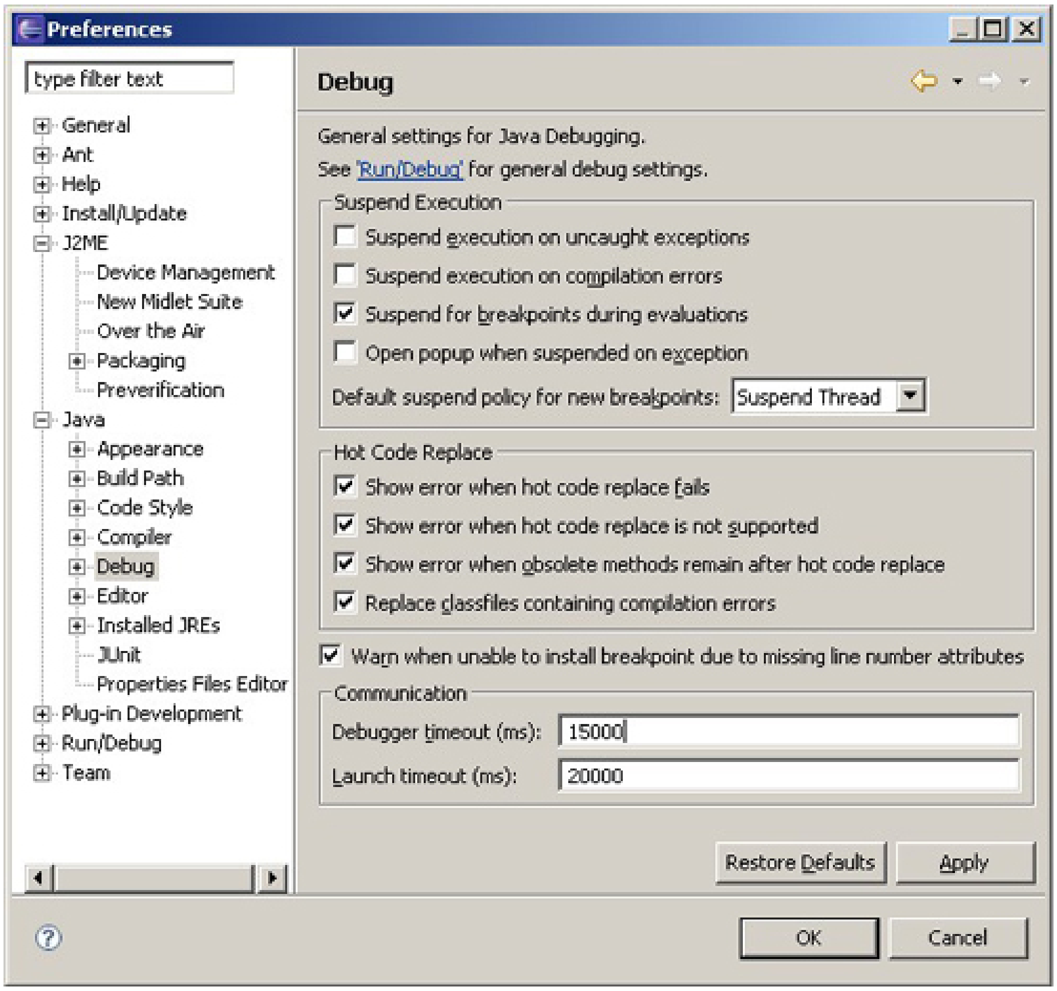
Specify the devices to be used by J2ME projects

Default	Group	Name	Configuration	Profile
☐	Sun Java(TM) Wirele...	MediaControlSkin	CLDC-1.1	MIDP-2.0
☐	Sun Java(TM) Wirele...	QwertyDevice	CLDC-1.1	MIDP-2.0
☐	Sun Java(TM) Wirele...	DefaultGrayPhone	CLDC-1.1	MIDP-2.0
☑	Sun Java(TM) Wirele...	DefaultColorPhone	CLDC-1.1	MIDP-2.0

4. Expand the "Java" item in the left pane and select **Debug**:

   1. Uncheck the **Suspend execution on uncaught exceptions** and **Suspend execution on compilation errors** options.
   2. Increase the "Debugger timeout (ms)" field to 15000.
   3. Click **OK** to save your changes and close the window.

Figure 24. Configure Java debugging



Preferences

Debug

General settings for Java Debugging.
See 'Run/Debug' for general debug settings.

Suspend Execution
- [ ] Suspend execution on uncaught exceptions
- [ ] Suspend execution on compilation errors
- [x] Suspend for breakpoints during evaluations
- [ ] Open popup when suspended on exception

Default suspend policy for new breakpoints: Suspend Thread

Hot Code Replace
- [x] Show error when hot code replace fails
- [x] Show error when hot code replace is not supported
- [x] Show error when obsolete methods remain after hot code replace
- [x] Replace classfiles containing compilation errors

- [x] Warn when unable to install breakpoint due to missing line number attributes

Communication
Debugger timeout (ms): 15000
Launch timeout (ms): 20000

You have successfully installed and configured the Eclipse IDE, the Sun Java Wireless Toolkit, and the EclipseME plug-in. With these technologies installed you will be able to follow the discussion and examples in this tutorial.

Section 3. The Location API (JSR 179)

Which CLDC version?
The Location API will not work on CLDC 1.0 devices because floating-point support had not yet been introduced with this version. The Location API works on CLDC 1.1x-compliant devices.

The Location API, also known as JSR 179, is an optional package to the Java ME libraries. The Location API provides you with the necessary functions to access the geographical location of a device, thus facilitating location-based application development. The platform minimum requirements to support the Location API are CLDC 1.1 and MIDP 2.0.

Using the Location API to determine the geographical location of a device is straightforward:

1. Verify that your target platform has the optional Location API installed.
2. Establish a `LocationProvider` instance.
3. Capture location information.

In addition, there are two optional steps:

1. Create a `LocationListener` to listen for location update events.
2. Create a `ProximityListener` to notify the application when the device is within a defined range of a specific location.

I'll go through the required steps first, then discuss the optional ones.

Step 1: Verify the Location API package

To use the Location API to acquire location coordinates, you must verify that a device has the optional Location API installed. Check the system property `microedition.location.version` as shown in Listing 1.

Listing 1. Location package discovery

```
if (System.getProperty("microedition.location.version") == null) {
    /* Location Package is NOT installed */
} else {
    /* Location Package is installed */
}
```

Step 2: Establish a LocationProvider instance

To retrieve coordinates you must first establish a `LocationProvider` instance specifying a `Criteria` object. As the name implies, the `Criteria` object contains the specific criteria you have determined the `LocationProvider` must match. Listing 2 shows how to establish a `LocationProvider` instance. The default settings for the `Criteria` object are shown in Table 1.

Listing 2. Establishing a LocationProvider instance

```


LocationProvider lp;
Criteria cr = new Criteria();

/*  You may also set cr to null denoting the least restrictive
criteria for a provider to meet  */
//optionally set specific criteria ...

try {
  lp = LocationProvider.getInstance(cr);
} catch (LocationException le) {
  //error
}
```

Table 1. Default values for the Criteria class

Field	Value	Option
Horizontal accuracy	NO_REQUIREMENT	int (meters)
Vertical accuracy	NO_REQUIREMENT	int (meters)
Preferred response time	NO_REQUIREMENT	int (milliseconds)
Preferred power consumption	NO_REQUIREMENT	int (POWER_USAGE_{HIGH/MEDIUM/LOW})
Cost allowed	true (service provider allowed to charge device user)	boolean
Speed and course required	false	boolean
Altitude required	false	boolean
Address info required	false	boolean

Step 3: Capture location information

The QualifiedCoordinates class

The `QualifiedCoordinates` class subclasses the `Coordinates` class and thus inherits the following methods:

- `azimuthTo(Coordinates)`
- `convert(double, int)`
- `convert(double, int)`
- `distance(Coordinates)`
- `getAltitude()`
- `getLatitude()`
- `getLongitude()`
- `setAltitude(float)`
- `setLatitude(double)`
- `setLongitude(double)`

Next, you'll use the `Location.getQualifiedCoordinates()` method to retrieve a `QualifedCoordinates` object from your target device. With the `QualifedCoordinates` object, you can retrieve the longitude, latitude, and altitude in WGS84 datum using the methods `QualifiedCoordinates.getLongitude()`, `QualifiedCoordinates.getLatitude()`, and `QualifiedCoordinates.getAltitude()`.

After you have a target device's coordinates, you can use future coordinate-capture events to calculate its heading/azimuth (in degrees) and distance (in meters) using `Coordinates.azimuthTo(Coordinates to)` and `Coordinates.distance(Coordinates to)`. You can also retrieve a device's speed in meters-per-second (m/s) by using the method `Location.getSpeed()`.

In summary, you can now capture the location and speed of a target device. Using this acquired data, you can calculate the device's distance and heading. Listing 3 shows the acquisition of a device's location and speed along with the calculation of its directional heading/azimuth and distance traveled.

Listing 3. Location and speed acquisition with distance and heading calculation

```
try {
  // Get the QualifiedCoordinates
  QualifiedCoordinates c = loc.getQualifiedCoordinates();
  double longitude = c.getLongitude();
  double latitude = c.getLatitude();
  float altitude = c.getAltitude();
```

```
   /* if this is the first acquisition, set the current coordinates
    as the old coordinates and continue, otherwise we will calculate
    the distance traveled and heading */

   if (null == oldCoordinates){
     oldCoordinates = new Coordinates(latitude,longitude,altitude);
   }
   else{
     // Calculate the total distance traveled
     distance += c.distance(oldCoordinates);

     curCoordinates = new Coordinates(latitude,longitude,altitude);

     // Calculate the heading
     float azimuth = oldCoordinates.azimuthTo(curCoordinates);

     oldCoordinates.setAltitude(altitude);
     oldCoordinates.setLatitude(latitude);
     oldCoordinates.setLongitude(longitude);
   }

 } catch (Exception e) {
   //  not able to retrieve location information
 }
```

The following optional steps can enhance or enable the functions of certain types of applications.

Optional step 1: Setting the LocationListener

setLocationListener IllegalArgumentException
The `setLocationListener(LocationListener listener, int interval, int timeout, int maxAge)` method throws an `IllegalArgumentException` if any of the following parameter constraints are met:

1. interval < -1
2. interval != -1 and
    1. timeout > interval
    2. maxage > interval
    3. timeout < 1 and timeout != -1
    4. maxage < 1 and maxage != -1

Tracing routes and displaying live information to a device end-user requires capturing or updating location details at timed intervals. The Location API provides a `LocationListener` interface for this purpose. The `LocationListener` is registered with a `LocationProvider`, with a defined interval at which it attempts to capture location details.

To register a `LocationListener`, use the method `LocationProvider.setLocationListener(LocationListener listener, int interval, int timeout, int maxAge)`.

LocationListener methods

The `LocationListener` has two methods:

1. `locationUpdated(LocationProvider provider, Location location)`
2. `providerStateChanged(LocationProvider provider, int state)`

The `locationUpdated(LocationProvider provider, Location location)` method is called by the `LocationProvider` at the defined interval for the registered `LocationListener` providing the updated location.

The `providerStateChanged(LocationProvider provider, int state)` method is called on state changes in the `LocationProvider`. The states include: `AVAILABLE`, `TEMPORARILY_UNAVAILABLE`, and `OUT_OF_SERVICE`.

Optional step 2: Setting the ProximityListener

The `ProximityListener` interface provides the necessary functions to listen and notify your application if the device enters a defined proximity. The proximity is defined as a location `Coordinates` and a radius (positive value of type *float*).

To register a `ProximityListener`, use the `LocationProvider.addProximityListener(ProximityListener listener, Coordinates coordinates, float proximityRadius)` method.

ProximityListener methods

The `ProximityListener` has two methods:

1. `monitoringStateChanged(boolean isMonitoringActive)`
2. `proximityEvent(Coordinates coordinates, Location location)`

About proximity events
When the `proximityEvent` method is called by the listener, the

listener is unregistered. An application must re-register the proximity listener to be notified of additional proximity events for defined coordinates.

The `monitoringStateChanged(boolean isMonitoringActive)` method is called by the listener to notify the application of a change in state. The change in state is a boolean meaning that the `ProximityListener` has either become active or inactive. In the inactive state, the listener is still registered but is inactive for some reason determined by the provider.

The `proximityEvent(Coordinates coordinates, Location location)` method is called by the listener, notifying the application that the device has entered the defined proximity boundary. With this notification, the coordinates of the defined proximity are given along with the current coordinates of the device.

Section 4. Develop a location-based application

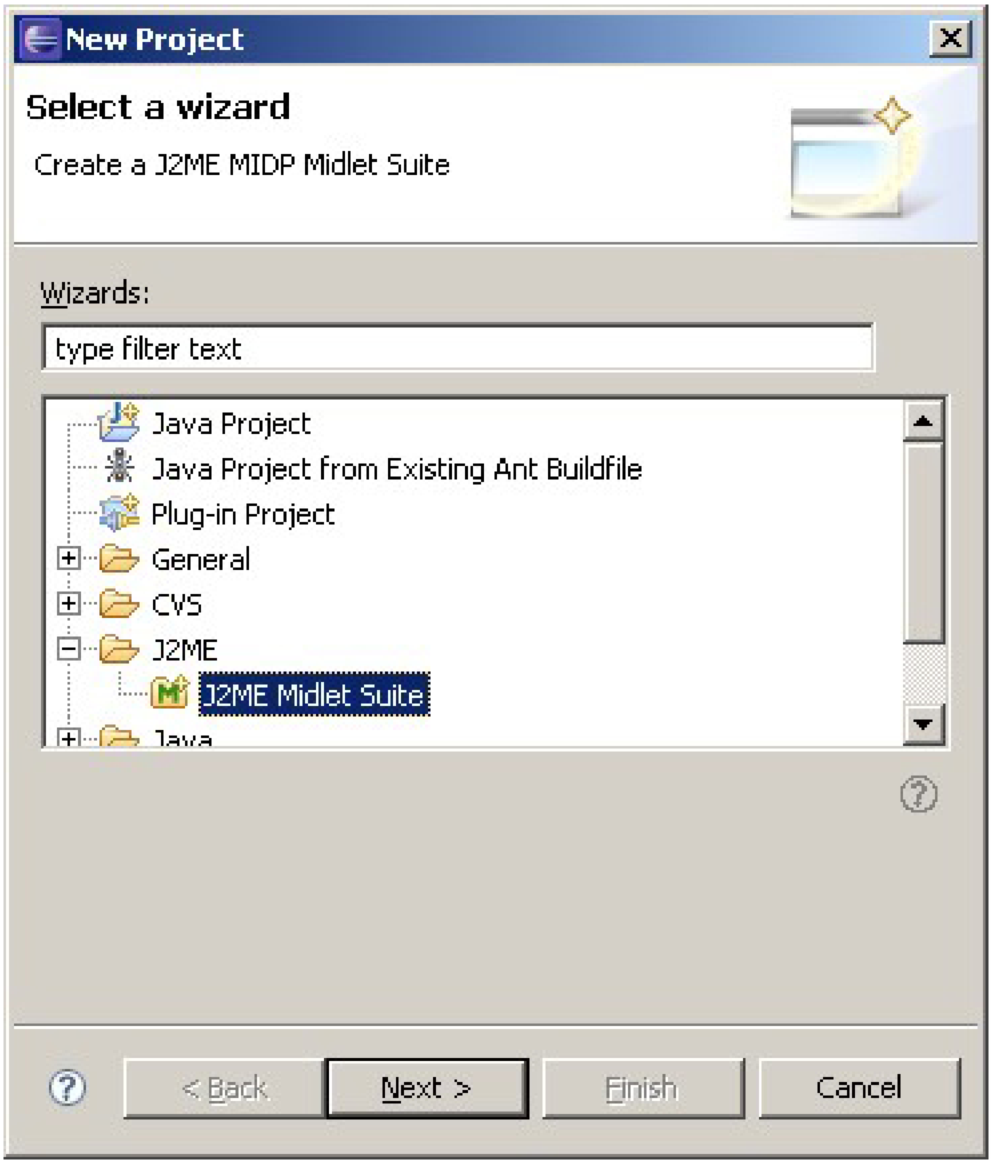
In this section you use the Eclipse IDE to set up a new Java ME project, import the LBSSample MIDlet source, and then walk through the steps of developing a simple location-based application.

Create the J2ME project

Create a Java ME project for building the example application as follows:

1. From the Eclipse File menu, select **New > Project...**
   - Expand the *J2ME* item in the list, select J2ME Midlet Suite, and click **Next**.

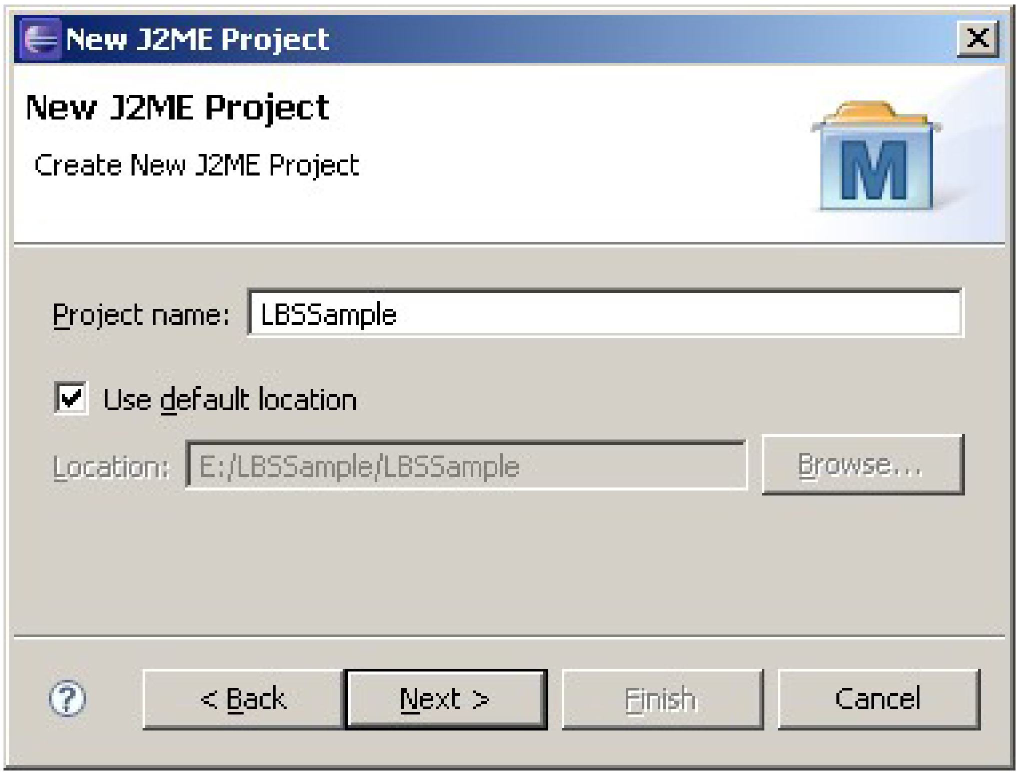
   Figure 25. Select the J2ME MIDlet suite



2. Enter `LBSSample` in the project wizard's Project Name field and click **Next**.

   - You can either leave the "Use default location" box checked or uncheck the selection and define your own project-location path.

Figure 26. Name the project



New J2ME Project

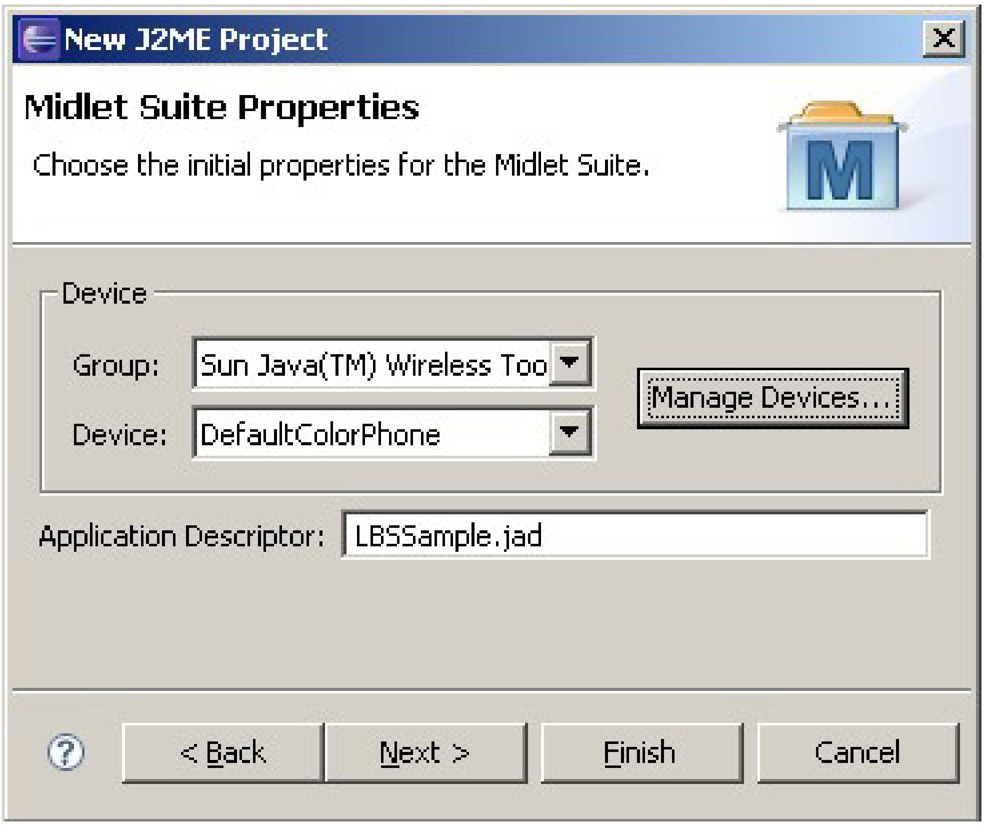
Create New J2ME Project

Project name: LBSSample

[✓] Use default location

Location: E:/LBSSample/LBSSample

3. No further configuration is required, click **Finish** to complete the process of creating the new project.
Figure 27. Click Finish to complete the process



New J2ME Project

Midlet Suite Properties

Choose the initial properties for the Midlet Suite.

Device

Group: Sun Java(TM) Wireless Too

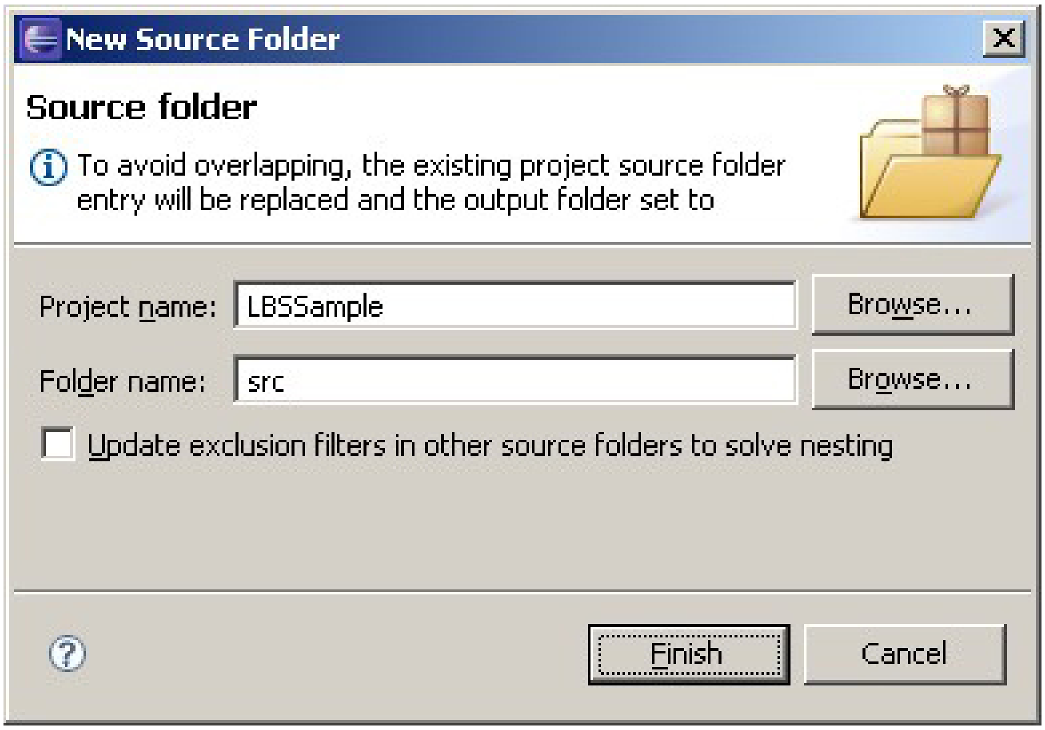
Device: DefaultColorPhone

Application Descriptor: LBSSample.jad

Create a source folder

Next, you'll create a source folder in the J2ME LBSSample project:

1. Right-click on the LBSSample project and select **New > Source Folder**
2. Enter `src` in the Folder Name field.
3. Click **Finish**.
   Figure 28. Create a source folder


New Source Folder

Source folder

To avoid overlapping, the existing project source folder entry will be replaced and the output folder set to

Project name: LBSSample

Folder name: src

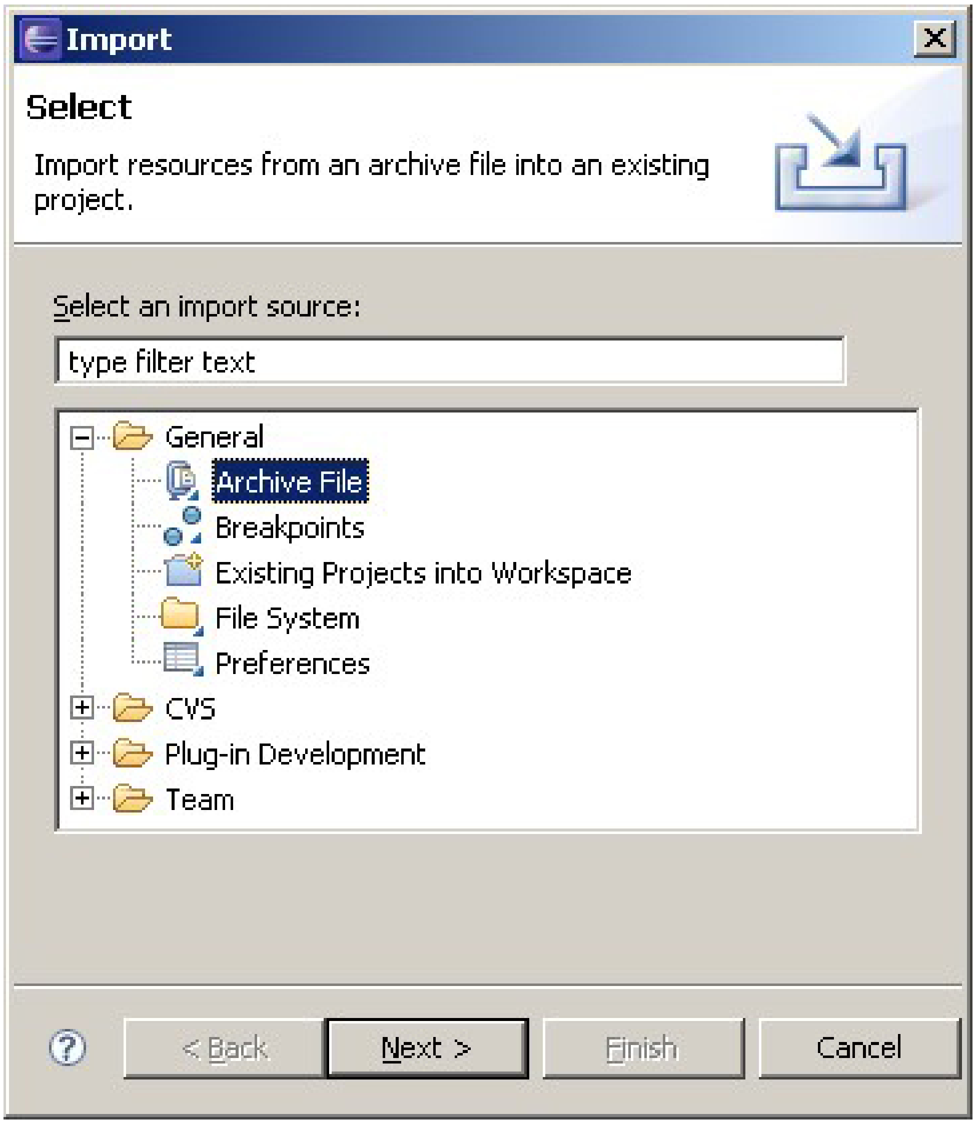
Update exclusion filters in other source folders to solve nesting

Import the LBS MIDlet class

To run an application on a Java embedded device, you must write a MIDlet. For the purpose of this tutorial, you will simply import the tutorial MIDlet class LBSMIDlet into the LBSSample project. After you have imported the LBSMIDlet, I will walk you through its implementation of the JSR 179 Location API.

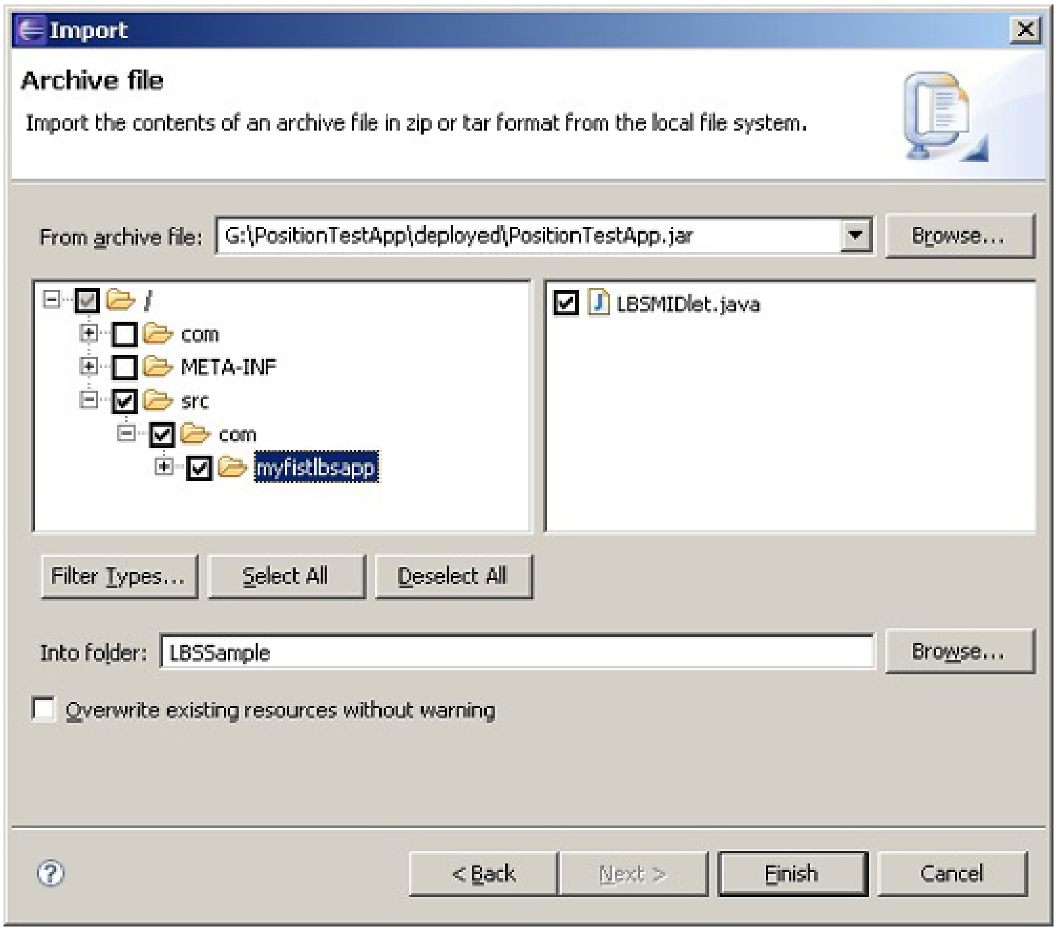
Import the base LBSSample MIDlet class as follows:

1. Right-click on the LBSSample project and select **Import**.
2. From the "Select an import source" list, expand **General** and select **Archive File**.
3. Click **Next**.
   Figure 29. Select the Import source



4. Import the source folder from the archive:
   - Enter the path where you unzipped the tutorial zip file and `PositionTestApp.jar`.
   - Expand the archive tree and de-select the "com" and "META_INF" checkboxes as shown in Figure 30.
   - Click **Finish** to import the `LBSMIDlet`.

Figure 30. Import the LBSMIDlet



The LBSSample MIDlet

The LBSSample MIDlet is programmed to do the following:

- Verify that the target device supports the Location API.
- Create a `LocationProvider` instance with default `Criteria` for the target device.
- Set a `LocationListener` and `ProximityListener` for the device.
- Capture location information and calculate the results.

In the next sections you'll see how each of these functions is implemented in the MIDlet.

Verify Location API support

The LBSSample MIDlet uses the system property `microedition.location.version` to discover if a target device supports the Location API, as shown in Listing 4.

Listing 4. Check for Location API support

```
public boolean hasLocationAPI(){

        if (System.getProperty("microedition.location.version") == null) {
                return false;
        } else {
            return true;
        }
}
```

Create a LocationProvider instance

For devices supporting the Location API, the LBSSample app then creates a `LocationProvider` instance with default `Criteria`, as shown in Listing 5.

Listing 5. Create a LocationProvider instance

```
void createLocationProvider() {
   if ( lp==null ) {

       Criteria cr = new Criteria();

       try {
           lp = LocationProvider.getInstance(cr);
       } catch (LocationException le) {
           le.printStackTrace();
       }
   }
}
```

Set the LocationListener and ProximityListener

What is a waypoint?
A *waypoint* is a reference point with specific latitude, longitude, and altitude coordinates.

The application sets the `LocationListener` and `ProximityListener` when a device end-user elects to start the application or create a waypoint. Code for this is displayed in **bold** in Listing 6.

Listing 6. Setting the LocationListener and ProximityListener

```
public void commandAction(Command command, Displayable displayable) {

  if(command == exitCommand)
     {
       destroyApp(false);
       notifyDestroyed();
```

```
      }
   else if(command == startCommand){
                      Thread getLocationThread = new Thread() {
                      public void run(){
                      setListener();
                              }
                      };
                      getLocationThread.start();
                      }
   else if (command == towaypointCommand){
         display.setCurrent(waypointForm);
         }
   else if (command == waypointCommand){
                      wplat = Double.valueOf(((
                      TextField)waypointForm.get(waylat)).getString()).doubleValue();
                      wplon = Double.valueOf(((
                      TextField)waypointForm.get(waylon)).getString()).doubleValue();
                      wpalt = Double.valueOf(((
                      TextField)waypointForm.get(wayalt)).getString()).doubleValue();
                      wprad = Double.valueOf(((
                      TextField)waypointForm.get(wayrad)).getString()).doubleValue();
                      final Coordinates waypoint = new Coordinates((float) wplat,(float)
 wplon,
                      (float) wpalt);
                      Thread getProximityThread = new Thread() {
                      public void run(){
                      setProximity(waypoint,wprad);
                      }
                      };
                      getProximityThread.start();
                      }
   else if (command == tolocationCommand){
         display.setCurrent(locationForm);
         }
 }
```

LocationListener settings

The `LocationListener` is set with a listening interval of two seconds. The timeout and maxage parameters are set to the provider defaults. You can see the `LocationListener` settings in Listing 7.

Listing 7. The setListener method

```
 private void setListener(){
         lp.setLocationListener(this,2,-1,-1);
 }
```

ProximityListener settings

The waypoint and radius are specified by the end-user, then passed to the `setProximity` method to enable the `ProximityListener`. In Listing 8 the `ProximityListener` is added to the provider.

Listing 8. The setProximity method

```
 private void setProximity(Coordinates waypoint, double wprad){
  try {
    LocationProvider.addProximityListener(this,waypoint,(float) wprad);
  } catch (LocationException e) {
    e.printStackTrace();
  }
}
```

Best practices
It is recommended that you use a separate thread for acquiring location coordinates and processing the results, in order to avoid any blocked threads for future event triggers.

Receiving location events

The `LocationListener` provides location updates using its `locationUpdated` method, as shown in Listing 9.

Listing 9. The locationUpdated method

```
public void locationUpdated(final LocationProvider locProvider,
final Location loc) {

  Thread getLocationThread = new Thread(){
    public void run(){
      getLocation(locProvider,loc);
    }
  };
  getLocationThread.start();
}

private void getLocation(LocationProvider locProvider, Location
loc){
  getLocation(loc);
  ((StringItem)locationForm.get(proximity)).setLabel("You are
outside the " +
      wprad +  " meter radius of the defined zone");
}
```

Receiving proximity events

Re-register the ProximityListener
`ProximityListener` registration is cancelled with each `proximityEvent`. To continue receiving `ProximityEvent` notifications, re-register the `ProximityListener`, as shown in **bold** in Listing 10.

The `ProximityListener` provides an event using the `proximityEvent` method in Listing 10. Note that `proximityEvent` uses a separate thread to avoid any future thread blocking.

Listing 10. The proximityEvent method

```
public void proximityEvent(final Coordinates coordinates, final
Location loc) {
  Thread getLocationThread = new Thread(){
    public void run(){
                  getLocation(coordinates,loc);
                }
        };
        getLocationThread.start();
}

private void getLocation(Coordinates c, Location loc){
        getLocation(loc);
        ((StringItem)locationForm.get(proximity)).setLabel("You
are " +
loc.getQualifiedCoordinates().distance(c) +
                        " meters from (lat/long/alt)" +
c.getLatitude()+"/"+c.getLongitude()+"/"+c.getAltitude());
        setProximity(c,wprad);
}
```

Location coordinates and calculations

The `getLocation` method, shown in Listing 11 is used to get location coordinates and calculate a device's distance and azimuth. All details are then updated on the J2ME device screen.

Listing 11. The getLocation method

```
private void getLocation(Location loc){
  try {
    QualifiedCoordinates c = loc.getQualifiedCoordinates();
    if (null == oldCoordinates){
      oldCoordinates = new Coordinates(c.getLatitude(),c.getLongitude(),c.getAltitude());
    }
    else{
      distance += c.distance(oldCoordinates);
      curCoordinates = new Coordinates(c.getLatitude(),c.getLongitude(),c.getAltitude());
      az = (int)oldCoordinates.azimuthTo(curCoordinates);
      oldCoordinates.setAltitude(c.getAltitude());
      oldCoordinates.setLatitude(c.getLatitude());
      oldCoordinates.setLongitude(c.getLongitude());
    }

    if (c != null){
      ((StringItem)locationForm.get(lat)).setText(String.valueOf(c.getLatitude()));
      ((StringItem)locationForm.get(lon)).setText(String.valueOf(c.getLongitude()));
      ((StringItem)locationForm.get(alt)).setText(String.valueOf(c.getAltitude())+" m");

      ((StringItem)locationForm.get(speed)).setText(
        String.valueOf(loc.getSpeed()/1000*3600)+" km/h");

      ((StringItem)locationForm.get(course)).setText(String.valueOf(az)+" "+
      findDirection(az));

      ((StringItem)locationForm.get(dist)).setText(String.valueOf(distance/1000)+" km");
    }

  } catch (SecurityException se) {
    // The MIDlet is not authorized to retrieve location information
    }
}
```

Section 5. Test the application

Finally, you're ready to test the application. In this section you will learn how to do that using the Sun Java Wireless Toolkit with a preset device route.

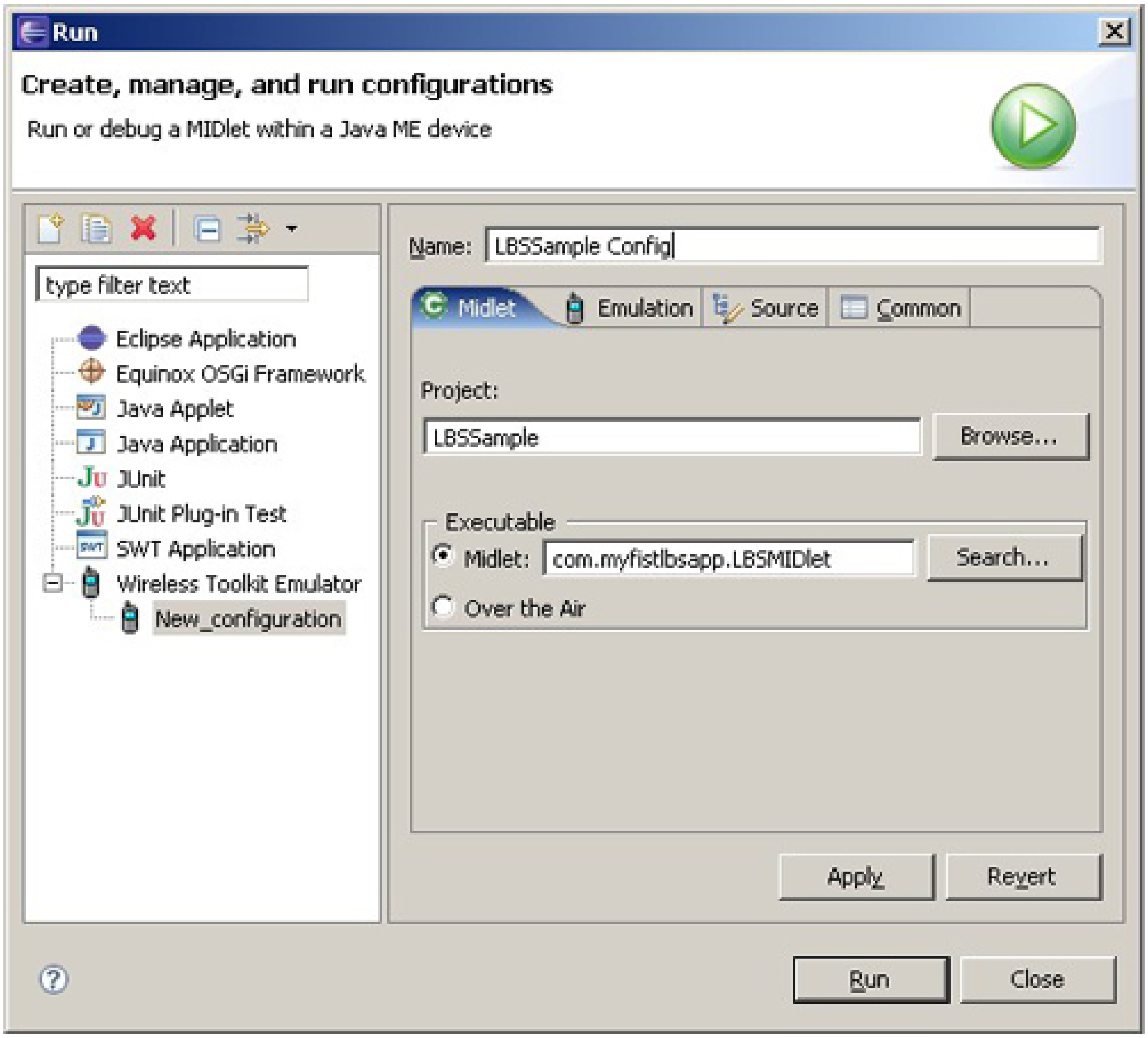
Configure the run settings

First, you'll configure the run settings for both the MIDlet and the emulator.

1. Select **Run** from the Eclipse Run menu.
2. Enter `LBSSample config` in the Name field.
3. Choose the MIDlet radio in the Executable section and enter `com.myfistlbsapp.LBSMIDlet` in the textbox. (See Figure 31 below.)
4. Switch to the Emulation tab. (See Figure 32.)
5. From the "Security Domain" drop-down list, select `manufacturer`. (See Figure 31.)
6. Click **Apply**.

When you click Run, your MIDlet and emulator windows should look as shown in Figure 31 and Figure 32.

Figure 31. Run configuration for the MIDlet

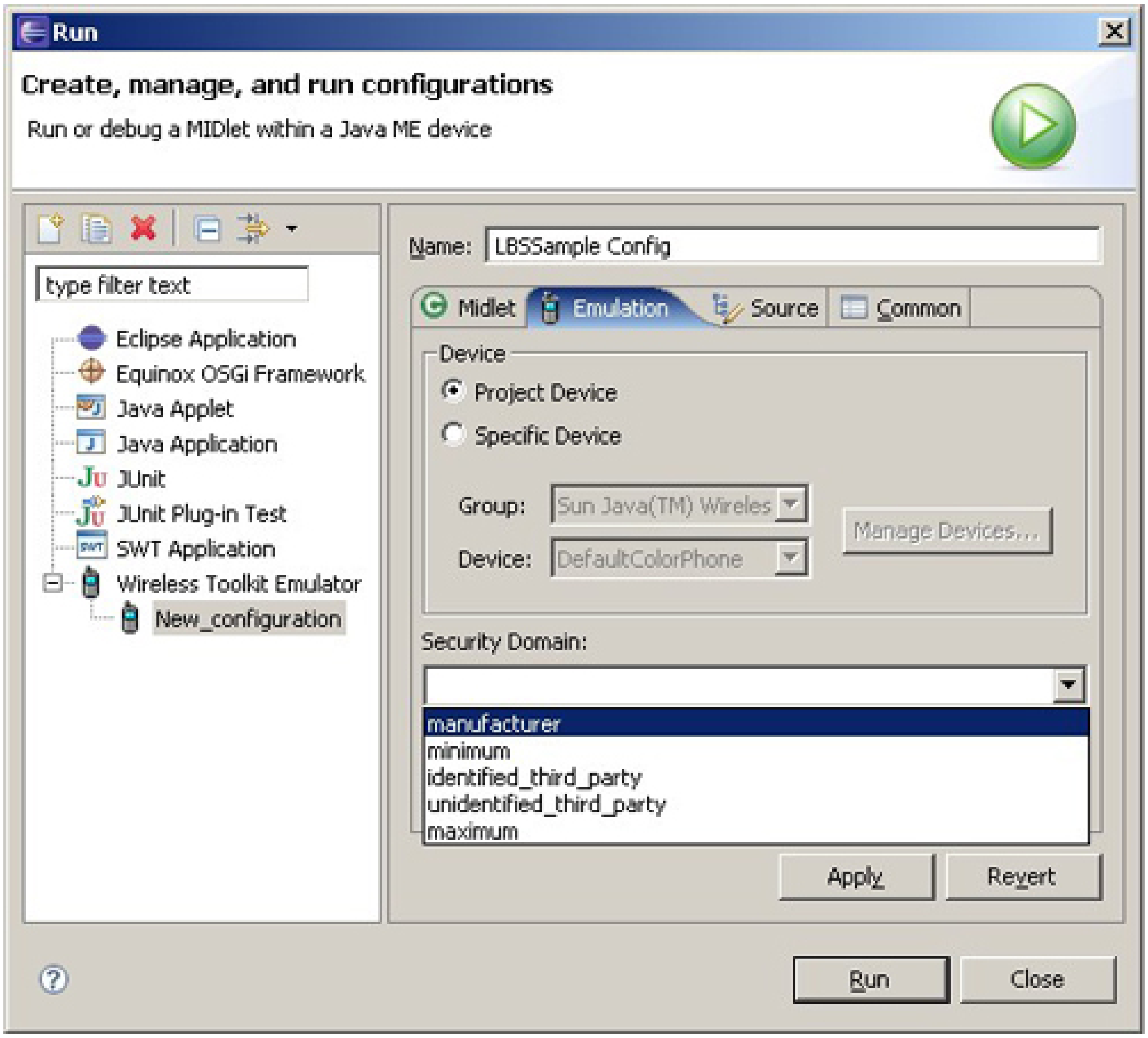


Name: LBSSample Config

Project: LBSSample

Midlet: com.myfistlbsapp.LBSMIDlet

Figure 32. Run configuration for the emulator


Run

Create, manage, and run configurations

Run or debug a MIDlet within a Java ME device

Name: LBSSample Config

Device: [✓ Project Device] Specific Device

Group: Sun Java(TM) Wireles

Device: DefaultColorPhone

Security Domain:

manufacturer
minimum
identified_third_party
unidentified_third_party
maximum

Create a waypoint and set the ProximityListener

Next, you set a waypoint and radius for the `ProximityListener`. The `ProximityListener` notifies you when the device has entered the defined area.

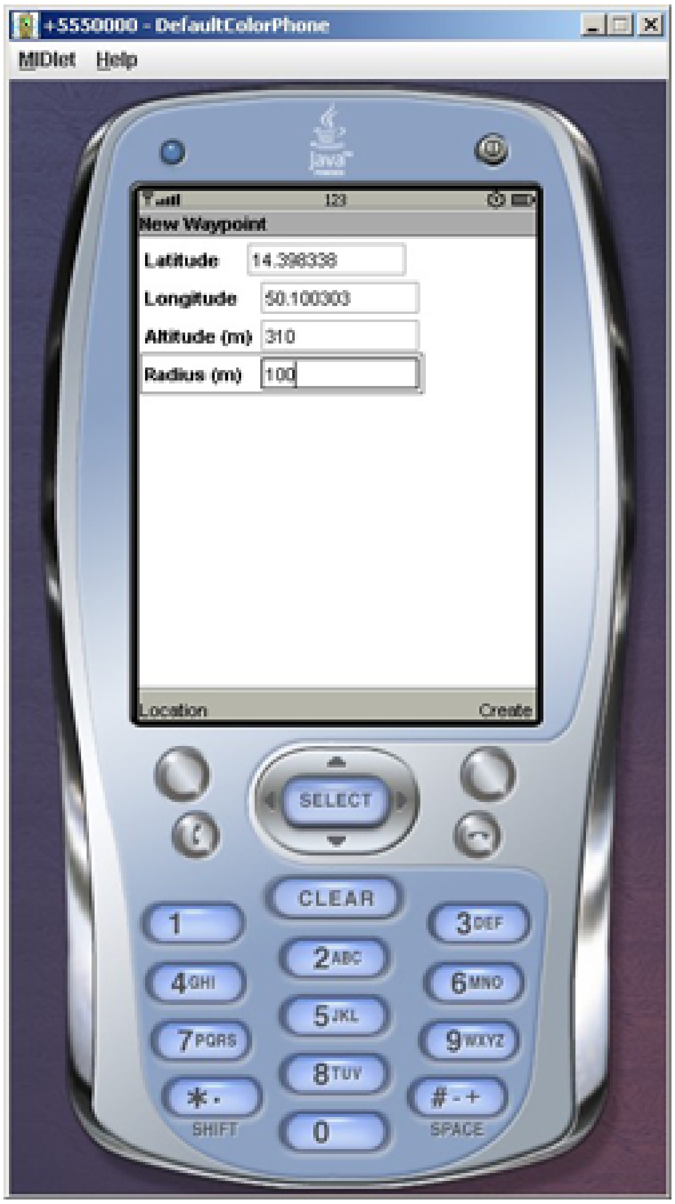
1. Click the Menu button on the phone emulator and select **New Waypoint**. The resulting screen is shown in Figure 33.

2. Enter the following values in the Create Waypoint Screen:
   - Latitude: `14.398338`
   - Longitude: `50.100303`
   - Altitude: `310`

- Radius: `100`

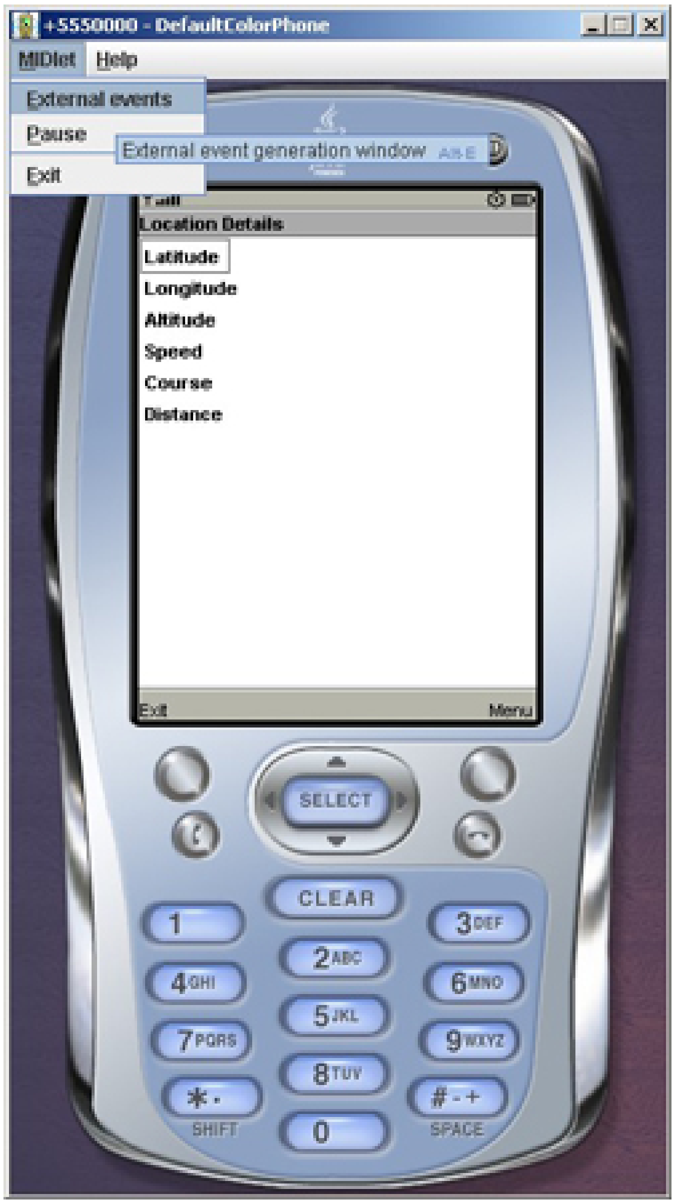
3. Click the corresponding button for Create on the emulator.
4. Click the corresponding button for Location on the emulator.

Figure 33. Create a waypoint

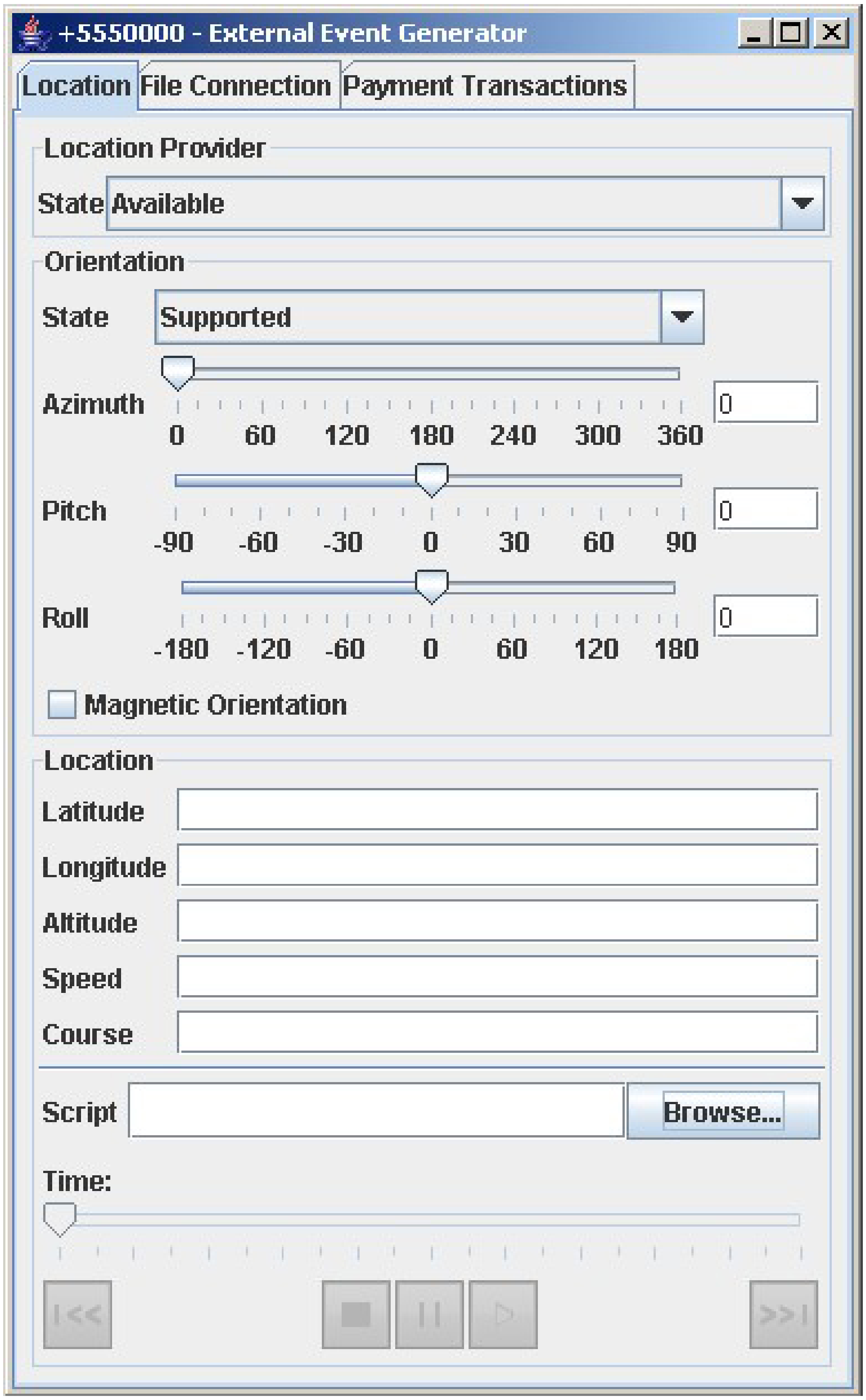


Load the preset route

A preset route makes it possible to test your application's responses to listener events. Your next step is to load the preset route.

1. From the phone emulator's MIDlet menu, choose **External Events**, as shown in Figure 34:
   Figure 34. The phone emulator's MIDlet menu
   

2. Load the citywalk.xml sample route:
   1. In the External Events window click **Browse,** as shown in Figure 35:
      Figure 35. Browse the External Events window



+5550000 - External Event Generator

Location | File Connection | Payment Transactions

Location Provider

State: Available

Orientation

State: Supported

Azimuth: 0

Pitch: 0

Roll: 0

Magnetic Orientation

Location

Latitude

Longitude

Altitude

Speed

Course

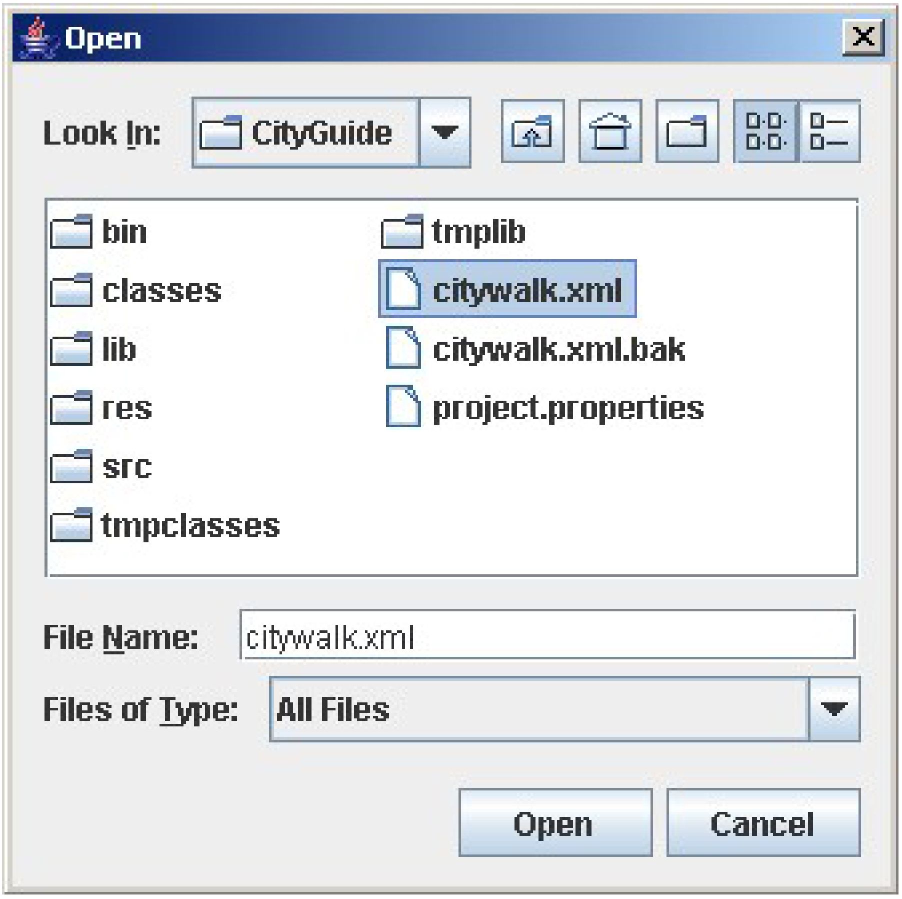
Script

Browse...

Time:

2. In the Browsing window, browse to `CityGuide\citywalk.xml` and click **Open**, as shown in Figure 36:

Figure 36. Select a route



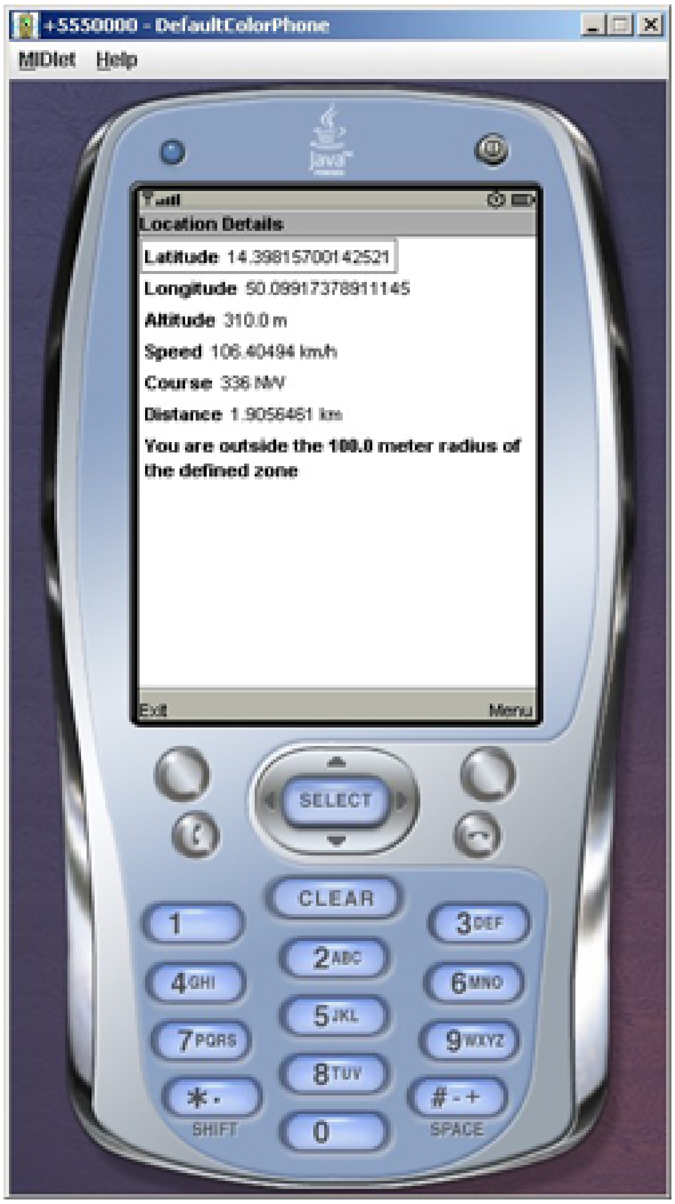
File Name: citywalk.xml

3. Back in the External Events window, click **Start** at the bottom.

Run the example

Everything is set up. Now all you need to do is run the application in the emulator and listen for location updates or proximity events.

1. Click the button corresponding with the menu option on the display of the phone emulator.
2. Select **Start**.
3. You should now see the target device's location coordinates with speed, heading, and direction details similar to those in Figure 37.
   Figure 37. Running the LBSSample application



Latitude 14.398157001 42521
Longitude 50.09917378911145
Altitude 310.0 m
Speed 106.40494 km/h
Course 336 NW
Distance 1.9056461 km
You are outside the 100.0 meter radius of the defined zone

For another option, try toggling the Location Provider State in the External Event window and see what happens!

Section 6. Summary

This tutorial covered all the steps required to get you started with building a location-based application using the Eclipse SDK, the EclipseME plug-in, and the

JSR 179 Location API.

After walking through the steps to configure the recommended development environment for the tutorial example, I showed you how to configure your Eclipse SDK and the EclipseME plug-in for Java ME location-based application development. Next, I covered the core components of the Location API, using code snippets to demonstrate the practical usage of the API. Finally, I explained and demonstrated the process of building and testing a simple location-based application.

Building location-based service applications using the JSR 179 Location API is straightforward. With what you've learned in this tutorial, you are ready to develop your own simple location-based applications for Java ME devices.

Downloads

Description	Name	Size	Download method
Tutorial source code	lbssample.zip	12KB	HTTP

Information about download methods

Resources

Learn

- "Roaming charges: Can you base my location now?" (developerWorks, November 2005): This column provides great ideas in location-based application development.
- "Developing J2ME applications with EclipseME" (developerWorks, November 2004): A tutorial introduction to developing J2ME applications using the Eclipse IDE, EclipseME plug-in, and the Sun Java Wireless Toolkit.
- "RFID to the rescue" (developerWorks, May 2006): Consider a completely different approach to location-based services.
- JSR 179 package specification: This article documents the Location APIs in detail.
- JSR 179: Location API for J2ME: Keep track of updates to the JSR 179 specification.

Get products and technologies

- Java platform homepage: Get the latest version of the Java Standard Edition Software Development Kit.
- Eclipse 3.2 SDK: Download Eclipse from the Eclipse Foundation Web site.
- Sun Java Wireless Toolkit 2.5 beta: Download the wireless toolkit and phone emulator used to test the example application.
- EclipseME supported wireless toolkits: Learn about other wireless toolkits supported by the EclipseME plug-in.
- IBM trial products for download: Build your next development project with IBM trial software, available for download directly from developerWorks.

About the author

Kevin L Sally
Kevin Sally is a senior IT architect at IBM Canada. He is a Certified IT Architect in the Applications Architecture discipline and is currently a consultant in the IBM Software Group Lab Services for WebSphere. Kevin has experience building solutions that span multiple platforms, products, and technologies. You can contact Kevin at ksally@ca.ibm.com.