

CAPLIN

# Caplin Trader 2.0

---

## How To Deploy Caplin Trader Applications

May 2010

CONFIDENTIAL

# Contents

<b>1</b>	<b>Preface.....</b>	<b>1</b>
1.1	What this document contains.....	1
	About Caplin document formats .....	1
1.2	Who should read this document.....	1
1.3	Related documents.....	2
1.4	Typographical conventions.....	2
1.5	Feedback.....	3
1.6	Acknowledgments.....	3
1.7	Open Source Software .....	3
<b>2</b>	<b>How Caplin trader is deployed.....</b>	<b>4</b>
2.1	The deployment descriptor file (web.xml).....	4
2.2	Deploying a new Caplin Trader WAR file .....	4
<b>3</b>	<b>Loading the correct Caplin Trader version.....</b>	<b>5</b>
3.1	How version redirection works.....	5
3.2	Version redirection in more detail.....	6
3.3	Configuring version redirection.....	12
	Version redirection for IBM Web Sphere servers .....	14
	Enabling version redirection in application.jsp .....	15
	Enabling version redirection when themes are used .....	17
	Using a fronting server .....	19
<b>4</b>	<b>Downloading Caplin Trader resources efficiently.....</b>	<b>21</b>
4.1	The Expiry Time (Cache Files) Filter.....	21
4.2	The GZip Filter.....	22
<b>5</b>	<b>Accessing themed resources.....</b>	<b>23</b>
5.1	Setting the theme dynamically.....	25
<b>6</b>	<b>Appendix A: Background to version redirection.....</b>	<b>26</b>
<b>7</b>	<b>Appendix B: Filter definitions in web.xml.....</b>	<b>28</b>
<b>8</b>	<b>Glossary of terms and acronyms.....</b>	<b>29</b>

# 1 Preface

## 1.1 What this document contains

This document contains information on how to deploy for live operation an Ajax-based client application that has been built using the Caplin Trader framework.

**Tip:** For convenience, the client application being deployed is referred to in the rest of this document as “the Caplin Trader application”.

### About Caplin document formats

This document is supplied in three formats:

- ◆ Portable document format (*.PDF* file), which you can read on-line using a suitable PDF reader such as Adobe Reader®. This version of the document is formatted as a printable manual; you can print it from the PDF reader.
- ◆ Web pages (*.HTML* files), which you can read on-line using a web browser. To read the web version of the document navigate to the *HTMLDoc\_m\_n* folder and open the file *index.html*.
- ◆ Microsoft HTML Help (*.CHM* file), which is an HTML format contained in a single file. To read a *.CHM* file just open it – no web browser is needed.

#### For the best reading experience

On the machine where your browser or PDF reader runs, install the following Microsoft Windows® fonts: Arial, Courier New, Times New Roman, Tahoma. You must have a suitable Microsoft license to use these fonts.

#### Restrictions on viewing .CHM files

You can only read *.CHM* files from Microsoft Windows.

Microsoft Windows security restrictions may prevent you from viewing the content of *.CHM* files that are located on network drives. To fix this either copy the file to a local hard drive on your PC (for example the Desktop), or ask your System Administrator to grant access to the file across the network. For more information see the Microsoft knowledge base article at <http://support.microsoft.com/kb/896054/>.

## 1.2 Who should read this document

This document is intended for System Administrators and Developers who are responsible for deploying a new version of a Caplin Trader application in a live environment.

**Note:** Caplin Trader applications are deployed on industry standard Java™ application servers, such as JBoss, Oracle® WebLogic, and IBM® WebSphere®. This document assumes the reader is familiar with how such servers are configured and managed, and how in general to configure applications for deployment on them.

### 1.3 Related documents

## ◆ Caplin Trader Overview

A business and technical overview of Caplin Trader.

## ◆ How To Troubleshoot Client Connection Problems

Describes some typical problems that end-users might experience when a Caplin Trader application is attempting to communicate with servers such as the application server and Caplin Liberator.

## 1.4 Typographical conventions

The following typographical conventions are used to identify particular elements within the text.

Type	Uses
<b>aMethod</b>	Function or method name
<i>aParameter</i>	Parameter or variable name
<i>/AFolder/Afile.txt</i>	File names, folders and directories
<div>Some code;</div>	Program output and code examples
The value=10 attribute is...	Code fragment in line with normal text
Some text in a dialog box	Dialog box output
Something typed in	User input – things you type at the computer keyboard
<b>XYZ Product Overview</b>	Document name
◆	Information bullet point
■	Action bullet point – an action you should perform

**Note:** Important Notes are enclosed within a box like this.  
Please pay particular attention to these points to ensure proper configuration and operation of the solution.

**Tip:** Useful information is enclosed within a box like this.  
Use these points to find out where to get more help on a topic.

Information about the applicability of a section is enclosed in a box like this.  
For example: “This section only applies to version 1.3 of the product.”

## 1.5 Feedback

Customer feedback can only improve the quality of our product documentation, and we would welcome any comments, criticisms or suggestions you may have regarding this document.

Visit our feedback web page at <https://support.caplin.com/documentfeedback/>.

## 1.6 Acknowledgments

*Adobe® Reader* is a registered trademark of Adobe Systems Incorporated in the United States and/or other countries.

*Windows* is a registered trademark of Microsoft Corporation in the United States and other countries.


*IBM* and *WebSphere* are trademarks of International Business Machines Corporation in the United States, other countries, or both.

*Oracle* is a registered trademark of Oracle and/or its affiliates.

*Java* and *JSP* are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. or other countries.

## 1.7 Open Source Software

Caplin Trader incorporates the following Open Source software:

Open Source item	Use in Caplin Trader	Further information
gzip	The gzip compression algorithm is used (via standard Java utility classes) to compress Caplin Trader application resource files on the application server before they are downloaded to client browsers. See <a href="#">The GZip Filter</a>  .	<a href="http://www.gzip.org">http://www.gzip.org</a> <a href="http://java.sun.com/javase/6/docs/api/java/util/zip/package-summary.html">http://java.sun.com/javase/6/docs/api/java/util/zip/package-summary.html</a>

## 2 How Caplin trader is deployed

A Caplin Trader application is served to client browsers from a Java application server. On most application servers, deployable applications are packaged as WAR files. The entry point to a Caplin Trader application is through a JavaServer Page (JSP™); in the Caplin Trader Reference Implementation this page is called *application.jsp*.

If the Caplin Trader application is deployed as a WAR file named *caplintrader.war*, and the URL of the application server is `https://www.example.com`, then the absolute URL of the Caplin Trader application would be:

```
https://www.example.com/caplintrader/application.jsp
```

This URL would typically be made available to end-users as a link on a portal web site; the end-user can click on the link to start the application. An end-user can also bookmark the URL in their web browser and subsequently start Caplin Trader by selecting that bookmark.

In the example above, the relative URL **caplintrader** is taken from the name of the WAR file (*caplintrader.war*), but most application servers allow you to override this default name.

**Note:** Caplin Trader applications must be deployed on a Java application server that implements version 2.4 or later of the Java Servlet Specification.

**Note:** The rest of this document assumes that your Caplin Trader application is deployed as a WAR file.

**Tip:** For detailed information on how to deploy applications on your application server, refer to the documentation supplied with the server.

### 2.1 The deployment descriptor file (web.xml)

Web applications hosted on Java application servers are configured for deployment by means of a deployment descriptor file. This XML format file is called *web.xml* and is located in the server's *WEB-INF* folder. The format of the XML configuration is an industry standard (for example, see the Web Application Deployment Descriptor schema on the Sun Developer Network web site at <http://java.sun.com/xml/ns/javaee/index.html>).

**Tip:** An example *web.xml* file is supplied with the Caplin Trader Reference Implementation; you can modify this file as required, for use when deploying your own Caplin Trader application.

### 2.2 Deploying a new Caplin Trader WAR file

- To deploy a new version of your Caplin Trader application, replace the existing WAR file on the application server with the new WAR file.

For detailed information on how to do this, refer to the documentation provided with your application server.

**Tip:** A typical method of deployment is to physically overwrite the existing WAR with the new one.

## 3 Loading the correct Caplin Trader version

When an end-user runs a Caplin Trader application for the first time, the browser caches the majority of the application files. This ensures that the next time the end-user runs the Caplin Trader application in the same browser, the application starts up faster because the browser already has the majority of the application files; the files do not need to be served by the application server again.

But how does the browser get the latest application files when a new version of the Caplin Trader application is installed on the application server? The solution is through a version redirection facility built into the deployed application. Before deploying the Caplin Trader application, you will need to configure the application and the WAR with information about the new version (for details, see [Configuring version redirection](#) <sup>[12]</sup>).

### 3.1 How version redirection works

Version redirection ensures that when a new version of a Caplin Trader application has been deployed on the application server, client browsers are served with a complete set of application files for this version. In summary, it works like this:

- ◆ The entry point to the Caplin Trader application (a JavaServer page, called *application.jsp* in the Reference Implementation), is never cached at the browser. A filter at the application server sets a cache header that ensures the browser always picks up the latest entry point JSP.
- ◆ This entry point JSP includes an HTML `<base>` tag that specifies a base URL for all application file requests. The base URL includes the version number of the Caplin Trader application. If the browser does not have application files that match this version number in its cache, then new application files are requested from the application server.
- ◆ A filter at the application server (the Version Redirection Filter) removes the version number from all URLs before locating the requested resources. This is necessary because Caplin Trader application code does not include version information in its resource folder names.

This solution has the benefit that the URL of the Caplin Trader application “home page” does not need to change when a new version is installed on the application server, so bookmarks to Caplin Trader remain valid. Cached Caplin Trader files can also have a long expiry time set, so the browser does not have to query the server to check for their freshness (see [The Expiry Time \(Cache Files\) Filter](#) <sup>[21]</sup>).

Files relating to older versions of the Caplin trader application remain in the browser cache until removed during normal browser housekeeping, but are no longer used by the application.

Also see [Appendix A: Background to version redirection](#) <sup>[26]</sup>.

## 3.2 Version redirection in more detail

The following diagrams illustrate how version redirection ensures that a newly deployed version of the Caplin Trader application is always fully downloaded to requesting clients, by passing resource requests through various filters on the application server. These filters are defined in the application server's [deployment descriptor file](#) <sup>4</sup> (*WEB-INF/web.xml*).

### URLs for requesting Caplin Trader resources

Assume the URL of the server hosting the Caplin Trader application is:

```
https://www.example.com
```

and the deployed application is accessed on the server through the relative URL `caplintrader` (see note below).

The entry point to Caplin Trader is through a Java server page called *application.jsp*, so in this case, the absolute URL of the Caplin Trader application would be:

```
https://www.example.com/caplintrader/application.jsp.
```

This URL would typically be presented to end-users as a link on a portal web site; the end-user clicks on the link to start Caplin Trader. They might also bookmark the link in their web browser.

When the Caplin Trader application subsequently requests resources, the URLs specifying those resources are relative to the path specified in the entry point URL. For example, in the Caplin Trader Reference Implementation, one of the first resources required is the JavaScript file *bootstrap.js*, which is located on the server at `dependencies/bootstrap/bootstrap.js`. When version redirection is not enabled, this resource would be requested via the URL:

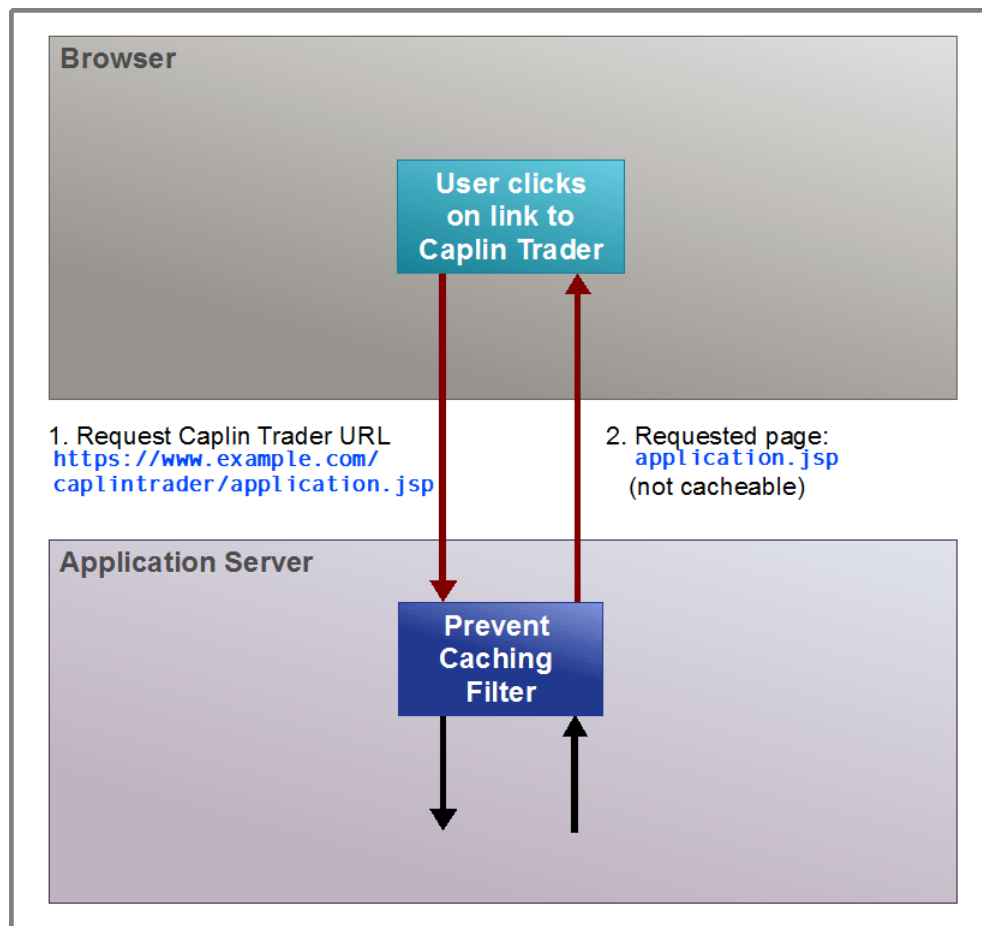
```
https://www.example.com/caplintrader/dependencies/bootstrap/bootstrap.js
```

**Note:** The default relative URL of the application is taken from the name of the WAR; for example `caplintrader` from the file name *caplintrader.war*. Most application servers also allow you to override this default through appropriate configuration – see the documentation for your particular application server.



## Requesting application.jsp via the Prevent Caching Filter

When an end-user requests the Caplin Trader application, the URL for *application.jsp* (or its equivalent) is sent to the application server, where it is passed through a Prevent Caching filter; see the flow marked “1.” in the following diagram:



### Initial request to run Caplin Trader

The Prevent Caching filter sets the cache headers for requested resources. It ensures that *application.jsp* is never cached in client browsers, so that when the version of the application changes, client requests for the base Caplin Trader URL (see the flow marked “1”) are always directed to the application server (see the flow marked “2.”). This ensures that the browser always gets the *latest* version of *application.jsp*.

The Prevent Caching filter is defined in the application server's *web.xml* file, as follows.

**Prevent Caching filter definition in *web.xml*:**

```
<filter>
  <filter-name>PreventCachingFilter</filter-name>
  <filter-class>com.caplin.appserver.utils.filters.NoCacheFilter
</filter-class>
</filter>

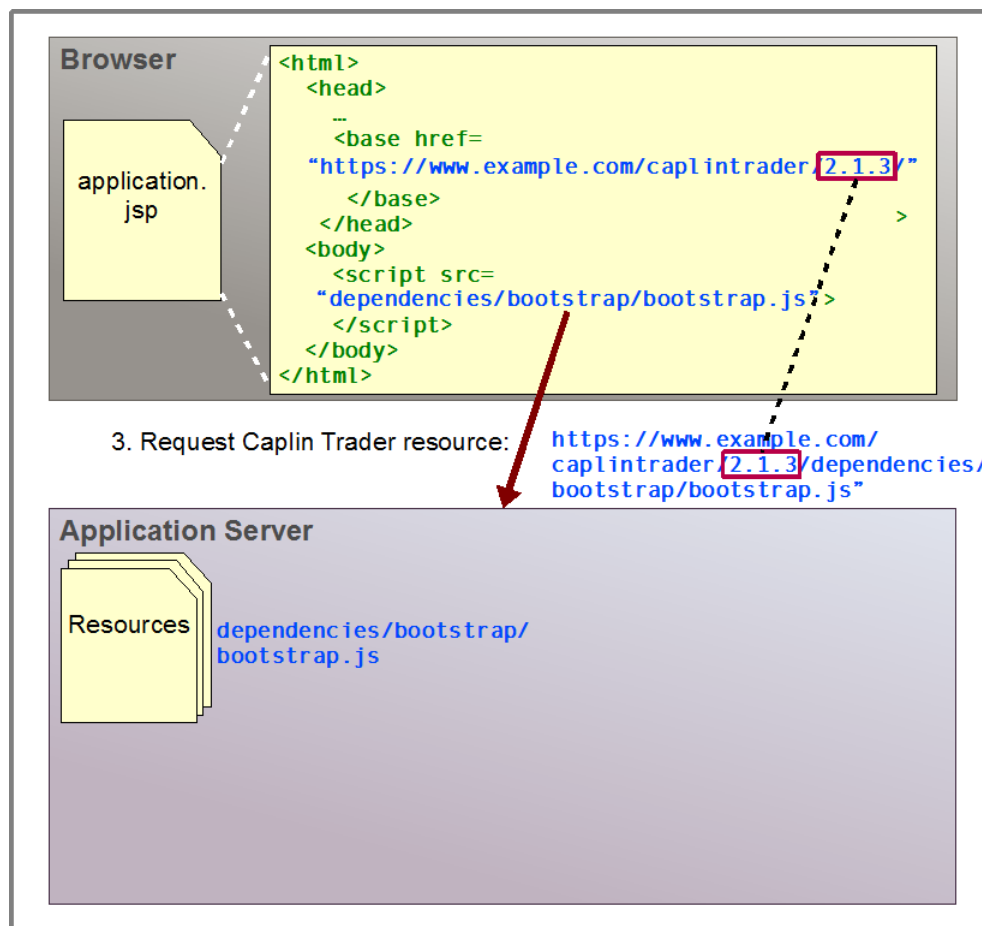
...

<filter-mapping>
  <filter-name>PreventCachingFilter</filter-name>
  <url-pattern>/public/*</url-pattern>
  <dispatcher>REQUEST</dispatcher>
  <dispatcher>FORWARD</dispatcher>
</filter-mapping>

<filter-mapping>
  <filter-name>PreventCachingFilter</filter-name>
  <url-pattern>*.jsp</url-pattern>
  <dispatcher>REQUEST</dispatcher>
  <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

## Requesting resources via the Version Redirection Filter

When the downloaded *application.jsp* runs in the client browser, it requests other Caplin Trader resources. For example, in the Caplin Trader Reference Implementation, one of the first resources requested is the JavaScript file *bootstrap.js*, as shown in the following diagram:



### Requesting a Caplin Trader resource (part 1)

The relative URL of *bootstrap.js* is `dependencies/bootstrap/bootstrap.js`. Normally this would be relative to the URL of the page that requests the resources, so in this example the absolute URL of the *bootstrap.js* file is

```
https://www.example.com/caplintrader/dependencies/bootstrap/bootstrap.js.
```

If this file is already in the browser's cache it will probably not be downloaded from the application server.

However, *application.jsp* contains an HTML `<base>` tag that defines a different base URL for relative URLs. This base URL contains the Caplin Trader application version:

```
https://www.example.com/caplintrader/2.1.3
```

In the above example, the absolute URL of *bootstrap.js* also contains the application version:

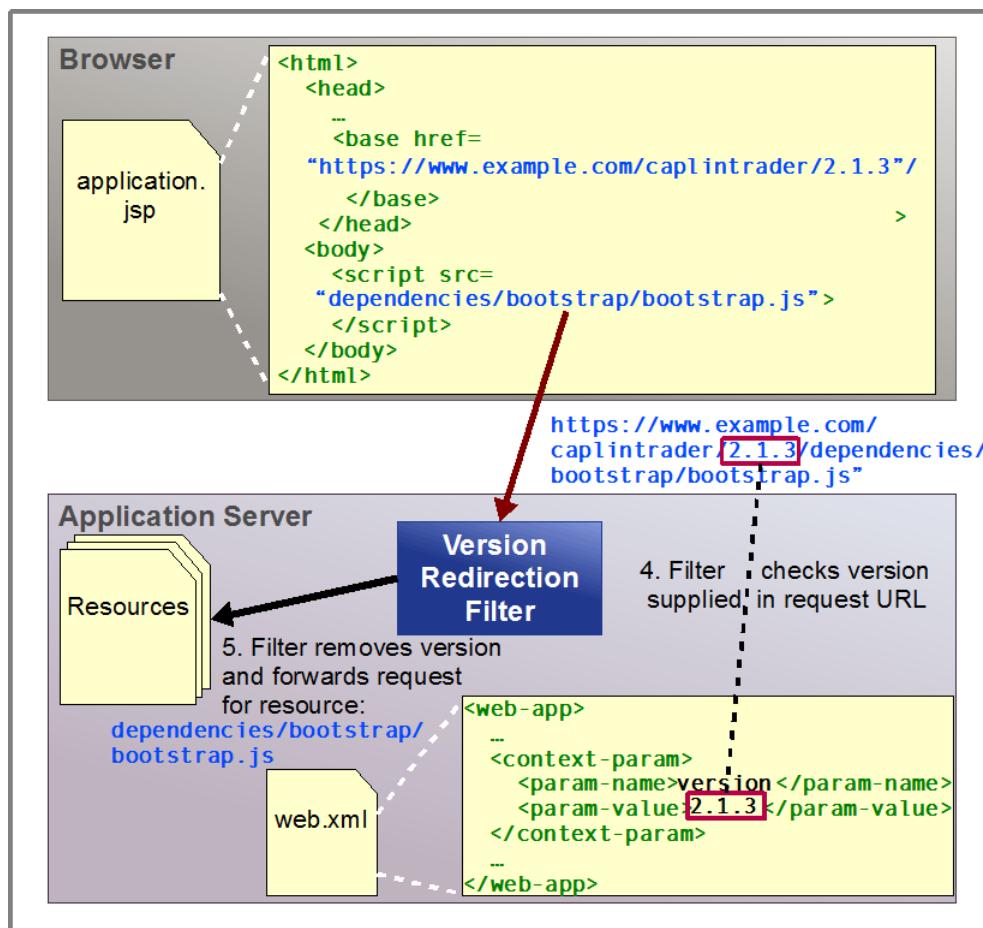
```
https://www.example.com/caplintrader/2.1.3/  
dependencies/bootstrap/bootstrap.js
```

Typically *application.jsp* creates the URL in the `<base>` tag dynamically, using information obtained from the application server; see [Enabling version redirection in application.jsp](#).

When the end-user runs the Caplin Trader application for the first time after version 2.1.3 has been deployed, the browser cache only contains a copy of *bootstrap.js* from the URL of the previous application version:

```
https://www.example.com/caplintrader/2.1.2/  
dependencies/bootstrap/bootstrap.js
```

So the browser must request the *newer* version (2.1.3) file from the application server. When the application server receives this request it passes it through a *Version Redirection Filter*, as shown in the following diagram. Because Caplin Trader code does not include version information within its resource file structure, the filter strips out the version number from resource requests, ensuring that the requests are directed to the correct folders:



Requesting a Caplin Trader resource (part 2)

The Version Redirection filter is defined in the application server's *web.xml* file, as follows.

**Version Redirection Filter definition in *web.xml*:**

```
<filter>
  <filter-name>VersionRedirectionFilter</filter-name>
  <filter-class>
    com.caplin.appserver.utils.filters.VersionRedirectionFilter
  </filter-class>
  <init-param>
    <param-name>log.level</param-name>
    <param-value>SEVERE</param-value>
  </init-param>
</filter>
```

The filter reads the value of a context parameter called `version` from the *web.xml* file.

**Context parameter `version` in *web.xml*:**

```
<web-app>
  ...
  <context-param>
    <param-name>version</param-name>
    <param-value>2.1.3</param-value>
  </context-param>
  ...
</web-app>
```

The filter then looks for this version number in the URL of the requested resource; in this case:

`https://www/example.com/caplintrader/2.1.3/dependencies/bootstrap.js`

On finding the matching version, the filter strips it out to form the version-free path *dependencies/bootstrap/bootstrap.js*. It forwards the modified request to the server, which can now find the file and return it to the client.

The filter has an associated filter mapping, which specifies that all incoming URLs matching the pattern `/*` (that is, *all* URLs) are passed to the filter.

**Version Redirection Filter mapping in *web.xml*:**

```
<filter-mapping>
  <filter-name>VersionRedirectionFilter</filter-name>
  <url-pattern>*</url-pattern>
</filter-mapping>
```

### 3.3 Configuring version redirection

Follow these steps to configure version redirection for your Caplin Trader application deployment:

1. Make sure the portal link that an end-user clicks on to start the Caplin Trader application links to a URL of the following form:

`https://{AppServerURL}/{AppPath}/{MyApplication.jsp}`

where:

- `{AppServerURL}` is the URL of the application server (for example `www.example.com`).
- `{AppPath}` is the relative path of the deployed Caplin Trader application within the server. By default this is the name of the WAR file containing the application (for example the `caplintrader` part of `caplintrader.war`).
- `{MyApplication.jsp}` is the name of the JavaServer page that loads and starts the application. In the Caplin Trader Reference Implementation, this page is called `application.jsp`.

For example, the link on the portal web site could be:

`https://www.example.com/caplintrader/application.jsp`

You can also configure your application to have a different `{AppPath}`. The documentation for your particular application server should explain how to do this.

**Note:** For security reasons, it is strongly recommended that the URL for your Caplin Trader application specifies an HTTPS connection rather than HTTP.

2. Put the correct application version number in the `version` context parameter in the `web.xml` file:

```
<web-app>
...
<context-param>
  <param-name>version</param-name>
  <param-value>2.1.3</param-value>
</context-param>
...
</web-app>
```

**Tip:** In the Caplin Trader Reference Implementation, the version number is automatically inserted in the `web.xml` file when the WAR is built from source code. You may wish to adopt this approach for your own Caplin Trader application.

3. Make sure that in *web.xml* the filter mapping for the Version Redirection Filter is the *first* mapping to be specified, so that at run time the filter is the first one to be called in the chain.  
(At deployment time, the filter chain is constructed in the order that the filter mappings are declared in the *web.xml* file.)

**Specify Version Redirection Filter mapping first in the list in *web.xml*:**

```
<filter-mapping>
  <filter-name>VersionRedirectionFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>

<filter-mapping>
  <filter-name>GZipFilter</filter-name>
  <url-pattern>*.js</url-pattern>
  <dispatcher>REQUEST</dispatcher>
  <dispatcher>FORWARD</dispatcher>
</filter-mapping>
<filter-mapping>
...

```

4. Make sure all the other filters in *web.xml* are set up correctly. See the sections about filters in [Downloading Caplin Trader resources efficiently](#)<sup>[21]</sup> and [Accessing themed resources](#)<sup>[23]</sup>.

The following sections discuss some additional points that you may need to consider when configuring version redirection.

- ◆ [Version redirection for IBM Web Sphere servers](#)<sup>[14]</sup>
- ◆ [Enabling version redirection in application.jsp](#)<sup>[15]</sup>
- ◆ [Enabling version redirection when themes are used](#)<sup>[17]</sup>
- ◆ [Using a fronting server](#)<sup>[19]</sup>

## Version redirection for IBM Web Sphere servers

Version **6.1** of the IBM WebSphere application server has a bug that prevents the Version Redirection Filter from behaving correctly.

When the WebSphere server receives a request for a URL with a Caplin Trader application version in it, such as:

```
https://www.example.com/caplintrader/2.1.3/dependencies/bootstrap/bootstrap.js
```

it looks for this resource on the server but does not find it, because the resource is actually at:

```
https://www.example.com/caplintrader/dependencies/bootstrap/bootstrap.js
```

The bug means that when the server decides the resource does not *physically* exist, it returns a "404 FileNotFoundException" to the client even though there is a filter (in our case the Version Redirection Filter) to handle the request. As a result, the filter is never called.

- There is a fix for this fault posted on the IBM support web site at <http://www-01.ibm.com/support/docview.wss?uid=swg24014758>.
- When you have applied the fix, you must also enable it to ensure that the Version Redirection Filter is called when the server decides that the requested resource is not physically present at the specified URL.

To enable the fix, set the Webcontainer custom property  
`com.ibm.ws.webcontainer.invokefilterscompatibility`  
to the value `'true'`.



## Enabling version redirection in application.jsp

The entry point to the Caplin Trader Reference Implementation is through the JavaServer page *application.jsp*. The code in this page dynamically constructs the `<base>` tag for the application. This allows the application to automatically enable version redirection for live deployments, but turns it off in development environments where developers need finer control over which files are replaced in the browser cache.

You may wish to adopt a similar approach for your Caplin Trader application. In the Reference Implementation:

- ◆ *application.jsp* queries the server to construct the string that defines the base URL (for example, `https://www.example.com/caplintrader/`). It also queries the server to obtain the version of the Caplin Trader application from the `version` context parameter in the *web.xml* file.
- ◆ The `version` context parameter controls whether version redirection is turned on or off. In development environments `version` has the value `@version@`. If `version` is set to `@version@`, or is `null` (because it is not defined in *web.xml*), *application.jsp* does not generate a `<base>` tag; hence version redirection is turned off.
- ◆ When the Caplin Trader application is built for live deployment, `version` is set to the actual version of the application (at Caplin this is done automatically as part of the build process). When *application.jsp* finds that `version` is not `null` and is not set to `@version@`, it generates a `<base>` tag containing the application version, so that version redirection is turned on.

The code in *application.jsp* that enables or disables version redirection could look like this:

```
...
<%
// Get the individual parts of the URL for the <base> tag:

String baseTag = "";
String baseUrl = "";
String version = application.getInitParameter("version");
// For example, "2.1.3"
String baseVersion = "";

String separator = application.getInitParameter("separator");
if (separator == null)
{
    separator = "/";
}

String contextPath = request.getContextPath(); // For example, "/caplintrader"
String scheme = request.getScheme(); // "http", "https", ...
String serverName = request.getServerName(); // For example, "www.example.com"
int serverPort = request.getServerPort(); // For example, "8080"

baseUrl =
    scheme + "://" + serverName + ":" + serverPort + contextPath + separator;
// For example, "https://www.example.com:8080/caplintrader/"

// Only generate a <base> tag if there is a context parameter
// called version in the web.xml file on the application server
// AND the value of version is not "@version@".

if (version != null && !version.equals("@version@"))
{
    baseVersion = version + separator; // For example, "2.1.3/"
    baseTag = "<base href=\"" + baseUrl + baseVersion + "\"></base>";
    // For example, the <base> tag string could be:
    // "<base href=\"https://www.example.com:8080/caplintrader/2.1.3/></base>"
}
%>
...

<html>
<head>
...
<%=baseTag%>
...
</head>
...

```

**Note:** If access to Caplin Trader resources is through a fronting server, the above code may not be sufficient. See [Using a fronting server](#) <sup>19</sup>.

## Enabling version redirection when themes are used

Your Caplin Trader application may support alternative themes (look, feel, and configuration) by redirecting requests for images and other resources to one of a number of different theme directories (for more information about this see [Accessing themed resources](#)<sup>[23]</sup>). If the application does this, the theme setting should be included in the `<base>` tag URL, so that when the end-user changes theme, all the required resources for the new theme are downloaded to the browser.

The following code shows how *application.jsp* can be changed to dynamically set the theme in the `<base>` tag. The code assumes that the required theme has been determined by some means (for example, by allowing the end-user to select a theme from a drop-down list when they log in), and the selected theme setting has been stored in a server session attribute called `theme`.

The Version Redirection Filter removes the theme information from the URL of the requested resource, provided it appears in the correct position in the URL, as shown in the following example.

The code in *application.jsp* when themes are used:

```
...
<%
// Get the individual parts of the URL for the <base> tag:

String baseTag = "";
String baseUrl = "";
String version = application.getInitParameter("version");
// For example, "2.1.3"

String baseVersion = "";

String separator = application.getInitParameter("separator");
if (separator == null)
{
    separator = "/";
}

String contextPath = request.getContextPath(); // For example, "/caplintrader"
String scheme = request.getScheme(); // "http", "https", ...
String serverName = request.getServerName(); // For example, "www.example.com"
int serverPort = request.getServerPort(); // For example, "8080"

baseUrl =
    scheme + "://" + serverName + ":" + serverPort + contextPath + separator;
// For example, "https://www.example.com:8080/caplintrader/"

// Only generate a <base> tag if there is a context parameter
// called version in the web.xml file on the application server
// AND the value of version is not "@version@"

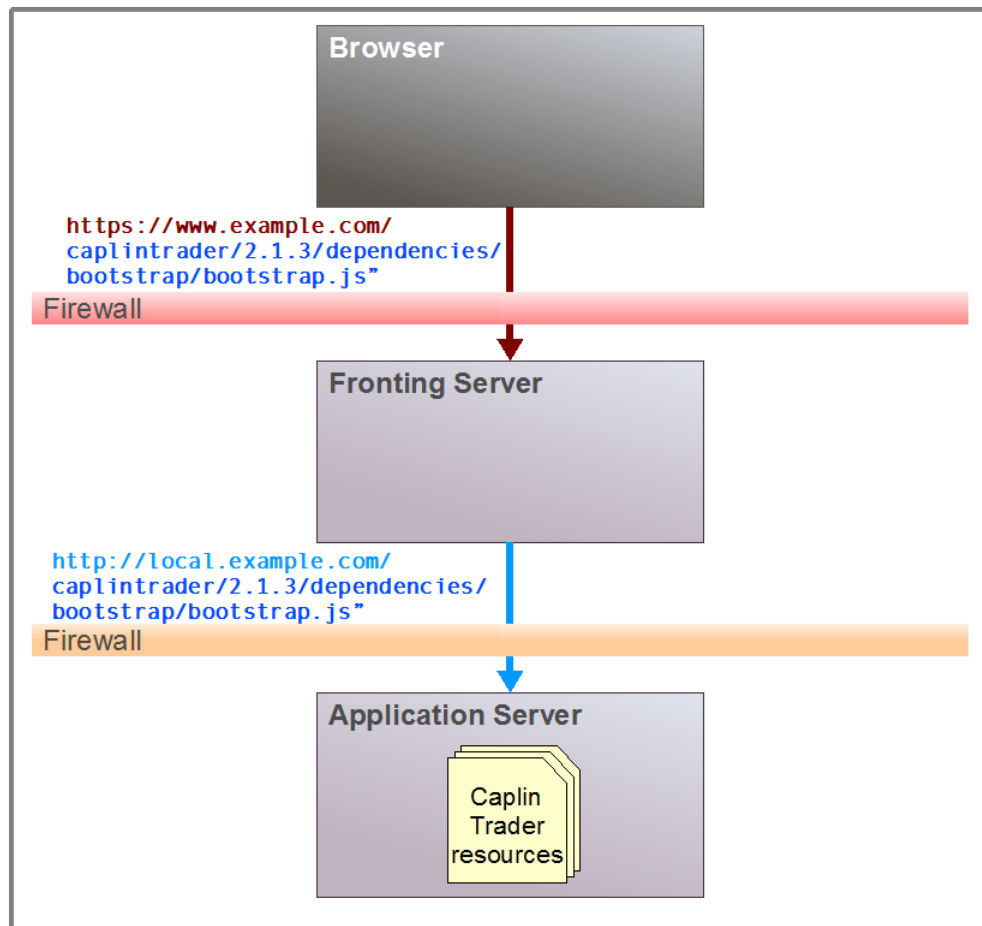
if (version != null && !version.equals("@version@"))
{
    baseVersion = version + separator; // For example, "2.1.3/"
    String currentTheme = (String)session.getAttribute("theme");
    baseTag = "<base href=\"" + baseUrl + baseVersion
        + currentTheme + separator + "\"></base>";
    // For example, the <base> tag string could be:
    // "<base href=\"https://www.example.com:8080/caplintrader/2.1.3/pastel/>
    // </base>"
    // where the theme is identified as "pastel".
}
%>
...

<html>
    <head>
        ...
        <%=baseTag%>
        ...
    </head>
    ...

```

## Using a fronting server

For security reasons, access to Caplin Trader resources may be through a fronting server, as the following diagram illustrates:



### Accessing Caplin Trader resources via a fronting server

The diagram above is a typical example of a publicly available fronting server that handles requests for Caplin Trader resources, which in this case are located at `https://www.example.com/caplintrader`.

The fronting server forwards requests to the relevant application server, *modifying the URL of the requested resource* in the process. In the request for `bootstrap.js` shown above, the fronting server changes the server URL from `www.example.com` to `local.example.com`. Because the application server is behind a firewall, the fronting server also changes the protocol from HTTPS to HTTP.

When a fronting server is used in this way, the example code shown in [Enabling version redirection in application.jsp](#) will not work. It fails because version redirection generates a base tag in `application.jsp` with the wrong protocol and server URL (that of the fronting server rather than that of the application server). In the diagram above, the resource requested by the fronting server has the correct version number (2.1.3), but the wrong protocol and server URL (`http://local.example.com` instead of `https://www.example.com`).

One solution to this problem is to configure the fronting server to add an additional HTTP header to the URL before it forwards the URL on to the application server. The additional header must contain the URL of the application server and the relative path of the deployed application on that server. The additional header is available to *application.jsp*, which can use it to generate the correct `<base>` tag at the application server.

For example, if the fronting server is configured to add the HTTP header `ClientURL` (set to `"https://www.example.com:8080/caplintrader"`), the code in *application.jsp* would look something like this (the places where this code differs from the example in [Enabling version redirection in application.jsp](#)<sup>[15]</sup> are highlighted):

```
...
<%
// Get the individual parts of the URL for the <base> tag:

String baseTag = "";
String baseUrl = "";
String clientUrl = request.getHeader("ClientUrl");
// The HTTP header "ClientURL" holds the URL
// of the original request for application.jsp.
// For example, "https://www.example.com:8080/caplintrader"

String version = application.getInitParameter("version");
// For example, "2.1.3"
String baseVersion = "";

String separator = application.getInitParameter("separator");
if (separator == null)
{
    separator = "/";
}

baseUrl =
    scheme + "://" + serverName + ":" + serverPort + contextPath + separator;
// For example, "https://www.example.com:8080/caplintrader/"

// Only generate a <base> tag if the client URL is available in
// the request header called ClientURL
// AND there is a context parameter
// called version in the web.xml file on the application server
// AND the value of version is not "@version@"

if (clientURL != null && version != null && !version.equals("@version@"))
{
    baseUrl = clientUrl + separator;
    // For example, "https://www.example.com:8080/caplintrader/"
    baseVersion = version + separator; // For example, "2.1.3/"
    baseTag = "<base href=\"\" + clientURL + baseVersion + "\"></base>";
    // For example, the <base> tag string could be:
    // "<base href=https://www.example.com:8080/caplintrader/2.1.3/></base>"
}
%>
...

<html>
<head>
...
<%=baseTag%>
...
</head>
...

```

## 4 Downloading Caplin Trader resources efficiently

For a good user experience, it is important that when an end-user requests the Caplin Trader application, it starts up as quickly as possible. Several mechanisms are provided to optimize the application loading time. These are:

- ◆ The Expiry Time (Cache Files) Filter – sets the expiry time on files cached in the browser.
- ◆ The GZip Filter – compresses files before they are downloaded.

### 4.1 The Expiry Time (Cache Files) Filter

The Caplin Trader web application's *web.xml* file defines an Expiry Time filter that is installed on the application server when the Caplin Trader application is deployed. This filter sets the expiry time of Caplin Trader web pages to a large interval (currently a year).

The large expiry time helps to reduce client-server traffic when the browser checks pages in its cache for validity. When an end-user starts the Caplin Trader application for the second and subsequent times after the application version has changed, the vast majority of required files should already be in the cache and the files will not have expired. (This is as long as the end-user has not cleared their browser cache since the first time the application was loaded.) The application therefore loads and starts faster and consumes fewer network resources when loading.

- To ensure end-users experience the shortest possible delay between requesting the Caplin Trader application and it starting, make sure this filter is specified in *web.xml*, as follows:

**Expiry Time (Cache Files) Filter definition in *web.xml*:**

```
<filter>
  <filter-name>CacheFilesFilter</filter-name>
  <filter-class>com.caplin.appserver.utils.filters.ExpiryFilter
  </filter-class>
</filter>
...

<filter-mapping>
  <filter-name>CacheFilesFilter</filter-name>
  <url-pattern>/source/themes/*</url-pattern>
  <dispatcher>REQUEST</dispatcher>
  <dispatcher>FORWARD</dispatcher>
</filter-mapping>

<filter-mapping>
  <filter-name>CacheFilesFilter</filter-name>
  ...
```

In the Caplin Trader Reference Implementation, filter mappings (`<filter-mapping>`) are supplied to ensure that files matching the following URL patterns (`<url-pattern>`) have long expiry times set:

```
/source/themes/*  
/conf/*  
*.js  
*.png  
*.gif  
*.xml  
*.css
```

## 4.2 The GZip Filter

The Caplin Trader web application's *web.xml* file defines a GZip filter. The filter uses the gzip algorithm to compress requested resources before sending them to the client (provided the requesting browser supports this). This minimizes the amount of data that has to be sent to the client, improving the load time when an end-user starts a newer version of the Caplin Trader application for the first time.

- To ensure end-users experience the shortest possible delay between requesting the Caplin Trader application and it starting, make sure this filter is specified in *web.xml*, as follows:

**GZip Filter definition in *web.xml*:**

```
<filter>  
  <filter-name>GZipFilter</filter-name>  
  <filter-class>com.caplin.appserver.utils.filters.GZipFilter</filter-class>  
</filter>  
...  
  
<filter-mapping>  
  <filter-name>GZipFilter</filter-name>  
  <url-pattern>*.js</url-pattern>  
  <dispatcher>REQUEST</dispatcher>  
  <dispatcher>FORWARD</dispatcher>  
</filter-mapping>  
  
<filter-mapping>  
  <filter-name>GZipFilter</filter-name>  
  ...
```

- Make sure the filter mappings for this filter (`<filter-mapping>`) are in the following order of `<url-pattern>`:

```
<url-pattern>*.js</url-pattern>  
<url-pattern>*.html</url-pattern>  
<url-pattern>*.jsp</url-pattern>  
<url-pattern>*.xml</url-pattern>  
<url-pattern>*.css</url-pattern>
```

The `<dispatcher>` settings are the same for each `<filter-mapping>` (see the example above).



## 5 Accessing themed resources

The Caplin Trader web application's *web.xml* file can define a Theme Redirection Filter. This is an optional filter that allows the theme (look, feel, and configuration) of a Caplin Trader application to be changed by redirecting requests for images and other resources to one of a number of different theme directories. For example, the Caplin Trader Reference Implementation has two alternative layout themes, which are called "noir" and "pastel". The Reference Implementation uses the Theme Redirection Filter to load resources for the theme that was selected when the application started up.

**Theme Redirection Filter definition in *web.xml*:**

```
<filter>
  <filter-name>ThemeRedirectionFilter</filter-name>
  <filter-class>
    com.caplin.appserver.utils.filters.ThemeRedirectionFilter
  </filter-class>
  <init-param>
    <param-name>log.level</param-name>
    <param-value>SEVERE</param-value>
  </init-param>
</filter>
...

<filter-mapping>
  <filter-name>ThemeRedirectionFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
...
```

Assume the deployed Caplin Trader application is accessed on the server through the relative URL *caplintrader*, and at run time it requests an image *myimage.gif* at the relative URL *source/images/* (say through an image tag: ``). Without the Theme Redirection Filter, the client request would load the image from the server folder *caplintrader/source/images/*.

When the Theme Redirection Filter is in place, requests for images on the relative path *source/images/* are redirected to *source/themes/%theme%/images/* where *%theme%* is the value of a session variable that is set to the theme currently in use. The initial value of *%theme%* is defined as a context parameter, in *web.xml*, as follows:

```
<web-app>
  ...
  <context-param>
    <param-name>theme</param-name>
    <param-value>pastel</param-value>
  </context-param>
  ...
</web-app>
```

With the *theme* context parameter set to *pastel*, the Theme Redirection Filter redirects a client request for *source/images/myimage.gif* to *source/themes/pastel/images/myimage.gif*.

The Theme Redirection Filter filter also redirects requests that include the path `source/theme/` to the path `source/themes/%CurrentTheme%/`. This allows you to easily include alternative configuration files that are applied according to the current theme. For example, you may wish to vary the layout of Trade Tickets depending on the theme. In the Caplin Trader Reference Implementation the configuration file `caplintrader.xml` contains the following definition for the FX Trade Ticket:

```
<showDialog id="fx-trade-ticket"
            src="source/theme/dialogs/dialog_fxticket.xml"
            defer="true"/>
```

This configuration specifies that the XML configuration for the Trade Ticket is obtained from the relative URL:

```
source/theme/dialogs/dialog_fxticket.xml"
```

At run time, the Theme Redirection Filter will change this relative URL to:

```
source/themes/%CurrentTheme%/dialogs/dialog_fxticket.xml
```

- If you have coded your Caplin Trader application to use different themes, make sure this filter is specified in the `web.xml` file.
- Specify the filter mappings for the Theme Redirection Filter in `web.xml` *after* the mappings for the Version Redirection Filter; for correct selection of resources the Version Redirection Filter must be called first.
- If you only want to set the theme according to the value of the `theme` context parameter in `web.xml` (that is, you do not want the theme to be determined dynamically when the Caplin Trader application starts), the application must execute a JavaServer page containing the following code, *before* any theme-related resources are requested. This code ensures that the Theme Redirection Filter picks up the `theme` value defined in `web.xml`.

```
<%
...
String defaultTheme = application.getInitParameter("theme");
                        // Get theme from web.xml
session.setAttribute("theme", defaultTheme);
                        // Set theme as a session variable
                        // for the Theme Redirection Filter to use.
...
%>
```

## 5.1 Setting the theme dynamically

The theme of a Caplin Trader application can also be set dynamically according to the session variable called `theme`. The Theme Redirection Filter obtains the current theme from this session variable. The Reference Implementation contains an example of how to set the theme session variable; see the listing below of the JSP file *CaplinTrader/utills/jsp/theme\_setter.jsp*.

- The code that sets the theme must be executed *before* the application requests any theme-related resources.

### JSP for setting the theme (*theme\_setter.jsp*)

```
<%@ include file="getThemeList.jsp" %>
<%
String defaultTheme = application.getInitParameter("theme");
    // Default theme is the initial theme setting
    // as defined in the web.xml <context-param> called theme.

String queryTheme = request.getParameter("theme");
    // Get the theme value passed in the request that called this page.

HashMap userThemes = getThemeList(session);
    //Get the list of themes available to this session.

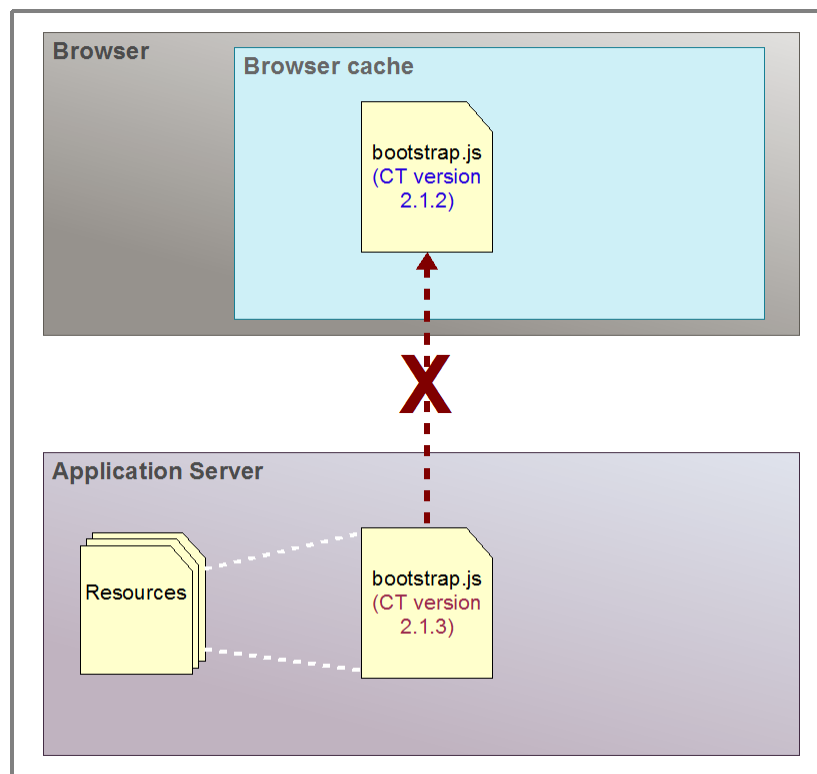
if (userThemes.size() > 0
    && queryTheme != null
    && userThemes.containsKey(queryTheme))
{
    //
    // If theme list not empty
    // AND a theme value was passed in the request
    // AND the passed theme is valid (is in the theme list)...
    //
    session.setAttribute("theme", queryTheme);
    // Set the theme for the session to be the one passed in the request.
}
else
{
    //
    // No theme passed in the request, or theme is not valid.
    //
    session.setAttribute("theme", defaultTheme);
    // Set the theme for the session to be the default one.
}
%>
```

## 6 Appendix A: Background to version redirection

When an end-user runs a Caplin Trader application, a number of Caplin Trader files are downloaded from the application server to the browser on the client, including the JavaScript code that runs the application, HTML and CSS files, XML-based configuration, and image files. Most browsers will cache these files in the same way that they cache files received from any web site. Caching helps to speed up access to the Caplin Trader application – the next time the end-user accesses the application, the startup time is reduced, as many, or even all, of the required files are already in the cache so they do not need to be downloaded again.

However, caching causes a conflict when the version of the Caplin Trader application held on the server has been updated. When a new version of the application is deployed, it is important that when the end-user next runs it, the entire new version of the application is downloaded to the client, effectively replacing any previous version(s) that may be in the browser's cache. If this did not happen, the browser would only download files that it thought, or the application server thought had changed. When the application ran it would contain a mixture of code and configuration, some being the new version and some the old. This mismatch could cause the application to fail or execute incorrectly, which would not be acceptable behavior.

The following diagram shows how browser caching prevents the client from loading updated versions of Caplin Trader files.



**Cached file is not updated**

In the diagram above, a new version of the Caplin Trader application, 2.1.3, has been deployed on the application server. This includes a new version of the JavaScript file *bootstrap.js*. However, because the old version, 2.1.2, of *bootstrap.js* is still in the browser cache from when the end-user previously ran version 2.1.2 of the application, the browser will probably not load version 2.1.3 of the file from the application server. Version 2.1.2 of *bootstrap.js* may not be compatible with other JavaScript files in Caplin

Trader application version 2.1.3. For example, the old version 2.1.2 bootstrap might not load a new essential resource for 2.1.3, causing the application to behave in an unexpected way.

A simple solution to this problem would be to ask end-users to clear their browser caches before accessing the new version of the Caplin Trader application. However, this approach is unacceptable for a number of reasons:

- ◆ Many end-users would object to it. They would rightly expect the application to run correctly without them needing to carry out housekeeping operations.
- ◆ It is impractical. There could be a large number of end-users, and many of them will be customers of the bank, not their employees.
- ◆ It will not always work. For example, different browsers have different caching strategies, and in some cases a simple housekeeping procedure may not remove all the unwanted files.

Caplin Trader therefore employs a more sophisticated approach to solving this problem, its [version redirection facility](#)<sup>[5]</sup>. Version redirection allows clients to get the most benefit from browser caching without compromising the ability to replace the cached version of the Caplin Trader application with a newer version.

#### Preventing unnecessary “freshness” queries

In most browsers, expiry dates are not set on cached files by default. As a result, even when requested resources are already in the browser cache, the browser must still query the application server to find out whether each such resource is still “fresh”; that is, whether it has changed on the server since it was cached in the browser.

For a Caplin Trader application, which consists of large numbers of small files (such as JavaScript files, XML, CSS, HTML, and image files), this would cause a large amount of unnecessary query—response traffic between the client and the application server. This traffic could significantly increase the application start-up time, even when the correct version of the application is already in the browser cache.

To mitigate this problem, Caplin Trader deployments should ensure that all downloaded resources have a long expiry time set against them. When this is done, the browser only needs to request a resource from the application server if the resource expiry time has been exceeded, or if the Caplin Trader version has changed. Resource expiry times are set by the [Expiry Time \(Cache Files\) Filter](#)<sup>[2]</sup> on the application server.

## 7 Appendix B: Filter definitions in web.xml

This appendix lists the Caplin Trader specific filters that are defined in the *web.xml* file for a Caplin Trader web application. These are the filters that are shipped with the Caplin Trader Reference Implementation; the definition of these filters may be different in your Caplin Trader application.

Filter	For description see:
Prevent Caching Filter	<a href="#">Requesting application.jsp via the Prevent Caching Filter</a> <sup>7</sup>
Version Redirection Filter	<a href="#">Requesting resources via the Version Redirection Filter</a> <sup>9</sup>
Expiry Time (Cache Files) Filter	<a href="#">The Expiry Time (Cache Files) Filter</a> <sup>21</sup>
GZip filter	<a href="#">The GZip filter</a> <sup>22</sup>
Theme Redirection Filter	<a href="#">Accessing themed resources</a> <sup>23</sup>

## 8 Glossary of terms and acronyms

This section contains a glossary of terms, abbreviations, and acronyms relating to deploying Caplin Trader applications.

Term	Definition
<b>Ajax</b>	<p><u>A</u>syncronous <u>J</u>avaScript and <u>X</u>ML.</p> <p>A combination of Web technologies used to implement interactive Web clients.</p>
<b>Application server</b>	<p>Software that serves up web pages (typically with dynamically constructed content) and web-applications, for rendering or execution by client web browsers.</p> <p><b>Caplin Trader applications</b> are served from a <b>Java application server</b>. In this document the term “application server” means “Java application server”.</p>
<b>Caplin Liberator</b>	<p>Caplin Liberator is a real-time financial internet hub that delivers trade messages and market data to and from subscribers over any network.</p>
<b>Caplin Trader</b>	<p>A <b>Rich Internet Application</b> framework for constructing <b>Caplin Xaqua client</b> applications for browser-based trading. It includes a comprehensive set of trading GUI components.</p>
<b>Caplin Trader application</b>	<p>A <b>Caplin Xaqua client</b> that has been built using <b>Caplin Trader</b>.</p>
<b>Caplin Trader Reference Implementation</b>	<p><b>Caplin Trader</b> ships with a Reference Implementation. This is a <b>Caplin Trader application</b> that provides a fully-functional, single-dealer trading client that can easily be modified, customized, and extended to create bespoke offerings.</p>
<b>Caplin Xaqua</b>	<p>A framework for building single-dealer platforms that enables banks to deliver multi-product trading direct to client desktops.</p>
<b>Caplin Xaqua client</b>	<p>A client desktop application that interfaces with <b>Caplin Xaqua</b> to deliver multi-product trading to end-users. The application can be implemented in any technology that is supported by Caplin Xaqua.</p>
<b>Deployment descriptor</b>	<p>In the Java Platform, Enterprise Edition, a deployment descriptor describes how a web application or enterprise application should be deployed. It directs a deployment tool to deploy a module or application with specific container options, security settings and describes specific configuration requirements. XML is used for the syntax of these deployment descriptor files. For web applications, the deployment descriptor must be called <i>web.xml</i> and must reside in a <i>WEB-INF</i> subdirectory at the web application root.</p> <p>Definition from Wikipedia contributors, “Deployment Descriptor”, <i>Wikipedia, The Free Encyclopedia</i>, <a href="http://en.wikipedia.org/w/index.php?title=Deployment_Descriptor&amp;oldid=336293377">http://en.wikipedia.org/w/index.php?title=Deployment_Descriptor&amp;oldid=336293377</a> (accessed February 23, 2010).</p>
<b>End-user</b>	<p>A person who uses a piece of software for its intended purpose. For example, financial traders are the end-users of a <b>Caplin Trader application</b>.</p>

Term	Definition
<b>Java application server</b>	<p>An application server that hosts Java Server Pages (<b>JSPs</b>) and Java Servlets. Such servers can also host resources implemented in other technologies, such as HTML, CSS, XML, and JavaScript.</p> <p><b>Caplin Trader applications</b> are served from a <b>Java application server</b>.</p>
<b>JSP</b>	See <b>JavaServer Page</b> .
<b>JavaServer Page</b>	A Java technology for serving dynamically generated Web pages
<b>Live operation</b>	The process of running some software to perform tasks in the real world, as opposed to running it just for test purposes.
<b>Reference Implementation</b>	The <b>Caplin Trader Reference Implementation</b> .
<b>Rich Internet Application</b>	A web application, such as an application implemented using <b>Caplin Trader</b> , that has the features and functions of a desktop application, but which does not need to be installed on the client platform. Rich Internet Applications typically run in web browsers.
<b>WAR</b>	<p><u>Web Application Archive</u></p> <p>A JAR file used to distribute a collection of JavaServer Pages, servlets, Java classes, XML files, tag libraries and static Web pages (HTML and related files) that together constitute a Web application.</p> <p>Definition from Wikipedia contributors, "WAR (Sun file format)", <i>Wikipedia, The Free Encyclopedia</i>, <a href="http://en.wikipedia.org/wiki/Web_application_archive">http://en.wikipedia.org/wiki/Web_application_archive</a> (accessed February 2010).</p>



## Contact Us

Caplin Systems Ltd  
Triton Court  
14 Finsbury Square  
London EC2A 1BR  
Telephone: +44 20 7826 9600  
Fax: +44 20 7826 9610  
[www.caplin.com](http://www.caplin.com)

The information contained in this publication is subject to UK, US and international copyright laws and treaties and all rights are reserved. No part of this publication may be reproduced or transmitted in any form or by any means without the written authorization of an Officer of Caplin Systems Limited.

Various Caplin technologies described in this document are the subject of patent applications. All trademarks, company names, logos and service marks/names ("Marks") displayed in this publication are the property of Caplin or other third parties and may be registered trademarks. You are not permitted to use any Mark without the prior written consent of Caplin or the owner of that Mark.

This publication is provided "as is" without warranty of any kind, either express or implied, including, but not limited to, warranties of merchantability, fitness for a particular purpose, or non-infringement.

This publication could include technical inaccuracies or typographical errors and is subject to change without notice. Changes are periodically added to the information herein; these changes will be incorporated in new editions of this publication.

Caplin Systems Limited may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

This publication may contain links to third-party web sites; Caplin Systems Limited is not responsible for the content of such sites.