Configuration of Application Designer

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. The following topics are covered:

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

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>
```

`cis.home`	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 *Design Time Mode and Runtime Mode*. The parameter can either be set to a directory name (for example, *c:/cisinstall/tomcat/webapps/cis/*) or to the parameter `REALPATH`. `REALPATH` is dynamically interpreted at runtime. The name is internally requested by using the `getRealpath()` method of the servlet context. **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).
`cis.log`	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. You can use the `REALPATH` parameter and you can specify the log location relative to the directory (for example, `REALPATH/log`). You can specify `TEMP` 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 `TEMP`, your application is clusterable - the application server will tell Application Designer for each node where to store log information. There is a second parameter influencing the log: this parameter is located in the file *cisconfig.xml*, its name is `startmonitoringthread`. 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 `startmonitoringthread` 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 `CISLOG` section from the *web.xml* file. In this case, Application Designer will use the log interface provided by the servlet context for writing log information.

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`	Default: true. 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. Setting this parameter to "false" can help to improve performance, especially if you are not using the newest hardware. Values: true/false.
`browserpopuponerror`	Default: false. Defines how Application Designer handles it if the application behind an Application Designer page throws an error. 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. When you set `browserpopuponerror` 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. Values: true/false.
`createhttpsession`	Default: false. 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 `createhttpsession` to "true" in this case. Values: true/false.

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

`framebuffersize`	Default: 3. 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. The `framebuffersize` defines the number of buffered pages. Increasing the `framebuffersize` 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. Value: integer number.
`loglevel`	Default: EWI. Defines the message types that are to be logged. Values: E (error) W (warning) I (information) D (debug) You can specify any combination of message types by concatenating the message types. Example: "EW" logs all error and warning messages. "EWI" additionally logs information messages. **Caution:** When having set `debugmode` to "true", the `loglevel` filter is automatically bypassed and all messages are logged. `debugmode` is stronger than `loglevel`.
`logtoscreen`	Default: false. 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. Values: true/false.
`maxitemsinfieldcombo`	Default: 100. The FIELD control provides for a predefined pop-up method `openIdValueComboOrPopup`. 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. Value: integer number.

`multilanguagemanager`	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 `com.softwareag.cis.multilanguage.IMLManager` interface - in case you already have an existing framework for multi language management. Value: the name of the class that supports Application Designer's multi language interface.
`onlinehelpmanager`	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 implement a corresponding Application Designer Java interface (`com.softwareag.cis.onlinehelp.IOHManager`). Value: the name of the interface.
`requestclienthost`	Default: false. 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. Values: true/false.
`requestdataconverter`	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 `com.softwareag.cis.server.IRequestDataConverter` for more information. See also *Security Aspects* in the *Special Development Topics*. Value: name of a class that implements the interface `com.softwareag.cis.server.IRequestDataConverter`.
`sessionidasthreadname`	Default: true. On start of each page request processing, the Application Designer runtime calls the method `Thread.setName` with the current session ID (default). You can set this parameter to "false" to instruct the Application Designer runtime not to touch the thread's name. Values: true/false.
`sessiontimeout`	Default: 3600 (1 hour). 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. Value: integer number.

`startmonitoringthread`	Default: true. If set to "true", a monitoring thread is opened which wakes up every 5 seconds. The thread performs the following activities: 1. It initiates a garbage collection periodically (every two minutes). 2. It writes all log information into a log file (every five seconds). 3. It calls the clean up of sessions which are timed out (every two minutes) What happens if the monitoring thread is not started? 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 *web.xml* file, but is written to the servlet container's logging. 3. Timing out sessions is not done every two minutes but every thousand requests. **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". Values: true/false.
`textencoding`	Default: UTF-8. 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. See also *Unicode* in the *Multi Language Management* documentation.
`urlsessiontimeout`	When Application Designer times out a session (see the `sessiontimeout` 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. Value: the URL of the page that is to be shown instead of the default page.
`usemessagepopup`	Default: false. Set this parameter to "true" in order to show status messages as message pop-ups. Values: true/false.

`useowncl assloader`	Default: true. If set to "true", Application Designer uses its own class loader to load application classes. 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. **Caution:** The Application Designer class loader automatically searches for classes in certain directories (*<project>/appclasses/classes* and *<project>/appclasses/lib*). If you do not use the Application Designer class loader, you have to set up your environment accordingly. Values: true/false.
`xmldatamanager`	This parameter defines the file name of the class which implements the `com.softwareag.cis.xmldata.IXMLDataManager` interface. You can specify an own class here. The `com.softwareag.cis.xmldata.XMLDataManagerFactory` creates an instance using a constructor without any parameter.
`zipcontent`	Default: true. 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. Sometimes you may want to actually "see" what is being sent (maybe you have a test tool that captures the HTTP protocol). Set `zipcontent` to "false" if you do not want Application Designer to zip the data content returned to the client. Values: true/false.

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_<xxxxx>.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

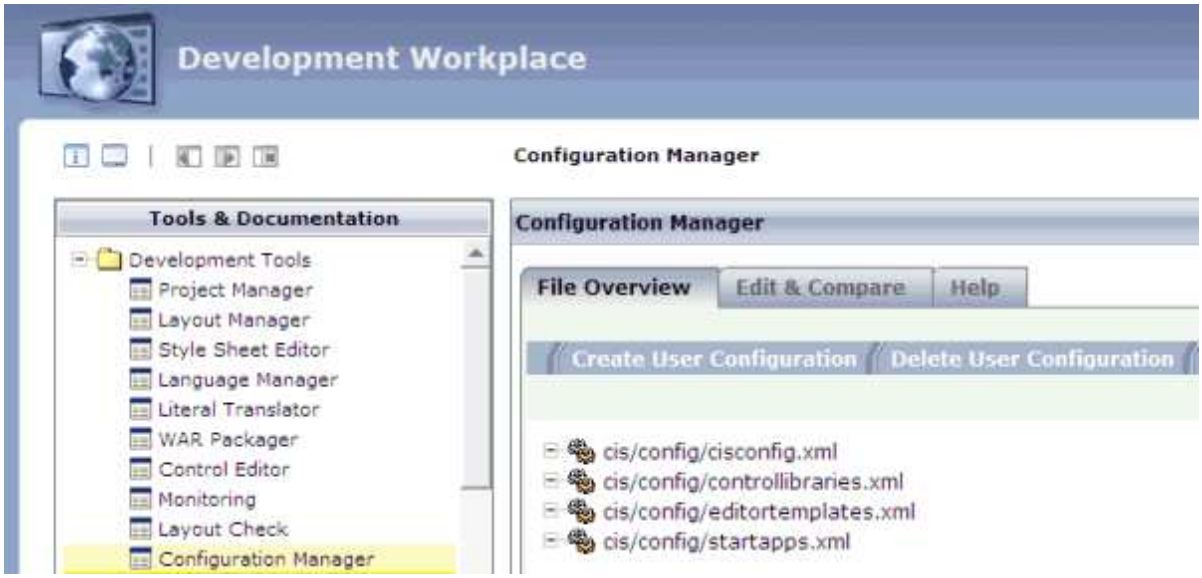
Instead of modifying the default configuration files, it is recommended that you modify the corresponding *user_*.xml* files.

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.