RZ 3683 (# 99693) 02/12/2007
Computer Science 97 pages

Research Report

Programming Hamlets

R. Pawlitzek

IBM Research
Zurich Research Laboratory
8803 Rüschlikon, Switzerland
rpa@zurich.ibm.com

Programming Hamlets

Separate content from presentation with this servlet-based framework

Skill Level: Intermediate

René Pawlitzek (rpa@zurich.ibm.com)
Research and Development Engineer
IBM

24 May 2005

Updated 13 Mar 2007

See various aspects of Hamlet programming through a number of practical Hamlet examples. The examples are part of WebZEC (Web-based Zurich Event Console) -- a fast, browser-based console to quickly navigate in intrusion-detection alarms. With these samples, you can develop a good understanding of how to use Hamlets for Web-based application development and how Hamlets work.

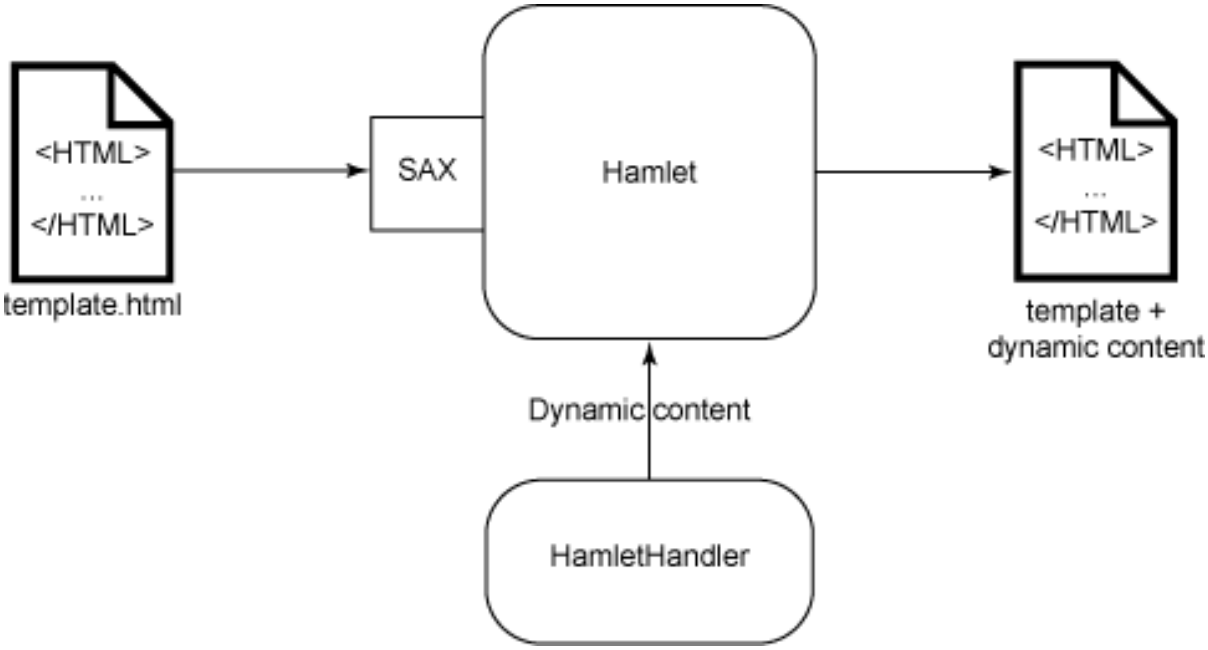
Section 1. Introduction

About Hamlets

Hamlets provide an easy-to-use, easy-to-understand, lightweight, small-footprint, servlet-based content creation framework that facilitates the development of Web-based applications. The Hamlet framework not only supports but also enforces the complete separation of content and presentation. Its simple and elegant design does not hide the familiar underlying servlet infrastructure.

A *Hamlet* is a servlet extension that uses SAX (the Simple API for XML) to read XHTML template files containing presentation; while the XHTML file is being read, the Hamlet uses a small set of callback functions (implemented by a `HamletHandler`) to dynamically add content on the fly to those places in the template that are marked with special tags and IDs. This is illustrated in Figure 1.

Figure 1. A Hamlet uses SAX for reading content from a template file and calls a HamletHandler to add dynamic content



Hamlets should be used in combination with other Web technologies. An ideal blend for a Hamlet-based Web application consists of a number of components:

- Hamlets written in the Java™ language provide the code that is responsible for creating the dynamic content.
- XHTML template files contain the basic layout of dynamic Web pages and thus provide the presentation.
- Cascading Style Sheets (CSS) give Web applications a consistent look and feel.

The project name *Hamlets* used in this tutorial refers to an internal project for development of a servlet-based framework for separation of content and presentation in Web-based applications. For a more thorough introduction to Hamlets and additional information, you can read my previous developerWorks articles "Introducing Hamlets," "Implementing Hamlets," and "Compiling Hamlets" (see Resources for links).

About this tutorial

This tutorial aims to illustrate various aspects of Hamlet programming by providing a number of practical Hamlet examples. The samples should give you a good understanding of how Hamlets are used for Web-based application development and how Hamlets work. In the first example, Clock example, I will draw parallels to standard Windows®-based C or C++ development to illustrate how Hamlets make Web-based development easy.

The code for both the Clock Example and the subsequent examples is part of the

Web-based Zurich Event Console (WebZEC), a browser-based application for monitoring intrusion detection events. WebZEC provides an intuitive graphical user interface for security analysts to monitor and investigate intrusion alarms. Its front end was built solely with Hamlets in a very short time frame.

Prerequisites and notes on the sample code

This tutorial assumes that the reader has experience with HTML or XHTML; thus, the XHTML code used in the examples is not explained. It is also assumed that the reader is familiar with the Java language and the concepts of object-oriented programming. Hamlets are servlet extensions, and therefore a good understanding of Java servlets is essential.

The examples in this tutorial were built with the Apache Ant build utility and deployed on the Apache Tomcat servlet container. They require the hamlet.jar library, which is the core of the Hamlet framework. You can find the complete source code that makes up this library in Appendix A. The Hamlet code is listed in this tutorial for your edification. It has been released on the IBM alphaWorks site as the *IBM servlet-based content creation framework* (see Resources for a link).

The two files in Appendix B -- ContextProperties.java and Utilities.java -- contain functionality that is used by the examples. These files are not really necessary to program Hamlets; however, they make life a lot easier. They could be included with a Web-based application or they could be put into a library (such as util.jar) for reuse. It is important to note that Utilities.java depends on mail.jar, the JavaMail API.

Section 2. Clock example

Introduction to the clock example

If you were asked to develop a servlet-based application to show the current time, you would probably come up with something similar to the following code:

```
/**
 *
 * © Copyright International Business Machines Corporation
2004, 2006.
 * All rights reserved.
 *
 * The 'Clock' servlet provides the current time.
 *
 * File     : Clock.java
 * Created  : 2004/03/22
 *
 * @author    René Pawlitzek  (rpa@zurich.ibm.com)
 * @version   1.00, 2006/12/01
 * @since     JDK 1.3
```

```
 *
 */


package com.ibm.webzec.apps.server;


import java.io.*;
import java.util.*;
import java.text.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.ibm.webzec.libs.util.*;


public class Clock extends HttpServlet {


  public void doGet (HttpServletRequest req,
HttpServletResponse res) throws ServletException, IOException {

    // get formats
    ContextProperties props = ContextProperties.getProperties
(this);
    String format = props.getStringProperty ("ShortTimeFormat",
"HH:mm");
    String timeZone = props.getStringProperty ("TimeZone",
"GMT");
    SimpleDateFormat dateFormat = new SimpleDateFormat
(format);
    dateFormat.setTimeZone (Utilities.getTimeZone (timeZone));

    res.setContentType ("text/html");
    PrintWriter out = res.getWriter ();

    out.println ("<HTML>");
    out.println ("  <HEAD>");
    out.println ("    <META HTTP-EQUIV=Refresh CONTENT=10>");
    out.println ("    <TITLE>Clock</TITLE>");
    out.println ("  </HEAD>");
    out.println ("  <BODY BGCOLOR=#CFCFCF>");
    out.println ("    <TABLE WIDTH=100%>");
    out.println ("      <TR>");
    out.println ("      <TD VALIGN=middle ALIGN=center
HEIGHT=30>");
    out.println ("      <FONT SIZE=2 FACE=Helvetica>");
    try {
      Date now = new Date ();
      String str = dateFormat.format (now);
      out.println (str);
    } catch (Exception e) {
    } // try
    out.println ("      </FONT>");
    out.println ("      </TD>");
    out.println ("      </TR>");
    out.println ("    </TABLE>");
    out.println ("  </BODY>");
    out.println ("</HTML>");

  } // doGet


  public String getServletInfo () {
    return "Clock servlet";
  } // getServletInfo


} // Clock


/* ----- End of File ----- */
```

As you can see, HTML code is embedded in the Java code inside the `doGet()`

method. In general, you do not want your HTML code to contain any formatting information (such as font size, color, and so on); you should use Cascading Style Sheets (CSS) instead. Even worse, this example mixes HTML with Java code. This is almost never a good idea because it results in severe maintenance problems:

- You can no longer use HTML tools to prepare, check, and view the HTML code, because it's buried in Java code and difficult to extract.
- Java developers and Web designers are forced to work on the same source files, which creates bottlenecks and increases the opportunity for errors.

The Hamlet framework not only eases but enforces the complete separation of content and presentation. It also solves the maintenance problems mentioned above. Take a closer look.

Step 1: Create the XHTML template for the clock

In the world of traditional Windows application programming, a developer uses a dialog editor to create resource files. These files typically end in an .rc extension and contain the controls of dialogs. The Hamlet framework is not much different. A Web designer uses an (X)HTML editor to create XHTML template files that contain the presentation of a Web application. Placeholders (dummy values) are used for sections in the templates that will be replaced when the templates are served at runtime. For the clock example, you can create the following XHTML code with the help of an (X)HTML editor:

```
<HTML>
  <HEAD>
    <META HTTP-EQUIV="Refresh" CONTENT="10" />
    <TITLE>Clock</TITLE>
    <LINK REL="stylesheet" TYPE="text/css" HREF="Status.css" />
  </HEAD>
  <BODY>
    <TABLE CLASS="container">
      <TR>
        <TD HEIGHT="30">
          <P CLASS="time">
            12:34
          </P>
        </TD>
      </TR>
    </TABLE>
  </BODY>
</HTML>
```

In this example, the string `12:34` is a placeholder. Such a placeholder is useful during the design phase to help perfect a pleasant layout, but it will be replaced at runtime.

Step 2: Add special tags for the clock

After creating the resource files, a Windows programmer continues by replacing all

numerical control IDs generated by the dialog editor with C/C++ preprocessor names, typically stored in a C/C++ header file:

```
#define ID_ABOUT_DIALOG_OK      10000
#define ID_ABOUT_DIALOG_CANCEL  10001
#define ID_ABOUT_DIALOG_HELP    10002
```

The Hamlet framework does not work much differently. All sections within the XHTML template files that will be dynamically created or repeated are marked with a <REPLACE> or <REPEAT> tag together with an `ID` attribute. The `ID` attribute is the only link between content and presentation. It separates the two completely.

```
<HTML>
  <HEAD>
    <META HTTP-EQUIV="Refresh" CONTENT="10" />
    <TITLE>Clock</TITLE>
    <LINK REL="stylesheet" TYPE="text/css" HREF="Status.css" />
  </HEAD>
  <BODY>
    <TABLE CLASS="container">
      <TR>
        <TD HEIGHT="30">
          <P CLASS="time">
            <REPLACE ID="time">12:34</REPLACE>
          </P>
        </TD>
      </TR>
    </TABLE>
  </BODY>
</HTML>
```

For the clock example, you will want to replace the string `12:34` (the placeholder) with the current time at runtime. Therefore, you mark that section with a <REPLACE> tag and add an `ID` attribute with the value `time`. (Note that you won't need any <REPEAT> tags for this example; you'll see how they work later in this tutorial.)

Step 3: Write the Hamlet code

Next, the Windows developer writes C/C++ code and uses the preprocessor names to access the controls in the dialog. Typically, you will see code that looks something like this:

```
setWindowText (GetDlgItem (hwnd, ID_ABOUT_DIALOG_VERSION), buffer);
```

`GetDlgItem()` returns the handle (a reference) to the **VERSION** label in the **About** dialog. The handle is then used in `setWindowText()` to set the label's text property.

A Hamlet programmer will now start writing the Java code that is responsible for creating the dynamic content -- the current time, in your case. First, you would create a public class, `Clock`, that extends `Hamlet`. The `Hamlet` class is an extension of `HttpServlet` and inherits the familiar servlet methods: `init()`,

`doGet()`, `doPost()`, `getServletInfo()`, and `destroy()`. You would use some code in the clock's `init()` method to configure logging. Note that the configuration parameters are stored as context properties.

```
public class Clock extends Hamlet {

  ...

  public void init () throws ServletException {
    // get properties
    ContextProperties props = ContextProperties.getProperties (this);
    // configure logging
    Utilities.configLog (props);
    category.debug ("init");
  } // init
  ...

} // Clock
```

Once a user sends a request for the deployed clock Web application, the Hamlet's `doGet()` method is invoked. `doGet()` needs to respond to the user request by returning an HTML page that shows the current time. Remember that part of this work is already done; in the section above entitled "Step 2: Add special tags for the clock", you prepared an XHTML template file containing a placeholder -- `12:34` -- for the current time. You'll pass the name of this template file to the `serveDoc()` method of the Hamlet framework. In addition to the template, you'll pass a `handler` to the `serveDoc()` method. The handler is an instance of `ClockHandler` and is responsible for providing the current time. It is created with `new ClockHandler (this, dateFormat)`.

```
public class Clock extends Hamlet {

  ...

  public void doGet (HttpServletRequest req, HttpServletResponse res) throws
ServletException {
    try {
      category.debug ("doGet");
      // get properties
      ContextProperties props = ContextProperties.getProperties (this);
      // get formats
      String format = props.getStringProperty ("ShortTimeFormat");
      String timeZone = props.getStringProperty ("TimeZone");
      category.debug ("short time format: " + format);
      category.debug ("time zone: " + timeZone);
      SimpleDateFormat dateFormat = new SimpleDateFormat (format);
      dateFormat.setTimeZone (Utilities.getTimeZone (timeZone));
      // serve document
      HamletHandler handler = new ClockHandler (this, dateFormat);
      serveDoc (req, res, "ClockTemplate.html", handler);
    } catch (Exception e) {
      category.error ("", e);
      throw new ServletException (e);
    } // try
  } // doGet
  ...

} // Clock
```

The `dateFormat` is required by the handler to properly format the current time. It is configured in `doGet()` with the time zone and short time format, which are declared

in the deployment descriptor file (web.xml) and retrieved with a call to `ContextProperties.getProperties()`. (You'll see the entire content of the deployment descriptor in the section below entitled "Step 4: Compile and build the clock project.")

```
  ...

  <!-- time & date config -->

  <context-param>
    <param-name>TimeZone</param-name>
    <param-value>GMT+1</param-value>
  </context-param>
  <context-param>
    <param-name>ShortTimeFormat</param-name>
    <param-value>HH:mm</param-value>
  </context-param>

  ...
```

The `serveDoc()` method performs most of the actual magic. With the help of a SAX reader (obtained from a pool of readers), it reads the content of the XHTML template file and invokes the `handler`'s `getElementReplacement()` callback method when a <REPLACE> tag is encountered. In the section above entitled "Step 2: Add special tags for the clock," you put one <REPLACE> tag in the XHTML template file to mark the section where you want the current time inserted. Therefore, you can expect a call from the Hamlet framework asking for the current time. In preparation, you need to implement the `getElementReplacement()` method in the private class `ClockHandler`. The code in this method provides the actual time to the Hamlet framework, which will use it to replace the placeholder at runtime.

```
public class Clock extends Hamlet {

  ...

  private static class ClockHandler extends HamletHandler  {

    private  SimpleDateFormat  dateFormat;

    public ClockHandler (Hamlet hamlet, SimpleDateFormat dateFormat) {
      super (hamlet);
      this.dateFormat = dateFormat;
    } // ClockHandler

    public String getElementReplacement  (String id, String name, Attributes
atts) throws Exception {
      if (id.equals ("time"))
        return dateFormat.format (new Date ());
      return "?";
    } // getElementReplacement

  } // ClockHandler
  ...
} // Clock
```

In this example, you don't need to check the `ID` attribute (stored in the `id` parameter) in the `getElementReplacement()` callback, because you marked only one section in the XHTML template file with the <REPLACE> tag. You might

have just returned the current time without checking the `ID` attribute. The `ID` attribute becomes important when you have more than one marked section in your XHTML template file -- if you had both `time` and `date` sections, for example. The `ID` attribute helps your code decide what information to provide in such cases. It controls the creation of dynamic content.

Here is the complete code for the Clock Hamlet:

```
/**
 *
 * © Copyright International Business Machines Corporation 2004, 2006.
 * All rights reserved.
 *
 * The 'Clock' servlet provides the current time.
 *
 * File     : Clock.java
 * Created  : 2004/05/19
 *
 * @author    Rene Pawlitzek  (rpa@zurich.ibm.com)
 * @version   1.00, 2006/11/27
 * @since     JDK 1.3
 *
 */


package com.ibm.webzec.apps.server;


import java.util.*;
import java.text.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.ibm.hamlet.*;
import com.ibm.webzec.libs.util.*;
import org.apache.log4j.*;
import org.xml.sax.*;


public class Clock extends Hamlet {


  // log4j
  private static Category category = Category.getInstance
(Clock.class.getName ());


  private static class ClockHandler extends HamletHandler {

    private  SimpleDateFormat  dateFormat;

    public ClockHandler (Hamlet hamlet, SimpleDateFormat dateFormat) {
      super (hamlet);
      this.dateFormat = dateFormat;
    } // ClockHandler

    public String getElementReplacement (String id, String name, Attributes
atts) throws Exception {
      if (id.equals ("time"))
        return dateFormat.format (new Date ());
      return "?";
    } // getElementReplacement

  } // ClockHandler


  public void init () throws ServletException {
    // get properties
    ContextProperties props = ContextProperties.getProperties (this);
    // configure logging
    Utilities.configLog (props);
```

```
    category.debug ("init");
  } // init


  public void doGet (HttpServletRequest req, HttpServletResponse res) throws
ServletException {
    try {
      category.debug ("doGet");
      // get properties
      ContextProperties props = ContextProperties.getProperties (this);
      // get formats
      String format = props.getStringProperty ("ShortTimeFormat");
      String timeZone = props.getStringProperty ("TimeZone");
      category.debug ("short time format: " + format);
      category.debug ("time zone: " + timeZone);
      SimpleDateFormat dateFormat = new SimpleDateFormat (format);
      dateFormat.setTimeZone (Utilities.getTimeZone (timeZone));
      // serve document
      HamletHandler handler = new ClockHandler (this, dateFormat);
      serveDoc (req, res, "ClockTemplate.html", handler);
    } catch (Exception e) {
      category.error ("", e);
      throw new ServletException (e);
    } // try
  } // doGet


  public String getServletInfo () {
    category.debug ("getServletInfo");
    return "Clock servlet";
  } // getServletInfo


  public void destroy () {
    category.debug ("destroy");
  } // destroy


} // Clock


/* ----- End of File ----- */
```

Step 4: Compile and build the clock project

In general, a Windows developer uses a makefile to compile the C/C++ sources and resource files for a project. The makefile usually also contains instructions to link the compiled objects together into an executable.

For the Clock Web application, you can write an Ant script to compile and build the project. To successfully compile and run the code, you'll need a few libraries: hamlet.jar (the core of the Hamlet framework), log4j.jar (the logging package), and xerces.jar (the XML parser). You can find links to information on Ant, Xerces, and log4j in the Resources section.

The following Ant script produces a Web application archive (WAR), webzec.war, from the Clock Hamlet.

```
<!-- build.xml -->


<!-- author: Rene Pawlitzek (rpa@zurich.ibm.com) -->
```

```
<project name="build" default="dist" basedir=".">


  <!-- set global properties for this build script -->
  <property name="src" value="..\src\java"/>
  <property name="dst" value=".\webapp\WEB-INF\classes"/>
  <property name="dst2" value=".\webapp\WEB-INF"/>
  <property name="dst3" value=".\webapp"/>


  <target name="init">
    <tstamp/>
    <echo message="Today: ${TODAY}"/>
    <mkdir dir="${dst}"/>
  </target>


  <target name="compile" depends="init">
    <!-- compile java code -->
    <javac srcdir="${src}" destdir="${dst}"/>
  </target>


  <target name="copy" depends="compile">
    <!-- copy web.xml file -->
    <copy todir="${dst2}">
      <fileset dir="${src}/com/ibm/webzec/apps/server">
        <include name="web.xml"/>
      </fileset>
    </copy>
    <copy todir="${dst3}">
      <fileset dir="${src}/com/ibm/webzec/apps/server">
         <include name="*.gif"/>
         <include name="*.html"/>
         <include name="*.css"/>
      </fileset>
    </copy>
  </target>


  <target name="dist" depends="copy">
    <!-- jar class files -->
    <jar jarfile="webzec.war" basedir="${dst3}"/>
  </target>


</project>


<!-- End of File -->
```

If you want to run this Web application with the Apache Tomcat servlet container, the Web Application Archive must contain a deployment descriptor (web.xml) like the following. For more information on Tomcat, see the links in the Resources.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

  <display-name>WebZEC</display-name>
  <description>
    The web-based Zurich Event Console (WebZEC)
  </description>
```

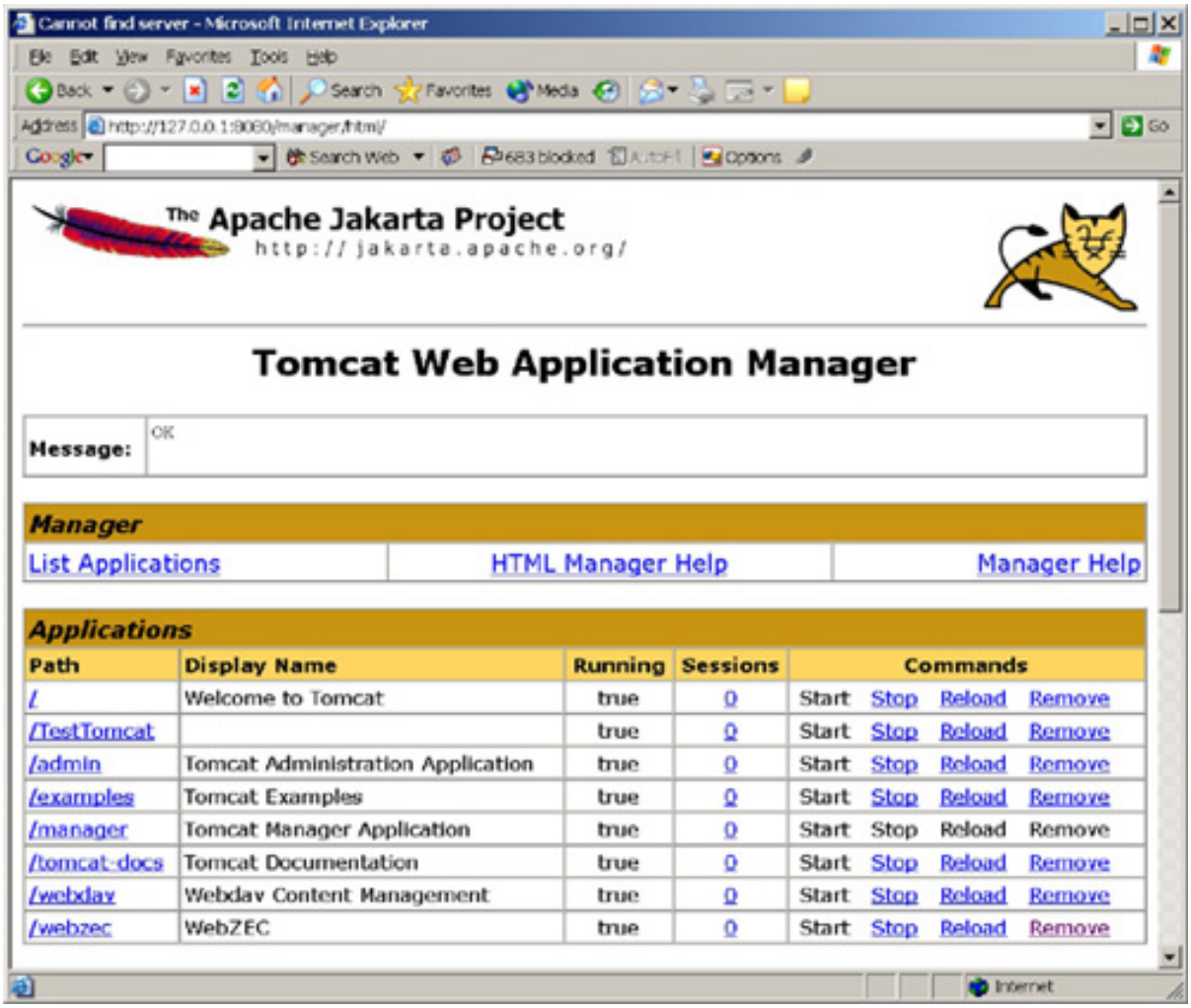
```
<!-- log4j config -->

<context-param>
  <param-name>log4j.rootCategory</param-name>
  <param-value>DEBUG, A1</param-value>
</context-param>
<context-param>
  <param-name>log4j.appender.A1</param-name>
  <param-value>org.apache.log4j.FileAppender</param-value>
</context-param>
<context-param>
  <param-name>log4j.appender.A1.File</param-name>
  <param-value>c:\\WebZEC.log</param-value>
</context-param>
<context-param>
  <param-name>log4j.appender.A1.layout</param-name>
  <param-value>org.apache.log4j.PatternLayout</param-value>
</context-param>
<context-param>
  <param-name>log4j.appender.A1.layout.ConversionPattern</param-name>
  <param-value>%-4r [%t] %-5p %c %x - %m%n</param-value>
</context-param>


<!-- time & date config -->

<context-param>
  <param-name>TimeZone</param-name>
  <param-value>GMT+1</param-value>
</context-param>
<context-param>
  <param-name>ShortTimeFormat</param-name>
  <param-value>HH:mm</param-value>
</context-param>


<!-- servlet config -->

<servlet>
  <servlet-name>
    Clock
  </servlet-name>
  <servlet-class>
    com.ibm.webzec.apps.server.Clock
  </servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>
    Clock
  </servlet-name>
  <url-pattern>
    /Clock.html
  </url-pattern>
</servlet-mapping>

<servlet>
  <servlet-name>
    Login
  </servlet-name>
  <servlet-class>
    com.ibm.webzec.apps.server.Login
  </servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>
    Login
  </servlet-name>
  <url-pattern>
    /Login.html
  </url-pattern>
</servlet-mapping>

<servlet>
  <servlet-name>
```

```
      PropertiesView
    </servlet-name>
    <servlet-class>
      com.ibm.webzec.apps.server.PropertiesView
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>
      PropertiesView
    </servlet-name>
    <url-pattern>
      /PropertiesView.html
    </url-pattern>
  </servlet-mapping>

  <servlet>
    <servlet-name>
      ListViewList
    </servlet-name>
    <servlet-class>
      com.ibm.webzec.apps.server.ListViewList
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>
      ListViewList
    </servlet-name>
    <url-pattern>
      /ListViewList.html
    </url-pattern>
  </servlet-mapping>

  <servlet>
    <servlet-name>
      EventView
    </servlet-name>
    <servlet-class>
      com.ibm.webzec.apps.server.EventView
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>
      EventView
    </servlet-name>
    <url-pattern>
      /EventView.html
    </url-pattern>
  </servlet-mapping>

  <servlet>
    <servlet-name>
      OptionView
    </servlet-name>
    <servlet-class>
      com.ibm.webzec.apps.server.OptionView
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>
      OptionView
    </servlet-name>
    <url-pattern>
      /OptionView.html
    </url-pattern>
  </servlet-mapping>

  <servlet>
    <servlet-name>
      ErrorView
    </servlet-name>
    <servlet-class>
      com.ibm.webzec.apps.server.ErrorView
    </servlet-class>
  </servlet>
```

```
  <servlet-mapping>
    <servlet-name>
      ErrorView
    </servlet-name>
    <url-pattern>
      /ErrorView.html
    </url-pattern>
  </servlet-mapping>

</web-app>
```

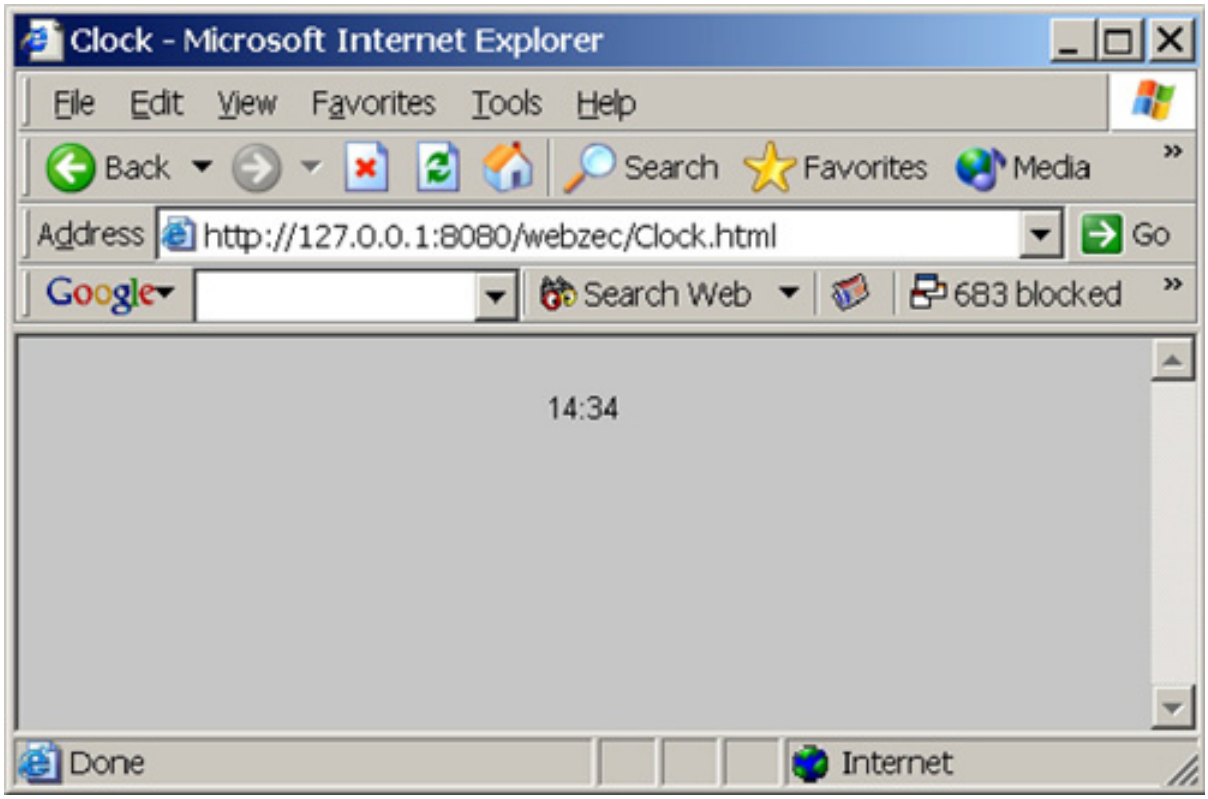
Deploy the webzec.jar file with the help of the Tomcat Web Application Manager:

Figure 2. Deploy the webzec.jar file from the Tomcat Web Application Manager



After deployment, you can now access the clock Web application using your favorite browser:

Figure 3. View of the clock Web application in a browser



Congratulations! You have just developed, compiled, built, and deployed your first Hamlet-based Web application.

Section 3. Login example

Introduction to the login example

In the next example, you will continue your examination of WebZEC. Most sophisticated Web applications require some sort of authentication; in this section, the Login example demonstrates how you can help set up this kind of functionality with Hamlets.

Create the XHTML code for the login

Start this project by producing an XHTML login page, as shown in Figure 4.

Figure 4. A sample XHTML login page

Here is the page's XHTML code:

```
<HTML>
  <HEAD>
    <TITLE>Welcome to WebZEC</TITLE>
    <LINK REL="stylesheet" TYPE="text/css" HREF="View.css" />
  </HEAD>
  <BODY>
    <P CLASS="title">
      <BR />Welcome to WebZEC!
    </P>
    <P CLASS="text">
      Please enter your UserID and Password
    </P>
    <FORM ACTION="Login.html" METHOD="POST">
      <TABLE CLASS="dialog" CELLSPACING="10" ALIGN="CENTER">
        <TR>
          <TH><P CLASS="label">User ID:</P></TH>
          <TD><INPUT CLASS="entryfield" TYPE="TEXT" NAME="UserID" SIZE="25"
ACCESSKEY="u" /></TD>
        </TR>
        <TR>
          <TH><P CLASS="label">Password:</P></TH>
          <TD><INPUT CLASS="entryfield" TYPE="PASSWORD" NAME="Password"
SIZE="25" ACCESSKEY="p" /></TD>
        </TR>
        <TR>
          <TD COLSPAN="2">
            <P CLASS="buttons">
              <INPUT CLASS="button" TYPE="SUBMIT" VALUE=" Login " />
              <INPUT CLASS="button" TYPE="RESET" VALUE=" Clear " />
            </P>
          </TD>
```

```
        </TR>
      </TABLE>
    </FORM>
    <P CLASS="text">
      (Mode: <REPLACE ID="Mode">Demo</REPLACE>)
    </P>
    <INCLUDE ID="Copyright" SRC="Copyright.html" />
  </BODY>
</HTML>
```

A closer look reveals that the formatting information (font size, colors, and so on) is separated from the actual page and stored in a Cascading Style Sheet (View.css) for reuse. The code has a placeholder (`Demo`) for the value of the *mode,* and the section is marked with the `<REPLACE>` tag together with an `ID` attribute with the value `Mode`. (The mode is an indicator of the Web application's configuration; possible values are `Demo`, `Debug`, and so on. Such an indicator on the Login page can be very helpful.) The template also contains an `<INCLUDE ID="Copyright" SRC="Copyright.html" />` tag to display a reusable copyright message.

Write the Hamlet for the login

You are now ready to write a Hamlet that provides the dynamic content at runtime (the actual value of the mode) and that processes the user ID and password input when a user clicks the **Login** button. First, create a public class, `Login`, that extends `Hamlet`. In this class's `init()` method, the context properties are read, logging is initialized, and an authorization processor is retrieved (from the authorization processor factory) and configured.

```
public class Login extends Hamlet {

  ...

  public void init () throws ServletException {
    try {
      // get properties
      ContextProperties props = ContextProperties.getProperties (this);
      // configure logging
      Utilities.configLog (props);
      category.debug ("init");
      // log properties
      props.debug ();
      // get mode
      String mode = props.getStringProperty ("Mode");
      // configure authorization instance
      AuthProcessorInterface authorization =
AuthProcessorFactory.getAuthProcessor (mode);
      authorization.configure (props, null);
    } catch (Exception e) {
      category.error ("", e);
      throw new ServletException (e);
    } // try
  } // init
  ...

} // Login
```

The `doGet()` method is called when a user request is received. In this example, the name of the Login XHTML template file (LoginTemplate.html) is passed to the `serveDoc()` method of the Hamlet framework together with a `handler`. The

`handler` is an instance of `LoginHandler`. It is created with `new LoginHandler (this, mode)` and provides the actual content (the mode) for the template. Prior to this, the mode is retrieved from the context properties.

```
public class Login extends Hamlet {

  ...
  public void doGet (HttpServletRequest req, HttpServletResponse res) throws
ServletException {
    try {
      category.debug ("doGet");
      // get properties
      ContextProperties props = ContextProperties.getProperties (this);
      // get mode
      String mode = props.getStringProperty ("Mode", "n/a");
      // serve document
      HamletHandler handler = new LoginHandler (this, mode);
      serveDoc (req, res, "LoginTemplate.html", handler);
    } catch (Exception e) {
      category.error ("", e);
      throw new ServletException (e);
    } // try
  } // doGet
  ...

} // Login
```

With the help of a SAX reader, the `serveDoc()` method parses the content of the XHTML template file. The `handler`'s `getElementReplacement()` method is called when the <REPLACE ID="Mode"> tag is encountered. The code in the `getElementReplacement()` method provides the actual value for the mode. In this example, it is again not strictly necessary to check the value of the `ID` attribute (stored in the `id` parameter), because the XHTML template file only includes one marked section. Nevertheless, the code checks the `id` parameter as a good practice.

```
public class Login extends Hamlet {

  ...
  private static class LoginHandler extends HamletHandler {

    private String mode;

    public LoginHandler (Hamlet hamlet, String mode) {
      super (hamlet);
      this.mode = mode;
    } // LoginHandler

    public String getElementReplacement (String id, String name, Attributes
atts) throws Exception {
      if (id.equals ("Mode"))
        return mode;
      return "?";
    } // getElementReplacement

  } // LoginHandler  ...

} // Login
```

So far, this example is not much different from the previous one. However, this code has one additional feature: the ability to process input (the user ID and password). When the user clicks the **Login** button on the Login page, the `doPost()` method is

called; in this method, the user ID and password are retrieved with the `getParameter()` method. The parameters are passed to the authorization processor (obtained with the `getAuthProcessor()` method). If the user is authorized to access this application, the code creates and initializes a session object and redirects him or her to a separate ListView.html page (the content of which is unimportant for the purposes of this tutorial). If authorization fails, the code presents the AccessDenied.html page.

```
public class Login extends Hamlet {

  ...
  public void doPost (HttpServletRequest req, HttpServletResponse res) throws
ServletException {
    try {
      category.debug ("doPost");
      // get properties
      ContextProperties props = ContextProperties.getProperties (this);
      // get parameters
      String userID = req.getParameter ("UserID");
      String password = req.getParameter ("Password");
      category.debug ("UserID : " + userID);
      category.debug ("Password : " + password);
      // get mode
      String mode = props.getStringProperty ("Mode");
      // get authorization instance
      AuthProcessorInterface authorization =
AuthProcessorFactory.getAuthProcessor (mode);
      // is user allowed?
      if (authorization.isAuthorized (userID, password)) {
        // create a session
        HttpSession session = req.getSession ();
        // set current view
        session.setAttribute (WebApp.currentView, "ListView.html");
        // set default options
        OptionView.setDefaultAttributes (session);
        // create web application
        WebApp.createWebApp (props, userID, password);
        // go to current view
        String currentView = (String) session.getAttribute
(WebApp.currentView);
        res.sendRedirect (res.encodeURL (req.getContextPath () + "/" +
currentView));
      } else {
        res.sendRedirect (res.encodeURL (req.getContextPath () +
"/AccessDenied.html"));
      } // if
    } catch (Exception e) {
      category.error ("", e);
      throw new ServletException (e);
    } // try
  } // doPost
  ...

} // Login
```

View the complete Hamlet code for the login

Here is the complete code for the Login Hamlet:

```
/**
 *
 * © Copyright International Business Machines Corporation 2004, 2006.
 * All rights reserved.
```

```
 *
 * The 'Login' servlet handles login requests.
 *
 * File     : Login.java
 * Created  : 2004/03/16
 *
 * @author    Rene Pawlitzek  (rpa@zurich.ibm.com)
 * @version   1.00, 2006/11/28
 * @since     JDK 1.3
 *
 */


package com.ibm.webzec.apps.server;


import javax.servlet.*;
import javax.servlet.http.*;
import com.ibm.hamlet.*;
import com.ibm.webzec.libs.db.*;
import com.ibm.webzec.libs.util.*;
import org.apache.log4j.*;
import org.xml.sax.*;


public class Login extends Hamlet {


  // log4j
  private static Category category = Category.getInstance
(Login.class.getName ());


  private static class LoginHandler extends HamletHandler {

    private String mode;

    public LoginHandler (Hamlet hamlet, String mode) {
      super (hamlet);
      this.mode = mode;
    } // LoginHandler

    public String getElementReplacement (String id, String name, Attributes
atts) throws Exception {
      if (id.equals ("Mode"))
        return mode;
      return "?";
    } // getElementReplacement

  } // LoginHandler


  public void init () throws ServletException {
    try {
      // get properties
      ContextProperties props = ContextProperties.getProperties (this);
      // configure logging
      Utilities.configLog (props);
      category.debug ("init");
      // log properties
      props.debug ();
      // get mode
      String mode = props.getStringProperty ("Mode");
      // configure authorization instance
      AuthProcessorInterface authorization =
AuthProcessorFactory.getAuthProcessor (mode);
      authorization.configure (props, null);
    } catch (Exception e) {
      category.error ("", e);
      throw new ServletException (e);
    } // try
  } // init
```

```
  public void doPost (HttpServletRequest req, HttpServletResponse res) throws
ServletException {
    try {
      category.debug ("doPost");
      // get properties
      ContextProperties props = ContextProperties.getProperties (this);
      // get parameters
      String userID = req.getParameter ("UserID");
      String password = req.getParameter ("Password");
      category.debug ("UserID : " + userID);
      category.debug ("Password : " + password);
      // get mode
      String mode = props.getStringProperty ("Mode");
      // get authorization instance
      AuthProcessorInterface authorization =
AuthProcessorFactory.getAuthProcessor (mode);
      // is user allowed?
      if (authorization.isAuthorized (userID, password)) {
        // create a session
        HttpSession session = req.getSession ();
        // set current view
        session.setAttribute (WebApp.currentView, "ListView.html");
        // set default options
        OptionView.setDefaultAttributes (session);
        // create web application
        WebApp.createWebApp (props, userID, password);
        // go to current view
        String currentView = (String) session.getAttribute
(WebApp.currentView);
        res.sendRedirect (res.encodeURL (req.getContextPath () + "/" +
currentView));
      } else {
        res.sendRedirect (res.encodeURL (req.getContextPath () +
"/AccessDenied.html"));
      } // if
    } catch (Exception e) {
      category.error ("", e);
      throw new ServletException (e);
    } // try
  } // doPost


  public void doGet (HttpServletRequest req, HttpServletResponse res) throws
ServletException {
    try {
      category.debug ("doGet");
      // get properties
      ContextProperties props = ContextProperties.getProperties (this);
      // get mode
      String mode = props.getStringProperty ("Mode", "n/a");
      // serve document
      HamletHandler handler = new LoginHandler (this, mode);
      serveDoc (req, res, "LoginTemplate.html", handler);
    } catch (Exception e) {
      category.error ("", e);
      throw new ServletException (e);
    } // try
  } // doGet


  public String getServletInfo () {
    category.debug ("getServletInfo");
    return "Login servlet";
  } // getServletInfo


  public void destroy () {
    category.debug ("destroy");
  } // destroy


} // Login
```

```
/* ----- End of File ----- */
```

Note that you can perform the `doPost()` processing with another Hamlet to separate it from the rendering code. In so doing, you would implement the MVC design pattern, which is beneficial for more complex applications.

Section 4. PropertiesView example

Introduction to the PropertiesView example

I continue this tutorial with the PropertiesView example. PropertiesView displays the configuration properties of the Web application -- that is, the context parameters stored in the web.xml file:

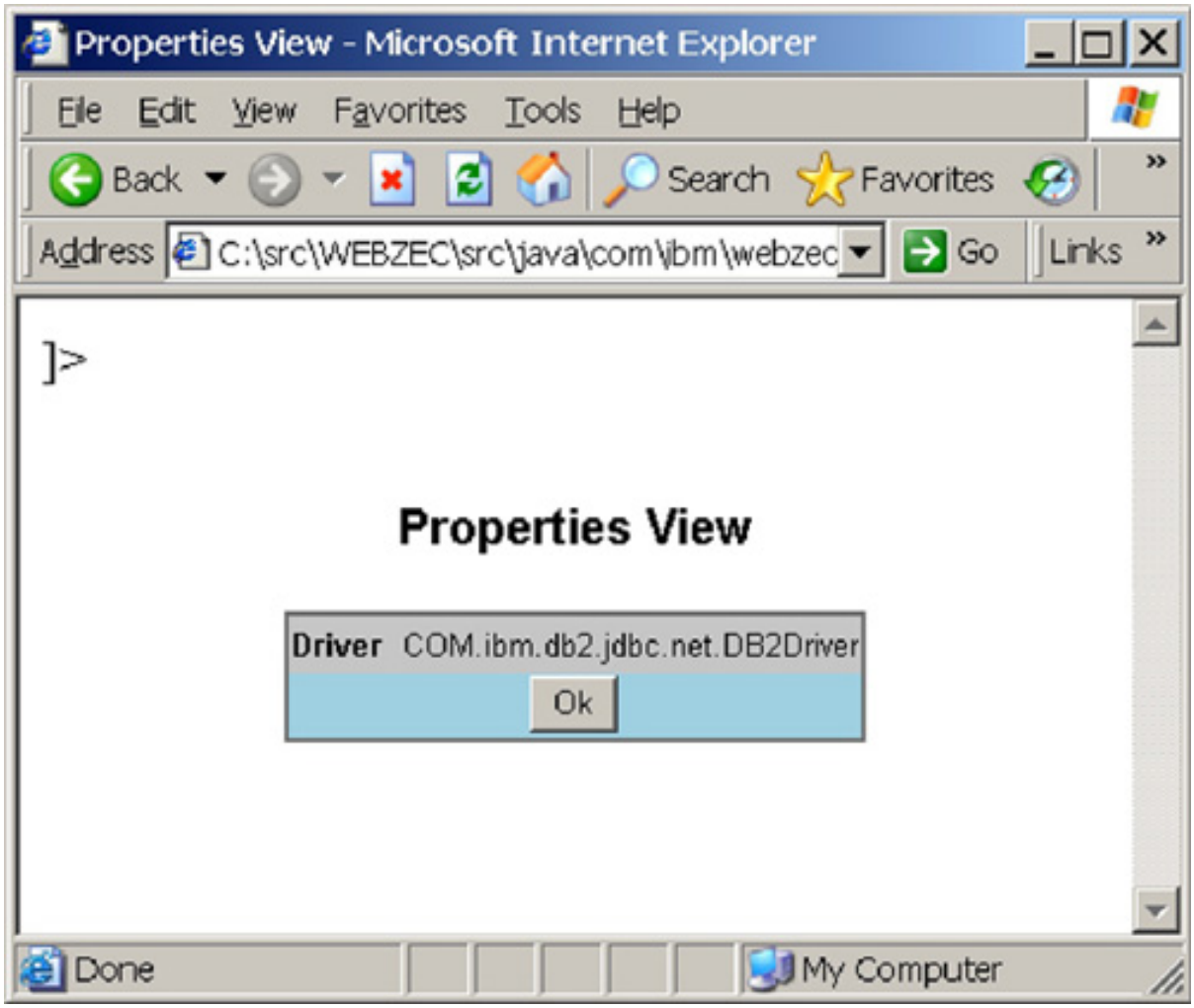
```
...

<!-- mode of operation -->

<context-param>
  <param-name>Mode</param-name>
  <param-value>demo</param-value>
</context-param>


<!-- log4j config -->

<context-param>
  <param-name>log4j.rootCategory</param-name>
  <param-value>DEBUG, A1</param-value>
</context-param>
<context-param>
  <param-name>log4j.appender.A1</param-name>
  <param-value>org.apache.log4j.FileAppender</param-value>
</context-param>
<context-param>
  <param-name>log4j.appender.A1.File</param-name>
  <param-value>c:\\WebZEC.log</param-value>
</context-param>
<context-param>
  <param-name>log4j.appender.A1.layout</param-name>
  <param-value>org.apache.log4j.PatternLayout</param-value>
</context-param>
<context-param>
  <param-name>log4j.appender.A1.layout.ConversionPattern</param-name>
  <param-value>%-4r [%t] %-5p %c %x - %m%n</param-value>
</context-param>


<!-- mail config -->

<context-param>
  <param-name>MailHost</param-name>
  <param-value>mailer.zurich.ibm.com</param-value>
</context-param>
<context-param>
  <param-name>MailSender</param-name>
  <param-value>webzec@zurich.ibm.com</param-value>
</context-param>
```

```
<context-param>
  <param-name>MailReceiver</param-name>
  <param-value>rpa@zurich.ibm.com</param-value>
</context-param>


<!-- time & date config -->

<context-param>
  <param-name>TimeZone</param-name>
  <param-value>GMT+1</param-value>
</context-param>
<context-param>
  <param-name>DateFormat</param-name>
  <param-value>MM/dd/yy  HH:mm:ss</param-value>
</context-param>
<context-param>
  <param-name>TimeFormat</param-name>
  <param-value>HH:mm:ss</param-value>
</context-param>
<context-param>
  <param-name>ShortTimeFormat</param-name>
  <param-value>HH:mm</param-value>
</context-param>
...
```

Create XHTML code for PropertiesViewTemplate

The project begins again with the creation of an XHTML template (PropertiesViewTemplate.html), as shown in Figure 5.

Figure 5. Create an XHTML template (PropertiesViewTemplate.html)



Here's the code:

```
<!DOCTYPE WebZEC [ <!ENTITY nbsp " "> ]>
<HTML>
  <HEAD>
    <TITLE>Properties View</TITLE>
    <LINK REL="stylesheet" TYPE="text/css" HREF="View.css" />
  </HEAD>
  <BODY>
    <P CLASS="title">
      <BR />Properties View
    </P>
    <FORM ACTION="PropertiesView.html" METHOD="POST">
      <TABLE CLASS="dialog" CELLPADDING="1" CELLSPACING="0" ALIGN="CENTER">
        <REPEAT ID="Rows">
        <TR ID="Row" CLASS="odd">
          <TH><P CLASS="name"><REPLACE ID="Name">Driver</REPLACE></P></TH>
          <TD> </TD>
          <TD><P CLASS="value"><REPLACE
ID="Value">COM.ibm.db2.jdbc.net.DB2Driver</REPLACE></P></TD>
        </TR>
        </REPEAT>
        <TR>
          <TH COLSPAN="3">
            <P CLASS="buttons">
              <INPUT CLASS="button" TYPE="Submit" VALUE=" Ok " />
            </P>
          </TH>
        </TR>
      </TABLE>
```

```
    </FORM>
  </BODY>
</HTML>
```

A careful look at the code reveals that the <TABLE> tag contains just a single row. The row itself contains two <REPLACE> tags and is surrounded by the <REPEAT> tag.

Write the Hamlet for PropertiesView

Next, you need to create a public class, `PropertiesView`, that extends `Hamlet`. The class's `init()` method loads the configuration properties by calling `ContextProperties.getProperties()`. The properties are then passed to the `Utilities.configLog()` method to configure logging.

```
public class PropertiesView extends Hamlet {

  ...
  public void init () throws ServletException {
    // get properties
    ContextProperties props = ContextProperties.getProperties (this);
    // configure logging
    Utilities.configLog (props);
    category.debug ("init");
  } // init
  ...

} // PropertiesView
```

Remember, the `doGet()` method is invoked when a user request is received. The `doGet()` method passes the XHTML template file (PropertiesViewTemplate.html) to the Hamlet's `serveDoc()` method together with a previously created instance (`handler`) of the `PropertiesViewHandler` class. As seen in the previous examples, the handler provides the dynamic content (the context properties) for the template. Note that an HTTP session is required for all this to happen. An HTTP session is created when the user logs on successfully. The user is redirected to the LoginHere.html page if the HTTP session does not exist -- in other words, if no login has been performed.

```
public class PropertiesView extends Hamlet {

  ...
  public void doGet (HttpServletRequest req, HttpServletResponse res) throws
ServletException {
    try {
      category.debug ("doGet");
      HttpSession session = req.getSession (false);
      if (session != null) {
        // get properties
        ContextProperties props = ContextProperties.getProperties (this);
        // serve document
        HamletHandler handler = new PropertiesViewHandler (this, props);
        serveDoc (req, res, "PropertiesViewTemplate.html", handler);
      } else {
        res.sendRedirect (res.encodeURL (req.getContextPath () +
"/LoginHere.html"));
      } // if
    } catch (Exception e) {
```

```
      category.error ("", e);
      throw new ServletException (e);
    } // try
  } // doGet
  ...

} // PropertiesView
```

The `serveDoc()` method does some really helpful work for you. It uses a SAX reader to parse the content of the XHTML template file. If, during the parsing process, the reader encounters a <REPEAT> tag, all XHTML content is recorded until the </REPEAT> tag is found. In this case, the recorded content represents a single line in the table:

```
        <TR ID="Row" CLASS="odd">
          <TH><P CLASS="name"><REPLACE ID="Name">Driver</REPLACE></P></TH>
          <TD> </TD>
          <TD><P CLASS="value"><REPLACE
ID="Value">COM.ibm.db2.jdbc.net.DB2Driver</REPLACE></P></TD>
        </TR>
```

Next, the `getElementRepeatCount()` callback of the `PropertiesViewHandler` is invoked. This gives you a chance to indicate the number of rows in the final table. The code returns the number of properties:

```
  private static class PropertiesViewHandler extends HamletHandler {

    ...

    public int getElementRepeatCount (String id, String name, Attributes
atts) throws Exception {
      return list.size ();
    } // getElementRepeatCount
    ...

  } // PropertiesViewHandler
```

The recorded content (representing a single line in the table) plays back (that is, repeats) `list.size()` times. During this process, the `getElementReplacement()` callback in the `PropertiesViewHandler` class is invoked, giving you the opportunity to replace the dummy content (`Driver`, `COM.ibm.db2.jdbc.net.DB2Driver`) with the actual content: the name and value of the properties. Note that `getElementReplacement()` also increments the list index that was initially set to zero. The `PropertiesViewHandler` constructor sorts the keys of the context properties to display them in alphabetical order. You can do this with a call to `Collections.sort()`.

```
  private static class PropertiesViewHandler extends HamletHandler {

    ...

    public PropertiesViewHandler (Hamlet hamlet, ContextProperties props) {
      super (hamlet);
      this.props = props;
      // sort list
      list = new ArrayList (props.keySet ());
      Collections.sort (list);
    } // PropertiesViewHandler
```

```
    public String getElementReplacement (String id, String name, Attributes
atts) throws Exception {
      if (id.equals ("Name")) {
        key = (String) list.get (index++);
        return key;
      } else if (id.equals ("Value")) {
        return props.getProperty (key);
      } // if
      return "?";
    } // getElementReplacement
    ...

  } // PropertiesViewHandler
```

This example makes use of a third callback function: `getElementAttributes()`. The `getElementAttributes()` callback is a technique to add, remove, or change the attributes of elements in the XHTML code. An `ID` attribute is used to mark the elements that need to have their attributes processed. During the parsing of the XHTML template file, the SAX reader calls the `handler`'s `getElementAttributes()` method for each tag with an `ID` attribute. The tag's `ID` attribute selects the appropriate processing in `getElementAttributes()`.

The table rows are colored. This is done with the help of the Cascading Style Sheet (View.css) referenced at the top of the XHTML template file.

```
<LINK REL="stylesheet" TYPE="text/css" HREF="View.css" />
```

For each table row, you need to specify the display style class by adding a `CLASS` attribute to the <TR> tag. The single table row in the PropertiesViewTemplate.html file contains the `CLASS` attribute with the (default) value `odd`.

```
<TR ID="Row" CLASS="odd">
```

The `PropertiesViewHandler` uses the `getElementAttributes()` callback to overwrite this value for each generated row with the appropriate value at runtime. Rows with an odd index will contain the `CLASS="odd"` attribute, and rows with an even index will contain the `CLASS="even"` attribute. The helper function `Helpers.getAttributes()` facilitates this process. It takes a set of attributes and overwrites the value of one particular attribute.

```
  private static class PropertiesViewHandler extends HamletHandler {

    private static final  String  row[] = { "odd", "even" };

    ...
    public Attributes getElementAttributes (String id, String name,
Attributes atts) throws Exception {
      if (id.equals ("Row"))
        atts = Helpers.getAttributes (atts, "CLASS", "" + row[index % 2]);
      return atts;
    } // getElementAttributes
    ...
```

```
   } // PropertiesViewHandler
```

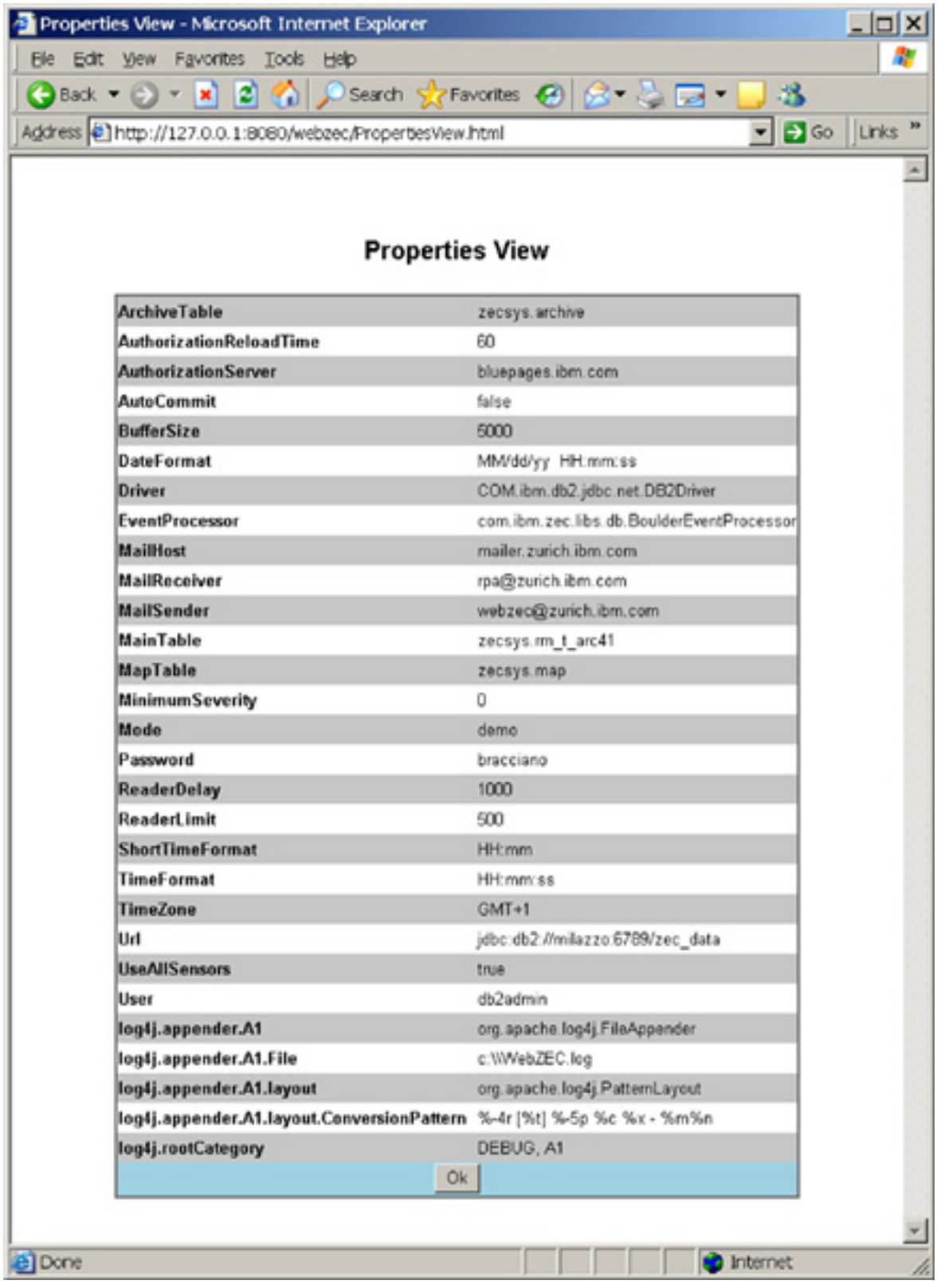
Note that you can also use `Helpers.getAttributes(atts, aNewAttribute, aAttributeValue)` to add a new attribute, and `Helpers.getAttributes(atts, aAttribute, null)` to remove an attribute.

The PropertiesView page also contains an **OK** button. When a user clicks this button, the `doPost()` method is called and the user is redirected to the current view (if there is an existing HTTP session).

```
public class PropertiesView extends Hamlet {

  ...
  public void doPost (HttpServletRequest req, HttpServletResponse res) throws
ServletException {
    try {
      category.debug ("doPost");
      HttpSession session = req.getSession (false);
      if (session != null) {
        String currentView = (String) session.getAttribute
(WebApp.currentView);
        res.sendRedirect (res.encodeURL (req.getContextPath () + "/" +
currentView));
      } else {
        res.sendRedirect (res.encodeURL (req.getContextPath () +
"/LoginHere.html"));
      } // if
    } catch (Exception e) {
      category.error ("", e);
      throw new ServletException (e);
    } // try
  } // doPost
  ...

} // PropertiesView
```

And that's all there is to the PropertiesView example. At runtime, the PropertiesView Hamlet will render the page as in Figure 6.

Figure 6. View of the PropertiesView example



View the complete PropertiesView Hamlet code

Here is the complete code for the PropertiesView Hamlet:

```
/**
 *
 * © Copyright International Business Machines Corporation 2004, 2006.
 * All rights reserved.
 *
 * The 'PropertiesView' servlet provides the properties view.
 *
 * File     : PropertiesView.java
 * Created  : 2004/09/29
 *
 * @author    Rene Pawlitzek  (rpa@zurich.ibm.com)
 * @version   1.00, 2006/11/28
 * @since     JDK 1.3
 *
 */


package com.ibm.webzec.apps.server;


import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.ibm.hamlet.*;
import com.ibm.hamlet.helpers.*;
import com.ibm.webzec.libs.util.*;
import org.apache.log4j.*;
import org.xml.sax.*;


public class PropertiesView extends Hamlet {


  // log4j
  private static Category category = Category.getInstance
(PropertiesView.class.getName ());


  private static class PropertiesViewHandler extends HamletHandler {


    private static final  String  row[] = { "odd", "even" };


    private  int               index = 0;
    private  List              list;
    private  String            key;
    private  ContextProperties  props;


    public PropertiesViewHandler (Hamlet hamlet, ContextProperties props) {
      super (hamlet);
      this.props = props;
      // sort list
      list = new ArrayList (props.keySet ());
      Collections.sort (list);
    } // PropertiesViewHandler


    public int getElementRepeatCount (String id, String name, Attributes
atts) throws Exception {
      return list.size ();
    } // getElementRepeatCount


    public String getElementReplacement (String id, String name, Attributes
atts) throws Exception {
      if (id.equals ("Name")) {
        key = (String) list.get (index++);
        return key;
      } else if (id.equals ("Value")) {
        return props.getProperty (key);
      } // if
```

```
      return "?";
    } // getElementReplacement


    public Attributes getElementAttributes (String id, String name,
Attributes atts) throws Exception {
      if (id.equals ("Row"))
        atts = Helpers.getAttributes (atts, "CLASS", "" + row[index % 2]);
      return atts;
    } // getElementAttributes


  } // PropertiesViewHandler



  public void init () throws ServletException {
    // get properties
    ContextProperties props = ContextProperties.getProperties (this);
    // configure logging
    Utilities.configLog (props);
    category.debug ("init");
  } // init


  public void doPost (HttpServletRequest req, HttpServletResponse res) throws
ServletException {
    try {
      category.debug ("doPost");
      HttpSession session = req.getSession (false);
      if (session != null) {
        String currentView = (String) session.getAttribute
(WebApp.currentView);
        res.sendRedirect (res.encodeURL (req.getContextPath () + "/" +
currentView));
      } else {
        res.sendRedirect (res.encodeURL (req.getContextPath () +
"/LoginHere.html"));
      } // if
    } catch (Exception e) {
      category.error ("", e);
      throw new ServletException (e);
    } // try
  } // doPost


  public void doGet (HttpServletRequest req, HttpServletResponse res) throws
ServletException {
    try {
      category.debug ("doGet");
      HttpSession session = req.getSession (false);
      if (session != null) {
        // get properties
        ContextProperties props = ContextProperties.getProperties (this);
        // serve document
        HamletHandler handler = new PropertiesViewHandler (this, props);
        serveDoc (req, res, "PropertiesViewTemplate.html", handler);
      } else {
        res.sendRedirect (res.encodeURL (req.getContextPath () +
"/LoginHere.html"));
      } // if
    } catch (Exception e) {
      category.error ("", e);
      throw new ServletException (e);
    } // try
  } // doGet


  public String getServletInfo () {
    category.debug ("getServletInfo");
    return "PropertiesView servlet";
  } // getServletInfo
```

```
  public void destroy () {
    category.debug ("destroy");
  } // destroy

} // PropertiesView

/* ----- End of File ----- */
```

Section 5. ListViewList example

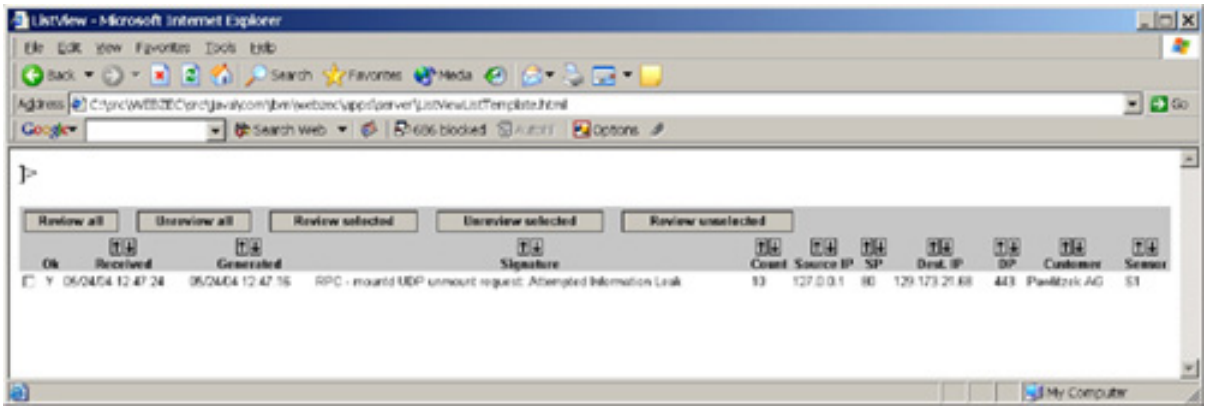
Introduction to the ListViewList example

In this section, you'll look at a slightly more complex Hamlet that populates a table with Intrusion Detection events. In addition, the table provides functionality to sort each column.

Create XHTML code for the ListViewList example

Figure 7 is a view of the XHTML template (ListViewListTemplate.html).

Figure 7. View of the XHTML template (ListViewListTemplate.html)



The page is constructed with the following XHTML code:

```
<!DOCTYPE WebZEC [ <!ENTITY nbsp " "> ]>
<HTML>
  <HEAD>
    <TITLE>ListView</TITLE>
    <LINK REL="stylesheet" TYPE="text/css" HREF="View.css" />
  </HEAD>
  <BODY>
    <FORM METHOD="POST" ACTION="ListView.html" TARGET="APPLICATION">
    <TABLE CLASS="list" CELLSPACING="0" CELLPADDING="5" ALIGN="CENTER">
      <TD>
        <P CLASS="Header">
          <INPUT CLASS="flatbutton" TYPE="SUBMIT" NAME="Op"
```

```
            VALUE="Review all" />   
          <INPUT CLASS="flatbutton" TYPE="SUBMIT" NAME="Op"
            VALUE="Unreview all" />   
          <INPUT CLASS="flatbutton" TYPE="SUBMIT" NAME="Op"
            VALUE="Review selected" />   
          <INPUT CLASS="flatbutton" TYPE="SUBMIT" NAME="Op"
            VALUE="Unreview selected" />   
          <INPUT CLASS="flatbutton" TYPE="SUBMIT" NAME="Op"
            VALUE="Review unselected" />
        </P>
      </TD>
    </TABLE>
    <TABLE CLASS="list" CELLSPACING="0" CELLPADDING="0">
      <TR>
        <TH></TH>
        <TH></TH>
        <TH><P CLASS="header">
          <A HREF="ListViewList.html?SortColumn=1" TARGET="LIST">
          <IMG CLASS="icon" SRC="up.gif" /></A>
          <A HREF="ListViewList.html?SortColumn=-1" TARGET="LIST">
          <IMG CLASS="icon" SRC="down.gif" /></A>
          </P></TH>
        <TH><P CLASS="header">
          <A HREF="ListViewList.html?SortColumn=2" TARGET="LIST">
          <IMG CLASS="icon" SRC="up.gif" /></A>
          <A HREF="ListViewList.html?SortColumn=-2" TARGET="LIST">
          <IMG CLASS="icon" SRC="down.gif" /></A>
          </P></TH>
        <TH><P CLASS="header">
          <A HREF="ListViewList.html?SortColumn=3" TARGET="LIST">
          <IMG CLASS="icon" SRC="up.gif" /></A>
          <A HREF="ListViewList.html?SortColumn=-3" TARGET="LIST">
          <IMG CLASS="icon" SRC="down.gif" /></A>
          </P></TH>
        <TH><P CLASS="header">
          <A HREF="ListViewList.html?SortColumn=4" TARGET="LIST">
          <IMG CLASS="icon" SRC="up.gif" /></A>
          <A HREF="ListViewList.html?SortColumn=-4" TARGET="LIST">
          <IMG CLASS="icon" SRC="down.gif" /></A>
          </P></TH>
        <TH><P CLASS="header">
          <A HREF="ListViewList.html?SortColumn=5" TARGET="LIST">
          <IMG CLASS="icon" SRC="up.gif" /></A>
          <A HREF="ListViewList.html?SortColumn=-5" TARGET="LIST">
          <IMG CLASS="icon" SRC="down.gif" /></A>
          </P></TH>
        <TH><P CLASS="header">
          <A HREF="ListViewList.html?SortColumn=6" TARGET="LIST">
          <IMG CLASS="icon" SRC="up.gif" /></A>
          <A HREF="ListViewList.html?SortColumn=-6" TARGET="LIST">
          <IMG CLASS="icon" SRC="down.gif" /></A>
          </P></TH>
        <TH><P CLASS="header">
          <A HREF="ListViewList.html?SortColumn=7" TARGET="LIST">
          <IMG CLASS="icon" SRC="up.gif" /></A>
          <A HREF="ListViewList.html?SortColumn=-7" TARGET="LIST">
          <IMG CLASS="icon" SRC="down.gif" /></A>
          </P></TH>
        <TH><P CLASS="header">
          <A HREF="ListViewList.html?SortColumn=8" TARGET="LIST">
          <IMG CLASS="icon" SRC="up.gif" /></A>
          <A HREF="ListViewList.html?SortColumn=-8" TARGET="LIST">
          <IMG CLASS="icon" SRC="down.gif" /></A>
          </P></TH>
        <TH><P CLASS="header">
          <A HREF="ListViewList.html?SortColumn=9" TARGET="LIST">
          <IMG CLASS="icon" SRC="up.gif" /></A>
          <A HREF="ListViewList.html?SortColumn=-9" TARGET="LIST">
          <IMG CLASS="icon" SRC="down.gif" /></A>
          </P></TH>
        <TH><P CLASS="header">
          <A HREF="ListViewList.html?SortColumn=10" TARGET="LIST">
          <IMG CLASS="icon" SRC="up.gif" /></A>
          <A HREF="ListViewList.html?SortColumn=-10" TARGET="LIST">
```

```
          <IMG CLASS="icon" SRC="down.gif" /></A>
          </P></TH>
      </TR>
      <TR>
        <TD></TD>
        <TD><P CLASS="header">Ok</P></TD>
        <TD><P CLASS="header">Received</P></TD>
        <TD><P CLASS="header">Generated</P></TD>
        <TD><P CLASS="header">Signature</P></TD>
        <TD><P CLASS="header">Count</P></TD>
        <TD><P CLASS="header">Source IP</P></TD>
        <TD><P CLASS="header">SP</P></TD>
        <TD><P CLASS="header">Dest. IP</P></TD>
        <TD><P CLASS="header">DP</P></TD>
        <TD><P CLASS="header">Customer</P></TD>
        <TD><P CLASS="header">Sensor</P></TD>
      </TR>
      <REPEAT ID="rows">
      <TR ID="Color" BGCOLOR="#FFFFFF">
        <TD><INPUT ID="Index" CLASS="checkbox" TYPE="CHECKBOX" NAME="Index"
          VALUE="0" /></TD>
        <TD><P CLASS="row"> 
          <REPLACE ID="Reviewed">Y</REPLACE> </P></TD>
        <TD><P CLASS="row"> 
          <REPLACE ID="Reception">05/24/04 12:47:24</REPLACE> </P></TD>
        <TD><P CLASS="row"> 
          <REPLACE ID="Generation">05/24/04 12:47:16</REPLACE> </P></TD>
        <TD><P CLASS="row"> 
          <A ID="Link" HREF="EventView.html?Index={0}" CLASS="dynamic"
            TARGET="APPLICATION">
          <REPLACE ID="Signature">RPC - mountd UDP unmount request:
            Attempted Information Leak</REPLACE></A> </P></TD>
        <TD><P CLASS="row"> 
          <REPLACE ID="Count">10</REPLACE> </P></TD>
        <TD><P CLASS="row"> 
          <REPLACE ID="SrcIP">127.0.0.1</REPLACE> </P></TD>
        <TD><P CLASS="row"> 
          <REPLACE ID="SrcPort">80</REPLACE> </P></TD>
        <TD><P CLASS="row"> 
          <REPLACE ID="DstIP">129.173.21.68</REPLACE> </P></TD>
        <TD><P CLASS="row"> 
          <REPLACE ID="DstPort">44</REPLACE>3 </P></TD>
        <TD><P CLASS="row"> 
          <REPLACE ID="Customer">Pawlitzek AG</REPLACE> </P></TD>
        <TD><P CLASS="row"> 
          <REPLACE ID="Sensor">S1</REPLACE> </P></TD>
      </TR>
      </REPEAT>
    </TABLE>
    </FORM>
  </BODY>
</HTML>
```

The code reveals that the table consists of a single row with dummy values. The row itself contains several <REPLACE> tags and is surrounded by the <REPEAT>tag.

Write the Hamlet for the ListViewList example

Next, you need to create a public class, `ListViewList`, that extends `Hamlet`. The `init()` method configures logging.

```
public class ListViewList extends Hamlet {

  ...
  public void init () throws ServletException {
    // get properties
```

```
    ContextProperties props = ContextProperties.getProperties (this);
    // configure logging
    Utilities.configLog (props);
    category.debug ("init");
  } // init
  ...

} // ListViewList
```

The `doGet()` method (which is called when a user request is received) retrieves the date format from the context properties and the sort column from the request object, along with the display options and the vector containing the events from the session object. Next, the Hamlet code passes these objects to the `ListViewListHandler` constructor to create an instance (`handler`). As you saw earlier, the `handler` provides the dynamic content for the template at runtime. Then `doGet()` passes the name of the XHTML template file (ListViewListTemplate.html) and the `handler` to the Hamlet's `serveDoc()` method. Note that an HTTP session is required for all this to happen. This session is created in the `doGet()` method of the Login Hamlet with the `getSession()` statement when the user logs on successfully.

```
public class ListViewList extends Hamlet {

  ...
  public void doGet (HttpServletRequest req, HttpServletResponse res) throws
ServletException {
    try {
      category.debug ("doGet");
      HttpSession session = req.getSession (false);
      if (session != null) {
        // get properties
        ContextProperties props = ContextProperties.getProperties (this);
        // get formats
        String format = props.getStringProperty ("DateFormat");
        String timeZone = props.getStringProperty ("TimeZone");
        category.debug ("date format: " + format);
        category.debug ("time zone: " + timeZone);
        SimpleDateFormat dateFormat = new SimpleDateFormat (format);
        dateFormat.setTimeZone (Utilities.getTimeZone (timeZone));
        // get parameter
        int sortColumn = 0;
        try {
          sortColumn = Integer.parseInt (req.getParameter ("SortColumn"));
        } catch (Exception e) {
        } // try
        // get display options
        Boolean b;
        b = (Boolean) session.getAttribute (OptionView.showReviewedEvents);
        boolean showReviewedEvents = b.booleanValue ();
        b = (Boolean) session.getAttribute (OptionView.showNewEventsOnTop);
        boolean showNewEventsOnTop = b.booleanValue ();
        // retrieve events vector
        Vector events = (Vector) session.getAttribute (WebApp.events);
        // serve document
        HamletHandler handler = new ListViewListHandler (this, events,
sortColumn,
          showReviewedEvents, showNewEventsOnTop, dateFormat);
        serveDoc (req, res, "ListViewListTemplate.html", handler);
      } else {
        res.sendRedirect (res.encodeURL (req.getContextPath () +
"/LoginHere.html"));
      } // if
    } catch (Exception e) {
      category.debug ("", e);
      throw new ServletException (e);
    } // try
  } // doGet
  ...
```

```
} // ListViewList
```

Things get more interesting now. With the help of a SAX reader, the `serveDoc()` method parses the content of the XHTML template file. If the reader comes across the <REPEAT> tag, all the ensuing XHTML content is recorded. When the corresponding </REPEAT> tag is encountered, the recording stops and the `handler`'s `getElementRepeatCount()` callback is invoked. The code in the `getElementRepeatCount()` method returns the number of events in the event vector.

```
  private static class ListViewListHandler extends HamletHandler implements
Comparator {

    ...
    public int getElementRepeatCount (String id, String name, Attributes
atts) throws Exception {
      return size;
    } // getElementRepeatCount
    ...

  } // ListViewListHandler
```

You don't need to check the value of the `ID` attribute (stored in the `id` parameter), because only one section in the XHTML template file was marked with the <REPEAT> tag. If you had marked several sections, you would need a way to distinguish them. To do so, you add an `ID` attribute with a distinct value to the <REPEAT> tags (for example, <REPEAT `ID="rows"`>).

After `getElementRepeatCount()` returns the number of repeats, the recorded content is played back (repeated) `event.size()` times. During this playback phase, the `getElementReplacement()` callback is invoked, giving you a chance to provide the details (source IP, destination IP, signature, and so on) for each displayed event.

```
  private static class ListViewListHandler extends HamletHandler implements
Comparator {

    ...
    public String getElementReplacement (String id, String name, Attributes
atts) throws Exception {
      if (id.equals ("Reviewed")) {
        return event.isReviewed () ? "Y" : "-";
      } else if (id.equals ("Reception")) {
        return dateFormat.format (event.getReceptionDate ());
      } else if (id.equals ("Generation")) {
        return dateFormat.format (event.getGenerationDate ());
      } else if (id.equals ("Signature")) {
        return event.getSignature ();
      } else if (id.equals ("Count")) {
        return "" + event.getEventCount ();
      } else if (id.equals ("SrcIP")) {
        return event.getSrcIP ();
      } else if (id.equals ("SrcPort")) {
        return "" + event.getSrcPort ();
      } else if (id.equals ("DstIP")) {
        return event.getDstIP ();
      } else if (id.equals ("DstPort")) {
        return "" + event.getDstPort ();
      } else if (id.equals ("Customer")) {
```

```
        return event.getCustomerName ();
      } else if (id.equals ("Sensor")) {
        return event.getSensorName ();
      } // if
      return "?";
    } // getElementReplacement
    ...

  } // ListViewListHandler
```

You can use the `getElementAttributes()` callback to implement a number of features. You'll want to color the background of each row depending on the event's severity: fatal and critical events are displayed with a red background, warning events with a yellow background, and harmless events with a green background. In the XHTML code, the <TR> tag contains an `ID` attribute with the value `Color` and a `BGCOLOR` attribute with the value `#FFFFFF` (white).

```
      <REPEAT>
      <TR ID="Color" BGCOLOR="#FFFFFF">
      ...
      </TR>
      </REPEAT>
```

The `getElementAttributes()` method gives you a chance to change the value of the `BGCOLOR` attribute at runtime. To change the value of the `BGCOLOR` attribute in the `<TR ID="Color" BGCOLOR="#FFFFFF">` tag, the `ID` attribute (stored in the `id` parameter) is checked in the `getElementAttributes()` callback. If the value of the `id` parameter is `Color`, `event.getColor()` is invoked to get the color that corresponds to the event's severity. Next, `Helpers.getAttributes()` overwrites the value of the `BGCOLOR` attribute with the new RGB color code and returns the modified attribute set to the Hamlet framework.

```
  private static class ListViewListHandler extends HamletHandler implements
Comparator {

    ...
    public Attributes getElementAttributes (String id, String name,
Attributes atts) throws Exception {
      if (id.equals ("Color")) {
        // get the event descriptor
        int i = getIndex (index, size, showNewEventsOnTop);
        event = (EventDesc) events.elementAt (i);
        Color color = event.getColor ();
        int rgb = color.getRGB ();
        String col = Integer.toHexString (rgb);
        atts = Helpers.getAttributes (atts, "BGCOLOR", "#" + col.substring
(2));
      } else if (id.equals ("Index")) {
        ...
      } else if (id.equals ("Link")) {
        ...
      } // if
      return atts;
    } // getElementAttributes
    ...

  } // ListViewListHandler
```

To get more detailed information about an event, a user can click on the signature

field of that event. Obviously, the table must contain links to bring up the appropriate EventView.html page. (You'll see this page in the EventView example, which we'll discuss next.) The `getElementAttributes()` callback is used to create these links. In the XHTML template file, you can find the <A> tag with the HREF attribute.

```
      <REPEAT>
      <TR ID="Color" BGCOLOR="#FFFFFF">
        ...
        <TD>
          <P CLASS="row">&nsbsp;;
          <A ID="Link" HREF="EventView.html?Index={0}" CLASS="dynamic"
            TARGET="APPLICATION">
          <REPLACE ID="Signature">
            RPC - mountd UDP unmount request:
            Attempted Information Leak
          </REPLACE>
          </A> 
          </P>
        </TD>
        ...
      </TR>
      </REPEAT>
```

The value of the HREF attribute (`EventView.html?Index={0}`) is a template for the links and must be filled in for each event with the actual value -- the index of the event, in this case. This is done with the help of the `MessageFormat` class from the `java.text` package and the `Helpers.getAttributes()` method. The EventView.html page needs to know this index in order to display the correct event details.

```
  private static class ListViewListHandler extends HamletHandler implements
Comparator {

    ...
    public Attributes getElementAttributes (String id, String name,
Attributes atts) throws Exception {
      if (id.equals ("Color")) {
        ...
      } else if (id.equals ("Index")) {
        ...
      } else if (id.equals ("Link")) {
        String format = atts.getValue ("HREF");
        String tmp = "" + getIndex (index, size, showNewEventsOnTop);
        Object[] arg = { tmp };
        tmp = MessageFormat.format (format, arg);
        atts = Helpers.getAttributes (atts, "HREF", tmp);
        // increment index
        index++;
      } // if
      return atts;
    } // getElementAttributes
    ...

  } // ListViewListHandler
```

Each row in the table starts with a checkbox. A user can mark events to make a selection once the table is displayed in the browser window.

```
      <REPEAT>
      <TR ID="Color" BGCOLOR="#FFFFFF">
        <TD>
```

```
          <INPUT ID="Index" CLASS="checkbox" TYPE="CHECKBOX" NAME="Index"
VALUE="0" />
        </TD>
        ...
      </TR>
      </REPEAT>
```

When the user clicks one of the buttons at the top of the screen (**Review All,** for example), the selection is transmitted to the server for processing. On the server side, you need to know the index of each selected event. You can use the `getElementAttributes()` callback to provide this information. The code in the `getElementAttributes()` method overwrites the value of the `VALUE` attribute in the `<INPUT ID="Index">` tag with the index of the event using `Helpers.getAttributes()`. The value `0` is just a placeholder at design time.

```
  private static class ListViewListHandler extends HamletHandler implements
Comparator {

    ...
    public Attributes getElementAttributes (String id, String name,
Attributes atts) throws Exception {
      if (id.equals ("Color")) {
        ...
      } else if (id.equals ("Index")) {
        int i = getIndex (index, size, showNewEventsOnTop);
        atts = Helpers.getAttributes (atts, "VALUE", "" + i);
      } else if (id.equals ("Link")) {
        ...
      } // if
      return atts;
    } // getElementAttributes
    ...

  } // ListViewListHandler
```

The processing of the selected events is performed in a separate ListView Hamlet (not discussed in this tutorial) as specified in the `<FORM METHOD="POST" ACTION="ListView.html" TARGET="APPLICATION">` tag. It is separated from the rendering functionality (the ListViewList Hamlet) in order to implement the MVC design pattern.

Sorting works as follows: When a user clicks one of the arrow buttons, the ListViewList Hamlet is requested (for example, http://ListViewList.html?SortColumn=3) with a sort column parameter. The invoked `doGet()` method retrieves the vector containing the events from the session object and passes it to the `ListViewListHandler` constructor. After obtaining a `Collator` object for sorting in alphabetical order, the constructor calls `processEvents()` to process the event vector using one of the display options (`showReviewedEvents`) and the sort column. `processEvents()` removes the reviewed events if the user only wishes to display the non-reviewed events. Next, it sorts the event vector by calling `Collections.sort()` if the sort column is not zero.

```
  private static class ListViewListHandler extends HamletHandler implements
Comparator {
```

```
    ...
    private void processEvents (Vector events, boolean showReviewedEvents,
int sortColumn) {
      if (!showReviewedEvents) {
        // hide reviewed events
        for (int i = events.size() - 1; i >= 0; i--) {
          EventDesc event = (EventDesc) events.elementAt (i);
          if (event.isReviewed ())
            events.remove (i);
        } // for
      } // if
      // sort events
      category.debug ("Sort column: " + sortColumn);
      if (sortColumn != 0)
        Collections.sort (events, this);
    } // processEvents


    public ListViewListHandler (Hamlet hamlet, Vector events, int sortColumn,
      boolean showReviewedEvents, boolean showNewEventsOnTop,
      SimpleDateFormat dateFormat) {

      super (hamlet);
      this.events = events;
      this.sortColumn = sortColumn;
      this.showNewEventsOnTop = showNewEventsOnTop;
      this.dateFormat = dateFormat;
      if (events != null) {
        // get collator for sorting
        collate = Collator.getInstance ();
        processEvents (events, showReviewedEvents, sortColumn);
        size = events.size ();
      } // if
    } // ListViewListHandler
    ...


  } // ListViewListHandler
```

Sorting a collection requires the `Comparator` interface. The `ListViewListHandler` class extends `HamletHandler` and implements the `Comparator` interface, which consists of a single method called `compare()`. The code in `processEvents()` passes the instance implementing the `Comparator` interface (`this`) to the sort function. The sort function calls the `compare()` method of the `Comparator` interface to find out how to order the objects in the collection.

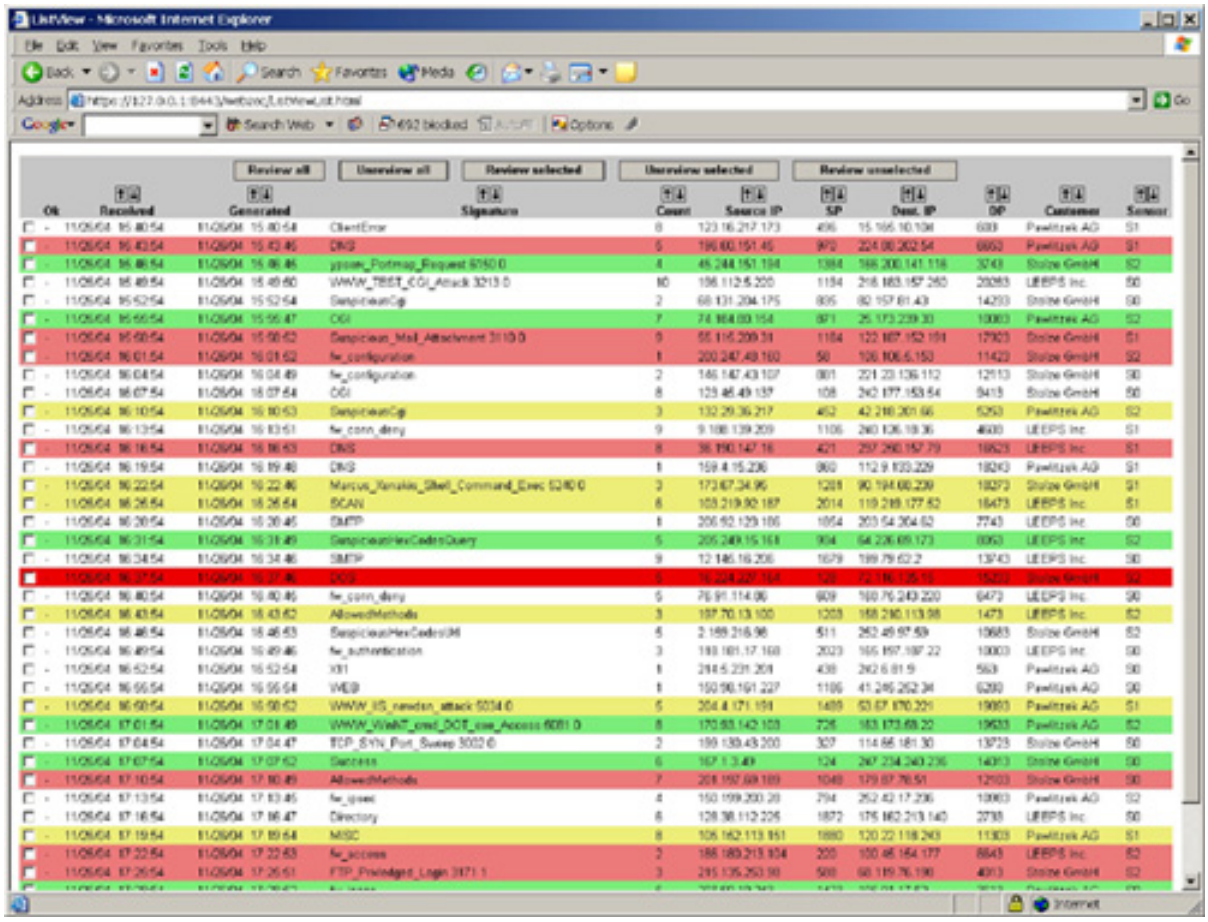
```
  private static class ListViewListHandler extends HamletHandler implements
Comparator {


    ...

    /* ----- implementation of Comparator ----- */


    public int compare (Object obj1, Object obj2) {
      EventDesc e1 = (EventDesc) obj1;
      EventDesc e2 = (EventDesc) obj2;
      int c = 0;
      switch (Math.abs (sortColumn)) {
        case 0:  // reviewed
          break;
        case 1:  // reception time
          c = compareLong (e1.getReceptionTime (), e2.getReceptionTime ());
          break;
        case 2:  // generation time
          c = compareLong (e1.getGenerationTime (), e2.getGenerationTime ());
          break;
```

```
        case 3:  // signature
          c = collate.compare (e1.getSignature (), e2.getSignature ());
          break;
        case 4:  // total events
          c = compareInt ((1 + e1.getRepeatCount ()), (1 + e2.getRepeatCount
()));
          break;
        case 5:  // source ip
          c = collate.compare (e1.getSrcIP (), e2.getSrcIP ());
          break;
        case 6:  // source port
          c = compareInt (e1.getSrcPort (), e2.getSrcPort ());
          break;
        case 7:  // destination ip
          c = collate.compare (e1.getDstIP (), e2.getDstIP ());
          break;
        case 8:  // destination port
          c = compareInt (e1.getDstPort (), e2.getDstPort ());
          break;
        case 9:  // customer name
          c = collate.compare (e1.getCustomerName (), e2.getCustomerName ());
          break;
        case 10:  // sensor name
          c = collate.compare (e1.getSensorName (), e2.getSensorName ());
          break;
      } // switch
      if (sortColumn > 0)
        return -c;
      else
        return c;
    } // compare


  } // ListViewListHandler
```

The `compare()` function specifies the sorting order for each event attribute. It uses the `collate` object to obtain the sorting order for strings. The `collate` object adheres to the specified rules of each locale.

Now that you understand how the ListViewList Hamlet works, take a look at Figure 8 to see it in action.

Figure 8. View of the ListViewList Hamlet in action



View the complete ListViewList Hamlet code

Here is the complete code for the ListViewList example:

```
/**
 *
 * © Copyright International Business Machines Corporation 2004, 2006.
 * All rights reserved.
 *
 * The 'ListViewList' servlet provides the list view.
 *
 * File     : ListViewList.java
 * Created  : 2004/03/18
 *
 * @author    Rene Pawlitzek  (rpa@zurich.ibm.com)
 * @version   1.00, 2006/11/28
 * @since     JDK 1.3
 *
 */


package com.ibm.webzec.apps.server;


import java.awt.*;
import java.util.*;
import java.text.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.ibm.hamlet.*;
import com.ibm.hamlet.helpers.*;
```

```
import com.ibm.webzec.libs.db.*;
import com.ibm.webzec.libs.util.*;
import org.apache.log4j.*;
import org.xml.sax.*;


public class ListViewList extends Hamlet {


  // log4j
  private static Category category = Category.getInstance
(ListViewList.class.getName ());


  private static class ListViewListHandler extends HamletHandler implements
Comparator {


    private  int               size = 0, index = 0, sortColumn;
    private  boolean           showNewEventsOnTop;
    private  Vector            events;
    private  Collator          collate;
    private  EventDesc         event;
    private  SimpleDateFormat  dateFormat;


    private static int getIndex (int index, int size, boolean
showNewEventsOnTop) {
      if (!showNewEventsOnTop)
        return index;
      else
        return size - 1 - index;
  } // getIndex


    private void processEvents (Vector events, boolean showReviewedEvents,
int sortColumn) {
      if (!showReviewedEvents) {
        // hide reviewed events
        for (int i = events.size() - 1; i >= 0; i--) {
          EventDesc event = (EventDesc) events.elementAt (i);
          if (event.isReviewed ())
            events.remove (i);
        } // for
      } // if
      // sort events
      category.debug ("Sort column: " + sortColumn);
      if (sortColumn != 0)
        Collections.sort (events, this);
    } // processEvents


    public ListViewListHandler (Hamlet hamlet, Vector events, int sortColumn,
      boolean showReviewedEvents, boolean showNewEventsOnTop,
      SimpleDateFormat dateFormat) {

      super (hamlet);
      this.events = events;
      this.sortColumn = sortColumn;
      this.showNewEventsOnTop = showNewEventsOnTop;
      this.dateFormat = dateFormat;
      if (events != null) {
        // get collator for sorting
        collate = Collator.getInstance ();
        processEvents (events, showReviewedEvents, sortColumn);
        size = events.size ();
      } // if

    } // ListViewListHandler


    public int getElementRepeatCount (String id, String name, Attributes
atts) throws Exception {
      return size;
```

```
    } // getElementRepeatCount


    public String getElementReplacement (String id, String name, Attributes
atts) throws Exception {
      if (id.equals ("Reviewed")) {
        return event.isReviewed () ? "Y" : "-";
      } else if (id.equals ("Reception")) {
        return dateFormat.format (event.getReceptionDate ());
      } else if (id.equals ("Generation")) {
        return dateFormat.format (event.getGenerationDate ());
      } else if (id.equals ("Signature")) {
        return event.getSignature ();
      } else if (id.equals ("Count")) {
        return "" + event.getEventCount ();
      } else if (id.equals ("SrcIP")) {
        return event.getSrcIP ();
      } else if (id.equals ("SrcPort")) {
        return "" + event.getSrcPort ();
      } else if (id.equals ("DstIP")) {
        return event.getDstIP ();
      } else if (id.equals ("DstPort")) {
        return "" + event.getDstPort ();
      } else if (id.equals ("Customer")) {
        return event.getCustomerName ();
      } else if (id.equals ("Sensor")) {
        return event.getSensorName ();
      } // if
      return "?";
    } // getElementReplacement


    public Attributes getElementAttributes (String id, String name,
Attributes atts) throws Exception {
      if (id.equals ("Color")) {
        // get the event descriptor
        int i = getIndex (index, size, showNewEventsOnTop);
        event = (EventDesc) events.elementAt (i);
        Color color = event.getColor ();
        int rgb = color.getRGB ();
        String col = Integer.toHexString (rgb);
        atts = Helpers.getAttributes (atts, "BGCOLOR", "#" + col.substring
(2));
      } else if (id.equals ("Index")) {
        int i = getIndex (index, size, showNewEventsOnTop);
        atts = Helpers.getAttributes (atts, "VALUE", "" + i);
      } else if (id.equals ("Link")) {
        String format = atts.getValue ("HREF");
        String tmp = "" + getIndex (index, size, showNewEventsOnTop);
        Object[] arg = { tmp };
        tmp = MessageFormat.format (format, arg);
        atts = Helpers.getAttributes (atts, "HREF", tmp);
        // increment index
        index++;
      } // if
      return atts;
    } // getElementAttributes


    /* ----- implementation of Comparator ----- */


    private int compareInt (int a, int b) {
      if (a > b)
        return 1;
      else if (a < b)
        return -1;
      else
        return 0;
    } // compareInt


    private int compareLong (long a, long b) {
      if (a > b)
```

```
        return 1;
      else if (a < b)
        return -1;
      else
        return 0;
    } // compareLong


    public int compare (Object obj1, Object obj2) {
      EventDesc e1 = (EventDesc) obj1;
      EventDesc e2 = (EventDesc) obj2;
      int c = 0;
      switch (Math.abs (sortColumn)) {
        case 0:  // reviewed
          break;
        case 1:  // reception time
          c = compareLong (e1.getReceptionTime (), e2.getReceptionTime ());
          break;
        case 2:  // generation time
          c = compareLong (e1.getGenerationTime (), e2.getGenerationTime ());
          break;
        case 3:  // signature
          c = collate.compare (e1.getSignature (), e2.getSignature ());
          break;
        case 4:  // total events
          c = compareInt ((1 + e1.getRepeatCount ()), (1 + e2.getRepeatCount
()));
          break;
        case 5:  // source ip
          c = collate.compare (e1.getSrcIP (), e2.getSrcIP ());
          break;
        case 6:  // source port
          c = compareInt (e1.getSrcPort (), e2.getSrcPort ());
          break;
        case 7:  // destination ip
          c = collate.compare (e1.getDstIP (), e2.getDstIP ());
          break;
        case 8:  // destination port
          c = compareInt (e1.getDstPort (), e2.getDstPort ());
          break;
        case 9:  // customer name
          c = collate.compare (e1.getCustomerName (), e2.getCustomerName ());
          break;
        case 10:  // sensor name
          c = collate.compare (e1.getSensorName (), e2.getSensorName ());
          break;
      } // switch
      if (sortColumn > 0)
        return -c;
      else
        return c;
    } // compare


  } // ListViewListHandler



  public void init () throws ServletException {
    // get properties
    ContextProperties props = ContextProperties.getProperties (this);
    // configure logging
    Utilities.configLog (props);
    category.debug ("init");
  } // init


  public void doGet (HttpServletRequest req, HttpServletResponse res) throws
ServletException {
    try {
      category.debug ("doGet");
      HttpSession session = req.getSession (false);
      if (session != null) {
        // get properties
```

```
        ContextProperties props = ContextProperties.getProperties (this);
        // get formats
        String format = props.getStringProperty ("DateFormat");
        String timeZone = props.getStringProperty ("TimeZone");
        category.debug ("date format: " + format);
        category.debug ("time zone: " + timeZone);
        SimpleDateFormat dateFormat = new SimpleDateFormat (format);
        dateFormat.setTimeZone (Utilities.getTimeZone (timeZone));
        // get parameter
        int sortColumn = 0;
        try {
          sortColumn = Integer.parseInt (req.getParameter ("SortColumn"));
        } catch (Exception e) {
        } // try
        // get display options
        Boolean b;
        b = (Boolean) session.getAttribute (OptionView.showReviewedEvents);
        boolean showReviewedEvents = b.booleanValue ();
        b = (Boolean) session.getAttribute (OptionView.showNewEventsOnTop);
        boolean showNewEventsOnTop = b.booleanValue ();
        // retrieve events vector
        Vector events = (Vector) session.getAttribute (WebApp.events);
        // serve document
        HamletHandler handler = new ListViewListHandler (this, events,
sortColumn,
          showReviewedEvents, showNewEventsOnTop, dateFormat);
        serveDoc (req, res, "ListViewListTemplate.html", handler);
      } else {
        res.sendRedirect (res.encodeURL (req.getContextPath () +
"/LoginHere.html"));
      } // if
    } catch (Exception e) {
      category.debug ("", e);
      throw new ServletException (e);
    } // try
  } // doGet


  public String getServletInfo () {
    category.debug ("getServletInfo");
    return "ListViewList servlet";
  } // getServletInfo


  public void destroy () {
    category.debug ("destroy");
  } // destroy


} // ListViewList


/* ----- End of File ----- */
```

Section 6. EventView example

Introduction to the EventView example

The tutorial continues with the EventView example. EventView displays all event-specific details. In general, this Hamlet is called from ListViewList when detailed information about an event is requested.

Create the XHTML code for EventViewTemplate

The XHTML template (EventViewTemplate.html) contains the following code:

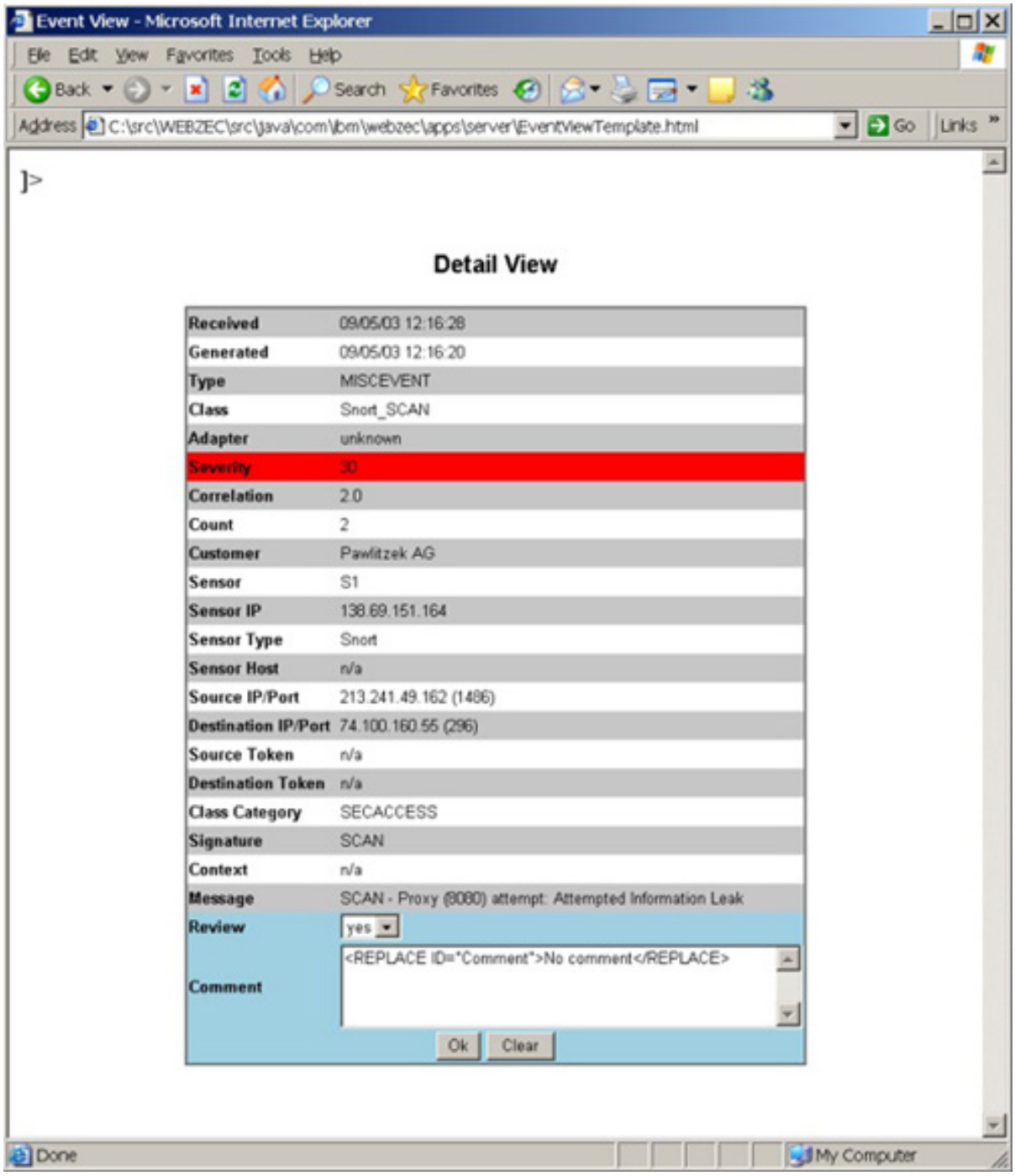
```
<!DOCTYPE WebZEC [ <!ENTITY nbsp " "> ]>
<HTML>
  <HEAD>
    <TITLE>Event View</TITLE>
    <LINK REL="stylesheet" TYPE="text/css" HREF="View.css" />
  </HEAD>
  <BODY>
    <P CLASS="title">
      <BR />Detail View
    </P>
    <FORM ACTION="EventView.html" METHOD="POST">
    <TABLE CLASS="dialog" CELLPADDING="1" CELLSPACING="0" ALIGN="CENTER">
      <TR CLASS="odd">
        <TH><P CLASS="name">Received</P></TH>
        <TD> </TD>
        <TD><P CLASS="value"><REPLACE ID="Reception">09/05/03
12:16:28</REPLACE></P></TD>
      </TR>
      <TR CLASS="even">
        <TH><P CLASS="name">Generated</P></TH>
        <TD> </TD>
        <TD><P CLASS="value"><REPLACE ID="Generation">09/05/03
12:16:20</REPLACE></P></TD>
      </TR>
        <TR CLASS="odd">
        <TH><P CLASS="name">Type</P></TH>
        <TD> </TD>
        <TD><P CLASS="value"><REPLACE ID="Type">MISCEVENT</REPLACE></P></TD>
      </TR>
      <TR CLASS="even">
        <TH><P CLASS="name">Class</P></TH>
        <TD> </TD>
        <TD><P CLASS="value"><REPLACE
ID="Class">Snort_SCAN</REPLACE></P></TD>
      </TR>
      <TR CLASS="odd">
        <TH><P CLASS="name">Adapter</P></TH>
        <TD> </TD>
        <TD><P CLASS="value"><REPLACE ID="Adapter">unknown</REPLACE></P></TD>
      </TR>
      <TR ID="Color" BGCOLOR="#FF0000">
        <TH><P CLASS="name">Severity</P></TH>
        <TD> </TD>
        <TD><P CLASS="value"><REPLACE ID="Severity">30</REPLACE></P></TD>
      </TR>
      <TR CLASS="odd">
        <TH><P CLASS="name">Correlation</P></TH>
        <TD> </TD>
        <TD><P CLASS="value"><REPLACE ID="Correlation">2.0</REPLACE></P></TD>
      </TR>
      <TR CLASS="even">
        <TH><P CLASS="name">Count</P></TH>
        <TD> </TD>
        <TD><P CLASS="value"><REPLACE ID="Count">2</REPLACE></P></TD>
      </TR>
      <TR CLASS="odd">
        <TH><P CLASS="name">Customer</P></TH>
        <TD> </TD>
        <TD><P CLASS="value"><REPLACE ID="Customer">Pawlitzek
AG</REPLACE></P></TD>
      </TR>
      <TR CLASS="even">
        <TH><P CLASS="name">Sensor</P></TH>
        <TD> </TD>
        <TD><P CLASS="value"><REPLACE ID="Sensor">S1</REPLACE></P></TD>
      </TR>
```

```
      <TR CLASS="odd">
        <TH><P CLASS="name">Sensor IP</P></TH>
        <TD> </TD>
        <TD><P CLASS="value"><REPLACE
ID="SensorIP">138.69.151.164</REPLACE></P></TD>
      </TR>
      <TR CLASS="even">
        <TH><P CLASS="name">Sensor Type</P></TH>
        <TD> </TD>
        <TD><P CLASS="value"><REPLACE
ID="SensorType">Snort</REPLACE></P></TD>
      </TR>
      <TR CLASS="odd">
        <TH><P CLASS="name">Sensor Host</P></TH>
        <TD> </TD>
        <TD><P CLASS="value"><REPLACE ID="SensorHost">n/a</REPLACE></P></TD>
      </TR>
      <TR CLASS="even">
        <TH><P CLASS="name">Source IP/Port</P></TH>
        <TD> </TD>
        <TD><P CLASS="value"><REPLACE ID="SrcIPPort">213.241.49.162
(1486)</REPLACE></P></TD>
      </TR>
      <TR CLASS="odd">
        <TH><P CLASS="name">Destination IP/Port</P></TH>
        <TD> </TD>
        <TD><P CLASS="value"><REPLACE ID="DstIPPort">74.100.160.55
(296)</REPLACE></P></TD>
      </TR>
      <TR CLASS="even">
        <TH><P CLASS="name">Source Token</P></TH>
        <TD> </TD>
        <TD><P CLASS="value"><REPLACE ID="SrcToken">n/a</REPLACE></P></TD>
      </TR>
      <TR CLASS="odd">
        <TH><P CLASS="name">Destination Token</P></TH>
        <TD> </TD>
        <TD><P CLASS="value"><REPLACE ID="DstToken">n/a</REPLACE></P></TD>
      </TR>
      <TR CLASS="even">
        <TH><P CLASS="name">Class Category</P></TH>
        <TD> </TD>
        <TD><P CLASS="value"><REPLACE
ID="Category">SECACCESS</REPLACE></P></TD>
      </TR>
      <TR CLASS="odd">
        <TH><P CLASS="name">Signature</P></TH>
        <TD> </TD>
        <TD><P CLASS="value"><REPLACE ID="Signature">SCAN</REPLACE></P></TD>
      </TR>
      <TR CLASS="even">
        <TH><P CLASS="name">Context</P></TH>
        <TD> </TD>
        <TD><P CLASS="value"><REPLACE ID="Context">n/a</REPLACE></P></TD>
      </TR>
      <TR CLASS="odd">
        <TH><P CLASS="name">Message</P></TH>
        <TD> </TD>
        <TD><P CLASS="value"><REPLACE ID="Message">SCAN - Proxy (8080)
attempt: Attempted Information Leak</REPLACE></P></TD>
      </TR>
      <TR>
        <TH><P CLASS="name">Review</P></TH>
        <TD> </TD>
        <TD>
        <SELECT CLASS="dropdown" size="1" NAME="Review">
          <OPTION>yes</OPTION>
          <OPTION>no</OPTION>
        </SELECT>
        </TD>
      </TR>
      <TR>
        <TH><P CLASS="name">Comment</P></TH>
        <TD> </TD>
```

```
        <TD>
        <TEXTAREA CLASS="textarea" COLS="60" ROWS="4" NAME="Comment"><REPLACE
ID="Comment">No comment</REPLACE></TEXTAREA>
        </TD>
      </TR>
      <TR>
        <TH COLSPAN="3">
          <P CLASS="buttons">
            <INPUT ID="Index" TYPE="Hidden" NAME="Index" VALUE="" />
            <INPUT CLASS="button" TYPE="Submit" VALUE=" Ok " />
            <INPUT CLASS="button" TYPE="Reset" VALUE=" Clear " />
          </P>
        </TH>
      </TR>
    </TABLE>
    </FORM>
  </BODY>
</HTML>
```

At design time, the template looks like Figure 9.

Figure 9. View of the XHTML template (EventViewTemplate.html)



Write the EventView Hamlet

Next, you need to create a public class named `EventView` that extends `Hamlet`. The code in the class's `init()` method configures the logging facilities.

```
public class EventView extends Hamlet {

  ...
  public void init () throws ServletException {
    // get properties
```

```
    ContextProperties props = ContextProperties.getProperties (this);
    // configure logging
    Utilities.configLog (props);
    category.debug ("init");
  } // init
  ...

} // EventView
```

As I mentioned when discussing the ListViewList example (see the section above entitled "Write the Hamlet for the ListViewList example"), a user can get more detailed information about an event by clicking on the event's signature field. The ListViewList table contains links to bring up the EventView.html page. For the first event in the table, the EventView.html page is called with `EventView.html?Index=0`.

The code in the `doGet()` method retrieves the date format, the actual value of the index with `getParameter()`, and the events vector. These objects are then passed to the `EventViewHandler` constructor to create an instance (`handler`). Next, the `serveDoc()` method is called with the template and the `handler`.

```
public class EventView extends Hamlet {

  ...
  public void doGet (HttpServletRequest req, HttpServletResponse res) throws
ServletException {
    try {
      category.debug ("doGet");
      HttpSession session = req.getSession (false);
      if (session != null) {
        // get properties
        ContextProperties props = ContextProperties.getProperties (this);
        // get formats
        String format = props.getStringProperty ("DateFormat");
        String timeZone = props.getStringProperty ("TimeZone");
        category.debug ("date format: " + format);
        category.debug ("time zone: " + timeZone);
        SimpleDateFormat dateFormat = new SimpleDateFormat (format);
        dateFormat.setTimeZone (Utilities.getTimeZone (timeZone));
        // get parameter
        String str = req.getParameter ("Index");
        int index = Integer.parseInt (str);
        category.debug ("Index : " + index);
        // get events
        Vector events = (Vector) session.getAttribute (WebApp.events);
        // serve document
        HamletHandler handler = new EventViewHandler (this, events, index,
dateFormat);
        serveDoc (req, res, "EventViewTemplate.html", handler);
      } else {
        res.sendRedirect (res.encodeURL (req.getContextPath () +
"/LoginHere.html"));
      } // if
    } catch (Exception e) {
      category.error ("", e);
      throw new ServletException (e);
    } // try
  } // doGet
  ...

} // EventView
```

In `serveDoc()`, a SAX reader parses the XHTML template file. It calls the `handler`'s `getElementReplacement()` method to supply the actual event

details, which overwrite the mock-up data.

```
  private static class EventViewHandler extends HamletHandler {

    ...
    public String getElementReplacement (String id, String name, Attributes
atts) throws Exception {
      if (id.equals ("Reception")) {
        return dateFormat.format (event.getReceptionDate ());
      } else if (id.equals ("Generation")) {
        return dateFormat.format (event.getGenerationDate ());
      } else if (id.equals ("Type")) {
        return event.getType ();
      } else if (id.equals ("Class")) {
        return event.getClassName ();
      } else if (id.equals ("Adapter")) {
        return event.getAdapterVersion ();
      } else if (id.equals ("Severity")) {
        return "" + event.getSeverity ();
      } else if (id.equals ("Correlation")) {
        return "" + event.getCorrelationLevel ();
      } else if (id.equals ("Count")) {
        return "" + event.getEventCount ();
      } else if (id.equals ("Customer")) {
        return event.getCustomerName ();
      } else if (id.equals ("Sensor")) {
        return event.getSensor ();
      } else if (id.equals ("SensorIP")) {
        return event.getSensorIP ();
      } else if (id.equals ("SensorType")) {
        return event.getSensorType ();
      } else if (id.equals ("SensorHost")) {
        return event.getSensorHostname ();
      } else if (id.equals ("SrcIPPort")) {
        return event.getSrcIP () + "  (" + event.getSrcPort () + ")";
      } else if (id.equals ("DstIPPort")) {
        return event.getDstIP () + "  (" + event.getDstPort () + ")";
      } else if (id.equals ("SrcToken")) {
        return event.getSrcToken ();
      } else if (id.equals ("DstToken")) {
        return event.getDstToken ();
      } else if (id.equals ("Category")) {
        return event.getClassCategory ();
      } else if (id.equals ("Signature")) {
        return event.getSignature ();
      } else if (id.equals ("Context")) {
        return event.getContext ();
      } else if (id.equals ("Message")) {
        return event.getMessage ();
      } else if (id.equals ("Comment")) {
        return event.getComment ();
      } // if
      return "?";
    } // getElementReplacement
    ...


  } // EventViewHandler
```

You can use the `getElementAttributes()` callback to color a single row in the table. It displays the event's severity. The following code in the XHTML template file renders this row:

```
        ...
        <TR ID="Color" BGCOLOR="#FF0000">
          <TH><P CLASS="name">Severity</P></TH>
          <TD> </TD>
          <TD><P CLASS="value">
```

```
            <REPLACE ID="Severity">30</REPLACE>
          </P></TD>
        </TR>
        ...
        <INPUT ID="Index" TYPE="Hidden" NAME="Index" VALUE="" />
        ...
```

The <TR> tag contains an `ID` and a `BGCOLOR` attribute. The Hamlet will overwrite the default value `#FF0000` (red) of the `BGCOLOR` attribute with the RGB value representing the event's severity in the `getElementAttributes()` callback using `Helpers.getAttributes()`. `Helpers.getAttributes()` is also used to store the index of the selected event in a hidden input tag. The index is retrieved again in the `doPost()` method when the user submits his or her input.

```
  private static class EventViewHandler extends HamletHandler {

    ...
    public Attributes getElementAttributes (String id, String name,
Attributes atts) throws Exception {
      if (id.equals ("Color")) {
        Color color = event.getColor ();
        int rgb = color.getRGB ();
        String col = Integer.toHexString (rgb);
        atts = Helpers.getAttributes (atts, "BGCOLOR", "#" + col.substring
(2));
      } else if (id.equals ("Index")) {
        atts = Helpers.getAttributes (atts, "VALUE", "" + index);
      } // if
      return atts;
    } // getElementAttributes
    ...

  } // EventViewHandler
```

In addition to the table that displays the event details, the XHTML template file contains a number of input controls: a drop-down list, a text area, and two buttons. The drop-down list marks an event as reviewed or not reviewed, and the text area allows an operator to comment on a particular event. The **Clear** button clears the text area and the selection of the drop-down list. When a user clicks the **OK** button, the user input is submitted back to the server for processing.

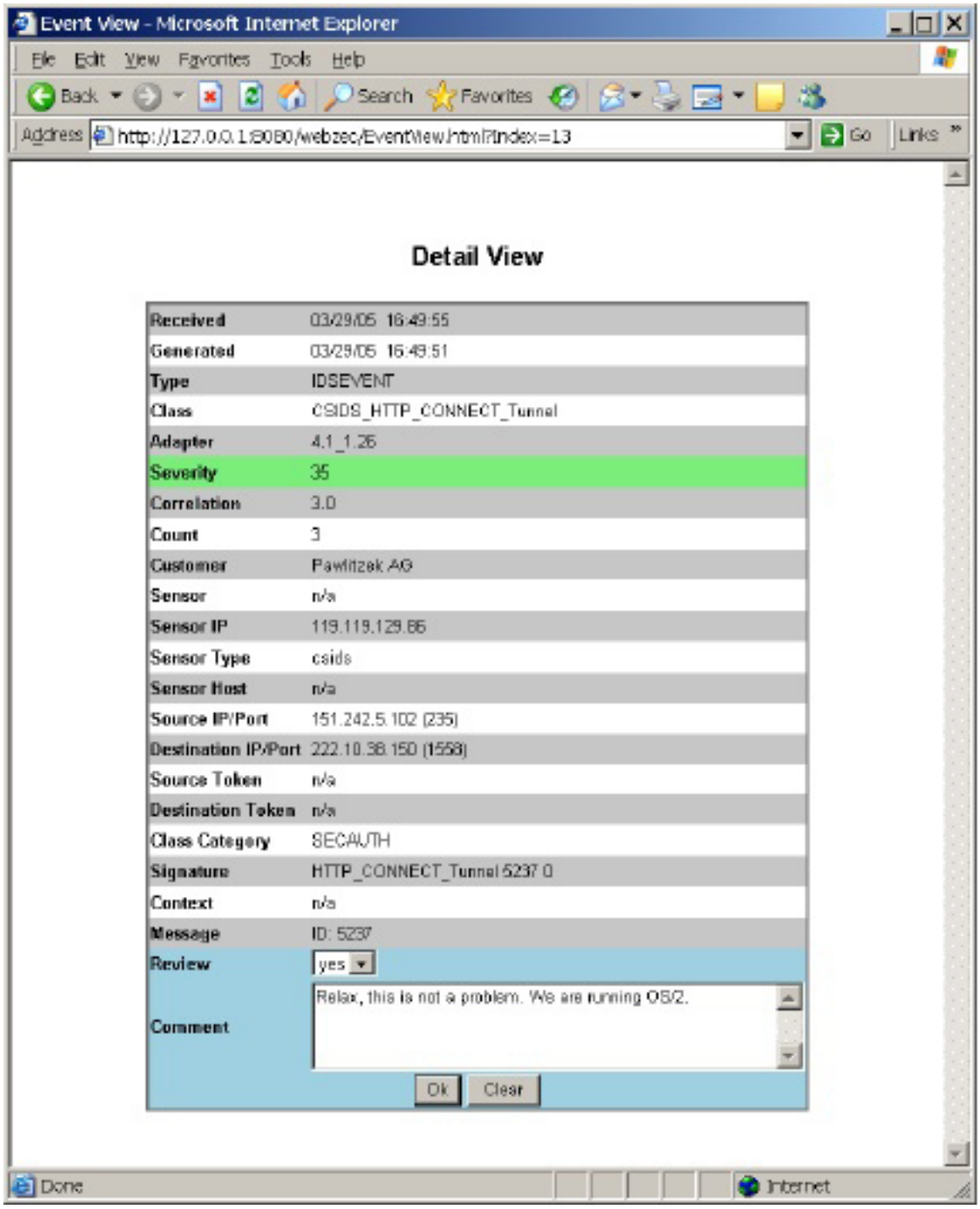
The `doPost()` method performs the processing as specified in the `<FORM ACTION="EventView.html" METHOD="POST" />` tag. It retrieves the user comment, the index of the selected event from the hidden input tag, and the selection of the drop-down list with `getParameter()`. Then, `eventModelStore.changeEvent()` is called to make changes to the event, before the user is redirected to the current view with a call to `sendRedirect()`.

```
public class EventView extends Hamlet {

  ...
  public void doPost (HttpServletRequest req, HttpServletResponse res) throws
ServletException {
    try {
      category.debug ("doPost");
      HttpSession session = req.getSession (false);
      if (session != null) {
        // get parameters
        String comment = req.getParameter ("Comment");
```

```
        String idx = req.getParameter ("Index");
        int index = Integer.parseInt (idx);
        String str = req.getParameter ("Review");
        str = str.toLowerCase ();
        Boolean review = new Boolean ("yes".equals (str));
        // get events
        Vector events = (Vector) session.getAttribute (WebApp.events);
        // change event
        WebApp webApp = WebApp.getWebApp ();
        EventModelStoreInterface eventModelStore = webApp.getEventModelStore
();
        EventDesc event = (EventDesc) events.elementAt (index);
        eventModelStore.changeEvent (event, review, comment);
        // go back to current view
        String currentView = (String) session.getAttribute
(WebApp.currentView);
        res.sendRedirect (res.encodeURL (req.getContextPath () + "/" +
currentView));
      } else {
        res.sendRedirect (res.encodeURL (req.getContextPath () +
"/LoginHere.html"));
      } // if
    } catch (Exception e) {
      category.error ("", e);
      throw new ServletException (e);
    } // try
  } // doPost
  ...

} // EventView
```

Figure 10 provides a snapshot of EventView showing the details of an actual event at runtime.

Figure 10. View of EventView showing details of an actual event at runtime



View the complete EventView Hamlet code

Here's the complete code for the EventView example:

```
/**
 *
 * © Copyright International Business Machines Corporation 2004, 2006.
 * All rights reserved.
 *
 * The 'EventView' servlet provides a detailed event view.
```

```
 *
 * File     : EventView.java
 * Created  : 2004/05/10
 *
 * @author    Rene Pawlitzek  (rpa@zurich.ibm.com)
 * @version   1.00, 2006/11/28
 * @since     JDK 1.3
 *
 */


package com.ibm.webzec.apps.server;


import java.awt.*;
import java.text.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.ibm.hamlet.*;
import com.ibm.hamlet.helpers.*;
import com.ibm.webzec.libs.db.*;
import com.ibm.webzec.libs.util.*;
import org.apache.log4j.*;
import org.xml.sax.*;


public class EventView extends Hamlet {


  // log4j
  private static Category category = Category.getInstance
(EventView.class.getName ());


  private static class EventViewHandler extends HamletHandler {


    private  int               index;
    private  EventDesc         event;
    private  SimpleDateFormat  dateFormat;


    public EventViewHandler (Hamlet hamlet, Vector events, int index,
SimpleDateFormat dateFormat) {
      super (hamlet);
      this.index = index;
      this.dateFormat = dateFormat;
      event = (EventDesc) events.elementAt (index);
    } // EventViewHandler


    public String getElementReplacement (String id, String name, Attributes
atts) throws Exception {
      if (id.equals ("Reception")) {
        return dateFormat.format (event.getReceptionDate ());
      } else if (id.equals ("Generation")) {
        return dateFormat.format (event.getGenerationDate ());
      } else if (id.equals ("Type")) {
        return event.getType ();
      } else if (id.equals ("Class")) {
        return event.getClassName ();
      } else if (id.equals ("Adapter")) {
        return event.getAdapterVersion ();
      } else if (id.equals ("Severity")) {
        return "" + event.getSeverity ();
      } else if (id.equals ("Correlation")) {
        return "" + event.getCorrelationLevel ();
      } else if (id.equals ("Count")) {
        return "" + event.getEventCount ();
      } else if (id.equals ("Customer")) {
        return event.getCustomerName ();
      } else if (id.equals ("Sensor")) {
        return event.getSensor ();
```

```
      } else if (id.equals ("SensorIP")) {
        return event.getSensorIP ();
      } else if (id.equals ("SensorType")) {
        return event.getSensorType ();
      } else if (id.equals ("SensorHost")) {
        return event.getSensorHostname ();
      } else if (id.equals ("SrcIPPort")) {
        return event.getSrcIP () + "  (" + event.getSrcPort () + ")";
      } else if (id.equals ("DstIPPort")) {
        return event.getDstIP () + "  (" + event.getDstPort () + ")";
      } else if (id.equals ("SrcToken")) {
        return event.getSrcToken ();
      } else if (id.equals ("DstToken")) {
        return event.getDstToken ();
      } else if (id.equals ("Category")) {
        return event.getClassCategory ();
      } else if (id.equals ("Signature")) {
        return event.getSignature ();
      } else if (id.equals ("Context")) {
        return event.getContext ();
      } else if (id.equals ("Message")) {
        return event.getMessage ();
      } else if (id.equals ("Comment")) {
        return event.getComment ();
      } // if
      return "?";
    } // getElementReplacement


    public Attributes getElementAttributes (String id, String name,
Attributes atts) throws Exception {
      if (id.equals ("Color")) {
        Color color = event.getColor ();
        int rgb = color.getRGB ();
        String col = Integer.toHexString (rgb);
        atts = Helpers.getAttributes (atts, "BGCOLOR", "#" + col.substring
(2));
      } else if (id.equals ("Index")) {
        atts = Helpers.getAttributes (atts, "VALUE", "" + index);
      } // if
      return atts;
  } // getElementAttributes


  } // EventViewHandler


  public void init () throws ServletException {
    // get properties
    ContextProperties props = ContextProperties.getProperties (this);
    // configure logging
    Utilities.configLog (props);
    category.debug ("init");
  } // init


  public void doPost (HttpServletRequest req, HttpServletResponse res) throws
ServletException {
    try {
      category.debug ("doPost");
      HttpSession session = req.getSession (false);
      if (session != null) {
        // get parameters
        String comment = req.getParameter ("Comment");
        String idx = req.getParameter ("Index");
        int index = Integer.parseInt (idx);
        String str = req.getParameter ("Review");
        str = str.toLowerCase ();
        Boolean review = new Boolean ("yes".equals (str));
        // get events
        Vector events = (Vector) session.getAttribute (WebApp.events);
        // change event
        WebApp webApp = WebApp.getWebApp ();
        EventModelStoreInterface eventModelStore = webApp.getEventModelStore
```

```
();
        EventDesc event = (EventDesc) events.elementAt (index);
        eventModelStore.changeEvent (event, review, comment);
        // go back to current view
        String currentView = (String) session.getAttribute
(WebApp.currentView);
        res.sendRedirect (res.encodeURL (req.getContextPath () + "/" +
currentView));
      } else {
        res.sendRedirect (res.encodeURL (req.getContextPath () +
"/LoginHere.html"));
      } // if
    } catch (Exception e) {
      category.error ("", e);
      throw new ServletException (e);
    } // try
  } // doPost


  public void doGet (HttpServletRequest req, HttpServletResponse res) throws
ServletException {
    try {
      category.debug ("doGet");
      HttpSession session = req.getSession (false);
      if (session != null) {
        // get properties
        ContextProperties props = ContextProperties.getProperties (this);
        // get formats
        String format = props.getStringProperty ("DateFormat");
        String timeZone = props.getStringProperty ("TimeZone");
        category.debug ("date format: " + format);
        category.debug ("time zone: " + timeZone);
        SimpleDateFormat dateFormat = new SimpleDateFormat (format);
        dateFormat.setTimeZone (Utilities.getTimeZone (timeZone));
        // get parameter
        String str = req.getParameter ("Index");
        int index = Integer.parseInt (str);
        category.debug ("Index : " + index);
        // get events
        Vector events = (Vector) session.getAttribute (WebApp.events);
        // serve document
        HamletHandler handler = new EventViewHandler (this, events, index,
dateFormat);
        serveDoc (req, res, "EventViewTemplate.html", handler);
      } else {
        res.sendRedirect (res.encodeURL (req.getContextPath () +
"/LoginHere.html"));
      } // if
    } catch (Exception e) {
      category.error ("", e);
      throw new ServletException (e);
    } // try
  } // doGet


  public String getServletInfo () {
    category.debug ("getServletInfo");
    return "EventView servlet";
  } // getServletInfo


  public void destroy () {
    category.debug ("destroy");
  } // destroy


} // EventView


/* ----- End of File ----- */
```

Section 7. OptionView example

Introduction to the OptionView example

The next example shows how to use a Hamlet to implement an options dialog. Most applications require such a dialog to specify preferences.

Create the XHTML code for OptionViewTemplate

Again, the project begins with the design of an XHTML template file: OptionViewTemplate.html. Here is the template's XHTML code:

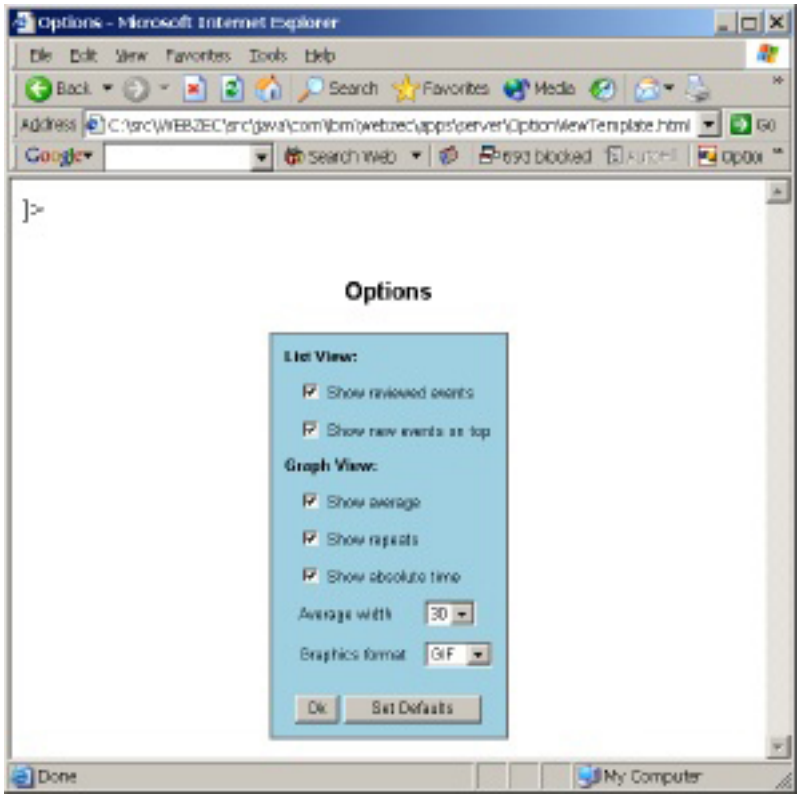
```
<!DOCTYPE WebZEC [ <!ENTITY nbsp " "> ]>
<HTML>
  <HEAD>
    <TITLE>Options</TITLE>
    <LINK REL="stylesheet" TYPE="text/css" HREF="View.css" />
  </HEAD>
  <BODY>
    <P CLASS="title">
      <BR />Options
    </P>
    <FORM ACTION="OptionView.html" METHOD="POST">
      <TABLE CLASS="dialog" CELLSPACING="10" ALIGN="CENTER">
        <TR>
          <TD COLSPAN="3">
            <P CLASS="name">
               List View:
            </P>
          </TD>
        </TR>
        <TR>
          <TD />
          <TD COLSPAN="2">
            <P CLASS="value">
              <INPUT ID="ShowReviewedEvents" CLASS="checkbox" TYPE="CHECKBOX"
NAME="ShowReviewedEvents" VALUE="true" />
               Show reviewed events
            </P>
          </TD>
        </TR>
        <TR>
          <TD />
          <TD COLSPAN="2">
            <P CLASS="value">
              <INPUT ID="ShowNewEventsOnTop" CLASS="checkbox" TYPE="CHECKBOX"
NAME="ShowNewEventsOnTop" VALUE="true" />
               Show new events on top
            </P>
          </TD>
        </TR>
        <TR>
          <TD COLSPAN="3">
            <P CLASS="name">
               Graph View:
            </P>
          </TD>
        </TR>
        <TR>
          <TD />
```

```
          <TD COLSPAN="2">
            <P CLASS="value">
              <INPUT ID="ShowAverage" CLASS="checkbox" TYPE="CHECKBOX"
NAME="ShowAverage" VALUE="true" />
              Show average
            </P>
          </TD>
        </TR>
        <TR>
          <TD />
          <TD COLSPAN="2">
            <P CLASS="value">
              <INPUT ID="ShowRepeats" CLASS="checkbox" TYPE="CHECKBOX"
NAME="ShowRepeats" VALUE="true" />
              Show repeats
            </P>
          </TD>
        </TR>
        <TR>
          <TD />
          <TD COLSPAN="2">
            <P CLASS="value">
              <INPUT ID="ShowAbsoluteTime" CLASS="checkbox" TYPE="CHECKBOX"
NAME="ShowAbsoluteTime" VALUE="true" />
              Show absolute time
            </P>
          </TD>
        </TR>
        <TR>
          <TD />
          <TD>
            <P CLASS="value">
              Average width 
            </P>
          </TD>
          <TD>
            <SELECT CLASS="dropdown" size="1" NAME="AverageWidth">
              <OPTION ID="AverageWidth" WIDTH="5">5</OPTION>
              <OPTION ID="AverageWidth" WIDTH="10">10</OPTION>
              <OPTION ID="AverageWidth" WIDTH="15">15</OPTION>
              <OPTION ID="AverageWidth" WIDTH="20">20</OPTION>
              <OPTION ID="AverageWidth" WIDTH="25">25</OPTION>
              <OPTION ID="AverageWidth" WIDTH="30">30</OPTION>
            </SELECT>
          </TD>
        </TR>
        <TR>
          <TD />
          <TD>
            <P CLASS="value">
              Graphics format 
            </P>
          </TD>
          <TD>
            <SELECT CLASS="dropdown" size="1" NAME="GraphicsFormat">
              <OPTION ID="GraphicsFormat" FORMAT="SVG">SVG</OPTION>
              <OPTION ID="GraphicsFormat" FORMAT="GIF">GIF</OPTION>
            </SELECT>
          </TD>
        </TR>
        <TR>
        </TR>
        <TR>
          <TD COLSPAN="3">
            <P CLASS="buttons">
              <INPUT CLASS="button" TYPE="SUBMIT" VALUE=" Ok " />
              <INPUT CLASS="button" TYPE="SUBMIT" NAME="SetDefaults" VALUE="
Set Defaults " />
            </P>
          </TD>
        </TR>
      </TABLE>
    </FORM>
  </BODY>
```

```
</HTML>
```

Figure 11 shows what the template looks like in a browser.

Figure 11. View of the XHTML template file OptionViewTemplate.html



This template file uses a Cascading Style Sheet (View.css) and contains five checkboxes, two drop-down lists, and two buttons. There are no <REPLACE> or <REPEAT>tags; thus, there will be no need to substitute placeholders with actual content at runtime.

Write the OptionView Hamlet

As usual, the `doGet()` method is called when a user request is received. First, the `doGet()` method looks for an HTTP session. If the user logged in successfully using the login page, an HTTP session exists and the code in the `doGet()` method passes the name of the XHTML template file (OptionViewTemplate.html) and an instance of `OptionViewHandler` to the Hamlet framework by calling `serveDoc()`.

```
public class OptionView extends Hamlet {

  ...
  public void doGet (HttpServletRequest req, HttpServletResponse res) throws
ServletException {
    try {
      category.debug ("doGet");
      HttpSession session = req.getSession (false);
      if (session != null) {
        // serve document
        HamletHandler handler = new OptionViewHandler (this, session);
```

```
        serveDoc (req, res, "OptionViewTemplate.html", handler);
      } else {
        res.sendRedirect (res.encodeURL (req.getContextPath () +
"/LoginHere.html"));
      } // if
    } catch (Exception e) {
      category.error ("", e);
      throw new ServletException (e);
    } // try
  } // doGet
  ...

} // OptionView
```

The `serveDoc()` method parses the XHTML template file but won't call the `handler`'s `getElementReplacement()` or `getElementRepeatCount()` method because the template does not contain any <REPLACE> or <REPEAT> tags. However, the `getElementAttributes()` callback is invoked, giving you a chance to work with the attributes of certain tags. The Hamlet uses this opportunity to add `CHECKED` attributes to the `<INPUT>` tags (representing the checkboxes) and `SELECTED` attributes to the `<OPTION>` tags (representing the drop-down choices).

For example, if the user has specified that he or she wishes to see reviewed events, you need to add the `CHECKED` attribute to the `<INPUT ID="ShowReviewedEvents" CLASS="checkbox" TYPE="CHECKBOX" NAME="ShowReviewedEvents" VALUE="true" />` tag. This is done in the private `getAttributes()` function. It adds the `CHECKED` attribute if the `showReviewedEvents` attribute in the HTTP session is set to `true`. The other checkboxes are processed similarly.

```
  private static class OptionViewHandler extends HamletHandler {

    ...

    private Attributes getAttributes (Attributes atts, HttpSession session,
String name) {
      Boolean b = (Boolean) session.getAttribute (name);
      if (b.booleanValue ())
        atts = Helpers.getAttributes (atts, "CHECKED", "checked");
      return atts;
    } // getAttributes


    public Attributes getElementAttributes (String id, String name,
Attributes atts) throws Exception {
      if (id.equals (showReviewedEvents)) {
        atts = getAttributes (atts, session, showReviewedEvents);
      } else if (id.equals (showNewEventsOnTop)) {
        atts = getAttributes (atts, session, showNewEventsOnTop);
      } else if (id.equals (showAverage)) {
        atts = getAttributes (atts, session, showAverage);
      } else if (id.equals (showRepeats)) {
        atts = getAttributes (atts, session, showRepeats);
      } else if (id.equals (showAbsoluteTime)) {
        atts = getAttributes (atts, session, showAbsoluteTime);
      } else if (id.equals (averageWidth)) {
        ...
      } else if (id.equals (graphicsFormat)) {
        ...
      } // if
      return atts;
    } // getElementAttributes
```

```
  } // OptionViewHandler
```

You also need to add the `SELECTED` attribute to the `<OPTION>` tags.

```
            ...
            <SELECT CLASS="dropdown" size="1" NAME="AverageWidth">
              <OPTION ID="AverageWidth" WIDTH="5">5</OPTION>
              <OPTION ID="AverageWidth" WIDTH="10">10</OPTION>
              <OPTION ID="AverageWidth" WIDTH="15">15</OPTION>
              <OPTION ID="AverageWidth" WIDTH="20">20</OPTION>
              <OPTION ID="AverageWidth" WIDTH="25">25</OPTION>
              <OPTION ID="AverageWidth" WIDTH="30">30</OPTION>
            </SELECT>
            ...
            <SELECT CLASS="dropdown" size="1" NAME="GraphicsFormat">
              <OPTION ID="GraphicsFormat" FORMAT="SVG">SVG</OPTION>
              <OPTION ID="GraphicsFormat" FORMAT="GIF">GIF</OPTION>
            </SELECT>
            ...
```

No `SELECTED` attributes were added to the `<OPTION>` tags in the XHTML template file, but obviously one option in each drop-down list should be marked as selected. So you need to add the `SELECTED` attributes at runtime with the help of `Helpers.getAttributes()` in the `getElementAttributes()` callback. For example, if the user chose an average width of 25, the `<OPTION ID="AverageWidth" WIDTH="25">` tag should have the `SELECTED` attribute. For the drop-down list containing the average width options, you can do this with the following code:

```
  private static class OptionViewHandler extends HamletHandler {

    ...
    public Attributes getElementAttributes (String id, String name,
Attributes atts) throws Exception {
      if (id.equals (showReviewedEvents)) {
        ...
      } else if (id.equals (showNewEventsOnTop)) {
        ...
      } else if (id.equals (showAverage)) {
        ...
      } else if (id.equals (showRepeats)) {
        ...
      } else if (id.equals (showAbsoluteTime)) {
        ...
      } else if (id.equals (averageWidth)) {
        String width = atts.getValue ("WIDTH");
        Integer i = (Integer) session.getAttribute (averageWidth);
        String curWidth = i.toString ();
        if (curWidth.equals (width))
          atts = Helpers.getAttributes (atts, "SELECTED", "selected");
      } else if (id.equals (graphicsFormat)) {
        String format = atts.getValue ("FORMAT");
        String curFormat = (String ) session.getAttribute (graphicsFormat);
        if (curFormat.equals (format))
          atts = Helpers.getAttributes (atts, "SELECTED", "selected");
      } // if
      return atts;
    } // getElementAttributes
    ...


  } // OptionViewHandler
```

In the `getElementAttributes()` method, the average width is retrieved from the value of the `WIDTH` attribute. Next, `Helpers.getAttributes()` is called to add the `SELECTED` attribute if the `averageWidth` attribute in the HTTP session does match the retrieved value. The addition of the `SELECTED` attribute for the **Graphics Format** drop-down list is done in the same way.

The Hamlet also contains the `doPost()` method, which processes the user's input. If the user presses the **OK** button, the `OptionView`'s `doPost()` method is called because the `<FORM>` tag contains the `ACTION="OptionView.html"` attribute. Assume for a moment that the user selects the **Show reviewed events** option. In this case, a name-value pair (`ShowReviewedEvents, true`) is transmitted from the browser to the server. The `doPost()` method calls `setBooleanAttribute(session, req, showReviewedEvents)` to store the user's choice in the session object. `setIntegerAttribute()` and `setStringAttribute()` are called to process the selections from the drop-down lists. Note that input parameter processing only occurs if an HTTP session exists. In other words, the user must have logged in successfully.

```
public class OptionView extends Hamlet {

  ...
  public void doPost (HttpServletRequest req, HttpServletResponse res) throws
ServletException {
    try {
      category.debug ("doPost");
      HttpSession session = req.getSession (false);
      if (session != null) {
        // get options and set properties
        setBooleanAttribute (session, req, showReviewedEvents);
        setBooleanAttribute (session, req, showNewEventsOnTop);
        setBooleanAttribute (session, req, showAverage);
        setBooleanAttribute (session, req, showRepeats);
        setBooleanAttribute (session, req, showAbsoluteTime);
        setIntegerAttribute (session, req, averageWidth);
        setStringAttribute (session, req, graphicsFormat);
        if (req.getParameter ("SetDefaults") != null) {
          // set defaults
          setDefaultAttributes (session);
          HamletHandler handler = new OptionViewHandler (this, session);
          serveDoc (req, res, "OptionViewTemplate.html", handler);
        } else {
          // redirect
          String currentView = (String) session.getAttribute
(WebApp.currentView);
          res.sendRedirect (res.encodeURL (req.getContextPath () + "/" +
currentView));
        } // if
      } else {
        res.sendRedirect (res.encodeURL (req.getContextPath () +
"/LoginHere.html"));
      } // if
    } catch (Exception e) {
      category.error ("", e);
      throw new ServletException (e);
    } // try
  } // doPost
  ...

} // OptionView
```

View the complete OptionView Hamlet code

Here is the complete Hamlet code for the OptionView example:

```
/**
 *
 * © Copyright International Business Machines Corporation 2004, 2006.
 * All rights reserved.
 *
 * The 'OptionView' servlet provides the option view.
 *
 * File     : OptionView.java
 * Created  : 2004/07/08
 *
 * @author    Rene Pawlitzek  (rpa@zurich.ibm.com)
 * @version   1.00, 2006/11/28
 * @since     JDK 1.3
 *
 */


package com.ibm.webzec.apps.server;


import javax.servlet.*;
import javax.servlet.http.*;
import com.ibm.hamlet.*;
import com.ibm.hamlet.helpers.*;
import com.ibm.webzec.libs.util.*;
import org.apache.log4j.*;
import org.xml.sax.*;


public class OptionView extends Hamlet {


  // log4j
  private static Category category = Category.getInstance
(OptionView.class.getName ());


  public static final  String  gif = "GIF";
  public static final  String  svg = "SVG";

  private static final  Boolean  showReviewedEventsDefault = Boolean.FALSE;
  private static final  Boolean  showNewEventsOnTopDefault = Boolean.FALSE;
  private static final  Boolean  showAverageDefault        = Boolean.TRUE;
  private static final  Boolean  showRepeatsDefault        = Boolean.TRUE;
  private static final  Boolean  showAbsoluteTimeDefault   = Boolean.FALSE;
  private static final  Integer  averageWidthDefault       = new Integer
(15);
  private static final  String   graphicsFormatDefault     = svg;

  public static final  String  showReviewedEvents = "ShowReviewedEvents";
  public static final  String  showNewEventsOnTop = "ShowNewEventsOnTop";
  public static final  String  showAverage        = "ShowAverage";
  public static final  String  showRepeats        = "ShowRepeats";
  public static final  String  showAbsoluteTime   = "ShowAbsoluteTime";
  public static final  String  averageWidth       = "AverageWidth";
  public static final  String  graphicsFormat     = "GraphicsFormat";


  public static void setDefaultAttributes (HttpSession aSession) {
    aSession.setAttribute (showReviewedEvents, showReviewedEventsDefault);
    aSession.setAttribute (showNewEventsOnTop, showReviewedEventsDefault);
    aSession.setAttribute (showAverage, showAverageDefault);
    aSession.setAttribute (showRepeats, showRepeatsDefault);
    aSession.setAttribute (showAbsoluteTime, showAbsoluteTimeDefault);
    aSession.setAttribute (averageWidth, averageWidthDefault);
    aSession.setAttribute (graphicsFormat, graphicsFormatDefault);
  } // setDefaultAttributes


  public static boolean isSVG (HttpSession aSession) {
    String format = (String) aSession.getAttribute (graphicsFormat);
```

```
    return svg.equals (format);
  } // isSVG


  public static boolean isGIF (HttpSession aSession) {
    String format = (String) aSession.getAttribute (graphicsFormat);
    return gif.equals (format);
  } // isGIF


  private static class OptionViewHandler extends HamletHandler {


    private  HttpSession  session;


    public OptionViewHandler (Hamlet hamlet, HttpSession session) {
      super (hamlet);
      this.session = session;
    } // OptionViewHandler


    private Attributes getAttributes (Attributes atts, HttpSession session,
String name) {
      Boolean b = (Boolean) session.getAttribute (name);
      if (b.booleanValue ())
        atts = Helpers.getAttributes (atts, "CHECKED", "checked");
      return atts;
    } // getAttributes


    public Attributes getElementAttributes (String id, String name,
Attributes atts) throws Exception {
      if (id.equals (showReviewedEvents)) {
        atts = getAttributes (atts, session, showReviewedEvents);
      } else if (id.equals (showNewEventsOnTop)) {
        atts = getAttributes (atts, session, showNewEventsOnTop);
      } else if (id.equals (showAverage)) {
        atts = getAttributes (atts, session, showAverage);
      } else if (id.equals (showRepeats)) {
        atts = getAttributes (atts, session, showRepeats);
      } else if (id.equals (showAbsoluteTime)) {
        atts = getAttributes (atts, session, showAbsoluteTime);
      } else if (id.equals (averageWidth)) {
        String width = atts.getValue ("WIDTH");
        Integer i = (Integer) session.getAttribute (averageWidth);
        String curWidth = i.toString ();
        if (curWidth.equals (width))
          atts = Helpers.getAttributes (atts, "SELECTED", "selected");
      } else if (id.equals (graphicsFormat)) {
        String format = atts.getValue ("FORMAT");
        String curFormat = (String ) session.getAttribute (graphicsFormat);
        if (curFormat.equals (format))
          atts = Helpers.getAttributes (atts, "SELECTED", "selected");
      } // if
      return atts;
    } // getElementAttributes


  } // OptionViewHandler



  public void init () throws ServletException {
    // get properties
    ContextProperties props = ContextProperties.getProperties (this);
    // configure logging
    Utilities.configLog (props);
    category.debug ("init");
  } // init


  private void setBooleanAttribute (HttpSession session, HttpServletRequest
req, String param) {
```

```
    String val = req.getParameter (param);
    Boolean b = Boolean.valueOf (val);
    session.setAttribute (param, b);
  } // setBooleanAttribute


  private void setIntegerAttribute (HttpSession session, HttpServletRequest
req, String param) {
     String val = req.getParameter (param);
     Integer i = new Integer (val);
     session.setAttribute (param, i);
  } // setIntegerAttribute


  private void setStringAttribute (HttpSession session, HttpServletRequest
req, String param) {
    String val = req.getParameter (param);
    session.setAttribute (param, val);
  } // setStringAttribute


  public void doPost (HttpServletRequest req, HttpServletResponse res) throws
ServletException {
    try {
      category.debug ("doPost");
      HttpSession session = req.getSession (false);
      if (session != null) {
        // get options and set properties
        setBooleanAttribute (session, req, showReviewedEvents);
        setBooleanAttribute (session, req, showNewEventsOnTop);
        setBooleanAttribute (session, req, showAverage);
        setBooleanAttribute (session, req, showRepeats);
        setBooleanAttribute (session, req, showAbsoluteTime);
        setIntegerAttribute (session, req, averageWidth);
        setStringAttribute (session, req, graphicsFormat);
        if (req.getParameter ("SetDefaults") != null) {
          // set defaults
          setDefaultAttributes (session);
          HamletHandler handler = new OptionViewHandler (this, session);
          serveDoc (req, res, "OptionViewTemplate.html", handler);
        } else {
          // redirect
          String currentView = (String) session.getAttribute
(WebApp.currentView);
          res.sendRedirect (res.encodeURL (req.getContextPath () + "/" +
currentView));
        } // if
      } else {
        res.sendRedirect (res.encodeURL (req.getContextPath () +
"/LoginHere.html"));
      } // if
    } catch (Exception e) {
      category.error ("", e);
      throw new ServletException (e);
    } // try
  } // doPost


  public void doGet (HttpServletRequest req, HttpServletResponse res) throws
ServletException {
    try {
      category.debug ("doGet");
      HttpSession session = req.getSession (false);
      if (session != null) {
        // serve document
        HamletHandler handler = new OptionViewHandler (this, session);
        serveDoc (req, res, "OptionViewTemplate.html", handler);
      } else {
        res.sendRedirect (res.encodeURL (req.getContextPath () +
"/LoginHere.html"));
      } // if
    } catch (Exception e) {
      category.error ("", e);
      throw new ServletException (e);
```

```
    } // try
  } // doGet


  public String getServletInfo () {
    category.debug ("getServletInfo");
    return "OptionView servlet";
  } // getServletInfo


  public void destroy () {
    category.debug ("destroy");
  } // destroy


} // OptionView


/* ----- End of File ----- */
```

Section 8. ErrorView example

Introduction to the ErrorView example

You'll wrap up this tutorial with the ErrorView example. The ErrorView displays diagnostic information (error messages, error codes, stack traces, and so on) if an error occurs in the Web application.

Create the XHTML code for ErrorViewTemplate

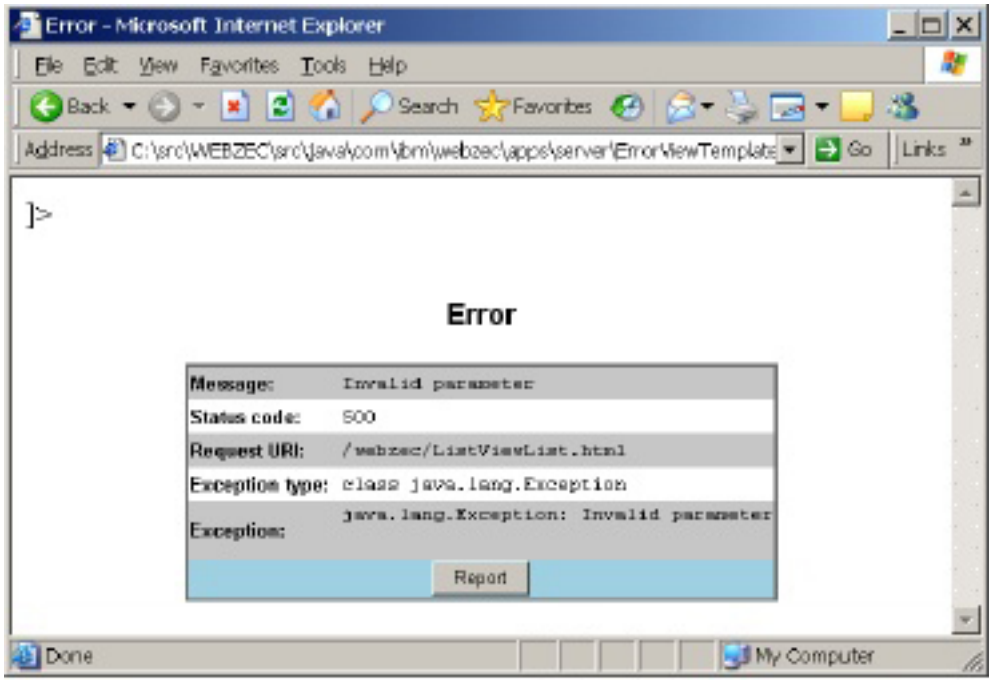
Again, the project begins with the creation of an XHTML template file: ErrorViewTemplate.html.

```
<!DOCTYPE WebZEC [ <!ENTITY nbsp " "> ]>
<HTML>
  <HEAD>
    <TITLE>Error</TITLE>
    <LINK REL="stylesheet" TYPE="text/css" HREF="View.css" />
  </HEAD>
  <BODY>
    <SCRIPT LANGUAGE="JavaScript">
       if (window != top) top.location.href = location.href;
    </SCRIPT>
    <P CLASS="title">
      <BR />Error
    </P>
    <FORM ACTION="ErrorView.html" METHOD="POST">
    <TABLE CLASS="dialog" CELLPADDING="1" CELLSPACING="0"
ALIGN="CENTER">
      <TR CLASS="odd">
        <TH><P CLASS="name">Message:</P></TH>
        <TD> </TD>
        <TD><P CLASS="mono"><REPLACE ID="Message">Invalid
parameter</REPLACE></P></TD>
      </TR>
```

```
      <TR CLASS="even">
        <TH><P CLASS="name">Status code:</P></TH>
        <TD> </TD>
        <TD><P CLASS="mono">
          <REPLACE ID="StatusCode">500</REPLACE>
        </P></TD>
      </TR>
      <TR CLASS="odd">
        <TH><P CLASS="name">Request URI:</P></TH>
        <TD> </TD>
        <TD><P CLASS="mono">
          <REPLACE
ID="RequestURI">/webzec/ListViewList.html</REPLACE>
        </P></TD>
      </TR>
      <TR CLASS="even">
        <TH><P CLASS="name">Exception type:</P></TH>
        <TD> </TD>
        <TD><P CLASS="mono">
          <REPLACE ID="ExceptionType">class
java.lang.Exception</REPLACE>
        </P></TD>
      </TR>
      <TR CLASS="odd">
        <TH><P CLASS="name">Exception:</P></TH>
        <TD> </TD>
        <TD>
        <PRE CLASS="mono">
          <REPLACE ID="Exception">java.lang.Exception: Invalid
parameter</REPLACE>
        </PRE></TD>
      </TR>
      <TR>
        <TH COLSPAN="3">
          <P CLASS="buttons">
            <INPUT ID="Message" TYPE="hidden" NAME="Message"
VALUE="" />
            <INPUT ID="StatusCode" TYPE="hidden" NAME="StatusCode"
VALUE="" />
            <INPUT ID="RequestURI" TYPE="hidden" NAME="RequestURI"
VALUE="" />
            <INPUT ID="ExceptionType" TYPE="hidden"
NAME="ExceptionType" VALUE="" />
            <INPUT ID="Exception" TYPE="hidden" NAME="Exception"
VALUE="" />
            <INPUT CLASS="button" TYPE="Submit" VALUE=" Report " />
          </P>
        </TH>
      </TR>
    </TABLE>
    </FORM>
  </BODY>
</HTML>
```

Figure 12 illustrates what the template code looks like in a browser.

Figure 12. View of the XHTML template file ErrorViewTemplate.html



Again, you use a Cascading Style Sheet (View.css) to provide consistent formatting. The page contains mock-up data, which presents a fictional error message. Actual content replaces the mock-up data at runtime.

Write the ErrorView Hamlet

To start the next phase, implement the ErrorView hamlet. The `doGet()` method (called when a user request is received) retrieves the error information and passes it to the `ErrorViewHandler` constructor. This constructor creates an instance (`handler`) of `ErrorViewHandler`. As in all the examples before, the `handler` provides the dynamic content (the error information) to the Hamlet framework. Next, `serveDoc()` is called with the template and the `handler` to start parsing the XHTML template file (ErrorViewTemplate.html).

```
public class ErrorView extends Hamlet {

  ...
  public void doGet (HttpServletRequest req, HttpServletResponse res) throws
ServletException {
    try {
      category.debug ("doGet");
      // error message
      String message = na;
      Object obj = req.getAttribute ("javax.servlet.error.message");
      if (obj != null) message = obj.toString ();
      // status code
      String statusCode = na;
      obj = req.getAttribute ("javax.servlet.error.status_code");
      if (obj != null) statusCode = obj.toString ();
      // request URI
      String requestURI = req.getRequestURI ();
      obj = req.getAttribute ("javax.servlet.error.request_uri");
      if (obj != null) requestURI = obj.toString ();
      // exception type
      String exceptionType = na;
      obj = req.getAttribute ("javax.servlet.error.exception_type");
```

```
      if (obj != null) exceptionType = obj.toString ();
      // exception
      String exception = na;
      Throwable t = (Throwable) req.getAttribute
("javax.servlet.error.exception");
      if (t != null) exception = Utilities.getStackTraceAsString (t);
      // serve document
      HamletHandler handler = new ErrorViewHandler (this, message,
statusCode,
        requestURI, exceptionType, exception);
      serveDoc (req, res, "ErrorViewTemplate.html", handler);
    } catch (Exception e) {
      category.error ("", e);
      throw new ServletException (e);
    } // try
  } // doGet
  ...

} // ErrorView
```

The SAX parser will invoke the `handler`'s `getElementReplacement()` callback when it encounters a <REPLACE> tag. In the `getElementReplacement()` method, the actual values for error message, status code, request URI, exception type, and exception (stack trace) are provided. These values were retrieved when the `doGet()` method was executed. As always, the `ID` attribute (stored in the `id` parameter) controls the creation of dynamic content.

```
  private static class ErrorViewHandler extends HamletHandler {

    ...
    public String getElementReplacement (String id, String name, Attributes
atts) throws Exception {
      if (id.equals ("Message")) {
        return message;
      } else if (id.equals ("StatusCode")) {
        return statusCode;
      } else if (id.equals ("RequestURI")) {
        return requestURI;
      } else if (id.equals ("ExceptionType")) {
        return exceptionType;
      } else if (id.equals ("Exception")) {
        return exception;
      } // if
      return "?";
    } // getElementReplacement
    ...

  } // ErrorViewHandler
```

The XHTML template file also contains a button labeled **Report.** When a user clicks this button, he sends valuable error information about the application's failure back to the developer through e-mail. Thus, you don't just want the error information to display on the page; you also want it to be included in hidden input fields, so the application can retrieve the information when the page is submitted back to report the error.

The hidden input fields already exist in the XHTML template file, but they contain empty values.

```
     <INPUT ID="Message" TYPE="hidden" NAME="Message" VALUE="" />
     <INPUT ID="StatusCode" TYPE="hidden" NAME="StatusCode" VALUE="" />
     <INPUT ID="RequestURI" TYPE="hidden" NAME="RequestURI" VALUE="" />
```

```
     <INPUT ID="ExceptionType" TYPE="hidden" NAME="ExceptionType" VALUE="" />
     <INPUT ID="Exception" TYPE="hidden" NAME="Exception" VALUE="" />
```

With the help of the `Helpers.getAttributes()` method in the `getElementAttributes()` callback, you can overwrite the empty values at runtime.

```
  private static class ErrorViewHandler extends HamletHandler {

    ...
    public Attributes getElementAttributes (String id, String name,
Attributes atts) throws Exception {
      if (id.equals ("Message")) {
        atts = Helpers.getAttributes (atts, "VALUE", message);
      } else if (id.equals ("StatusCode")) {
        atts = Helpers.getAttributes (atts, "VALUE", statusCode);
      } else if (id.equals ("RequestURI")) {
        atts = Helpers.getAttributes (atts, "VALUE", requestURI);
      } else if (id.equals ("ExceptionType")) {
        atts = Helpers.getAttributes (atts, "VALUE", exceptionType);
      } else if (id.equals ("Exception")) {
        atts = Helpers.getAttributes (atts, "VALUE", exception);
      } // if
      return atts;
    } // getElementAttributes
    ...

  } // ErrorViewHandler
```

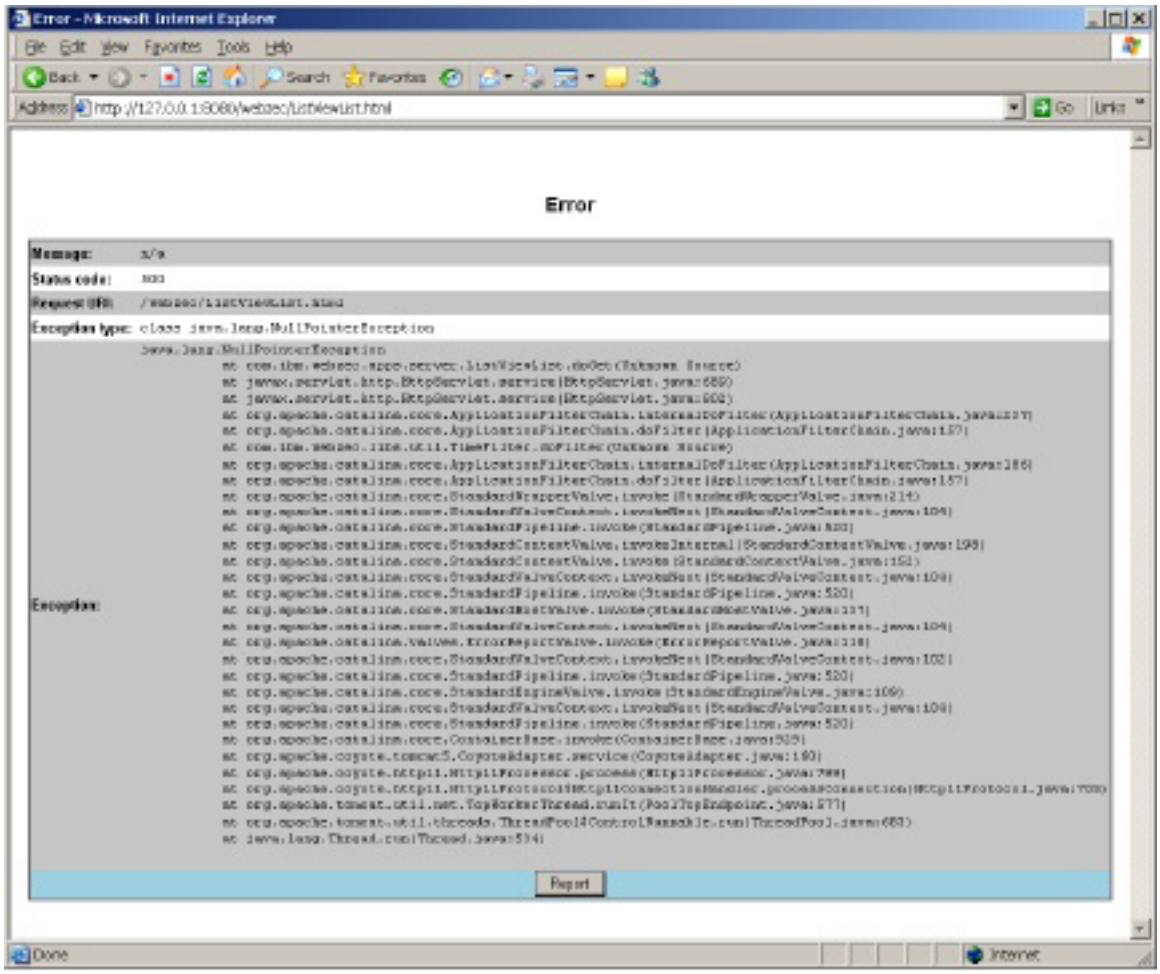
When a user clicks the **Report** button, the `doPost()` method is called. It retrieves the error information from the hidden input fields with `getParameter()`, composes an e-mail with this information, and fires it off by calling `Utilities.sendMail()`. It then redirects the user to the Logout.html page.

```
public class ErrorView extends Hamlet {

  ...
  public void doPost (HttpServletRequest req, HttpServletResponse res) throws
ServletException {
    try {
      category.debug ("doPost");
      // get properties
      ContextProperties props = ContextProperties.getProperties (this);
      // get mail info
      String host = props.getStringProperty ("MailHost");
      String from = props.getStringProperty ("MailSender");
      String to = props.getStringProperty ("MailReceiver");
      // get parameters
      String message = req.getParameter ("Message");
      String statusCode = req.getParameter ("StatusCode");
      String requestURI = req.getParameter ("RequestURI");
      String exceptionType = req.getParameter ("ExceptionType");
      String exception = req.getParameter ("Exception");
      // construct subject and text
      String subject = errorTitle;
      String text = getText (message, statusCode, requestURI, exceptionType,
exception);
      // log parameters
      category.debug ("Host: " + host);
      category.debug ("From: " + from);
      category.debug ("To: " + to);
      category.debug ("Subject: " + subject);
      category.debug ("Text: " + text);
      // send report
      Utilities.sendMail (host, from, to, subject, text);
```

```
      // logout
      res.sendRedirect (res.encodeURL (req.getContextPath () +
"/Logout.html"));
    } catch (Exception e) {
      category.error ("", e);
      throw new ServletException (e);
    } // try
  } // doPost
  ...

} // ErrorView
```

This completes the ErrorView example. Hopefully, you will rarely see ErrorView. Figure 13 displays a `NullPointerException` in ErrorView.

Figure 13. Display of a NullPointerException in ErrorView



View the complete ErrorView Hamlet code

Here is the complete code for the ErrorView example:

```
/**
 *
 * © Copyright International Business Machines Corporation 2004, 2006.
 * All rights reserved.
 *
 * The 'ErrorView' servlet provides a detailed error view.
```

```
 *
 * File     : ErrorView.java
 * Created  : 2004/12/06
 *
 * @author    Rene Pawlitzek  (rpa@zurich.ibm.com)
 * @version   1.00, 2006/11/27
 * @since     JDK 1.3
 *
 */


package com.ibm.webzec.apps.server;


import javax.servlet.*;
import javax.servlet.http.*;
import com.ibm.hamlet.*;
import com.ibm.hamlet.helpers.*;
import com.ibm.webzec.libs.util.*;
import org.apache.log4j.*;
import org.xml.sax.*;


public class ErrorView extends Hamlet {


  // log4j
  private static Category category = Category.getInstance
(ErrorView.class.getName ());


  private static final String  cr                 = "\n";
  private static final String  na                 = "n/a";
  private static final String  delimiter          = " : ";
  private static final String  errorTitle         = "WebZEC error";
  private static final String  messageLabel       = "Message";
  private static final String  statusCodeLabel    = "Status code";
  private static final String  requestURILabel    = "Request URI";
  private static final String  exceptionTypeLabel = "Exception type";
  private static final String  exceptionLabel     = "Exception";



  private static class ErrorViewHandler extends HamletHandler {


    private  String  message, statusCode, requestURI, exceptionType,
exception;


    public ErrorViewHandler (Hamlet hamlet, String message, String
statusCode,
      String requestURI, String exceptionType, String exception) {

      super (hamlet);
      this.message = message;
      this.statusCode = statusCode;
      this.requestURI = requestURI;
      this.exceptionType = exceptionType;
      this.exception = exception;

    } // ErrorViewHandler


    public String getElementReplacement (String id, String name, Attributes
atts) throws Exception {
      if (id.equals ("Message")) {
        return message;
      } else if (id.equals ("StatusCode")) {
        return statusCode;
      } else if (id.equals ("RequestURI")) {
        return requestURI;
      } else if (id.equals ("ExceptionType")) {
        return exceptionType;
```

```
      } else if (id.equals ("Exception")) {
        return exception;
      } // if
      return "?";
    } // getElementReplacement


    public Attributes getElementAttributes (String id, String name,
Attributes atts) throws Exception {
      if (id.equals ("Message")) {
        atts = Helpers.getAttributes (atts, "VALUE", message);
      } else if (id.equals ("StatusCode")) {
        atts = Helpers.getAttributes (atts, "VALUE", statusCode);
      } else if (id.equals ("RequestURI")) {
        atts = Helpers.getAttributes (atts, "VALUE", requestURI);
      } else if (id.equals ("ExceptionType")) {
        atts = Helpers.getAttributes (atts, "VALUE", exceptionType);
      } else if (id.equals ("Exception")) {
        atts = Helpers.getAttributes (atts, "VALUE", exception);
      } // if
      return atts;
    } // getElementAttributes


  } // ErrorViewHandler


  public void init () throws ServletException {
    // get properties
    ContextProperties props = ContextProperties.getProperties (this);
    // configure logging
    Utilities.configLog (props);
    category.debug ("init");
  } // init


  private String getText (String message, String statusCode, String
requestURI,
    String exceptionType, String exception) {

    StringBuffer buf = new StringBuffer ();
    buf.append (messageLabel);
    buf.append (delimiter);
    buf.append (message);
    buf.append (cr);
    buf.append (statusCodeLabel);
    buf.append (delimiter);
    buf.append (statusCode);
    buf.append (cr);
    buf.append (requestURILabel);
    buf.append (delimiter);
    buf.append (requestURI);
    buf.append (cr);
    buf.append (exceptionTypeLabel);
    buf.append (delimiter);
    buf.append (exceptionType);
    buf.append (cr);
    buf.append (exceptionLabel);
    buf.append (delimiter);
    buf.append (exception);
    buf.append (cr);
    return buf.toString ();

  } // getText


  public void doPost (HttpServletRequest req, HttpServletResponse res) throws
ServletException {
    try {
      category.debug ("doPost");
      // get properties
      ContextProperties props = ContextProperties.getProperties (this);
      // get mail info
      String host = props.getStringProperty ("MailHost");
```

```
      String from = props.getStringProperty ("MailSender");
      String to = props.getStringProperty ("MailReceiver");
      // get parameters
      String message = req.getParameter ("Message");
      String statusCode = req.getParameter ("StatusCode");
      String requestURI = req.getParameter ("RequestURI");
      String exceptionType = req.getParameter ("ExceptionType");
      String exception = req.getParameter ("Exception");
      // construct subject and text
      String subject = errorTitle;
      String text = getText (message, statusCode, requestURI, exceptionType,
exception);
      // log parameters
      category.debug ("Host: " + host);
      category.debug ("From: " + from);
      category.debug ("To: " + to);
      category.debug ("Subject: " + subject);
      category.debug ("Text: " + text);
      // send report
      Utilities.sendMail (host, from, to, subject, text);
      // logout
      res.sendRedirect (res.encodeURL (req.getContextPath () +
"/Logout.html"));
    } catch (Exception e) {
      category.error ("", e);
      throw new ServletException (e);
    } // try
  } // doPost


  public void doGet (HttpServletRequest req, HttpServletResponse res) throws
ServletException {
    try {
      category.debug ("doGet");
      // error message
      String message = na;
      Object obj = req.getAttribute ("javax.servlet.error.message");
      if (obj != null) message = obj.toString ();
      // status code
      String statusCode = na;
      obj = req.getAttribute ("javax.servlet.error.status_code");
      if (obj != null) statusCode = obj.toString ();
      // request URI
      String requestURI = req.getRequestURI ();
      obj = req.getAttribute ("javax.servlet.error.request_uri");
      if (obj != null) requestURI = obj.toString ();
      // exception type
      String exceptionType = na;
      obj = req.getAttribute ("javax.servlet.error.exception_type");
      if (obj != null) exceptionType = obj.toString ();
      // exception
      String exception = na;
      Throwable t = (Throwable) req.getAttribute
("javax.servlet.error.exception");
      if (t != null) exception = Utilities.getStackTraceAsString (t);
      // serve document
      HamletHandler handler = new ErrorViewHandler (this, message,
statusCode,
        requestURI, exceptionType, exception);
      serveDoc (req, res, "ErrorViewTemplate.html", handler);
    } catch (Exception e) {
      category.error ("", e);
      throw new ServletException (e);
    } // try
  } // doGet


  public String getServletInfo () {
    category.debug ("getServletInfo");
    return "ErrorView servlet";
  } // getServletInfo


  public void destroy () {
```

```
    category.debug ("destroy");
  } // destroy

} // ErrorView

/* ----- End of File ----- */
```

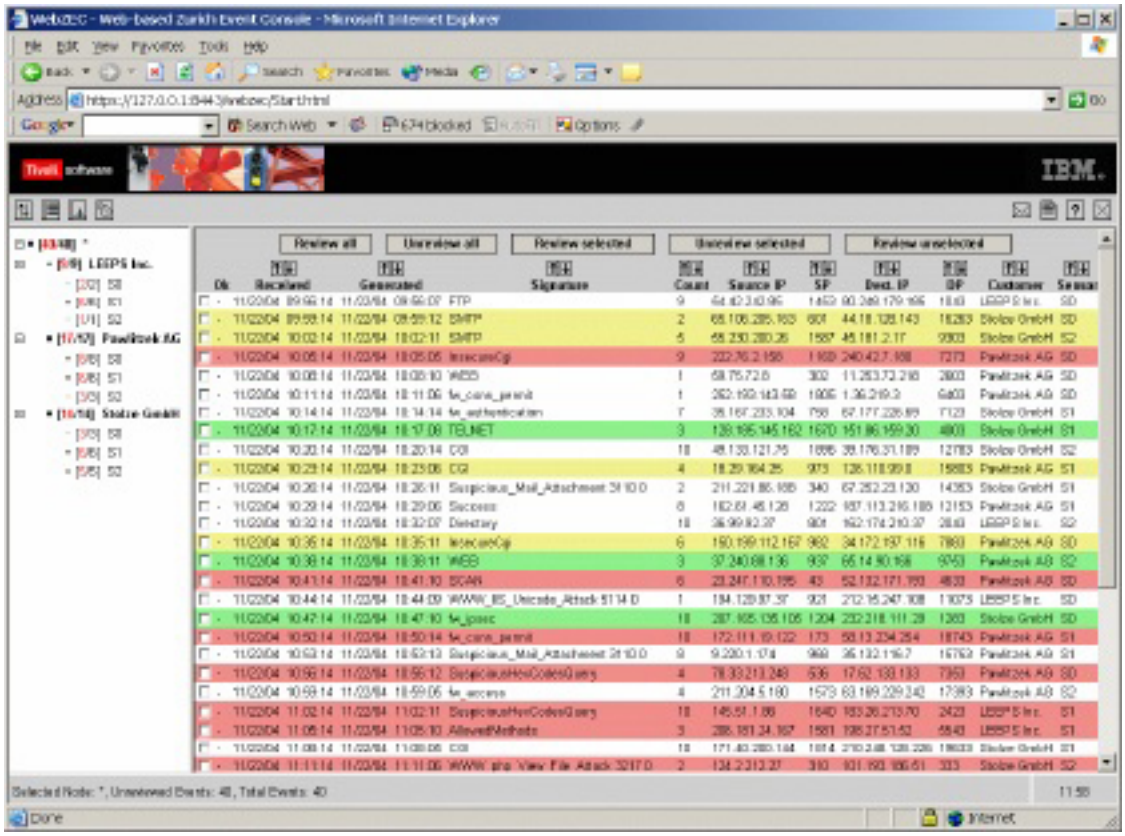
Section 9. Wrap up

The Hamlet framework offers complete separation of content and presentation, doesn't hide the familiar underlying servlet infrastructure, and is easy to use and to understand. The development of Hamlet-based Web applications is similar to the development of traditional stand-alone Windows applications. This allows Windows programmers to start immediately and produce results quickly. The resulting Web-based applications can be accelerated using a template compiler (see my article "Compiling Hamlets" for more information; there's a link in Resources).

With the help of Hamlets and a good understanding of the previous examples, writing your own Web-based applications should be a snap. If you choke on JSPs, you might want to try the Hamlet maneuver. Happy programming!

I leave you with a screenshot of WebZEC in action.

Figure 14. WebZEC in action



Section 10. Appendix A: Sample Hamlet implementation

Note that a detailed description of the source code listed below can be found in the article "Implementing Hamlets" (see Resources for a link).

ContentHandler class

```
/**
 *
 * © Copyright International Business Machines Corporation 2005, 2006.
 * All rights reserved.
 *
 * The 'ContentHandler' interface defines the functionality of a content
handler.
 *
 * File     : ContentHandler.java
 * Created  : 2005/08/16
 *
 * @author    Rene Pawlitzek (rpa@zurich.ibm.com)
 * @version   2.00, 2005/08/16
 * @since     JDK 1.3
 *
 * History  : 2005/08/16, rpa, new file
 *            2006/03/14, rpa, code review
 *
 */
```

```
package com.ibm.hamlet;


import java.io.*;
import org.xml.sax.*;


public interface ContentHandler {

  public int getElementRepeatCount (String id, String name, Attributes atts)
    throws Exception;

  public String getElementReplacement (String id, String name, Attributes
atts)
    throws Exception;

  public Attributes getElementAttributes (String id, String name, Attributes
atts)
    throws Exception;

  public InputStream getElementIncludeSource (String id, String name,
Attributes atts)
    throws Exception;

} // ContentHandler


/* ----- End of File ----- */
```

DefaultEngine class

```
/**
 *
 * © Copyright International Business Machines Corporation 2005, 2006.
 * All rights reserved.
 *
 * The 'DefaultEngine' class provides a default implementation of the
 * 'TemplateEngine' interface.
 *
 * File     : DefaultEngine.java
 * Created  : 2005/08/16
 *
 * @author    Rene Pawlitzek (rpa@zurich.ibm.com)
 * @version   2.00, 2005/08/16
 * @since     JDK 1.3
 *
 * History  : 2005/08/16, rpa, new file
 *            2005/08/16, rpa, introduced TemplateEngine interface
 *            2005/08/16, rpa, removed tree dependency on Swing
 *            2006/03/14, rpa, code review
 *
 */


package com.ibm.hamlet;


import java.io.*;
import java.util.*;
import org.apache.log4j.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;


public class DefaultEngine extends DefaultHandler implements TemplateEngine {
```

```
  // log4j
  private static Category category = Category.getInstance
(DefaultEngine.class);


  private static ReaderPool readerPool = new ReaderPool ();


  private static final String  REPEAT_TAG   = "REPEAT";     // repeat
  private static final String  INCLUDE_TAG  = "INCLUDE";    // include
  private static final String  REPLACE_TAG  = "REPLACE";    // replace
  private static final String  ID_ATTRIBUTE = "ID";         // id


  private  boolean         ignore, record;
  private  TreeNode        root, curNode;
  private  PrintWriter     out;
  private  ContentHandler  handler;


  public void perform (InputStream in, ContentHandler handler, PrintWriter
out)
    throws Exception {

    XMLReader reader = null;
    try {
      this.out = out;
      this.handler = handler;
      reader = readerPool.getReader ();
      InputSource inputSource = new InputSource (in);
      reader.setErrorHandler (this);
      reader.setContentHandler (this);
      reader.parse (inputSource);
    } finally {
      if (reader != null)
        readerPool.returnReader (reader);
    } // try

  } // perform


  private void playback (TreeNode node) throws Exception {
    int childIndex = node.getChildCount () - 1;
    for (int i = 0; i < node.getCount (); i++) {
      Object obj = node.elementAt (i);
      if (obj instanceof StartElement) {
        StartElement elem = (StartElement) obj;
        if (REPEAT_TAG.equals (elem.localName)) {
          int count = 0;
          String id = elem.atts.getValue (ID_ATTRIBUTE);
          if (id != null)
            count = handler.getElementRepeatCount (id, elem.localName,
elem.atts);
          for (int j = 0; j < count; j++) {
            playback ((TreeNode) node.getChildAt (childIndex));
            if (handler.getElementRepeatCount (id, elem.localName, elem.atts)
== 0)
              break;
          } // for
          childIndex--;
        } else
          startElement (elem.namespaceURI, elem.localName, elem.qName,
elem.atts);
      } else if (obj instanceof EndElement) {
        EndElement elem = (EndElement) obj;
        if (!REPEAT_TAG.equals (elem.localName))
          endElement (elem.namespaceURI, elem.localName, elem.qName);
      } else if (obj instanceof CharElement) {
        CharElement elem = (CharElement) obj;
        characters (elem.str.toCharArray (), 0, elem.str.length ());
      } // if
    } // for
  } // playback
```

```
  /* ----- private classes ----- */


  private static class TreeNode {

    private  TreeNode   parent = null;
    private  ArrayList  stack = new ArrayList ();
    private  ArrayList  children = new ArrayList ();

    void add (Object obj) {
      stack.add (obj);
    } // add

    Object elementAt (int index) {
      return stack.get (index);
    } // elementAt

    Object getChildAt (int index) {
      return children.get (index);
    } // getChildAt

    int getCount () {
      return stack.size ();
    } // getCount

    int getChildCount () {
      return children.size ();
    } // getChildCount

    public void insert (TreeNode node, int position) {
      node.parent = this;
      children.add (position, node);
    } // insert

    public TreeNode getParent () {
      return parent;
    } // getParent

  } // TreeNode


  private static class StartElement {

    public String     namespaceURI;
    public String     localName;
    public String     qName;
    public Attributes atts;

    public StartElement (String namespaceURI, String localName, String qName,
Attributes atts) {
      this.namespaceURI = namespaceURI;
      this.localName = localName;
      this.qName = qName;
      this.atts = new AttributesImpl (atts);
    } // StartElement

  } // StartElement


  private static class EndElement {

    public String namespaceURI;
    public String localName;
    public String qName;

    public EndElement (String namespaceURI, String localName, String qName) {
      this.namespaceURI = namespaceURI;
      this.localName = localName;
      this.qName = qName;
    } // EndElement

  } // EndElement
```

```
  private static class CharElement {

    public String str;

    public CharElement (String str) {
      this.str = str;
    } // CharElement

  } // CharElement


  /* ----- implementation of ContentHandler ----- */


  public void startDocument () throws SAXException {
    root = new TreeNode ();
    curNode = root;
  } // startDocument


  public void endDocument () throws SAXException {
    curNode = null;
    root = null;
  } // endDocument


  public void startElement (String namespaceURI, String localName, String
qName, Attributes atts)
    throws SAXException {

    try {
      // category.debug ("local name: " + localName + ", qname: " + qName);
      if (REPEAT_TAG.equals (localName)) {
        record = true;
        StartElement elem = new StartElement (namespaceURI, localName, qName,
atts);
        curNode.add (elem);
        TreeNode newNode = new TreeNode ();
        curNode.insert (newNode, 0);
        curNode = newNode;
      } else if (REPLACE_TAG.equals (localName)) {
        if (record) {
          StartElement elem = new StartElement (namespaceURI, localName,
qName, atts);
          curNode.add (elem);
        } else {
          ignore = true;
          String id = atts.getValue (ID_ATTRIBUTE);
          if (id != null)
            out.print (handler.getElementReplacement (id, localName, atts));
        } // if
      } else if (INCLUDE_TAG.equals (localName)) {
        if (record) {
          StartElement elem = new StartElement (namespaceURI, localName,
qName, atts);
          curNode.add (elem);
        } else {
          ignore = true;
          String id = atts.getValue (ID_ATTRIBUTE);
          if (id != null)
            atts = handler.getElementAttributes (id, localName, atts);
          InputStream in = handler.getElementIncludeSource (id, localName,
atts);
          RuntimeUtilities.printInclude (out, in);
        } // if
      } else {
        if (record) {
          StartElement elem = new StartElement (namespaceURI, localName,
qName, atts);
          curNode.add (elem);
        } else {
          String id = atts.getValue (ID_ATTRIBUTE);
          if (id != null)
```

```
            atts = handler.getElementAttributes (id, localName, atts);
          RuntimeUtilities.printTag (out, localName, atts);
        } // if
      } // if
    } catch (Exception e) {
      category.debug (e.getMessage (), e);
      throw new SAXException (e);
    } // try

  } // startElement


  public void endElement (String namespaceURI, String localName, String
qName)
    throws SAXException {

    try {
      if (REPEAT_TAG.equals (localName)) {
        curNode = (TreeNode) curNode.getParent ();
        EndElement elem = new EndElement (namespaceURI, localName, qName);
        curNode.add (elem);
        if (curNode.equals (root)) {
          record = false;
          playback (root);
          root = new TreeNode ();
          curNode = root;
        } // if
      } else if (REPLACE_TAG.equals (localName) || INCLUDE_TAG.equals
(localName)) {
        if (record) {
          EndElement elem = new EndElement (namespaceURI, localName, qName);
          curNode.add (elem);
        } else {
          ignore = false;
        } // if
      } else {
        if (record) {
          EndElement elem = new EndElement (namespaceURI, localName, qName);
          curNode.add (elem);
        } else {
          out.print ("</");
          out.print (localName);
          out.print (">");
        } // if
      } // if
    } catch (Exception e) {
      category.debug (e.getMessage (), e);
      throw new SAXException (e);
    } // try

  } // endElement


  public void characters (char[] ch, int start, int length) throws
SAXException {
    String str = new String (ch, start, length);
    if (record) {
      CharElement elem = new CharElement (str);
      curNode.add (elem);
    } else {
      if (!ignore)
        out.print (str);
    } // if
  } // characters


  /* ----- implementation of ErrorHandler ----- */


  private void logError (String str, SAXParseException e) {
    category.error (str + " (Line: " + e.getLineNumber () + ", Column: "
      + e.getColumnNumber () + "): " + e.getMessage ());
  } // logError
```

```
  public void warning (SAXParseException e) throws SAXException {
    logError ("Warning", e);
    throw e;
  } // warning


  public void error (SAXParseException e) throws SAXException {
    logError ("Error", e);
    throw e;
  } // error


} // DefaultEngine


/* ----- End of File ----- */
```

Hamlet class

```
/**
 *
 * © Copyright International Business Machines Corporation 2004, 2006.
 * All rights reserved.
 *
 * The 'Hamlet' class provides an infrastructure to separate content from
 * presentation. Extend from this class to serve documents. Overwrite the
 * 'getElementRepeatCount', 'getElementReplacement', 'getElementAttributes',
 * and 'getElementIncludeSource' methods in order to provide dynamic content.
 * For more information see:
 *
 * http://www.ibm.com/developerworks/web/library/wa-hamlets/
 *
 *
 * File     : Hamlet.java
 * Created  : 2004/05/19
 *
 * @author    Rene Pawlitzek (rpa@zurich.ibm.com)
 * @version   2.00, 2004/05/19
 * @since     JDK 1.3
 *
 * History  : 2004/05/19, rpa, new file
 *            2004/08/31, rpa, code review
 *            2005/02/03, rpa, reduced number of callback parameters
 *            2005/02/07, rpa, code review
 *            2005/02/16, rpa, added support for includes
 *            2005/08/16, rpa, separated template engine code
 *            2005/08/16, rpa, introduced getElementIncludeSource
 *            2005/08/17, rpa, code review
 *            2006/03/09, rpa, added findTemplateClass
 *            2006/03/14, rpa, code review
 *            2006/07/11, rpa, added session id for includes
 *
 */


package com.ibm.hamlet;


import java.io.*;
import java.net.*;
import javax.servlet.http.*;
import org.apache.log4j.*;
import org.xml.sax.*;


public abstract class Hamlet extends HttpServlet implements ContentHandler {
```

```
  // log4j
  private static Category category = Category.getInstance (Hamlet.class);


  private  int       port;
  private  String    sid, path, oldTemplate;
  private  Template  templateClass;


  public int getElementRepeatCount (String id, String name, Attributes atts)
    throws Exception {
    return 0;
  } // getElementRepeatCount


  public String getElementReplacement (String id, String name, Attributes
atts)
    throws Exception {
    return "";
  } // getElementReplacement


  public Attributes getElementAttributes (String id, String name, Attributes
atts)
    throws Exception {
    return atts;
  } // getElementAttributes


  public InputStream getElementIncludeSource (String id, String name,
Attributes atts)
    throws Exception {
    URL url = getIncludeURL (atts.getValue ("SRC"));
    return url.openStream ();
  } // getElementIncludeSource


  public String getDocumentType () {
    return "text/html";
  } // getDocumentType


  public URL getIncludeURL (String fileName) throws Exception {
    StringBuffer buf = new StringBuffer (path);
    buf.append ("/");
    int pos = fileName.indexOf ("?");
    if (pos != -1) {
      buf.append (fileName.substring (0, pos));
      buf.append (sid);
      buf.append (fileName.substring (pos));
    } else {
      buf.append (fileName);
      buf.append (sid);
    } // if
    category.debug ("Include URL: " + buf.toString ());
    return new URL ("http", "localhost", port, buf.toString (), null);
  } // getIncludeURL


  public TemplateEngine getTemplateEngine () {
    return new DefaultEngine ();
  } // getTemplateEngine


  private void initialize (HttpServletRequest req) {
    sid = "";
    HttpSession session = req.getSession (false);
    if (session != null)
      sid = ";jsessionid=" + session.getId ();
    port = req.getServerPort ();
    path = req.getContextPath ();
  } // initialize
```

```
  private void findTemplateClass (String template) throws Exception {
    if (!template.equals (oldTemplate)) {
      String className = RuntimeUtilities.getClassName (template);
      try {
        category.debug ("Loading class '" + className + "' ...");
        Class c = Class.forName (className);
        templateClass = (Template) c.newInstance ();
        category.debug ("Class '" + className + "' loaded");
        System.out.println ("Class '" + className + "' loaded");
      } catch (ClassNotFoundException e) {
        category.debug ("Cannot load class '" + className + "'");
      } // try
      oldTemplate = template;
    } // if
  } // findTemplateClass


  private void serveDoc (PrintWriter out, String template, ContentHandler
handler)
    throws Exception {

    findTemplateClass (template);
    if (templateClass != null) {
      templateClass.serveDoc (out, handler);
    } else {
      TemplateEngine engine = getTemplateEngine ();
      InputStream in = getServletContext ().getResourceAsStream (template);
      category.debug ("Parsing '" + template + "' ...");
      long t1 = System.currentTimeMillis ();
      engine.perform (in, handler, out);
      long t2 = System.currentTimeMillis ();
      category.debug ("Parsed '" + template + "' in " + (t2 - t1) + " ms.");
    } // if

  } // serveDoc


  public void serveDoc (HttpServletRequest req, HttpServletResponse res,
    String template, ContentHandler handler) throws Exception {
    initialize (req);
    PrintWriter out = res.getWriter ();
    res.setContentType (getDocumentType ());
    serveDoc (out, template, handler);
  } // serveDoc


  public void serveDoc (HttpServletRequest req, HttpServletResponse res,
    String template) throws Exception {
    serveDoc (req, res, template, this);
  } // serveDoc


  public void serveDoc (HttpServletRequest req, HttpServletResponse res,
    Class c, ContentHandler handler) throws Exception {
    initialize (req);
    PrintWriter out = res.getWriter ();
    res.setContentType (getDocumentType ());
    Template templateClass = (Template) c.newInstance ();
    templateClass.serveDoc (out, handler);
  } // serveDoc


} // Hamlet


/* ----- End of File ---- */
```

HamletHandler class

```
/**
 *
 * © Copyright International Business Machines Corporation 2005, 2006.
 * All rights reserved.
 *
 * The 'HamletHandler' class provides a default implementation of the
 * 'ContentHandler' interface.
 *
 * File     : HamletHandler.java
 * Created  : 2005/08/16
 *
 * @author    Rene Pawlitzek (rpa@zurich.ibm.com)
 * @version   2.00, 2005/08/16
 * @since     JDK 1.3
 *
 * History  : 2005/08/16, rpa, new file
 *            2006/03/14, rpa, code review
 *
 */


package com.ibm.hamlet;


import java.io.*;
import java.net.*;
import org.xml.sax.*;


public class HamletHandler implements ContentHandler {


  private  Hamlet  hamlet;


  public HamletHandler (Hamlet hamlet) {
    this.hamlet = hamlet;
  } // HamletHandler


  public int getElementRepeatCount (String id, String name, Attributes atts)
    throws Exception {
    return 0;
  } // getElementRepeatCount


  public String getElementReplacement (String id, String name, Attributes
atts)
    throws Exception {
    return "";
  } // getElementReplacement


  public Attributes getElementAttributes (String id, String name, Attributes
atts)
    throws Exception {
    return atts;
  } // getElementAttributes


  public InputStream getElementIncludeSource (String id, String name,
Attributes atts)
    throws Exception {
    URL url = hamlet.getIncludeURL (atts.getValue ("SRC"));
    return url.openStream ();
  } // getElementIncludeSource


} // HamletHandler


/* ----- End of File ----- */
```

ReaderPool class

```
/**
 *
 * © Copyright International Business Machines Corporation 2004, 2006.
 * All rights reserved.
 *
 * The 'ReaderPool' class provides a pooling mechanism for XML readers.
 *
 * File     : ReaderPool.java
 * Created  : 2004/09/08
 *
 * @author    Rene Pawlitzek  (rpa@zurich.ibm.com)
 * @version   2.00, 2004/09/08
 * @since     JDK 1.3
 *
 * History  : 2004/09/08, rpa, new file
 *            2006/03/14, rpa, code review
 *
 */


package com.ibm.hamlet;


import java.util.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;


public class ReaderPool {


  private Hashtable readers = new Hashtable ();


  public ReaderPool () {
    String className = System.getProperty ("org.xml.sax.parser");
    if (className == null)
      System.setProperty ("org.xml.sax.parser",
"org.apache.xerces.parsers.SAXParser");
  } // ReaderPool


  public synchronized XMLReader getReader () throws Exception {
    Enumeration enum = readers.keys ();
    while (enum.hasMoreElements ()) {
      XMLReader reader = (XMLReader) enum.nextElement ();
      Boolean b = (Boolean) readers.get (reader);
      if (b.equals (Boolean.FALSE)) {
        readers.put (reader, Boolean.TRUE);
        return reader;
      } // if
    } // while
    XMLReader reader = XMLReaderFactory.createXMLReader ();
    readers.put (reader, Boolean.TRUE);
    return reader;
  } // getReader


  public synchronized void returnReader (XMLReader aReader) {
    if (readers.containsKey (aReader))
      readers.put (aReader, Boolean.FALSE);
  } // returnReader


} // ReaderPool


/* ----- End of File ----- */
```

RuntimeUtilities class

```
/**
 *
 * © Copyright International Business Machines Corporation 2006.
 * All rights reserved.
 *
 * The 'RuntimeUtilities' class provides runtime utilities.
 *
 * File     : RuntimeUtilities.java
 * Created  : 2006/03/09
 *
 * @author    Rene Pawlitzek (rpa@zurich.ibm.com)
 * @version   2.00, 2006/03/09
 * @since     JDK 1.3
 *
 * History  : 2006/03/09, rpa, new file
 *            2006/03/14, rpa, code review
 *
 */


package com.ibm.hamlet;


import java.io.*;
import org.xml.sax.*;


public class RuntimeUtilities {


  public static void printInclude (PrintWriter out, InputStream in) throws
Exception {
    if (in == null)
      return;
    BufferedReader r = new BufferedReader (new InputStreamReader (in));
    String line = r.readLine ();
    while (line != null) {
      out.print (line);
      line = r.readLine ();
      if (line != null)
        out.println ();
    } // while
    r.close ();
    in.close ();
  } // printInclude


  public static void printTag (PrintWriter out, String name, Attributes atts)
{
    out.print ("<");
    out.print (name);
    for (int i = 0; i < atts.getLength (); i++) {
      out.print (" ");
      out.print (atts.getLocalName (i));
      out.print ("=\"");
      out.print (atts.getValue (i));
      out.print ("\"");
    } // for
    out.print (">");
  } // printTag


  public static String getClassName (String templateName) {
    int pos = templateName.lastIndexOf (".");
    if (pos != -1)
      templateName = templateName.substring (0, pos);
    return templateName;
  } // getClassName
```

```
} // RuntimeUtilities


/* ----- End of File ----- */
```

Template class

```
/**
 *
 * © Copyright International Business Machines Corporation 2006.
 * All rights reserved.
 *
 * The 'Template' interface defines the functionality of a compiled template.
 *
 * File     : Template.java
 * Created  : 2006/03/09
 *
 * @author    Rene Pawlitzek (rpa@zurich.ibm.com)
 * @version   2.00, 2006/03/09
 * @since     JDK 1.3
 *
 * History  : 2006/03/09, rpa, new file
 *            2006/03/14, rpa, code review
 *
 */


package com.ibm.hamlet;

import java.io.*;

public interface Template {

  public void serveDoc (PrintWriter writer, ContentHandler handler) throws
Exception;

} // Template


/* ----- End of File ----- */
```

TemplateEngine class

```
/**
 *
 * © Copyright International Business Machines Corporation 2005, 2006.
 * All rights reserved.
 *
 * The 'TemplateEngine' interface defines the functionality of a template
engine.
 *
 * File     : TemplateEngine.java
 * Created  : 2005/08/16
 *
 * @author    Rene Pawlitzek (rpa@zurich.ibm.com)
 * @version   2.00, 2005/08/16
 * @since     JDK 1.3
 *
 * History  : 2005/08/16, rpa, new file
 *            2006/03/14, rpa, code review
 *
 */
```

```
package com.ibm.hamlet;

import java.io.*;

public interface TemplateEngine {

  public void perform (InputStream in, ContentHandler handler, PrintWriter
out) throws Exception;

} // TemplateEngine


/* ----- End of File ----- */
```

Section 11. Appendix B: Utility classes

ContextProperties class

```
/**
  *
  * © Copyright International Business Machines Corporation 2004, 2005.
  * All rights reserved.
  *
  * The 'ContextProperties' class collects all context init parameters with
  * a Properties class.
  *
  * Code from page 546 of Java Servlet Programming copyright © 2001 by
  * O'Reilly & Associates, Inc. Reprinted with permission of O'Reilly Media.
  *
  * File     : ContextProperties.java
  * Created  : 2004/03/16
  *
  * @author    Rene Pawlitzek  (rpa@zurich.ibm.com)
  * @version   1.00, 2004/03/16
  * @since     JDK 1.3
  *
  * History  : 2004/03/16, rpa, new file
  *            2004/08/31, rpa, code review
  */


package com.ibm.webzec.libs.util;


import java.util.*;
import javax.servlet.*;
import org.apache.log4j.*;


public class ContextProperties extends Properties {


  // log4j
  private static Category category =
    Category.getInstance (ContextProperties.class.getName ());


  private  static  ContextProperties  properties;


  public ContextProperties (ServletContext context) {
```

```
    Enumeration props = context.getInitParameterNames ();
    while (props.hasMoreElements ()) {
      String name = (String) props.nextElement ();
      String value = (String) context.getInitParameter (name);
      put (name, value);
    } // while
  } // ContextProperties


  public String getStringProperty (String aName) throws Exception {
    String val = getProperty (aName);
    if (val == null)
      throw new Exception ("Parameter " + aName + " not defined");
    else if (val.length () == 0)
      throw new Exception ("Parameter " + aName + " is empty");
    else
      return val;
  } // getStringProperty


  public String getStringProperty (String aName, String aDef) {
    try {
      return getStringProperty (aName);
    } catch (Exception e) {
      return aDef;
    } // try
  } // getStringProperty


  public int getIntProperty (String aName) throws Exception {
    return Integer.parseInt (getStringProperty (aName));
  } // getIntProperty


  public int getIntProperty (String aName, int aDef) {
    try {
      return getIntProperty (aName);
    } catch (Exception e) {
      return aDef;
    } // try
  } // getIntProperty


  public boolean getBooleanProperty (String aName) throws Exception {
    String val = getStringProperty(aName).toLowerCase ();
    if ((val.equalsIgnoreCase ("true")) ||
        (val.equalsIgnoreCase ("on")) ||
        (val.equalsIgnoreCase ("yes"))) {
      return true;
    } else if ((val.equalsIgnoreCase ("false")) ||
               (val.equalsIgnoreCase ("off")) ||
               (val.equalsIgnoreCase ("no"))) {
      return false;
    } else {
      throw new Exception ("Parameter " + aName + " is not a boolean");
    } // if
  } // getBooleanProperty


  public boolean getBooleanProperty (String aName, boolean aDef) {
    try {
      return getBooleanProperty (aName);
    } catch (Exception e) {
      return aDef;
    } // try
  } // getBooleanProperty


  public void debug () {
    Enumeration enumeration = keys ();
    category.debug ("Properties: ");
    while (enumeration.hasMoreElements ()) {
      String name = (String) enumeration.nextElement ();
      String value = getProperty (name);
```

```
      category.debug ("  " + name + " := " + value);
    } // while
  } // debug


  /* ----- static methods ----- */


  public static synchronized ContextProperties getProperties (GenericServlet
aServlet) {
    ServletContext context = aServlet.getServletContext ();
    if (properties == null)
      properties = new ContextProperties (context);
    return properties;
  } // getProperties


} // ContextProperties


/* ----- End of File ----- */
```

Utilities class

```
/**
 *
 * © Copyright International Business Machines Corporation 2004, 2005.
 * All rights reserved.
 *
 * The 'Utilities' class provides utility functions.
 *
 * File     : Utilities.java
 * Created  : 2004/03/16
 *
 * @author    Rene Pawlitzek  (rpa@zurich.ibm.com)
 * @version   1.00, 2004/03/16
 * @since     JDK 1.3
 *
 * History  : 2004/03/16, rpa, new file
 *            2004/08/31, rpa, code review
 */


package com.ibm.webzec.libs.util;


import java.io.*;
import java.util.*;
import javax.mail.*;
import javax.mail.internet.*;
import javax.servlet.http.*;
import org.apache.log4j.*;


public class Utilities {


  // log4j
  private static Category category = Category.getInstance
(Utilities.class.getName ());


  private static boolean initLog = false;


  public static String getStackTraceAsString (Throwable t) {
    ByteArrayOutputStream bytes = new ByteArrayOutputStream ();
    PrintWriter writer = new PrintWriter (bytes, true);
    t.printStackTrace (writer);
```

```
    return bytes.toString ();
  } // getStackTraceAsString


  public static TimeZone getTimeZone (String aID) {
    try {
      return TimeZone.getTimeZone (aID);
    } catch (Exception e) {
      return TimeZone.getDefault ();
    } // try
  } // getTimeZone


  public static void sendMail (String aMailHost, String aFrom, String aTo,
    String aSubject, String aText) throws Exception {

    Properties props = new Properties ();
    props.put ("mail.smtp.host", aMailHost);
    Session session = Session.getDefaultInstance (props, null);
    MimeMessage message = new MimeMessage (session);
    message.setFrom (new InternetAddress (aFrom));
    message.addRecipient (Message.RecipientType.TO, new InternetAddress
(aTo));
    message.setSubject (aSubject);
    message.setText (aText);
    Transport.send (message);

  } // sendMail


  public static void debugRequestParams (HttpServletRequest aRequest) {
    Enumeration names = aRequest.getParameterNames ();
    category.debug ("Request Parameters: ");
    while (names.hasMoreElements ()) {
      String name = (String) names.nextElement ();
      String[] values = (String[]) aRequest.getParameterValues (name);
      StringBuffer buf = new StringBuffer ();
      for (int i = 0; i < values.length; i++) {
        buf.append (values[i]);
        if (i != (values.length - 1))
          buf.append (", ");
      } // for
      category.debug ("  " + name + " = " + buf.toString ());
    } // while
  } // debugRequestParams


  public static void debugSessionAttrs (HttpSession aSession) {
    Enumeration names = aSession.getAttributeNames ();
    category.debug ("Session Attributes: ");
    while (names.hasMoreElements ()) {
      String name = (String) names.nextElement ();
      Object value = aSession.getAttribute (name);
      category.debug ("  " + name + " = " + value.toString ());
    } // while
  } // debugSessionAttrs


  public static boolean booleanValueOf (String str) {
    Boolean b = Boolean.valueOf (str);
    return b.booleanValue ();
  } // booleanValueOf


  public synchronized static void configLog (Properties aProps) {
    if (!initLog) {
      PropertyConfigurator.configure (aProps);
      initLog = true;
    } // if
  } // configLog


} // Utilities
```

```
/* ----- End of File ----- */
```

Resources

Learn

- Get started with "Introducing Hamlets," René Pawlitzek (developerWorks, March 2005).
- Understand how Hamlets are implemented with "Implementing Hamlets," René Pawlitzek (developerWorks, February 2006).
- Accelerate Hamlets by reading "Compiling Hamlets," René Pawlitzek (developerWorks, June 2006).
- Read Sun's Java Servlet API White Paper to get started.
- Learn more from *Java Servlet Programming,* J. Hunter and W. Crawford (O'Reilly, 2001).
- Get the facts from the Cascading Style Sheets home page at the W3C.
- Improve your knowledge with *Cascading Style Sheets: The Definitive Guide,* E. Meyer (O'Reilly, 2004).
- Then, get smarter with the *CSS Cookbook,* C. Schmitt (O'Reilly, 2004).
- Check out the HyperText Markup Language (HTML) home page at the W3C.
- Dig deeper with *HTML and XHTML: The Definitive Guide,* C. Musciano and B. Kennedy (O'Reilly, 2002).
- Learn more by reading *Ant: The Definitive Guide,* J.E. Tilly and E.M. Burke (O'Reilly, 2002).
- Read up in *Tomcat: The Definitive Guide,* J. Brittain and I.F. Darwin (O'Reilly, 2003).
- Find hundreds more Web development resources on the developerWorks Web development zone.

Get products and technologies

- Check out the official Web site for SAX.
- More important information is at the Xerces home page.
- Get info from the Apache Ant Project home page.
- Start at the Apache Jakarta Tomcat home page.
- Check out the log4j home page.
- Check out Hamlets on alphaWorks, where the project is called the IBM servlet-based content creation framework.

About the author

René Pawlitzek
René Pawlitzek is a citizen of Liechtenstein and holds an engineering degree in computer science from the Swiss Federal Institute of Technology (ETH Zürich). René works as a research and development engineer focusing on security information management solutions for the Advanced Operating Environment group (formerly Global Security Analysis Lab (GSAL)) at the IBM Zürich Research Laboratory in Switzerland. Before coming to IBM, he worked in California for Hewlett-Packard, WindRiver Systems, and Borland International.