

Application Designer

Configuration

Version 8.3.4

July 2014

This document applies to Application Designer Version 8.3.4.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Copyright © 2005-2014 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors..

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA, Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at <http://documentation.softwareag.com/legal/>.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at <http://documentation.softwareag.com/legal/> and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices and license terms, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". This document is part of the product documentation, located at <http://documentation.softwareag.com/legal/> and/or in the root installation directory of the licensed product(s).

Document ID: CIT-CONFIG-834-20140722

Table of Contents

Preface	v
1 Browser Configuration	1
Supported Browsers	2
JavaScript Enabling	2
Browser Caching	4
2 Background Information	7
Application Designer Web Application	8
Creating a Second Application Designer Web Application inside Your Tomcat Installation	9
Adding Application Designer to an Existing Web Application	10
Building a Web Application Archive	10
3 Design Time Mode and Runtime Mode	11
When to Use which Mode	12
Setup	12
Class Loader Considerations	13
File Access Considerations	13
4 Advice for Using Application Designer During Development	15
Do not Use the Web Application cis	16
Select the Right Directory Location for Your Web Application	16
Typical Steps for Upgrading Your Web Application	17
Resource Files to be Shared in a Versioning System	20
Integrating Application Designer into Your Build Process	21
5 Configuration of Application Designer	23
Overview of Configuration Files	24
web.xml	24
cisconfig.xml	26
controllibraries.xml - Adding Control Libraries	34
editortemplates.xml	35
editor.xml - Available Controls	35
startapps.xml - Applications to be Started	35
Customizing Configuration Files	35
6 HTTP Data Volume Considerations	39
The Browser's Loading Behavior	40
How does Application Designer Fit in?	41
Configuration Options for Optimized HTTP Usage	42
7 Using ANT for Upgrades	47
ANT File for Take Over	48
Step by Step	51
Running the ANT Script	52

Preface

This documentation describes how to configure Application Designer. It is subdivided into the following topics:

Browser Configuration	How to configure supported browsers.
Background Information	Information that is helpful when transferring Application Designer to a servlet container of your choice.
Design Time Mode and Runtime Mode	Describes the difference between Application Designer running in design time and runtime environments.
Advice for Using Application Designer During Development	Important tips that you should consider before you start using Application Designer in the development process.
Configuration of Application Designer	Information on fine-tuning the configuration of Application Designer.
HTTP Data Volume Considerations	Information on the data volume that is transferred between server and browser client at runtime.
Using ANT for Upgrades	Recommendations for using ANT for upgrading.

1 Browser Configuration

- Supported Browsers 2
- JavaScript Enabling 2
- Browser Caching 4

Supported Browsers

Application Designer runs on browsers supporting DHTML (HTML and JavaScript) to a certain level. The browsers supporting this level are listed below:

Internet Explorer 8, 9, 10 and 11 ⁽¹⁾

Mozilla Firefox Extended Support Release 17 and 24 ⁽²⁾

Safari 5.1 on Windows and Mac OS X

Google Chrome ⁽³⁾

Notes:

⁽¹⁾ Application Designer Version 8.3.4 is the last version that supports Internet Explorer 8.

⁽²⁾ Only the Extended Support Releases of Mozilla Firefox are explicitly supported. Due to frequent upgrades of the Mozilla Firefox consumer release, the compatibility of Application Designer with future versions of Mozilla Firefox cannot be fully guaranteed. Possible incompatibilities will be removed during the regular maintenance process of Application Designer.

⁽³⁾ The Google Chrome support is based on Google Chrome Version 31. Due to frequent version upgrades of Google Chrome, compatibility of Application Designer with future versions of Google Chrome cannot be fully guaranteed. Possible incompatibilities will be removed during the regular maintenance process of Application Designer.

You do not have to install any further plug-ins or additional software for these browsers. For example, Application Designer does not require that any kind of Java Virtual Machine is installed on the client side.

JavaScript Enabling

Application Designer pages are interactive pages: the interactivity is internally implemented by the usage of JavaScript inside the pages. As a consequence, JavaScript has to be enabled.

JavaScript enabling is explained below for the following browsers:

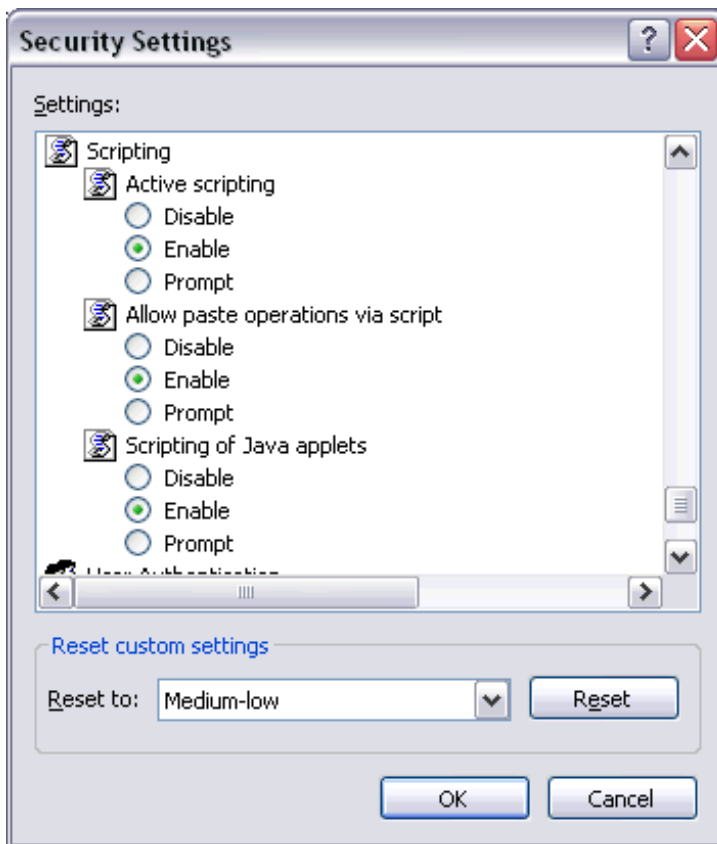
- [Internet Explorer](#)

- Mozilla Firefox

Internet Explorer

In Internet Explorer, you enable JavaScript via **Tools > Internet Options**. On the **Security** tab of the resulting dialog box, you can see that Internet Explorer provides different web content zones.

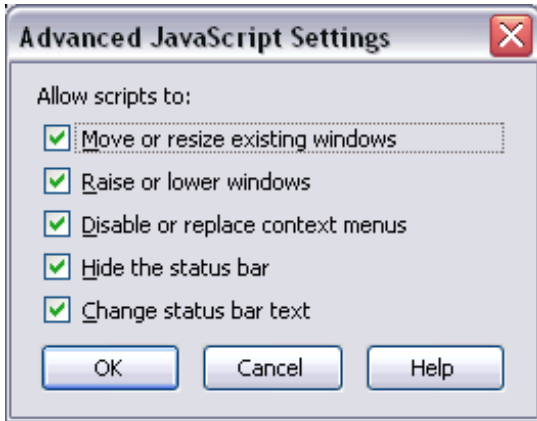
Each zone may have different attributes controlling security-relevant parameters. Make sure that in the zones in which Application Designer pages are available the security settings are set to allow the execution of JavaScript inside a page.



Mozilla Firefox

In Mozilla Firefox, JavaScript is switched on and off on a central level.

Open the **Content** tab of the **Options** dialog box (**Tools > Options**) and make sure that the **Enable JavaScript** option is enabled. When you choose the **Advanced** button next to this option, you can set the following options:



Browser Caching

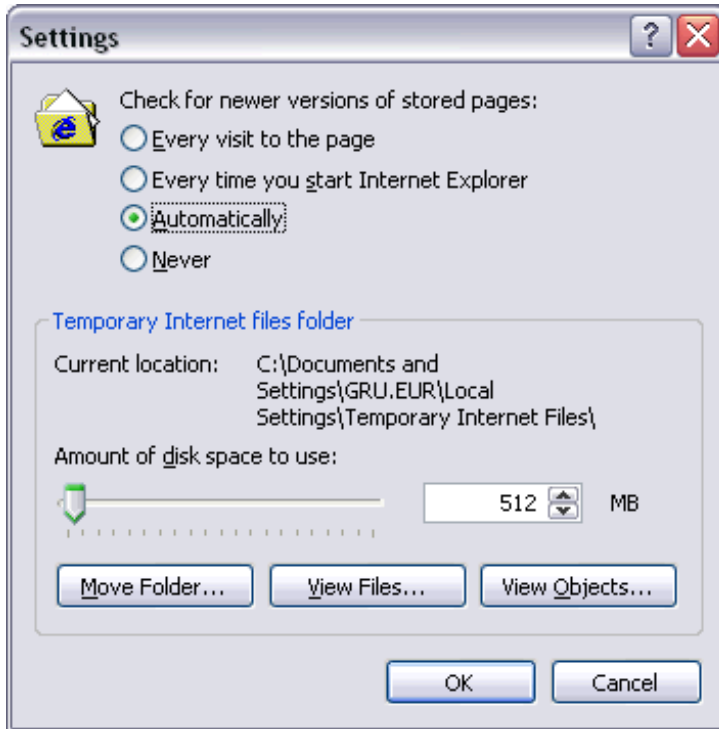
When working with Application Designer pages as a client front-end, make sure to set up the browser caching in such a way that it does not reload a page every time it is accessed by the browser. The reason for this is that Application Designer's HTML pages stay stable in the browser. They do not contain any application data but are more comparable to small programs. The actual application data is filled into the pages dynamically at runtime.

The browser caching setup is explained below for the following browsers:

- Internet Explorer
- Mozilla Firefox

Internet Explorer

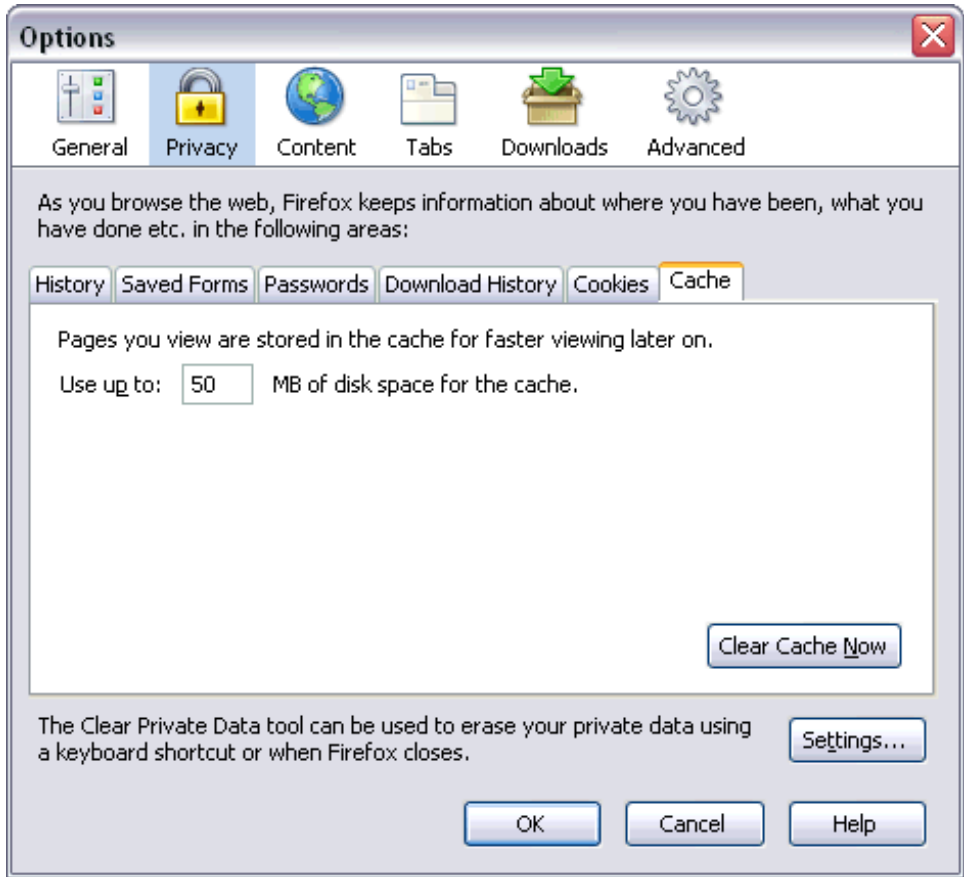
In Internet Explorer, you set up caching via **Tools > Internet Options**. On the **General** tab of the resulting dialog box, choose the **Settings** button in the **Temporary Internet files** group box. The following dialog box appears:



Either select the option **Automatically** or **Every time you start Internet Explorer**.

Mozilla Firefox

With Mozilla Firefox, you do not have to care about the browser's cache strategy. Open the **Options** dialog box (**Tools > Options**) and have a look.



2 Background Information

- Application Designer Web Application 8
- Creating a Second Application Designer Web Application inside Your Tomcat Installation 9
- Adding Application Designer to an Existing Web Application 10
- Building a Web Application Archive 10

Application Designer runs as a web application in any kind of servlet container supporting the servlet API. Information on the required version of the Servlet specification is provided in the section *Hardware and Software Requirements*.

There are multiple servlet containers available on the market. Tomcat is the most commonly used servlet container of the open source world, but there are also others that in most cases are part of Java EE-based application server environments: Websphere from IBM, BEA Weblogic, Sun IPlanet, SAP Web Application Server, JBoss and many others.

This chapter provides background information that is helpful when transferring Application Designer to a servlet container of your choice. This chapter is not designed to be useful when installing Application Designer for the first time - use the standard Tomcat servlet container for doing your first steps.

Application Designer Web Application

Have a look at your installation's `<installdir>/tomcat/webapps` directory. You see the subdirectory `cis`. This is the name of the web application which is used by default.

The `cis` directory contains the following subdirectories:

<code>/cis/</code>	General information.
<code> /config/</code>	Configuration files.
<code> /licensekey/</code>	License key file.
<code> /styles/</code>	Standard location of style sheet files.
<code> /temp/</code>	Temporary files.
<code>/cisdemos/</code>	Application Designer demo project.
<code>/cisdocumentation/</code>	Documentation.
<code>/EclipsePlugin/</code>	Directory for the Eclipse plug-in.
<code>/HTMLBasedGUI/</code>	Application Designer base project.
<code>/META-INF/</code>	Web application standard directory.
<code>/SWTBasedGUI/</code>	Directory for the SWT client.
<code>/WEB-INF/</code>	Web application standard directory.
<code>/web.xml</code>	Web application's configuration file.
<code>/classes/</code>	Web application's classes.
<code>/lib/</code>	Web application's libraries.

Configuration of web.xml

As with any other web application, the *web.xml* file contains configuration information which you have to take care of when deploying. The file contains information about the servlets which are part of the Application Designer web application. There is one servlet `Connector` that contains important configuration parameters: `cis.home` and `cis.log`. See the description of the [web.xml](#) file for further information.

If only working with Application Designer GUIs, you do not have to pay further attention to the *web.xml* file. If working in more complex scenarios in which you might also define other servlets or when you access Enterprise Java Beans, you have to adapt the *web.xml* file to your needs (for example, you have to add bean reference information).

Multiple Deployments

The name "cis" for the web application is just the default installation name for the web application. You can easily name it in a different way. As with any other correct web application, you can also deploy it multiple times to the same servlet engine.

This is important for you in case you add the Application Designer web application to an existing web application on your side. You add Application Designer to your web application as you add normal libraries, i.e. Application Designer is now running under the control of your own web application.

The following sections provide more information on this aspect.

Creating a Second Application Designer Web Application inside Your Tomcat Installation

To demonstrate the usage of Application Designer as a standard web application, you can create a second Application Designer instance inside your Tomcat environment (and a third, fourth, etc.): simply copy the whole *cis* web application directory and paste it with a new name in Tomcat's *webapps* directory.

Step by step: let us assume that the name of the new application is *secondcis*:

- Copy the directory `<installdir>/tomcat/webapps/cis`.
- Paste it in the *webapps* directory and rename the new directory to *secondcis*. As a result, you have the following directories:

```
<installDir>/tomcat/webapps/cis  
/secondcis
```

Each web application instance now contains an independent Application Designer. It is no problem to run different versions of Application Designer inside one servlet container.

If you want to access the demo workplace of the first instance, you reference the following:

```
http://localhost:51000/cis/HTMLBasedGUI/workplace/demo.html
```

If you want to access the demo workplace of the second instance you reference the following:

```
http://localhost:51000/secondcis/HTMLBasedGUI/workplace/demo.html
```

You do not have to make further configuration steps when working with Tomcat: the standard *web.xml* contains the `REALPATH` parameter as a pointer to the web application's directory. This means that the directory path is determined automatically. If you create a second web application in a different servlet container, you might have to adjust the *web.xml* file after copying so that the new web application points to the right directory.

Adding Application Designer to an Existing Web Application

The same way we created a second instance of Application Designer in the section before, we can now add Application Designer to your existing web application in which you want to use Application Designer functions. Just copy the *cis* directory inside your web application's directory and merge the *web.xml* files of both web applications.

Building a Web Application Archive

A web application archive can be simply built by zipping one whole web application directory into a corresponding *.war* file.

However, there may also be more complex scenarios in which you want your deployable runtime to be structured differently compared to your design time. See the section [Design Time Mode and Runtime Mode](#).

Application Designer also provides the WAR Packager tool that takes over the building of the *.war* file. This tool is designed to cover very basic scenarios of packaging a web application archive file. It should only be used if you are not familiar with ANT build processes. See *WAR Packager* in the *Development Workplace* documentation for more information.

3 Design Time Mode and Runtime Mode

- When to Use which Mode 12
- Setup 12
- Class Loader Considerations 13
- File Access Considerations 13

Application Designer may run in two different modes:

Design Time Mode

All resource files which are required by Application Designer are read from the file system using the `cis.home` parameter value inside the *web.xml* configuration file.

The Application Designer class loader may be used. This means you can use the feature to dynamically reload application classes without having to restart the web application all the time.

Runtime Mode

All resource files are read internally via mechanisms of the servlet container, by which a web application can access its resource files.

The Application Designer class loader must not be used.

When to Use which Mode

The design time mode is typically used in the following scenarios:

- During development.
- With productive installations, if they are not clustered.

The runtime mode is used in the following scenarios:

- Productive installations which are distributed by the servlet container or application server on several cluster nodes.

The design time mode has the advantage that all resources are read from the file system, and are not blocked after access. This means that you can recreate and change these resources without restarting the web application. This simplifies the development a lot.

Setup

The switch from design time mode and runtime mode is configured in the *web.xml* file:

- If the `cis.home` parameter is set, the design time mode is switched on.

```
<init-param id="CISHOME">
  <param-name>cis.home</param-name>
  <param-value>REALPATH</param-value>
</init-param>
```

- If the `cis.home` parameter is not set, the runtime mode is switched on.

```
<!--
<init-param id="CISHOME">
  <param-name>cis.home</param-name>
  <param-value>REALPATH</param-value>
</init-param>
-->
```

Class Loader Considerations

Application Designer may use its own class loader below the web application's class loader. The purpose of this class loader is to dynamically replace classes during development in order to run newly compiled versions of your software without having to restart the web application all the time.

In the configuration file [cisconfig.xml](#), you can switch this possibility on or off.

See *Appendix D - Class Loader Concepts* for more information on what it means to change between “own Application Designer class loader” and “standard runtime with web application class loader”. Both expect classes to be located at different locations. As a consequence, you have to copy classes accordingly in order to bring your application from design time mode to runtime mode.

File Access Considerations

In design time mode (having a defined `cis.home` directory), classes and Application Designer resources (multi language files) are read from the file system. The reason is that classes can be reloaded without restarting the web application. In runtime mode, this is not done anymore: classes are read by the web application class loader, resources are read via the servlet context.

Consequence: there is no dependency from any file access to a predefined directory - Application Designer is completely clusterable.

4 Advice for Using Application Designer During Development

- Do not Use the Web Application cis 16
- Select the Right Directory Location for Your Web Application 16
- Typical Steps for Upgrading Your Web Application 17
- Resource Files to be Shared in a Versioning System 20
- Integrating Application Designer into Your Build Process 21

This chapter contains useful configuration advices for developing applications with Application Designer.

Do not Use the Web Application *cis*

Do not place your development in the standard web application *cis*. Always use a parallel web application so that *cis* is the “Application Designer reference” aside your own development.

Advantage: if installing Application Designer upgrades, you are always sure that these will never directly touch your development system. You can install upgrades inside the *cis* directory, have a look at them and then explicitly decide to take them over into your development web application.



Important: If you take over, always take over the whole reference web application, not parts of it.

Select the Right Directory Location for Your Web Application

By default, Tomcat administers its web applications within the directory *tomcat/webapps*. For each web application, there is one corresponding directory. For example, Application Designer is represented by directory *cis*.

You do not have to follow this structure. You can also decide to have your installation directory in a directory which is completely separated from your Tomcat installation.

There may be several reasons to do so:

- You may want to move the web application directory into your development project directories, i.e. you do not want to have your development objects distributed on several locations of your hard disk.
- You may want to have different Tomcat installations (for example, different versions) and use the same web application inside these Tomcat installations.

Example

Create the file *abcde.xml* in the directory *conf/Catalina/localhost*. Define the file in the following way:

```
<Context docBase="C:/Development/project/webui/webapp"
  privileged="true"
  antiResourceLocking="false"
  antiJARLocking="false">
</Context>
```

In this example, the web application *abcde* (this is the name of the file) is located in the directory *C:/Development/project/webui/webapp*. The directory that is defined by `docBase` is the web application directory which is internally structured in the following way:

```
webapp/cis
  /cisdemos
  ...
  ...
  /WEB-INF
  ...
```

Typical Steps for Upgrading Your Web Application

Software AG constantly publishes patched installation versions. With each patch, there is a brief documentation on the fixes that were done with the patch. The Application Designer build of a certain version always includes all patches of previous builds.

When installing a build, you have to follow a certain procedure in order to correctly take over the files of the new build and to save some information that you might have changed and which may get overwritten:

1. Start the installation wizard of the corresponding Application Designer build and select a temporary directory (*c:/temp/buildYYYYMMDD*) as the destination. As a consequence, the wizard will install a directory structure such as the following:

```
c:/temp/buildYYYYMMDD
  /tomcat
    /webapps
      /cis
        /cis
        /cisdemos
        ...
```

2. Make safety copies of all configuration files that Application Designer always delivers as part of the installation and that you may have changed. Application Designer offers a tool for this; see [Creating Safety Copies of Configuration Files](#) for more information.
3. Copy the content from the *cis* directory into your own web application.
4. Regenerate your layout definitions. You can do this

- either by using the Layout Manager in the development workplace (open your project(s) and regenerate all layout definitions as described in the *Development Workplace* documentation under *Generating HTML Pages*) or by using Ajax Developer;
 - or - and better: define a script (batch) in which you call the Application Designer class `com.softwareag.cis.gui.generate.HTMLGeneratorWholeDirectory` in order to regenerate all layout definitions of one project in one step. See also *Java API for HTML Page Mass Regeneration* in the *Development Workplace* documentation.
5. If you have defined your own style sheet using the Style Sheet Editor, proceed as described in the *Development Workplace* or Ajax Developer documentation under *Regenerating Your Own Style Sheet from the Style Sheet Template*.

Automate the Upgrade Using ANT

We strongly recommend that you use ANT for automating the upgrade of an installation. All steps (taking over the files and regenerating the layouts) can be easily defined using an ANT script. See [Using ANT for Upgrades](#) in which details are explained and a sample ANT script is provided.

Creating Safety Copies of the Configuration Files

In case you created your customized configuration files with the names *user_*.xml*, you simply have to make sure to keep your *user_*.xml* files. In addition, you have to make a safety copy of the file *web.xml*. For information on how to create custom configuration files with the names *user_*.xml*, see [Customizing Configuration Files](#).

In case, you modified Application Designer's default configuration files, you can create safety copies as described below.

A Configuration Manager is available for managing the core configuration files and which helps you to keep them consistent through upgrades of Application Designer. This tool is designed for “non-ANT-minded” developers. All “ANT-minded” developers should refer to the ANT way of upgrading your system.

We recommend using ANT. Nevertheless, the description of the Configuration Manager is provided below.

All configurations that are delivered with each Application Designer build and that you might modify on your own are critical when it comes to applying new builds. Use the Configuration Manager that comes with Application Designer as described below.

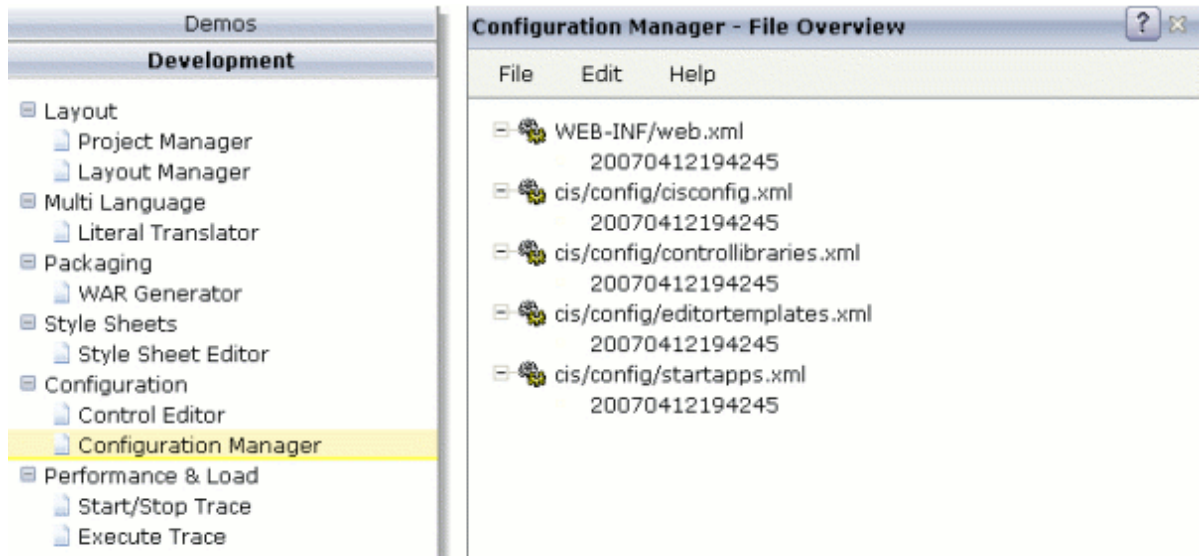
► To create safety copies of the configuration files

- 1 Invoke the demo workplace.
- 2 In the navigation frame, choose the **Development** button.
- 3 In the **Configuration** node, choose **Configuration Manager**.

A list of all critical files is shown.

- 4 From the **File** menu, choose **Create Safety Copies**.

The system automatically creates copies of the current files. The name of each copy includes a timestamp.

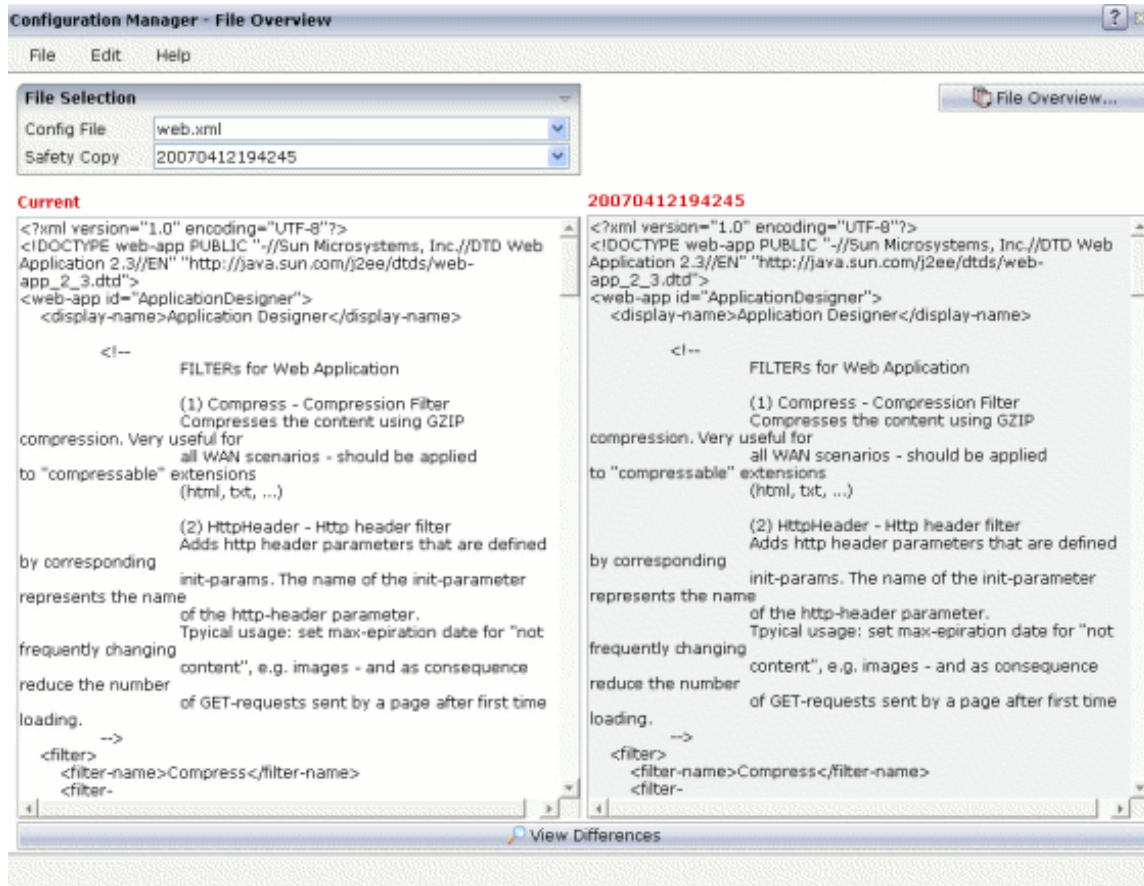


After having created the safety copies, you can, for example, upgrade your system to a new Application Designer build by copying/installing it over your existing application.

► **To check the differences after an upgrade**

- 1 After the upgrade, start the Configuration Manager once again (as described above).
- 2 Invoke the context menu for a safety copy and choose the **Compare** command.

The differences between the new version of the file and the saved version are shown.



- 3 You can edit the current file on the left side and take over changes from the right. Save the file by choosing **Save Current** from the **File** menu.

 **Note:** You can choose the **View Differences** button to better see what has changed.

Resource Files to be Shared in a Versioning System

When working in teams, you typically you use a central source versioning system (for example, CVS, Perforce, Visual Source Safe or others).

The resources to be synchronized are:

- **Adapter Java Sources**

The location of the source files is up to you.

■ Layout Definitions

Each Application Designer project has a set of XML layout definitions that represent the layout of pages. The definitions are transferred to generated HTML by using the Application Designer development tools. The definitions are stored in the *xml* directory of a project.

■ Translation Information

If using the standard multi language management, literal translations are stored in comma-separated files in the project's *multilanguage* directory.

■ Help Texts

If using the standard online help management, help texts are stored in the project's *help* directory.

■ Other Web Resources

Typically you have images and additional HTML pages that you use in your project. For example, your project has a subdirectory *images* that contains all GIF and JPG files.

■ Configuration Files

It may also make sense to share configuration files. For example, you may modify the *web.xml* file in order to access EJBs. In this case, you may centrally define the *web.xml* file and automatically synchronize it through the source code versioning system. The same may apply for all other configuration files.

Integrating Application Designer into Your Build Process

Typically you define the build process to consistently build up a system out of its resources. You may use simple scripts to do so, or you may use tools such as ANT.

See the Java API documentation for the following classes in order to find information on how to automate the generation of Application Designer layouts into HTML pages:

- `com.softwareag.cis.gui.generate.HTMLGenerator`
- `com.softwareag.cis.gui.generate.HTMLGeneratorWholeDirectory`

5 Configuration of Application Designer

▪ Overview of Configuration Files	24
▪ web.xml	24
▪ cisconfig.xml	26
▪ controllibraries.xml - Adding Control Libraries	34
▪ editortemplates.xml	35
▪ editor.xml - Available Controls	35
▪ startapps.xml - Applications to be Started	35
▪ Customizing Configuration Files	35

In general, you can use Application Designer from scratch, that is: without any further configuration. This chapter explains certain options for fine-tuning Application Designer.

Overview of Configuration Files

The *web.xml* file is located according to the servlet specification:

```
<webapplication>/WEB-INF/web.xml
```

Inside the Application Designer installation's web application there is a directory *cis/config* in which you can find the Application Designer configuration files.

```
<webapplication>/cis/config/cisconfig.xml  
<webapplication>/cis/config/controllibraries.xml  
<webapplication>/cis/config/editor.xml  
<webapplication>/cis/config/editortemplates.xml  
<webapplication>/cis/config/startapps.xml
```

web.xml

The *web.xml* file contains:

- technical information about the servlets that are defined inside Application Designer and how they are accessed, and
- configuration information.

This section only focuses on the configuration information.

Inside the definition for the servlet `Connector` there are two `init-param` elements that are relevant for the system configuration:

```
<servlet id="Connector">  
  <servlet-name>Connector</servlet-name>  
  <display-name>Connector</display-name>  
  <servlet-class>com.softwareag.cis.server.Connector</servlet-class>  
    <init-param id="CISHOME">  
      <param-name>cis.home</param-name>  
      <param-value>REALPATH</param-value>  
    </init-param>  
    <init-param id="CISLOG">  
      <param-name>cis.log</param-name>  
      <param-value>REALPATH/../../../../log/</param-value>  
    </init-param>
```

```
<load-on-startup>1</load-on-startup>
</servlet>
```

<p><code>cis.home</code></p>	<p>This parameter points to the directory location of the web application - if you are using it in design time mode. During design time, Application Designer needs to know this file location in order to correctly place generated page files and other information. At runtime, especially if running the Application Designer web application in a clustered application server scenario, this parameter should be wiped out. For further information, see the section <i>Design Time Mode and Runtime Mode</i>.</p> <p>The parameter can either be set to a directory name (for example, <code>c:/cisinstall/tomcat/webapps/cis/</code>) or to the parameter <code>REALPATH</code>.</p> <p><code>REALPATH</code> is dynamically interpreted at runtime. The name is internally requested by using the <code>getRealpath()</code> method of the servlet context.</p> <p>Caution: The above mentioned method is not supported by all servlet containers, it is only supported by the servlet containers that explicitly deploy into the file system (such as Tomcat, Jetty and IBM Websphere).</p>
<p><code>cis.log</code></p>	<p>This parameter points to the directory to which log information is written. Take care in a clustered scenario that the directory is not set to a fixed directory value: the directory may not be available on each cluster node.</p> <p>You can use the <code>REALPATH</code> parameter and you can specify the log location relative to the directory (for example, <code>REALPATH/log</code>).</p> <p>You can specify <code>TEMP</code> to indicate that the log is written to a temporary directory that every servlet container must provide as part of the servlet specification. When using <code>TEMP</code>, your application is clusterable - the application server will tell Application Designer for each node where to store log information.</p> <p>There is a second parameter influencing the log: this parameter is located in the file <i>cisconfig.xml</i>, its name is <code>startmonitoringthread</code>. The Application Designer log file is only written if this parameter is set to "true". Reason: the log is not directly written to the log file but is always buffered in memory first. The monitoring thread is started every 5 seconds and writes the buffered data to the file system. If the <code>startmonitoringthread</code> is not started, the log is automatically written to the logging that is provided by the servlet container. (Internally, the servlet context's log method is used.) The same will happen if you wipe out the <code>CISLOG</code> section from the <i>web.xml</i> file. In this case, Application Designer will use the log interface provided by the servlet context for writing log information.</p>

cisconfig.xml

The following topics are covered below:

- [General Parameters](#)
- [Directory for Performance Traces](#)
- [Central Class Path Extensions for Development](#)

General Parameters

The *cisconfig.xml* file contains some general control information. The following is a very basic example:

```
<cisconfig startmonitoringthread="true"
  requestclienthost="false"
  debugmode="false"
  loglevel="EWI"
  logtoscreen="false"
  sessiontimeout="3600"
  xmldatamanager="com.softwareag.cis.xmldata.filebased.XMLDataManager"
  useownclassloader="true"
  browserpopuponerror="false"
  framebuffersize="3"
  ↵
  onlinehelpmanager="com.softwareag.cis.onlinehelp.projectbased.FrameHelpOHManager"
  textencoding="UTF-8"
  enableadapterpreload="true">
</cisconfig>
```

animatecontrols	<p>Default: true.</p> <p>Defines how Application Designer handles the animation of controls. There are several controls that can be rendered in an animated way and in a standard way.</p> <p>Setting this parameter to "false" can help to improve performance, especially if you are not using the newest hardware.</p> <p>Values: true/false.</p>
browserpopuponerror	<p>Default: false.</p> <p>Defines how Application Designer handles it if the application behind an Application Designer page throws an error.</p> <p>By default (false), the browser switches to an error screen. In the screen, the user can only abort the current function. This is the default way in which any kind of inconsistency is automatically omitted.</p>

	<p>When you set <code>browserpopuponerror</code> to "true", the browser opens a pop-up window in which the error is output. This setting should only be used during development because it may cause inconsistencies in the application.</p> <p>Values: true/false.</p>
<code>clientsideerrorinstatusbar</code>	<p>Default: false.</p> <p>By default, client-side error messages are displayed as pop-ups.</p> <p>When you set this parameter to "true", client-side error messages are displayed in the status bar.</p> <p>Values: true/false.</p>
<code>collectionorblocklimit</code>	<p>Default: 300.</p> <p>Defines the maximum number of items in a grid after which the framework automatically switches from client-side scrolling to server-side scrolling. See also the description of <code>onloadbehaviour="collectionorblock"</code> in <i>Setting the Client-Side Loading Behavior</i> in <i>Layout Elements</i>.</p>
<code>createhttpsession</code>	<p>Default: false.</p> <p>Internally, Application Designer does not require HTTP session management that is provided by the servlet container. Some application servers (especially in clustered scenarios in which Application Designer runs in several nodes) require an explicit HTTP session ID to be used in order to route requests from a browser client always to the right application server node in the cluster. Set <code>createhttpsession</code> to "true" in this case.</p> <p>Values: true/false.</p>
<code>debugmode</code>	<p>Default: false.</p> <p>A log is written permanently into Application Designer's <i>log</i> directory. When <code>debugmode</code> is set to "true", a lot of information which normally is not required is written to the log.</p> <p>Be aware that you can also set the debug mode dynamically within your running system. Application Designer provides a monitoring tool in which you can switch the debug mode on and off.</p> <p>Values: true/false.</p>
<code>defaultcss</code>	<p>You can set your own default style sheet for your entire application. For example:</p>

	<code>../cis/styles/MY_STYLE.css</code>
<code>defaultlanguage</code>	<p>Default: en (English).</p> <p>Defines the language that is to be used by default when starting Application Designer. If not set, "en" is used. See <i>Multi Language Management</i> for detailed information on using different languages with Application Designer.</p>
<code>designtimeclassloader</code>	<p>By default, Application Designer uses an own class loader for accessing adapter classes at design time. (You can switch this off by specifying <code>useownclassloader="false"</code>.)</p> <p>With the <code>designtimeclassloader</code>, you can explicitly select a class loader class that Application Designer is to use. This allows you to use class loaders that offer special functions such as reading encrypted class files.</p> <p>Value: the name of a class loader class.</p>
<code>enableadapterpreload</code>	<p>Default: true.</p> <p>By default, the server sends all required responses at once to the client, even if different adapters are involved.</p> <p>If set to "false", a separate data transfer occurs for each involved adapter.</p>
<code>errorreactionadapter</code>	<p>In case of an unhandled application error, the Application Designer runtime navigates to an error page. The class name specified in <code>errorreactionadapter</code> is the Java adapter for this error page.</p> <p>If an error reaction adapter is not specified, a default adapter is used which shows the error's stack trace.</p> <p>The Application Designer framework contains a second error reaction adapter with the class name <code>com.softwareag.cis.server.SecureErrorReactionAdapter</code>. For security reasons, this adapter does not show a stack trace but only an error message.</p> <p>You can write your own error reaction adapter and create your own error page. An error reaction adapter must implement one of the interfaces <code>com.softwareag.cis.server.ISecureErrorReactionAdapter</code> or <code>com.softwareag.cis.server.IErrorReactionAdapter</code>. For more information, see the corresponding Java documentation.</p>
<code>fieldnumerictypesrightaligned</code>	<p>Default: false.</p> <p>Set this parameter to "true" in order to right-align text within the FIELD control when using the data type <code>int</code>, <code>long</code> or <code>float</code>.</p> <p>Values: true/false.</p>

flushreceivespreviousfocused	<p>Default: false.</p> <p>By default, during a flush event the adapter gets as focus information the input control that <i>received</i> the focus. Set this parameter to "true" if during a flush event your application relies on getting as focus information the input control that <i>lost</i> the focus.</p> <p>For Natural applications this means: By default, the Natural system variable *CURS-FIELD contains during the flush event the value of the Natural system function POS for the input control that received the focus.</p> <p>Values: true/false.</p>
framebuffersize	<p>Default: 3.</p> <p>Each page in the browser client runs inside a surrounding page. This surrounding page offers a couple of internal functions, one of them to buffer contained Application Designer pages: if a user opens the first page and then navigates to a second page, the first page is internally kept inside a frame buffer. If returning to the first page later on, the browser does not have to build up the first page from scratch but just switches to the buffered page.</p> <p>The <code>framebuffersize</code> defines the number of buffered pages. Increasing the <code>framebuffersize</code> means that more resources are used on the client (browser) side. When changing this value, you should test the memory consumption on the client side before rolling out the change to productively running implementations.</p> <p>Value: integer number.</p>
loglevel	<p>Default: EWI.</p> <p>Defines the message types that are to be logged. Values:</p> <p>E (error) W (warning) I (information) D (debug)</p> <p>You can specify any combination of message types by concatenating the message types.</p> <p>Example: "EW" logs all error and warning messages. "EWI" additionally logs information messages.</p> <p>Caution: When having set <code>debugmode</code> to "true", the <code>loglevel</code> filter is automatically bypassed and all messages are logged. <code>debugmode</code> is stronger than <code>loglevel</code>.</p>
logtoscreen	<p>Default: false.</p>

	<p>If this parameter is set to "true", all Application Designer log information is also output to the command screen from which you started Application Designer. This parameter should only be set to "true" if running in development mode.</p> <p>Values: true/false.</p>
maxitemsinfieldcombo	<p>Default: 100.</p> <p>The FIELD control provides for a predefined pop-up method <code>openIdValueComboOrPopup</code>. Depending on the size of the list of valid values, the list is either shown in a combo box or in a pop-up. Use this parameter to control the maximum number of entries that are to be shown in the combo box.</p> <p>Value: integer number.</p>
maxworkplaceactivities	<p>Default: -1 (unlimited).</p> <p>The maximum number of workplace activities in a workplace application. For reactions when the maximum number is reached, see <i>Customizing the MFWPFUNCTIONS Behavior in Working with Pages</i>.</p>
multilanguagemanager	<p>Internally, Application Designer uses an interface to retrieve the translation information for a certain text ID and a certain language. A default implementation is available that stores the corresponding language information in files that are part of the web application. You can build your own multi language manager - by using the <code>com.softwareag.cis.multilanguage.IMLManager</code> interface - in case you already have an existing framework for multi language management.</p> <p>Value: the name of the class that supports Application Designer's multi language interface.</p>
natuppercase	<p>Default: false.</p> <p>Set this parameter to "true" if your Natural program only allows Latin upper-case characters. This is the case, for example, if your Natural program uses the Hebrew codepage CP803.</p> <p>Important: Set the parameter <code>natuppercase="true"</code> before you implement your main program with Natural for Ajax. If you set this parameter after the implementation, you will have to change all Latin lower-case characters to upper-case manually.</p> <p>Values: true/false.</p>
onlinehelpmanager	<p>Application Designer accesses a certain URL when the user presses F1 on certain controls (for example, fields, check boxes and others). Application Designer transfers a corresponding help ID that is defined with the control into a URL and opens this URL in a pop-up window. If you have your own mechanisms for defining this URL, you can</p>

	<p>implement a corresponding Application Designer Java interface (<code>com.softwareag.cis.onlinehelp.IOHManager</code>).</p> <p>Value: the name of the interface.</p>
<code>requestclienthost</code>	<p>Default: false.</p> <p>If a client sends an HTTP request, it is determined for the first request from which client this request is coming. This operation is sometimes quite expensive. For this reason, you can switch it off. If switched off, there is no disadvantage in normal operation, besides in the monitoring tool you cannot identify which session belongs to which client.</p> <p>Values: true/false.</p>
<code>requestdataconverter</code>	<p>Application Designer allows to pass each value that is input by the user through an explicit data converter on the server side, prior to passing this value to the application. In the data converter, you can implement certain security checks, for example, you can prevent users from inputting string sequences containing inline JavaScript or SQL scripting. See the interface <code>com.softwareag.cis.server.IRequestDataConverter</code> for more information. See also <i>Security Aspects</i> in the <i>Special Development Topics</i>.</p> <p>Value: name of a class that implements the interface <code>com.softwareag.cis.server.IRequestDataConverter</code>.</p>
<code>sessionidasthreadname</code>	<p>Default: true.</p> <p>On start of each page request processing, the Application Designer runtime calls the method <code>Thread.setName</code> with the current session ID (default).</p> <p>You can set this parameter to "false" to instruct the Application Designer runtime not to touch the thread's name.</p> <p>Values: true/false.</p>
<code>sessiontimeout</code>	<p>Default: 3600 (1 hour).</p> <p>Application Designer sessions are timed out according to the value defined with this parameter. This is the definition of the timeout phase in seconds. By default, 3600 is defined in the configuration file. If no parameter is specified in the configuration file, 7200 is used.</p> <p>Value: integer number.</p>
<code>startmonitoringthread</code>	<p>Default: true.</p> <p>If set to "true", a monitoring thread is opened which wakes up every 5 seconds. The thread performs the following activities:</p> <ol style="list-style-type: none"> 1. It initiates a garbage collection periodically (every two minutes). 2. It writes all log information into a log file (every five seconds).

	<p>3. It calls the clean up of sessions which are timed out (every two minutes)</p> <p>What happens if the monitoring thread is not started?</p> <ol style="list-style-type: none"> 1. No garbage collection will be triggered by Application Designer. This is then the task of the servlet container around. 2. The log is not automatically written to the file location specified in the <i>web.xml</i> file, but is written to the servlet container's logging. 3. Timing out sessions is not done every two minutes but every thousand requests. <p>Caution: Some servlet containers do not allow to let the web application start new threads (for example, the Sun reference implementations do so). For these containers, the parameter must be set to "false".</p> <p>Values: true/false.</p>
suppressfocusmanagement	<p>Default: false.</p> <p>If you set this parameter to "true", no focus management in the client will be done after a server round trip. This means: The focus will not be set to focus-requesting controls such as "EDIT" fields with "ERROR" status after a server round trip.</p> <p>Usually, you do not set this parameter. If you need to suppress focus management for specific server round trips, you usually do this from within your adapter code for these specific server round trips. See also the <code>focusmgtprop</code> in the NATPAGE control. Only set this parameter to "true" if your application needs to do it vice versa: Suppress focus management for nearly all server round trips and only explicitly activate focus management for some specific server round trips from within your adapter code.</p> <p>Values: true/false.</p>
takeoutfieldpopuicon	<p>Default: false.</p> <p>Set this parameter to "true" in case you are using right-aligned FIELD controls with value help. This will avoid overlapping of the right-aligned text and the corresponding drop-down icon.</p> <p>Values: true/false.</p>
textencoding	<p>Default: UTF-8.</p> <p>By default, Application Designer reads and writes text files in UTF-8 format. You can tell Application Designer to use a different format (for example, for writing XML layout definitions). But be very careful and very aware of what you are doing.</p> <p>See also <i>Unicode</i> in the <i>Multi Language Management</i> documentation.</p>

<p>urlsessiontimeout</p>	<p>When Application Designer times out a session (see the <code>sessiontimeout</code> parameter) and the user tries to continue to work with the session, a page will be displayed inside the user's browser, indicating that a timeout happened with the user's session. By default, this page is an Application Designer page that you might not want to show to your application users.</p> <p>Value: the URL of the page that is to be shown instead of the default page.</p>
<p>usemessagepopup</p>	<p>Default: false.</p> <p>Set this parameter to "true" in order to show status messages as message pop-ups.</p> <p>Values: true/false.</p>
<p>useownclassloader</p>	<p>Default: true.</p> <p>If set to "true", Application Designer uses its own class loader to load application classes.</p> <p>This parameter may be set to "false" in certain environments, for example, if you use Application Designer inside an environment which requires all application classes to run in the environment's own class loader environment.</p> <p>Caution: The Application Designer class loader automatically searches for classes in certain directories (<code><project>/appclasses/classes</code> and <code><project>/appclasses/lib</code>). If you do not use the Application Designer class loader, you have to set up your environment accordingly.</p> <p>Values: true/false.</p>
<p>xmldatamanager</p>	<p>This parameter defines the file name of the class which implements the <code>com.softwareag.cis.xmldata.IXMLDataManager</code> interface. You can specify an own class here. The <code>com.softwareag.cis.xmldata.XMLDataManagerFactory</code> creates an instance using a constructor without any parameter.</p>
<p>zipcontent</p>	<p>Default: true.</p> <p>Between the browser and the server, data content is exchanged. By default, Application Designer zips the content before sending a response from the server to the browser client.</p> <p>Sometimes you may want to actually "see" what is being sent (maybe you have a test tool that captures the HTTP protocol). Set <code>zipcontent</code> to "false" if you do not want Application Designer to zip the data content returned to the client.</p> <p>Values: true/false.</p>

Directory for Performance Traces

The `requestrecording` section of the `cisconfig.xml` file indicates the directory in which recorded performance traces are stored.

```
<cisconfig ...>
  <requestrecording recordrequests="false"
    recorddirectory="c:/temp/traces/">
  </requestrecording>
</cisconfig>
```

See also:

- *Recording a Performance Trace* in the *Development Workplace* documentation.
- *Recording a Performance Trace* in the *Ajax Developer* documentation.

Central Class Path Extensions for Development

If you want to use your own class path extension, you may add a subsection to the `cisconfig.xml` file in which you extend the class path of the Application Designer class loader at development time:

```
<cisconfig ...>
  <classpathextension path="c:/development/centralclasses/classes"/>
  <classpathextension path="c:/development/centralclasses/libs/central.jar"/>
</cisconfig>
```

Each class path extension is listed with a reference to its physical path.

controllibraries.xml - Adding Control Libraries

In this file, all control libraries are registered which you use for your layout definitions. You only need to modify this file if you use non-Application Designer control libraries. For details, see *Customized Controls*.

editortemplates.xml

This file defines the layout templates that are offered for selection when you create a new layout with the Layout Painter. If you do not want to use the default templates, you can customize them. For details, see the comments in the *editortemplates.xml* file.

See also:

- *Layout Templates* in the *Development Workplace* documentation.
- *Layout Templates* in the *Ajax Developer* documentation.

editor.xml - Available Controls

This file contains data about all the controls which may be used inside the Layout Painter. You should never change this file - Application Designer offers a smart way to append your own definitions to the ones coming from Application Designer: You can create *editor_<xxxxxx>.xml* files in which you specify your delta compared to *editor.xml*. For details, see:

- *Using the Control Editor* in the *Development Workplace* documentation.

startapps.xml - Applications to be Started

It is possible to define that certain applications require to be started immediately inside the start processing of Application Designer. For details, see *Becoming a Member of the Startup Process* in the *Special Development Topics*.

Customizing Configuration Files

This description applies only available when Application Designer is part of a Natural for Ajax installation.

You can customize the following default configuration files:

cisconfig.xml
controllibraries.xml
editortemplates.xml
startapps.xml


However, modifying the above default configuration files has the following disadvantage: With each Application Designer version or update package, Application Designer brings its own default configuration files. If you forget to save your settings before installing an upgrade, your customized files will be overwritten by the upgrade. Therefore, it is more convenient if your customized files do not have the same names as the default files.

The Configuration Manager tool supports the creation of custom configuration files with the following names:

user_cisconfig.xml
user_controllibraries.xml
user_editortemplates.xml
user_startapps.xml

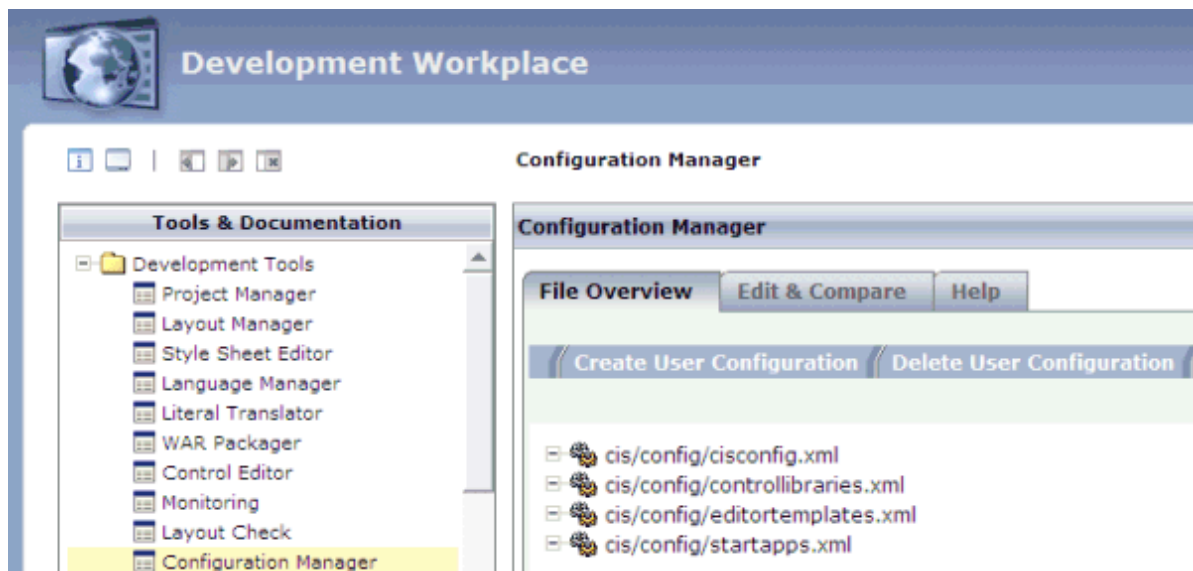
Instead of modifying the default configuration files, it is recommended that you modify the corresponding *user_*.xml* files.

Application Designer always checks whether a custom configuration file with the name *user_*.xml* exists. When it exists, Application Designer uses the *user_*.xml* file and ignores the default file. If a *user_*.xml* file does not exist, Application Designer uses the default file.

 **Note:** *editor.xml* is not intended to be modified. If you want to add your own controls, you should write your own *editor_<xxxxx>.xml* file as described above for the *editor.xml* file.

▶ **To create custom configuration files (*user_*.xml* files)**

- 1 Start the development workplace.
- 2 In the **Development Tools** node of the navigation frame (which is visible when the **Tools & Documentation** button has previously been chosen), choose **Configuration Manager**.



- 3 Choose the **Help** tab which is shown for the Configuration Manager.
- 4 See the help text for information on how to proceed.

6 HTTP Data Volume Considerations

- The Browser's Loading Behavior 40
- How does Application Designer Fit in? 41
- Configuration Options for Optimized HTTP Usage 42

This chapter discusses the data volume that is transferred between the server and the browser client at runtime. You should read this chapter carefully when deploying your Application Designer based application for productive usage - especially in scenarios in which limited bandwidth is available.

Application Designer is optimally designed to serve these kinds of applications, but you have to know and understand how a browser requests information from a server in order to configure your deployment in an optimal way.

The Browser's Loading Behavior

Browsers load data from the server by specifying a URL and getting back some content (for example, HTML text). This happens, for example, for any static files that are loaded:

- pages (HTML),
- images (GIF, JPG, etc.),
- other resources.

In order to reduce the volume of data exchanged, the browser uses caching. Once the content for a specific URL is loaded, the browser will keep the result in its cache, together with a timestamp that is passed as header information of the loaded content.

How does caching work (now assuming that you use the standard browser configuration)?

- Once the content is loaded into the browser, the browser will not ask again for the content in the browser session, i.e. if you keep your browser up and running, it will use the cached content.
- But if you close the browser and reopen it again, the browser will check whether the cached content is still up to date. It will send a URL to the server passing the timestamp of the cached content as part of the request. The server will check whether the timestamp of the cached content is valid - if so, it only sends back a short "OK" message to the browser, but does not include the content. If the timestamp is outdated, the server will send back the now up to date content.

You see: switching cache on (as it is in the standard configuration) still does mean that there is communication between the browser and the client when starting a new browser. The number of URLs that are sent as requests to the server is still the same - just the responses are smaller.

Imagine a page that internally holds 10 images. If not yet cached, there will be 11 requests towards the server, and 11 responses, each response containing the content (1 HTML page, 10 image byte-streams). Now closing the browser and reopening the page, you will see that again 11 requests are sent to the server. This time, the server will not send back the content, but just 11 acknowledgement messages that tell the browser that its cached content is still valid.

When improving the data transfer between browser and server, both aspects do play a significant role:

- The number of exchanged requests.
- The number of exchanged bytes.



Note: A request over satellite-based networks may have a roundtrip time of about 500 ms - even if very few pieces of information are actually exchanged.

How does Application Designer Fit in?

Application Designer pages are kept stable inside the browser. They are not permanently reloaded but exchange their data content with the server. The data content is exchanged in delta mode, i.e. only changed data properties are exchanged.

Result: once pages are loaded, Application Designer is “unbeatably” fast (compared to traditional web frameworks in which pages are constantly resent to the client) because of the clear separation of page loading and data loading. A data roundtrip (which is internally zipped) has a size of 1 to 3 kBytes typically (depending on the page).

What is the price? Application Designer pages contain JavaScript statements and reference to JavaScript libraries (that are shared cross pages, of course), i.e. pages are by nature bigger than plain HTML pages. Of course, most of the code is shared across different pages (by using libraries). However, loading an initial page without any information cached in the browser means:

1. loading the page itself (e.g. 50 kBytes)
2. loading libraries (e.g. 150 kBytes)
3. loading images (e.g. 30 kBytes)

and then finally:

4. loading page data (1-3 kBytes)

Only step 4 contains dynamic data. All other steps are requests against static content.

Consequence: you have to explicitly keep an eye on steps 1 to 3. From a communication perspective, they are the biggest parts of the initial load, but afterwards need not be reloaded into the browser because they contain static content.

Configuration Options for Optimized HTTP Usage

There are two ways to optimize the HTTP communication:

1. Use compression for all textual information.
2. Make use of browser cache and avoid roundtrips.

For both purposes, Application Designer offers so-called servlet filters. Filters are small programs that you can configure to wrap HTTP requests coming into a server. In the web application (via the *web.xml* file), you configure which filters are used in which way for which requests.

The two Application Designer filters are:

- `com.softwareag.cis.server.filter.CompressionFilter`
A filter that zips content that is loaded from the web application.
- `com.softwareag.cis.server.filter.HttpHeaderFilter`
A filter that sets HTTP header attributes for requests. One of the most important attributes that will be used here is the expiration date of content.

Both filters and the way to use them are explained below:

- [Filter for Compressing Content](#)
- [Filter for Setting HTTP Header Attributes](#)

Filter for Compressing Content

Have a look at the following example configuration (*/WEB-INF/web.xml* file):

```
<web-app id="cis">
  ...
  ...
  ...
  <filter>
    <filter-name>Compress</filter-name>
    ↵
  <filter-class>com.softwareag.cis.server.filter.CompressionFilter</filter-class>
  </filter>
  ...
  ...
  ...

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



```

    <filter-name>Compress</filter-name>
    <url-pattern>*.bmp</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>Compress</filter-name>
    <url-pattern>*.js</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>Compress</filter-name>
    <url-pattern>/servlet/StartCISPage*</url-pattern>
</filter-mapping>
...
...
...
<servlet id="Connector">
...
...
...

```

First of all, the filter is defined in the `<filter>...</filter>` section. Then, URL patterns are defined for which the filter is to be used. In the example, the compression filter compresses any requests that are HTML pages, non-compressed images (bitmaps), JavaScript library files and calls against the Application Designer servlet to start new Application Designer pages.

The example represents a reasonable filter configuration for running Application Designer: uncompressed content (text, bitmaps) is filtered, i.e. is zipped; compressed content (e.g. JPG images, PDF files) are passed through without being compressed again.

Of course, compression has its price on the server side. A compressed request requires some time for executing the compression, but it is typically worth it.

Filer for Setting HTTP Header Attributes

Each request to which a server responds holds a certain number of HTTP header attributes that surround the content of the response. One important HTTP header parameter is the expiration date of a page.

Let us come back to the caching discussion previously in this section. A browser holds cached content associated with the timestamp of loading. If closing and reopening a browser, it will still send content requests to the server to check whether its content is valid, and will receive acknowledgement messages or refreshed content.

When now using the header attribute `max-age`, you can tell the browser that a content has a certain stability: for example, you tell the browser that when loading a page today, it will not expire in the next 24 hours. When now being closed and reopened, the browser will only send HTTP requests to the server that are expired. Now, with defining expiration dates, the browser will really reduce the number of URLs that are sent to the server when being reopened.

```
...
...
...
<filter>
  <filter-name>HttpHeader</filter-name>
  <filter-class>com.softwareag.cis.server.filter.HttpHeaderFilter</filter-class>
  <init-param>
    <param-name>Cache-Control</param-name>
    <param-value>max-age=86400</param-value>
  </init-param>
</filter>
...
...
...
<filter-mapping>
  <filter-name>HttpHeader</filter-name>
  <url-pattern>*.gif</url-pattern>
</filter-mapping>
<filter-mapping>
  <filter-name>HttpHeader</filter-name>
  <url-pattern>*.jpg</url-pattern>
</filter-mapping>
...
...
...
```

In the filter definition, you see that a header attribute `max-age` is set for filtered HTTP responses. The value is set to 86400 seconds (this is 24 times 3600s - one day). The filter is applied to GIF files and JPG files.

Now it depends on some environment conditions how to configure the filter. Ad hoc candidates for filtering are:

- All graphic files. Typically they do not change too frequently and they do not have a direct impact on the consistency of an application.
- Static content, e.g. PDF documents.

Of course you might also think about filtering other files such as:

- JavaScript libraries.
- Generated Application Designer HTML pages.

But be aware of the following: if you upgrade your server software (for example, you modify some of your pages), you will have to explicitly tell your users to clear their cache. Otherwise, they will keep their old pages until they are expired.

The Application Designer recommendation for long term caching via the expiration date is:

- You must use it for any static images.

- You should be very careful with any content-related parts of your application.

Be aware of the fact that you can define the filter several times inside your *web.xml* definition:

```
...
...
...
<filter>
  <filter-name>HttpHeaderLongTerm</filter-name>
  <filter-class>com.softwareag.cis.server.filter.HttpHeaderFilter</filter-class>
  <init-param>
    <param-name>Cache-Control</param-name>
    <param-value>max-age=86400</param-value>
  </init-param>
</filter>
<filter>
  <filter-name>HttpHeaderShortTerm</filter-name>
  <filter-class>com.softwareag.cis.server.filter.HttpHeaderFilter</filter-class>
  <init-param>
    <param-name>Cache-Control</param-name>
    <param-value>max-age=3600</param-value>
  </init-param>
</filter>
```


7 Using ANT for Upgrades

- ANT File for Take Over 48
- Step by Step 51
- Running the ANT Script 52

After installing Application Designer, the web application "cis" will be available inside a servlet engine. The servlet engine either comes with the Application Designer installation (when using the installation wizard) or is your own servlet engine (when using the *.war* distribution).

When starting to develop with Application Designer, we recommend the following procedure:

- Make a copy of the standard "cis" web application into a new, parallel running web application. You can use a name of your choice; in this chapter we will use the name "ciscopy".
- Use the "cis" web application as an "untouched" Application Designer reference and make all of your development inside "ciscopy".

Reason: when installing new or patched versions of Application Designer, you are sure that changes will not immediately have an effect on your web application. There is one buffer between what comes from Software AG and your development.

The take over from the reference "cis" web application to your copy "ciscopy" should be automated in order to reduce the effort when applying new Application Designer versions. We recommend to do this automation using an ANT script definition.

This chapter gives an example of an ANT script definition. The ANT script can be used for both the first, initial copying from the reference and for continuous copying from the reference.

You are not yet familiar with ANT? No problem - just download ANT from <http://ant.apache.org/>, extract it into a directory of your choice, that's it.

ANT File for Take Over

Let us have a look at the XML file that defines the ANT job:

```
<?xml version="1.0"?>
<project name="Copy Into Parallel Project" default="generatehtml">

    <description>Copy Default Application Designer Environment (cis) to an existing
Web Project (ciscopy)</description>

    <!--
        The following three properties need to be updated according to your
        situation.
    -->
    <property name="from.dir" value="C:/TEMP/11111/tomcat/webapps/cis"/>
    <property name="to.dir" value="C:/TEMP/11111/tomcat/webapps/ciscopy"/>
    <property name="project" value="yourproject"/>

    <path id="generate.classpath">
        <pathelement location="${to.dir}/WEB-INF/lib/cis.jar"/>
    </path>
```

```

<target name="copyfiles" description="Copy Files">

    <!-- Copy cis configuration directory-->
    <copy file="${to.dir}/cis/config/cisconfig.xml"
        tofile="${to.dir}/cis/config/cisconfig_customer.xml" ↵
failonerror="false"/>
    <copy file="${to.dir}/cis/config/startapps.xml"
        tofile="${to.dir}/cis/config/startapps_customer.xml" ↵
failonerror="false"/>
    <copy file="${to.dir}/cis/config/controllibraries.xml"
        tofile="${to.dir}/cis/config/controllibraries_customer.xml" ↵
failonerror="false"/>
    <copy file="${to.dir}/cis/config/editortemplates.xml"
        tofile="${to.dir}/cis/config/editortemplates_customer.xml" ↵
failonerror="false"/>
    <copy file="${to.dir}/cis/config/adapterTemplate.java"
        tofile="${to.dir}/cis/config/adapterTemplate_customer.java" ↵
failonerror="false"/>
    <copy file="${to.dir}/cis/config/adapterTemplate.java"
        tofile="${to.dir}/cis/config/adapterTemplate_customer.java" ↵
failonerror="false"/>
    <copy file="${to.dir}/cis/config/fopClassTemplate.java"
        tofile="${to.dir}/cis/config/fopClassTemplate_customer.java" ↵
failonerror="false"/>

    <copy todir="${to.dir}/cis">
        <fileset dir="${from.dir}/cis">
            </fileset>
        </copy>

    <copy file="${from.dir}/cis/config/cisconfig.xml"
        tofile="${to.dir}/cis/config/cisconfig_sagreference.xml"/>
    <copy file="${from.dir}/cis/config/startapps.xml"
        tofile="${to.dir}/cis/config/startapps_sagreference.xml"/>
    <copy file="${from.dir}/cis/config/controllibraries.xml"
        tofile="${to.dir}/cis/config/controllibraries_sagreference.xml"/>
    <copy file="${from.dir}/cis/config/editortemplates.xml"
        tofile="${to.dir}/cis/config/editortemplates_sagreference.xml"/>
    <copy file="${from.dir}/cis/config/adapterTemplate.java"
        tofile="${to.dir}/cis/config/adapterTemplate_sagreference.java"/>
    <copy file="${from.dir}/cis/config/fopClassTemplate.java"
        tofile="${to.dir}/cis/config/fopClassTemplate_sagreference.java"/>

    <copy file="${to.dir}/cis/config/cisconfig_customer.xml"
        tofile="${to.dir}/cis/config/cisconfig.xml" failonerror="false"/>
    <copy file="${to.dir}/cis/config/startapps_customer.xml"
        tofile="${to.dir}/cis/config/startapps.xml" failonerror="false"/>
    <copy file="${to.dir}/cis/config/controllibraries_customer.xml"
        tofile="${to.dir}/cis/config/controllibraries.xml" failonerror="false"/>
    <copy file="${to.dir}/cis/config/editortemplates_customer.xml"

```

```

        tofile="${to.dir}/cis/config/editortemplates.xml" failonerror="false"/>
    <copy file="${to.dir}/cis/config/adaptTemplate_customer.java"
        tofile="${to.dir}/cis/config/adaptTemplate.java" failonerror="false"/>
    <copy file="${to.dir}/cis/config/fopClassTemplate_customer.java"
        tofile="${to.dir}/cis/config/fopClassTemplate.java" ↵
failonerror="false"/>

    <delete file="${to.dir}/cis/config/cisconfig_customer.xml"/>
    <delete file="${to.dir}/cis/config/startapps_customer.xml"/>
    <delete file="${to.dir}/cis/config/controllibraries_customer.xml"/>
    <delete file="${to.dir}/cis/config/editortemplates_customer.xml"/>
    <delete file="${to.dir}/cis/config/adaptTemplate_customer.java"/>
    <delete file="${to.dir}/cis/config/fopClassTemplate_customer.java"/>

    <!-- Copy HTMLBasedGUI directory -->
    <copy todir="${to.dir}/HTMLBasedGUI">
        <fileset dir="${from.dir}/HTMLBasedGUI">
            </fileset>
        </copy>

    <!-- Copy WEB-INF directory -->
    <copy file="${to.dir}/WEB-INF/web.xml"
        tofile="${to.dir}/WEB-INF/web_customer.xml" failonerror="false"/>
    <copy todir="${to.dir}/WEB-INF">
        <fileset dir="${from.dir}/WEB-INF">
            </fileset>
        </copy>
    <copy file="${from.dir}/WEB-INF/web.xml"
        tofile="${to.dir}/WEB-INF/web_sagreference.xml"/>
    <copy file="${to.dir}/WEB-INF/web_customer.xml"
        tofile="${to.dir}/WEB-INF/web.xml" failonerror="false"/>
    <delete file="${to.dir}/WEB-INF/web_customer.xml"/>

    <!-- Copy all Application Designer demo projects that should be available ↵
in the copy -->
    <!--
        <copy todir="${to.dir}/cisdemos"><fileset ↵
dir="${from.dir}/cisdemos"/></copy>
        -->

    </target>

    <!-- Regenerate HTML files in core projects -->
    <target name="generatehtml" depends="copyfiles" description="Regenerate HTML ↵
files of project">
        <mkdir dir="${to.dir}/${project}/log"/>
        <java classname="com.softwareag.cis.gui.generate.HTMLGeneratorWholeDirectory"
            classpathref="generate.classpath">
            <sysproperty key="cis.home" value="${to.dir}/"/>
            <arg value="${to.dir}/${project}/xml"/>
            <arg value="${to.dir}/${project}"/>
            <arg value="${to.dir}/${project}/log"/>

```



```
        <arg value="\${to.dir}/${project}/accesspath"/>
    </java>
</target>

</project>
```

Step by Step

The ANT file contains three major sections:

1. Declaration of properties.
2. Target name "copyfiles" - the copying from this "cis" reference into the "ciscopy" copy is defined in this section.
3. Target name "generatehtml" - the regeneration of XML layouts into HTML files is defined in this section.

Declaration of Properties

There are three properties:

1. The name of the directory of the "cis" reference web application.
2. The name of the directory of the "ciscopy" web application.
3. The name of your application project within the "ciscopy" web application.

Define the properties according to your configuration.

Target Name copyfiles

The copying is done in three steps:

1. Copying the directory *cis*.
2. Copying the directory *HTMLBasedGUI*.
3. Copying the directory *WEB-INF*.

For certain files that you may have changed during development (for example, *cis/cisconfig.xml*), the ANT file will create corresponding copies of the original files coming from Software AG - and will leave your version untouched.

Only the core directories of your "cis" reference will be copied. The demo application will not be copied (directory *cisdemos*). In the ANT script, you see a commented section that you can uncomment in order to also copy this directory.

Target Name generatehtml

In this step, the XML files of your project will be regenerated.



Caution: If you have created your own controls - with your own control classes (ITagHandler, IMacroTagHandler implementations) - you have to adapt the classpath that is used for generation to also contain your control handler classes.

Running the ANT Script

The following batch file for Windows shows how to start the ANT script, assuming the name of the file is *copy_reference.xml*:

```
set JAVA_HOME=C:\Java\jdk142
C:\ant\bin\ant.bat -buildfile copy_reference.xml
```

Adapt the file to your directory configuration.