

Threads and networking in J2ME

Skill Level: Intermediate

[John Muchow \(john@corej2me.com\)](mailto:john@corej2me.com)
Freelance Technical Writer

16 Mar 2004

Learn how to use threads in MIDlets to communicate over a network connection. Without multithreading, a MIDlet that requests a network connection blocks while waiting for a response from the network. In the real world, a user expects the application to continue running, even while a network connection is underway.

Section 1. Before you start

About this tutorial

This tutorial teaches you how to use threads in MIDlets to communicate over a network connection. Without multithreading, a MIDlet that requests a network connection blocks while waiting for a response from the network. In the real world, a user expects the application to continue running, even while a network connection is underway.

I begin the tutorial by demonstrating the problems of non-threaded MIDlets. From there, you learn what you need to know about threading, including the two approaches for working with threads in Java

In the final section of the tutorial, you'll build a multithreaded MIDlet. This application prompts the user for the URL of an image to download. Once the network connection has been initiated, and the download started, the user is returned to the main application interface. From here, even while the download is in progress, the user is able to move about the interface, including entering a new URL path. Upon completion of the download, the MIDlet notifies the user with a dialog box indicating success (hopefully, rather than failure), and the downloaded image is displayed.

After completing the tutorial you should have a solid understanding of the importance of multithreading and a foundation to build upon for creating your own threaded MIDlets.

Prerequisites

You'll need two software tools to complete this tutorial:

- **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 I will be here), you'll need to download JDK version 1.4 or greater. Download [JDK version 1.4.1](#).
- **The Wireless Toolkit (WTK):** The Sun Microsystems Wireless Toolkit (WTK) is an integrated development environment (IDE) for creating Java 2 Platform, Micro Edition (J2ME) MIDlets. The WTK download contains an IDE, as well as the libraries required for creating MIDlets. Download [J2ME Wireless Toolkit 2.0](#).

Install the software

The Java Development Kit (JDK)

Use the JDK documentation to install the JDK. You can choose either the default directory or specify another directory. If you choose to specify a directory, make a note of where you install the JDK. During the installation process for the Wireless Toolkit, the software attempts to locate the Java Virtual Machine (JVM); if it cannot locate the JVM, you are prompted for the JDK installation path.

The Wireless Toolkit (WTK)

This tutorial builds on an earlier developerWorks tutorial "MIDlet Development with the Wireless Toolkit" (see [Resources](#)), which explains the basics of creating MIDlets with the toolkit. This tutorial is an excellent starting point if you are new to the Wireless Toolkit.

The Wireless Toolkit is contained within a single executable file. Run this file to begin the installation process. It is recommended that you use the default installation directory. However, if you do not use the default directory, make sure the path you select does not include any spaces.

Section 2. Non-threaded MIDlet

Overview

It seems best to begin with the problem at hand, namely, understanding why you need to use multiple threads when building MIDlets that communicate over a network. This first section shows a short application with both a trivial user interface and a lackluster php script; nevertheless, you'll get the point regarding the importance of threading.

This MIDlet displays a quote that is stored on a remote server. There are a total of nine quotes available. The user enters a number between one and nine to retrieve and display one of the quotes. The goal of the MIDlet is to demonstrate how the application blocks once the network request is initiated.

Let's begin by creating the project. You'll see the application screenshots to see where the problem occurs once you have the program up and running.

Create MIDlet

For each MIDlet developed in this tutorial, follow the same set of steps you use when using the Wireless Toolkit:

1. Create the project.
2. Write the source code.
3. Compile and preverify the code.
4. Run the MIDlet.

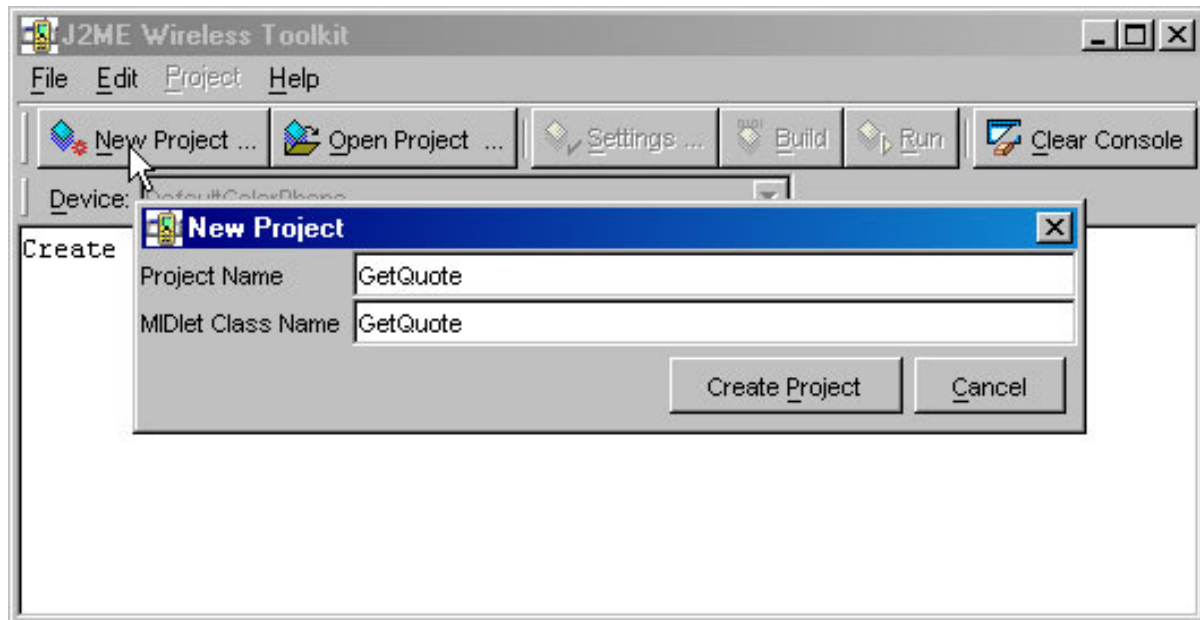
You'll start by creating a project in the next panel.

Create the project

1. Click **New Project**.
2. Enter the project name and MIDlet class name, as shown in Figure 1.

3. Click **Create Project**.

Figure 1. Create GetQuote project



Write the Java code

Begin by copying and pasting the following GetQuote.java code into a text editor:

```

/*-----
 * GetQuote.java
 *-----*/
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import java.io.*;

public class GetQuote extends MIDlet implements CommandListener
{
    private Display display; // Reference to Display object
    private Form fmMain; // The main form
    private Command cmRetrieve; // Command to get quote
    private Command cmExit; // Command to exit MIDlet
    private TextField tfQuote; // Phone number
    private String url = "http://www.corej2me.com/ibm/quotes2.php";

    /*-----
    * Constructor
    *-----*/
    public GetQuote()
    {
        display = Display.getDisplay(this);

        // Create commands
        cmRetrieve = new Command("Retrieve", Command.SCREEN, 1);
        cmExit = new Command("Exit", Command.EXIT, 1);
    }
}

```

```

// Textfield to prompt for quote number
tfQuote = new TextField("Get quote between 1 and 9:", "", 1,
    TextField.NUMERIC);

// Create Form, add Commands & textfield, listen for events
fmMain = new Form("Quote MIDlet");
fmMain.addCommand(cmExit);
fmMain.addCommand(cmRetrieve);
fmMain.append(tfQuote);
fmMain.setCommandListener(this);
}

// Called by application manager to start the MIDlet.
public void startApp()
{
    display.setCurrent(fmMain);
}

public void pauseApp()
{ }

public void destroyApp(boolean unconditional)
{ }

public void commandAction(Command c, Displayable s)
{
    if (c == cmRetrieve)
    {
        try
        {
            // Connect to the server, passing in request quote
            String str = connect(tfQuote.getString());

            fmMain.append("Quote: " + str);
        }
        catch (Exception e)
        {
            System.out.println("Unable to get quote " + e.toString());
        }
    }
    else if (c == cmExit)
    {
        destroyApp(false);
        notifyDestroyed();
    }
}

/*-----
* Connect to the remote server, run the php script
* and return the request quote.
*-----*/
private String connect(String number) throws IOException
{
    InputStream iStrm = null;
    ByteArrayOutputStream bStrm = null;
    HttpConnection http = null;
    String quote_string = null;

    try
    {
        // Create the connection
        http = (HttpConnection) Connector.open(url + "?number=" + number);

        System.out.println(url + "?number=" + number);

        //-----
        // Client Request
        //-----
        // 1) Send request method

```

```
http.setRequestMethod(HttpConnection.GET);

// If you experience connection/IO problems, try
// removing the comment from the following line
//http.setRequestProperty("Connection", "close");

// 2) Send header information

// 3) Send body/data - No data for this request

//-----
// Server Response
//-----
// 1) Get status Line
if (http.getResponseCode() == HttpConnection.HTTP_OK)
{
    // 2) Get header information

    // 3) Get data from server

    iStrm = http.openInputStream();
    int length = (int) http.getLength();

    int ch;
    bStrm = new ByteArrayOutputStream();
    while ((ch = iStrm.read()) != -1)
    {
        // Ignore any carriage returns/linefeeds
        if (ch != 10 && ch != 13)
            bStrm.write(ch);
    }

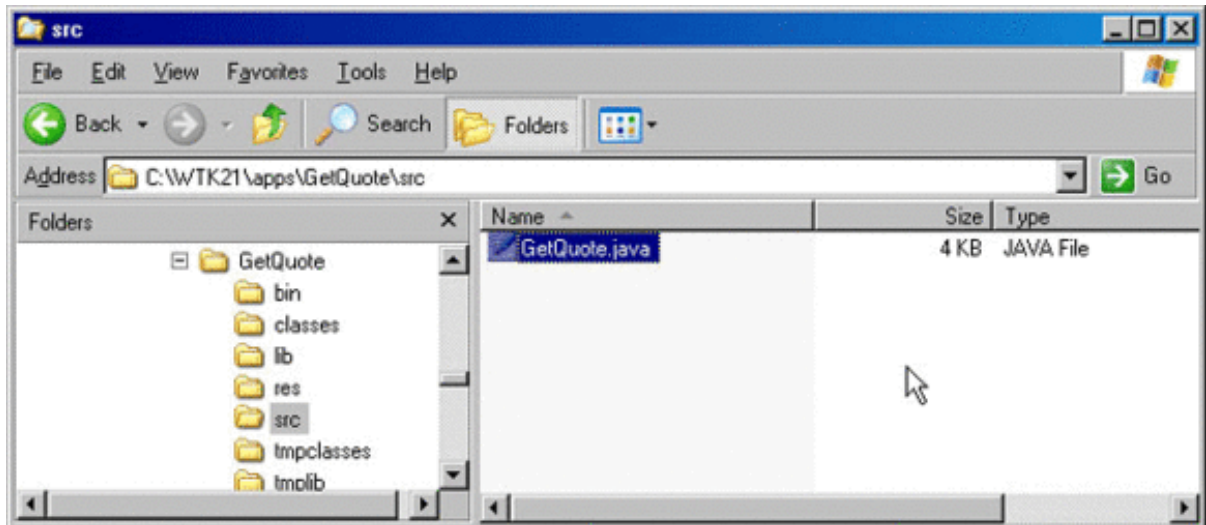
    quote_string = new String(bStrm.toByteArray());
}
}
finally
{
    // Clean up
    if (iStrm != null)
        iStrm.close();
    if (bStrm != null)
        bStrm.close();
    if (http != null)
        http.close();
}

return quote_string;
}
}
```

Save the Java code

When you create a new project, the WTK builds the proper directory structure for you. In this example, the WTK has created the C:\WTK21\apps\GetQuote directory and the necessary subdirectories. Save your Java source file as GetQuote.java in the src directory as shown in Figure 2. (Note that the drive and WTK directory vary depending on where you have installed the toolkit.)

Figure 2. Save GetQuote code



You'll review the details of the code in an upcoming section.

Write the PHP script

Now, write a short PHP script to return a quote based on the number submitted by the user. Copy and paste the following script into a text editor:

```
<?php

$quotes[] = "I think there is a world market for maybe five computers.
    Thomas Watson.";
$quotes[] = "The answer to life's problems aren't at the bottom of a
    bottle, they're on TV! Homer Simpson.";
$quotes[] = "I love California, I practically grew up in Phoenix.
    Dan Quayle.";
$quotes[] = "640K ought to be enough for anybody. Bill Gates.";
$quotes[] = "Some people are afraid of heights. Not me, I'm afraid
    of widths. Steven Wright";
$quotes[] = "Start every day off with a smile and get it over
    with. W. C. Fields ";
$quotes[] = "I am not a vegetarian because I love animals;
    I am a vegetarian because I hate plants. A. Whitney Brown";
$quotes[] = "A conclusion is simply the place where someone got
    tired of thinking. Arthur Block";
$quotes[] = "A rich man's joke is always funny. Proverb.";

//-----
// It all starts here.
// Get requested quote number parameter passed in the url
//-----
if (isset($_GET))
{
    $quote_num = $_GET["number"];
}

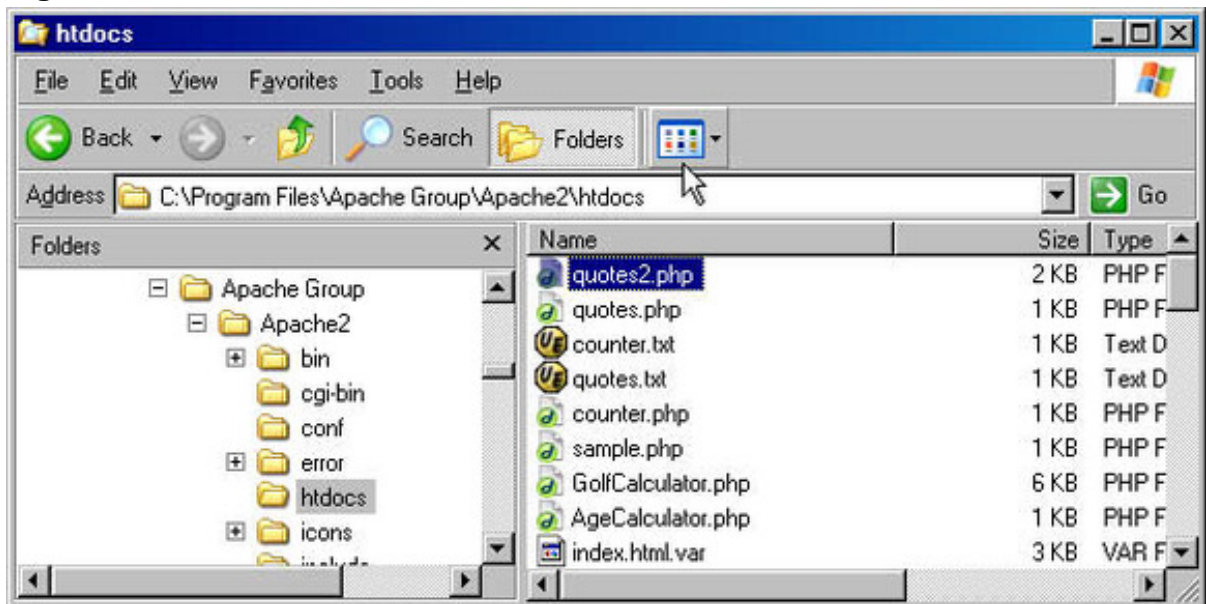
// Create a delay to make the problem of no-threads more obvious
sleep(5);
// User entered a quote number
if ($quote_num > 0)
{
```

```
// Decrement the quote requested because our array is zero based
echo $quotes[$quote_num - 1];
}
else // User did NOT enter a quote number, return a random quote
{
    // Initialize random number generator
    srand((double)microtime() * 1000000);
    // Get a random value that is between 0 and the length of our array
    (minus 1) rand_num = rand(0, (count($quotes) - 1));
    // Echo the quote from the array
    echo $quotes[$rand_num];
}
?>
```

Save the PHP code

Save the PHP source code as quotes2.php in the doc root on the Web server. Figure 3 shows the default location when installing Apache using the default configuration options.

Figure 3. Save GetQuotes PHP code

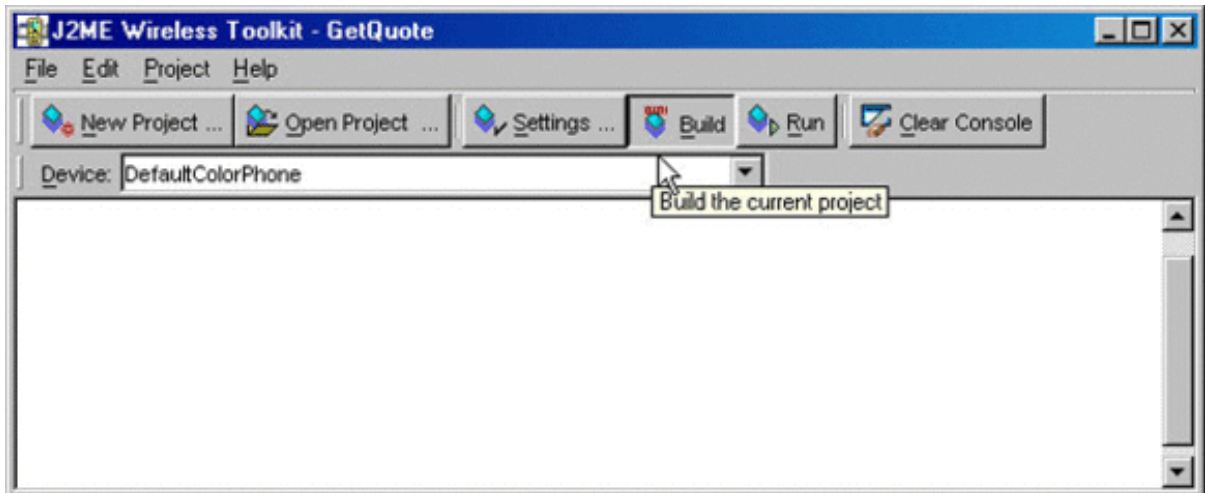


You'll look at the specifics of what is occurring in the PHP script in an upcoming section.

Compile and preverify

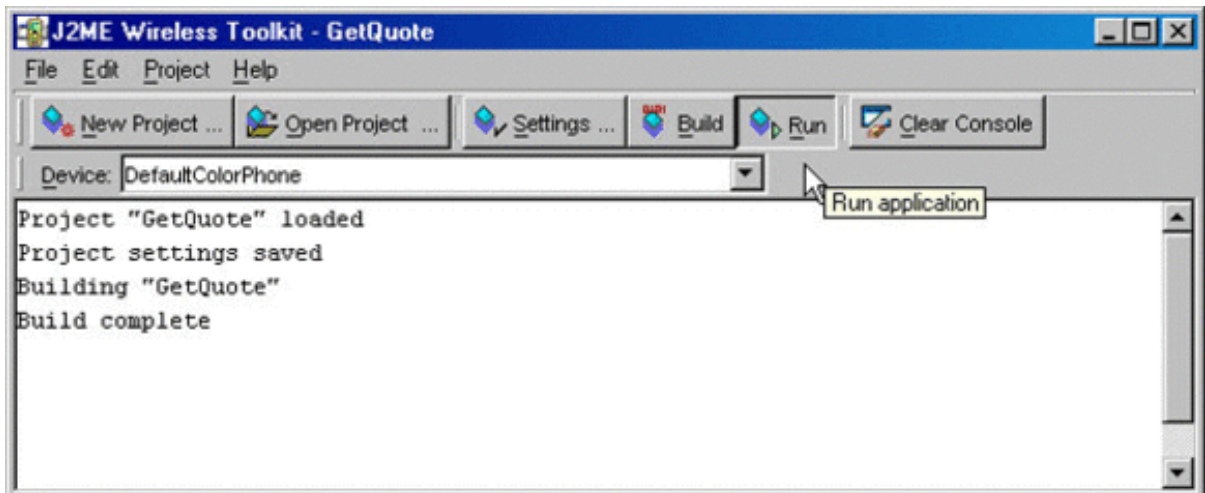
Click **Build** to compile, preverify, and package the MIDlet, as shown in Figure 4.

Figure 4. Build GetQuote project



Click **Run** to start the Application Manager.

Figure 5. Run GetQuote project



Start the GetQuote MIDlet

To start the GetQuote MIDlet, click **Launch**, as shown in Figure 6.

Figure 6. Launch GetQuote MIDlet



With the MIDlet running, I'll demonstrate where the problem lies. Figure 7 shows the textfield that prompts for a number between 1 and 9. This value represents which quote to retrieve from the server. The moment you select the Retrieve command, the network connection starts. Although difficult to show in a static screenshot, if you run the MIDlet you notice that you cannot perform any action on the MIDlet while the remote server is accessed. This includes the inability to select a different number in the textfield, as well as to exit the MIDlet!

Figure 7. Retrieving quotes



This is when the problem occurs. Once you select 'Retrieve' the application will block, waiting for the network connection to return the requested quote.

As an experiment, when running the MIDlet, try to change the quote number in the TextBox. You'll notice the MIDlet will not respond until the network connection is complete.

After the MIDlet has received a response from the server, the display is updated with the requested quote. See Figure 8.

Figure 8. Display quote



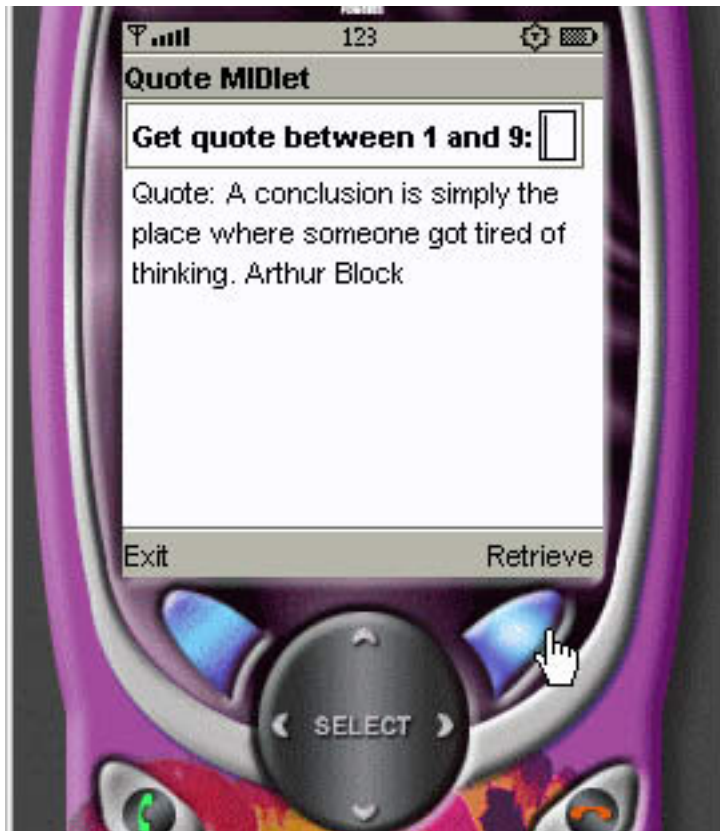
Once the quote has been retrieved from the server, and displayed on the device, the MIDlet interface is no longer locked waiting for the network connection to complete.

This is demonstrated in this screen shot by showing how the textfield contents can be cleared.

GetQuote MIDlet caveat

As a way to ensure the user doesn't attempt to stump the quote server, should the user elect to select the Retrieve option without entering a number in the textfield, the PHP script is smart enough to return a random quote. Figure 9 demonstrates this idea by retrieving and displaying a quote with no user input.

Figure 9. Get random quote



Code review: commandAction()

Let's briefly review how events are processed and the specifics of connecting to the server where the PHP script resides. Below is the code for processing commands. Notice the call to the connect() method, passing in the value from the textfield where the user entered their desired quote.

```
public void commandAction(Command c, Displayable s)
{
    if (c == cmRetrieve)
    {
        try
        {
            // Connect to the server, passing in request quote
            String str = connect(tfQuote.getString());
        }
    }
}
```

```

        fmMain.append("Quote: " + str);
    }
    catch (Exception e)
    {
        System.out.println("Unable to get quote " + e.toString());
    }
}
else if (c == cmExit)
{
    destroyApp(false);
    notifyDestroyed();
}
}

```

Inside the `connect()` method (which you'll see in the next panel) is where the heart of the code lies for connecting to the server and processing the server output. In the code above, after retrieving the server response, the display is updated by appending the requested quote onto the device.

Code review: `connect()`

If you have done any amount of communication over a network connection with J2ME, most of the code should look familiar. Notice how I pass the user requested quote number as part of the URL. For example, if the user asks for quote number 5, the resulting URL is: `http://www.corej2me.com/ibm/quotes2.php?number=5`. With the URL in hand, I simply open the connection, specify the request type (GET versus POST), and wait for the response.

```

private String connect(String number) throws IOException
{
    InputStream iStrm = null;
    ByteArrayOutputStream bStrm = null;
    HttpURLConnection http = null;
    String quote_string = null;

    try
    {
        // Create the connection
        http = (HttpURLConnection) Connector.open(url + "?number=" + number);

        System.out.println(url + "?number=" + number);

        //-----
        // Client Request
        //-----
        // 1) Send request method
        http.setRequestMethod(HttpURLConnection.GET);

        // If you experience connection/IO problems, try
        // removing the comment from the following line
        //http.setRequestProperty("Connection", "close");

        // 2) Send header information

        // 3) Send body/data - No data for this request

        //-----
        // Server Response
    }
}

```

```
//-----  
// 1) Get status Line  
if (http.getResponseCode() == HttpURLConnection.HTTP_OK)  
{  
    // 2) Get header information  
  
    // 3) Get data from server  
  
    iStrm = http.openInputStream();  
    int length = (int) http.getLength();  
  
    int ch;  
    bStrm = new ByteArrayOutputStream();  
    while ((ch = iStrm.read()) != -1)  
    {  
        // Ignore any carriage returns/linefeeds  
        if (ch != 10 && ch != 13)  
            bStrm.write(ch);  
    }  
  
    quote_string = new String(bStrm.toByteArray());  
}  
}  
finally  
{  
    // Clean up  
    if (iStrm != null)  
        iStrm.close();  
    if (bStrm != null)  
        bStrm.close();  
    if (http != null)  
        http.close();  
}  
  
return quote_string;  
}
```

Once the server responds, I check the status code to see if the connection request was successful. If so, I read the server response and store the result in the variable `quote_string`. Near the end of the `connect()` method you'll notice I return this same value from the method. If you go back to the previous panel, you'll see that the return value from the code shown above is what is displayed on the device.

This completes the loop for the MIDlet. You've requested a quote, sent the request to the server, and read and displayed the server response. All that is left is to review the PHP script that generates the requested quote. The next panel discusses the PHP code.

Code review: PHP script

To generate a quote, I've taken the easy way out by simply creating an array and populating each entry with a quote. A much more robust way to accomplish this would be to read the quotes from a file. Though not too difficult, I thought it best to keep the focus on the task at hand -- how to get the parameter passed in from the MIDlet and to return a quote that matches the request.

The first few lines show the nine quotes that are available, and where each is stored in a separate array entry. Begin by checking to see if a parameter was passed in from the MIDlet, If yes, get the value and store it in the variable `quote_num`. Next, you'll notice a call to `sleep()`, a request that asks the PHP script to pause for five seconds. You want to add the delay to ensure that the code on the other end (the MIDLET) has an obvious pause while waiting for this PHP script to complete. The intention is to show how the MIDlet blocks, and without this sleep request, those with a very fast network connection with this short PHP script could execute so quickly it would appear as though no blocking were taking place.

```
<?php

$quotes[] = "I think there is a world market for maybe five computers.
Thomas Watson.";
$quotes[] = "The answer to life's problems aren't at the bottom of a
bottle, they're on TV! Homer Simpson.";
$quotes[] = "I love California, I practically grew up in Phoenix.
Dan Quayle.";
$quotes[] = "640K ought to be enough for anybody. Bill Gates.";
$quotes[] = "Some people are afraid of heights. Not me, I'm afraid of
widths. Steven Wright";
$quotes[] = "Start every day off with a smile and get it over with.
W. C. Fields ";
$quotes[] = "I am not a vegetarian because I love animals; I am a
vegetarian because I hate plants. A. Whitney Brown";
$quotes[] = "A conclusion is simply the place where someone got tired
of thinking. Arthur Block";
$quotes[] = "A rich man's joke is always funny. Proverb.";

//-----
// It all starts here.
// Get requested quote number parameter passed in the url
//-----
if (isset($_GET))
{
    $quote_num = $_GET["number"];
}

// Create a delay to make the problem of no-threads more obvious
sleep(5);

// User entered a quote number
if ($quote_num > 0)
{
    // Decrement the quote requested because our array is zero based
    echo $quotes[$quote_num - 1];
}
else // User did NOT enter a quote number, return a random quote
{
    // Initialize random number generator
    srand((double)microtime() * 1000000);

    // Get a random value that is between 0 and the length of our array
    (minus 1) $rand_num = rand(0, (count($quotes) - 1));

    // Echo the quote from the array
    echo $quotes[$rand_num];
}

?>
```

If the quote number is greater than zero, you know the user passed in a value and

you can access the proper array entry to get the quote. Otherwise, the user did not specify a value in the text box and a random quote from the available list is returned. You do this by generating a random number between zero and the number of entries in the array, minus one. You must decrement by one because array values always begin with 0 (zero) in PHP. If you skip this step, you run the risk of getting a value greater than the number of entries in the array.

In either case, once you retrieve the quote from the array, you echo the string back to the MIDlet. That's all there is to creating a simple quote generator.

Non-threaded MIDlet summary

When writing applications within J2ME, you've now seen the problem with requesting a network connection without running the code in a separate thread. The reason the MIDlet blocks is that the code is running in a thread owned by the system. If the code executing in this thread waits for a connection to complete, the MIDlet waits as well.

For any production MIDlet, this type of coding simply wouldn't be acceptable. The solution is to offload the network processing code into a new thread of execution. As you'll see in the upcoming panels, this approach allows the network processing to take place in the background and the MIDlet user interface to remain responsive.

Now, let's move on to the next section to talk more about threading within J2ME.

Section 3. Threads

Overview

Threads allow more than one path of execution through a program. One example of a multithreaded program we are all familiar with is a Web browser. For instance, while a page is downloading, you can still move about the browser, open menu items, type in new Web addresses, and so on. With a single threaded application, such scenarios would not be possible. Relating back to the previous example (because there is not a thread created specifically for the network communication), the application can essentially perform only one operation: the network task. All other aspects of the program are put on hold, so to speak.

After a thread is created (which I'll discuss in the following panels), you start it by calling `start()`. Execution of the thread begins with a call to the `run()` method. You can

temporarily pause a thread by calling `sleep()`, which causes the thread execution to stop for a specified number of milliseconds. Unlike J2SE, there is no method call available to stop a thread. You'll see how to get around this when you develop the multithreaded MIDlet later in the tutorial.

Let's move on to see the two ways to create threads in J2ME.

Threading Techniques

As in J2SE, there are two approaches for working with threads. One is to directly subclass `Thread`, and the second is to implement the `Runnable` interface. I'll now walk through each of these options, and you'll see what makes one option better than the other.

A good place to start is by subclassing `Thread` directly. Let's see how this is done.

Subclass Thread

The first approach to working with threads is to extend the `Thread` class, as shown below.

```
class NetworkThread extends Thread
{
    ...
    public void run()
    {
        // Thread code goes here
        ...
    }
    ...
}
```

Creating an instance of the class and starting the thread is straightforward.

```
NetworkThread thread1 = new NetworkThread();
thread1.start();
...
```

Remember, invoking `start()` results in the `run()` method being called.

Implement Runnable interface

The second approach for creating threads is to implement the `Runnable` interface. The code example below shows how this might look.

```
public class AppletTest extends Applet implements Runnable
```

```
{
  ...
  public void run()
  {
    // Thread code goes here
    ...
  }
  ...
}
```

To start the AppletTest thread, you create a Thread object as shown below.

```
AppletTest someclass = new AppletTest();
...
Thread thread2 = new Thread(AppletTest);
thread2.start();
...
```

Although this is very similar to the previous example, a few minor adjustments are necessary to the syntax because you are not directly subclassing Thread.

Comparing two thread approaches

So what are the reasons for choosing one approach over another in working with threads? It is not really as much about choice as it is about which option is available given the classes you are working with. Let's refer back to the previous panel, where I implemented the Runnable interface. The example code from the panel is:

```
public class AppletTest extends Applet implements Runnable
{
  ...
  public void run()
  {
    // Thread code goes here
    ...
  }
  ...
}
```

AppletTest extends the Applet class. In the Java language, you cannot extend more than one class. To work around this limitation, and in order to allow support for a thread in AppletTest, the only option is to implement the Runnable interface. Determining whether to extend Thread or to implement the Runnable interface is not really a choice you need to make, it depends more on the current class definitions you have in place.

J2SE versus J2ME threads

As with many other features in J2SE, J2ME provides only a subset of functionality in an effort to reduce the overall size and increase the performance for constrained

mobile devices. When working with threads, I need to point out a few of the differences for those who have worked with threads previously in J2SE.

- **No thread groups.** Threads are processed on an object by object basis. You cannot start or stop a group of threads through one method call in J2ME.
- **No daemon threads.** Daemon threads are threads that essentially run to provide a benefit or service to other threads. For example, in the HotJava browser there is a daemon thread that reads images for any other object or thread that might need an image.
- **No stop() method.** A thread in J2ME lives until the run() method is exited. I'll show an example of how this is accomplished in the multithreaded network MIDlet.

Threads summary

In this last section you got a brief overview of threading in Java. You learned about the two ways of creating threads -- by subclassing Thread directly or by implementing the `Runnable` interface. I concluded by comparing the differences between the thread support in J2SE and J2ME.

Section 4. Multithreaded MIDlet

Overview

It can be put off no longer. Let's write the MIDlet I have been referring to since you began this tutorial -- a multithreaded J2ME application. The MIDlet downloads images over a network connection, and prompts the user for a URL to the image using a `TextBox` component. Once the user begins the download you will get the URL from the textbox, display a message in an `Alert` component indicating the download has begun, and finally, start a thread to begin the download.

The alert dialog box is of type non-modal, meaning it is displayed for a specific amount of time; in our case, three seconds. As soon as the dialog is no longer active, the user is returned to the textbox component. As the image is downloaded, the user can continue working with the application. For example, the URL to a new image could be entered in the textbox.

Once the download is complete, the MIDlet notifies the user by showing a new alert, indicating success or failure of the download. At this point, the alert is modal, which implies that it requires confirmation from the user before the alert is removed from the display. As the last step, once the alert is removed, the downloaded image is shown on the device display.

Let's walk through a few screenshots to further clarify the process just described.

Available images

The MIDlet begins by prompting for a URL to an image. There are several images that I have placed on the www.CoreJ2ME.com Web site. Here are the URLs and a picture of each associated image:

Figure 10. <http://www.corej2me.com/ibm/lighthouse.png>



Figure 11. <http://www.corej2me.com/ibm/car.png>



Figure 12. <http://www.corej2me.com/ibm/pond.png>



Screenshot -- get URL

The first task of the MIDlet is to prompt the user for the URL to an image. See Figure 13. Notice there are just two commands for event processing: Exit to quit the MIDlet, and View to download and view the image.

Figure 13. URL textbox



Screenshot - start download #1

After the user has opted to begin the download, I display an alert showing that the

download has begun. The name of the image is included in the alert message. As you'll see momentarily, displaying the name of the image is important, as you will attempt to create several separate threads, each downloading a different image. Without the image name displayed, it would get rather messy trying to determine which message applies to which download request.

Figure 14. Starting download #1 alert



Screenshot - downloading #1

After the alert has been displayed indicating the download has begun, the user is returned to the main interface. As shown in Figure 15, the download works in a separate thread, assuring the user the interface is still responsive, and allowing you to change the textbox to point to a new URL.

Figure 15. Edit the textbox URL



If you type quickly, you can enter a new URL before the first download completes. See Figure 16.

Figure 16. URL to car image

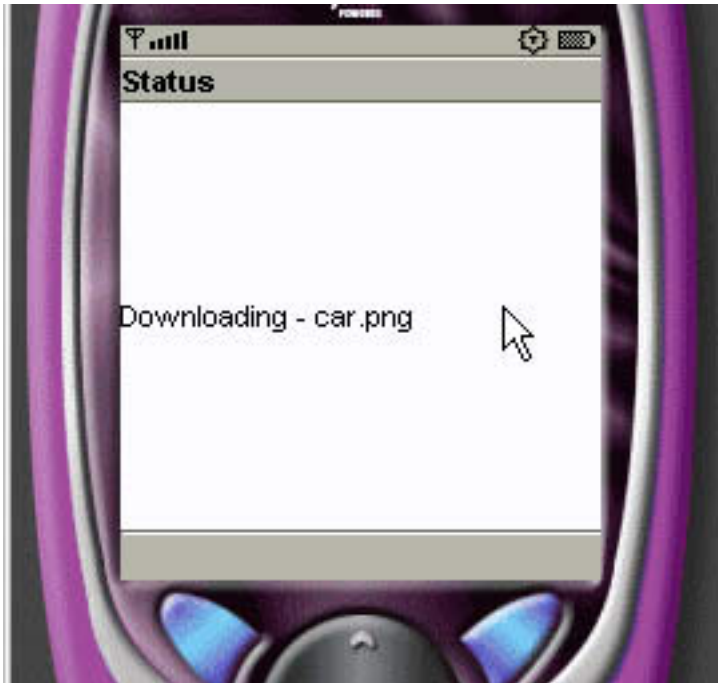


Select **View** to start another thread, initiating the download of a second image.

Screenshot - start download #2

At this point there are two threads running -- one for each of the downloads you have started. Figure 17 shows the alert message indicating that the second download is in process.

Figure 17. Starting download #2 alert



Once again, after initiating the download you are returned to the main user interface. Here, you can patiently wait for the downloads to finish, or enter the address of an additional URL. Figure 18 shows the textbox component and an updated URL to a third image.

Figure 18. Main user interface



Screenshot - download #1 complete

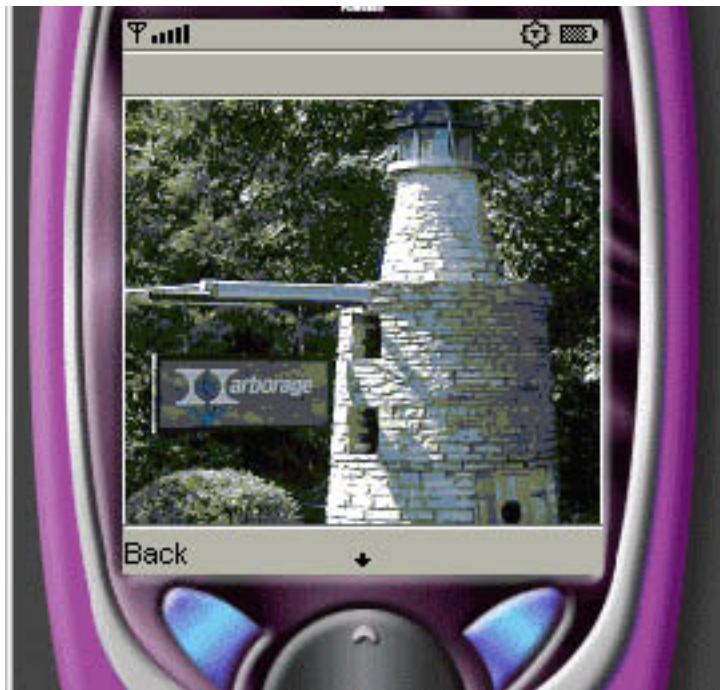
When the first download completes, you inform the user, as shown in Figure 19, through a message in an alert.

Figure 19. Download #1 complete



It should now be apparent why you display the file name inside the alert. Without the file name, and with multiple downloads in progress, you would quickly lose track of which messages apply to which download. Once the message is acknowledged, you display the image on the device. See Figure 20.

Figure 20. Download #1 image



Screenshot - download #2 complete

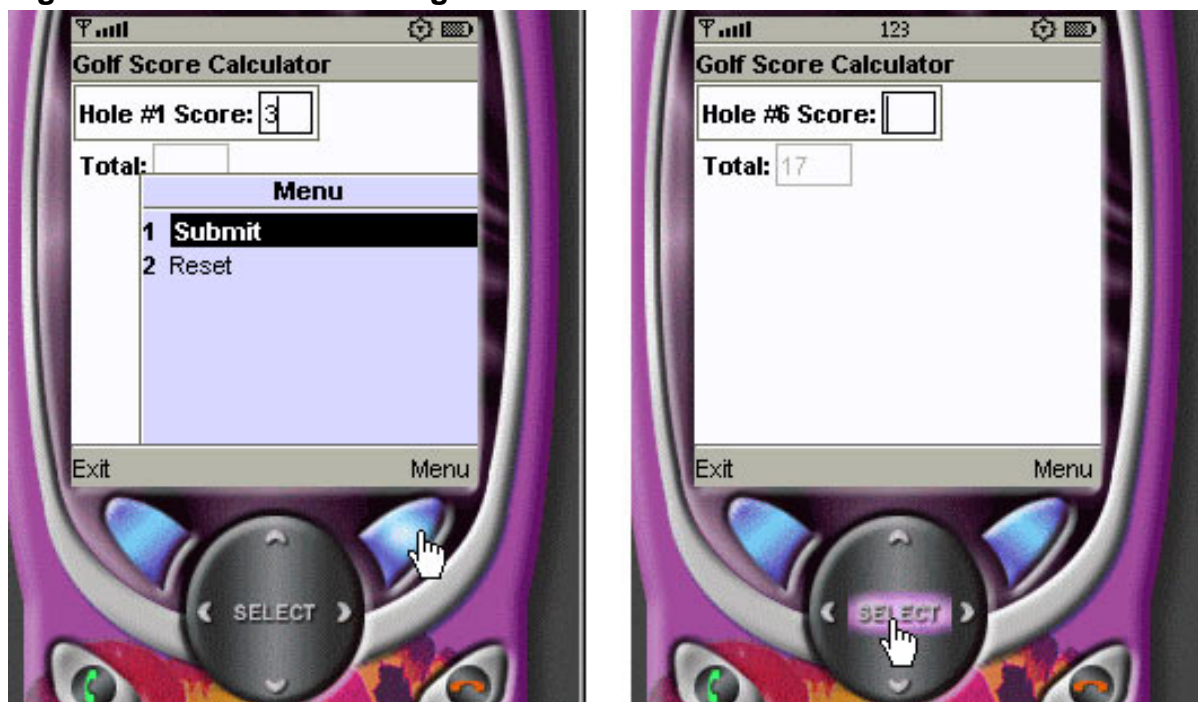
As before, when the download of the second request is complete, a message is displayed as shown in Figure 21.

Figure 21. Download #2 complete



Click **View** to see the output of the second download request.

Figure 22. Download #2 image



You can now see the benefit of working with multiple threads. Not only is the user interface responsive during network activity, you can take advantage of this time to begin downloading additional images.

Write the Java code

With an understanding of the workings of the MIDlet, let's write the Java code, and compile and run the program within the Wireless Toolkit.

Create a new project in the WTK with the name ViewImage. Copy and paste the Java source code below into a text editor and save it in the WTK \apps\ViewImage\src directory with the file name ViewImage.java.

```

/*-----
 * ViewImage.java
 *
 * Download and view png files. Downloading is
 * done in the background with a separate thread
 *-----*/
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import java.io.*;

/*-----
 * Constructor
 *-----*/
public class ViewImage extends MIDlet implements CommandListener
{
    private Display display;      // Main display object
    private TextBox tbMain;      // Textbox to prompt for URL
    private Alert alStatus;      // Dialog box for download status
    private Form fmViewImg;      // Form to display images
    private Command cmExit;      // Command to exit
    private Command cmView;      // Command to initiate image download
    private Command cmBack;      // Return to main screen
    private String imageName = null; // Name of image to download
    Image im = null;             // Downloaded image
    private static final int ALERT_DISPLAY_TIME = 3000;

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

        // Create the Main textbox with a maximum of 75 characters
        tbMain = new TextBox("Enter url",
            "http://www.corej2me.com/ibm/lighthouse.png", 75,
            TextField.ANY);

        // Create commands and add to textbox
        cmExit = new Command("Exit", Command.EXIT, 1);
        cmView = new Command("View", Command.SCREEN, 2);
        tbMain.addCommand(cmExit);
        tbMain.addCommand(cmView );

        // Set up a listener for textbox
        tbMain.setCommandListener(this);

        // Create the form that will hold the png image
        fmViewImg = new Form("");

        // Create commands and add to form
        cmBack = new Command("Back", Command.BACK, 1);
        fmViewImg.addCommand(cmBack);

        // Set up a listener for form
        fmViewImg.setCommandListener(this);
    }
}

```

```

}

public void startApp()
{
    // Display the textbox that prompts for URL
    display.setCurrent(tbMain);
}

public void pauseApp()
{ }

public void destroyApp(boolean unconditional)
{ }

/*-----
 * Process events
 *-----*/
public void commandAction(Command c, Displayable s)
{
    // If the Command button pressed was "Exit"
    if (c == cmExit)
    {
        destroyApp(false);
        notifyDestroyed();
    }
    else if (c == cmView)
    {
        // Save the name of the image to download so we can
        // display it in the alert (dialog box).
        // This will be the entry after the last '/' in the url
        imageName =
            tbMain.getString().substring(tbMain.getString().lastIndexOf('/')
            + 1);

        // Show alert indicating we are starting a download.
        // This alert is NOT modal, it appears for
        // approximately 3 seconds (see ALERT_DISPLAY_TIME)
        showAlert("Downloading - " + imageName, false, tbMain);

        // Create an instance of the class that will
        // download the file in a separate thread
        Download dl = new Download(tbMain.getString(), this, imageName);

        // Start the thread/download
        dl.start();
    }
    else if (c == cmBack)
    {
        display.setCurrent(tbMain);
    }
}

/*-----
 * Called by the thread after attempting to download
 * an image. If the parameter is 'true' the download
 * was successful, and the image is shown on a form.
 * If parameter is 'false' the download failed, and
 * the user is returned to the textbox.
 *
 * In either case, show an alert indicating the
 * the result of the download.
 *-----*/
public void showImage(boolean flag, String imageName)
{
    // Download failed
    if (flag == false)
    {
        // Alert followed by the main textbox
        showAlert("Download Failure", true, tbMain);
    }
}

```

```

    }
    else // Successful download...
    {
        ImageItem ii = new ImageItem(null, im, ImageItem.LAYOUT_CENTER,
            null);

        // If there is already an image, set (replace) it
        if (fmViewImg.size() != 0)
            fmViewImg.set(0, ii);
        else // Append the image to the empty form
            fmViewImg.append(ii);

        // Alert followed by the form holding the image
        showAlert("Download Successful - " + imageName, true, fmViewImg);
    }
}

/*-----
 * Show an alert with the parameters determining
 * the type (modal or not) and the displayable to
 * show after the alert is dismissed
 *-----*/
public void showAlert(String msg, boolean modal, Displayable
    displayable)
{
    // Create alert, add text, associate a sound
    alStatus = new Alert("Status", msg, null, AlertType.INFO);

    // Set the alert type
    if (modal)
        alStatus.setTimeout(Alert.FOREVER);
    else
        alStatus.setTimeout(ALERT_DISPLAY_TIME);

    // Show the alert, followed by the displayable
    display.setCurrent(alStatus, displayable);
}
}

/*-----
 * Class - Download
 *
 * Download an image file in a separate thread
 *-----*/
class Download implements Runnable
{
    private String url;
    private ViewImage MIDlet;
    private String imageName = null;
    private boolean downloadSuccess = false;

    public Download(String url, ViewImage MIDlet, String imageName)
    {
        this.url = url;
        this.MIDlet = MIDlet;
        this.imageName = imageName;
    }

    /*-----
     * Download the image
     *-----*/
    public void run()
    {
        try
        {
            getImage(url);
        }
        catch (Exception e)
        {

```

```

        System.err.println("Msg: " + e.toString());
    }
}

/*-----
 * Create and start the new thread
 *-----*/
public void start()
{
    Thread thread = new Thread(this);
    try
    {
        thread.start();
    }
    catch (Exception e)
    {
    }
}

/*-----
 * Open connection and download png into a byte array.
 *-----*/
private void getImage(String url) throws IOException
{
    ContentConnection connection = (ContentConnection)
        Connector.open(url);
    DataInputStream iStrm = connection.openDataInputStream();
    ByteArrayOutputStream bStrm = null;
    Image im = null;

    try
    {
        // ContentConnection includes a length method
        byte imageData[];
        int length = (int) connection.getLength();
        if (length != -1)
        {
            imageData = new byte[length];

            // Read the png into an array
            iStrm.readFully(imageData);
        }
        else // Length not available...
        {
            bStrm = new ByteArrayOutputStream();

            int ch;
            while ((ch = iStrm.read()) != -1)
                bStrm.write(ch);

            imageData = bStrm.toByteArray();
        }

        // Create the image from the byte array
        im = Image.createImage(imageData, 0, imageData.length);
    }
    finally
    {
        // Clean up
        if (connection != null)
            connection.close();
        if (iStrm != null)
            iStrm.close();
        if (bStrm != null)
            bStrm.close();
    }

    // Return to the caller the status of the download
    if (im == null)

```

```
MIDlet.showImage(false, null);
else
{
    MIDlet.im = im;
    MIDlet.showImage(true, imageName);
}
}
```

To understand what's taking place in this MIDlet, I'll revisit the code step by step in the upcoming panels.

Compile, preverify, and run

Within the WTK:

1. Select **Build** to compile, preverify, and package the ViewImage MIDlet.
2. Start the emulator by selecting **Run** in the WTK.
3. From within the emulator, select **Launch** to start the MIDlet. See Figure 23.

Figure 23. Launch



The main user interface is shown in Figure 24, which includes the textbox that prompts for the URL of the desired image to be downloaded.

Figure 24. MIDlet

Code review: MIDlet constructor

I begin the code review with the MIDlet constructor. The TextBox component is initialized to a default URL, and I then specify a maximum of 75 input characters, and allow any type of character to be input (TextField.ANY). This MIDlet has two commands for the processing of events: cmExit to exit the application and cmView to start the download of an image. Both commands are created and appended to the textbox component. The final step associated with the textbox is to initialize a listener to listen for events.

```
public ViewImage()
{
    display = Display.getDisplay(this);

    // Create the Main textbox with a maximum of 75 characters
    tbMain = new TextBox("Enter url",
        "http://www.corej2me.com/ibm/lighthouse.png", 75,
        TextField.ANY);

    // Create commands and add to textbox
    cmExit = new Command("Exit", Command.EXIT, 1);
    cmView = new Command("View", Command.SCREEN, 2);
    tbMain.addCommand(cmExit);
    tbMain.addCommand(cmView );

    // Set up a listener for textbox
    tbMain.setCommandListener(this);

    // Create the form that will hold the png image
    fmViewImg = new Form("");
}
```

```
// Create commands and add to form
cmBack = new Command("Back", Command.BACK, 1);
fmViewImg.addCommand(cmBack);

// Set up a listener for form
fmViewImg.setCommandListener(this);
}
```

Also within the constructor is the code for allocating a Form, which displays the downloaded image. When the form is displayed, you need a way to return to the main user interface (the textbox). I accomplish this with an additional command, `cmBack`. To wrap up this code block, the command `cmBack` is added to the form and a listener is created to manage events on the form.

Code review: event processing

There are three events you are concerned with capturing. The first is a request to exit the MIDlet. The code for processing this option is the same as always: Call `destroyApp()` to clean up any resources used (none in this case), and notify the Application Management Software that the MIDlet can be safely shutdown.

The second option is for the user to choose the `cmView` command to download an image. You begin by extracting the name of the image file from the URL, saving the string into the variable `imageName`. With the file name in hand, you call the `showAlert()` method to notify the user that you are beginning the download, indicating the specific file you are after. The final steps are to create an instance of the class that manages the thread and to begin the download by starting the thread.

```
public void commandAction(Command c, Displayable s)
{
    // If the Command button pressed was "Exit"
    if (c == cmExit)
    {
        destroyApp(false);
        notifyDestroyed();
    }
    else if (c == cmView)
    {
        // Save the name of the image to download so we can
        // display it in the alert (dialog box).
        // This will be the entry after the last '/' in the url
        imageName =
        tbMain.getString().substring(tbMain.getString().lastIndexOf('/')
        + 1);

        // Show alert indicating we are starting a download.
        // This alert is NOT modal, it appears for
        // approximately 3 seconds (see ALERT_DISPLAY_TIME)
        showAlert("Downloading - " + imageName, false, tbMain);

        // Create an instance of the class that will
        // download the file in a separate thread
        Download dl = new Download(tbMain.getString(), this, imageName);

        // Start the thread/download
        dl.start();
    }
}
```

```
    }  
    else if (c == cmBack)  
    {  
        display.setCurrent(tbMain);  
    }  
}
```

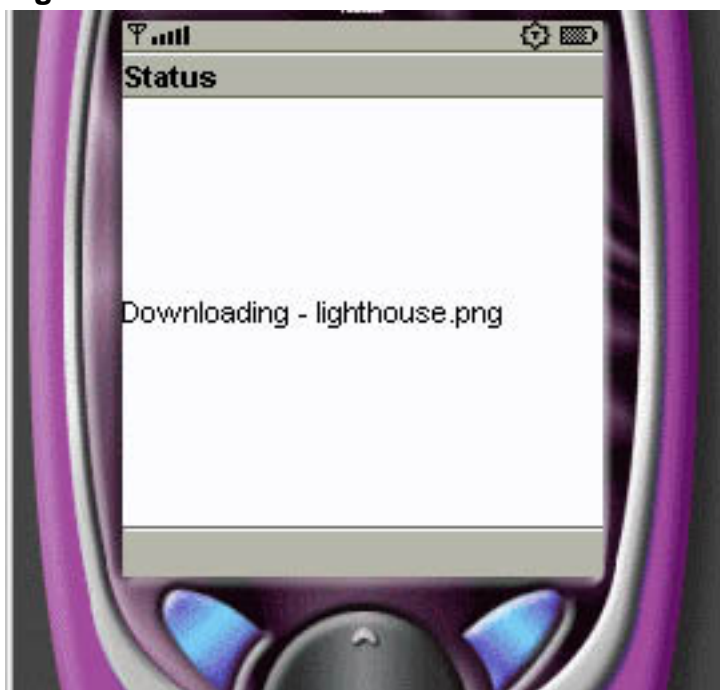
The last command you capture is `cmBack`, which is displayed on the form that shows the downloaded image. This command returns you to the main user interface, or the textbox where an image URL is specified.

Code review: alert overview

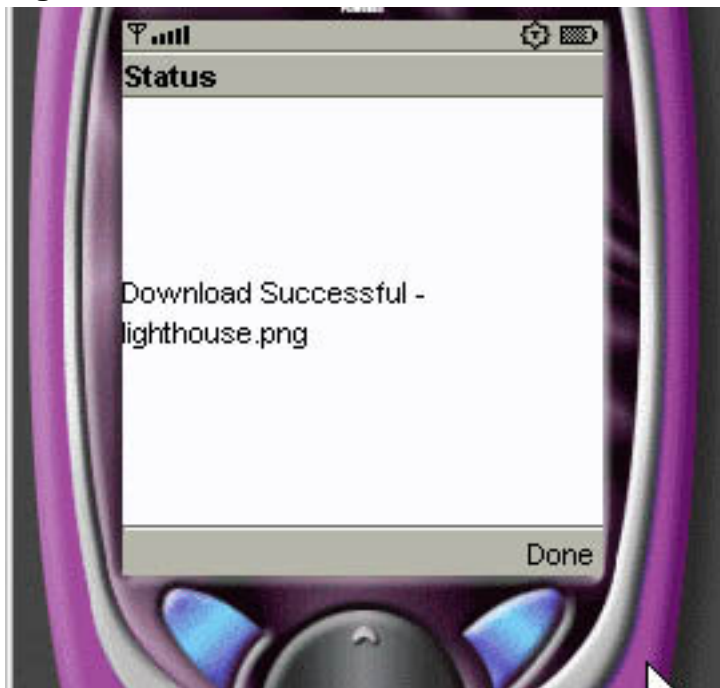
If you recall, there are alerts displayed at two different points in this MIDlet. Once when you begin downloading and again when the download is complete. The code for managing both alerts appears in one method, `showAlert()`. Depending on the parameters sent to the method, you can display either a modal alert (requiring user acknowledgement) or a non-modal (timed) alert.

When starting a download, I display a non-modal alert for three seconds. See Figure 24.

Figure 24. Non-modal alert



After the download is complete, I show a modal alert, requiring the user to acknowledge the dialog.

Figure 25. Modal alert

Note: You can identify the difference between the modal and non-modal alert if you look closely at the figures. A modal alert requires user acknowledgement, therefore the alert has a command associated with it; in this case a command titled Done is shown. A non-modal alert has no command because it appears on the display for a predetermined number of seconds, leaving the user with no means to dismiss the dialog.

The following panel discusses the code inside the `showAlert()` method.

Code review: `showAlert()` method

The `showAlert()` method accepts three parameters, the first being the message to display. The second parameter is a boolean indicating whether the alert is to be modal or non-modal.

```
public void showAlert(String msg, boolean modal, Displayable
    displayable)
{
    // Create alert, add text, associate a sound
    alStatus = new Alert("Status", msg, null, AlertType.INFO);

    // Set the alert type
    if (modal)
        alStatus.setTimeout(Alert.FOREVER);
    else
        alStatus.setTimeout(ALERT_DISPLAY_TIME);
}
```

```

    // Show the alert, followed by the displayable
    display.setCurrent(alStatus, displayable);
}

```

The last parameter to this method is vital to the flow of the MIDlet. `displayable` represents which displayable object is shown, after the alert is dismissed.

For example, when this method is called after initiation of a download, I display a message indicating that the download has begun, which is followed by a return to the main user interface. In this case, the textbox component is the displayable sent into this method. The second instance is when I call this method after a download is complete. This method displays a message indicating success or failure of the download, followed by the form that holds the image. For this scenario, the form is the displayable sent into this method.

Code review: download class

The `Download` class houses the code for the thread. Below is the basic shell of the class, with comments embedded to indicate the flow.

```

class Download implements Runnable
{
    private String url;
    private ViewImage MIDlet;
    private String imageName = null;
    private boolean downloadSuccess = false;

    /*-----
    * Constructor
    *-----*/
    public Download(String url, ViewImage MIDlet, String imageName)
    {
        // Initialize the class
    }

    /*-----
    * Download the image
    *-----*/
    public void run()
    {
        // Where the thread code resides.
        // Here is where we call getImage();
    }

    /*-----
    * Create and start the new thread
    *-----*/
    public void start()
    {
        // Create and start a new thread, which results
        // in a call to the run() method
    }

    /*-----
    * Open connection and download png into a byte array.
    *-----*/
    private void getImage(String url) throws IOException
    {

```

```
        // Download the image
    }
}
```

I'll cover each method in more detail in the following panels. However, you can get a pretty good idea of the flow. Once this class is created, I call `start()` to create a new thread. Inside this same method I request the thread to begin processing, which results in a call to `run()`. Inside the `run()` method is where I request the image be downloaded by jumping to the `getImage()` method.

As explained earlier, there is not a way to stop a thread in J2ME. Instead, once the `run()` method is exited, the thread is essentially complete. For this example, what it really means is that once the method called by `run()`, namely `getImage()`, is finished processing, the thread is no longer active. In other words, even though I exit the `run()` method, the code continues to execute (in the background) until the `getImage()` method is finished.

Code review: starting thread

Taking a closer look inside the `Download` class, you see just how easy it is to create and start a thread.

```
public void start()
{
    Thread thread = new Thread(this);
    try
    {
        thread.start();
    }
    catch (Exception e)
    {
    }
}
```

The call to `Thread.start()` is the request to begin processing. The thread begins execution inside the `run()` method, shown in the next panel.

Code review: running thread

The code inside the `run()` method is exceptionally short in this case. It's nothing more than a call to `getImage()`, where a network connection is established and the requested image is downloaded.

```
public void run()
{
    try
    {
        getImage(url);
    }
}
```

```
        catch (Exception e)
        {
            System.err.println("Msg: " + e.toString());
        }
    }
}
```

Notice the thread is not stopped anywhere in the code. Leaving the `run()` method is sufficient to accomplish this same task. When `getImage()` is complete, the thread is finished as well.

Code review: downloading image

In `getImage()`, you establish the network connection, download the requested image, build an image from the downloaded data, and call `showImage()` to complete the process.

After the connection is established, you call `connection.getLength()` to determine whether the remote server sent the length of the image as part of the return data. If so, you can read the image in one fell swoop using `iStrm.readFully()`. If the length is not available, you have to read data from the input stream one character at a time. In either case, the end result is the same, with the image data stored in the `imageData` byte array.

```
private void getImage(String url) throws IOException
{
    ContentConnection connection = (ContentConnection)
    Connector.open(url);
    DataInputStream iStrm = connection.openDataInputStream();
    ByteArrayOutputStream bStrm = null;
    Image im = null;

    try
    {
        // ContentConnection includes a length method
        byte imageData[];
        int length = (int) connection.getLength();
        if (length != -1)
        {
            imageData = new byte[length];

            // Read the png into an array
            iStrm.readFully(imageData);
        }
        else // Length not available...
        {
            bStrm = new ByteArrayOutputStream();

            int ch;
            while ((ch = iStrm.read()) != -1)
                bStrm.write(ch);

            imageData = bStrm.toByteArray();
        }

        // Create the image from the byte array
        im = Image.createImage(imageData, 0, imageData.length);
    }
}
```

```
finally
{
    // Clean up
    if (connection != null)
        connection.close();
    if (iStrm != null)
        iStrm.close();
    if (bStrm != null)
        bStrm.close();
}

// Return to the caller the status of the download
if (im == null)
    MIDlet.showImage(false, null);
else
{
    MIDlet.im = im;
    MIDlet.showImage(true, imageName);
}
}
```

After the image data is downloaded, you create an `Image` based on the data retrieved. Before you exit, you close any open connections and call `showImage()` to complete the download process.

Code review: show image

The last steps for completing the code are to display an alert dialog indicating the failure or success of the download and to display the image on the device.

The code inside `showImage()` begins by determining whether the download was successful, based on the flag parameter passed from the `getImage()` method. If `flag` is false, you display the alert indicating download failure. Take note of the three parameters passed to `showAlert("Download Failure", true, tbMain)`. The first is the message to display. Next, you pass in `true` indicating this is a modal dialog box. Finally, `tbMain` is the displayable that is shown when the alert is acknowledged. `tbMain` is the textbox on the main user interface.

```
public void showImage(boolean flag, String imageName)
{
    // Download failed
    if (flag == false)
    {
        // Alert followed by the main textbox
        showAlert("Download Failure", true, tbMain);
    }
    else // Successful download...
    {
        ImageItem ii = new ImageItem(null, im, ImageItem.LAYOUT_CENTER,
            null);

        // If there is already an image, set (replace) it
        if (fmViewImg.size() != 0)
            fmViewImg.set(0, ii);
        else // Append the image to the empty form
            fmViewImg.append(ii);
    }
}
```

```
        // Alert followed by the form holding the image
        showAlert("Download Successful - " + imageName, true, fmViewImg);
    }
}
```

In the case of a successful download, you create an `ImageItem` object that includes the image to display and the preferred layout location (centered on the device). The null parameters specify no label for the image and no alternative text, respectively.

You then check to see if the form that displays the image is of a size not equal to 0 (zero). When greater than 0, you set the `imageitem` equal to the first entry on the form, effectively replacing any previous image. If there is no image on the form, you append the `imageitem` onto the form.

The final step is to display the alert indicating success. Once again, notice the parameters to the `showAlert` method. You pass in the message to display, `true` to request a modal dialog, and the last parameter is the displayable object to show after the alert is dismissed, which is the form you created that holds the downloaded image.

Multithreaded MIDlet summary

In this section you have written a multithreaded MIDlet that downloads images over a network connection. You began by showing screenshots and walking through the application. This was followed by writing the Java code to create the MIDlet. Once the project was up and running, you worked your way through each aspect of the code, focusing on the details for creating multithreaded applications in J2ME.

Section 5. Summary

This tutorial has given you a broad coverage of working with MIDlets that communicate over a network connection. The intention was to focus on the importance of using threads and writing code to demonstrate how to do so.

You started the tutorial by writing a non-threaded MIDlet that would retrieve quotes from a remote server. The reason for developing this application was to show the problems associated with network access from a MIDlet that did not use a separate thread for communication. I demonstrated how the user interface blocked while waiting for a network connection to complete.

In the second section of this tutorial, I introduced threading. This included a

discussion of the two threading techniques available in the Java language: creating a class that subclasses `Thread`, and implementing the `Runnable` interface. In this section, you also studied how J2ME threads differ from those available within J2SE.

The final section was dedicated to creating a multithreaded MIDlet. As in the previous section, I began with a series of screenshots to demonstrate the MIDlet you were about to write. This included showing how the multithreaded MIDlet could download more than one image at a time. I wrote the code to make it happen, and concluded with a comprehensive discussion of each aspect of the program.

Hopefully, you now have a good appreciation of why writing multithreaded applications is so important. You also have a starting point for building more comprehensive MIDlets that can use threading to increase the useability of your applications.

Resources

Learn

- This site provides information on the [Java Development Kit 1.4.2](#)
- Additional articles and resources can be found at [Core J2ME](#).
- You can read about J2ME networking basics in this developerWorks article, "[Networking with J2ME](#)." (developerWorks, September 2002)
- Learn the specifics of working with the [Generic Connection Framework](#) in this developerWorks article (developerWorks, January 2004).
- "[MIDlet development with the Wireless Toolkit](#)" (developerWorks, March 2003) guides you through the basic steps for MIDlet development with J2ME.
- If you are new to threading in the Java language, check out [Essential Java Classes](#), a good introduction to threading.
- The [WebSphere® Everyplace Micro Environment](#) provides an end-to-end solution connecting cellular phones, PDAs, and other pervasive devices to e-business.
- The alphaWorks [Web Services Toolkit for Mobile Devices](#) provides tools and a run time environment for developing applications that use Web services on small mobile devices, gateway devices, and intelligent controllers.

Get products and technologies

- Download the [J2ME Wireless Toolkit](#) for use with this article.

About the author

John Muchow

[John Muchow](#) is a freelance technical writer and the author of [Core J2ME Technology and MIDP](#) (Prentice Hall, Dec. 2001), a best-selling J2ME book that has been translated into Chinese, Korean, and Italian. Visit [Core J2ME](#) for additional source code, articles, and developer resources. You can e-mail John at john@corej2me.com for information about technical writing projects.