7.5 Using Servlets to Generate JPEG Images

Although servlets often generate HTML output, they certainly don't *always* do so. For example, Section 7.3 (Building Excel Spreadsheets) shows a servlet that builds Excel spreadsheets and returns them to the client. Here, we show you how to generate JPEG images.

First, let us summarize the two main steps servlets have to perform to build multimedia content.

1. **Inform the browser of the content type they are sending.** To accomplish this task, servlets set the `Content-Type` response header by using the `setContentType` method of `HttpServletResponse`.

2. **Send the output in the appropriate format.** This format varies among document types, of course, but in most cases you send binary data, not strings as you do with HTML documents. Consequently, servlets will usually get the raw output stream by using the `getOutputStream` method, rather than getting a `PrintWriter` by using `getWriter`.

Putting these two steps together, servlets that generate non-HTML content usually have a section of their `doGet` or `doPost` method that looks like this:

```
response.setContentType("type/subtype");
OutputStream out = response.getOutputStream();
```

Those are the two general steps required to build non-HTML content. Next, let's look at the specific steps required to generate JPEG images.

1. **Create a `BufferedImage`.**

   You create a `java.awt.image.BufferedImage` object by calling the `BufferedImage` constructor with a width, a height, and an image representation type as defined by one of the constants in the `BufferedImage` class. The representation type is not important, since we do not manipulate the bits of the `BufferedImage` directly and since most types yield identical results when converted to JPEG. We use `TYPE_INT_RGB`. Putting this all together, here is the normal process:

   ```
   int width = ...;
   int height = ...;
   BufferedImage image =
     new BufferedImage(width, height,
                       BufferedImage.TYPE_INT_RGB);
   ```

2. **Draw into the `BufferedImage`.**

   You accomplish this task by calling the image's `getGraphics` method, casting the resultant `Graphics` object to `Graphics2D`, then making use of Java 2D's rich set of drawing operations, coordinate transformations, font settings, and fill patterns to perform the drawing. Here is a simple example.

   ```
   Graphics2D g2d = (Graphics2D)image.getGraphics();
   g2d.setXxx(...);
   g2d.fill(someShape);
   g2d.draw(someShape);
   ```

3. **Set the `Content-Type` response header.**

As already discussed, you use the `setContentType` method of `HttpServletResponse` for this task. The MIME type for JPEG images is `image/jpeg`. Thus, the code is as follows.

```
response.setContentType("image/jpeg");
```

4. **Get an output stream.**

   As discussed previously, if you are sending binary data, you should call the `getOutputStream` method of `HttpServletResponse` rather than the `getWriter` method. For instance:

   ```
   OutputStream out = response.getOutputStream();
   ```

5. **Send the `BufferedImage` in JPEG format to the output stream.**

   Before JDK 1.4, accomplishing this task yourself required quite a bit of work. So, most people used a third-party utility for this purpose. In JDK 1.4 and later, however, the `ImageIO` class greatly simplifies this task. If you are using an application server that supports J2EE 1.4 (which includes servlets 2.4 and JSP 2.0), you are guaranteed to have JDK 1.4 or later. However, standalone servers are not absolutely required to use JDK 1.4, so be aware that this code depends on the Java version. When you use the `ImageIO` class, you just pass a `BufferedImage`, an image format type (`"jpg"`, `"png"`, etc.—call `ImageIO.getWriterFormatNames` for a complete list), and either an `OutputStream` or a `File` to the `write` method of `ImageIO`. Except for catching the required `IOException`, that's it! For example:

   ```
   try {
     ImageIO.write(image, "jpg", out);
   } catch(IOException ioe) {
     System.err.println("Error writing JPEG file: " + ioe);
   }
   ```

Listing 7.7 shows a servlet that reads `message`, `fontName`, and `fontSize` parameters and passes them to the `MessageImage` utility (Listing 7.8) to create a JPEG image showing the message in the designated face and size, with a gray, oblique-shadowed version of the message shown behind the main string. If the user presses the Show Font List button, then instead of building an image, the servlet displays a list of font names available on the server.

Listing 7.7 ShadowedText.java

```
package coreservlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.awt.*;

/** Servlet that generates JPEG images representing
 *  a designated message with an oblique-shadowed
 *  version behind it.
 */

public class ShadowedText extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    String wantsList = request.getParameter("showList");
    if (wantsList != null) {
      showFontList(response);
```

```
    } else {
      String message = request.getParameter("message");
      if ((message == null) || (message.length() == 0)) {
        message = "Missing 'message' parameter";
      }
      String fontName = request.getParameter("fontName");
      if ((fontName == null) || (fontName.length() == 0)) {
        fontName = "Serif";
      }
      String fontSizeString = request.getParameter("fontSize");
      int fontSize;
      try {
        fontSize = Integer.parseInt(fontSizeString);
      } catch(NumberFormatException nfe) {
        fontSize = 90;
      }
      response.setContentType("image/jpeg");
      MessageImage.writeJPEG
        (MessageImage.makeMessageImage(message,
                                       fontName,
                                       fontSize),
         response.getOutputStream());
    }
  }

  private void showFontList(HttpServletResponse response)
      throws IOException {
    PrintWriter out = response.getWriter();
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";
    String title = "Fonts Available on Server";
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=CENTER>" + title + "</H1>\n" +
                "<UL>");
    String[] fontNames = MessageImage.getFontNames();
    for(int i=0; i<fontNames.length; i++) {
      out.println("  <LI>" + fontNames[i]);
    }
    out.println("</UL>\n" +
                "</BODY></HTML>");
  }
}
```

Listing 7.8 MessageImage.java

```
package coreservlets;

import java.awt.*;
import java.awt.geom.*;
import java.awt.image.*;
import java.io.*;
import javax.imageio.*;

/** Utilities for building images showing shadowed messages.
 *  <P>
 *  Requires JDK 1.4 since it uses the ImageIO class.
 *  JDK 1.4 is standard with J2EE-compliant app servers
 *  with servlets 2.4 and JSP 2.0. However, standalone
 *  servlet/JSP engines require only JDK 1.3 or later, and
 *  version 2.3 of the servlet spec requires only JDK
```

```
 *  1.2 or later. So, although most servers run on JDK 1.4,
 *  this code is not necessarily portable across all servers.
 */

public class MessageImage {

  /** Creates an Image of a string with an oblique
   *  shadow behind it. Used by the ShadowedText servlet.
   */

  public static BufferedImage makeMessageImage(String message,
                                               String fontName,
                                               int fontSize) {

    Font font = new Font(fontName, Font.PLAIN, fontSize);
    FontMetrics metrics = getFontMetrics(font);
    int messageWidth = metrics.stringWidth(message);
    int baselineX = messageWidth/10;
    int width = messageWidth+2*(baselineX + fontSize);
    int height = fontSize*7/2;
    int baselineY = height*8/10;
    BufferedImage messageImage =
      new BufferedImage(width, height,
                        BufferedImage.TYPE_INT_RGB);
    Graphics2D g2d = (Graphics2D)messageImage.getGraphics();
    g2d.setBackground(Color.white);
    g2d.clearRect(0, 0, width, height);
    g2d.setFont(font);
    g2d.translate(baselineX, baselineY);
    g2d.setPaint(Color.lightGray);
    AffineTransform origTransform = g2d.getTransform();
    g2d.shear(-0.95, 0);
    g2d.scale(1, 3);
    g2d.drawString(message, 0, 0);
    g2d.setTransform(origTransform);
    g2d.setPaint(Color.black);
    g2d.drawString(message, 0, 0);
    return(messageImage);
  }

  public static void writeJPEG(BufferedImage image,
                               OutputStream out) {
    try {
      ImageIO.write(image, "jpg", out);
    } catch(IOException ioe) {
      System.err.println("Error outputting JPEG: " + ioe);
    }
  }

  public static void writeJPEG(BufferedImage image,
                               File file) {
    try {
      ImageIO.write(image, "jpg", file);
    } catch(IOException ioe) {
      System.err.println("Error writing JPEG file: " + ioe);
    }
  }

  public static String[] getFontNames() {
    GraphicsEnvironment env =
      GraphicsEnvironment.getLocalGraphicsEnvironment();
    return(env.getAvailableFontFamilyNames());
  }
```

```
  /** We need a Graphics object to get a FontMetrics object
   *  (an object that says how big strings are in given fonts).
   *  But, you need an image from which to derive the Graphics
   *  object. Since the size of the "real" image will depend on
   *  how big the string is, we create a very small temporary
   *  image first, get the FontMetrics, figure out how
   *  big the real image should be, then use a real image
   *  of that size.
   */

  private static FontMetrics getFontMetrics(Font font) {
    BufferedImage tempImage =
      new BufferedImage(1, 1, BufferedImage.TYPE_INT_RGB);
    Graphics2D g2d = (Graphics2D)tempImage.getGraphics();
    return(g2d.getFontMetrics(font));
  }
}
```

Listing 7.9 (Figure 7-6) shows an HTML form used as a front end to the servlet. Figures 7-7 through 7-10 show some possible results. Just to simplify experimentation, Listing 7.10 presents an interactive application that lets you specify the message and font name on the command line, outputting the image to a file.

Listing 7.9 ShadowedText.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>JPEG Generation Service</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<H1 ALIGN="CENTER">JPEG Generation Service</H1>
Welcome to the <I>free</I> trial edition of our JPEG
generation service. Enter a message, a font name,
and a font size below, then submit the form. You will
be returned a JPEG image showing the message in the
designated font, with an oblique "shadow" of the message
behind it. Once you get an image you are satisfied with, right-click
on it (or click while holding down the SHIFT key) to save
it to your local disk.
<P>
The server is currently on Windows, so the font name must
be either a standard Java font name (e.g., Serif, SansSerif,
or Monospaced) or a Windows font name (e.g., Arial Black).
Unrecognized font names will revert to Serif. Press the
"Show Font List" button for a complete list.

<FORM ACTION="/servlet/coreservlets.ShadowedText">
  <CENTER>
    Message:
    <INPUT TYPE="TEXT" NAME="message"><BR>
    Font name:
    <INPUT TYPE="TEXT" NAME="fontName" VALUE="Serif"><BR>
    Font size:
    <INPUT TYPE="TEXT" NAME="fontSize" VALUE="90"><P>
    <INPUT TYPE="SUBMIT" VALUE="Build Image"><P>
    <INPUT TYPE="SUBMIT" NAME="showList" VALUE="Show Font List">
  </CENTER>
</FORM>

</BODY></HTML>
```

Listing 7.10 ImageTest.java

```
package coreservlets;

import java.io.*;

public class ImageTest {
  public static void main(String[] args) {
    String message = "Testing";
    String font = "Arial";
    if (args.length > 0) {
      message = args[0];
    }
    if (args.length > 1) {
      font = args[1];
    }
    MessageImage.writeJPEG
      (MessageImage.makeMessageImage(message, font, 40),
       new File("ImageTest.jpg"));
  }
}
```

Figure 7-6. Front end to the image-generation servlet.

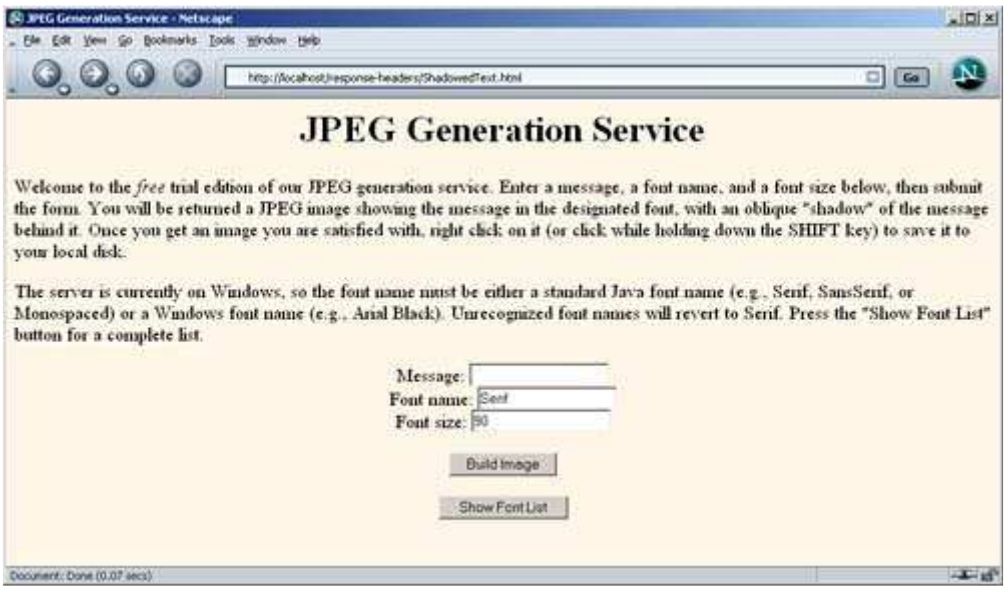


Figure 7-7. Result of servlet when the client selects Show Font List.

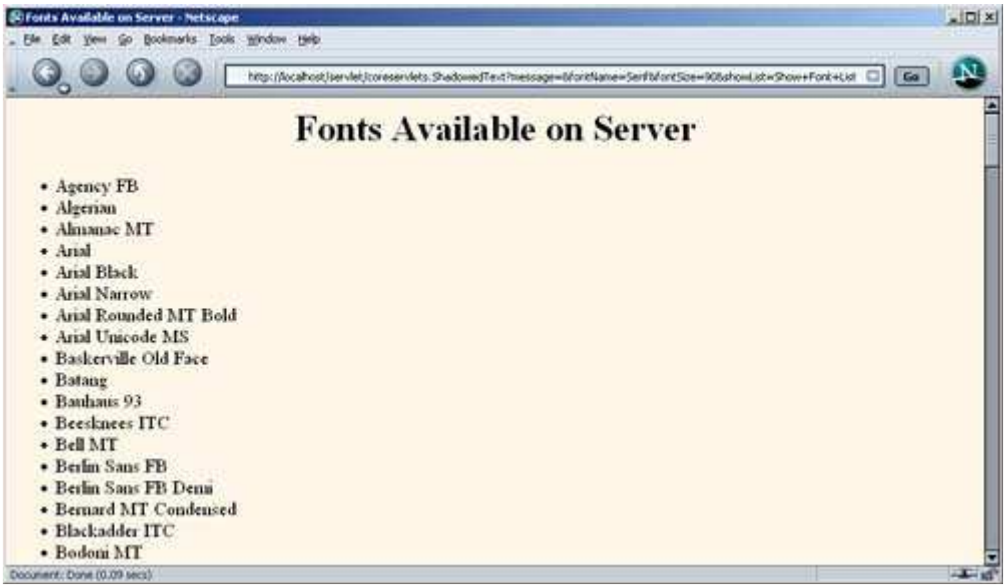
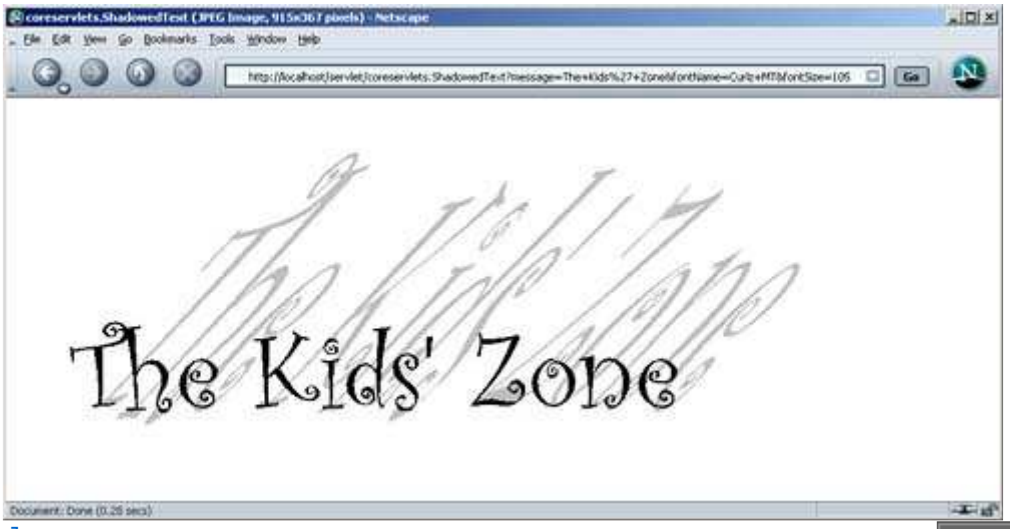


Figure 7-10. A third possible result of the image-generation servlet.

Figure 7-8. One possible result of the image-generation servlet. The client can save the image to disk as `somename.jpg` and use it in Web pages or other applications.

Figure 7-9. A second possible result of the image-generation servlet.



PREVIOUS NEXT