# MIDlet development with the Wireless Toolkit

Presented by developerWorks, your source for great tutorials

**ibm.com/developerWorks**

---

## Table of Contents

If you're viewing this document online, you can click any of the topics below to link directly to that section.

# Section 1. Introduction

## What is the Wireless Toolkit?

The Wireless Toolkit is an integrated development environment (IDE) for creating Java 2 Platform, Micro Edition (J2ME) applications, commonly referred to as *MIDlets.*

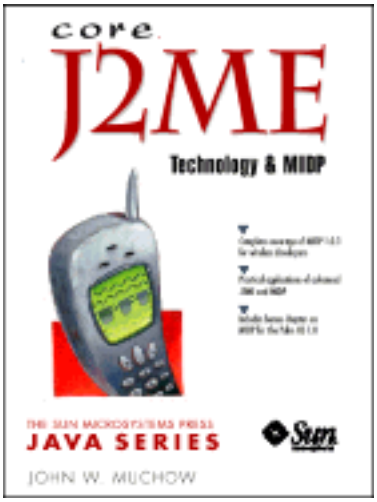
This tutorial will provide step-by-step introductions for downloading, installing, and configuring the toolkit. We'll also walk through the basic steps to compile, preverify, and package MIDlets.

To get a better feel for the depth of MIDP 2.0 support, we'll demonstrate the over-the-air provisioning tools to download MIDlets from a remote server. We'll also have a close look at some of the things you can do with the toolkit's security features, such as signing MIDlets, creating certificates, and setting permissions. Finally, we'll briefly cover some of the other features available in the toolkit, including monitoring tools and enhancements to match the MIDP 2.0 spec.

---

## About the author

*John Muchow* is the author of *Core J2ME Technology and MIDP*, a best-selling J2ME book that has been translated into Chinese, Korean, and Italian.




He is also a freelance wireless consultant, developer, and trainer. You can visit the *Core J2ME Web site* for additional source code, articles, and developer resources.

# Section 2. Download and install the software

## Required components

You'll need two software tools before you can go any further, and we'll talk about these in the next panels:

- The Java Development Kit (JDK)
- The Wireless Toolkit (WTK)

---

## Download the Java Development Kit (JDK)

The JDK provides the Java source code compiler and a utility to create Java Archive (JAR) files. When working with version 2.0 of the Wireless Toolkit (as we are in this tutorial), you will need to download JDK version 1.4 or greater.

*Download JDK version 1.4.1.*

---

## Download the Wireless Toolkit (WTK)

The WTK download contains an IDE as well as the libraries required for creating MIDlets.

*Download J2ME Wireless Toolkit 2.0.*

If you would like additional information about the libraries required to create MIDlets, you can peruse the following tutorial, which walks you through the creation of MIDlets using command-line tools.

"*The MIDlets Advantage*," John Muchow (*developerWorks*, March 2002).

Note that the Wireless Toolkit is currently only available for Windows platforms. Sun plans to release unsupported versions in the future for Linux and Solaris. There are no plans to support Mac OS X at this time.

---

# Install the software

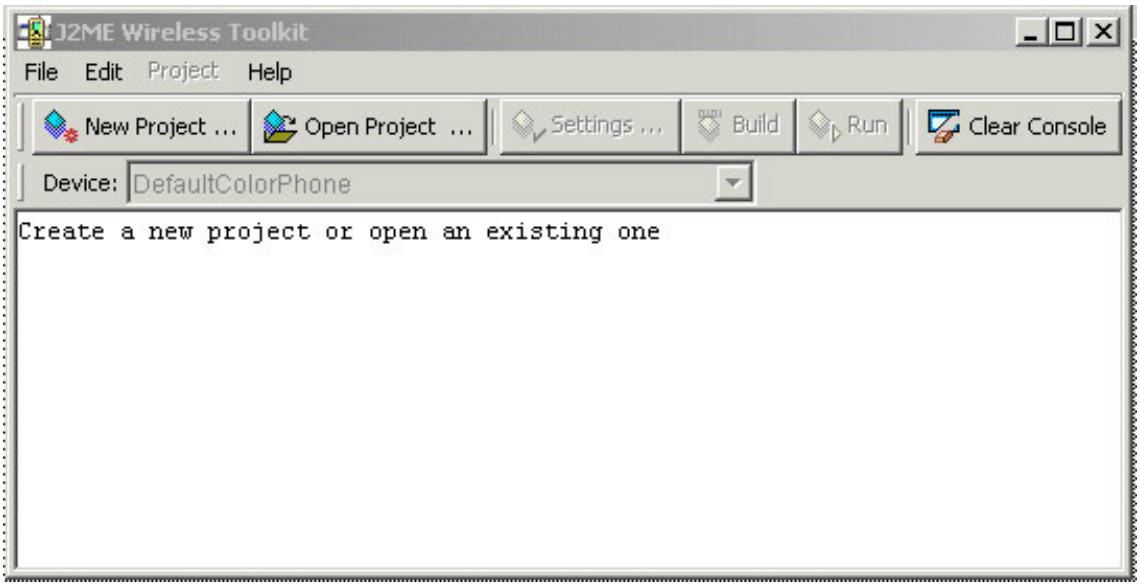
The Java Development Kit (JDK)
Install the JDK as directed in its accompanying documentation. You can choose the default directory or specify another if you prefer. If you choose to do the latter, make a note of where you install the JDK. During installation of the Wireless Toolkit, the software will attempt to locate the Java Virtual Machine (JVM); if it is unable to do so, you will be prompted for the JDK installation path.

The Wireless Toolkit (WTK)
The Wireless Toolkit is contained within a single executable file, which you downloaded in the previous panel. Run this file to begin the installation. We suggest that you use the default installation directory. However, if you do not use the suggested directory, make sure the path you select does *not* include any spaces.

# Section 3. Creating your first MIDlet

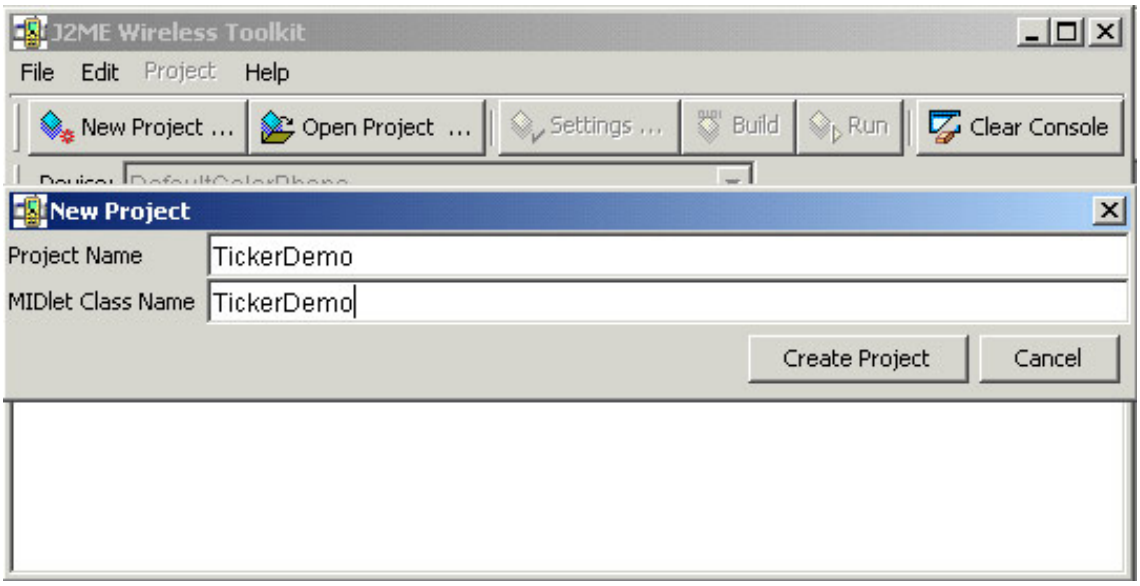
## Starting the KToolbar

Start the WTK by locating and running KToolbar. You should see a display similar to the figure below.




## Creating a project

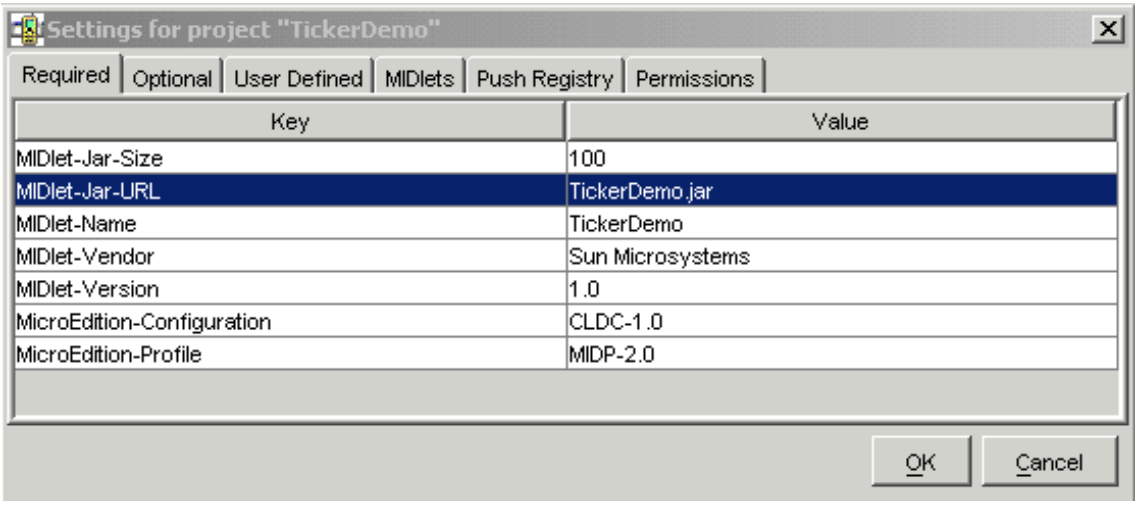
Let's begin by creating a new project within the WTK. Click the New Project icon. Enter the Project Name and MIDlet Class Name shown in the figure below, then click Create Project to finish this step.



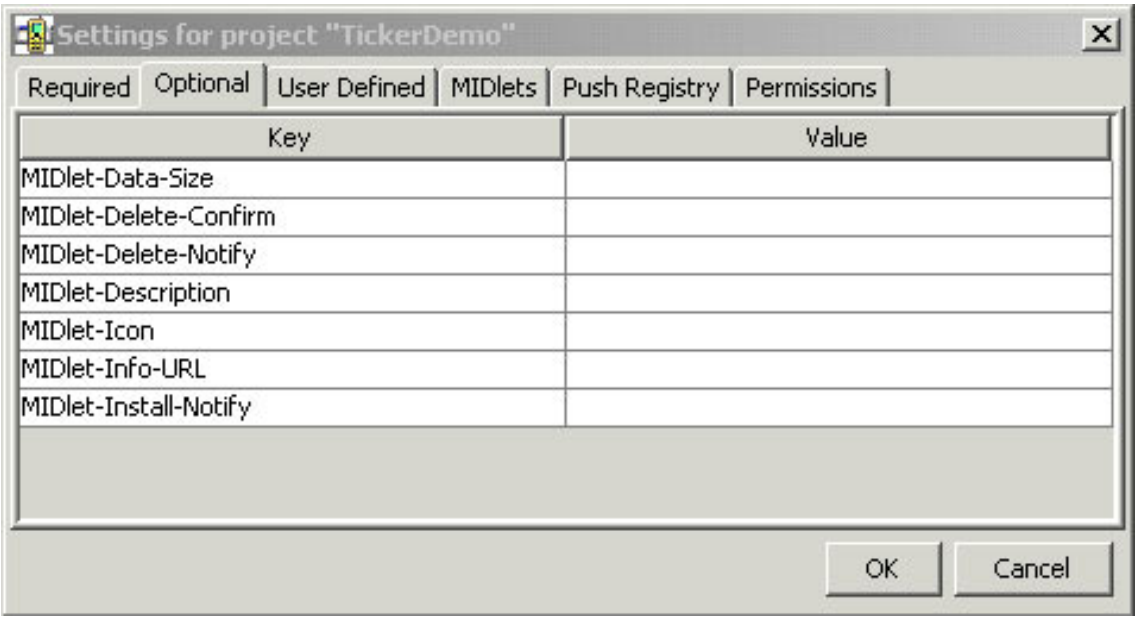

## Setting the project properties

One additional step is necessary before we can create the new project: We have to configure the MIDlet attributes. Attributes are divided into six areas; the discussion of each area below is followed by an illustrating figure.

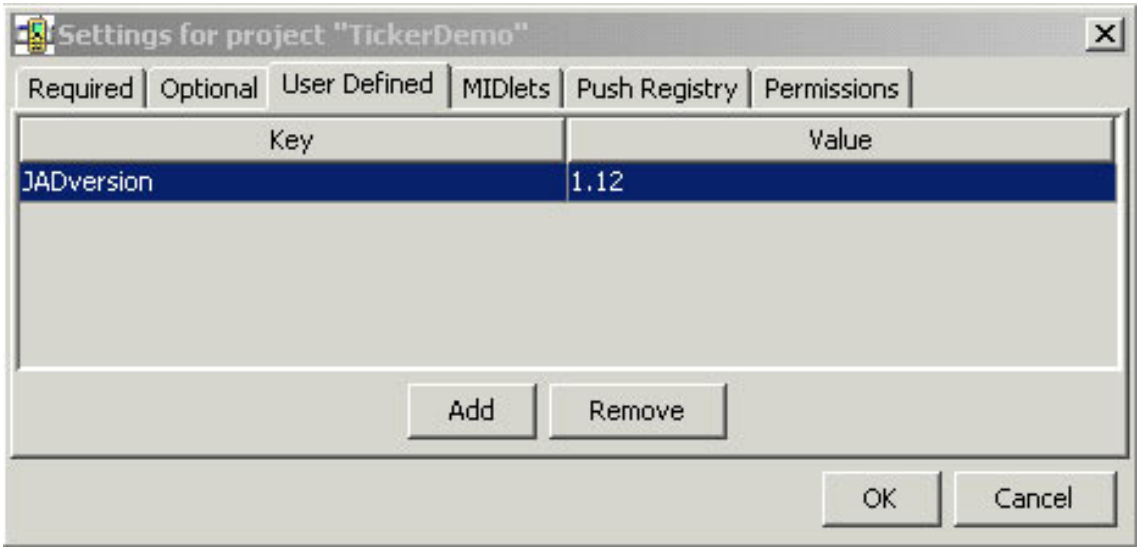
**Required attributes.** There is a total of seven attributes that you must specify when creating a MIDlet suite. These attributes are divided among the JAR manifest file and Java Application Descriptor (JAD) file. For our purpose, clarifying the breakdown of properties within the files is not of importance. One of the benefits of using an IDE is that we can leave the details to the implementation and concentrate our efforts on application development. The WTK creates default values for each attribute, as illustrated in the figure below; we'll use these default values for our MIDlet.



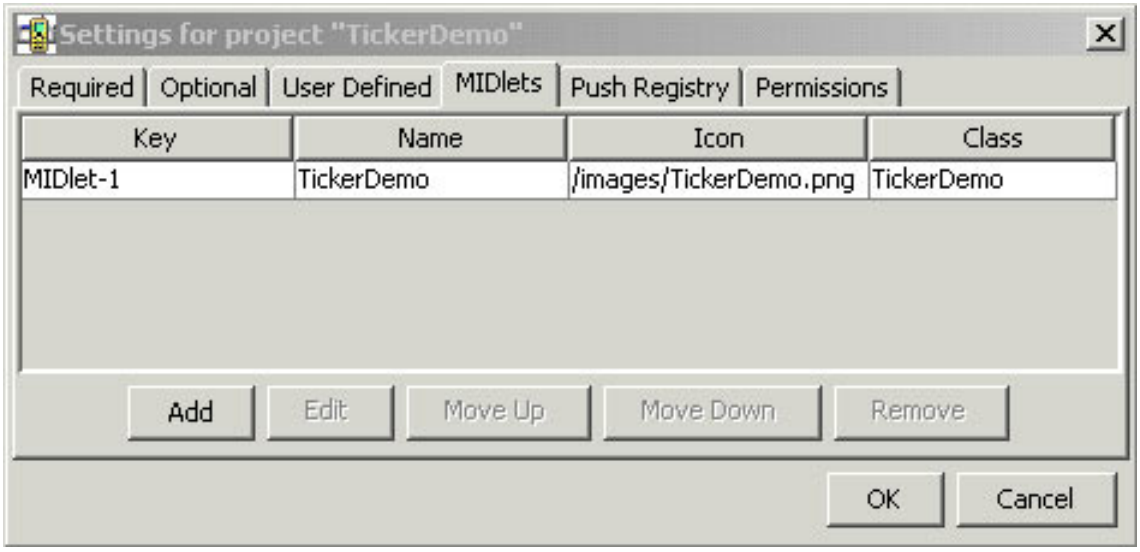

**Optional attributes.** These attributes are not required when creating a MIDlet suite; however, they are available if your application has to provide additional information beyond that in the required attributes.



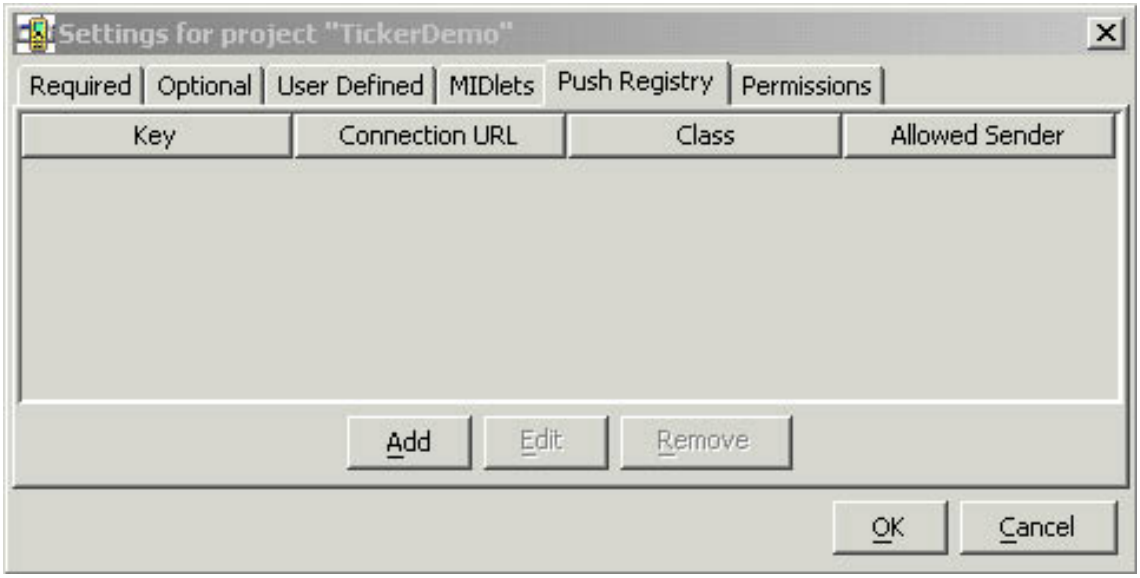

**User-defined attributes.** As a developer, you can create your own attributes. For example, as your application evolves, you will more than likely go through many iterations of your JAD file. As such, it may be handy to place a version number inside the file. The figure below shows how that might look.




**MIDlet attributes.** Here is where we specify attributes for the MIDlet(s) stored in the suite. By default, the WTK will fill in all the fields for MIDlet-1 based on the Project Name and MIDlet Class Name that we entered when creating the project. To provide for better organization, we'll place the icon for our MIDlet in the `images` directory. To do that, change the entry for the Icon property to that shown in the figure below. We'll see the icon momentarily.



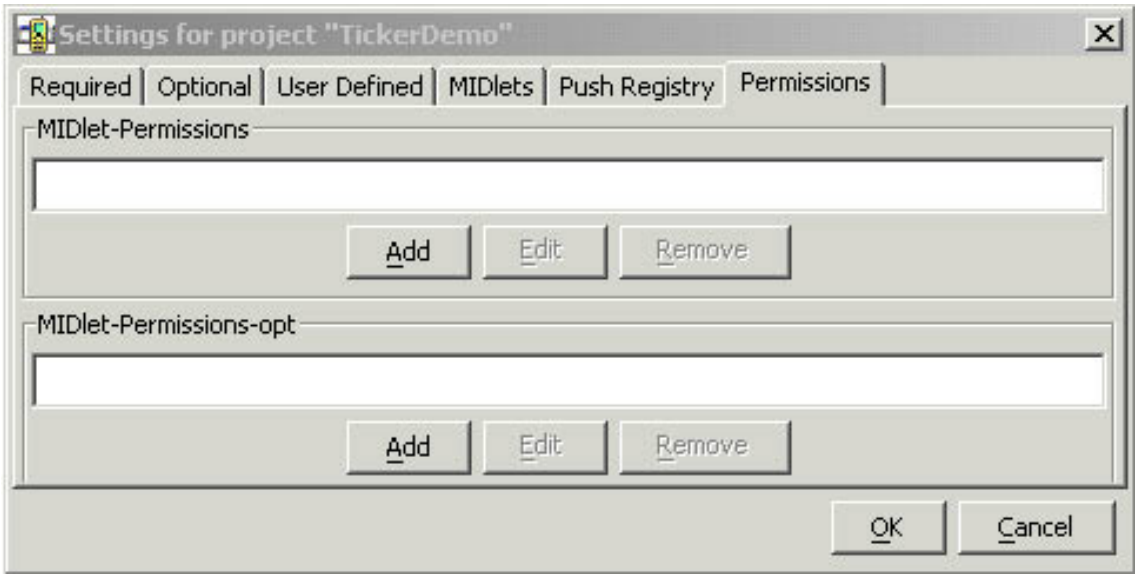

**Push registry attributes.** With the release of MIDP 2.0, MIDlets can listen for a connection from a remote resource; this process is often referred to as *pushing data.* This section is where you maintain a list of valid inbound connections.




We'll discuss the push registry in more detail later (see Push registry on page 40 ).

**Permission attributes.** Under the MIDP 1.0 spec, a MIDlet could only access the libraries (APIs) that were packaged inside the MIDlet suite; this was commonly called the *sandbox model.* Under this model, a MIDlet could not query information from the device, or otherwise interact outside the scope of the suite.

MIDP 2.0 introduces the concept of *trusted applications,* allowing access beyond the sandbox. In this section, you can add properties to specify which APIs are accessible. Attributes specified in MIDlet-Permissions are those that are required in order for the MIDlet to run. Those specified in MIDlet-Permissions-opt are optional.




We'll discuss security issues like these in more detail further on (see Security on page 34

).

---

## Writing the code

The following listing contains the code for a simple MIDlet that displays a `List` component and a scrolling `Ticker`.

At this point, it isn't important to understand the code. Rather, our intention is to successfully compile and preverify a MIDlet without any errors. Copy and paste the code below into your favorite text editor. I'll show you where to save the file in the next panel.

```
/*-------------------------------------------------
* TickerDemo.java
*-------------------------------------------------*/
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class TickerDemo extends MIDlet implements CommandListener
{
  private Display display;    // Reference to Display object
  private List lsProducts;   // Main productlist
  private Ticker tkSale;     // Ticker of what's on sale
  private Command cmExit;    // Command to exit the MIDlet

  public TickerDemo()
  {
    display = Display.getDisplay(this);

    cmExit = new Command("Exit", Command.SCREEN, 1);

    tkSale = new Ticker("Current Sale: Torchiere Lamp only $29.00");

    lsProducts = new List("Products", Choice.IMPLICIT);
    lsProducts.append("Floor Lamp", null);
    lsProducts.append("Chandelier", null);
    lsProducts.addCommand(cmExit);
    lsProducts.setCommandListener(this);
    lsProducts.setTicker(tkSale);
  }

  public void startApp()
  {
    display.setCurrent(lsProducts);
  }

  public void pauseApp()
  {
  }

  public void destroyApp(boolean unconditional)
```

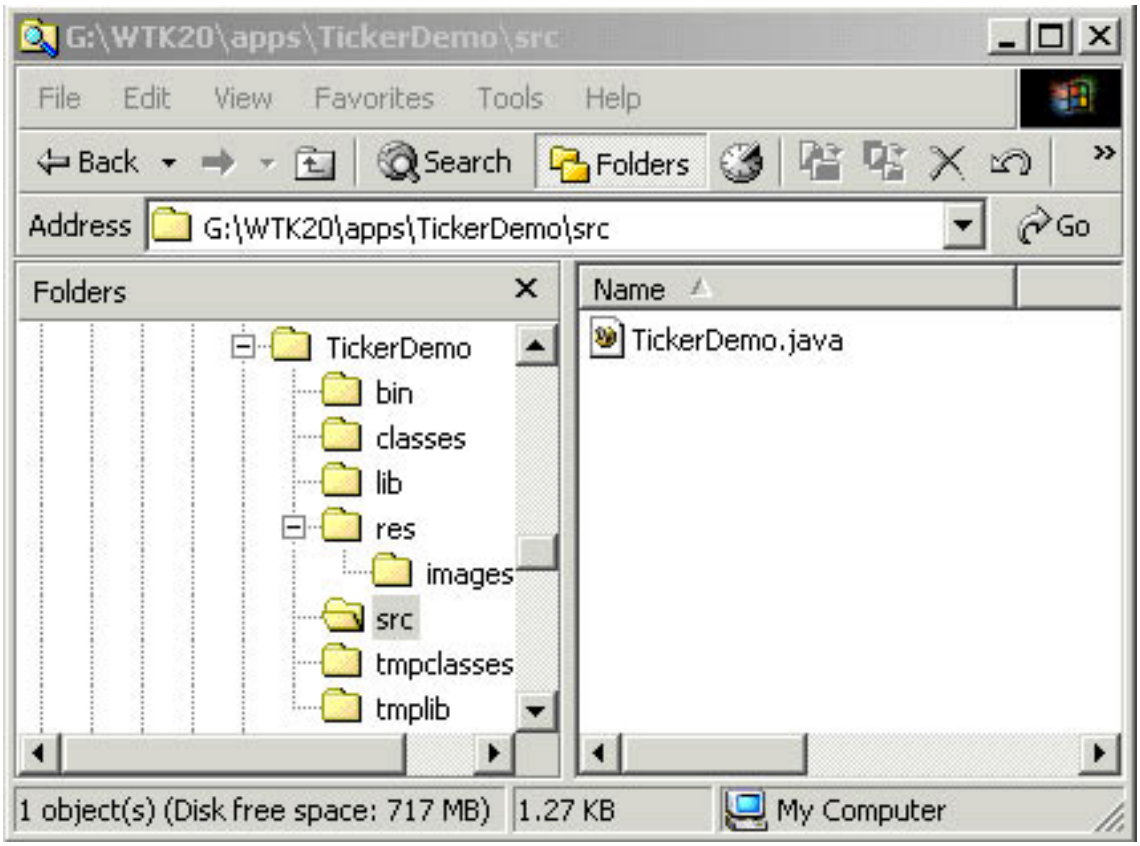
```
  {
  }

  public void commandAction(Command c, Displayable s)
  {
    if (c == cmExit)
    {
      destroyApp(true);
      notifyDestroyed();
    }
  }
}
```

---

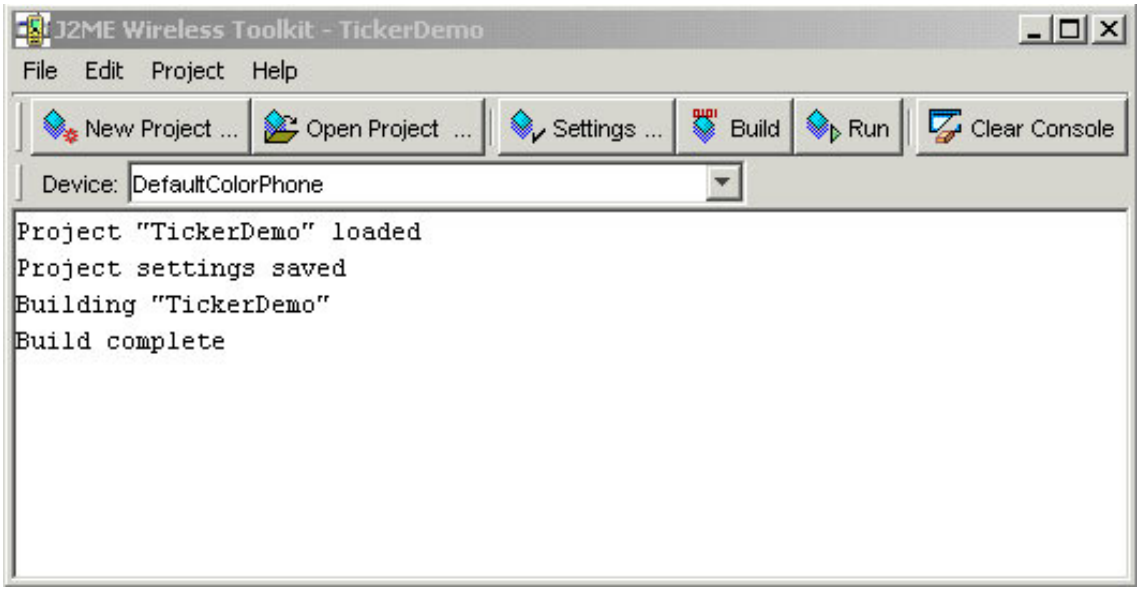
## Save, compile, and preverify

Save the source code from the last panel in the directory location shown below. You'll notice as you navigate the directory hierarchy that the WTK has created the `G:\WTK20` directory and all its subdirectories. The `src` directory is where you store source files.

**Note:** The drive and directory will vary depending on the location where you installed the toolkit.

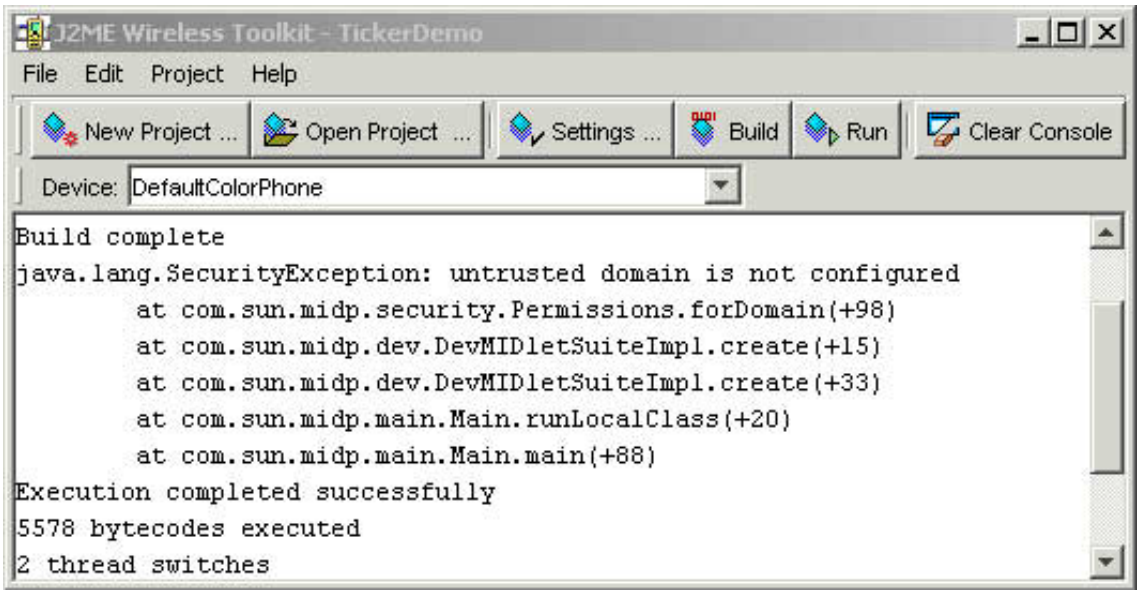


Once the file is saved, click the Build icon on the KToolbar. This will invoke the Java compiler, followed by the preverifier. Assuming that there are no errors, click Run to start the MIDlet.




## Troubleshooting

If you have previously installed MIDP software (other than the Wireless Toolkit) on your machine, there's a good chance that you will see the error message shown in the figure below. To resolve this, remove the environment variable `MIDP_HOME` and restart the WTK.




## Running the MIDlet

Once you've started the MIDlet, the Application Manager will appear. Our MIDlet name and icon will be shown on the display, as illustrated in the figure below.


Once you launch the MIDlet, you'll see a scrolling ticker moving across the top of the display, and a list of two products.


Current Sale: Torchiere Lamp o
Products
Floor Lamp
Chandelier
Exit
SELECT
CLEAR

# Section 4. Over-the-air provisioning

## The basics

WTK 2.0 has added much-needed support for simulating over-the-air (OTA) provisioning. This simulated OTA function demonstrates how a mobile phone locates and downloads MIDlets from a remote server.

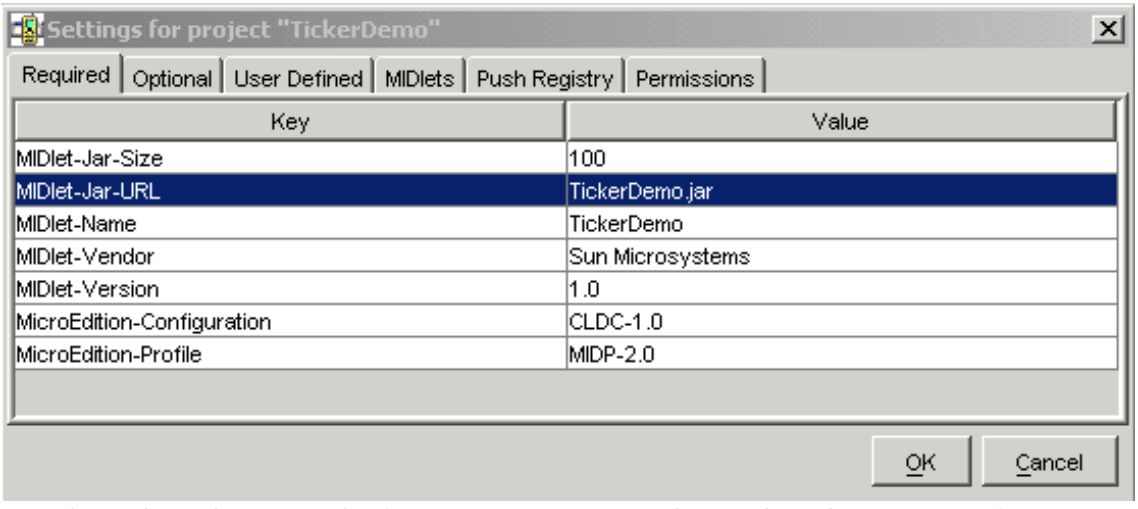
In this section, we'll walk through all the steps necessary to emulate packaging a MIDlet for public consumption, and locating, downloading, and installing a MIDlet from a remote resource. The basic process is as follows:

- Package the MIDlet into a JAR file, and create the associated JAD file.
- Create a simple HTML file referencing the JAD.
- Place the JAD, JAR, and HTML files on a Web server.
- Locate, download, and install the MIDlet.

---

## Packaging, Step 1: Update attributes

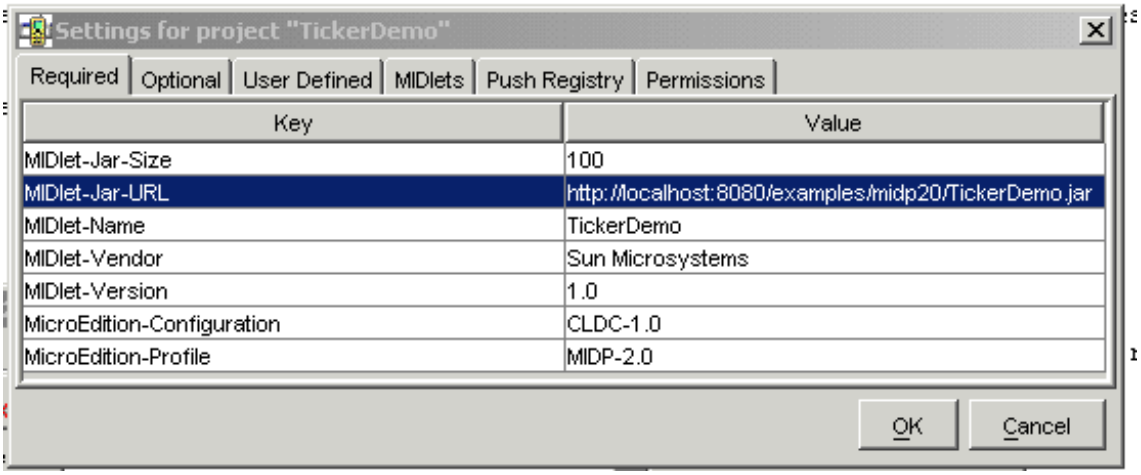
Creating the JAR and JAD files is a piece of cake with the WTK. However, before we do this, we need to make a small change to one of the required attributes we set previously. Make note of the entry for MIDlet-Jar-URL shown below:




In order to launch our MIDlet from a server, we need to update the URL to reference the location where the MIDlet will be stored. Here is the complete URL, based on my system configuration (alter as needed for your own system):

```
http://localhost:8080/examples/midp20/TickerDemo.jar
```

Update the MIDlet-Jar-URL attribute:



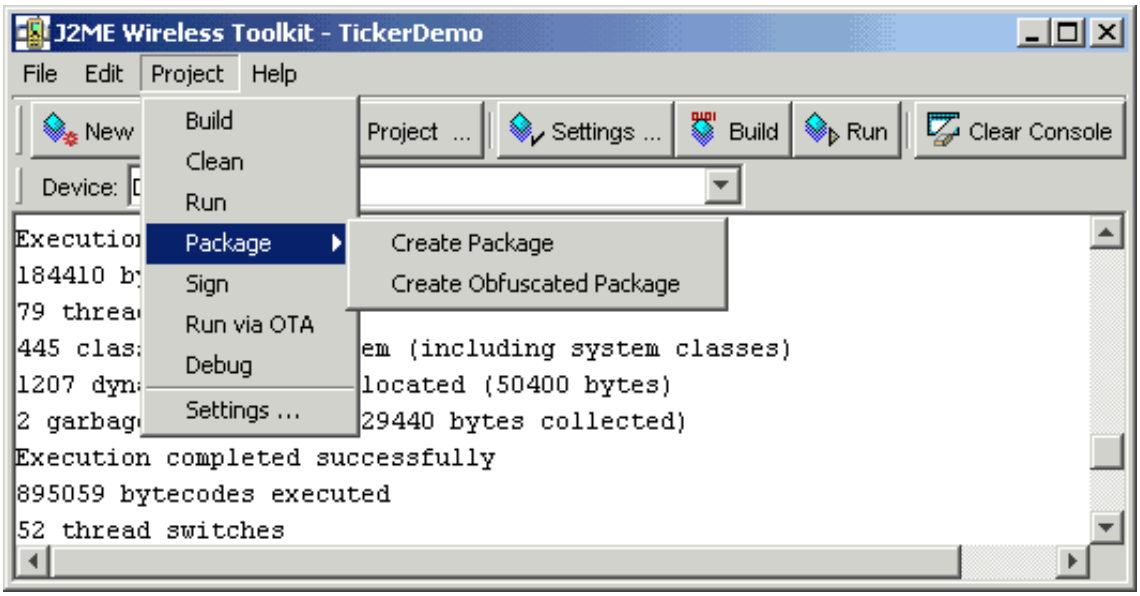

**Note:** If you don't have a Web server installed on your machine, give *Apache Tomcat* a try. It's an excellent Web server, and it's open source.

---

# Packaging, Step 2: Create the package

With the proper reference to the MIDlet JAR file's location now in place, we are ready to package the MIDlet -- to have the WTK create the JAR and JAD files, in other words. It's quite simple:

- Click on the Project menu
- Select Package
- Select Create Package




## Packaging, Step 3: Create the HTML file

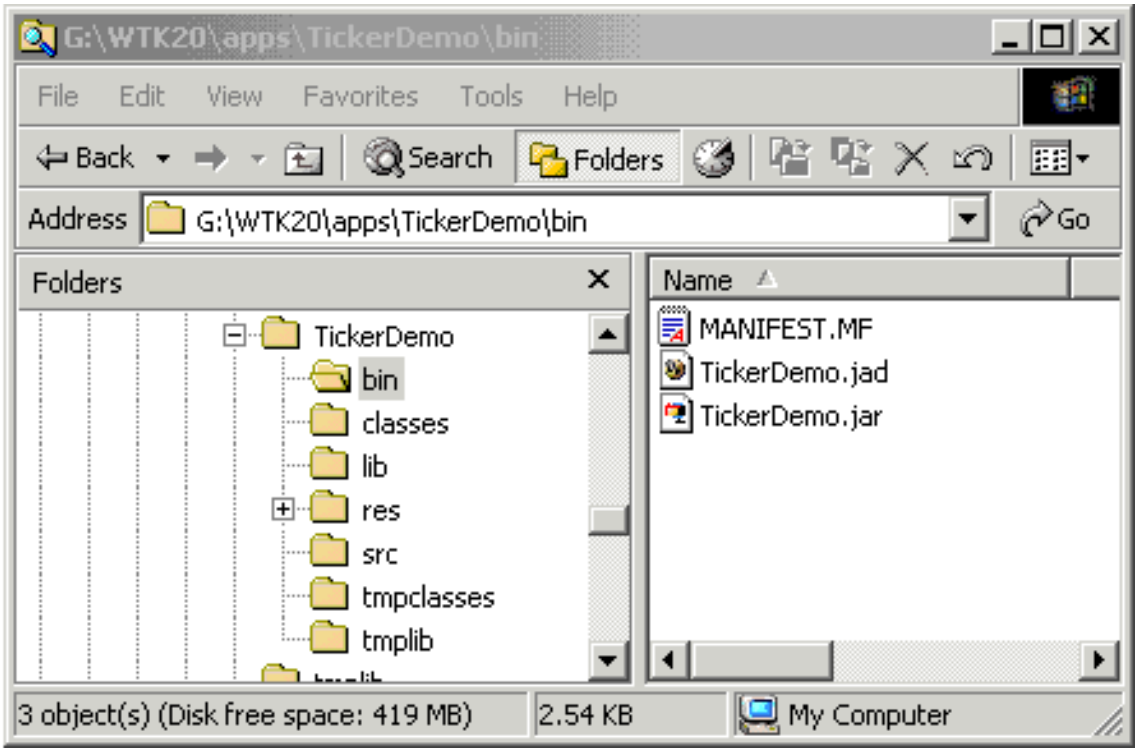
In order for the Application Manager to determine what files are available for download, we must create a simple HTML file:

```
<html>
<head>
<title>TickerDemo</title>
</head>
<body>
<a href="http://localhost:8080/examples/midp20/TickerDemo.jad">TickerDemo</a>
</body>
</html>
```

Type, or copy and paste the HTML above into a text editor and save it with the filename `TickerDemo.html`. We'll discuss where this file should end up in the next panel.

## Packaging, Step 4: Copy files to the Web server

If we navigate to the `bin` directory of our project, we'll see the files that the WTK has created for us:




The last step is to copy the JAD, JAR, and HTML files to our Web server. Consider again the URL we specified earlier for the MIDlet-Jar-URL attribute:

```
http://localhost:8080/examples/midp20/TickerDemo.jar
```

With this as our MIDlet-Jar-URL, we need to copy the files into the `examples\midp20` directory off the root of the Web server. Once these files are in place, we are ready to load the MIDlet.

---

# Quick summary

Let's regroup for a minute. Here's what we've accomplished up to this point:

- We wrote the code, and compiled and verified our MIDlet
- We updated the URL in the MIDlet-Jar-URL attribute to reference the location on the Web server where the JAR file will be located (`http://localhost:8080/examples/midp20/TickerDemo.jar`)
- We packaged the MIDlet (that is, we created the JAR and JAD files)
- We created a simple HTML file to reference the MIDlet
- We copied the JAR, JAD, and HTML files to the Web server

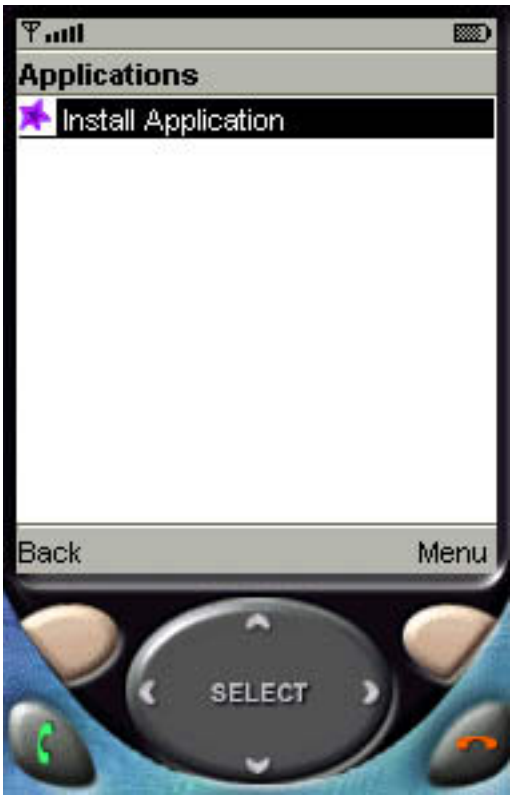
Now it's time to run the emulator, and locate, download, install, and run our MIDlet!

---

## Run the emulator via OTA

To begin, click the Project menu and select Run via OTA. In a moment, the emulator will start. The display should look like the figure below:


Launch the Application Manager by selecting Apps. There will be just one option -- Install Application -- shown on the display.

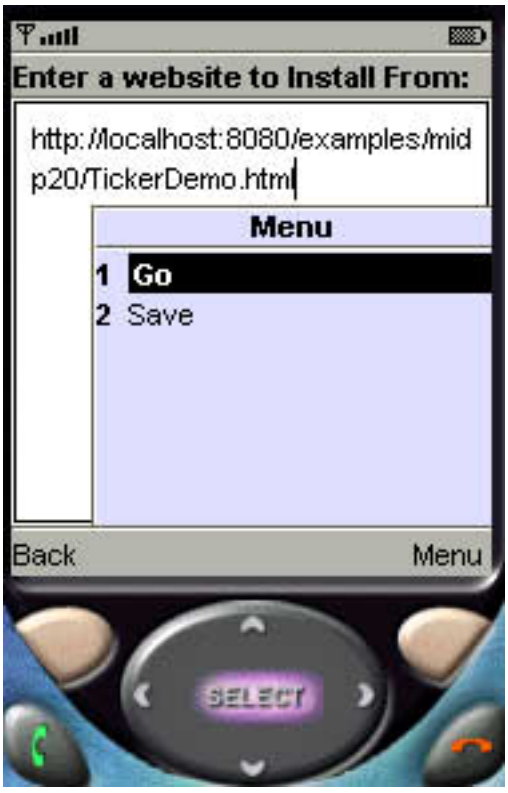



Start the program by clicking on the Menu item in the lower right-hand corner of the screen; once the menu is displayed, select Launch.

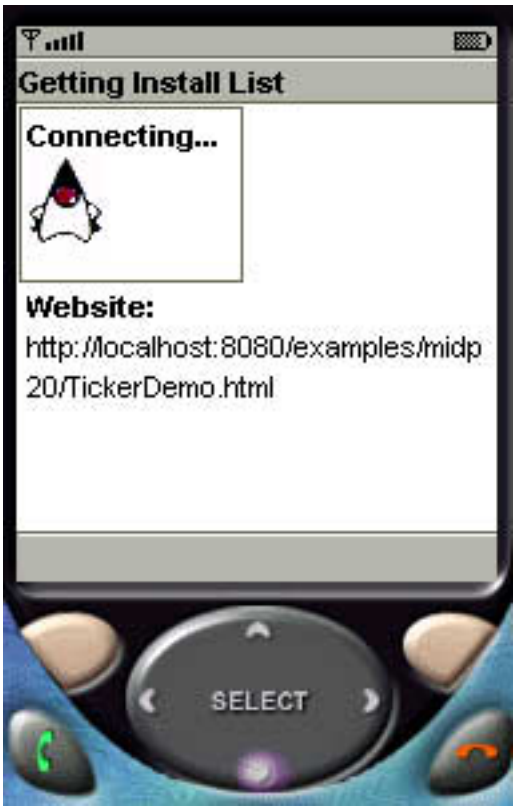

## Enter the Web server address

At this point, enter the address of our target Web server, with a specific reference to the HTML file, as shown in the figure below:


Click Menu in the lower right-hand corner of the screen, and then select Go.




The device will now attempt to connect to the server and load the HTML file. You'll see the following message:




## Installing the MIDlet, Step 1: Select an application

Once the emulator has parsed the HTML file, the Application Manager will display a list of MIDlets that are available for download.


Highlight your chosen MIDlet (obviously, we only have one to choose from here) and then click Install in the lower right-hand corner of the screen to begin. Notice in the next figure that the JAR file is not immediately downloaded. Rather, the first file to be sent to the device is the application descriptor file (JAD). There is a good reason for this, as we'll see in a moment.


## Installing the MIDlet, Step 2: Approving installation

Check out the figure below. The device displays the size of the JAR file and the version of the MIDlet, along with other information. The latter comes directly from the MIDlet-Version attribute we specified when creating the project.


Can you see why the Application Manager didn't download the JAR file immediately? If the JAR were too big, or if we had a more current version already installed, the Application Manager could notify us, and ask whether or not we wanted to continue.

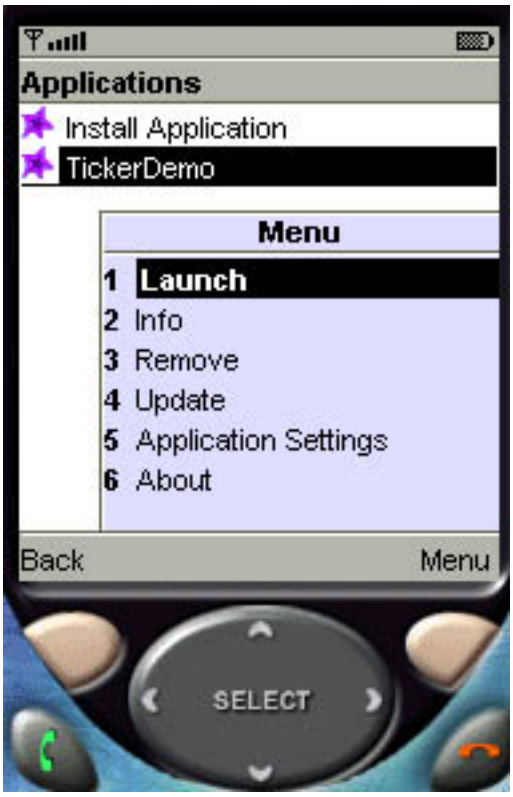
Click Install in the lower right-hand corner of the screen to download the JAR file. You'll see the display below as the file is downloaded onto the device.


Once the Application Manager has downloaded the suite, it will attempt to verify the contents, as shown in the figure below. If this step is successful (see Security on page 34 for some reasons why it might not be), the MIDlet will be ready to run on the device.


## Running the MIDlet

We're nearly there. The Application Manager now shows the installed MIDlet as one of its available applications.


Highlight our newly installed application, then click on the Menu option in the lower right-hand corner of the screen and choose Launch to load the MIDlet.




At long last, the MIDlet is running! The ticker scrolls across the top, with the product list displayed below.


Current Sale: Torchier
Products
Floor Lamp
Chandelier
Exit
SELECT

# Section 5. Security

## Introduction

As we've already noted, under the MIDP 1.0 spec, all MIDlets operated using a sandbox security model. Essentially, a MIDlet could only access APIs that were included as part of the MIDP specification (for example, record store, user interface components, etc.) or libraries included within the same MIDlet suite. Access was not permitted to any device-specific functionality.

MIDP 2.0 introduces the concept of *untrusted* and *trusted* applications. Untrusted MIDlets are equivalent to those in 1.0 -- that is, they are limited to APIs within the specification and the MIDlet suite. A trusted MIDlet, on the other hand, can have access outside the scope of the suite. For example, a device may expose functions that allow access to a phone list directory, or possibly device configuration settings.

In the subsequent panels, we'll give you an overview of how you can use the WTK to access these security functions.

---

## Security basics

In a nutshell, here's how MIDlet security works. A *protection domain* defines permissions that may be given to a MIDlet suite (in that domain). The domain owner holds the key, deciding who has access to the protected functions. It also decides how to identity what it considers a trusted MIDlet suite. One means of identification is through a X.509 Public Key Infrastructure (PKI) certificate.

In order to use PKI, a MIDlet suite must generate a PKI certificate. This process is known as *signing.* Once a suite is signed, a device can attempt to verify the signature, in turn granting access to protected functions.

As is apparent in this short discussion, security is not a topic for the faint of heart. Since our intention here is to learn how to use the WTK, and not to become security experts, let's get a taste of using the security features we have at our fingertips.

---

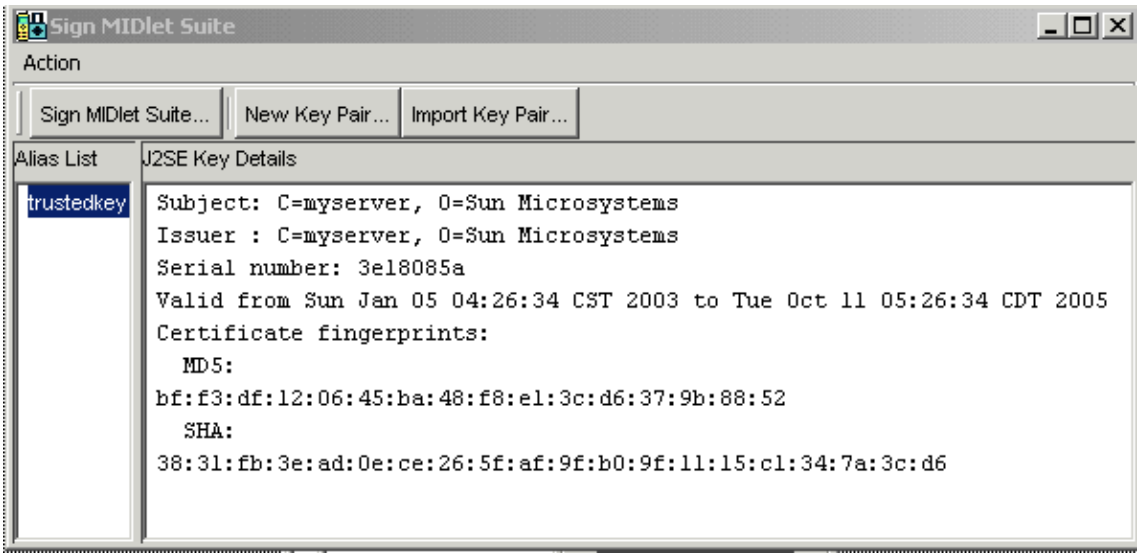
## Signing a MIDlet suite

In a production setting, a certificate would be generated by a certificate authority. For our needs, we can generate a certificate to simulate the creation of a signed MIDlet.

**Step 1:** Using the same TickerDemo project and source code you created earlier, package the MIDlet into a JAR and JAD file. Next, select Project from the KToolbar menu, then click Package, and choose Create Package.

**Step 2:** Select File from the KToolbar menu. Click Utilities. Click Sign MIDlet from the dialog box.

At this point, you'll see the dialog box shown below.




Make note of the range of dates for which the certificate is valid:

```
Valid from Sun Jan 05 04:26:34 CST 2003 to Tue Oct 11 05:26:34 CDT 2005
```

We'll come back to this when we test to see if the certificate is valid on the device.

**Step 3:** Click Sign MIDlet Suite.

---

# Load and run the signed MIDlet suite

Copy the new JAD and JAR to the Web server, into the same directory we used previously. Select Project from the KToolbar menu, then click Run via OTA. When prompted, enter the URL for the MIDlet suite -- again, the same one we used previously:

```
http://localhost:8080/examples/midp20/TickerDemo.html
```

Finally, install and run the MIDlet on the emulator.

Everything looks good -- for now!

---

## How can we verify a MIDlet?

So how was that process different from running our MIDlet without signing? It's as though signing the MIDlet were simply an exercise. However, there is more to the verification process than meets the eye. Below is a screenshot that you should recognize -- it's what you saw when the Application Manager verified the MIDlet:


The message would lead us to believe there is a verification process going on, even though we are simply emulating an actual device. In the next panel, we'll see if our assumption is true.

---

## Testing the certificate date

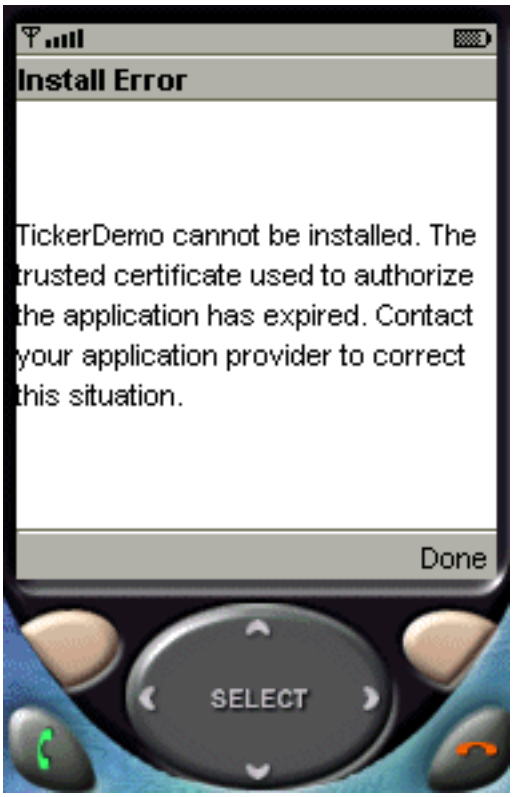
If you recall, when we signed the MIDlet, we saw the following:

```
Valid from Sun Jan 05 04:26:34 CST 2003 to Tue Oct 11 05:26:34 CDT 2005
```

If we change the date on our computer to one outside this valid range, and then attempt to load the MIDlet once again (using OTA), we should receive an error message stating that the certificate is no longer valid. Let's check it out:

- Change the date on your system to sometime in the year 2006.
- Select Project from the KToolbar menu. Click Run via OTA.
- When prompted, enter the URL we've been using for the MIDlet suite (`http://localhost:8080/examples/midp20/TickerDemo.html`).
- Install and run the MIDlet.

Everything should be fine until you reach the verification step. At that point, you will see an error message stating that the certificate to authenticate the application has expired.




There you have it! As advertised, the WTK attempts to properly validate certificates.

Now, you may be wondering: What was the WTK doing when it validated the earlier, unsigned version of our MIDlet? Even unsigned MIDlets need to be verified. In such a case, the verification process consists of making sure that the size of the JAR file matches the appropriate JAD file entry, as well as ensuring that the entries in the manifest and JAD are in sync.

## Requesting permissions

One additional point is worth mentioning. Since the previous example was only intended to demonstrate a means of configuring security options in the WTK, we didn't actually access a protected function or API. In the real world, there is an additional configuration step necessary to specify which protected resource we are after.

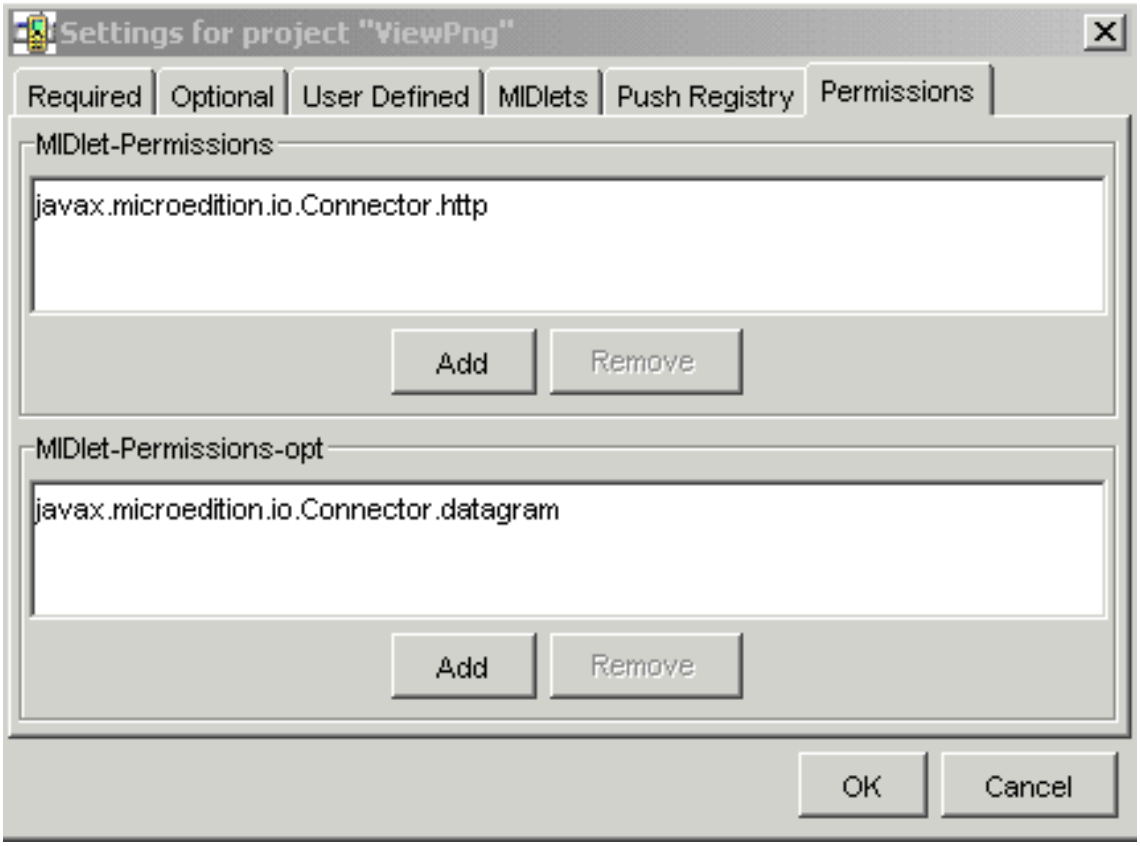
Before configuring our system, let's look at the two types of permissions:

---

## Setting permissions in the WTK

The Wireless Toolkit provides a tool to help you set permissions.

- From the KToolbar, click on Settings
- Click the Permissions tab
- Use the dialog box to add permissions



Any entries that are configured here will be added to the JAD file (in the `wtk20\apps\PROJECT_NAME\bin` directory).

---

## Security summary

Although we went through a fair number of steps to sign a MIDlet and verify that the device emulator does indeed validate the certificate, we've only touched the surface of security-related features in the WTK.

Should you create a MIDlet that you would like to distribute, you'll want to take a much closer look at this topic. The good news is that the documentation that is included with the WTK is quite comprehensive and will answer the great majority of your questions.

# Section 6. Push registry

## Introduction

MIDP 2.0 includes support for notifying MIDlets of incoming connections. The basic idea is that, when an incoming network connection attempts to communicate with the MIDlet, that MIDlet will be notified by the Application Manager. If the MIDlet is already running, the MIDlet will process the network communication. If the MIDlet is not active, the Application Manager will start the MIDlet through a call to the `startApp()` method.

In order for a MIDlet to receive an inbound request, it must register a list of valid connections. This is accomplished by adding entries to the JAD file or by calling the `registerConnection()` method when the MIDlet is running.

---

## JAD entries

When adding entries to the connection list inside a JAD file, you must use the following format:

```
MIDlet-Push-<n>: <ConnectionURL>, <MIDletClassName>, <AllowedSender>
```

Where:

- `ConnectionURL` is the URL of the connection.
- `MIDletClassName` identifies which MIDlet in the suite is associated with the connection.
- `AllowedSender` identifies which senders (that is, which sources of the inbound connection) are allowed to connect through the URL. Wild cards are supported. For example, `*` would indicate that all senders are welcome, whereas `128.1.3.*` would limit the incoming connections to the specified subnet.

Some examples:

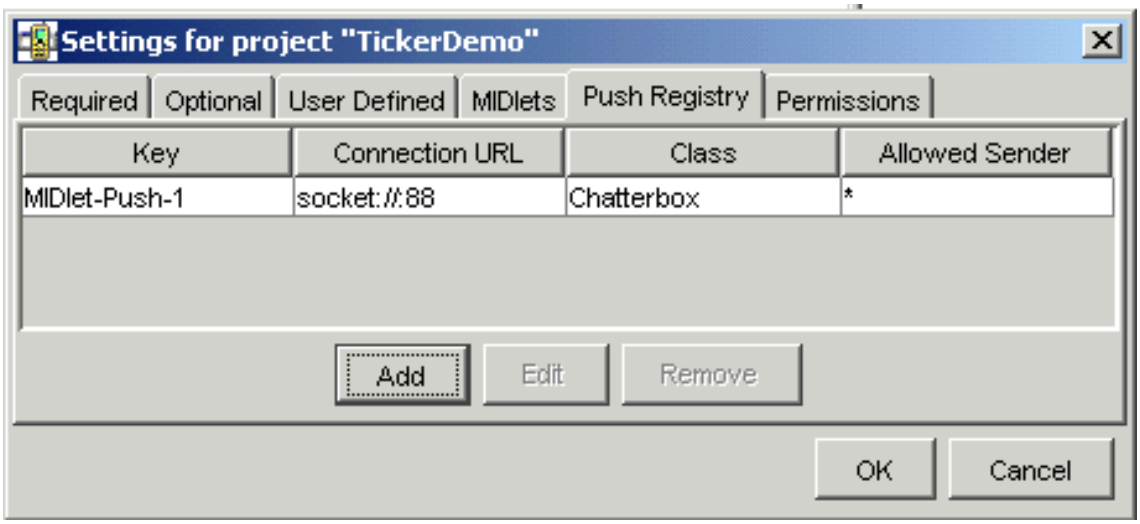
```
MIDlet-Push-1: socket://:88, Chatterbox, *
MIDlet-Push-2: datagram://:50000, Chatterbox, *
```

In the two lines above, we are specifying that the MIDlet with the name Chatterbox will accept either a socket or datagram connection (from the ports shown) from any sender.

## Push registry inside the WTK

Within the WTK, you can configure the outside connections that a MIDlet will accept, like so:

- Select Settings from the KToolbar menu
- Click the Push Registry tab




In the figure above, we have added a sample entry into the registry. If, after entering the information shown, you examine the JAD file for the project (in the `wtk20\apps\PROJECT_NAME\bin` directory), you'll see that it contains a new entry:

```
MIDlet-Push-1: socket://:88, Chatterbox, *
```

## Push registry summary

This quick overview skims the surface of the WTK's support for working with inbound connections. As with the security features we discussed, the Application Manager provides an exhaustive feature set to test your MIDlets' push features, including the ability to launch a MIDlet based on a connection request from a remote resource. It's pretty cool stuff.

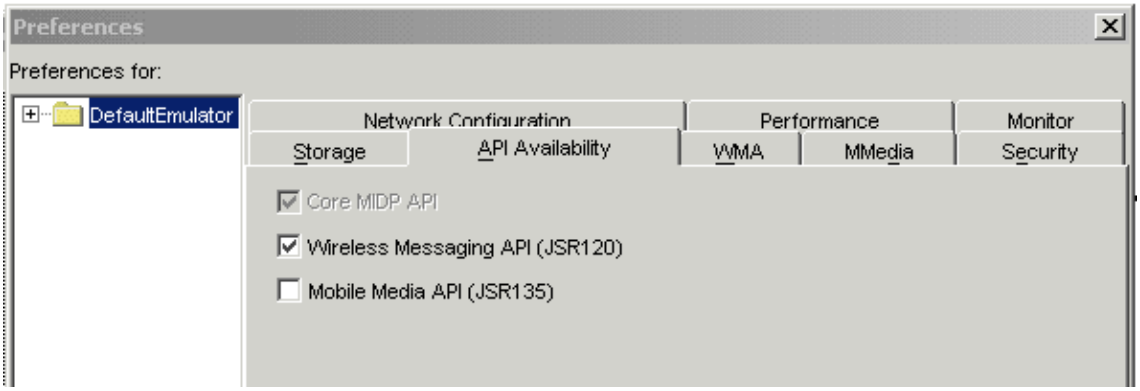
# Section 7. Wireless messaging

## Introduction

MIDP 2.0 does not support wireless messaging (such as Short Message Service) as part of the specification. However, because the Wireless Message API (WMA) is an optional package for J2ME, the WTK includes support for developing MIDlets using WMA.

You can configure the support for WMA as follows:

- Select Edit from the KToolbar menu
- Choose Preferences from the pull-down men
- Click the API Availability tab



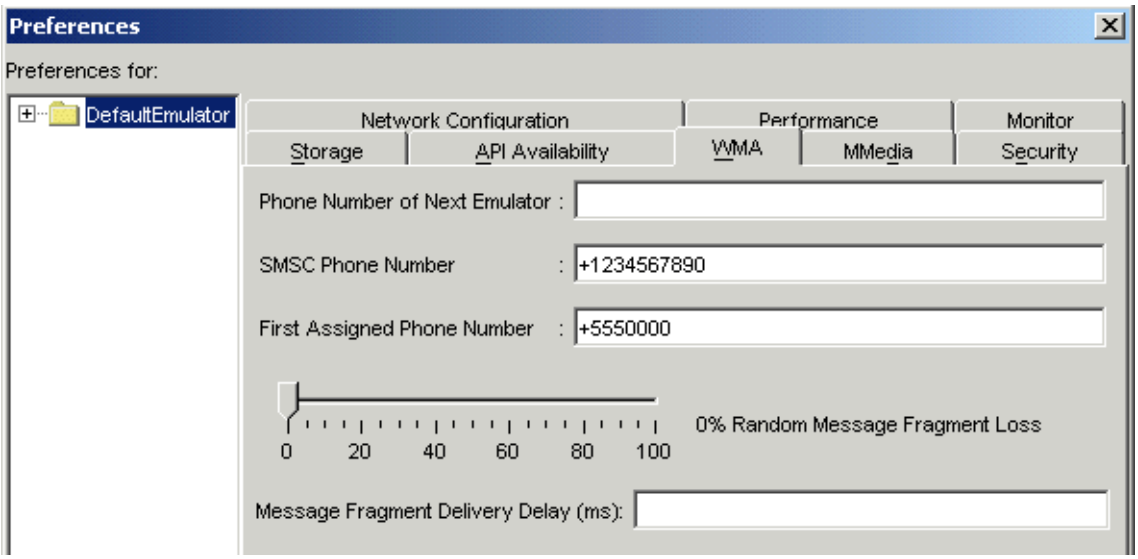

Make sure the Wireless Messaging API box is checked, as in the figure above.

---

## Wireless messaging: Testing

If you develop a MIDlet that supports wireless messaging, there are several configuration options available to assist in the testing process:

- Select Edit from the KToolbar menu
- Choose Preferences from the pull-down menu
- Click the WMA tab




Although we don't have the space here to walk through a complete example, this dialog box is where you would configure the emulator to support sending and receiving messages between devices (essentially, transmitting messages between multiple emulators). This is a handy feature if you are developing applications that allow communication across devices.

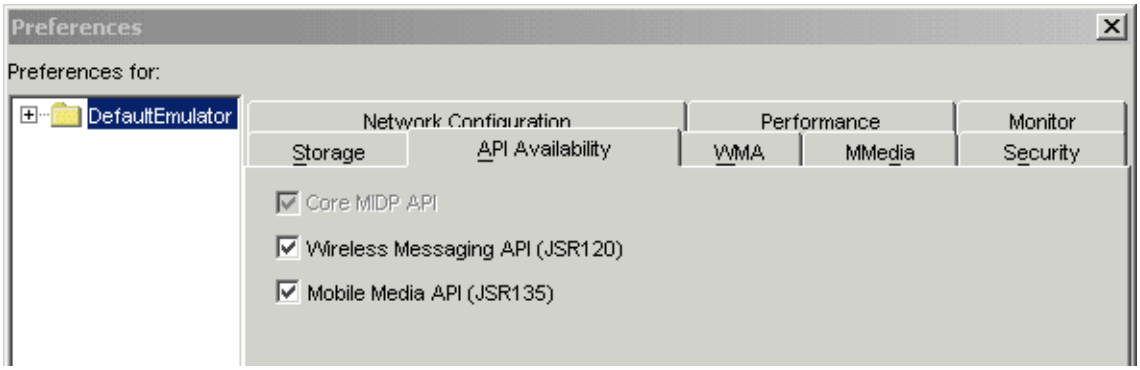
# Section 8. Audio support

## Introduction

The Mobile Media API (MMA) is for high-end J2ME devices that support advanced sound capabilities. For those devices that are limited in their sound support, MIDP includes a subset of MMA.

The WTK supports the development of MIDlets using either the full MMA or only the APIs that are part of the MIDP 2.0 specification. You can configure your support level within the Preferences menu.

- Select Edit from the KToolbar menu
- Choose Preferences from the pull-down menu
- Click the API Availability tab, as shown in the figure below



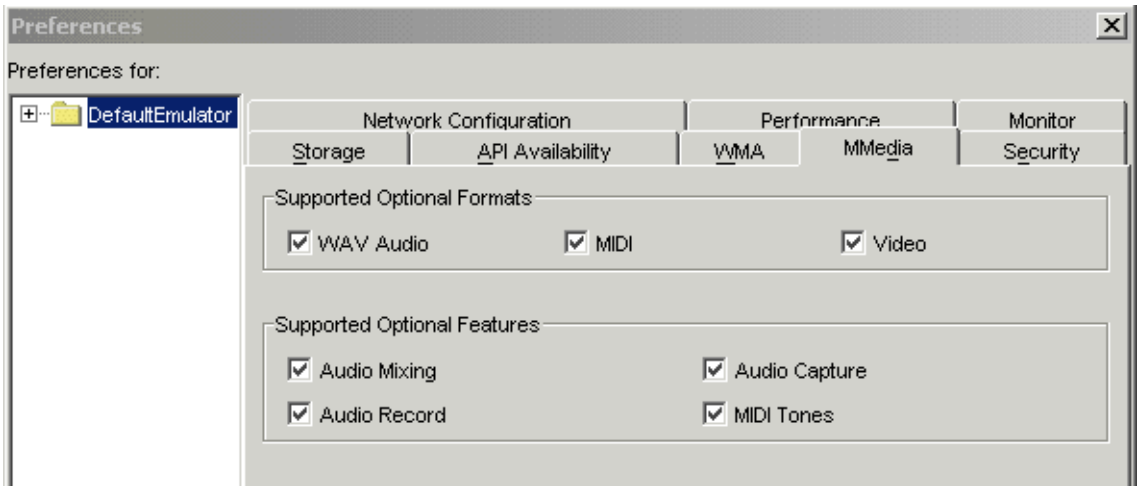

From here, you can select how much support you would like regarding the Mobile Media API -- that is, you can choose between core MIDP functionality or the entire Mobile Media API.

---

## Audio configuration

In addition to letting you configure for full or partial support of the Mobile Media API, the WTK also allows you to determine whether you would like to have support for additional file formats as well as various optional features.

- Select Edit from the KToolbar menu
- Choose Preferences from the pull-down menu
- Click the MMedia tab




From here you can choose the features that you would like to have accessible to your MIDlets.

# Section 9. Gaming API

## Overview

With the limited processing power of many J2ME devices, support for graphics with MIDP 1.0 was quite limited. This presented a challenge for game developers. With MIDP 2.0, a specific gaming API has been added to offer much more complete graphics support. It includes the following classes:

- `GameCanvas`: Provides the basic platform for a game user interface
- `Layer`: Represents a visual element of a game
- `LayerManager`: Manages one or more `Layers`
- `Sprite`: A visual element that can be animated, flipped, rotated, etc.
- `TiledLayer`: A grid of cells that can be filled with a set of images

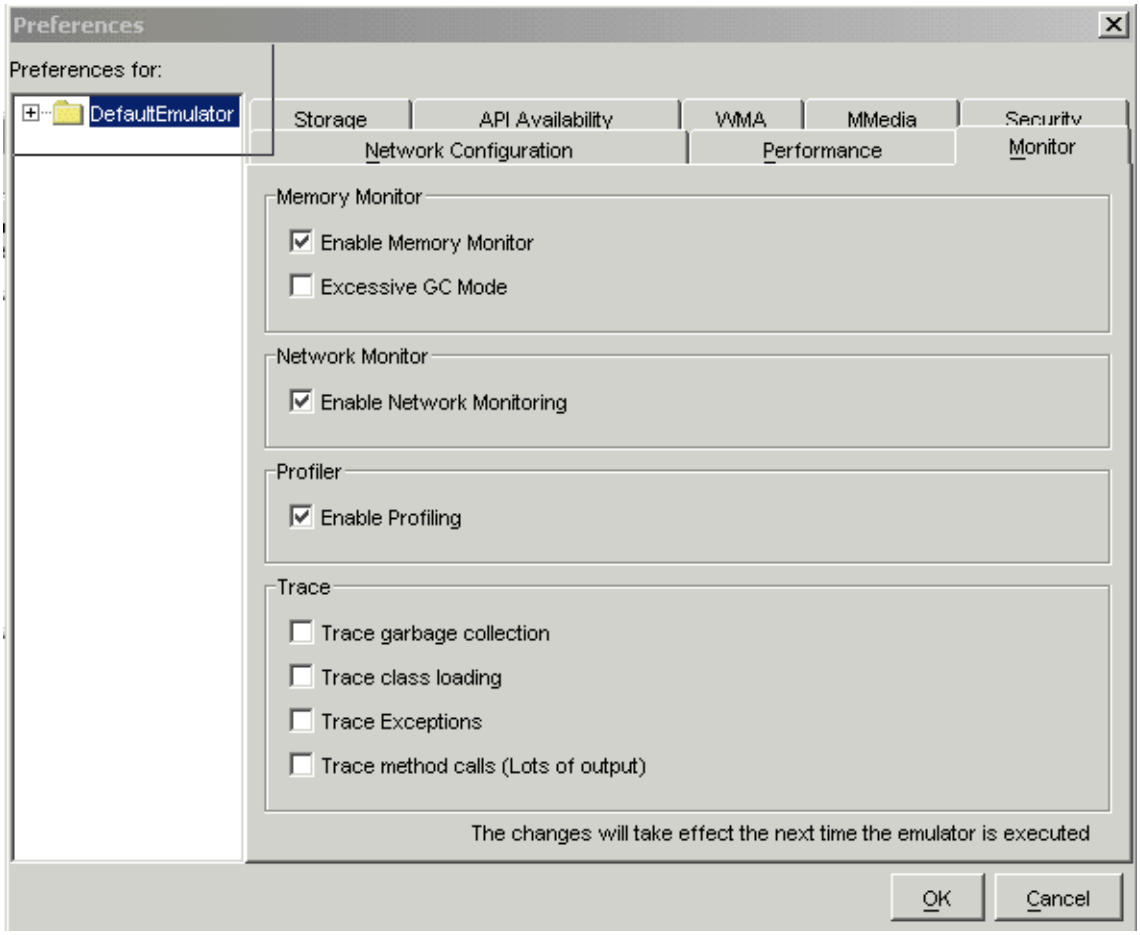
An additional gaming-related feature in MIDP 2.0 is support for storing images as an array of integers. This allows direct access to the image itself, facilitating manipulation of an image in memory.

# Section 10. Monitoring performance

## Introduction

Within the WTK, you can monitor application performance by viewing memory usage, watching network traffic, and examining method execution time. To enable any or all of the monitors:

- Select Edit from the KToolbar menu
- Choose Preferences from the pull-down menu
- Click the Monitor tab

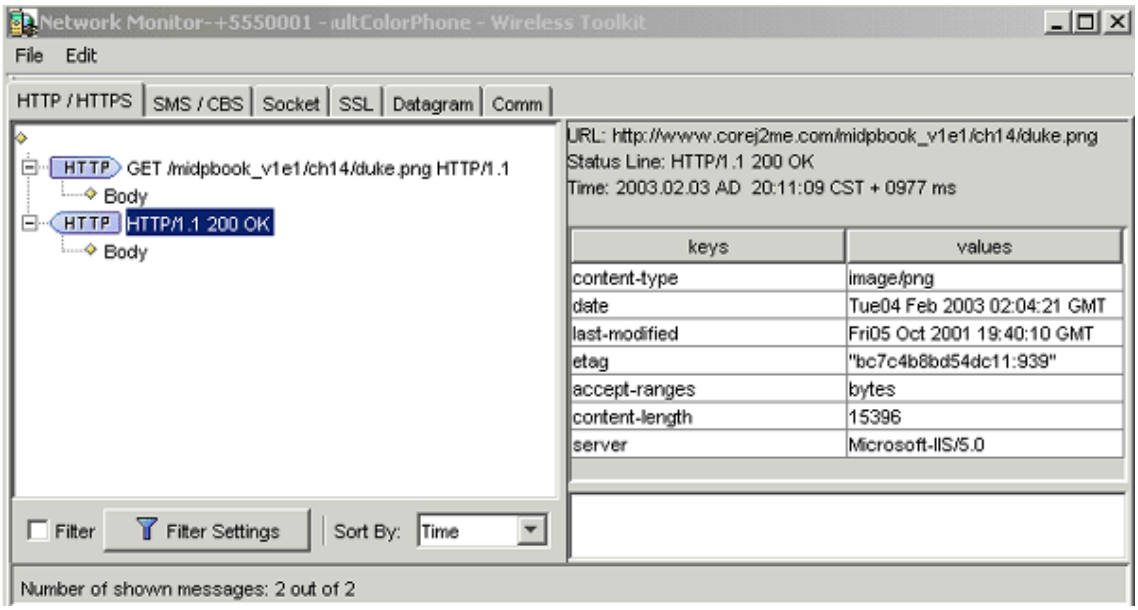



You can choose the options you'd like to keep track of by selecting the Network Monitor, the Memory Monitor, or the Profiler (which monitors application timing). When you run your MIDlet, an additional window will open for each monitor that you choose.

# Network Monitor

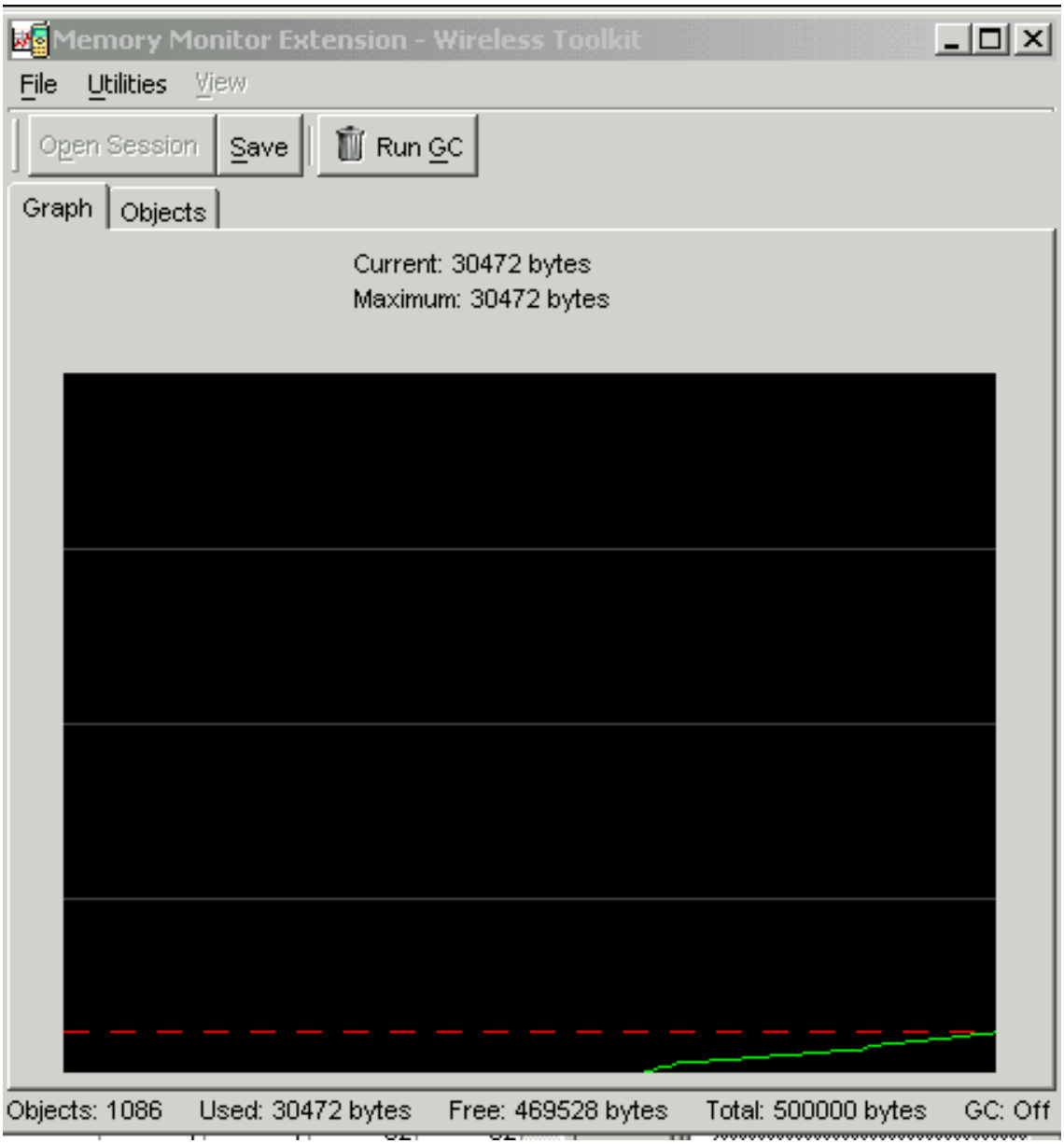
Through network monitoring, you can get a close look at the network traffic between your MIDlet and a remote resource. You can view traffic for the HTTP, HTTPS, datagrams, sockets, secure socket layers (SSL), comm, SMS, and CBS protocols.

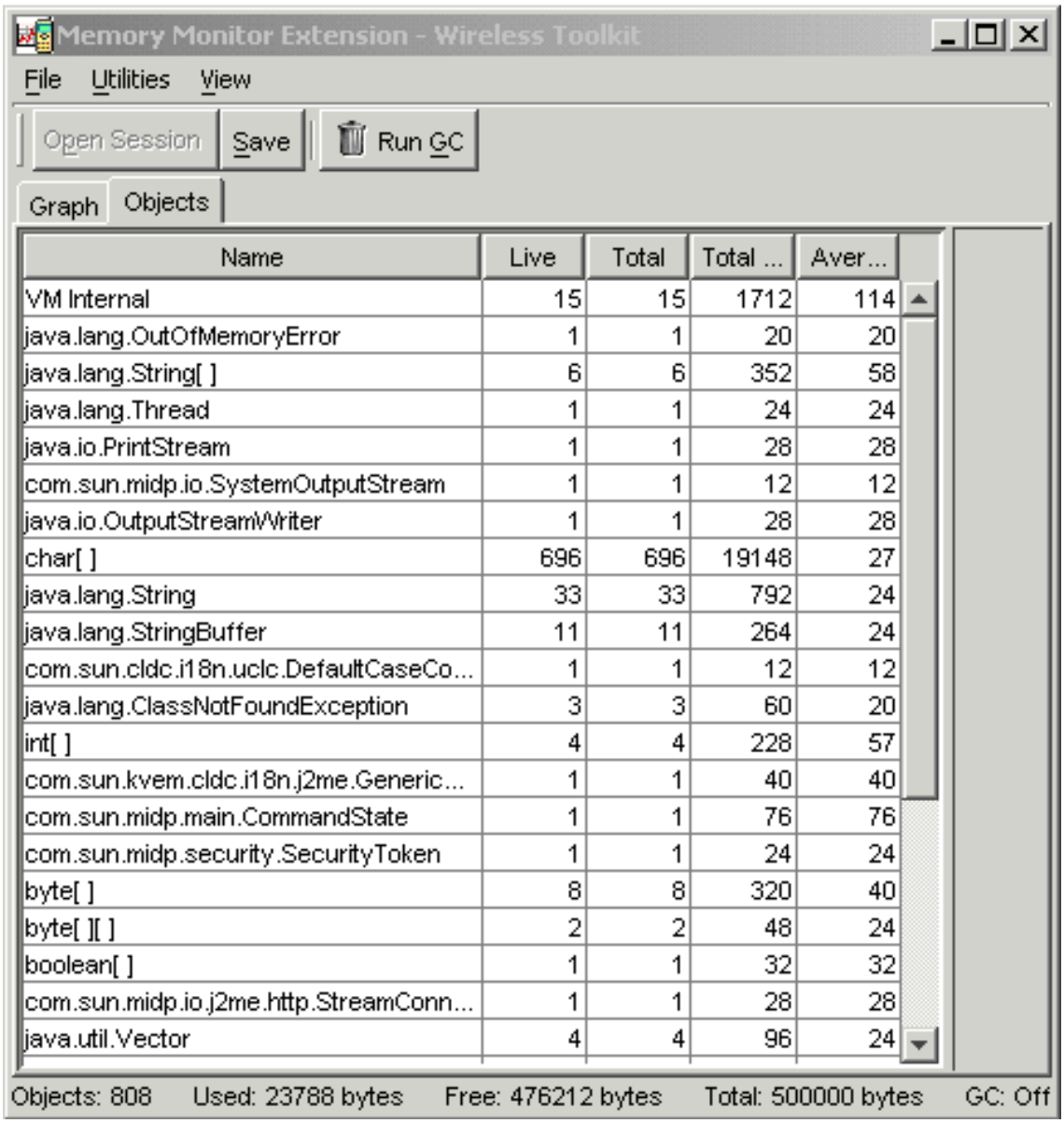
The Network Monitor displays both incoming and outgoing messages. You can also apply various filters to drill down to a specific set of messages.




# Memory Monitor

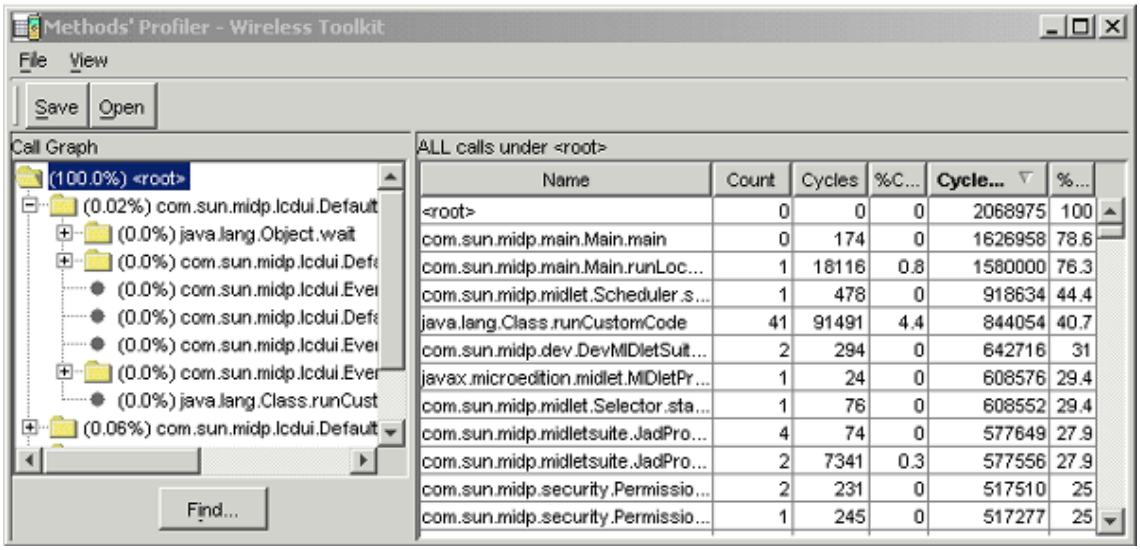
With the Memory Monitor, you can track how much memory your application uses. You can view memory usage in a graph or table format, with the latter showing memory allocation on a per-object basis.


Memory Monitor Extension - Wireless Toolkit
File
Utilities
View
Open Session
Save
Run GC
Graph
Objects
Current: 30472 bytes
Maximum: 30472 bytes
Objects: 1086
Used: 30472 bytes
Free: 469528 bytes
Total: 500000 bytes
GC: Off




## Application Profiler

Using the Application Profiler, you can view how much time each method within your MIDlet takes to execute. This is an excellent tool for identifying performance bottlenecks.


Methods' Profiler - Wireless Toolkit
File View
Save Open
Call Graph
(100.0%) <root>
(0.02%) com.sun.midp.lcdui.Default
(0.0%) java.lang.Object.wait
(0.0%) com.sun.midp.lcdui.Def
(0.0%) com.sun.midp.lcdui.Ever
(0.0%) com.sun.midp.lcdui.Def
(0.0%) com.sun.midp.lcdui.Ever
(0.0%) com.sun.midp.lcdui.Ever
(0.0%) java.lang.Class.runCust
(0.06%) com.sun.midp.lcdui.Default
Find...
ALL calls under <root>
Name Count Cycles %C... Cycle... %...
<root> 0 0 0 2068975 100
com.sun.midp.main.Main.main 0 174 0 1626958 78.6
com.sun.midp.main.Main.runLoc... 1 18116 0.8 1580000 76.3
com.sun.midp.midlet.Scheduler.s... 1 478 0 918634 44.4
java.lang.Class.runCustomCode 41 91491 4.4 844054 40.7
com.sun.midp.dev.DevMIDletSuit... 2 294 0 642716 31
javax.microedition.midlet.MIDletPr... 1 24 0 608576 29.4
com.sun.midp.midlet.Selector.sta... 1 76 0 608552 29.4
com.sun.midp.midletsuite.JadPro... 4 74 0 577649 27.9
com.sun.midp.midletsuite.JadPro... 2 7341 0.3 577556 27.9
com.sun.midp.security.Permissio... 2 231 0 517510 25
com.sun.midp.security.Permissio... 1 245 0 517277 25

# Section 11. Summary and resources

## New features in MIDP 2.0

As you might assume, the WTK supports all the new features that are included in the MIDP 2.0 specification. A few of these new features, not previously mentioned, are listed here:

- Enhanced support for laying out components onto a `Form`
- Ability to create custom `Item` components that can be placed onto a `Form`
- `Spacer`, a new `Item` that provides blank space between components
- `POPUP`, an added style for `ChoiceGroup` that mimics a combo box
- HTTPS support
- Many additional minor updates.

---

## Summary

The Wireless Toolkit provides an excellent starting point for developing applications with J2ME and MIDP. In this tutorial, we began by covering the basics of downloading, installing, and creating a simple MIDlet. This was followed by a close look inside over-the-air (OTA) provisioning, allowing you to simulate the process of packaging a MIDlet suite, installing it onto a Web server, and locating, downloading, and running it on a device.

Security-related features have been greatly enhanced in WTK 2.0. In this tutorial, you had the chance to sign a MIDlet and test the validation of a certificate through OTA provisioning. We also discussed how to configure the WTK when a class needs access to protected APIs.

Push registry, wireless messaging, audio, and gaming support are all new with MIDP 2.0. Here we had a chance to review the basic features of each. We concluded this tutorial with an overview showing how to monitor the performance of your MIDlet using the Network and Memory Monitors, as well as the application profiler.

At this point, armed with a basic knowledge of the tools available with the WTK, you can concentrate your efforts on learning the MIDP API and writing code. Maybe you'll be the one to write the first killer wireless application. Good luck!

---

# Links and Resources

# Section 12. Feedback

# Feedback

Please send us your feedback on this tutorial. We look forward to hearing from you!

---

## Colophon

This tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial generator. The open source Toot-O-Matic tool is an XSLT stylesheet and several XSLT extension functions that convert an XML file into a number of HTML pages, a zip file, JPEG heading graphics, and two PDF files. Our ability to generate multiple text and binary formats from a single source file illustrates the power and flexibility of XML. (It also saves our production team a great deal of time and effort.)

You can get the source code for the Toot-O-Matic at www6.software.ibm.com/dl/devworks/dw-tootomatic-p. The tutorial Building tutorials with the Toot-O-Matic demonstrates how to use the Toot-O-Matic to create your own tutorials. developerWorks also hosts a forum devoted to the Toot-O-Matic; it's available at www-105.ibm.com/developerworks/xml_df.nsf/AllViewTemplate?OpenForm&RestrictToCategory=11. We'd love to know what you think about the tool.