

Downloading images in J2ME

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.

1. Before you start	2
2. Networking basics in J2ME	4
3. MIDlet 1 -- Download and view image	6
4. MIDlet 2 -- Download with thread	21
5. Summary and resources	38
6. Feedback	39

Section 1. Before you start

About this tutorial

This tutorial demonstrates how to download and display images with J2ME (Java 2 Micro Edition) and the Mobile Information Device Profile (MIDP).

I'll walk through creating two J2ME applications, each showing how to open a network connection to a remote server, and download and display an image. The first application is bare-bones, with no frills. I'll build upon this application in the second example to add support for downloading of images in a background thread and providing consistent messages to the user to indicate the current status of the application.

By the end of this tutorial you see just how easy it is to begin writing network-aware applications with J2ME.

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](http://java.sun.com/products/jdk/1.4.1). (<http://java.sun.com/products/jdk/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](http://java.sun.com/products/j2mewtoolkit). (<http://java.sun.com/products/j2mewtoolkit>)

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)

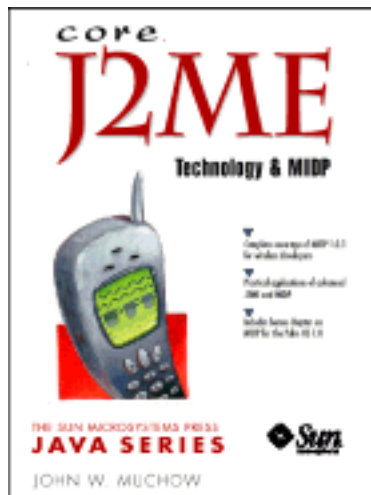
This tutorial builds on an earlier *developerWorks* tutorial "MIDlet Development with the Wireless Toolkit" (see [Resources](#) on page 38), which explains the basics of creating MIDlets with the toolkit. This tutorial is an excellent starting point if you are new to the Wireless Toolkit.

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.

About the author

[John Muchow](mailto:John@CoreJ2ME.com) (mailto:John@CoreJ2ME.com), a freelance technical writer and recruiter, is the author of [Core J2ME Technology and MIDP](#).



Visit [Core J2ME](http://www.CoreJ2ME.com/) (http://www.CoreJ2ME.com/) for additional source code, articles, and developer resources. Send John [e-mail](mailto:john@corej2me.com) (mailto:john@corej2me.com) for additional information about writing projects or technical recruiting.

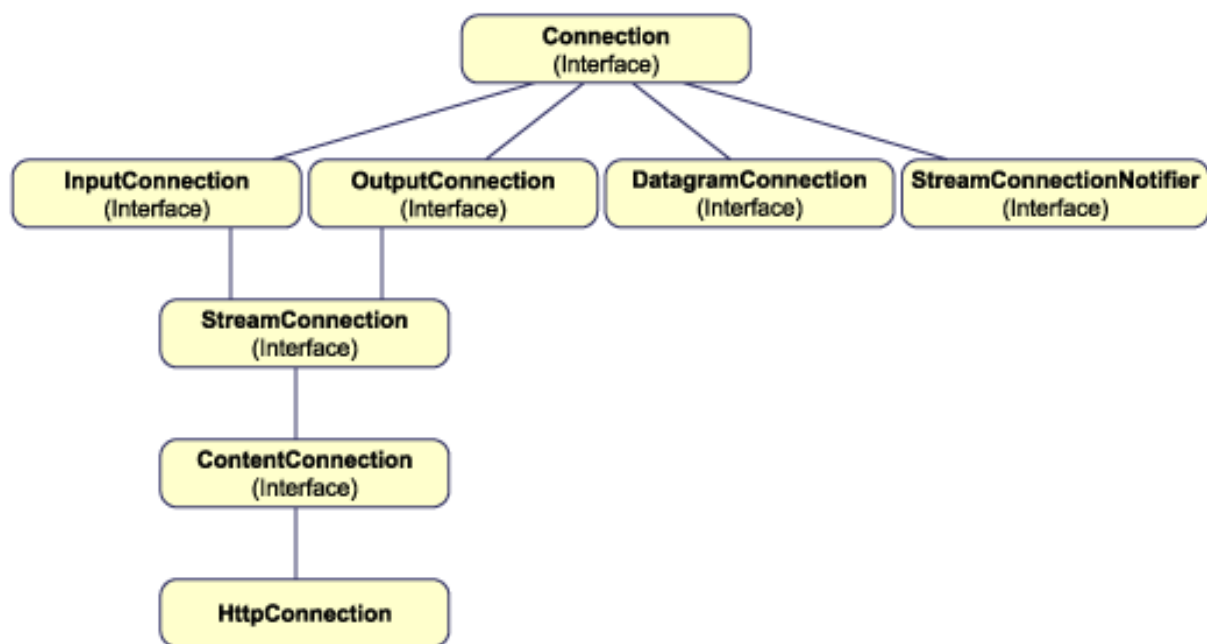
For technical questions or comments about the content of this tutorial, contact the author, John Muchow, at John@CoreJ2ME.com or click Feedback at the top of any panel.

Section 2. Networking basics in J2ME

Generic Connection Framework

With over 100 classes and interfaces in J2SE to support network communication, J2ME sought to provide an extensible framework without all the bulk. The Generic Connection Framework, or simply GCF, provides a subset of classes I/O and networking support.

Figure 1. Generic Connection Framework



Connector class

The idea behind the GCF is to provide one class that can support a connection of any type: file, http, datagram, and so on. The `Connector` class has the following format, and is the primary way to create a network or I/O connection:

```
Connector.Open("protocol:address;parameters");
```

Following are a few examples of how you might open various connections:

1. `Connector.Open("file://resources.txt");`
2. `Connector.Open("http://www.ibm.com/developerworks");`

At run time, `Connector` attempts to locate a class that implements the requested protocol. If a class is found, an object is returned that implements a `Connection` interface.

Methods for creating connections

Using the `Connector` class, there are 7 methods for creating connections:

1. `static Connection open(String name)`
2. `static Connection open(String name, int mode)`
3. `static Connection open(String name, int mode, boolean timeouts)`
4. `static InputStream openInputStream(String name)`
5. `static OutputStream openOutputStream(String name)`
6. `static DataInputStream openDataInputStream(String name)`
7. `static DataOutputStream openDataOutputStream(String name)`

ContentConnection class

`ContentConnection` is a Java interface provided in J2ME that extends `StreamConnection`. The full class declaration follows, including the method available:

```
public interface ContentConnection extends StreamConnection)
    public String getType()
    public String getEncoding()
    public long getLength()
```

Creating a ContentConnection

Below is a short code example using `ContentConnection`.

```
String url = "http://www.corej2me.com"
ContentConnection connection = (ContentConnection) Connector.
InputStream iStrm = connection.openInputStream();

// ContentConnection includes a length method
int length = (int) connection.getLength();
if (length > 0)
{
    byte imageData[] = new byte[length];

    // Read the data into an array
    iStrm.read(imageData);
}
```

Section 3. MIDlet 1 -- Download and view image

Creating an Image object

Before going any further, I must briefly discuss how to create an Image object. There are several methods for creating images; however, for our needs there is just one that I need mention -- creating an image from an array of data:

```
Image createImage(byte[] imageData, int imageOffset, int imageLength)
```

The data comprising the image will be read over a network connection and stored in an array. Once in the array, I'll call the above method to create the image. I'll cover the specifics of using `CreateImage()` in the first MIDlet we write.

View an image

The first MIDlet downloads an image from a Web server and displays the image on the device. The figure below shows the resulting image as shown in the Wireless Toolkit Emulator.

Figure 2. Image downloaded and displayed on device.

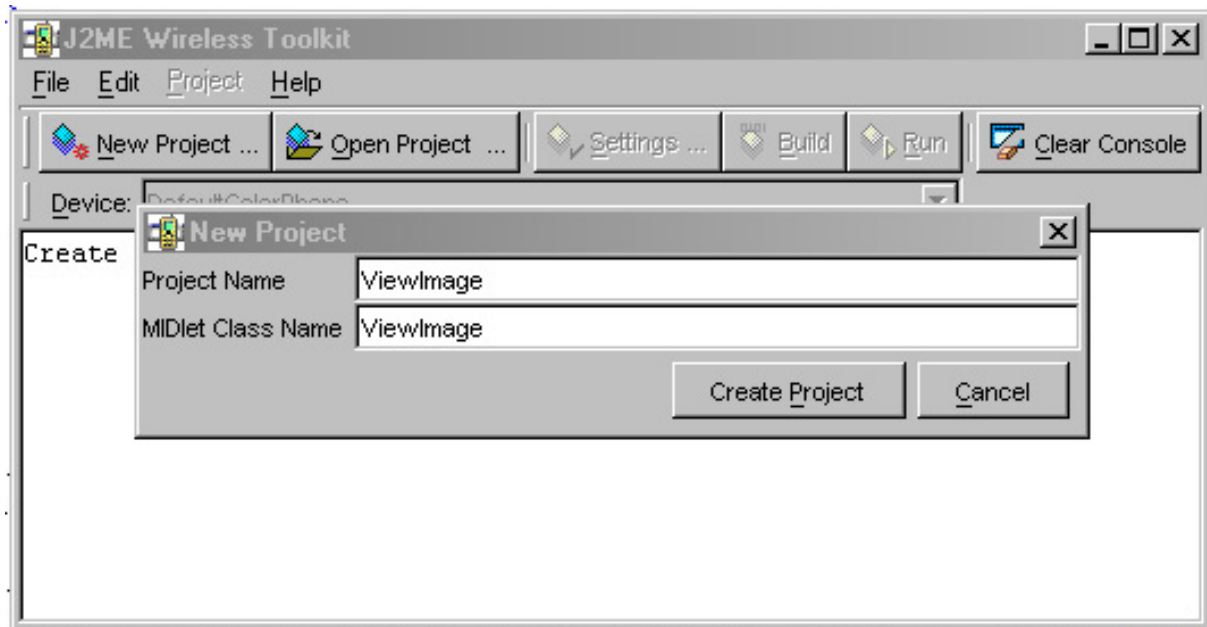


Create the project

Begin by creating a new project in the WTK.

1. Click **New Project**.
2. Enter the Project Name and MIDlet Class Name, as shown in Figure 3.
3. Click **Create Project** to complete this step.

Figure 3. Creating the View Image project

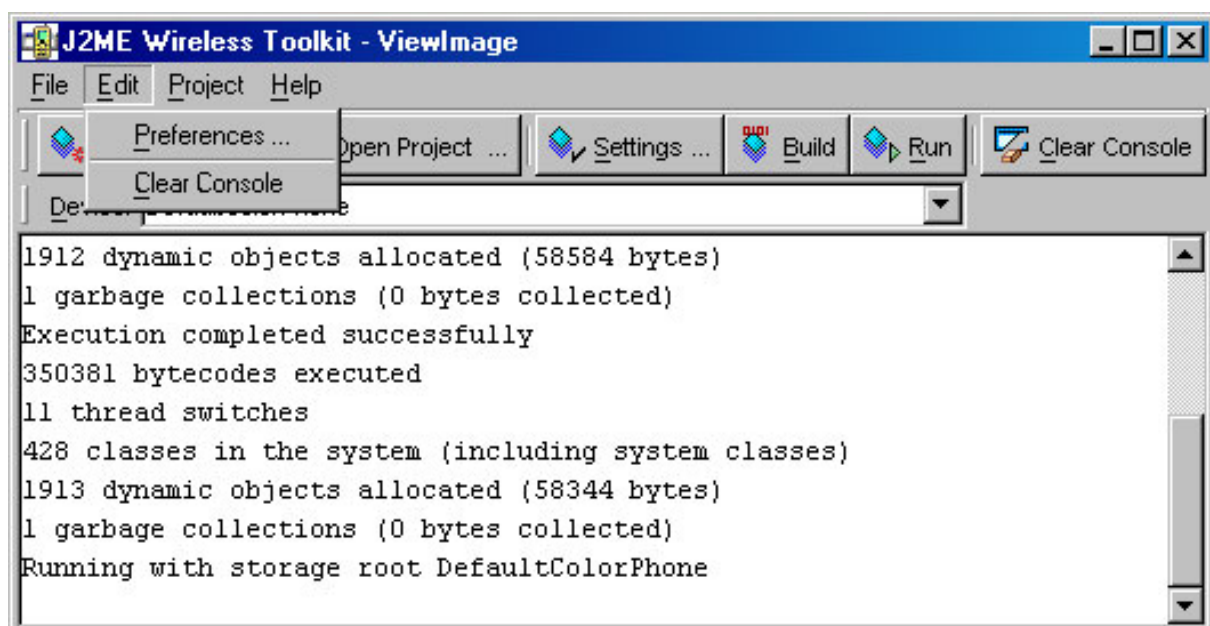


Set project preferences

Within MIDP 2.0, there is enhanced security support. One new feature is the concept of trusted and untrusted code. Only trusted code can access any network connection. Code deemed as untrusted requires user permission for each network connection. For our development, I'll set the permission to trusted.

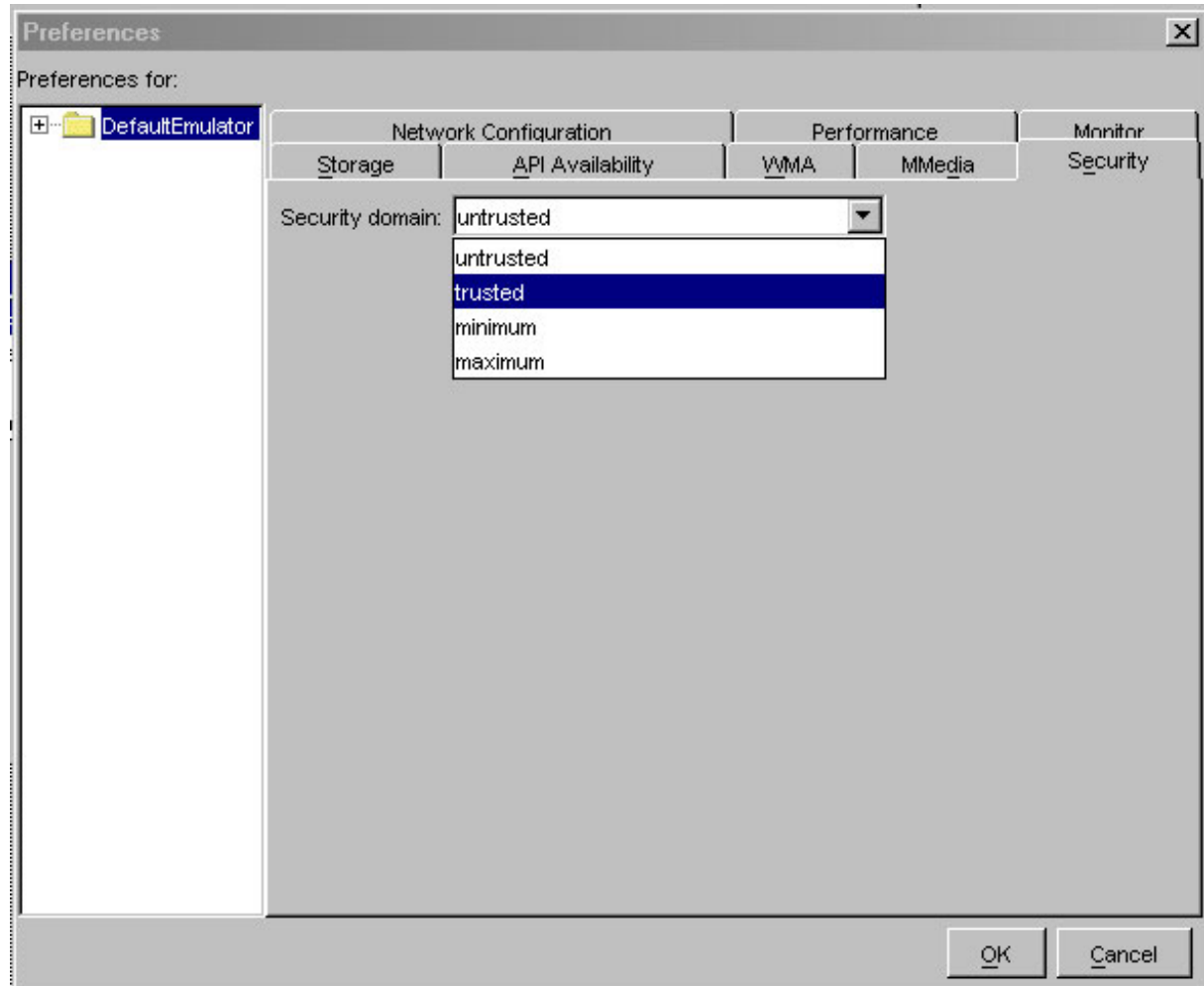
Click **Edit**, and choose **Preferences**.

Figure 4. Setting Preferences



Select **Trusted** from the dropdown menu, which will allow the MIDLET to perform any requested network communication.

Figure 5. Setting Preferences



Write the code

Copy and paste the ViewImage.java code into a text editor.

```
/*-----  
 * ViewImage.java  
 *  
 * Download and view a png file  
 **-----*/  
import javax.microedition.midlet.*;  
import javax.microedition.lcdui.*;  
import javax.microedition.io.*;  
import java.io.*;  
  
public class ViewImage extends MIDlet implements CommandListener  
{  
    private Display display;
```

```

private TextBox tbMain;
private Form fmViewImage;
private Command cmExit;
private Command cmView;
private Command cmBack;

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

    // Create the textbox with a maximum of 50 characters
    tbMain = new TextBox("Enter url", "http://www.corej2me.com/ibm/duke");

    // 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 image
    fmViewImage = new Form("");

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

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

public void startApp()
{
    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)
    {
        try
        {
            // Download image and set as the first (only) item on the form
            Image im;

            if ((im = getImage(tbMain.getString())) != null)
            {

```

```

        // Specify null for the label and alternate text
        ImageItem ii = new ImageItem(null, im, ImageItem.LAYOUT_DEFAULT);

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

    // Display the form with the image
    display.setCurrent(fmViewImage);
}
catch (Exception e)
{
    System.err.println("Msg: " + e.toString());
}
}
else if (c == cmBack) {
    display.setCurrent(tbMain);
}
}

/*-----
* Open connection and download png into a byte array.
*-----*/
private Image 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();
            bStrm.close();
        }

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

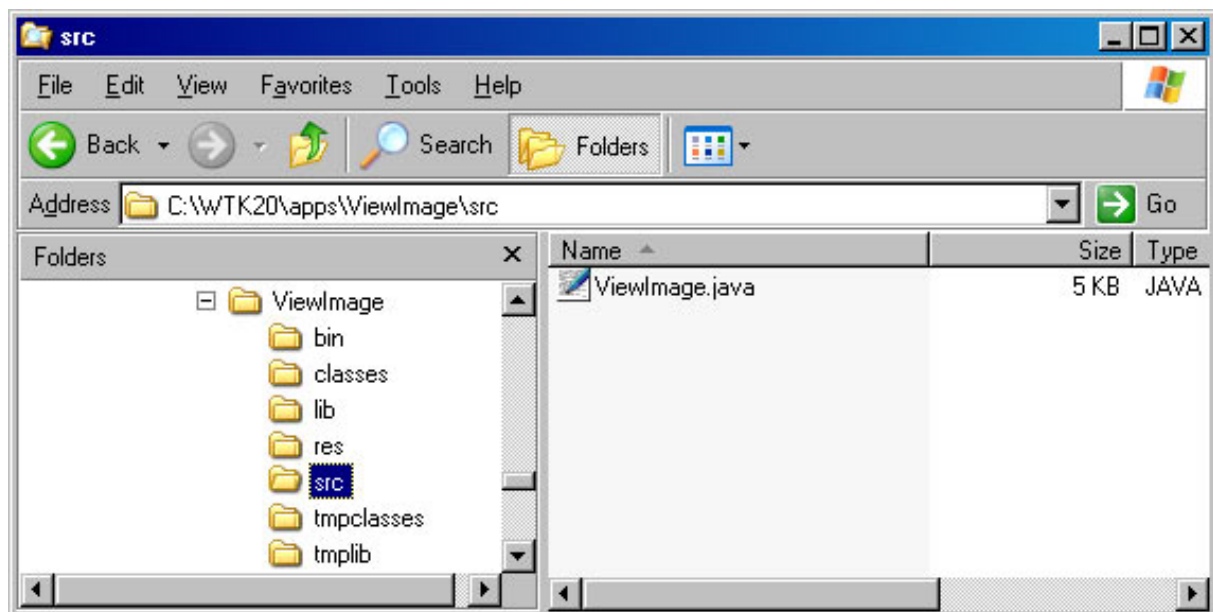
```

```
    // Clean up
    if (iStrm != null)
        iStrm.close();
    if (connection != null)
        connection.close();
    if (bStrm != null)
        bStrm.close();
}
return (im == null ? null : im);
}
```

Save, compile, and preverify

When you create a new project, the WTK builds the proper directory structure for you. In this example, the WTK created the C:\WTK20\apps\ViewImage directory and all the necessary subdirectories. Save your Java source file as ViewImage.java in the src directory, as shown in Figure 6.

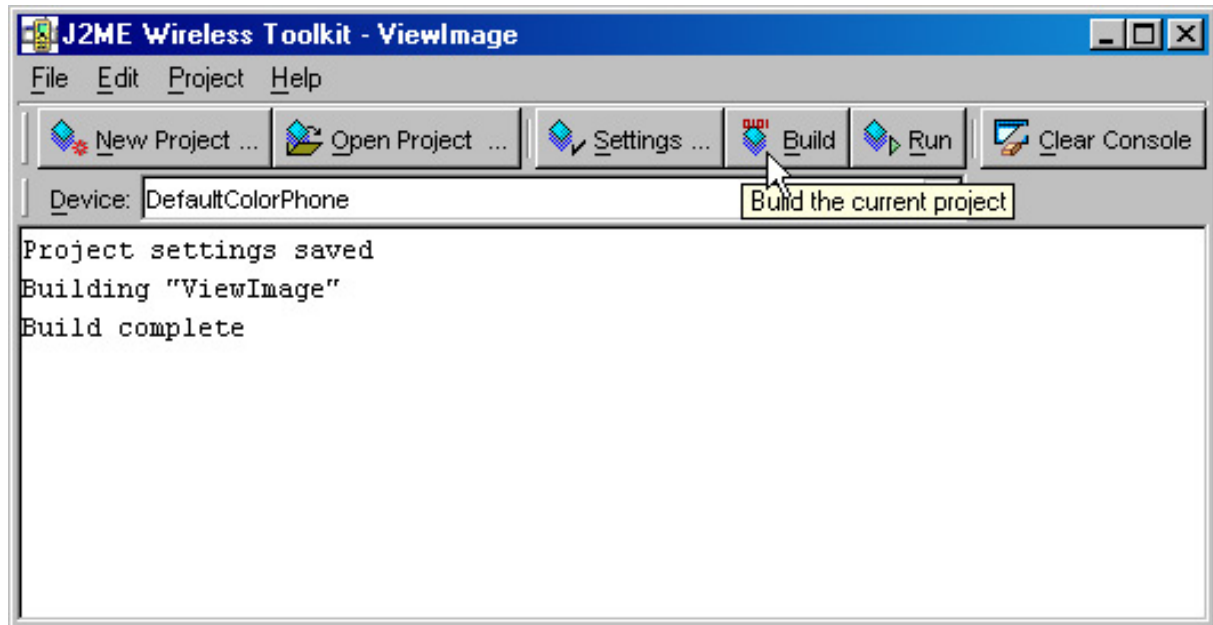
Figure 6. Save the ViewImage code



Note: The drive and WTK directory vary depending on where you installed the toolkit.

Click **Build** to compile, preverify, and package the MIDlet, as shown in Figure 7.

Figure 7. Build the ViewImage project



Then click **Run** to start the Application Manager.

Start the MIDlet

To start the ViewImage MIDlet, click **Launch**, as shown in Figure 8.

Figure 8. Run the ViewImage MIDlet



Downloading image

Once you launch the MIDlet you can specify the URL of the image you would like to download. There is an image available on the URL shown in Figure 9.

Figure 9. Specify URL of image to download.



Figure 10. Image downloaded and displayed on device.



Code review: User interface

Begin by reviewing the class declarations, including variables for working with the user interface as well as event handling

```
public class ViewImage extends MIDlet implements CommandListener
{
    private Display display;
    private TextBox tbMain;
    private Form fmViewImage;
    private Command cmExit;
    private Command cmView;
    private Command cmBack;
}
```

```

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

    // Create the textbox with a maximum of 50 characters
    tbMain = new TextBox("Enter url", "http://www.corej2me.com/ibm/duke");

    // 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 image
    fmViewImage = new Form("");

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

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

...
}

```

There are two primary user interface components: the `TextBox` and `Form`. The `Textbox` is used to prompt the user for the URL where the image is located. The `Form` will display the image once the image has been downloaded.

There are three commands for managing events. `cmView` initiates the image download. When the image is being shown on the `Form`, the command `cmBack` allows the user to return to the `TextBox` where the URL is specified. `cmExit` exits the MIDlet.

Code review: Downloading image

The method `getImage()` is passed the URL that the user entered in the `TextBox`. Create a `URLConnection` to open the network connection. For this example, I have chosen a `DataInputStream` to retrieve the incoming data.

Within the `try/catch` block I attempt to determine the length of the incoming data. If the length is available, I allocate an array to store the image and download the data. If the length is not available, I create a `ByteArrayOutputStream` and read characters one at a time over the connection.

```

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

```

```

{
    ContentConnection connection = (ContentConnection) Connector.open(u

    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();
            bStrm.close();
        }

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

```

Regardless of which means I use to download the data, the image will always be created from the resulting array of downloaded data. I specify the starting point in the array where the image data begins and the number of bytes to store. For this example, the array consists entirely of image data; therefore, I specify the entire array when allocating the image.

Code review: Event handling (Exit)

The first check is to see if the user requested to exit the MIDlet.

```

/*-----
 * Process events
 *-----*/

```

```

public void commandAction(Command c, Displayable s)
{
    // If the Command button pressed was "Exit"
    if (c == cmExit)
    {
        destroyApp(false);
        notifyDestroyed();
    }
    ...
}

```

Code review: Event handling (Download)

The bulk of event handling code revolves around downloading and displaying of images. I begin by calling the `getImage()` method to begin the download. If the image is successfully received, I allocate an `ImageItem`, specifying the image created during the download. The reason for creating an `ImageItem` is that a `Form` cannot display an image directly, it must be wrapped inside an `ImageItem` component.

```

/*-----
 * Process events
 *-----*/
public void commandAction(Command c, Displayable s)
{
    ...

    else if (c == cmView)
    {
        try
        {
            // Download image and set as the first (only) item on the form
            Image im;

            if ((im = getImage(tbMain.getString())) != null)
            {
                // Specify null for the label and alternate text
                ImageItem ii = new ImageItem(null, im, ImageItem.LAYOUT_DEFAULT);

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

            // Display the form with the image
            display.setCurrent(fmViewImage);
        }
        catch (Exception e)
        {
            System.err.println("Msg: " + e.toString());
        }
    }
}

```

If the form size is not 0 (zero), I know there is an existing image on the form, and I

therefore replace it with our new image. If the form is empty, simply append the new image.

The final step is to set the current displayable object to the form, which will display the downloaded image.

Code review: Event handling (Back)

The final option during event processing is shown when the user is viewing the downloaded image. The command `cmBack`, gives the user the option of returning to the `TextBox` where the URL is specified.

```
/*-----  
* Process events  
*-----*/  
public void commandAction(Command c, Displayable s)  
{  
    ...  
    else if (c == cmBack) {  
        display.setCurrent(tbMain);  
    }  
}
```

Section 4. MIDlet 2 -- Download with thread

Overview

The next MIDlet will essentially perform the same action as before. However, there will be two significant improvements. In the previous MIDlet, once the download was initiated the MIDlet essentially deadlocks, waiting for the download to complete. In a production application this would obviously be unacceptable.

The first improvement will be to add a separate thread of execution to make the application more responsive once the download has begun. The second enhancement will provide a means to notify the user of the status of the download.

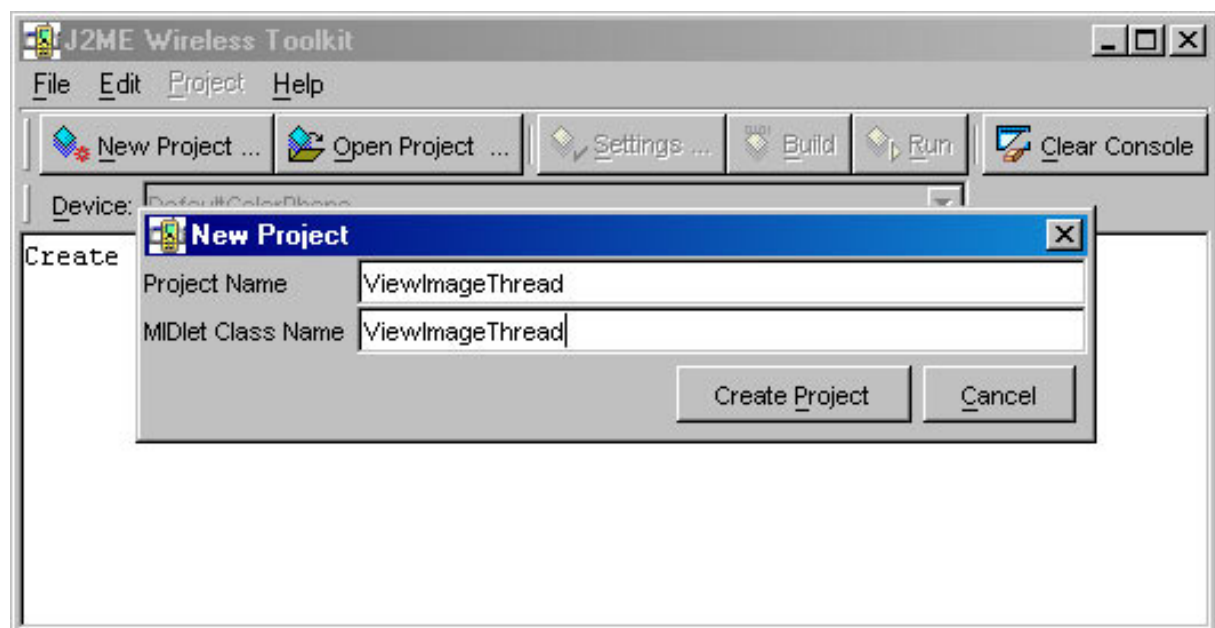
Here is how relaying status information will take place: When the download begins, I'll display an Alert dialog box indicating the download has begun. After a short pause the dialog will be automatically dismissed (no user intervention required). The user will be returned to TextBox displaying the URL. Once the download is complete, the Alert will again be shown, indicating the download is complete.

Create the project

Begin by creating a new project as we did before.

1. Click **New Project**.
2. Enter the Project Name and MIDlet Class Name, as shown in Figure 11.
3. Click **Create Project** to complete this step.

Figure 11. Creating the Download with Thread project



Write the code

Copy and paste the ViewImageThread.java code into a text editor.

```
/*-----  
* ViewImageThread.java  
*  
* Download and view a png file. The download is  
* done in the background with a separate thread  
*-----*/  
import javax.microedition.midlet.*;  
import javax.microedition.lcdui.*;  
import javax.microedition.io.*;  
import java.io.*;  
  
public class ViewImageThread extends MIDlet implements CommandListener  
{  
    private Display display;  
    private TextBox tbMain;  
        private Alert alStatus;  
    private Form fmViewImage;  
    private Command cmExit;  
    private Command cmView;  
    private Command cmBack;  
    private static final int ALERT_DISPLAY_TIME = 3000;  
    Image im = null;  
  
    public ViewImageThread()  
    {  
        display = Display.getDisplay(this);  
  
        // Create the Main textbox with a maximum of 75 characters  
        tbMain = new TextBox("Enter url", "http://www.corej2me.com/ibm/ange  
  
        // 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  
        fmViewImage = new Form("");  
  
        // Create commands and add to form  
        cmBack = new Command("Back", Command.BACK, 1);  
        fmViewImage.addCommand(cmBack);  
  
        // Set up a listener for form  
        fmViewImage.setCommandListener(this);  
    }  
  
    public void startApp()  
    {  
        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)
    {
        // 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", false, tbMain);

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

        // 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)
{
    // 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_DEFAULT,

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

        // Alert followed by the form holding the image
        showAlert("Download Successful", true, fmViewImage);
    }
}
}
```

```

/*-----
 * 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 ViewImageThread MIDlet;
    private boolean downloadSuccess = false;

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

    /*-----
     * 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();
                bStrm.close();
            }

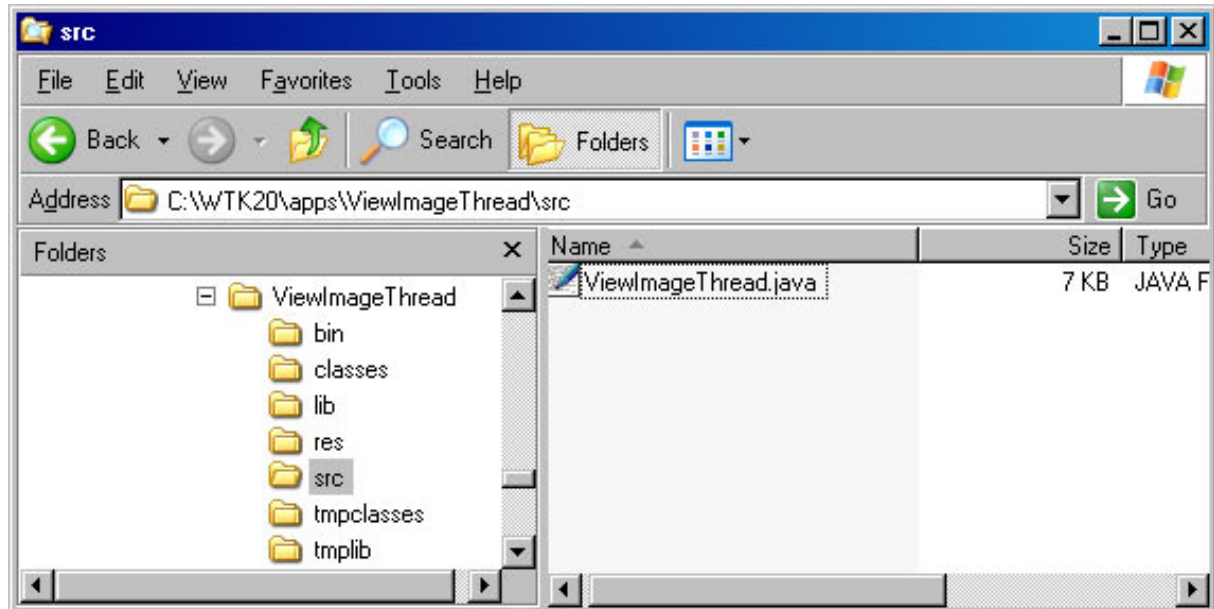
            // Create the image from the byte array
            im = Image.createImage(imageData, 0, imageData.length);
        }
        finally
        {
            // Clean up
            if (connection != null)
                connection.close();
            if (iStrm != null)
                iStrm.close();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

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

Save, compile, and preverify

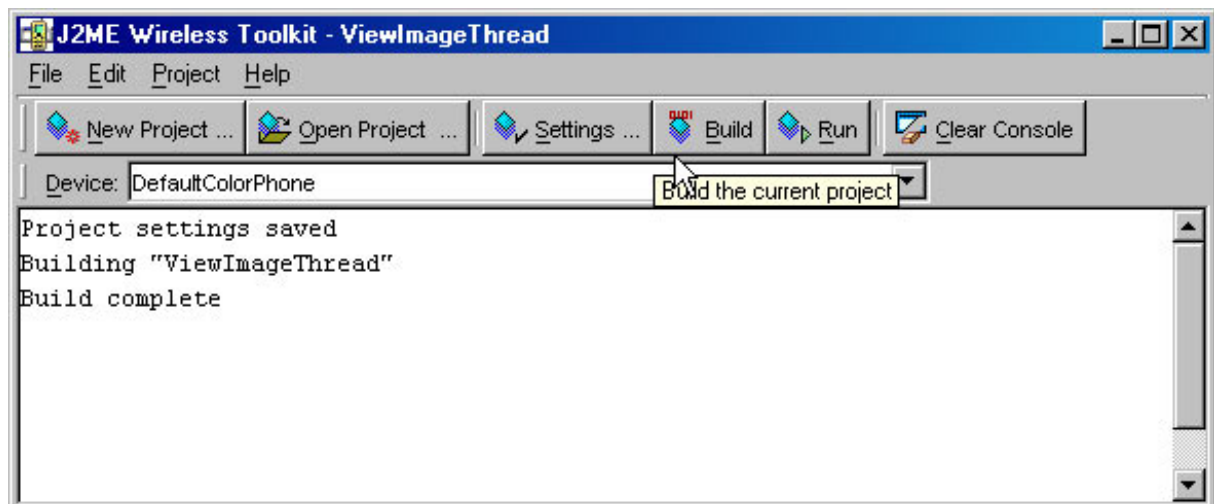
Save your Java source file as `ViewImageThread.java` in the `src` directory, as shown in Figure 12.

Figure 12. Save the ViewImage code



Click **Build** to compile, preverify, and package the MIDlet.

Figure 13. Build the ViewImageThread project



Then click **Run** to start the Application Manager.

Start the MIDlet

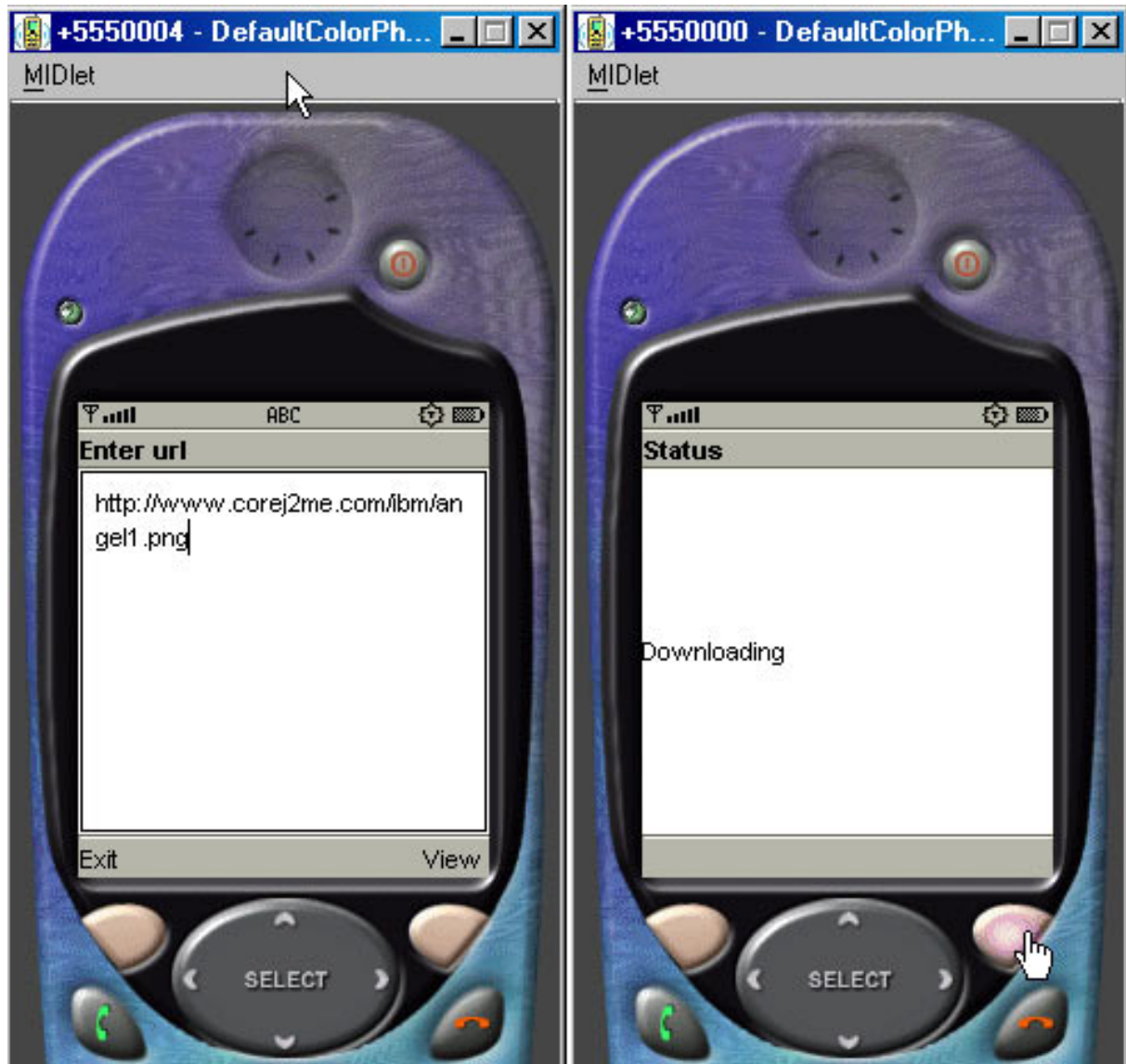
To start the `ViewImageThread` MIDlet, click **Launch**, as shown here.

Figure 14. Run the ViewImageThread MIDlet

Downloading Image

As before, you specify the URL of the image you would like to download. Once you choose 'View' a dialog box will be displayed showing the download has begun. See Figure 15. A background thread will be started that creates the network connection and starts the image transfer.

Figure 15. Alert dialog showing download status.



The dialog box is displayed for approximately 3 seconds. After that time, the dialog will disappear and the TextBox will once again become active. Notice in Figure 16 how the cursor can be moved about in the TextBox while the download is active.

Figure 16. Image downloading and TextBox is active.



Download complete

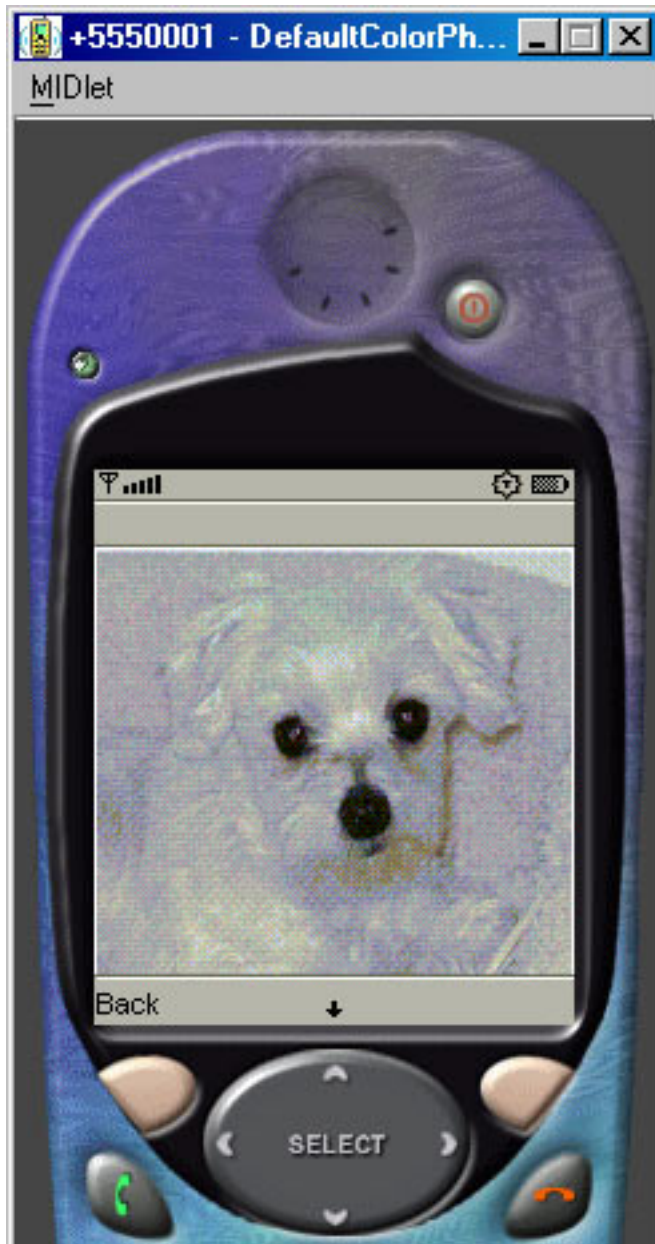
When the download is complete, an Alert dialog is displayed indicating the status of the download.

Figure 17. Alert dialog showing download status complete.



The final step is to view the image. Click **Done** to see the image displayed on the device.

Figure 18. Image downloaded and displayed on the device.



Code review: User interface

The primary interface has changed little over the previous MIDlet. I use a TextBox to get the URL, and a Form to display the downloaded image. The most notable change is the Alert dialog box definition.

```
public class ViewImageThread extends MIDlet implements CommandListener
{
    private Display display;
    private TextBox tbMain;
    private Alert alStatus;
    private Form fmViewImage;
    private Command cmExit;
    private Command cmView;
    private Command cmBack;
}
```

```
private static final int ALERT_DISPLAY_TIME = 3000;
Image im = null;

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

    // Create the Main textbox with a maximum of 75 characters
    tbMain = new TextBox("Enter url", "http://www.corej2me.com/ibm/ange

    // 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
    fmViewImage = new Form("");

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

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

```

Note the `ALERT_DISPLAY_TIME` variable. This indicates the number of milliseconds to display the Alert dialog when I first begin the download. I'll show the related code momentarily.

Code review: Event handling

As before, the `cmExit` and `cmBack` commands exit the MIDlet and return to the URL TextBox, respectively. If you look inside the code block for `cmView` event, you'll see the first reference to the Alert dialog. Notice the parameters that are passed to `showAlert()`. I'll cover these in the next panel.

Once the alert is shown, I create an instance of the `Download` class, this is essentially the background thread that downloads the image. The final call is to start the thread and begin the download.

```
/*-----
 * 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)
{
    // 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", false, tbMain);

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

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

```

Code review: Show alert

When displaying the Alert dialog, I include a message, indicate if the dialog is modal, and specify what component to show after the dialog is dismissed. The first call to `showAlert` (in previous panel) was as follows:

```
showAlert("Downloading", false, tbMain);
```

This requests the message "Downloading" be shown; the dialog be non-modal (displayed for a specified time -- 3000 milliseconds, which I defined earlier in `ALERT_DISPLAY_TIME`); and the user returned to the component `tbMain` once the dialog is dismissed.

```

/*-----
 * 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);
}

```

Code review: Download class

The Download class implements the Runnable interface, which indicates this is the background thread I have been alluding to. When the thread is started the method run() calls getImage() to download the image in the background.

```
/*-----  
* Class - Download  
*  
* Download an image file in a separate thread  
*-----*/  
class Download implements Runnable  
{  
    private String url;  
    private ViewImageThread MIDlet;  
    private boolean downloadSuccess = false;  
  
    public Download(String url, ViewImageThread MIDlet)  
    {  
        this.url = url;  
        this.MIDlet = MIDlet;  
    }  
  
    /*-----  
    * 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  
    {  
        ...  
    }  
}
```

Code review: Get image

The final method in the `Download` class is `getImage()`. This code is identical to the first MIDlet with the exception of the last few lines. See the example below.

```
/*-----  
* 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();  
            bStrm.close();  
        }  
  
        // 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);  
    else  
    {  
        MIDlet.im = im;  
        MIDlet.showImage(true);  
    }  
}
```

Once the download is complete, based on whether the image was successfully

retrieved, a call is made to `MIDlet.showImage()`, passing a boolean flag indicating the image status. The `showImage()` method is discussed in the next panel.

Code review: Show image

Called from the background thread, this method displays an Alert dialog indicating the success or failure of the download. Upon success, a new `ImageItem` is allocated to hold the downloaded image.

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

        // Alert followed by the form holding the image
        showAlert("Download Successful", true, fmViewImage);
    }
}
```

Note: This is a key transition point for the `MIDlet`. If the image was successfully downloaded, the `showAlert()` call is as follows:

```
showAlert("Download Successful", true, fmViewImage);
```

If the image download failed, the call is:

```
showAlert("Download Failure", true, tbMain);
```

In either case, an Alert modal dialog is shown. What's important to notice is the final

parameter. The difference lies with which component is shown once the dialog is dismissed. Upon a successful download, the Form `fmViewImage` is shown, which contains downloaded image. When the download fails, the URL TextBox, `tbMain`, is displayed, returning the user to primary program interface.

Section 5. Summary and resources

Summary

Establishing and communicating over a connection with J2ME is really quite simple. I demonstrated this concept by creating a MIDlet that creates a remote connection, downloads an image, and displays the result on a device emulator.

I concluded the tutorial by adding additional code to allow downloading of an image in a background thread. For most all but the simplest of MIDlets, understanding how to create and manage threads is very importance.

With this tutorial as a starting point, you can begin to write applications that cannot only communicate over a network, but can do so in an efficient manner without blocking, creating a more robust and responsive user experience.

Resources

- Click here for information on the HTTP Specification [Hypertext Transfer Protocol](http://www.ietf.org/rfc/rfc2616.txt) (<http://www.ietf.org/rfc/rfc2616.txt>).
- This IBM developerWorks tutorial [MIDlet development with the Wireless Toolkit](http://www-106.ibm.com/developerworks/edu/wi-dw-wikit-i.html) (<http://www-106.ibm.com/developerworks/edu/wi-dw-wikit-i.html>) guides you through the basic steps for MIDlet or J2ME compilation, preverification, and packaging.
- Information on the Java Development Kit 1.4.1 [JDK](http://java.sun.com/products/jdk/1.4.1) (<http://java.sun.com/products/jdk/1.4.1>) can be found here (<http://www-106.ibm.com/developerworks/edu/wi-dw-wikit-i.html>).
- Click here for the J2ME Wireless Toolkit [Wireless Toolkit](http://java.sun.com/products/j2mewtoolkit) (<http://java.sun.com/products/j2mewtoolkit>) (<http://java.sun.com/products/j2mewtoolkit>).
- Additional articles and resources can be found at [Core J2ME](http://www.CoreJ2ME.com) (<http://www.CoreJ2ME.com>).

Section 6. Feedback

Feedback

Please send us your feedback on this tutorial.

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.)

For more information about the Toot-O-Matic, visit www6.software.ibm.com/dl/devworks/dw-tootomatic-p .