NetBeans IDE 5.5 Tutorial for Web Applications

The Midnight Cookie Company application that you will build in this tutorial will show you how to do the following:

- Create a web project from existing sources
- Create a front controller servlet
- Create methods to process requests and create cookies
- Create the JSP page that receives the requests
- Use JSTL to internationalize the application
- Use the IDE's HTTP Monitor to analyze your deployed application

Below are explanations of some of the terms used in this tutorial.

- **Composite View.** A design pattern that is used to present information in JSP pages. This design pattern creates an aggregate view from component views. Component views might include dynamic and modular portions of the page. The Composite View design pattern pertains to web application design when you are creating a view from numerous subviews. Complex web pages frequently consist of content derived from various resources. The layout of the page is managed independently of the content of its subviews. For instance, a view might have subviews such as Navigation, Search, Feature Story, and Headline.
When creating a composite view, you can include static content and dynamic content. Static content might consist of an HTML file. Dynamic content is a fragment of a JSP page. The specific content can be determined at either JSP translation time or at runtime.

- **Front controllers.** Components that are responsible for routing incoming requests and enforcing navigation in web applications. For more information on the use of the Front Controller design pattern, see the J2EE Patterns catalog.

- **JSP Pages.** Files that are used in web applications to present information to end users and to enable data from end users to flow back to the server. JSP pages must be placed within a web application in order for the JSP pages to be executable within the IDE.

- **Servlets.** Java classes that execute within a servlet container, which is typically running within a web server. Servlets are used to do the following:
    - Generate dynamic content.
    - Extend the capabilities of web servers and web-enabled application servers.
    - Interact with web clients (typically a browser application such as Netscape or Internet Explorer) using a request-response paradigm.

This application uses the JavaServer Pages Standard Tag Library (JSTL) to fetch dynamic data and to internationalize the JSP pages used in this application.

As you build the Midnight Cookie Company application, this tutorial will walk you through the whole web development cycle in the IDE.

This tutorial should take approximately two hours to complete.

Setting Up Your Environment

Before you start writing code, you have to make sure you have all of the necessary software and that your project is set up correctly.

Installing the Software

Before you begin, you need to install the following software on your computer:

- NetBeans IDE 5.5 (download).
- Java Standard Development Kit (download) version 5 or 6.

Optionally, you can download and use the Sun Java System Application Server (download), JBoss, or WebLogic. However, the Tomcat Web Server that is bundled with the IDE provides all the support you need for two-tier web applications such as the one described in this quick start guide. An application server (such as the Sun Java System Application Server, JBoss, or WebLogic) is only required when you want to develop enterprise applications.

Registering the Server with the IDE

The bundled Tomcat Web Server is registered with the IDE automatically. However, before you can deploy to the Sun Java System Application Server, JBoss, or WebLogic, you have to register a local instance with the IDE. If you installed the NetBeans IDE 5.5/Sun Java System Application Server bundle, a local instance of the Sun Java System Application Server is registered automatically. Otherwise, take the following steps:

1. Choose Tools > Server Manager from the main window.
2. Click Add Server. Select the server type and give a name to the instance. Then click Next.
3. Specify the server information, the location of the local instance of the application server, and the domain to which you want to deploy.

Obtaining the required source files

To create the Midnight Cookie Company application, you need the `midnight.zip` file, which contains the source files on which you build the application. The source files include the `WEB-INF` folder, which contains the `classes`, `docs`, `tlds` and `lib` folders that you will use in this tutorial.

1. In your file system, create a folder for the unzipped application files. From now on, we will refer to this folder as *$UNZIPHOME*.
2. Click here to download the file.
3. Use an application that unzips files to unzip the `midnight.zip` file to *$UNZIPHOME*. *$UNZIPHOME* now has a `WEB-INF` folder, which contains the following files and folders:

File or Folder	Description
`web.xml`	Deployment descriptor for the web application.
`classes`	Contains the resource bundle property files that are used to internationalize the pages and contains the `com/midnightcookies/taghandlers` folder. The `taghandlers` folder contains the class files for the ContentHandler tag handler, the ButtonHandler tag handler, and the LinksHandler tag handler. The ContentHandler includes the contents of the specified JSP file. The ButtonHandler prints a submit button with a localized message. The LinksHandler displays a set of links, including a separator, from the tag attribute.
`docs`	Contains the `header.jsp` file, the `index.jsp` file, and the `nonesuch.jsp` file. During the tutorial, you will create the `main.jsp` page and place it in this folder. Also contains the `cookies` folder, which includes the `CookieCutter.jsp` file and the `CookieMake.jsp` file. During the tutorial, you will create the `Tray.jsp` page and place it in this folder. The `about` folder contains the `about.jsp` file, which outlines this web application's architecture.
`lib`	Contains the `standard.jar` file and the `jstl.jar` file. These two files contain the implementation for the JSTL. The `standard.jar` contains the TLDs and the tag handlers. The `jstl.jar` file contains other application program interfaces (APIs) that are needed.
`tlds`	Contains the proprietary `midnight.tld`.

Developing the Application

Creating a web project from the provided source files

1. Choose File > New Project (Ctrl-Shift-N). Under Categories, select Web. Under Projects, select Web Project with Existing Sources. Click Next.
2. In Location, click Browse to select the web application's document root. The document root is *$UNZIPHOME*, the folder to which you unzipped the `midnight.zip` file.
3. In Project Name, type `Midnight`. Change the Project Location to any folder on your computer. From now on, we will refer to this folder as *$PROJECTHOME*.
4. In Server, select the server.

5. In J2EE Specification Level, make sure that J2EE 1.4 is selected. Notice that the Context Path is `/Midnight`. Click Next.
6. In Web Pages Folder, select the document root (`$UNZIPHOME`). Notice that the `$PROJECTHOME/Midnight/WEB-INF/lib` and the `$PROJECTHOME/Midnight/WEB-INF/classes` folders are automatically selected as your lib and sources folders, respectively. Click Finish.
   You are prompted to let the IDE delete your class files, because they will automatically be compiled to your project's `build` folder when you build the project. Click Delete to let the IDE do this.

   The IDE imports your sources, deletes the class files, and creates the Midnight project in the IDE. You can view its logical structure in the Projects window and its file structure in the Files window.

Creating a front controller using a servlet

A front controller is responsible for routing incoming user requests and can enforce navigation in web applications. A front controller provides a single entry point through which requests for several resources in a web application pass. A front controller can reduce duplication of code in JSP pages, especially in cases where several resources require the same processing. For more information on front controllers, see the J2EE Patterns catalog.

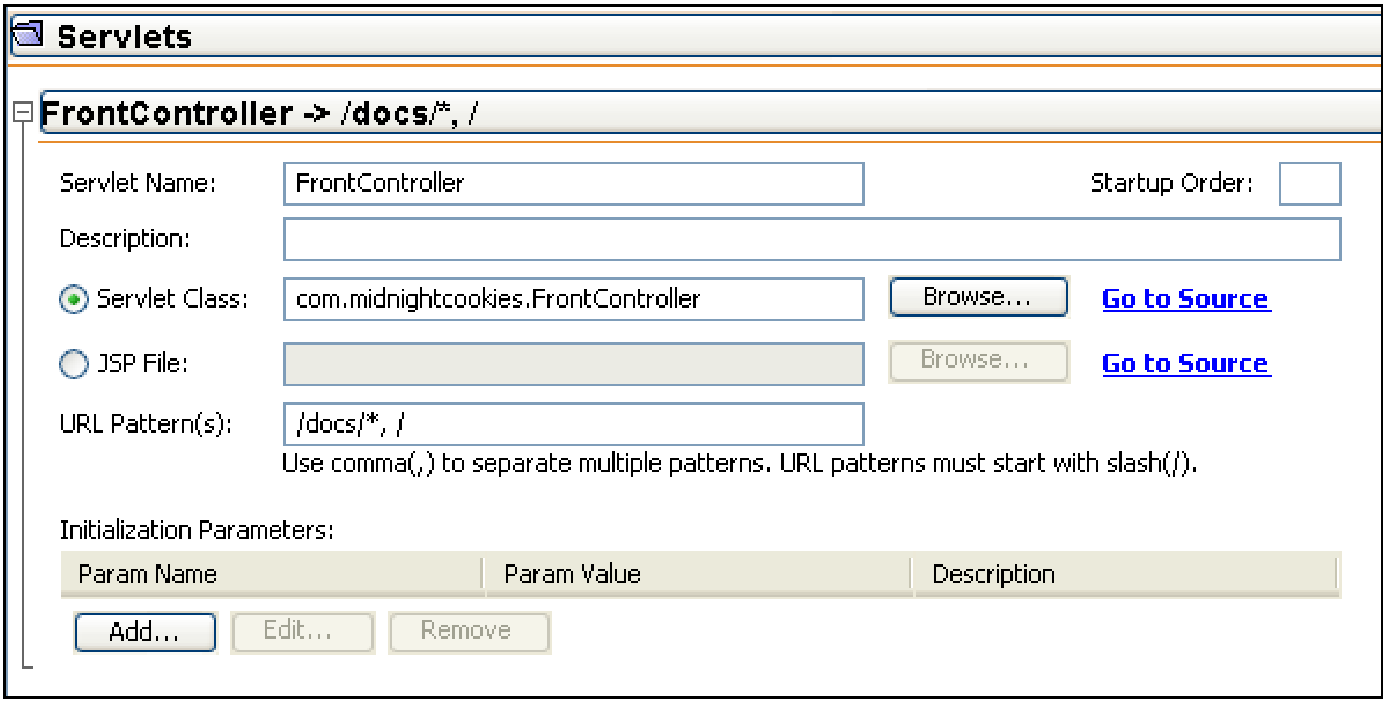
1. Expand the Midnight project node and the Source Packages node.
2. Right-click the `com.midnightcookies` package node and choose New > Servlet.
3. Enter `FrontController` in the Class Name text box and make sure that the `com.midnightcookies` package is selected in the Package drop-down. Click Next.
4. Change the value of the URL Pattern(s) text box to the following:

   ```
   /docs/*, /
   ```

   Note that the "`,`" in the line above is a delimiter between URL mappings. It enables you to create multiple URL mappings for each servlet.

5. Click Finish.
   `FrontController.java` opens in the Source Editor.

6. Expand the Web Pages node and the WEB-INF node. Double-click the `web.xml` node. The content of the deployment descriptor file, `web.xml`, is displayed in the graphical editor. Click Servlets at the top of the editor. Expand the node for the FrontController servlet's entry. Notice the servlet's URL Patterns, which you defined in step 4 above:



Servlets

FrontController -> /docs/*, /

Servlet Name: FrontController

Startup Order:

Description:

● Servlet Class: com.midnightcookies.FrontController

○ JSP File:

URL Pattern(s): /docs/*, /

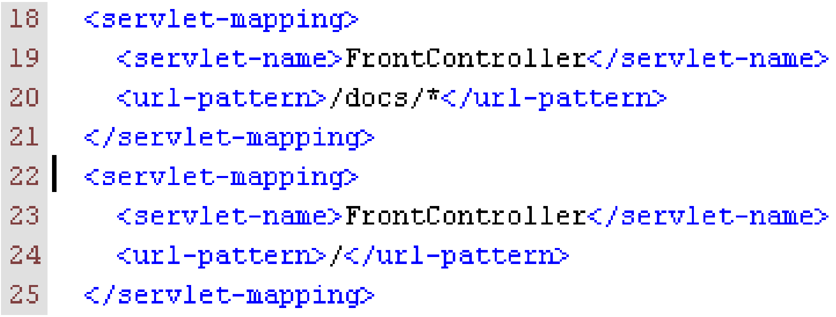
Use comma(,) to separate multiple patterns. URL patterns must start with slash(/).

Initialization Parameters:

Param Name	Param Value	Description

The servlet mapping "`/docs/*`" prevents the FrontController servlet from forwarding pages to itself, while the servlet mapping "`/`" gives the FrontController servlet a servlet mapping to handle a request such as `http://`*`hostname:portnumber`*`/midnight/`.

Click XML at the top of the editor. The content of the deployment descriptor file is shown in XML format. The servlet mappings are as follows:



```
<servlet-mapping>
  <servlet-name>FrontController</servlet-name>
  <url-pattern>/docs/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>FrontController</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

Adding the methods to process requests and to create a cookie

Now you will edit the `FrontController.java` file to include the logic needed to process the requests received by the web application. You will modify the `processRequest` method and create a new method called `createCookie`, which demonstrates how to create a cookie used in a web application.

1. In the Projects window, expand the FrontController.java node and the FrontController class node. The Fields node, Constructors node, Methods node, and Bean Patterns node are displayed.
2. Expand the Methods node and double-click the processRequest node. The Source Editor displays the `processRequest` method.
3. Replace the default `processRequest` method with the following code:

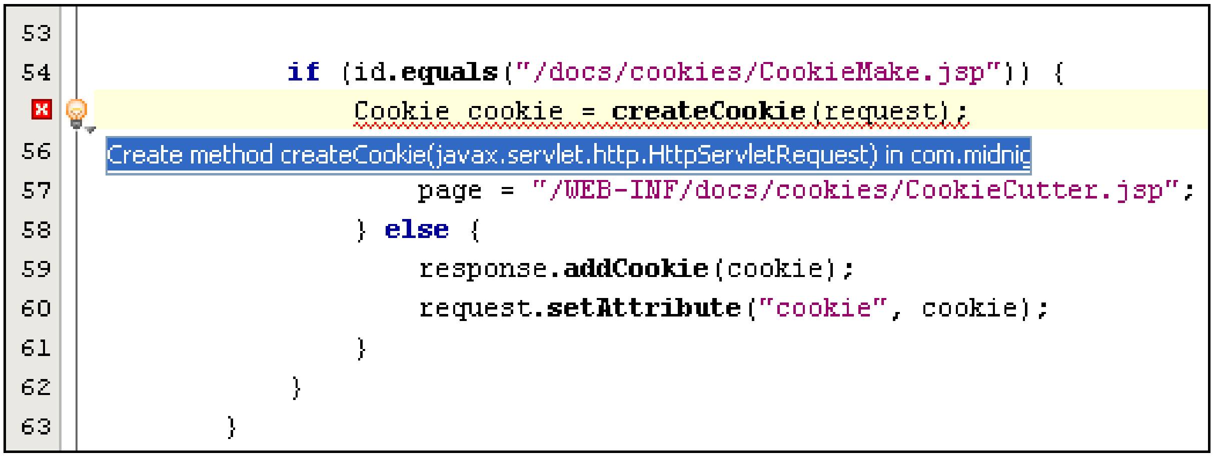
```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {
    String id = request.getRequestURI();
    id = id.substring(request.getContextPath().length());
    getServletContext().log
```

```
          ("Midnight Cookies FrontController received a request for " + id);

        // The page variable holds the path to the page that will be included
        // in the content area of the template page /WEB-INF/docs/main.jsp.
        // It is passed to the template in a request attribute.
        String page;

        // If no request is received or if the request received is
        // for the root, serve /WEB-INF/docs/index.jsp
        if (id == null || id.trim().equals("") || id.equals("/")){
            page = "/WEB-INF/docs/index.jsp";
        }

        // If a request received is for a file that does not end in
        // .jsp or .html, there was an error. Include the
        // error page (nonesuch.jsp) and set the nonesuch attribute with
        // the path requested.
        else if (!id.endsWith(".jsp") && !id.endsWith(".html")) {
            page = "/WEB-INF/docs/nonesuch.jsp";
            request.setAttribute("nonesuch", id);
        }
        else {
            page = "/WEB-INF".concat(id);

            if (id.equals("/docs/cookies/CookieMake.jsp")) {
                Cookie cookie = createCookie(request);
                if (cookie == null) {
                    page = "/WEB-INF/docs/cookies/CookieCutter.jsp";
                }
                else {
                    response.addCookie(cookie);
                    request.setAttribute("cookie", cookie);
                }
            }
        }

        request.setAttribute("page", page);
        try {
            request.getRequestDispatcher("/WEB-INF/docs/main.jsp").forward(request, response);
        }
        catch(Throwable t) {
            getServletContext().log(t.getMessage());
        }
    }
```

Note: After you enter code (either by typing or copying and pasting) into the Source Editor, place the cursor in the Source Editor and press Ctrl-Shift-F to automatically reformat the code. To show line numbers, right-click the left margin and choose Show Line Numbers from the contextual menu.

The line with the call to the `createCookie` method is underlined and marked as an error, similar to the following illustration. This is because the method does not exist yet.

Click Alt-Enter in the line with the error and notice that the IDE presents a suggestion of how to solve the error:

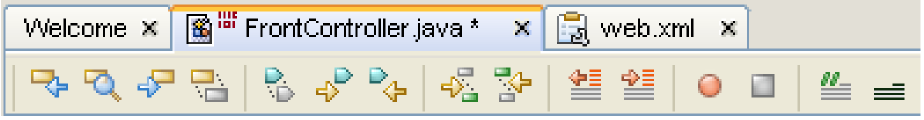


```
            if (id.equals("/docs/cookies/CookieMake.jsp")) {
                Cookie cookie = createCookie(request);
Create method createCookie(javax.servlet.http.HttpServletRequest) in com.midnic
                    page = "/WEB-INF/docs/cookies/CookieCutter.jsp";
                } else {
                    response.addCookie(cookie);
                    request.setAttribute("cookie", cookie);
                }
            }
        }
```

4. Select the suggestion. The IDE creates the skeleton `createCookie` method at the end of the file.
5. Fill out the new `createCookie` method by copying the following code and pasting it in the Source Editor:

```
    private Cookie createCookie(HttpServletRequest request) {

        String name = (String)request.getParameter("name");
        if (name == null || name.trim().equals("")) {
            request.setAttribute("noname", "noname");
            request.setAttribute("error", "true");
            return null;
        }

        String value = (String)request.getParameter("value");
        if (value == null || value.trim().equals("")) {
            request.setAttribute("novalue", "novalue");
            request.setAttribute("error", "true");
            return null;
        }

        System.out.println(name);
        System.out.println(value);

        Cookie cookie = null;
        try {
            cookie = new Cookie(name, value);
        }
        catch(IllegalArgumentException ex) {
            // Probably an illegal name
            ex.printStackTrace();
            request.setAttribute("error", "true");
            request.setAttribute("noname", "badname");
            return null;
        }

        String maxage = request.getParameter("maxage");
        if (maxage != null && !maxage.trim().equals("")) {
            try {
                cookie.setMaxAge(Integer.parseInt(maxage));
            }
            catch(NumberFormatException nfe) {
                nfe.printStackTrace();
                request.setAttribute("badnumber", "badnumber");
                request.setAttribute("error", "true");
                return null;
            }
        }

        String domain = request.getParameter("domain");
        if (domain != null && !domain.trim().equals("")) {
            cookie.setDomain(domain);
        }

        String path = request.getParameter("path");
        if (path != null && !path.trim().equals("")) {
            cookie.setPath(path);
        }

        String secure = request.getParameter("secure");
        if (secure != null && secure.equals("on")) {
            cookie.setSecure(true);
        } else {
            cookie.setSecure(false);
        }
        return cookie;
    }
```

6. The asterisk in the file's tab at the top of the Source Editor appears whenever there are changes that have not been saved to disk:



   Press Ctrl-S to save the document. The asterisk in the file's tab at the top of the Source Editor disappears.

7. In the Projects window, right-click the FrontController.java node and select Compile File (F9) to check that there were no errors introduced in the steps you have completed. The Output window displays messages from the compiler, including any errors. You can double-click the links in the error messages to locate the source of the error. Resolve all errors listed in the Output window before continuing with the tutorial.

Creating the JSP page that receives all the requests

The `main.jsp` page receives all the requests that the FrontController servlet processes. A dynamic include statement and a LinksHandler are used to create the page.

1. Expand the Midnight project node, the Web Pages node, and the WEB-INF node. Right-click the docs node, choose New > JSP, and name the JavaServer Pages file `main`. Click Finish.
   `main.jsp` opens in the Source Editor.

2. In the Source Editor, replace the contents of the `main.jsp` file with the following code:

```
<%@page contentType="text/html;charset=UTF-8"%>
<%@page buffer="none"%>
<%-- The <midnight:content/> tag which handles the contents section
is simplistically implemented to use an include. Hence this page
cannot be buffered --%>

<%@taglib prefix="midnight" uri="http://www.midnightcookies.com/midnight"%>

<html>
  <head><title>The Midnight Cookie Company</title></head>
  <body text="#996633" link="#cc6600" vlink="#993300" alink="#000000">
      <span font-style="sans-serif">
          <TABLE border="0">
              <tr>
                  <table border="0">
                      <tr><td width="80" height="100">&nbsp;</td>
                          <td width="500" height="100"

                              text="#996633" bgcolor="#ffff99"
                              valign="center"halign="center">
                              <jsp:include page="/WEB-INF/docs/header.jsp"
                                           flush="true"/>
                          </td>
                      </tr>
                      <tr><td width="90" height="300"
                              text="#996633" bgcolor="#ffff99"
                              valign="top">
                              <midnight:links separator="<br>"/>
                          </td>
                          <td width="500" height="300"
                              valign="top"cellpadding="15"
                              cellspacing="15">
                              <midnight:content/>
                          </td>
                      </tr>
                  </table>
              </tr>
              <tr><td width="580" height="50" text="#996633"
                      bgcolor="#ffff99" valign="top">
                      <midnight:links separator="|"/>
                  </td>
              </tr>
          </span>
      </TABLE>
  </body>
</html>
```

Note: After you enter code (either by typing or copying and pasting) into the Source Editor, place the cursor in the Source Editor and press Ctrl-Shift-F to automatically reformat the code.

3. Press Ctrl-S to save your changes.

Creating the parameter files used to handle user input

The parameter files that you create in this section illustrate the use of tags from tag libraries to handle internationalization. They also show how input is passed through the Request parameters, specifically through the JSP expression language (EL) expression `${param.input}`, which is set in the input field of the `Input.jsp` page.

1. Right-click the docs node and select New > File/Folder from the contextual menu. Under Categories, select Other and under File Types, select Folder. Click Next.
2. In the Name and Location panel, type `parameters` in the Folder Name text field and click Finish. The new parameters folder appears in the Projects window under the docs node.
3. Right-click the parameters node and choose New > JSP. Type `Input` in the JSP File Name text field and make sure that `WEB-INF/docs/parameters` is selected in the Folder text box. Click Finish.
   `Input.jsp` opens in the Source Editor.
4. In the Source Editor, replace the contents of the `Input.jsp` file with the following code:

```
<%@page contentType="text/html;charset=UTF-8"%>
<%@taglib prefix="fmt" uri="http://java.sun.com/jstl/fmt" %>
<%@taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%@taglib prefix="midnight" uri="/WEB-INF/tlds/midnight.tld"%>

<fmt:setBundle basename="com.midnightcookies.Midnight" var="bundle"
             scope="page"/>

<h3><fmt:message key="provide_input" bundle="${bundle}"/></h3>
<form method="POST" action="Output.jsp">
      <table>
          <tr>
              <td><fmt:message key="type_input" bundle="${bundle}"/>:<td>
              <%-- The value of the Input field will be placed in a
                    request parameter named "input" and it will be
                    passed to Output.jsp --%>
              <td><input type="text" size="20" name="input" value=""></td>
          </tr>
          <tr>
              <td><fmt:message key="submit" bundle="${bundle}"
                               var="buttonLabel" scope="page"/>
                   <midnight:button/>
              </td>
              <td></td>
          </tr>
      </table>
</form>
```

5. Press Ctrl-S to save your changes.
6. Create an `Output.jsp` file in the Parameters folder using the same method that you used to create the `Input.jsp` file.
7. In the Source Editor, replace the contents of the `Output.jsp` file with the following code:

```
<%@page contentType="text/html;charset=UTF-8"%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jstl/fmt" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>

<fmt:setBundle basename="com.midnightcookies.Midnight" var="bundle" scope="page"/>
<h3><fmt:message key="display_input" bundle="${bundle}"/></h3>
<fmt:message key="datareceived" bundle="${bundle}"/>:

<!--The following line gets the value of the request parameter named
"input". This is a demo application. For security reasons, one should
never echo user input without parsing first. -->
<c:out value="${param.input}"/>
```

8. Press Ctrl-S to save your changes.

Using JSTL to handle internationalization

The `Tray.jsp` page that you create in this section illustrates the use of the JSTL, instead of scripting, to internationalize the page. The `CookieCutter.jsp` file, `CookieMaker.jsp` file, and resource bundle property files have been provided with the `midnight.zip` file. You can take a closer look at those files and the `Tray.jsp` file for examples of how internationalization is handled in this application.

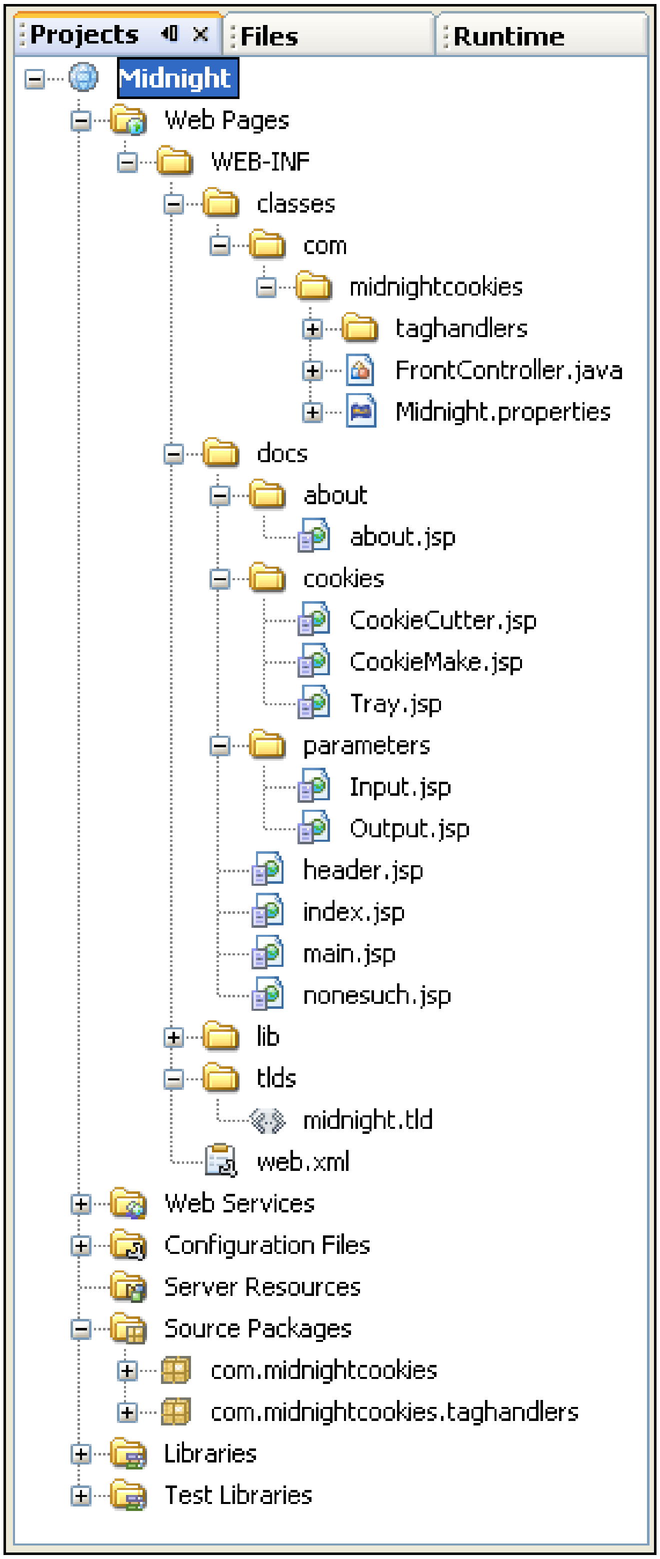
1. Expand the docs node and expand the cookies node.
2. Create a `Tray.jsp` file in the `cookies` folder using the same method that you used to create the other JSP files. Make sure that you put it in the `cookies` folder.
3. In the Source Editor, replace the contents of the `Tray.jsp` file with the following code:

```
<%@page contentType="text/html;charset=UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jstl/fmt" %>

<fmt:setBundle basename="com.midnightcookies.Midnight" var="bundle" scope="page"/>
<h3><fmt:message key="cookies" bundle="${bundle}"/></h3>

<table border="1">
   <tr>
      <th halign="center">#</th>
      <th align="left">
          <fmt:message key="name" bundle="${bundle}"/>
      </th>
      <th align="left">
          <fmt:message key="value" bundle="${bundle}"/>
      </th>
   <tr>
   <%-- We have to use expression to get at an array.
        If we use ${cookie} then we get a map and
        the entries are not automatically cast --%>
   <% request.setAttribute("cookies", request.getCookies()); %>
   <c:set var="i" value="0"/>

   <c:forEach var="ck" items="${cookies}">
      <c:set var="i" value="${i+1}"/>
      <tr>
          <td><c:out value="${i}"/></td>
          <td><c:out value="${ck.name}"/></td>
          <td><c:out value="${ck.value}"/></td>
      </tr>
  </c:forEach>
</table>
```

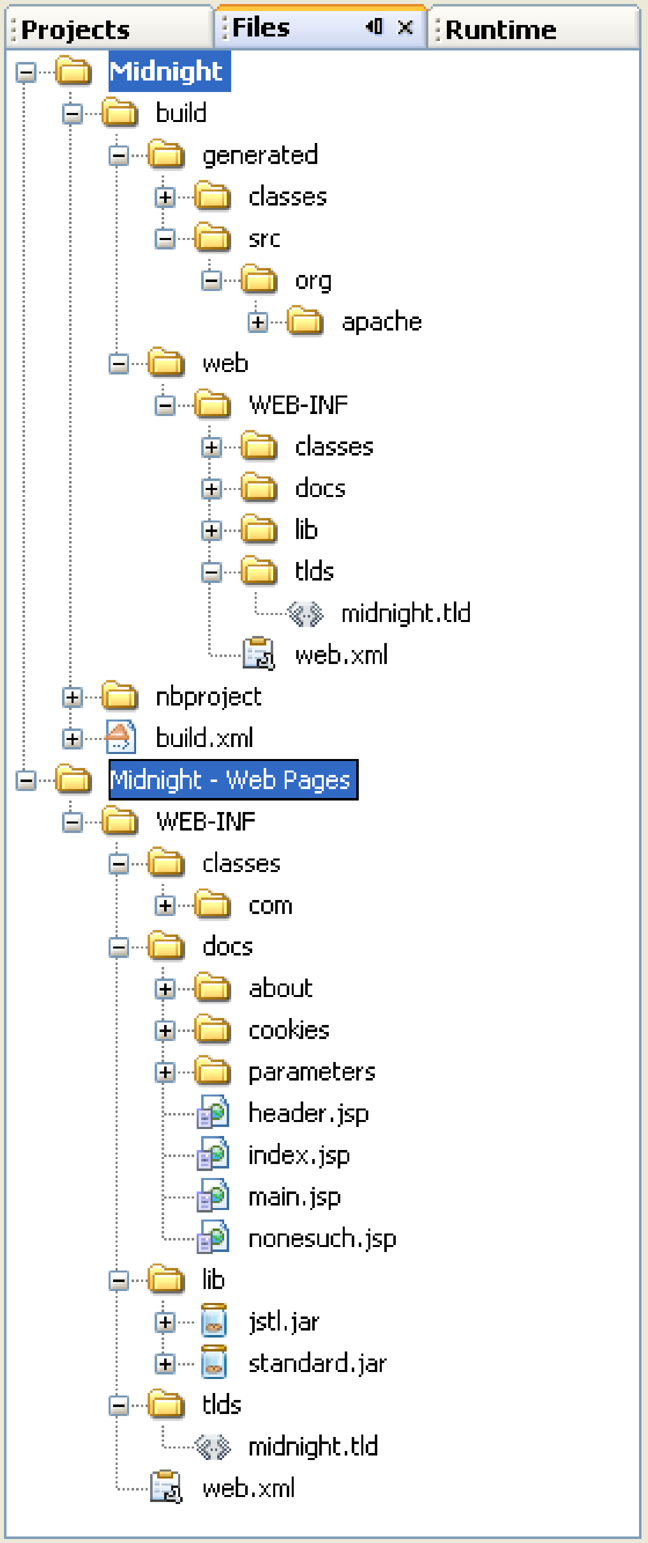
4. Press Ctrl-S to save your changes.

Wrapping up

Once you have completed the development stage as described above, the views in the Projects window and Files window should look as follows:



```
Midnight
  Web Pages
    WEB-INF
      classes
        com
          midnightcookies
            taghandlers
            FrontController.java
            Midnight.properties
      docs
        about
          about.jsp
        cookies
          CookieCutter.jsp
          CookieMake.jsp
          Tray.jsp
        parameters
          Input.jsp
          Output.jsp
        header.jsp
        index.jsp
        main.jsp
        nonesuch.jsp
      lib
      tlds
        midnight.tld
      web.xml
  Web Services
  Configuration Files
  Server Resources
  Source Packages
    com.midnightcookies
    com.midnightcookies.taghandlers
  Libraries
  Test Libraries
```



```
Midnight
  build
    generated
      classes
      src
        org
          apache
    web
      WEB-INF
        classes
        docs
        lib
        tlds
          midnight.tld
        web.xml
  nbproject
  build.xml
Midnight - Web Pages
  WEB-INF
    classes
      com
    docs
      about
      cookies
      parameters
      header.jsp
      index.jsp
      main.jsp
      nonesuch.jsp
    lib
      jstl.jar
      standard.jar
    tlds
      midnight.tld
    web.xml
```

Note: The application illustrated above deploys to the Tomcat Web Server. If you have registered and selected a different target server instead, you will not have a `META-INF/context.xml` file. For example, if your server is the Sun Java System Application Server, you will have a `WEB-INF/sun-web.xml` file as your deployment descriptor.

Running and Monitoring the Application

Enabling the HTTP Monitor

The HTTP Monitor is enabled by default for the Tomcat Web Server. However, for other servers, you must manually enable the HTTP Monitor. In addition, the HTTP Server must be started for the HTTP Monitor to work for other servers.

Do the following to prepare to use the HTTP Monitor:

1. Choose Tools > Server Manager from the main window.
2. Select the server.
3. Click Enable HTTP Monitor and then click Close.
4. In the Runtime window, right-click the HTTP Server node and choose Start HTTP Server.

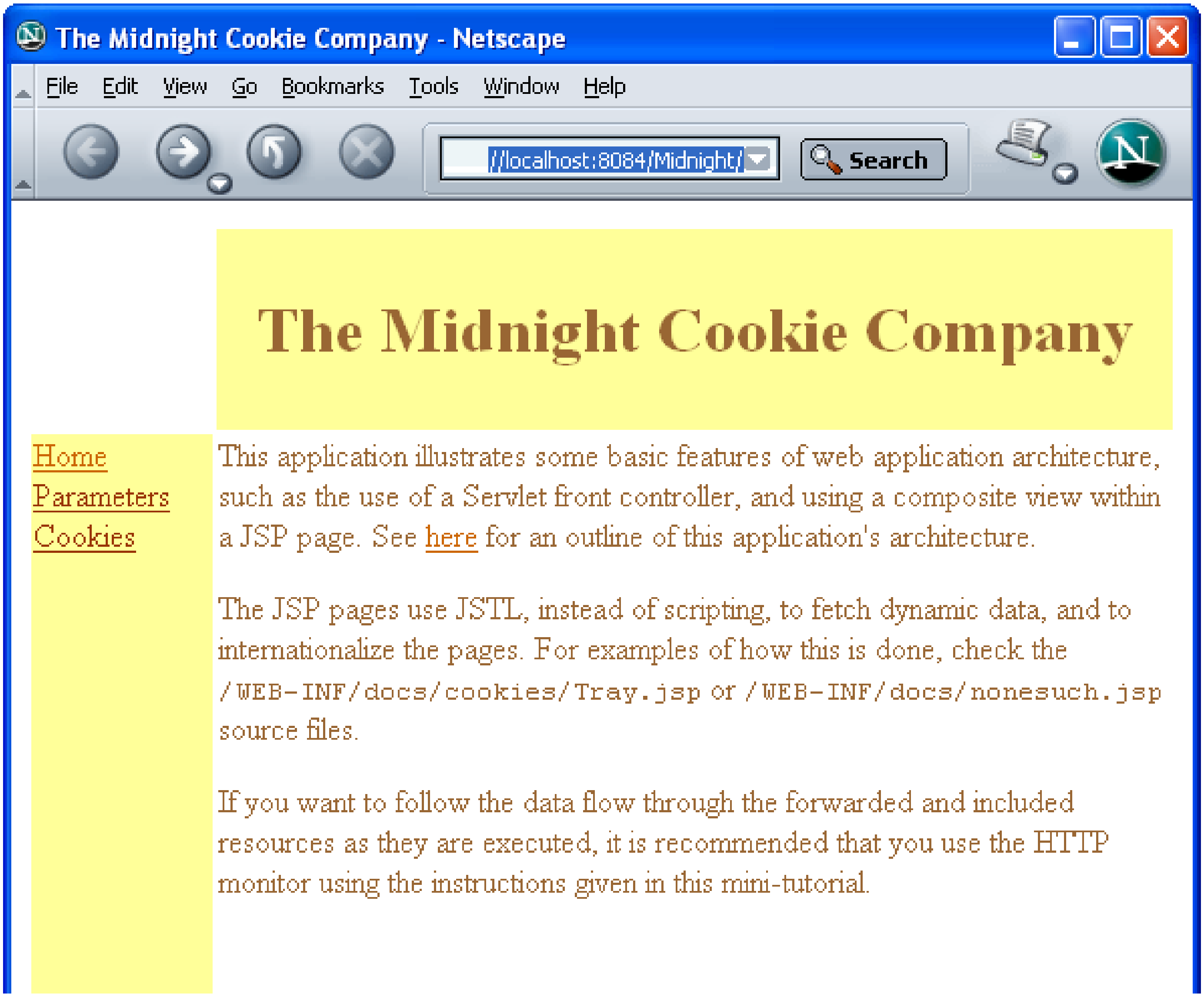
Verifying the Context Path

1. Right-click the Midnight project node and choose Properties.
2. In the Project Properties dialog box, click Run.
3. Make sure that the Context Path is set to `/Midnight`.

Running the application

1. In the Projects window, right-click the Midnight project node and choose Run Project from the contextual menu.
   A progress monitor window appears as the application is prepared and deployed to the web server.

   The web browser launches with the URL set to `http://`*hostname:port*`/Midnight`. The following example shows the web browser running on a Microsoft Windows platform.

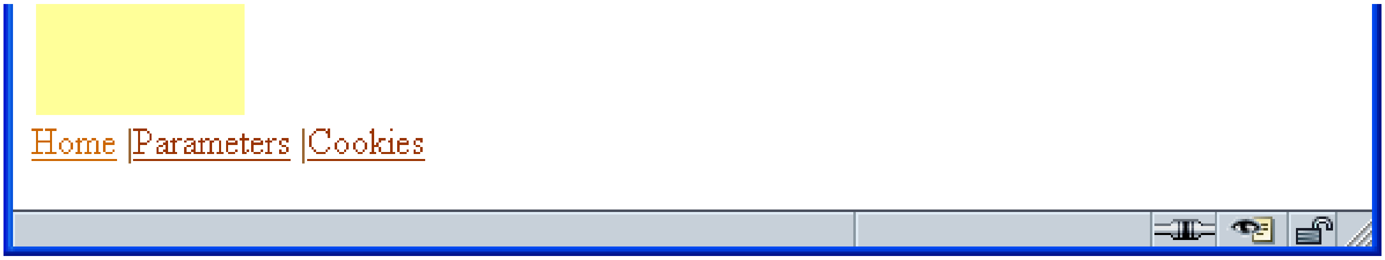


The Midnight Cookie Company

This application illustrates some basic features of web application architecture, such as the use of a Servlet front controller, and using a composite view within a JSP page. See here for an outline of this application's architecture.

The JSP pages use JSTL, instead of scripting, to fetch dynamic data, and to internationalize the pages. For examples of how this is done, check the `/WEB-INF/docs/cookies/Tray.jsp` or `/WEB-INF/docs/nonesuch.jsp` source files.

If you want to follow the data flow through the forwarded and included resources as they are executed, it is recommended that you use the HTTP monitor using the instructions given in this mini-tutorial.



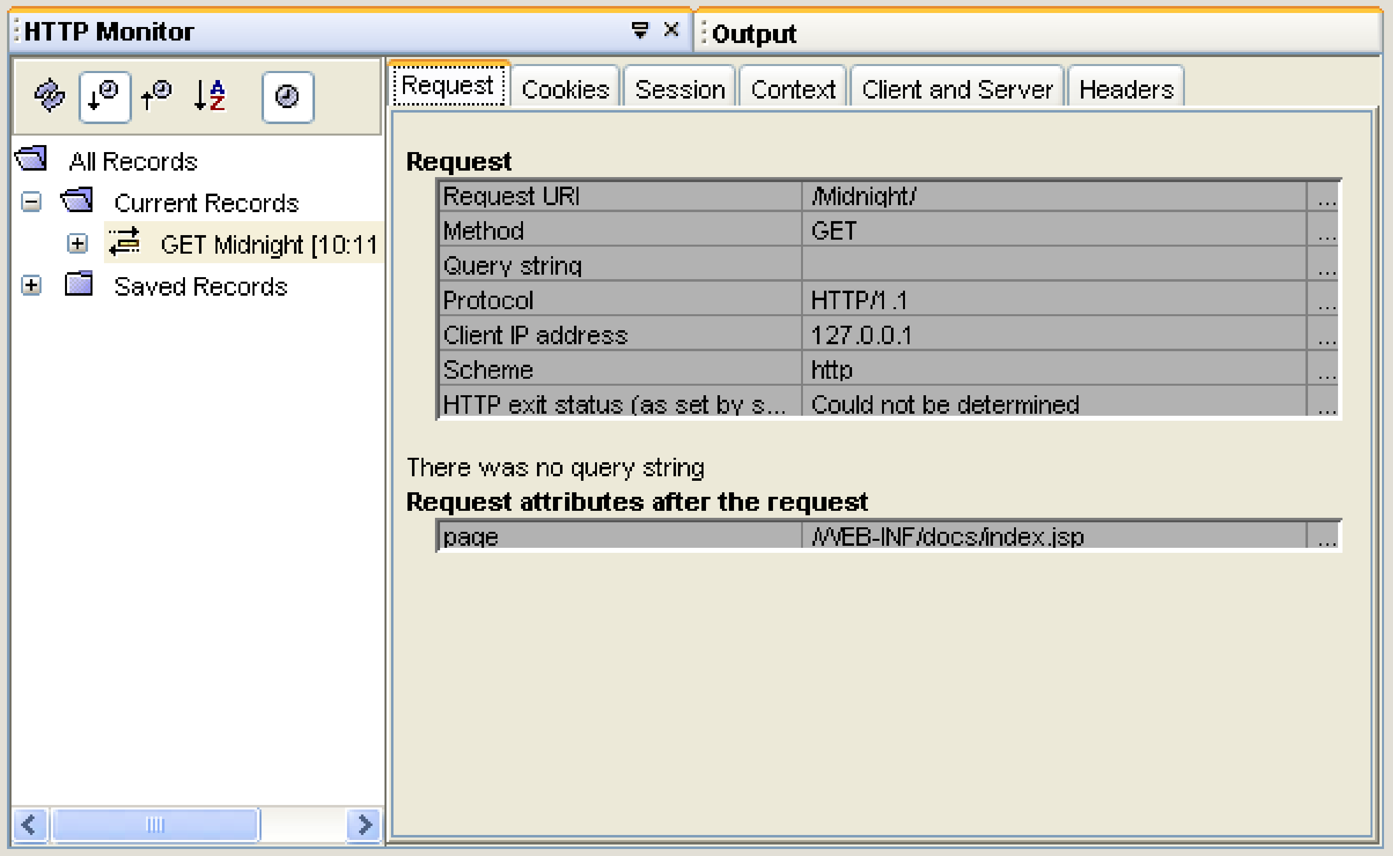
If you receive an error while trying to execute this application, see Troubleshooting Execution Problems for some possible solutions.

2. In the IDE, notice that the HTTP Monitor is displayed.
   The following sections illustrate how you can use the HTTP Monitor to monitor requests, sessions, and cookies.

Monitoring requests and sessions

1. In the HTTP Monitor, select the last entry under the Current Records node.
   The last entry represents the request for the Midnight web application.

2. Select the Request tab and you see information similar to the following.



Request

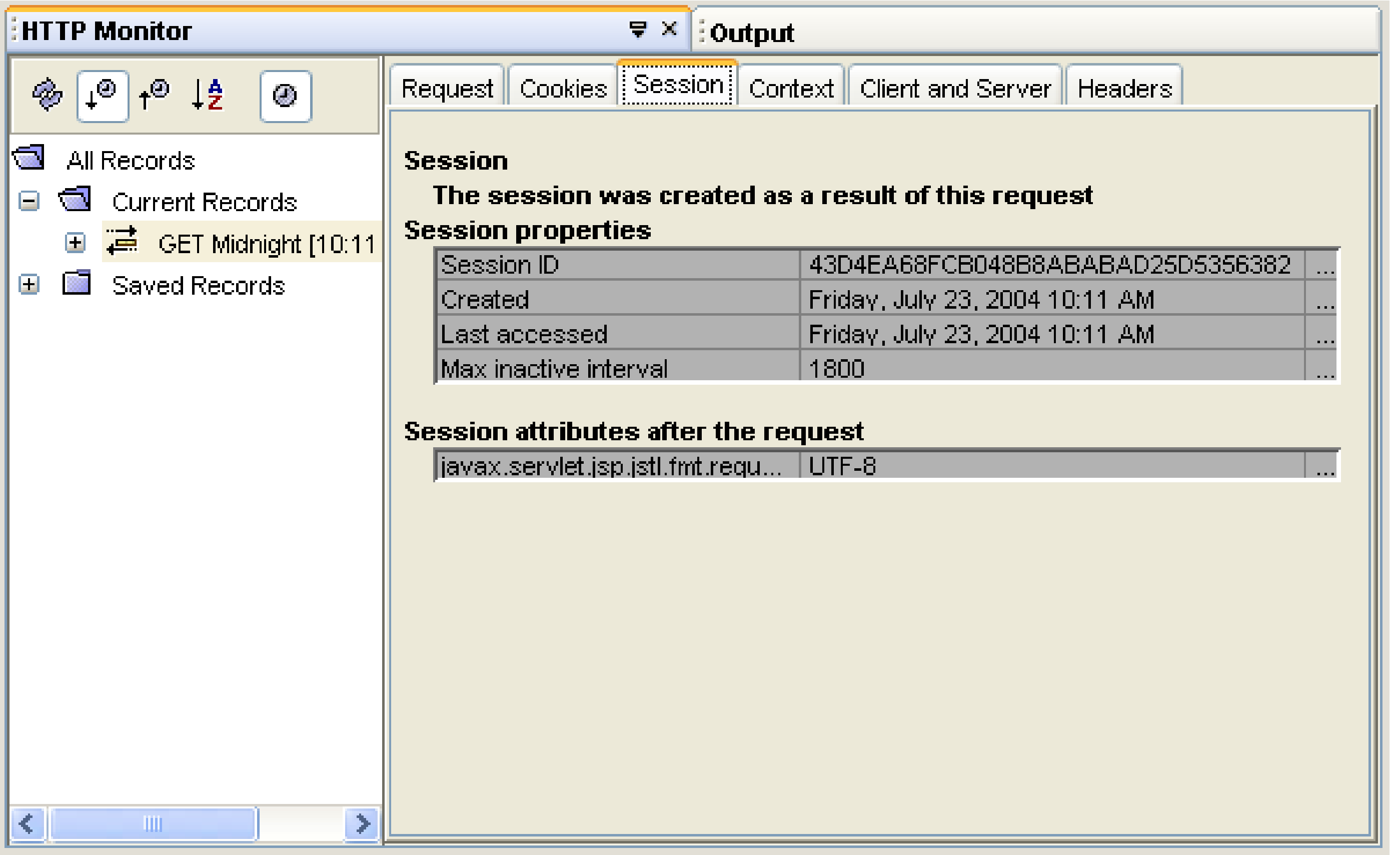
Request URI	/Midnight/	...
Method	GET	...
Query string		...
Protocol	HTTP/1.1	...
Client IP address	127.0.0.1	...
Scheme	http	...
HTTP exit status (as set by s...	Could not be determined	...

There was no query string

Request attributes after the request

page	/WEB-INF/docs/index.jsp	...

3. Select the Session tab and you see information similar to the following.



Session

The session was created as a result of this request

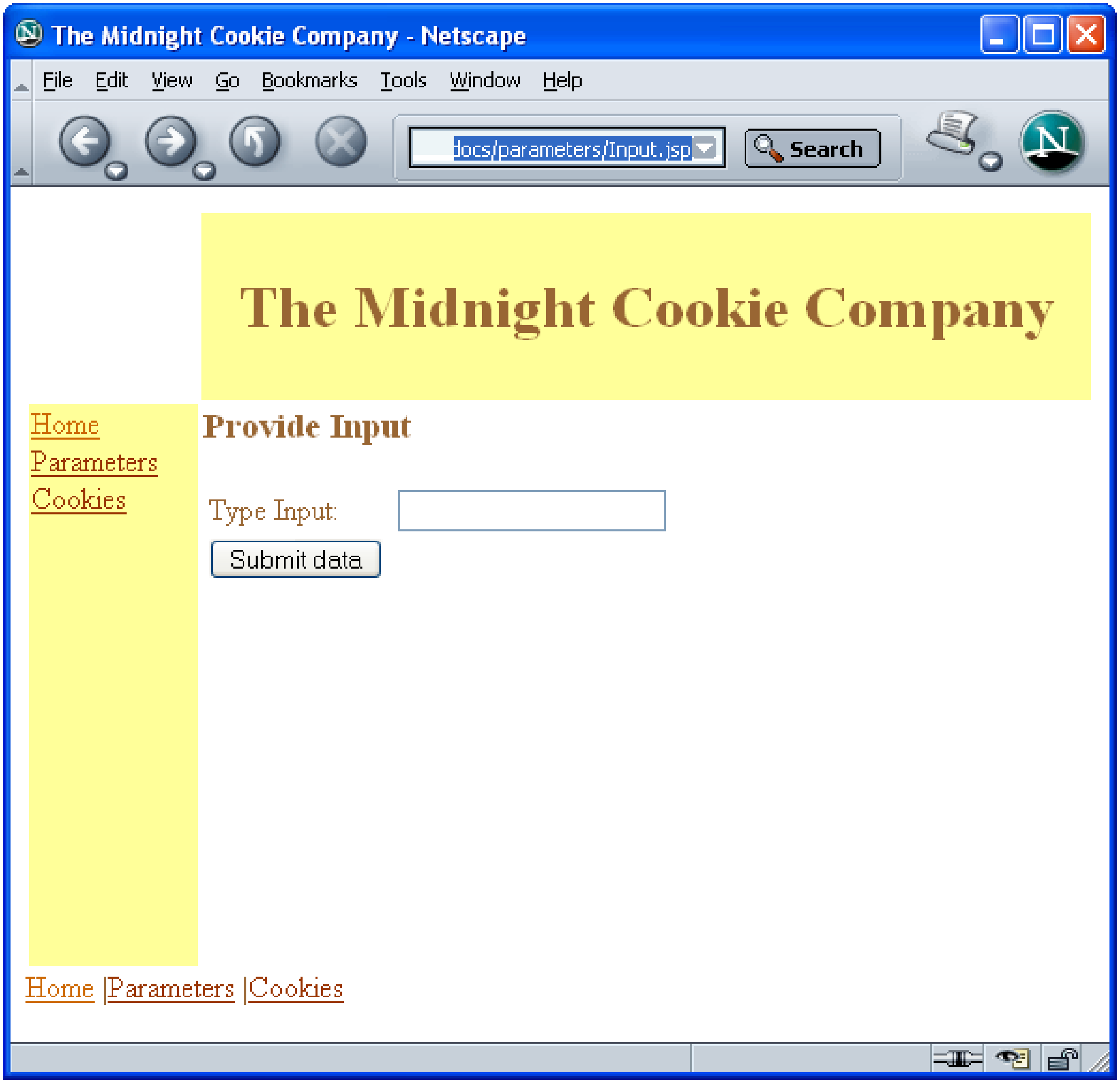
Session properties

Session ID	43D4EA68FCB048B8ABABAD25D5356382	...
Created	Friday, July 23, 2004 10:11 AM	...
Last accessed	Friday, July 23, 2004 10:11 AM	...
Max inactive interval	1800	...

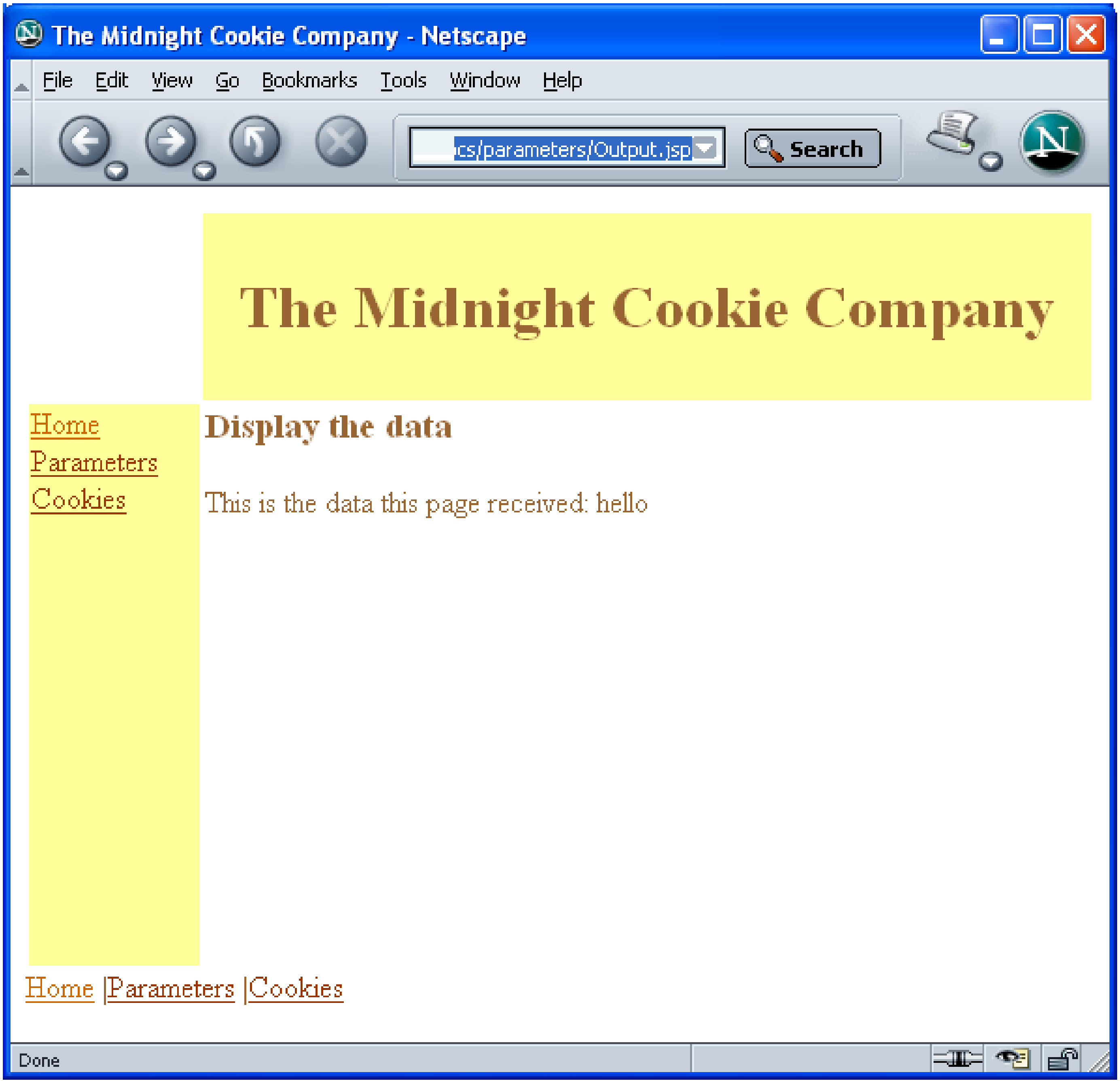
Session attributes after the request

javax.servlet.jsp.jstl.fmt.requ...	UTF-8	...

4. In your web browser, click the Parameters link to display the `Input.jsp` page.
   Your web browser displays information similar to the following.


The Midnight Cookie Company - Netscape
File Edit View Go Bookmarks Tools Window Help
docs/parameters/Input.jsp
Search
The Midnight Cookie Company
Home
Parameters
Cookies
Provide Input
Type Input:
Submit data
Home |Parameters |Cookies

5. In the Type Input text field, type `hello` and click Submit data.
   Your web browser displays the contents of the `Output.jsp` page, similar to the following.



The Midnight Cookie Company - Netscape
File Edit View Go Bookmarks Tools Window Help
cs/parameters/Output.jsp
Search
Done

The Midnight Cookie Company

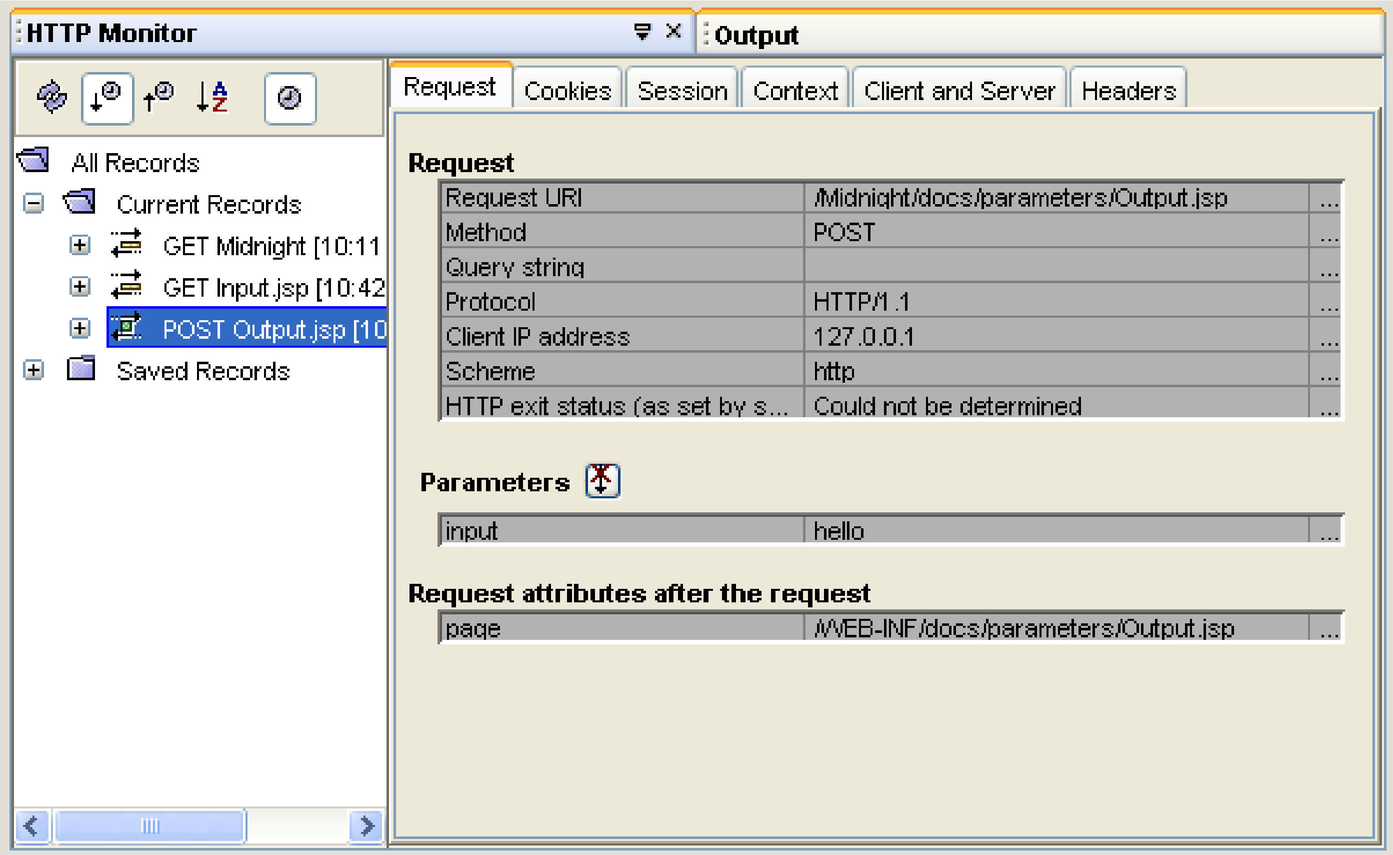
Home
Parameters
Cookies

Display the data

This is the data this page received: hello

Home |Parameters |Cookies

6. In the HTTP Monitor, select the last entry under the Current Records and select the Request tab.
   The HTTP Monitor displays information similar to the following.



Request

Request URI	/Midnight/docs/parameters/Output.jsp	...
Method	POST	...
Query string		...
Protocol	HTTP/1.1	...
Client IP address	127.0.0.1	...
Scheme	http	...
HTTP exit status (as set by s...	Could not be determined	...

Parameters

input	hello	...

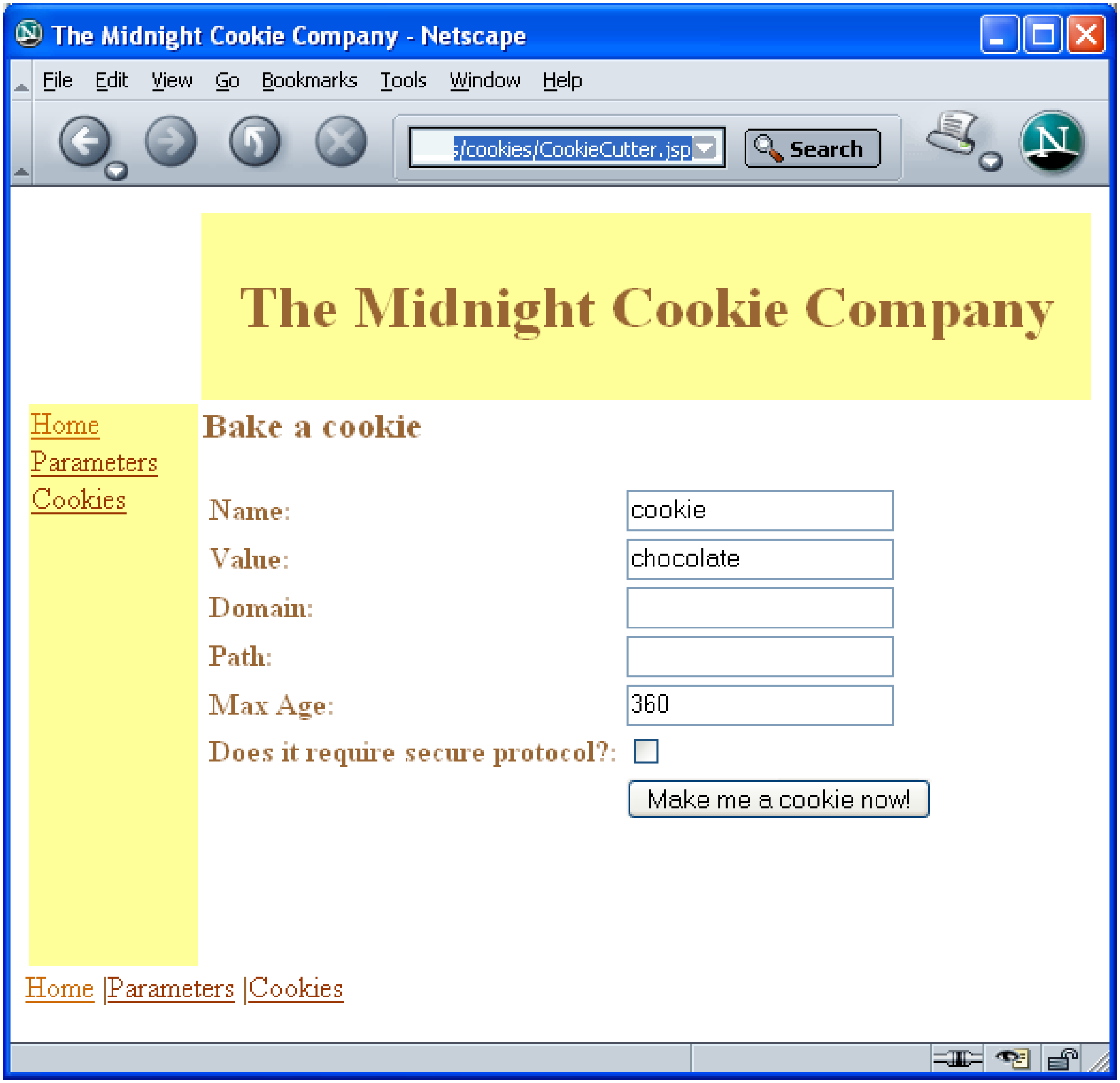
Request attributes after the request

page	/WEB-INF/docs/parameters/Output.jsp	...

Monitoring cookies

1. In your web browser, click the Cookies link.
   Your web browser displays the `CookieCutter.jsp` page.

2. In the Name text field, type `cookie`, and in the Value text field, type `chocolate`.

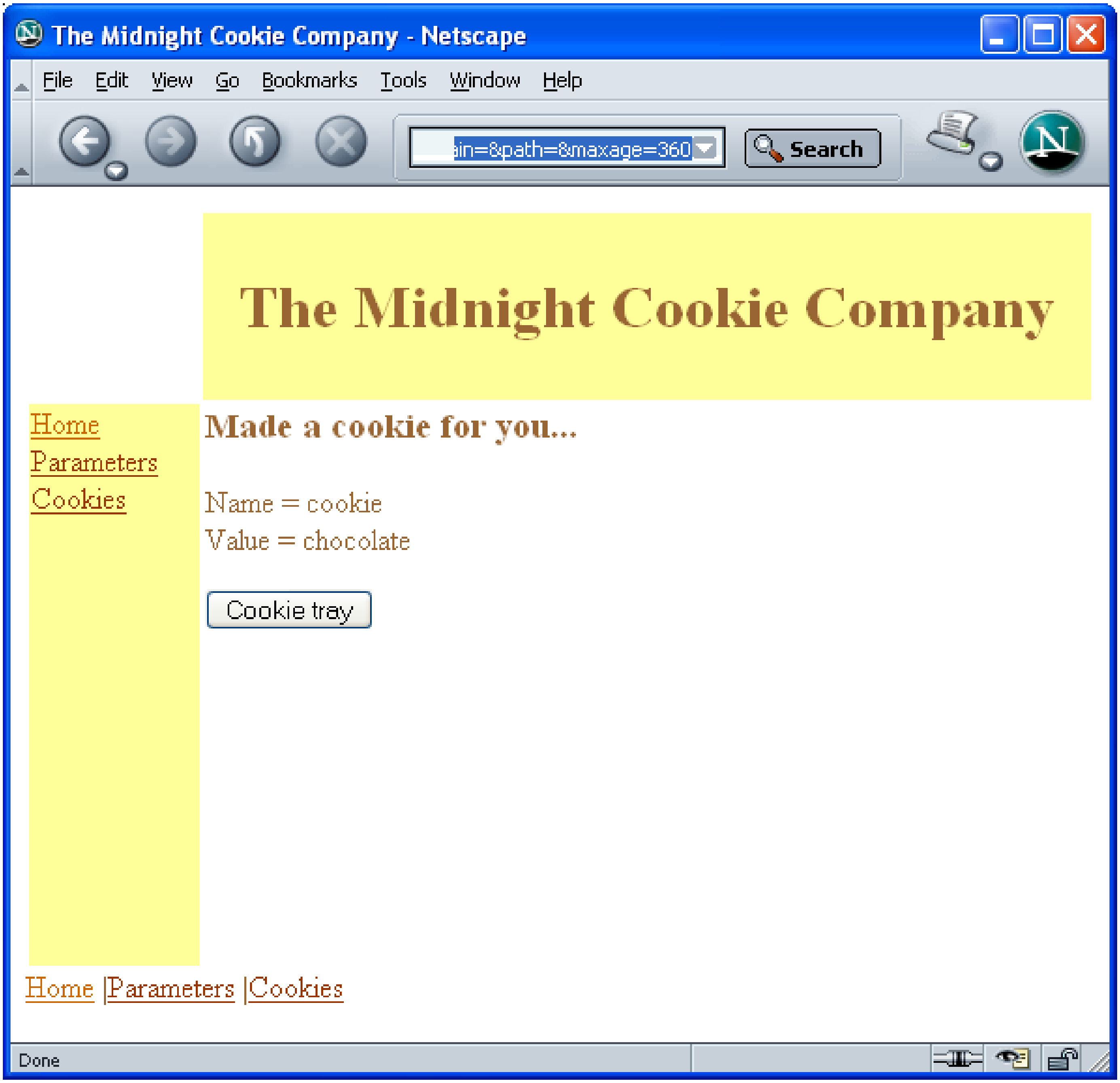

The Midnight Cookie Company - Netscape
File Edit View Go Bookmarks Tools Window Help
/cookies/CookieCutter.jsp
Search
Home
Parameters
Cookies
Make me a cookie now!
Home |Parameters |Cookies

The Midnight Cookie Company

Bake a cookie

Name:	cookie
Value:	chocolate
Domain:	
Path:	
Max Age:	360
Does it require secure protocol?:	□

3. Click "Make me a cookie now!".
   Your web browser displays information similar to the following.

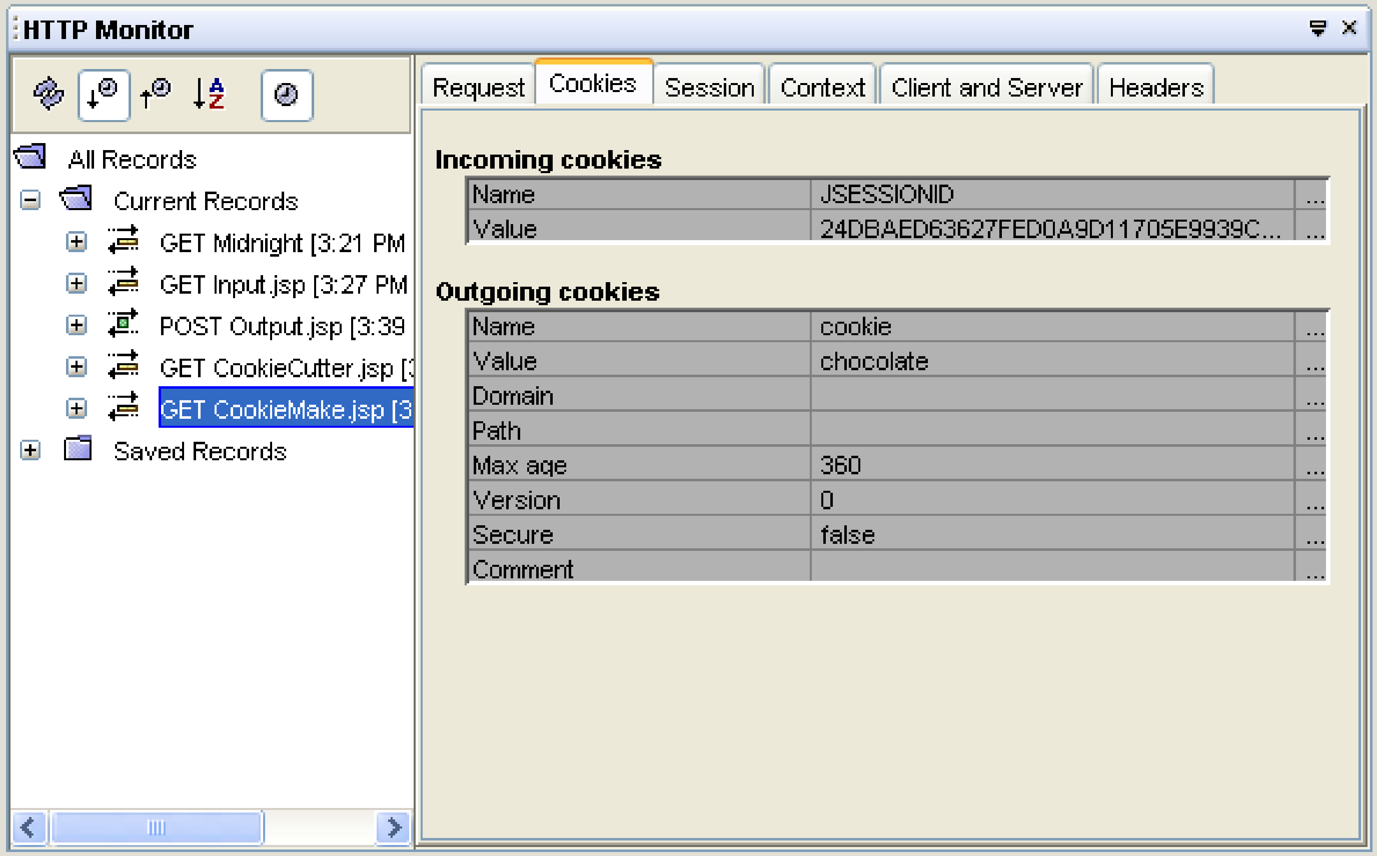

The Midnight Cookie Company - Netscape
File Edit View Go Bookmarks Tools Window Help
ain=&path=&maxage=360
Search
The Midnight Cookie Company
Home
Parameters
Cookies
Cookie tray
Home |Parameters |Cookies
Done

Made a cookie for you...

Name = cookie

Value = chocolate

4. In the HTTP Monitor, select the last entry under the Current Records node.
   The Cookies tab in the HTTP Monitor displays information similar to the following.



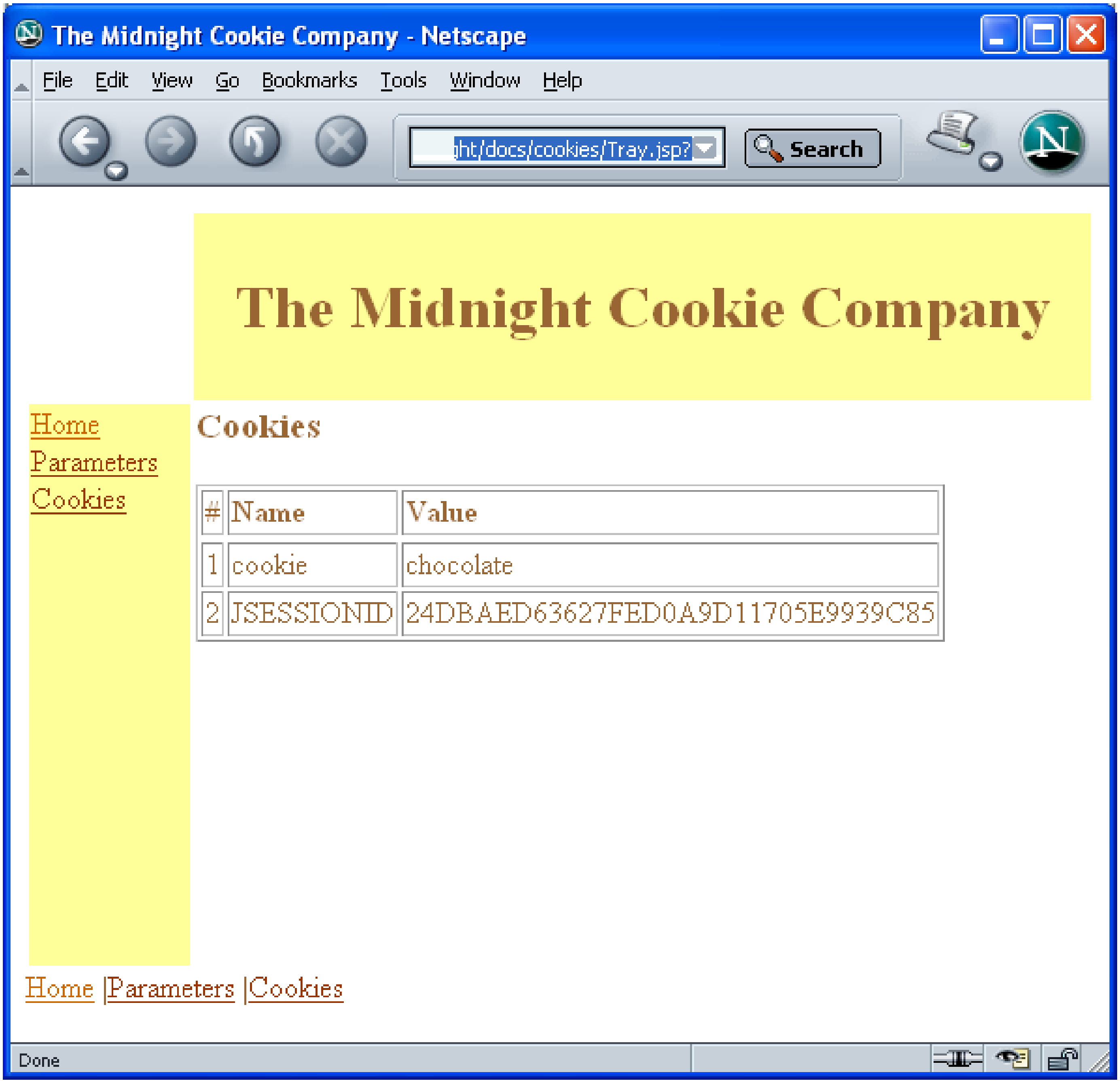
Incoming cookies

Name	JSESSIONID	...
Value	24DBAED63627FED0A9D11705E9939C...	...

Outgoing cookies

Name	cookie	...
Value	chocolate	...
Domain		...
Path		...
Max age	360	...
Version	0	...
Secure	false	...
Comment		...

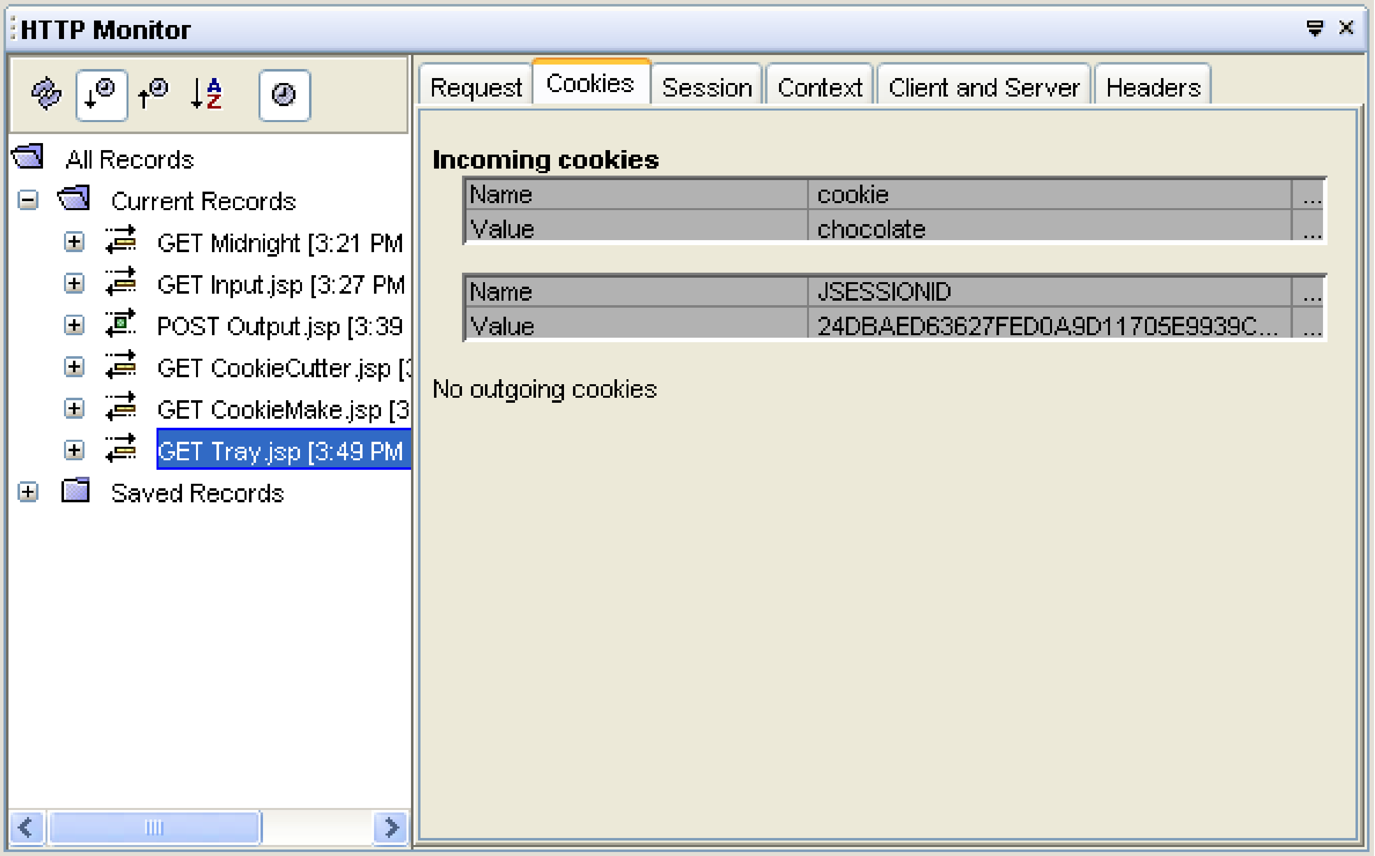
5. In your web browser, click Cookie tray, and your web browser displays information similar to the following.


The Midnight Cookie Company - Netscape
File Edit View Go Bookmarks Tools Window Help
ght/docs/cookies/Tray.jsp?
Search
The Midnight Cookie Company
Home
Parameters
Cookies
Home |Parameters |Cookies
Done

Cookies

#	Name	Value
1	cookie	chocolate
2	JSESSIONID	24DBAED63627FED0A9D11705E9939C85

6. In the HTTP Monitor, select the last entry under the Current Records node and select the Cookies tab. The HTTP Monitor displays information similar to the following.



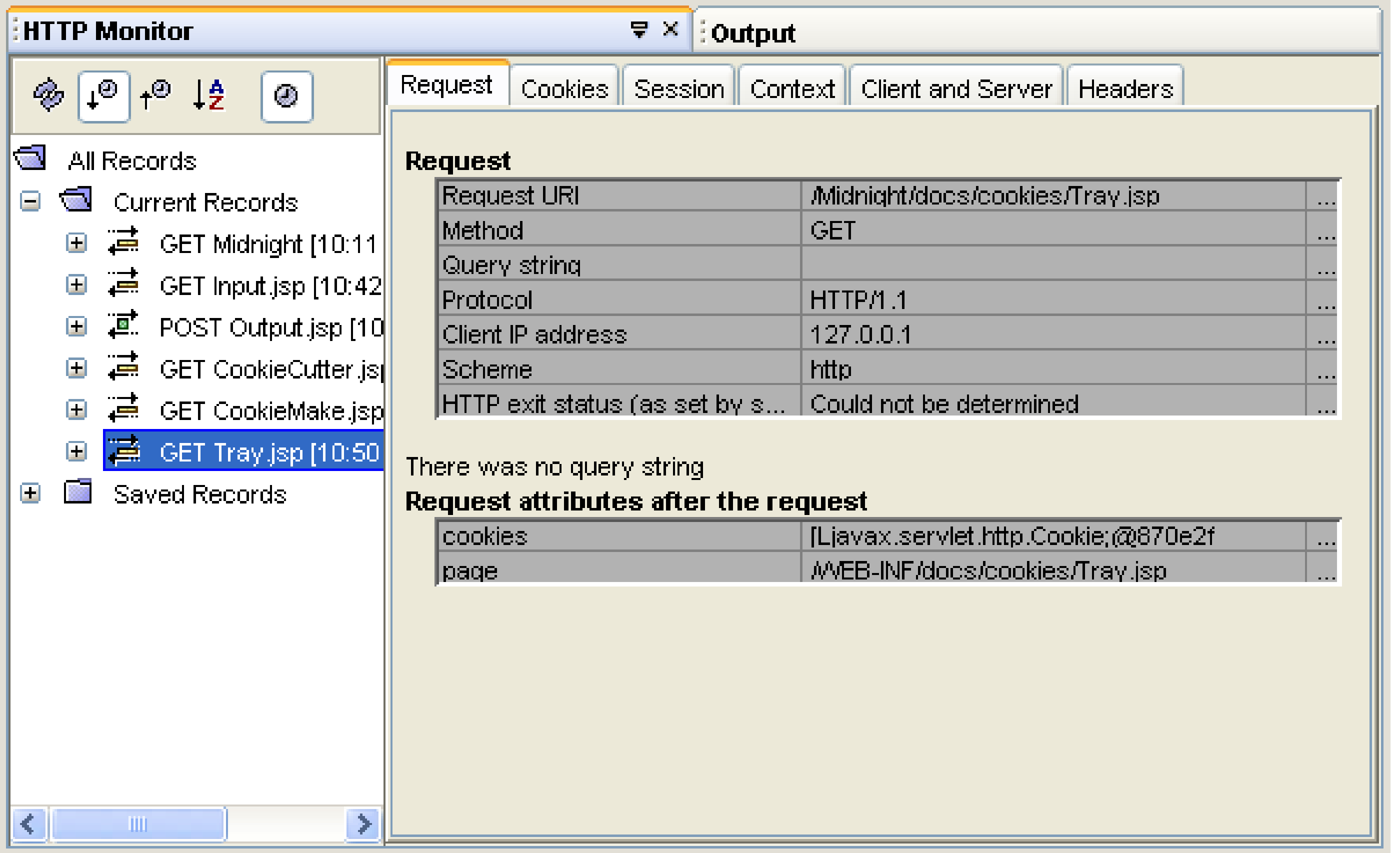
Incoming cookies

Name	cookie	...
Value	chocolate	...

Name	JSESSIONID	...
Value	24DBAED63627FED0A9D11705E9939C...	...

No outgoing cookies

7. Select the Request tab and the HTTP Monitor displays information similar to the following.



Request

Request URI	/Midnight/docs/cookies/Tray.jsp	...
Method	GET	...
Query string		...
Protocol	HTTP/1.1	...
Client IP address	127.0.0.1	...
Scheme	http	...
HTTP exit status (as set by s...	Could not be determined	...

There was no query string

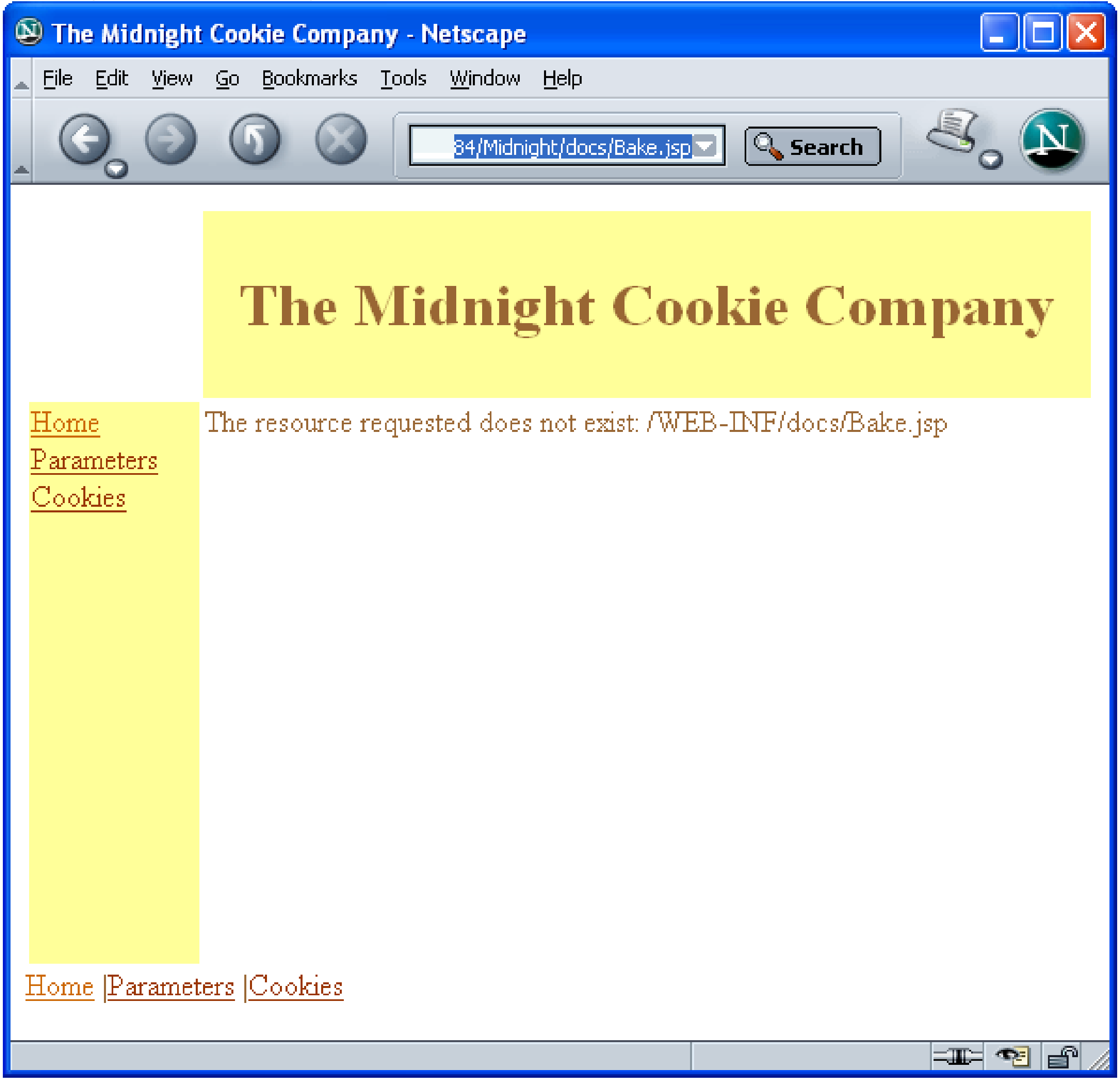
Request attributes after the request

cookies	[Ljavax.servlet.http.Cookie;@870e2f	...
page	/WEB-INF/docs/cookies/Tray.jsp	...

8. In the HTTP Monitor, right-click the same entry (GET Tray.jsp) under the Current Records node and choose Edit and Replay from the contextual menu. In the Edit and Replay dialog box, click the Cookies tab and click Add Cookie. Set the name to Ingredient and the value to Sugar. Click Send HTTP Request. The browser redisplays the page with the new cookie.

Monitoring requests for non-existent pages

1. In your web browser, type a URL for a page that does not exist to see how the `nonesuch.jsp` page is used. For example, you might type `http://localhost:8084/Midnight/docs/Bake.jsp` for the URL and your web browser displays something similar to the following.


The Midnight Cookie Company - Netscape
File Edit View Go Bookmarks Tools Window Help
84/Midnight/docs/Bake.jsp
Search

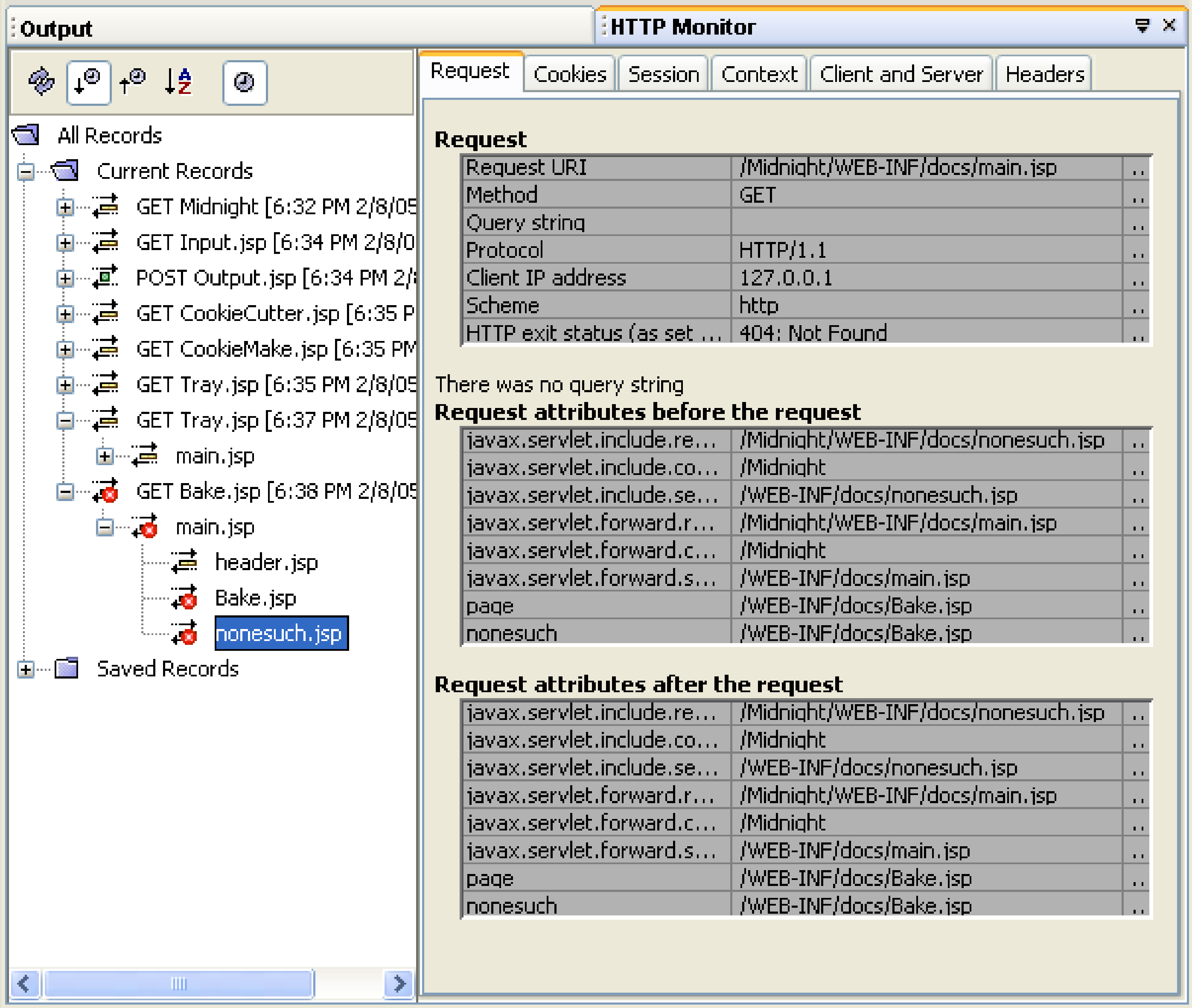
The Midnight Cookie Company

Home
Parameters
Cookies

The resource requested does not exist: /WEB-INF/docs/Bake.jsp

Home |Parameters |Cookies

2. In the HTTP Monitor, select the Request tab and you will see information similar to the following.


Output
HTTP Monitor
Request
Cookies
Session
Context
Client and Server
Headers
All Records
Current Records
GET Midnight [6:32 PM 2/8/05
GET Input.jsp [6:34 PM 2/8/0
POST Output.jsp [6:34 PM 2/
GET CookieCutter.jsp [6:35 P
GET CookieMake.jsp [6:35 PM
GET Tray.jsp [6:35 PM 2/8/05
GET Tray.jsp [6:37 PM 2/8/05
main.jsp
GET Bake.jsp [6:38 PM 2/8/05
main.jsp
header.jsp
Bake.jsp
nonesuch.jsp
Saved Records

Request

Request URI	/Midnight/WEB-INF/docs/main.jsp	..
Method	GET	..
Query string		..
Protocol	HTTP/1.1	..
Client IP address	127.0.0.1	..
Scheme	http	..
HTTP exit status (as set ...	404: Not Found	..

There was no query string

Request attributes before the request

javax.servlet.include.re...	/Midnight/WEB-INF/docs/nonesuch.jsp	..
javax.servlet.include.co...	/Midnight	..
javax.servlet.include.se...	/WEB-INF/docs/nonesuch.jsp	..
javax.servlet.forward.r...	/Midnight/WEB-INF/docs/main.jsp	..
javax.servlet.forward.c...	/Midnight	..
javax.servlet.forward.s...	/WEB-INF/docs/main.jsp	..
page	/WEB-INF/docs/Bake.jsp	..
nonesuch	/WEB-INF/docs/Bake.jsp	..

Request attributes after the request

javax.servlet.include.re...	/Midnight/WEB-INF/docs/nonesuch.jsp	..
javax.servlet.include.co...	/Midnight	..
javax.servlet.include.se...	/WEB-INF/docs/nonesuch.jsp	..
javax.servlet.forward.r...	/Midnight/WEB-INF/docs/main.jsp	..
javax.servlet.forward.c...	/Midnight	..
javax.servlet.forward.s...	/WEB-INF/docs/main.jsp	..
page	/WEB-INF/docs/Bake.jsp	..
nonesuch	/WEB-INF/docs/Bake.jsp	..

Notice that the content of `nonesuch.jsp` page is displayed in place of the non-existent `Bake.jsp` page.

Enabling internationalization

1. Modify the language preference used by your web browser to either French or Swedish in order to see the internationalization feature of this Midnight Cookie Company web application.
   Note that you might have to add these two languages to the list of languages currently available to your web browser.

2. Reload or refresh the pages you have viewed in the previous steps.
   Notice how the links in the left navigation bar, the links on the bottom of the pages, and the contents of some of the application's JSP pages have been translated to the language you have chosen.

   The web application uses the resource bundle property files to help translate some of the pages used in this web application. So, the translation is only as extensive as the contents of the resource bundle property files.

Troubleshooting Execution Problems

If you received an error while executing the Midnight Cookie Company application, following are some hints on some possible solutions to the problems:

You received an exception when the IDE tried to launch the browser you specified.

Check that the correct path is configured for the browser that you have specified to use with the IDE:

1. Select Tools > Options from the main window of the IDE.
2. Click Advanced Options.
3. Expand the IDE Configuration node and then expand Server and External Tool Settings.
4. Expand Web Browsers.
5. Check that the correct path to your browser executable is set. If not, click on the browse button or type the correct path.

You received a `File Not Found` error when attempting to display the JSP page.

If you set your web browser to use a proxy server, your web browser will attempt to route all files you want to view through the proxy server, including local files. You need to add `localhost` and your machine name to the list of bypassed hosts in your web browser's settings. For additional information, see the online help topic entitled "Accessing Local Files Through a Proxy" in the IDE's helpsystem. (Choose Help > Help Contents, expand the "Servers and Databases node" and then the "Web Browsers" node.)

You received a `File not found` error when you executed the web application.

Check the names of the JSP files you created in this tutorial. Make sure that you did not enter the `.jsp` file extension when you typed the name of the object in the New File wizard. Rename the files, if necessary.

Applying What You Have Learned

Add an Exit link that takes the user to a page that says "Goodbye" in the appropriate language

1. Right-click the `com/midnightcookies/Midnight.properties` file and choose Open from the contextual menu. Add the following keywords and values.

Keyword	Default	French	Swedish
exit	Exit	Sortie	Utgång
closingRemark	Goodbye	Au revoir	Hej då

   Use &aring; to display the å character.

2. Add an Exit.jsp content file. Use the message tag from the fmt tag library to display the message for the closingRemark keyword (Hint: look at Output.jsp). Don't forget to add a taglib directive for the fmt tag library.
3. Open the com/midnightcookies/taghandlers/LinksHandler.java file in the Source Editor and add a link to the Exit.jsp file. Use the exit keyword to get the link text from the properties bundle. Compile the class.
4. Execute the web application.

Exploring Further

Below are some other web application features that you might want to explore.

Examining the Midnight tag library

Expand the tlds/Midnight node to see the list of tags in the library. Look at the the Links tag in the Source Editor and at its Attributes to see that it takes a separator attribute.

Debugging web applications

Open the docs/main.jsp file in the Source Editor and set a breakpoint by clicking in the margin at approximately line 32 (the line that invokes midnight:content). Right-click the project node in the Projects or Files window and choose Debug Project from the contextual menu. The web application will stop at the breakpoint that you set. Press F7 to step into the code. Whenever a dialog appears because of missing source, press OK to select the default of stopping at the first line with source. Continue pressing F7 until you enter the otherDoStartTagOperations() function in the ContentHandler class. Set a breakpoint at line 50 (request. getRequestDispatcher(page).include(request, response)) and press Ctrl-F5 (continue). Place the cursor over "page", "request", and "response" to see the values. You can also see the values in the Local Variables tab. Note that the Call Stack tab shows the relationship between the JSP code and the generated servlet methods. Try setting other breakpoints. Right-click a variable and choose New Watch from the contextual menu. Click the Watches tab to see the values change as you execute the various pages in the web application. Choose Debug > Finish Session to quit the debugging session.

Testing JSP files

To run a public JSP file (one that is not under the WEB-INF folder), you simply right-click the file's node and choose Execute from the contextual menu. If a page is loaded, clicking the Browser's reload button will display any modifications that you have made. Private JSP files, like the ones in this tutorial, are a little harder to execute. One way to test a modified private JSP file without having to restart the web application and navigate to that page is to use the HTTP Monitor's replay action. Modify the docs/cookies/Tray.jsp file by setting the table's border to 10 and save your changes. In the HTTP Monitor, right-click a GET Tray.jsp record and choose Replay from the contextual menu. The Tray.jsp page should appear in the browser with a larger table border.

Glossary

Below are explanations of some of the terms used in this tutorial.

Composite View

A design pattern that is used to present information in JSP pages. This design pattern creates an aggregate view from component views. Component views might include dynamic and modular portions of the page. The Composite View design pattern pertains to web application design when you are creating a view from numerous subviews. Complex web pages frequently consist of content derived from various resources. The layout of the page is managed independently of the content of its subviews. For instance, a view might have subviews such as Navigation, Search, Feature Story, and Headline.

When creating a composite view, you can include static content and dynamic content. Static content might consist of an HTML file. Dynamic content is a fragment of a JSP page. The specific content can be determined at either JSP translation time or at runtime.

Front Controller

Front controllers are responsible for routing incoming requests and enforcing navigation in web applications. For more information on the use of the Front Controller design pattern, see the J2EE Patterns catalog.

JSP page

Pages that are used in web applications to present information to end users and to enable data from end users to flow back to the server. JSP pages must be placed within a web application in order for the JSP pages to be executable within the IDE.

Servlet

Servlets are Java classes that execute within a servlet container, which is typically running within a web server. Servlets are used to do the following:

- Generate dynamic content.
- Extend the capabilities of web servers and web-enabled application servers.
- Interact with web clients (typically a browser application such as Netscape or Internet Explorer) using a request-response paradigm

Send Us Your Feedback

Next Steps

For more information about using NetBeans IDE 5.5, see the following resources:

- Importing an Existing Web Application into NetBeans IDE
- Web Services (JAX-WS) in Java EE 5

To send comments and suggestions, get support, and keep informed on the latest developments on the NetBeans IDE J2EE development features, join the nbj2ee@netbeans.org mailing list. For more information about upcoming J2EE development features in NetBeans IDE, see j2ee.netbeans.org.