7 Chapter

Writing Your Own Servlet

This chapter shows you how you can deploy your mapping application using servlets (server-side Java programs).

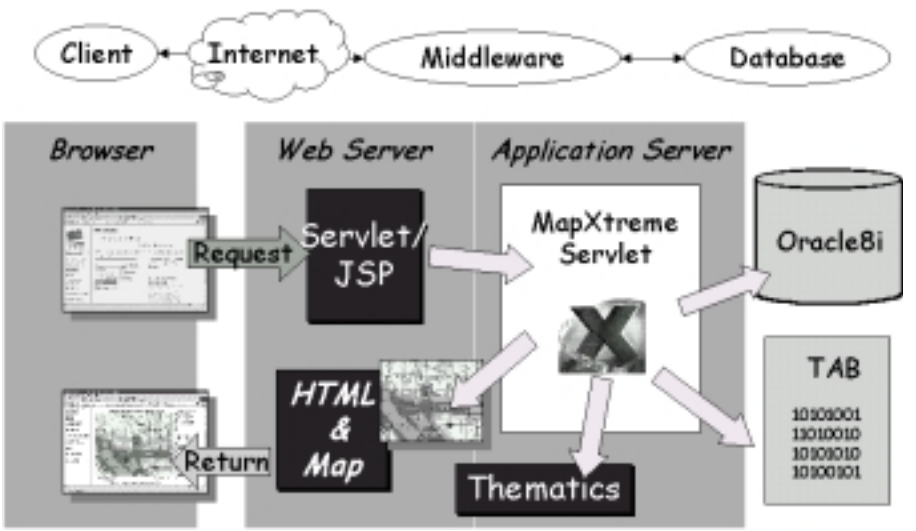
Introduction

Servlets are Java components that are used to extend the functionality of web servers. Servlets are to servers what applets are to browsers, except servlets have no graphical user interface.

The mapping engine in MapXtreme Java is deployed as a servlet. It conforms to the Java 2 Enterprise Edition architecture and must be run from within a servlet container, ideally one that is J2EE compliant (minimum JSDK 2.0) By doing so, the servlet container manages non-mapping tasks such as load balancing, threading, and fault tolerance, and allows MapXtremeServlet to do what it does best — handle requests for maps.

This chapter discusses how to develop a mapping servlet that acts as a "client servlet" to MapXtremeServlet. You would deploy your application in a three-tier configuration that has minimal requirements on the client side (e.g., a browser). This "thin-client" deployment is best when speed is important to you, such as for an Internet application. This configuration handles multiple requests with ease, and can be designed to forward requests to other servlets and servers.

The illustration below shows how a client servlet fits into a three-tier MapXtreme Java deployment.



Requirements for Using Servlets

Before compiling a servlet source program, you may need to configure your Java IDE to include a servlets jar file (e.g., servlets.jar) in your IDE's classpath.

Working with Servlets

Your servlet can contain a variety of mapping functionality to meet the needs of your application. For example, you can construct a servlet that offers basic map navigation such as pan, zoom in, zoom out, and measure distance between points. If you need a more sophisticated application, consider offering selection or thematic mapping capabilities.

This chapter covers the following topics:

- Sample servlet HTMLEmbeddedMapServlet (ships with MapXtreme Java)
- Servlet Wizard in JDeveloper 3.0 for Windows
- MapToolkit library of servlet methods
- Tutorial: Adding Thematic Shading to a Servlet

Sample Servlet: HTMLEmbeddedMapServlet

The HTMLEmbeddedMapServlet is a sample servlet that ships with MapXtreme Java. You can find it in \sampleapps\java\servlet. A pre-compiled version is also provided in a jar file in the MapXtreme directory (mxtjsampleservlet30.jar).

HTMLEmbeddedMapServlet provides an HTML page embedded with these basic mapping elements:

- Map frame that displays the map
- Radio buttons for Zoom In, Zoom Out, Pan
- Map Width box for user to type in new map width
- Apply button to apply the map width
- Layer Settings link that displays a table of layers that can be turned on or off
- Toggle for enlarging or reducing the map
- Scalebar

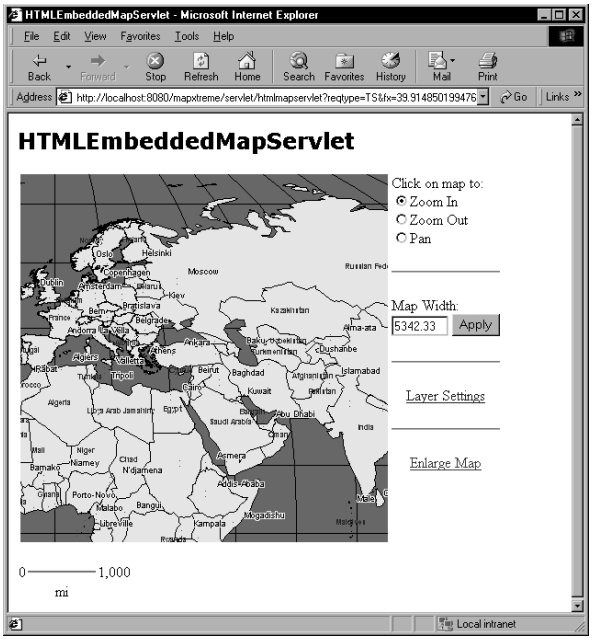
If you have already set up your servlet container using MapXtreme's Tomcat installer, you can run HTMLEmbeddedMapServlet. Open a browser and type the URL to MapXtremeServlet, such as:

```
http/localhost:8080/mapxtreme/servlet/htmlmapservlet
```

Your URL may be different depending on how you set it up.

If you haven't configured your servlet container and MapXtreme Java, see "Setting Up Your Servlet Container" on page 33 in Chapter 3: Getting Started, for instructions and troubleshooting guidelines.

For example, to customize this servlet change the map width or height by modifying the appropriate variables in HTMLEmbeddedMapServlet.java and recompiling.



Whenever you modify and recompile the servlet, you will have to copy its class files into the appropriate directory. For example, if you use JRun, copy the .class files into the JRun\servlets. If you use JavaWebServer, copy the .class files into JavaWebServer\servlets.

If you compile the samples, be sure to delete, move, or rename mxtjsampleservlets30.jar -- otherwise, your servlet container may use the

HTMLEmbeddedMapServlet.class file from the jar file, instead of using the class file that you compiled.

Note, however, that the HTMLEmbeddedMapServlet code sample allows you to change many settings without recompiling. The servlet loads many of its settings using standard servlet init parameters. For example, the name of the map to load (e.g., world.gst) can be overridden using an init parameter; thus, if you simply want to change the name of the map to load, you do not have to modify the servlet source code at all, you can simply edit the init parameter.

The following table describes the most important init parameters that are expected by the HTMLEmbeddedMapServlet sample. For a complete list of the init parameters used by the sample servlet, view the comments in HTMLEmbeddedMapServlet.java.

Init Parameter	Description	Example
filetoload	The full path to the map file that will be displayed (either a .gst or .mdf file).	C:\mxt\maps\world.gst
mappath	The path to the directory where geoset (.gst) map files are installed on the server. Not applicable for map definitions.	C:\mxt\maps
mapxtremeurl	The MapXtremeServlet URL	http://localhost:8080/servlet/com.mapinfo.mapxtreme.MapXtremeServlet

If you use a servlet container that provides an Administrator tool (such as JRun or JavaWebServer), use that tool to set up your initialization parameters, as described above. Some servlet containers might require you to specify init parameters in an XML file; the next two sections describe both.

Editing Init Parameters in Tomcat

You can modify init parameters for Tomcat by using a text editor to edit a web.xml file. The following example shows a <servlet></servlet> block that defines three init parameters for HTMLEmbeddedMapServlet -- the mappath, filetoload, and mapxtremeurl init parameters.

```
<servlet>
  <servlet-name>
    htmlmapservlet
  </servlet-name>
  <servlet-class>
    HTMLEmbeddedMapServlet
  </servlet-class>
  <init-param>
    <param-name>
      mappath
    </param-name>
    <param-value>
      C:\mxt\maps
    </param-value>
  </init-param>
  <init-param>
    <param-name>
      filetoload
    </param-name>
    <param-value>
      C:\mxt\maps\world.gst
    </param-value>
  </init-param>
  <init-param>
    <param-name>
      mapxtremeurl
    </param-name>
    <param-value>
      http://hostname/mapxtreme/servlet/mapxtreme[1]
    </param-value>
  </init-param>
</servlet>
```

1. The /mapxtreme reference at the end is the registered name for com.mapinfo.mapxtreme.MapXtremeServlet. It is set automatically during Tomcat/MapXtreme integration.

Editing Init Parameters in JavaWebServer

JavaWebServer provides an administrator utility that simplifies the task of adding or modifying init parameters. See also “Setting up MapXtreme with JavaWebServer 2.0” on page 39.

1. Run the JavaWebServer administrator (e.g., by browsing the URL http://localhost:9090/).
2. Click the Manage button, then click the Servlets button near the top of the dialog.
3. Click on the servlet name (in the list in the left of the dialog, under "Configure") for the HTMLEmbeddedMapServlet.
4. Click on the Properties tab to see a list of all init parameters currently defined for this servlet. This list is initially empty. To add an init parameter, click the Add button. Then type the init parameter name (such as filetoload) under "Name", and type in the value (such as C:\mxt\maps\asia.gst) under the Value column.

JDeveloper 3.x Servlet Wizard

MapXtreme 3.0 includes a "Wizard" for JDeveloper 3.x for Windows, which helps JDeveloper users configure a sample servlet. Instead of manually making a copy of HTMLEmbeddedMapServlet.java and manually configuring the various values inside the file, JDeveloper users can simply run the MapXtreme Map Servlet Wizard.

Note: The wizard is not available unless you checked the "JDeveloper Addins" check box in the MapXtreme Installer. This is for JDeveloper 3.0 or 3.1 for Windows only.

Using the MapXtreme Map Servlet Wizard addin also requires that you launch JDeveloper 3.x using a 1.2.2 internal VM.

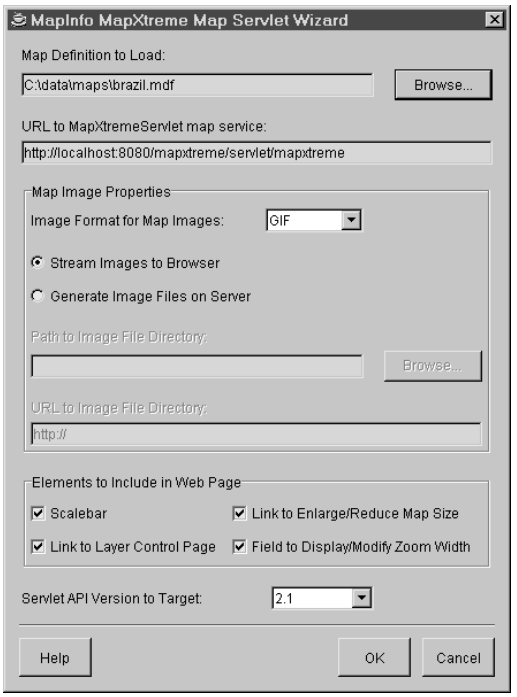
Once the Wizard is installed, you can bring up the Wizard either by running JDeveloper 3.x, and then:

- Choose File > New, go to the Web Objects tab, and double-click "MapInfo MapXtreme Map Servlet."

  or

- Choose File > New Project in JDeveloper. Then select "Project containing a new..." and select "MapInfo MapXtreme Map Servlet."

Whichever way you launch the Wizard, it displays a dialog box with various setup options (such as letting you choose whether the servlet should generate GIF or JPEG images). Choose the options that you want, then click OK. When you click OK, the wizard creates a new .java source code file in your active project. This .java file is basically a copy of the HTMLEmbeddedMapServlet sample that is included in the sampleapps directory — but it is a copy which has been modified to fit the selections you made in the dialog box.



In the Map Definition To Load field, you can type the full path to a map definition (.mdf) file, or the full path to a geoset (.gst) file.

At the bottom of the dialog is a menu allowing you to choose which servlet API version to use: 2.0, 2.1 (the default), or 2.2. You will need to change this setting to 2.0 if you are planning to deploy this servlet to Apache Jserv, because Jserv supports only Servlet API version 2.0.

Once you have run the Wizard to create a .java source file, you can edit, deploy,and run that servlet just as you would do with the regular HTMLEmbeddedMapServlet code example, described above.

Note: When you click OK, the Wizard generates a .java source code file, but you may need to make additional changes to your project before the source code will compile. Your project properties will need to include a library that references the MapXtreme jar files mxtj30.jar, devsup30.jar, mistyles30.jar and xml4j_1_1_16.jar.

Using the Servlet Utility Library (MapToolkit)

MapXtreme Java provides a library of helper methods for building a servlet. Given a MapJ object, the MapToolkit class helps you to construct the common elements of a map-enabled web page, such as a Layer Control form. The methods used in the sample application HTMLEmbeddedMapServlet are taken from this class.

Use this class to simplify the development of servlets. The following elements are included in the library:

Map Tools - radio buttons that identify map navigation elements (zoom in, zoom out, pan)

Zoom box - text field that displays the current map zoom and allows user to type in new zoom.

Layer Control - an HTML page that displays the layers in the map and allows the user to check or clear settings for selectability, visibility, and autolabeling.

Scale Bar - a viewable element that shows the scale for the map.

Map Size Toggle - link to enlarge or reduce the size of the map.

Methods in the Servlet MapToolkit

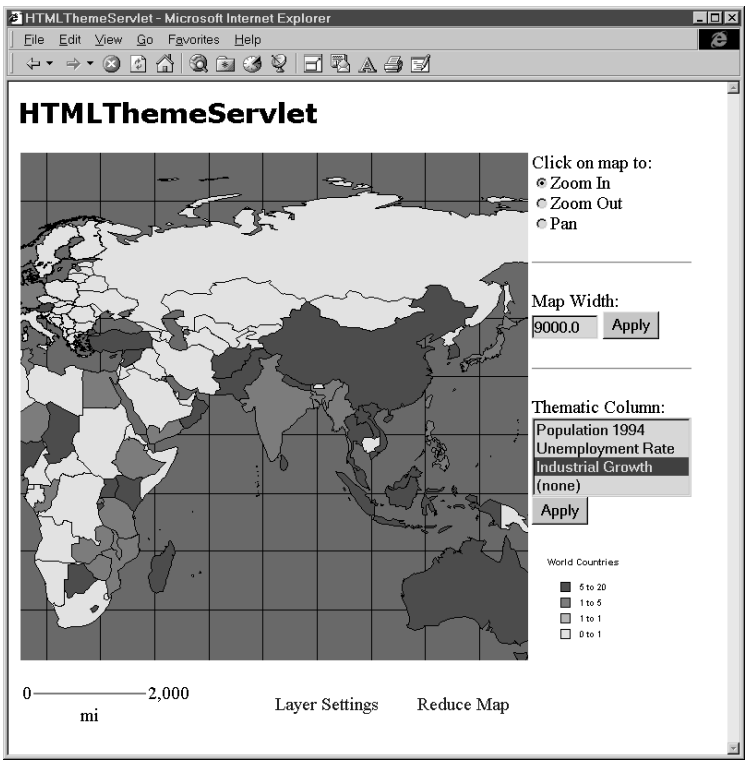
The MapToolkit provides the following methods. For a complete description of the MapToolkit class, see the HTML Reference installed on your machine under mapxtremejava\help\devsupport.

Method	Description
getHTMLLayerListControl	Returns a string representing an HTML page that acts as a "layer control dialog."
applyLayerSettings	Updates the map to reflect all options selected by the user in the Layer Control page.
getHTMLZoomControl	Returns a string representing a "zoom control" — a set of HTML tags that provide a text field for displaying the current map zoom width, and a Submit button to apply any new zoom width the user types in.
getHTMLMapToolsControl	Returns a string representing a set of radio buttons for Zoom In, Zoom Out, and Pan.
getHTMLScaleBar	Returns a string representing HTML syntax that defines a map scale bar. Specify a certain width, or specify zero and the scalebar will be sized to a round number approximately 1/4 the width of the map image.
getToolNumber	Returns an int that represents which tool number corresponds to which tool name.
getStr	Returns a string resource from a resource bundle.

Tutorial: Adding Thematic Capabilities to Your Servlet

Once you have successfully run the sample servlet, HTMLEmbeddedMapServlet, you might want to customize the servlet to include additional functionality. For example, you might want to give the user a way to apply thematic shading to the map.

This tutorial shows you how to add code to HTMLEmbeddedMapServlet to give it such thematic shading capabilities. The resulting servlet will display a list of column names (Population, IndustrialGrowth, etc.); the user can select a column name and click an Apply button to shade the World map according to that column. (Note: This exercise uses the world.gst table. If you use a different map, substitute the columns accordingly.)



This tutorial not only shows you how to create thematic shading, it can also help you to develop a better understanding of how the HTMLEmbeddedMapServlet works.

The thematic servlet example was created by making a copy of HTMLEmbeddedMapServlet, and adding code. This section explains the code that was added to support thematic shading.

Note: If you are in a hurry, you do not need to manually perform each of the steps described in this tutorial; the end result of the tutorial is provided in the sampleapps directory, by the name **HTMLThemeServlet.java**.

The tutorial covers the following:

- Overview of the Thematic Shading Process
- New Constants
- Displaying a List of Column Names
- Processing the Selected Column Name
- Displaying a Legend
- New Methods

Overview

Before inserting specific lines of code into HTMLEmbeddedMapServlet.java, let's summarize how the thematic shading feature will work.

- The servlet begins just as HTMLEmbeddedMapServlet begins; by generating an HTML page. The HTML page contains a form that displays a map image. The user has various ways of submitting the form (such as clicking the map to zoom in or out).
- We will add some new HTML tags to the map page, so that the page will also display a list of column names such as "Population" and "Unemployment". The user will select a column name from the list, then click an "Apply" button, to see the World layer shaded according to that column.
- When the user clicks the Apply button, the form is submitted. The servlet processes all form fields that are submitted. In doing so, the servlet will note which column name the user selected.
- If the servlet determines that the user clicked the Apply button to request thematic shading, then the servlet will call a new method (setTheme) to create that shading. The thematic shading basically color-codes the countries in the World table, according to the values in the data column that the user selected (e.g., countries with high Population appear in red, countries with low population appear in yellow).
- Once again, the servlet generates an HTML page to display the results. The page displays a map, although this time the map's appearance has changed: the World layer is now color-coded. Also, the page includes one additional

item: an **<img>** tag that displays a legend. The legend displays the numerical values associated with each color in the color-coded map.

Note that the servlet will be servicing different "types" of requests: requests for an HTML page; requests for an image of the map; and requests for an image of a legend. The first time the user goes to the servlet's URL:

```
http://localhost:8080/mapxtreme/servlet/theme
```

the servlet responds by sending down text representing an HTML page (i.e., a response of type "text/html"). The HTML page includes a tag that displays an image of the map; but the URL for that image tag is actually a servlet URL:

```
src="/servlet/theme?reqtype=IMG
```

When the browser attempts to display the map image, it makes a second request to the servlet. This time, the servlet replies by streaming an image down to the client (i.e., a response of type "image/gif").

If the user has created thematic shading, then the page contains another image tag, for an image of the legend. Again, the URL of this image tag is a servlet URL, but this servlet URL uses a slightly different format:

```
src="/servlet/theme?reqtype=LEGEND
```

In short, bear in mind that the servlet handles three different types of requests: HTML page, map image, and legend image. In other words, once you have added thematic shading to the map, the servlet will be called three times whenever the client re-displays the page.

With that plan in mind, we can begin adding code to the sample servlet. If you haven't already done so, make a copy of HTMLEmbeddedMapServlet.java. You might want to give your copy a different name, such as "Theme.java".

New Constants

Near the top of the file, where constants such as FF_MAP_IMAGE are defined, add these constants:

```
// Constants associated with thematic shading
private static final String FF_THEME_APPLY =
   "themeapply";
private static final String FF_THEME_COLUMN =
   "themecolumn";
```

```
private static final String FF_THEME_NONE = "(none)";
private static final String TABLE_TO_SHADE = "world.tab";
private static final String LEGEND_TITLE = "World
   Countries";
```

Note that theTABLE_TO_SHADE constant identifies a MapInfo table, world.tab. This tutorial will use theWorld table as an example. If you want to modify this example to use a different table name, changeTABLE_TO_SHADE accordingly. (Note, however, that other parts of the code reference specific columns in theWorld table, so you will also need to make other changes later on. See the **getHTMLColumnChooser** method defined below.)

The LEGEND_TITLE string is purely cosmetic; it appears as a title at the top of the thematic legend.

Displaying a List of Column Names

We want to enhance the map page to include a list of column names, such as "Unemployment Rate" and "Industrial Growth". To see thematic shading, the user will select a column name, then click the Apply button.

When the page first appears, no shading is displayed, and the default selected item in the list is "(none)".

To add items to the map page, modify the **getMapPage** method. Near the bottom of the getMapPage method, add code to display the list of column names and an Apply button. Most of this work is performed by calling the method **getHTMLColumnChooser,** which returns an HTML string that represents both the list of columns and the Apply button.

Beneath the list of column names, we will also add code to display an **<img>** tag, to display a legend for the thematic shading. However, we only display a legend when and if the user has actually created a theme.

To display the list of column names, the Apply button, and – conditionally – the legend, add the following code to the **getMapPage** method:

```
// Add a list of column names, to let the user choose
   which column

// to use for thematic shading.
```

```
String colName =
   (String)session.getValue("mapinfo.colname");

sb.append(getHTMLColumnChooser(colName));

// If the user created a theme, then display a legend to
   describe the theme.

if (colName != null && !colName.equals(FF_THEME_NONE)) {

sb.append("<img src=\"" + m_thisServletURL +

   "?" + FF_REQUEST_TYPE + "=LEGEND&refresh=" +
   rnd.nextLong() + "\">");

}
```

Where you place this code depends on where you want the list and the legend to appear on your page layout. However, make sure that you place it inside the <FORM> </FORM> block. You might insert the code just after the spot where we call **toolkit.getHTMLMapToolsControl**.

The code for the **getHTMLColumnChooser** method is provided later in this tutorial.

Processing the Selected Column Name

The servlet has a **getFormFieldsHT** method, which reads in all submitted form fields, performs field validation (e.g., to make sure the user didn't type in an invalid zoom width), and stores the results in a Hashtable object. Since we have added new form fields (a list of column names), we need to add code to the getFormFieldsHT method, so that the Hashtable will also contain the name of the column that the user chose.

Near the end of the getFormFieldsHT method, add two **if** blocks of code. The first **if** block tests whether the user clicked the Apply button that accompanies the list of column names; if the user did click the Apply button, call ht.put to record the fact that the user clicked the Apply button. (Later on, when we need to determine exactly how the user submitted the form, we will examine the HT_SUBMIT_BUTTON element of the Hashtable.)

Also add code to save the name of the column the user selected.

The code to add (near the end of the getFormFieldsHT) is shown in bold:

```
if (param != null) {
   ht.put(FF_REQUEST_TYPE, param);
}
// Handle case where user clicked on Apply Theme
if (req.getParameter(FF_THEME_APPLY) != null){
   ht.put(HT_SUBMIT_BUTTON, FF_THEME_APPLY);
}
// Note which column the user chose
param = req.getParameter(FF_THEME_COLUMN);
if (param != null) {
   ht.put(FF_THEME_COLUMN, param);
}
return ht;
```

Within the **sendHTMLResponse** method, we test whether the user submitted the page by clicking one of the various Submit buttons used in the application. Since we have added a new Submit button, add code to sendHTMLResponse to handle the case where the user has clicked the Apply button for the list of column names.

If you detect that the user clicked the Apply button, call **session.putValue** to save the name of the column that the user selected, and then call **setTheme** to apply the thematic shading. The code that you add is shown in bold:

```
if (strSubmitButton != null) {
   if (strSubmitButton.equals(FF_LS_APPLY)) {
      // User clicked Apply on Layer Settings page; apply
   changes.
      toolkit.applyLayerSettings(myMap);
   }
   else if (strSubmitButton.equals(FF_THEME_APPLY)) {
```

```
            // User clicked the Apply button to set thematic
        shading

            String columnName =
        (String)ht.get(FF_THEME_COLUMN);

            if (columnName == null) {

                columnName = FF_THEME_NONE;

            }

            session.putValue("mapinfo.colname", columnName);

            setTheme(myMap, TABLE_TO_SHADE, columnName,
        LEGEND_TITLE);

        }

        // TODO: If you add your own submit buttons to the
        page,

        // add an else if block to test for whether the user
        pressed

        // your new button, and respond accordingly.

    }
```

Note that we call setTheme even if the user selected the "(none)" item from the list of column names. In the case where the user selects "(none)", the setTheme method clears the theme from the World layer.

The code for the **setTheme** method is provided later in this tutorial.

Displaying a Legend

When the user adds thematic shading to the map, we want the page to include a legend that explains the meaning of the shading. The legend is displayed by including an <img> tag in the page. The code for generating the <img> tag was shown above, so you have already added it to the servlet:

```
        sb.append("<img src=\"" + m_thisServletURL +

            "?" + FF_REQUEST_TYPE + "=LEGEND&refresh=" +
        rnd.nextLong() + "\">");
```

Note the format of this image URL. This URL does not reference a static file that exists on the server; instead, the resulting URL looks like this:

<img src="/servlet/theme?reqtype=LEGEND&refresh=2691486241031874102">

This URL causes the browser to make an additional call back to the servlet. When we call the servlet this time, the servlet sees the query parameter "reqtype=LEGEND", and accordingly the servlet responds differently, by streaming an image of the legend down to the browser.

Locate the **try** block near the top of the **service** method. You will need to add an **else if** block, where you will call a new method called **sendImageResponseLegend**. The code that you add is shown in bold:

```
try {
if (strRequestType != null&&strRequestType.equals("IMG"))
   {
   // This servlet request was a request for a map image.
   // Stream an image directly down to the client.
   sendImageResponse(res, req);
}
else if (strRequestType != null&&
   strRequestType.equals("LEGEND")) {
   // This servlet request was a request for a legend
   image.
   // Stream an image directly down to the client.
   sendImageResponseLegend(res, req);
}
else {
   // This request is a request for an HTML page.
   sendHTMLResponse(res, req, ht, strRequestType);
   }
}
```

The code for the **sendImageResponseLegend** method is provided later in this tutorial.

New Methods

We added a **setTheme** method, which either sets or clears thematic shading.

```
private void setTheme(

   MapJ myMap, String tableName, String columnName,
String title) {

if (columnName == null ||
columnName.equals(FF_THEME_NONE)) {

   Layer lyr = myMap.getLayers().getLayer(tableName);

   lyr.getThemeList().removeAll();

} else {

   // The user did select a column; continue with the
theme....

   ColumnStatistics colStats;

   Vector rBreaks;

   int numBreaks = 4;

   Vector rends;

   RangedTheme rTheme;

   ThemeList tlist;

   Layer lyr = null;

   Rendition rendYellow = new Rendition();

   Rendition rendRed = new Rendition();

   lyr = myMap.getLayers().getLayer(tableName);

   lyr.getThemeList().removeAll();

   try {

      colStats =
lyr.fetchColumnStatistics(columnName);

      rBreaks = Bucketer.computeDistribution(

         numBreaks, colStats,
Bucketer.DISTRIBUTION_TYPE_EQUAL_COUNT);
```

```
            rendYellow.setValue(Rendition.FILL,
        Color.yellow);

            rendRed.setValue(Rendition.FILL, Color.red);

            rends =
        LinearRenditionSpreader.spread(numBreaks, rendYellow,
        rendRed);

            rTheme = new RangedTheme(columnName, rBreaks,
        rends, title);

            tlist = lyr.getThemeList();

            tlist.add(rTheme);

         } catch (Exception e) {

            e.printStackTrace();

         }

      }

   }
```

We added a **getHTMLColumnChooser** method, which returns a string of HTML, representing a list of column names and a Submit button to create shading for the specified column:

```
   private String getHTMLColumnChooser(String columnName) {

      StringBuffer sb = new StringBuffer();

      sb.append("<HR><P ALIGN=\"LEFT\">Thematic Shading
      Column:<BR>");

      sb.append("<SELECT NAME=\"" + FF_THEME_COLUMN + "\"
      SIZE=4>");

      String strSelected;

      if (columnName == null) {

         columnName = FF_THEME_NONE;

      }

      if (columnName.equals("Pop_1994")) {

         strSelected = "SELECTED ";
```

```
        } else {
            strSelected = "";
        }
        sb.append("<OPTION " + strSelected +
        "VALUE=Pop_1994>Population 1994</OPTION>");
        if (columnName.equals("Unempl_Rate")) {
            strSelected = "SELECTED ";
        } else {
            strSelected = "";
        }
        sb.append("<OPTION " + strSelected +
        "VALUE=Unempl_Rate>Unemployment Rate</OPTION>");
        if (columnName.equals("Indust_Growth")) {
            strSelected = "SELECTED ";
        } else {
            strSelected = "";
        }
        sb.append("<OPTION " + strSelected +
        "VALUE=Indust_Growth>Industrial Growth</OPTION>");
        if (columnName.equals(FF_THEME_NONE)) {
            strSelected = "SELECTED ";
        } else {
            strSelected = "";
        }
        sb.append("<OPTION " + strSelected + "VALUE=" +
        FF_THEME_NONE + ">(none)</OPTION>");
        sb.append("</SELECT><BR>");
        sb.append("<INPUT NAME=\"" + FF_THEME_APPLY + "\"
        TYPE=SUBMIT VALUE=\"Apply Shading\"></p>");
        return sb.toString();
    }
```

We added a **sendImageResponseLegend** method, which streams an image of the thematic legend down to the browser.

```
private  void sendImageResponseLegend(
   HttpServletResponse res, HttpServletRequest req) {
   HttpSession session = req.getSession(true);
   MapJ myMap = null;
   try {
      // Try to retrieve the user's previous MapJ object.
      myMap = (MapJ) session.getValue("mapinfo.mapj");
      if (myMap == null) {
         myMap = initMapJ();
      }
   } catch (Exception e) {
   }
   RangedThemeLegend rtl = getRangedThemeLegend(myMap,
   TABLE_TO_SHADE, 0);
   if (rtl != null) {
      res.setContentType(getMIMEType() );
      ServletOutputStream sos = null;
      try {
         sos = res.getOutputStream();
         rtl.toStream(sos, getMIMEType() );
      }
      catch (Exception e) {
         e.printStackTrace();
      }
      try {
         if (sos != null) {
            sos.close();
         }
      }  catch (Exception e) {
      }
   }
}
```

We also added a **getRangedThemeLegend** method, which is a simple helper method called from sendImageResponseLegend.

```
private RangedThemeLegend getRangedThemeLegend(
      MapJ myMap, String layerName, int legendNumber) {
   RangedThemeLegend rtLegend = null;
   Layer lyr = myMap.getLayers().getLayer(layerName);
   RangedTheme rt =
   (RangedTheme)lyr.getThemeList().elementAt(legendNumber
   );
   rtLegend =
   (RangedThemeLegend)rt.createDefaultLegend(null);
   return rtLegend;
}
```

The completed servlet is installed in the sampleapps directory, named **HTMLThemeServlet.java**. Note that the page layout for **HTMLThemeServlet.java** is slightly different than the page layout for HTMLEmbeddedMapServlet.java: the "Layer Settings" and "Enlarge Map" links are placed underneath the map, rather than to the right of the map, to make room for the list of column names. This change is purely cosmetic, and so it was not described in detail above. To see the exact code used to place the links underneath the map, view the source code for **HTMLThemeServlet.java**.