CAPLIN

Caplin Trader 2.0

JavaScript Preprocessor Configuration XML Reference

October 2010

Contents

1 Preface

1.1 What this document contains

This document describes the XML-based configuration that determines which files are served by the Caplin Trader JavaScript Preprocessor.

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 Software Developers who need to configure an application that is based on the Caplin Trader framework.

1.3 Related documents

- **Caplin Trader 2.0: Installation Guide**

  This document describes how to install Caplin Trader version 2.0 for evaluation, and for use in developing a Caplin Trader application.
- **Caplin Trader Overview**

  A business and technical overview of Caplin Trader.

1.4 Typographical conventions

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

Type	***Uses***
aMethod	Function or method name
aParameter	Parameter or variable name
/AFolder/Afile.txt	File names, folders and directories
`Some code;`	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
XYZ Product Overview	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.

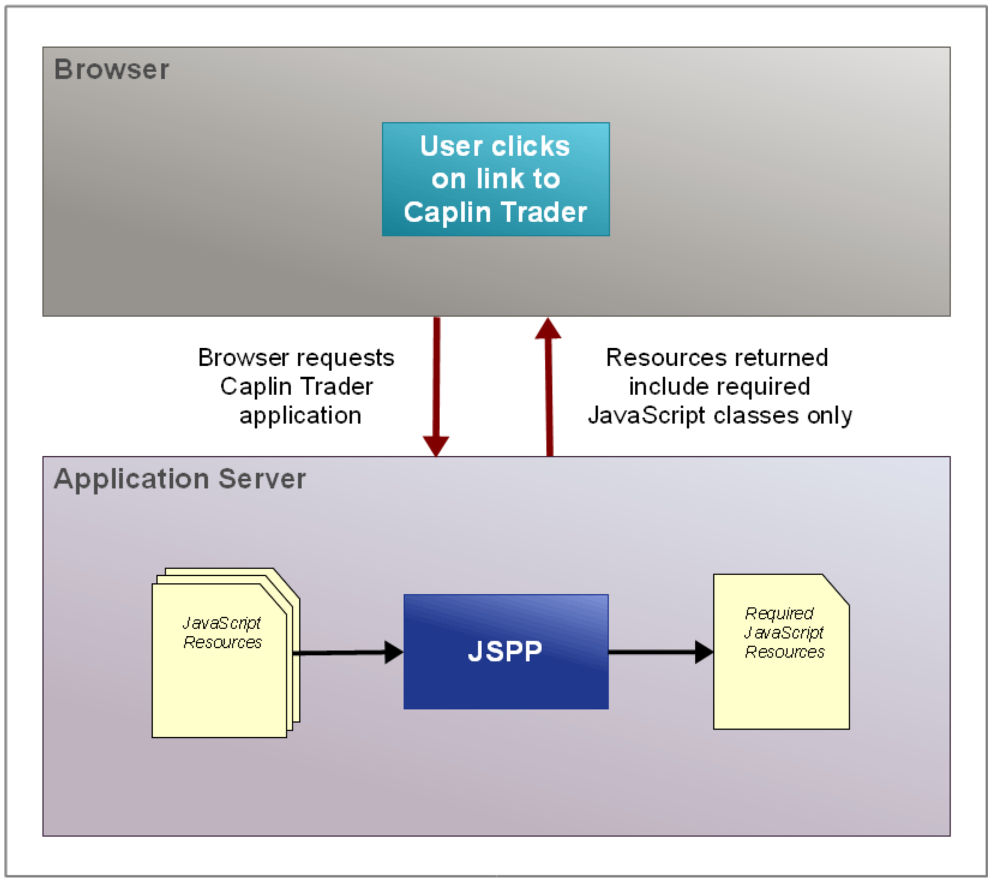
Java and *JavaScript* are trademarks of Sun Microsystems, Inc. in the U.S. or other countries.

2 Overview

The JavaScript Preprocessor (JSPP) is a Java servlet that resides on the application server, and runs when the server receives a browser request for the Caplin Trader application. Its function is to ensure faster application start up and to reduce the application payload in the client browser.

To achieve this, the JSPP parses the dependency tree of the JavaScript classes that comprise the application, and selects just those classes that are actually used. A particular application will typically not use all the Caplin Trader framework classes, so the JSPP ensures that only the classes required are actually sent to the client. It also concatenates the JavaScript class files held on the server into a single file, with the classes arranged in dependency order.

The JSPP can also be configured to select and concatenate source files from one of several different locations, depending on the value of a query string in the URL received from the client. This feature can be used in a development environment to run either the development version of the application or the built version of the application.


Browser
User clicks on link to Caplin Trader
Browser requests Caplin Trader application
Resources returned include required JavaScript classes only
Application Server
JavaScript Resources
JSPP
Required JavaScript Resources

JSPP selects only those JavaScript classes required by the application

3 Technical assumptions

The JSPP is supplied with the Caplin Trader development framework. To configure and use the JSPP as described in this document, your application must meet the following criteria:

1. The application must be developed using the Caplin Trader framework, version 2.0.0 or later.
2. The application must be served from an application server that supports JavaServer Page (JSP) technology.
3. The entry point to the application must be a JavaServer Page (such as *application.jsp*). The JSPP will function if the application is served using another technology, but the features described in The JavaServer Page 23 will not be available.
4. You must have created a development environment for the application as described in the document **Caplin Trader 2.0: Installation Guide**.

In the remainder of this document, it is assumed that you used the following parameter values to create your application development environment.

Application directory: *Novobank*

Application name: *Novotrader*

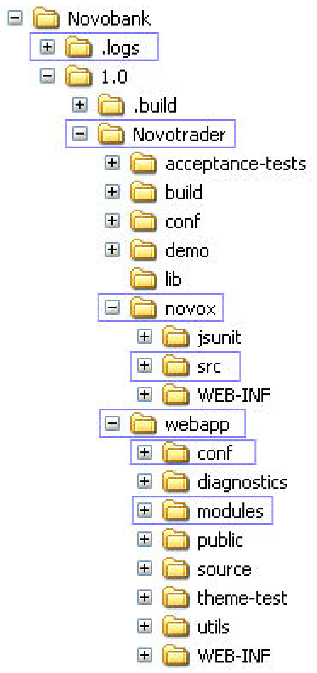
Application version: *1.0*

Application namespace: *novox*

When you read this document, substitute these parameter values with the values you actually used.

3.1 About the Caplin Trader development environment

When you create a development environment for your application, the directory structure will look something like this.



```
Novobank
  .logs
  1.0
    .build
    Novotrader
      acceptance-tests
      build
      conf
      demo
      lib
      novox
        jsunit
        src
        WEB-INF
      webapp
        conf
        diagnostics
        modules
        public
        source
        theme-test
        utils
        WEB-INF
```

The following table lists the top level directories in this structure (as highlighted above), with a brief description of what each directory contains. The information will help you to locate files when you configure the JSPP.

Directory	Content
.logs	The top level directory containing the application server log files.
Novotrader	The top level directory containing the development environment for the *Novotrader* application.
novox	The top level directory containing application specific (*novox* namespace) files and directories.
novox/src	Contains the configuration file *novox-resources.xml*. This file is created by the Caplin Trader build system when the *Novotrader* development environment is first created. It contains information about the application specific JavaScript classes that the JSPP must include when it builds the JavaScript dependency tree for *Novotrader*. You can change the content of this file, or create other application resource files, when you develop *Novotrader*.

Directory	**Content**
novox/src/novox	Contains the development copy of the JavaScript source code modules (*novox* namespace) for the *Novotrader* application. **This is where all your JavaScript development work is done.**
webapp	The Caplin Trader build system builds the *Novotrader* application to this directory. The directory contains the file *application.jsp*, which is the JavaServer Page entry point to the *Novotrader* application. This file contains a markup tag that causes the browser to request a resource served by the JSPP servlet. Without this tag, the JSPP would not run. You can change the content of this file when you develop *Novotrader*.
webapp/conf	Contains configuration files for the *Novotrader* application. The file *jspp.xml* contains XML that configures the JSPP. The XML specifies the location of the *resources.xml* files that the JSPP parses when it builds the JavaScript dependency tree. The XML also maps namespaces to paths. You can change the content of *jspp.xml* when you develop *Novotrader*.
webapp/modules	Contains the Caplin runtime framework modules (*caplin* namespace) that are required by the *Novotrader* application. The Caplin build system also copies application specific (*novox* namespace) modules to this directory. The directory also contains several configuration files with *resources.xml* in the filename. These files contain information about the runtime framework JavaScript classes that the JSPP must include when it builds the JavaScript dependency tree. **You *must not* modify or delete any of the resource files in this directory.**
webapp/WEB-INF	Contains the deployment descriptor file *web.xml*. A setting in this file determines the URL pattern that activates the JSPP servlet.

4 How the JavaScript Preprocessor works

The way that the JSPP selects and concatenates the JavaScript files that it returns to the browser can be considered in three stages:

1. The browser request 8
2. Building the dependency tree 10
3. Returning concatenated files 13

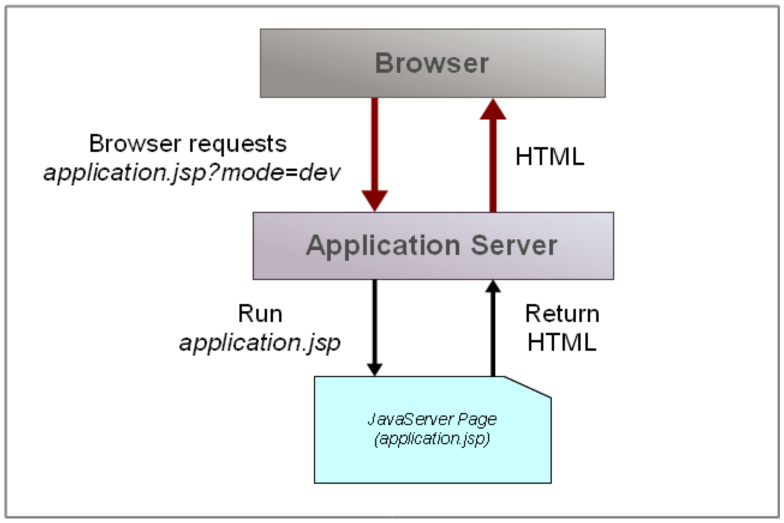
4.1 The browser request

- When the browser requests the *Novotrader* application from the application server, the request URL will look something like the following:

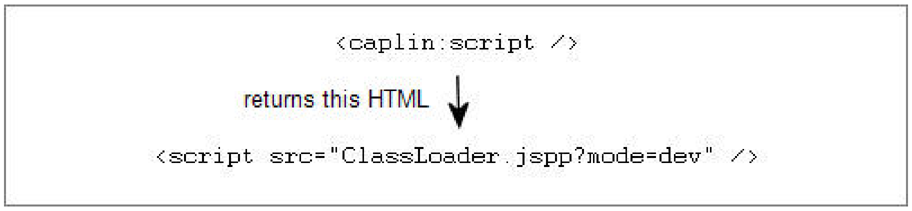
  ```
  http://trader.caplin.com:9090/1.0/Novotrader/webapp/application.jsp
  ?mode=dev
  ```

  The query string `?mode=dev` is optional, but can be used to specify a mode to the JSPP (see Building the dependency tree 10).

- In response to this request, the JavaServer Page (*application.jsp*) returns the HTML for the *Novotrader* application to the browser.



If a `<caplin:script />` tag is present in *application.jsp*, the returned HTML contains a `<script>` tag with a `src` attribute of the form `src="ClassLoader.jspp?mode=dev"`.

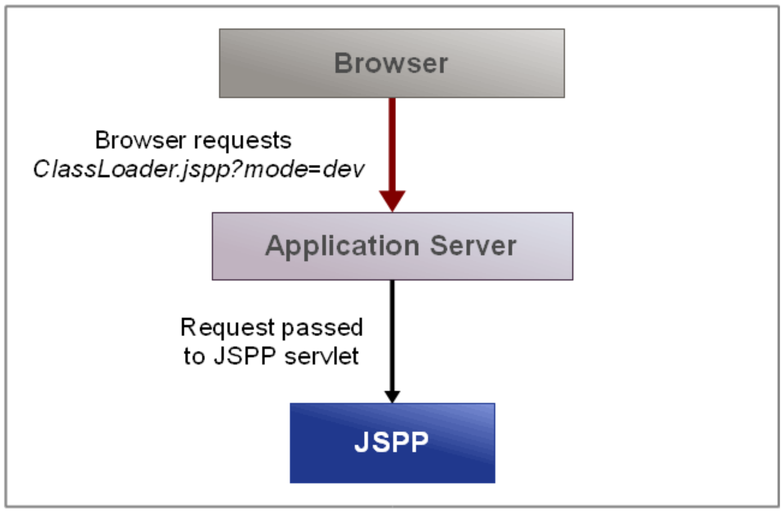


```
<caplin:script />
```

```
<script src="ClassLoader.jspp?mode=dev" />
```

- In response to this `<script>` tag being returned to the browser, the browser requests the resource `ClassLoader.jspp?mode=dev` from the application server.

- Because the requested resource matches the pattern `*.jspp`, the application server passes the request to the JSPP servlet.



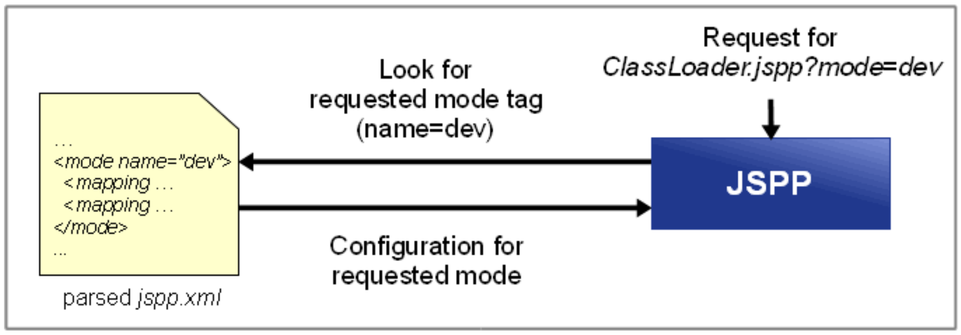
The URL pattern that causes the request to be passed to the JSPP is defined in the deployment descriptor file for the *Novotrader* application, *Novotrader\webapp\WEB-INF\web.xml*.

```
<!- Servlet mappings ->
  <servlet-mapping>
    <servlet-name>JavaScriptPreprocessorServlet</servlet-name>
    <url-pattern>*.jspp</url-pattern>
  </servlet-mapping>
```

4.2 Building the dependency tree

- When the JSPP servlet receives the request for `ClassLoader.jspp`, it parses the configuration file *Novotrader\webapp\conf\jspp.xml* (see JSPP servlet configuration 15).

  The JSPP gets the configuration for the requested mode by looking in the parsed configuration file for a `<mode>` tag with a `name` attribute that matches the mode in the request URL (in this case `?mode=dev`).


...
<mode name="dev">
<mapping ...
<mapping ...
</mode>
...
parsed jspp.xml
Look for requested mode tag (name=dev)
Request for ClassLoader.jspp?mode=dev
JSPP
Configuration for requested mode

  If this tag is not found (`<mode name="dev">`), or if a mode was not specified in the request URL, the JSPP uses the configuration specified by the default `<mode>` to build the dependency tree.

  jspp.xml

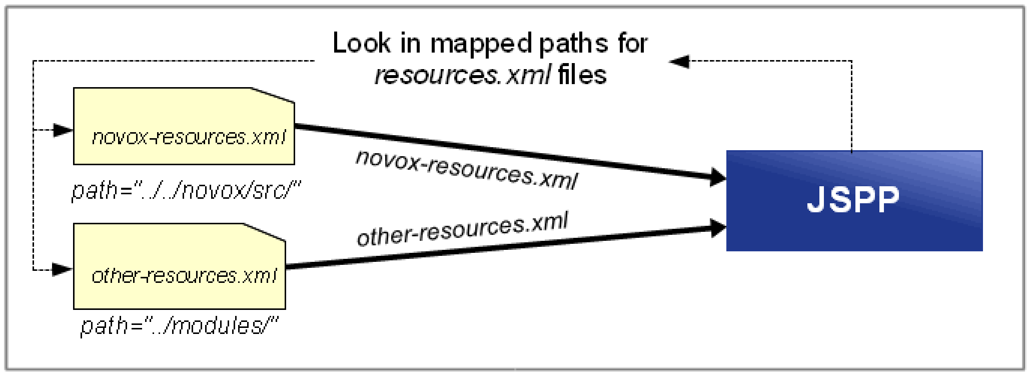
```
<mappings default="war">
  ...
  <mode name="war">
    ...
```

  Each `<mode>` tag has one or more child `<mapping>` tags that map a resource `namespace` to a resource `path` for that mode.

  Mappings in jspp.xml for <mode name="dev">

```
<mode name="dev">
  <mapping namespace="novox.*" path="../../novox/src"/>
  <mapping namespace="caplin.*" path="../modules"/>
</mode>
```

- For each child `<mapping>` tag, the JSPP looks in the mapped path for files with names that start with the mapped namespace (hyphen separated) and that end in *-resources.xml* (such as *novox-resources.xml*).



A *resources.xml* file lists the JavaScript classes required by a module, classes that the JSPP must concatenate and return to the browser.

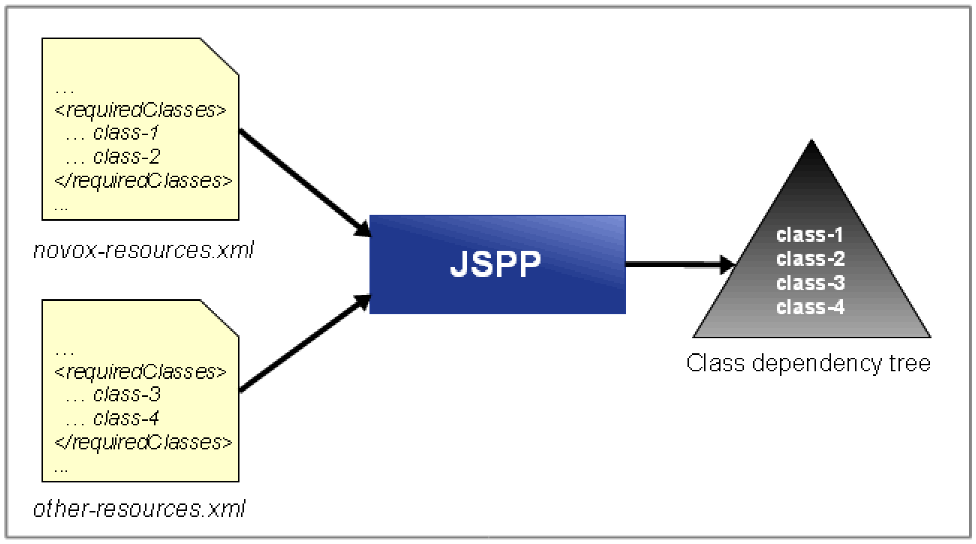
Example resources.xml file

```
<resources>
  <requiredClasses>
    ...
    <className>novox.renderer.DateElementRenderer</className>
    <className>novox.renderer.YieldElementRenderer</className>

    ...

  </requiredClasses>
</resources>
```

- For each *resources.xml* file that it finds, the JSPP adds the JavaScript classes required by that module to an internal class dependency tree.



- For each *resources.xml* file that it finds, the JSPP also looks in the path specified by the `<configurationPath>` tag for other XML configuration files (such as *gridDefinitions.xml*) that have tags with a `className` attribute.

Example resources.xml file (showing the <configurationPath> tag)

```
<resources>
  <requiredClasses>
    <configurationPath></configurationPath>
    ...

  </requiredClasses>
</resources>
```

Example gridDefinitions.xml file with className attributes

```
<decoratorMappings>
  <decoratorMapping id="dragDecorator"
                    className="caplin.grid.decorator.DragDecorator"/>
  <decoratorMapping id="dropDecorator"
                    className="caplin.grid.decorator.DropDecorator"/>
  ...

</decoratorMappings>
```

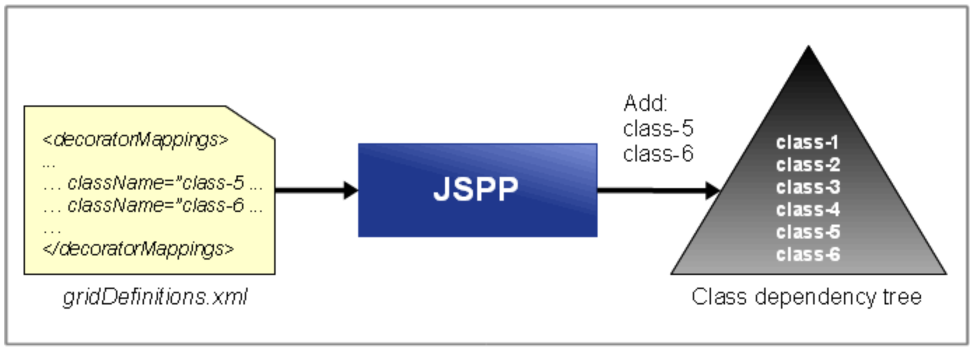
The configuration path is relative to the root path as defined in the deployment descriptor file for the *Novotrader* application, *Novotrader\webapp\WEB-INF\web.xml*.

Extract from deployment descriptor file

```
  <init-param>
    <param-name>root.path</param-name>
    <param-value>conf</param-value>
  </init-param>
```

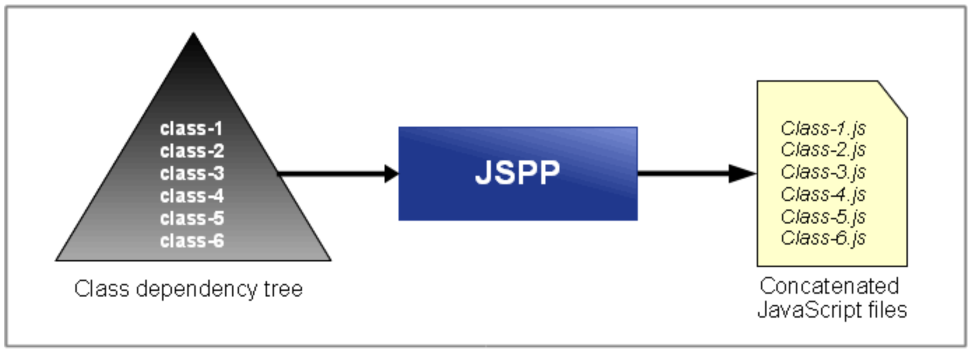
In this case, because the `<configurationPath>` tag is empty, the root path and the configuration path are the same (*conf*).

- For each `className` attribute that it finds, the JSPP adds the JavaScript class specified by that attribute to its internal class dependency tree.



4.3 Returning concatenated files

- For each JavaScript class in the internal class dependency tree, the JSPP adds the JavaScript file for that class, namespace, and mode to the list of files it concatenates.



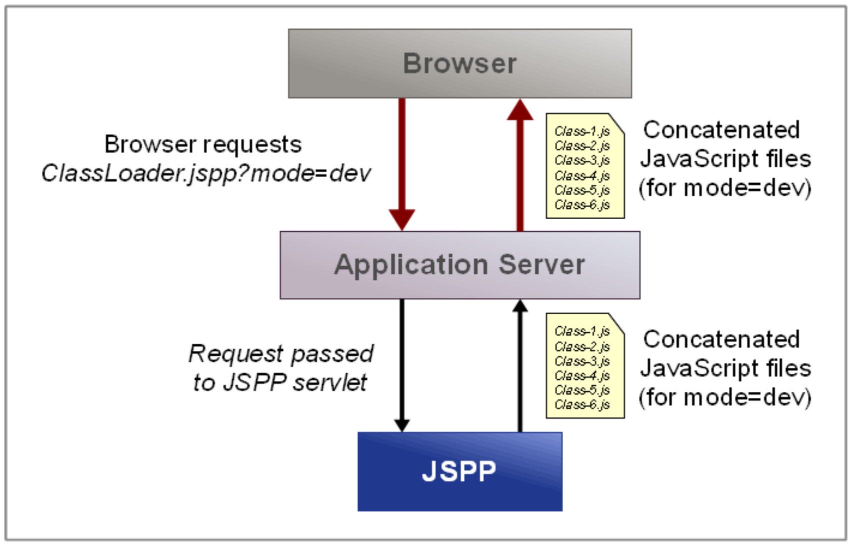
The JSPP locates the file for a required JavaScript class using the mapped `namespace` and `path` of the `<mapping>` tag in the *jspp.xml* configuration file.

Mappings in jspp.xml for <mode name="dev">

```
<mode name="dev">
  <mapping namespace="novox.*" path="../../novox/src"/>
  <mapping namespace="caplin.*" path="../modules"/>
</mode>
```

If a concatenated JavaScript file includes other JavaScript classes (using the `caplin.include` directive), the files for these Javascript classes are also concatenated.

- When all JavaScript classes in the dependency tree have been processed, the JSPP returns the concatenated JavaScript class files to the browser.



In this way, multiple JavaScript class files are returned to the browser in response to a single resource request (the request for `ClassLoader.jspp`).

By changing the mode specified in the application URL (in this case `?mode=dev`), you can request a different version of the *Novotrader* application (such as the built version rather than the development version). If you do not specify a mode, JavaScript files for the default mode are returned.

By changing the classes listed in the resource configuration file for the *Novotrader* application, *Novotrader\novox\src\novox-resources.xml*, you can control the application specific JavaScript classes that the JSPP concatenates and returns to the browser.

5 Example JavaScript Preprocessor configuration

The JSPP is configured by settings in a number of XML configuration files, and activated if the application server receives a request that matches a particular URL pattern.

5.1 JSPP servlet configuration

The JSPP servlet is configured by settings in the deployment descriptor file, *web.xml*. The format of the XML configuration is an industry standard (see the Web Application Deployment Descriptor schema on the Sun Developer Network website at http://java.sun.com/xml/ns/javaee/index.html).

In the following example, the application server is configured to pass resource requests that end in `.jspp` to the JavaScript Preprocessor.

URL pattern mapping

```
<servlet-mapping>
       <servlet-name>JavaScriptPreprocessorServlet</servlet-name>
       <url-pattern>*.jspp</url-pattern>
</servlet-mapping>
```

Note: Do not set this URL pattern to anything else. The JavaServer Page returns a `<script>` tag with a `src` attribute of the form `src="ClassLoader.jspp?mode=dev"` to the browser if the `<caplin:script />` tag is present in the JavaServer Page. If the configured URL pattern is not `*.jspp`, the JavaScript Preprocessor will not receive the request for the resource `ClassLoader.jspp` (see The browser request 8 for further information).

Other characteristics of the JSPP servlet are configured by name/value pair parameters in the deployment descriptor file (see Configuration Reference: JSPP Servlet 24).

JSPP servlet configuration

```
servlet>
  <servlet-name>JavaScriptPreprocessorServlet</servlet-name>
  <servlet-class>com.caplin.javascript.preprocessor.web.JavaScriptPreprocessorServlet
  </servlet-class>
  <init-param>
    <param-name>root.path</param-name>
    <param-value>conf</param-value>
  </init-param>
  <init-param>
    <param-name>jspp.config</param-name>
    <param-value>jspp.xml</param-value>
  </init-param>
  <init-param>
    <param-name>
    <param-name>jspp.xsd</param-name>
    <param-value>schema/JavaScriptPreprocessor.xsd</param-value>
  </init-param>
  <init-param>resources.xsd</param-name>
    <param-value>schema/Resources.xsd</param-value>
  </init-param>
  <init-param>
    <param-name>cache.expiry</param-name>
    <param-value>0</param-value>
  </init-param>
  <init-param>
    <param-name>log.level</param-name>
    <param-value>FINE</param-value>
  </init-param>
</servlet>
```

An explanation of the example servlet configuration

Here is an explanation of the JSPP servlet configuration used in this example.

- **`root.path`** The root path, relative to the servlet context, of all other JSPP configuration paths.

```
<param-name>root.path</param-name>
<param-value>conf</param-value>
```

The servlet context is the directory containing the JavaServer Page that serves the application. In this case the root path is set to `conf`, which specifies that all configuration file paths are relative to the *conf* directory.

In the case of the *Novotrader* development environment, the *conf* directory is inside the *webapp* directory (see About the Caplin Trader framework 6).

- **`jspp.config`** The path to the configuration file that contains namespace to path mappings for each JSPP mode (see Namespace configuration 18).

```
<param-name>jspp.config</param-name>
<param-value>jspp.xml</param-value>
```

In this case the file path is set to `jspp.xml`. Because the path is relative to the root path, the JSPP will expect to find the configuration file *jspp.xml* inside the *conf* directory.

- **`jspp.xsd`** The path to the XML schema for the configuration file specified by **jspp.config**.

```
<param-name>jspp.xsd</param-name>
<param-value>schema/JavaScriptPreprocessor.xsd</param-value>
```

The JSPP validates the namespace to path configuration file against this schema. If the validation fails, the JSPP will not start and an error is logged at the application server.

In this case, the file path is set to `schema/JavaScriptPreprocessor.xsd`.
Because the path is relative to the root path, the JSPP will expect to find the configuration file *schema/JavaScriptPreprocessor.xsd* inside the *conf* directory.

- **`resources.xsd`** The path to the XML schema for all resource configuration files (see Resource configuration 20).

```
<param-name>resources.xsd</param-name>
<param-value>schema/Resources.xsd</param-value>
```

The JSPP validates all resource configuration files against this schema. If the validation fails, an error is logged at the application server.

In this case the file path is set to `schema/Resources.xsd`. Because the path is relative to the root path, the JSPP will expect to find the configuration file *schema/Resources.xsd* inside the *conf* directory.

- **`cache.expiry`** The cache expiry time at the server for all JSPP configuration files and concatenated JavaScript files.

```
<param-name>cache.expiry</param-name>
<param-value>0</param-value>
```

In this case the cache expiry time is set to `0`, which means that JSPP configuration files and concatenated JavaScript files are not cached at the server. The cache expiry time must be set to `Infinity` when the application is deployed for production.

- **`log.level`** The Java log level for all JSPP error messages that are logged at the application server.

```
<param-name>log.level</param-name>
<param-value>FINE</param-value>
```

In this case the log level is set to `FINE`, which means that tracing information is logged.

5.2 Namespace configuration

When the JSPP servlet runs on the application server, it parses the XML configuration file pointed to by the **jspp.config** configuration parameter of the deployment descriptor file (see JSPP servlet configuration 15). This configuration file contains namespace to path mappings for one or more application modes.

The example below shows a typical namespace to path mappings XML configuration file.

XML for namespace to path mappings

```
<mappings default="war">
  <mode name="dev">
    <mapping namespace="novox.*" path="../../novox/src"/>
    <mapping namespace="caplin.*" path="../modules"/>
  </mode>
  <mode name="war">
    <errorMessage>Sorry, there is a technical fault and the
                  application cannot be loaded.</errorMessage>
    <mapping namespace="*" path="../modules"/>
  </mode>
</mappings>
```

An explanation of the example XML configuration

Here is an explanation of what the example XML configuration contains, and how the JSPP uses this information to return different JavaScript files according to the mode specified in the request URL.

- **`<mappings>`** contains configuration settings for two modes (`name="dev"` and `name="war"`).

```
<mappings default="war">
  <mode name="dev">
    ...

  <mode name="war">
    ...
</mappings>
```

If the URL in the application request contains the query string `?mode=dev`, the JSPP will use the `<mode name="dev">` configuration to concatenate and return JavaScript files.

If the URL in the application request contains the query string `?mode=war`, the JSPP will use the `<mode name="war">` configuration to concatenate and return JavaScript files.

If a mode is not present in the application request (typical when deployed in production), or if the requested mode or mapped paths do not exist, the JSPP will use the default configuration (`default="war"`) to concatenate and return JavaScript files.

- **`<mode>`** contains configuration settings for a single mode.

```
  <mode name="dev">
    <mapping ...

  <mode name="war">
    <mapping ...
```

In this case, namespace to path mappings are defined for two modes (`name="dev"` and `name="war"`).

- **`<mapping>`** maps a namespace to a path for a particular mode.

```
<mode name="dev">
    <mapping namespace="novox.*" path="../../novox/src"/>
    <mapping namespace="caplin.*" path="../modules"/>
  </mode>
  <mode name="war">
    <errorMessage> ... </errorMessage>
    <mapping namespace="*" path="../modules"/>
  </mode>
```

The `namespace` attribute defines the namespace, and the `path` attribute the path for that namespace. The JSPP uses this information in two ways:

1. When the JSPP builds the JavaScript class dependency tree, it looks in the top level directory of each mapped path for files with names that start with the mapped namespace (hyphen separated) and end in *-resources.xml* (such as *novox-resources.xml*). Each resource file lists the JavaScript classes required by a module, and the JSPP adds these classes to an internal class dependency tree (see Resource configuration 20).
2. When the JSPP concatenates the files for each class in the class dependency tree, it locates the files using the requested (or default) mode, and the namespace to path mapping for that mode.

In this case, the *caplin* namespace is mapped to the same path for each mode (`path="../modules"`), but the *novox* namespace is mapped to different paths for each mode. This means that the files returned by the JSPP in the *novox* namespace depend on the mode specified in the request URL, but the files returned by the JSPP in the *caplin* namespace are independent of the specified mode.

- **`<errorMessage>`** contains a message that is displayed in the client browser if the JSPP logs an error on the server.

```
<errorMessage>Sorry, there is a technical fault and the
              application cannot be loaded.</errorMessage>
```

If the `<errorMessage>` tag is not present in the configuration, a detailed error message is displayed in the client browser.

5.3 Resource configuration

A resource configuration file specifies the JavaScript classes required by a module. There are several modules in the *caplin* namespace, and each module has its own resource configuration file (such as *caplin-grid-resources.xml*).

Note: You must not modify or delete any resource configuration file in the *caplin* namespace.

If you created a development environment for the ***Novotrader*** application, you will find a resource configuration file for the *novox* module. This file is called *novox-resources.xml*, and is located in the *novox/source* directory of the *Novotrader* environment (see About the Caplin Trader framework 6). You can change the content of this file if you develop the *Novotrader* application.

Each resource configuration file must have a filename that is prefixed by the hyphen-separated namespace of the module, and which ends in *-resources.xml* (such as *novox-resources.xml* or *caplin-grid-resources.xml*).

The path to resource configuration files is defined by the `<mapping>` tag in the namespace to path configuration file (see Namespace configuration 18).

The following example is an extract from a typical resource configuration file for the *Novotrader* application. The example is an extract because the *Novotrader* resource file has several lines, but you only need to see some of these lines to understand the example.

XML extract from a resource configuration file

```
<resources>
  <requiredClasses>
    <configurationPath></configurationPath>

    <!-- caplin namespace classes -->

    <className>caplin.grid.GridColumn</className>
    <className>caplin.grid.RttpContainerGridDataProvider</className>

   <!-- novox namspace classes -->

    <className>novox.grid.decorator.RightClickMenuDecorator</className>

  </requiredClasses>
</resources>
```

An explanation of the example XML configuration

Here is an explanation of what the example XML configuration contains, and how the JSPP uses this information to select the files it concatenates.

- **`<requiredClasses>`** contains the classes required by the module.

```
<requiredClasses>
  ...

    <className>caplin.grid.GridColumn</className>
    <className>caplin.grid.RttpContainerGridDataProvider</className>

    <className>novox.grid.decorator.RightClickMenuDecorator</className>

  ...
</requiredClasses>
```

  In this case three classes are specified, two in the *caplin* namespace and one in the *novox* namespace.

- **`<className>`** contains the fully qualified name of a JavaScript class.

```
    <className>caplin.grid.GridColumn</className>
    <className>caplin.grid.RttpContainerGridDataProvider</className>

    <className>novox.grid.decorator.RightClickMenuDecorator</className>
```

  The JSPP adds each class defined by a `<className>` tag to an internal class dependency tree. When the JSPP concatenates JavaScript files, it is files for the classes in this dependency tree that it concatenates.

  The JSPP locates the JavaScript file for a required class using the requested (or default) mode, in conjunction with other configuration settings, as shown below.

  Namespace mapping for the requested mode (`?mode=war`, see Namespace configuration 18):

  `<mapping namespace="novox.*" path="../modules"/>`

  Root path (see JSPP servlet configuration 15):

  webapp/conf (assuming the servlet context is *webapp* and the **root.path** is set to *conf*)

  Path to the *modules* directory (relative to the root path):

  webapp/modules (because `../` is in the `path` definition of the `namespace` mapping)

  Required class:

  `novox.grid.decorator.RightClickMenuDecorator`

Relative path to the concatenated file (by convention, the path must mirror the package name):

novox/grid/decorator

Location of the concatenated file:

webapp/modules/novox/grid/decorator

Name of the concatenated file:

RightClickMenuDecorator.js

- **`<configurationPath>`** contains the path to other XML configuration files that may or may not specify required JavaScript classes.

```
<configurationPath></configurationPath>
```

The JSPP looks in the path specified by the `<configurationPath>` tag for other XML configuration files (such as *gridDefinitions.xml*) that have tags with a `className` attribute. For each `className` attribute found, the JSPP adds the JavaScript class specified by that attribute to its internal class dependency tree.

The configuration path is relative to the root path as defined in the deployment descriptor file (see JSPP servlet configuration 15). In this case, because the `<configurationPath>` tag is empty, the configuration path is set to the root path.

5.4 The JavaServer Page

The application server can be configured to pass requests that match a particular URL pattern to the JavaScript Preprocessor (see JSPP servlet configuration 15).

For example, the application server can be configured to pass all requests that match the pattern `*.jspp` to the JavaScript Preprocessor. To achieve this, add a `<caplin:script />` tag to the application JavaServer Page.

When the JavaServer Page returns HTML to the browser, it replaces the `<caplin:script />` tag with a tag of the form `<script src="ClassLoader.jspp?mode=xxx">`, where `xxx` is replaced by the mode present in the URL that requested the application.

A typical application request URL is:

```
http://trader.caplin.com:9090/1.0/Novotrader/webapp/application.jsp?
mode=war
```

In response to this request, the JavaServer Page returns the tag `<script src="ClassLoader.jspp?mode=war">` to the browser, which in turn requests the resource `ClassLoader.jspp?mode=war`. Because the request matches the configured URL pattern (`*.jspp`), the request is passed to the JSPP servlet.

The following is an extract from a JavaServer Page that contains a `<caplin:script />` tag.

Extract from a JavaServer Page

```
<%@ taglib uri="http://www.caplin.com/CaplinTrader/JavaScriptPreprocessor"
    prefix="caplin" %>
...

<html>
...
<caplin:script />
...
</html>
```

The `<%@ taglib>` tag is a directive, and declares that the JavaServer Page uses tags defined in a custom tag library. The directive must be placed before the `<caplin:script />` tag in the JavaServer Page, and the `<caplin:script />` tag must be a child of an `<html>` tag.

6 Configuration Reference: JSPP Servlet

This is the configuration reference for the JSPP servlet (see JSPP servlet configuration 15).

JSPP configuration parameters

Parameter	Required?	Default	Description
root.path	Yes		The path, relative to the servlet context, that all other JSPP configuration paths are relative to (see An explanation of the example servlet configuration 16).
jspp.config	Yes		The path to the configuration file that contains namespace to path mappings for each JSPP mode (see Namespace configuration 18).
jspp.xsd	Yes		The path to the XML schema for **jspp.config**. The JSPP validates the configuration file that **jspp.config** points to against this schema. If the validation fails, the JSPP will not start and an error is logged at the application server.
resources.xsd	Yes		The path to the XML schema for resource configuration files (see Resource configuration 20). The JSPP validates all resource configuration files against this schema. If the validation fails, an error is logged at the application server.
cache.expiry	No	Infinity	The server cache expiry time for all JSPP configuration files and concatenated JavaScript files. Valid values: *0* files are not cached (recommended application development setting). *N* cache expiry time in seconds (a positive integer). *Infinity* files are cached until the application server is re-started (this setting must be used when the application is deployed for production).
log.level	No	ALL	The Java log level of the JSPP. The JSPP logs error messages to this level at the application server. Valid values: SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST, and ALL. Application server log files are written to the *.logs/tomcat* directory of the installed development environment (see About the Caplin Trader development environment 6).

For a description of the available log levels, refer to the `java.util.logging` class in the API specification for the Java Platform, Standard Edition v1.4.2, available at http://download.oracle.com/javase/1.4.2/docs/api/java/util/logging/Level.html.

7 Configuration Reference: Namespace Mappings

This is the reference information for the configuration XML that maps JavaScript class namespaces to the relative path locations of these namespaces.

7.1 Ordering and nesting of tags

Each top level tag of the namespace mapping configuration XML is shown below, together with the child tags that it can typically contain (the children are in no particular order).

Tip: Advanced users may wish to consult the XML Schema (*JavaScriptPreprocessor.xsd*) for definitive information on the ordering and nesting of tags.

For a description of each tag and its attributes, see the Namespace Mappings XML Reference 26 section.

<mappings>

This is the outermost tag

```
<mappings>
   <mode></mode> (one or more)
</mappings>
```

<mode>

```
<mode>
   <errorMessage /> (zero or one)
   <mapping /> (one or more)
</mode>
```

<errorMessage>

```
<errorMessage /> (no children)
```

<mapping>

```
<mapping /> (no children)
```

7.2 Namespace Mappings XML Reference

This section describes the XML tags that you can use to configure namespace mappings for the JSPP.

Default attribute values

In the tables that follow, if an attribute is not required (Req? = 'N') and there is a default value specified, then not supplying the attribute is equivalent to setting the attribute to this default value. If an attribute is not required and the default is '(none)', then not supplying the attribute can result in one of two behaviors, depending on the particular attribute – either the behavior is as specified in the description column of the table, or there is no effect on the appearance or behavior of the component.

<errorMessage>

`<errorMessage>`

A message that is displayed in the client browser when the JSPP logs an error on the server. If <errorMessage> is not defined, the detailed log message that is logged on the server is also sent to the browser. The message only applies to the mode in which it is defined, and is normally used to prevent detailed log messages being displayed when the application is deployed for production.

Attributes: This tag has no attributes.

<mapping>

`<mapping>`

Maps a JavaScript class namespace to a path. When the JSPP is applying the configuration for the parent mode, it looks in the top level directory of the mapped path for files with names that start with the mapped namespace (hyphen separated) and that end in -resources.xml, and adds the JavaScript classes specified in these resource files to an internal class dependency tree. When the JSPP concatenates files in the dependency tree for the mapped namespace, it locates the files using the mapped path.

Attributes:

Name	Type	Default	Req?	Description
`namespace`	string	(none)	Y	The namespace to map. The * character is a wildcard character that matches any text if prefixed by '.' and placed at the end of the namespace (as in novox.*), or if it appears on its own (as in * where it matches any namespace). The * character is not treated as a wildcard character if it is placed anywhere else in the namespace.
`path`	string	(none)	Y	The path for the mapped namespace, relative to the root path (see the root.path parameter of the deployment descriptor file, as described in the 'Configuration Reference: JSPP Servlet' section of this document).

<mappings>

`<mappings>`

The outermost tag of the namespace mappings XML configuration, containing one or more <mode> tags.

Attributes:

Name	Type	Default	Req?	Description
`default`	string	(none)	Y	Identifies the name of the <mode> tag that contains the default namespace to path mappings configuration for the JSPP. The JSPP uses the default configuration if a configuration for the requested mode does not exist, or if a mode is not specified in the request URL. A <mode> tag with the default name must exist in the namespace mappings XML configuration.

<mode>

`<mode>`

Contains the namespace to path mappings for a named mode.

Attributes:

Name	Type	Default	Req?	Description
`name`	string	(none)	Y	The name of the mode, which must be unique across all defined modes.

8 Configuration Reference: Module Resources

This is the reference information for the configuration XML that specifies the JavaScript resources required by a module.

8.1 Ordering and nesting of tags

Each top level tag of the module resources configuration XML is shown below, together with the child tags that it can typically contain (the children are in no particular order).

Tip: Advanced users may wish to consult the XML Schema (*resources.xsd*) for definitive information on the ordering and nesting of tags.

For a description of each tag and its attributes, see the Module Resources XML Reference 29 section.

<resources>

This is the outermost tag

```
<resources>
   <requiredClasses></requiredClasses> (one only)
</resources>
```

<requiredClasses>

```
<requiredClasses> (the child tags can be in any order)
   <className /> (zero or more)
   <configurationPath /> (zero or more)
</requiredClasses>
```

<className>

```
<className /> (no children)
```

<configurationPath>

```
<configurationPath /> (no children)
```

8.2 Module Resources XML Reference

This section describes the XML tags that you use to specify the JavaScript resources required by a module.

<className>

`<className>`

The fully qualified name of a required JavaScript class (such as novox.grid.decorator.RightClickMenuDecorator). The JSPP adds required JavaScript classes to its internal class dependency tree (see <requiredClasses>).

Attributes: This tag has no attributes.

<configurationPath>

`<configurationPath>`

The path to a directory containing other XML files that configure the application (such as gridConfiguration.xml). The JSPP looks in these configuration files for tags that have className attributes, and adds the JavaScript classes specified by these attributes to its internal class dependency tree (see <requiredClasses>). The path is relaive to the root path as defined in the deployment descriptor file (see the root.path parameter in the 'Configuration Reference: JSPP Servlet' section of this document. If this tag is empty, the configuration path is the same as the root path.

Attributes: This tag has no attributes.

<requiredClasses>

`<requiredClasses>`

Specifies the classes required by a module (see <className> and <configurationPath>). The JSPP adds required classes to its internal class dependency tree. It is the files for classes in this dependency tree that the JSPP concatenates and returns to the browser.

Attributes: This tag has no attributes.

<resources>

`<resources>`

The outermost tag of the module resources configuration XML.

Attributes: This tag has no attributes.

9 Configuration Reference: The JavaServer Page

This is the configuration reference for the custom tags that can be inserted in the JavaServer Page that serves the Caplin Trader application (see The JavaServer Page 23).

9.1 <%@ taglib %>

`<%@ taglib %>`

A directive that declares the JavaServer Page uses tags defined in a custom tag library. The directive must be placed before any custom tag from that library in the JavaServer Page.

Attributes:

Name	Type	Default	Req?	Description
`uri`	string	(none)	Y	The URI that uniquely identifies the custom tag library. For the <caplin:script /> tag, this must be set to "http://www.caplin.com/CaplinTrader/JavaScriptPreprocessor".
`prefix`	string	(none)	Y	The prefix that distinguishes tags provided by a given tag library from tags provided by other tag libraries. For the <caplin:script /> tag, this must be set to "caplin".

9.2 <caplin:script />

`<caplin:script />`

A custom tag that can be placed in the JavaServer Page that serves the Caplin Trader application. When the JavaServer Page returns HTML to the browser, it replaces the `<caplin:script />` tag with a tag of the form `<script src="ClassLoader.jspp?mode=xxx">`, where `xxx` is replaced by the mode present in the URL that requested the application.

A typical application request URL is:

```
http://trader.caplin.com:9090/1.0/Novotrader/webapp/application.jsp?
mode=war
```

In response to this request, the JavaServer Page returns the following tag to the browser:

```
<script src="ClassLoader.jspp?mode=war">
```

If a mode is not present in the URL that requested the application, the JavaServer Page returns the following tag to the browser:

```
<script src="ClassLoader.jspp>
```

The `<caplin:script />` tag must be a child of an `<html>` tag in the JavaServer Page.

Attributes: This tag has no attributes.

10 Glossary of terms and acronyms

This section contains a glossary of terms, abbreviations, and acronyms relating to the JavaScript Preprocessor.

Term	Definition
Application server	Software that serves up web pages (typically with dynamically constructed content) and web applications, for rendering or execution by client web browsers. **Caplin Trader applications** are served from a **Java application server**. In this document the term “application server” means “Java application server”.
Caplin Trader	A web application framework for constructing browser-based financial trading applications.
Caplin Trader application	A web application that is built using **Caplin Trader**.
Deployment descriptor	A file (named *web.xml*) containing the XML that configures a Java web application. When the application server receives a request for the application, it uses the configuration in the deployment descriptor file to map the request URL to the code (such as Java servlets and filters) that handles the request.
End-user	A person who uses a piece of software for its intended purpose. For example, financial traders are the end-users of a **Caplin Trader application**.
Java application server	An application server that hosts JavaServer Pages (**JSP**s) and Java servlets. Such servers can also host resources implemented in other technologies, such as HTML, CSS, XML, and JavaScript.
JavaScript Preprocessor	A Java servlet residing on the application server that can be configured to concatenate and return only those JavaScript files required by the requested **Caplin Trader application**.
JavaServer Page	A Java technology for serving dynamically generated web pages.
JSP	See JavaServer Page.
JSPP	See JavaScript Preprocessor
Mode	A query string that may or may not be present in the URL that requests the **Caplin Trader application**. The **JSPP** can be configured to concatenate and return JavaScript files from different locations (such as the development or built version), depending on the requested mode. If a mode query string is not present in the request URL, the JSPP returns JavaScript files from a location specified by the default mode.
Module	A number of JavaScript classes that are related by the functionality they provide.
Production version	The version of a **Caplin Trader application** that is deployed to serve **end-users**, as opposed to the development or test version.

Single-dealer platforms for the capital markets

CAPLIN

Contact Us

Caplin Systems Ltd

Cutlers Court

115 Houndsditch

London EC3A 7BR

Telephone: +44 20 7826 9600

Fax: +44 20 7826 9610

www.caplin.com