Natural for Ajax

Configuration and Administration

Version 9.3.2

February 2025

ADABAS & NATURAL

This document applies to Natural for Ajax Version 9.3.2 and all subsequent releases.

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

Document ID: NJX-NATNJX-CONFIG-ADMIN-932-20250213

Table of Contents

Configuration and Administration

1 About this Documentation

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
`Monospace font`	Identifies service names and locations in the format *`folder.subfolder.service`*, APIs, Java classes, methods, properties.
`Italic`	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Product Documentation

You can find the product documentation on our documentation website at **https://documentation.softwareag.com**.

Product Training

You can find helpful product training material on our Learning Portal at **https://learn.softwareag.com**.

Tech Community

You can collaborate with Software GmbH experts on our Tech Community website at **https://tech-community.softwareag.com**. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software GmbH news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at **https://github.com/softwareag** and **https://containers.softwareag.com/products** and discover additional Software GmbH resources.

Product Support

Support for Software GmbH products is provided to licensed customers via our Empower Portal at **https://empower.softwareag.com**. Many services on this portal require that you have an account. If you do not yet have one, you can request it at **https://empower.softwareag.com/register**. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

Data Protection

Software GmbH products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

2 About the Logon Page

Starting a Natural Application from the Logon Page

When you start Natural for Ajax in the browser, a logon page appears. The entries in this logon page depend on the settings in your **Session Configuration**.

In order to start a Natural application from the logon page, you enter the following URL inside your browser:

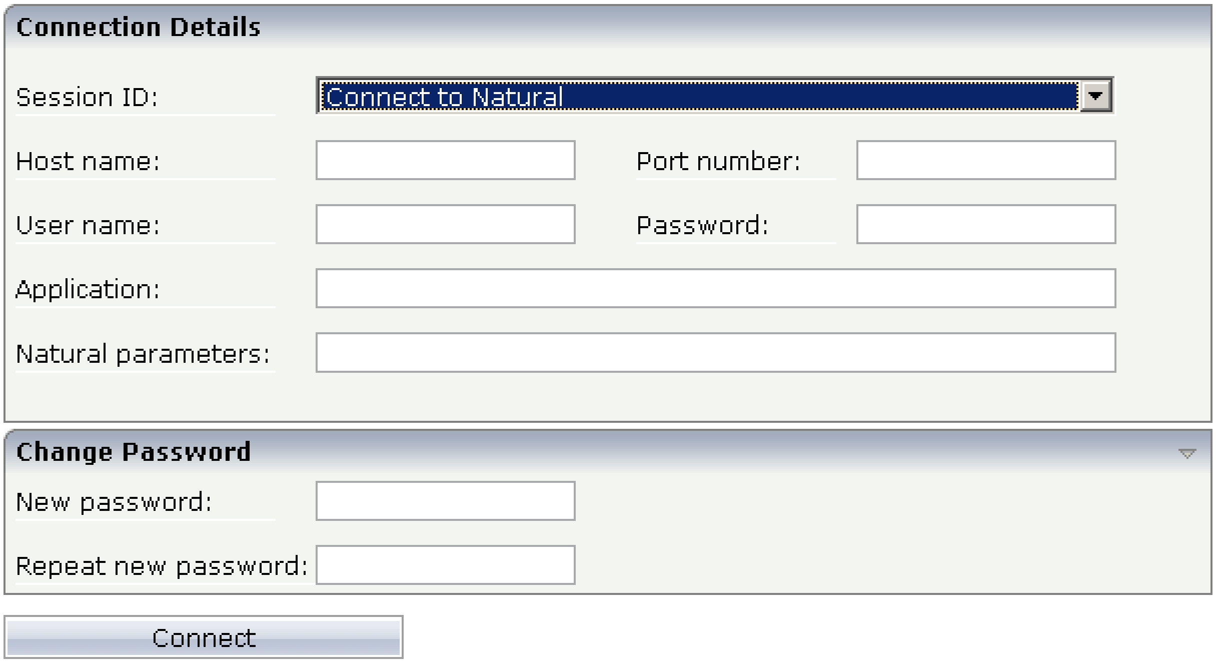
```
http://<host>:<port>/cisnatural/servlet/StartCISPage?PAGEURL=/cisnatural/NatLogon.html
```

where *<host>* and *<port>* are the host name and port number of your application server.

Examples of Logon Pages

For each session definition that has been configured in the configuration file, an entry appears on the logon page. If the user selects the corresponding entry, only those parameters that were not pre-configured in the configuration file need to be specified in the logon page in order to start the application. Usually, you will preconfigure all connection parameters except user name and password.

The following example shows part of a logon page which results from a configuration file in which no special entries are defined for a session:



Connection Details

Session ID: Connect to Natural

Host name:

Port number:

User name:

Password:

Application:

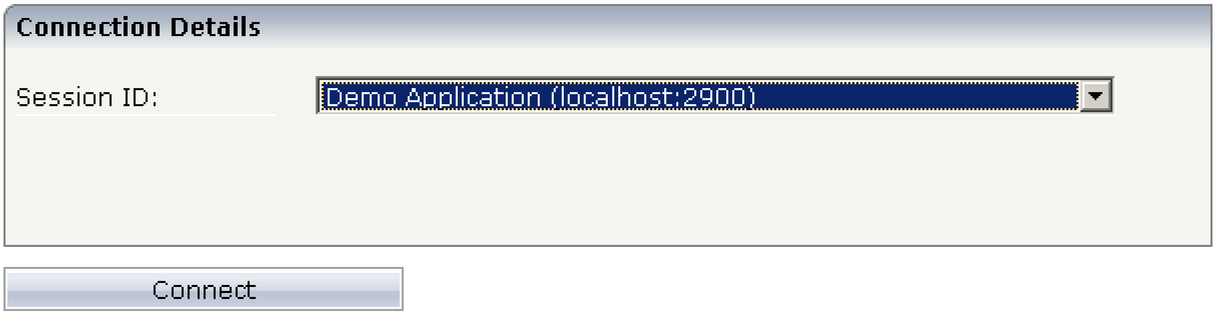
Natural parameters:

Change Password

New password:

Repeat new password:

The following example shows part of a logon page which results from a configuration file in which many settings are already predefined (including user ID and password):



Connection Details

Session ID: Demo Application (localhost:2900)

To log on to a session, you have to specify all required information in the logon page (for example, you select a session from the corresponding drop-down list box). When you choose the **Connect** button, the screen for the selected session appears.

Dynamically Changing the CICS Transaction Name, Host Name, and Port Number when Starting a Session

The following description applies if you want to switch to a different CICS transaction on a mainframe. It is also possible to select a different host and port number.

You specify the dynamic server parameters in the same text box in which you also specify the dynamic parameters for the Natural environment. So that the dynamic server parameters can be evaluated, it is important that you specify it before any Natural parameters, using the following syntax:

```
<TA_NAME=name TA_HOST=hostname TA_PORT=number>
```

name
Can be 1 to 4 characters long. This must be the name of an existing CICS transaction which applies to a CICS Adapter. It will override the transaction name, currently defined in the configuration file for the CICS Adapter on the Natural Web I/O Interface server (NWO). Ask your administrator for further information.

hostname
Can be 1 to 32 characters long. This must be the name, or the TCP/IP address of the host the desired CICS is running. It will override the hostname (RFE_CICS_TA_HOST), currently defined in the configuration file for the CICS Adapter on the Natural Web I/O Interface server (NWO). Ask your administrator for further information.

number
Can be a numeric value in the range from 1 to 65535. It is the TCP/IP port of the CICS-supplied listener. It will override the hostname (RFE_CICS_TA_PORT), currently defined in the config-

uration file for the CICS Adapter on the Natural Web I/O Interface server (NWO). Ask your administrator for further information.

Make sure to put the entire definition in angle brackets. Dynamic server parameters can be separated by a comma or a blank character. Only the parameters to be overwritten must be specified. When this definition is followed by a Natural parameter, insert a blank before the Natural parameter.

Example:

```
<TA_NAME=NA82 TA_HOST=HOST.LOCAL.NET TA_PORT=1898> STACK=(LOGON SYSCP)
```

If one of the dynamic server parameters is incorrect, an error message occurs, and the session cannot be started.

If an incorrect CICS transaction id is specified, the error message `EZY1315E 09/03/24 15:17:46 INVALID TRANID=G PARTNER INET ADDR=ip-address` occurs.

If an incorrect hostname or port number is specified, the error message `NAT9939 Open socket connection failed` occurs.

Note: The above definition of dynamic server parameters can also be specified in the configuration tool, in the same place where you also specify the Natural parameters, and together with the **URL parameter** `xciParameters.natparam`.

Specifying a Password in the Logon Page

The following information applies when the field for entering a password appears on the logon page. This field does not appear when a password has already been defined in the configuration file.

Under Windows and Linux, you always have to enter the operating system password, even if Natural Security is active.

On the mainframe, this is different: When Natural Security is not active, you have to enter the operating system password. When Natural Security is active, you have to enter the Natural Security password.

Changing the Password in the Logon Page

Currently, this functionality is only available for Natural for Linux and Natural for Windows.

The following information applies when the fields for entering a user ID and a password appear on the logon page. These fields do not appear when user ID and password have already been defined in the configuration file; in this case, it is not possible to change the password in the logon page.

When your password has expired, you are automatically asked for a new password. When you try to log on with your current password, an error message appears and input fields for changing the password are shown.

To change the password

1 Click the title **Change Password** to show the content of this input area.

The following two input fields are shown in the logon page:

- **New password**
- **Repeat new password**

2 Enter your user ID and your current password as usual.

3 Enter the new password in the two input fields.

4 Choose the **Connect** button to change the password.

Browser Restrictions

The browser's “Back” and “Forward” buttons do not work with Natural for Ajax and should therefore not be used.

If you want to run two Natural sessions in parallel, you have to start a new instance of the browser (for example, by choosing the corresponding icon in the Quick Launch toolbar of Windows). You must not use the browser's “New Window” function. This would result in one session running in two browsers, which is not allowed.

3 Natural for Ajax Runtime Tools

Natural for Ajax provides tools to configure sessions and the runtime of Natural clients.

The following sections cover the features and functionalies.

Invoking the Natural for Ajax Runtime Tools

The runtime tool are automatically installed when you install Natural for Ajax.

They can be accessed from the default index page. You can also access them via the following URL:

http(s)://<host>:<port>/cisnatural/servlet/StartCISPage?PAGEURL=/njxruntimetools/njxruntimetools-main.html&POPUPTITLE=Natural%20for%20Ajax%20RuntimeTools

Note: You may wish to protect the configuration tool against unauthorized access. See section *Configuring Container-Managed Security* for information on how to restrict the access to sensitive areas of the application server environment. In NaturalONE you can access the Runtime Tools via the Ajax Developer context menu on any Ajax enabled Natural project.

Session Configuration

The following configuration settings and procedures are discussed below:

- Invoking the Session Configuration Tool
- Global Settings
- Adding a New Session
- Editing a Session
- Overview of Session Options
- Duplicating a Session
- Deleting a Session
- Adding a New User

- Saving the Configuration

Invoking the Session Configuration Tool

To invoke this the Session Configuration tool, click link in the left frame.

The Session Configuration tool appears in the right frame. It shows the global settings and lists all sessions and users that are currently defined.

Global Settings

The global settings apply to all defined sessions. The settings are grouped.

Each group has individual settings as listed below:

<table>
<tr><td><b>Trace directory</b></td><td>Optional. Location of a different trace directory.

When a different trace directory is not defined, the trace files are written to the default trace directory. By default, the trace files are written to the directory which has been set by the Java property <code>java.io.tmpdir</code> . On Windows, this is normally the environment variable <code>TMP</code> for the user who started the application server. On Linux, this is normally <i>/tmp</i> or <i>/var/tmp</i> .

You can also set this property in the start script for the application server. The following examples apply to Wildfly.

■ Example for Windows (<i>run.bat</i>):

<code>set JAVA_OPTS=%JAVA_OPTS% -Djava.io.tmpdir=C:\temp</code>

■ Example for Linux (<i>run.sh</i>):

<code>set JAVA_OPTS="$JAVA_OPTS -Djava.io.tmpdir=/tmp</code>

Tracing can be enabled individually for each session (see <i>Overview of Session Options</i> below). However, it should only be enabled when requested by Software AG support.</td></tr>
</table>

Server Trace

Last activity timeout (n seconds)	The number of seconds that the client waits for the next user activity. When the defined number of seconds has been reached without user activity, the session is closed. The default is 3600 seconds.

Session Timeout

Truststore path	Optional. The path to your trust file. See *Configuring SSL* for further information.
Truststore password	If your trust file is password-protected, you have to specify the appropriate password. When you do not specify the password for a password-protected trust file, the trust file cannot be opened and it is thus not possible to open an SSL session. When your trust file is not password-protected, you should not specify a password.

SSL/TLS

Client id	The client identifier of your client at your OpenID Provider. This identifier is assigned to you when you register your client with your OpenID Provider.
Client secret	The client secret only known to your client and your OpenID Provider.
Natural user claim	The name of the claim in the token, which provides the value of the Natural username.
Natural user – remove email suffix	If email is specified as Natural user claim then the e-mail suffix is removed and only the remaining part is used as username for Natural.
Request claims using scope	Additional claims are requested from the provider via the scope parameter and integrated into the token.
Request claims using the claims request parameter	Additional custom claims are requested from the provider via the claims request parameter and integrated into the token.
Authorization endpoint	The OpenID Provider server endpoint where the user is asked to authenticate and grant the client access.
Token endpoint	The client exchanges the code received from the authorisation endpoint for a token.
Logout endpoint	A client can use the logout endpoint to clear the provider-side session and cookies for a web browser. If used, you will need to log in at the OpenID Provider again.

OpenID Connect

Use secure logon page	If selected, users are authenticated before they access the Natural for Ajax logon page. See also *Configuring Application-Managed Authentication* . If **Use SAML for JAAS-based authentication** is also selected, the secure logon page is only shown in the case of an error in the SAML artifact validation.

JAAS

Adding a New Session

You can add a new session to the configuration file.

To add a new session

1. Choose the **Create** button in the Sessions panel.

   The **Edit Session** page appears.
2. Specify all required information as described below in the section *Overview of Session Options*.
3. Choose the **SAVE** button.

   The new session is written to the configuration file, and you are returned to the **Session Configuration** page.

Editing a Session

You can edit any existing session in the configuration file.

To edit a session

1. Choose the **Edit** icon for the session that you want to edit in the session table.

   The **Edit Session** page appears.
2. Specify all required information as described below in the section *Overview of Session Options*.
3. Choose the **SAVE** button.

   The new session is written to the configuration file, and you are returned to the **Session Configuration** page.

Overview of Session Options

The Edit Session page appears when you

- add a new session, or
- edit an existing session.

The individual options are grouped under tabs. The following options are provided:

Session ID	Mandatory. A session name of your choice. On the logon page, the session name is provided in a drop-down list box.
Application name	Optional. An application name of your choice. This name is used to provide more individual disconnect messages. The name entered here will replace the term "Application" in the standard disconnect message of Natural for Ajax applications. For example, if the field **Application name** contains the word "Calculator" the disconnect message "Application finished successfully" would instead be "Calculator finished successfully" . Similar custom messages will be displayed if the application ends with an error.

Disconnect behavior	Defines whether the default disconnect page is to be shown when a Natural program ends, or whether the current browser tab is to be closed. You can select the required setting from the drop-down list box.
	- **Display disconnect page**
	Default. The disconnect page is always displayed.
	- **Always close browser**
	The current browser tab is closed. When the last tab in the browser is closed, the entire browser is closed.
	- **Close browser, but display disconnect page on error**
	The current browser tab is closed. When the last tab in the browser is closed, the entire browser is closed. However, in the case of an unexpected error, the disconnect page is shown.
	Internet Explorer allows closing the browser with JavaScript by default. For Firefox, however, this has to activated as follows:
	1. In the Firefox address bar, type the following:
	`about:config`
	A warning message appears, saying that changing the advanced settings can be harmful and that you should only continue if you are sure of what you are doing.
	2. Choose the **I'll be careful, I promise!** button.
	3. In the **Filter** text box, type the following:
	`dom.allow_scripts_to_close_windows`
	4. Set this value to "true" by double-clicking on the entry (default: "false").

Trace	Should only be set to **Yes** when requested by Software AG support.
Timeout (in seconds)	The number of seconds that the client waits for a response after an updated page was sent to the Natural session. When the defined number of seconds has been reached without response, the session is closed. The default is 60 seconds. Normally, you need not change this value.

General

Host name	The name or TCP/IP address of the server on which Natural and the Natural Web I/O Interface server are running. When this is specified, the corresponding field does not appear on the logon page.
Port number	The TCP/IP port number on which the Natural Web I/O Interface server is listening. When this is specified, the corresponding field does not appear on the logon page.
Use SSL	If set to **Yes** , a secure connection is established between Natural for Ajax on the application server and the Natural Web I/O Interface server. **Important:** If you want to use SSL with Natural for Mainframes, one of the corresponding mainframe types must be selected; the type must not be **Undefined** or **Natural for Windows or Linux** . The other way around, if you want to use SSL with Natural for Windows or Linux, you must not select one of the mainframe types; the type may also be **Undefined** in this case.

Natural Connection

Natural Server Type	The platform on which user ID and password are authenticated. You can select the required setting from the drop-down list box. ■ **Undefined** Default. User ID and password can have a maximum of 32 characters. See also the description for Natural for Windows or Linux below. ■ **Natural for Mainframes** User ID and password can have a maximum of 8 characters. ■ **Natural for Mainframes with Natural Security** User ID and password can have a maximum of 8 characters. The user ID must comply with the Natural naming conventions for library names . ■ **Natural for Windows or Linux** User ID and password can have a maximum of 32 characters. When a domain is required, you have to specify it together with the user ID (in the form " *domain* \ *user-ID* ").
OpenID Connect (OIDC)	Use OIDC for authentication to Natural. Be sure that you have filled in the mandatory OIDC settings of the global session settings.
JAAS-based authentication	If selected, application-managed authentication is used. This setting implies that **Forward credentials** is also selected. See also *Configuring Application Managed Authentication* . If **Use SAML for JAAS-based authentication** is also selected in the global settings, the user name and password are not visible in the Natural for Ajax logon page. See also *Using Software AG Security Infrastructure* .
Forward credentials	If selected, the security credentials from the application server are forwarded to the Natural Web I/O Interface server, thus sparing the user a second logon. See also *Forwarding the User Credentials to Natural* .
User name	Optional. A valid user ID for the current machine. When this is specified, the corresponding field does not appear on the logon page.

User name in upper case	If selected, the input field for the user ID is in upper-case mode.
Password	Optional. A valid password for the above user ID. Under Windows and Linux, this is always the operating system password of the user, even if Natural Security is active. On the mainframe, this is different: When Natural Security is not active, this is the operating system password of the user. When Natural Security is active, this is the Natural Security password. When a password is specified, the corresponding field does not appear on the logon page. The configuration tool saves the password in encrypted form.
Save user credentials	Applies only to applications that are designed as Application Designer workplaces. If set to **Yes** (default), the default behavior of the option **Share session user** applies. If set to **No** , the user credentials (user ID and password) are not saved in the Application Designer session and are therefore not available for an Application Designer subsession. An example for a workplace application is available under the following URL: *http://* `<host>:<port>` *\/cisnatural/servlet/StartCISPage?PAGEURL=/njxdemos/wpworkplace.html* where `<host>` and `<port>` are the host name and port number of your application server.
Share session user	Applies only to applications that are designed as Application Designer workplaces. If set to **No** (default), the user credentials of the main Application Designer session are automatically used in an Application Designer subsession if the server and port of the subsession is the same as in the main session. If the server and port are not the same, the user has to specify the user ID and password in a logon dialog. If set to **Yes** , the user credentials of the Application Designer main session are always used for all Application Designer subsessions on all involved servers - even if the server and port are different.

Authentication

Application	■ **Natural for Mainframes** The name of the Natural program or a command sequence that starts your application as you would enter it on the `NEXT` prompt. Example: `TEST01 data1,data2`

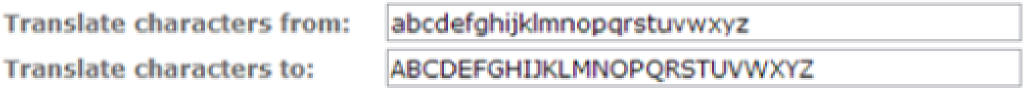
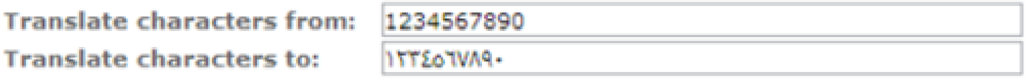
	■ **Natural for Linux** The name of the Linux shell script for starting the Natural application (a file similar to *nwo.sh*). ■ **Natural for Windows** The name of the Windows command file (*.bat*) for starting the Natural application. When this is specified, the corresponding field does not appear on the logon page.

Natural parameters	Optional. Parameters for starting the Natural application. This can be stack parameters, a parameter file/module or other Natural-specific information.
	■ **Natural for Mainframes**
	Used to pass dynamic Natural profile parameters to the session, for example:
	`SYSPARM=(MYPARMS) STACK=(LOGON MYAPPL)`
	Note: It is recommended to specify the Natural program that starts the application with the option **Application** instead of passing it with the profile parameter `STACK` .
	■ **Natural for Linux and Natural for Windows**
	Used when the above shell script (Linux) or command file (Windows) uses the parameter `$5` after "natural" , for example:
	`PARM=MYPARM STACK=(LOGON MYLIB;MENU)`

Natural Application

Language	You can select the required language from the drop-down list box. It is also possible to select **Set in workplace** from the drop-down list box. When selected, the language that is currently used in the workplace will also be used for the next start of Natural. See also *Multi-Language Management* . Default: English.
Show style sheet selector	With Natural for Ajax, the users can switch to another style sheet during a running session. If set to **No** , the users are no longer able to select another style sheet.
Style sheet	The name of the style sheet which determines the colors, fonts and PF key button style of the current session. See *Using Style Sheets* . When this element is specified, a fixed style sheet is used. In this case, the corresponding field does not appear on the logon page and the user is thus not able to select a different style sheet.
Double-click behavior	Applies only to Natural maps, not to rich GUI pages. The key that is to be simulated when double-clicking an output field. By default, this is the ENTER key. It is possible to disable the double-click behavior, or to define a function key (PF1 through PF12). You can select the required setting from the drop-down list box.

	Tip: When context-sensitive help has been defined for the output fields, it may be useful to define PF1 . The help function will then be invoked when the user double-clicks an output field.
PF keys display style	Applies only to Natural maps, not to rich GUI pages. By default, only the named function keys are shown as buttons. It is also possible to show buttons for all function keys, including those which do not have names. You can specify whether to display buttons for 12, 24, 36 or 48 function keys. Each line always contains 12 function key buttons. The first line also contains a button for the ENTER key. Each function key button is always displayed at the same position. You can select the required setting from the drop-down list box.
Screen rows	Applies only to Natural maps, not to rich GUI pages. The number of rows in the output window. Possible values: minimum 24, no upper limit. Default: 24. Not used by Natural for Mainframes which uses the profile parameter `TMODEL` instead.
Screen columns	Applies only to Natural maps, not to rich GUI pages. The number of columns in the output window. Possible values: minimum 80, no upper limit. Default: 80. Not used by Natural for Mainframes which uses the profile parameter `TMODEL` instead.
Show function key numbers	Applies only to Natural maps, not to rich GUI pages. If set to **Yes** , the PF key numbers are shown next to the PF keys.
Check for numeric input	If set to **Yes** (default), numeric input fields are validated. In this case, only the following characters are allowed in numeric input fields (in addition to the numbers "0" through "9"): *`blank`* + (plus) - (minus) _ (underscore) , (comma . (period) ? (question mark) If set to **No** , numeric input fields are not validated.
Auto skip	Applies only to Natural maps, not to rich GUI pages. If set to **Yes** , the cursor is automatically placed in the next input field when the last possible character has been entered in the current input field. If set to **No** (default), the cursor remains in the current input field.

Translate characters from / Translate characters to	Applies only to Natural maps, not to rich GUI pages. Optional. Used to translate characters that are sent from the host or entered by the user into different characters to be shown on the screen. You enter the characters to be translated in the **Translate characters from** text box and the characters to be used instead in the **Translate characters to** text box (in the same order as in the first text box). Both text boxes must contain the same amount of characters. The following example shows how to translate all lower-case characters to upper-case characters:  Translate characters from: abcdefghijklmnopqrstuvwxyz Translate characters to: ABCDEFGHIJKLMNOPQRSTUVWXYZ The following example shows how to translate the so-called “European numerals” to “Arabic-Indic numerals” :  Translate characters from: 1234567890 Translate characters to: ١٢٣٤٥٦٧٨٩٠
Filler character	Applies only to Natural maps, not to rich GUI pages. Optional. The filler character that is to be removed from the input fields. An application can define, for example, an underscore (_) as the filler character. Trailing filler characters will be removed from the input fields, and leading filler characters will be replaced with blanks.

Web I/O Pages

Duplicating a Session

You can add a copy of any existing session to the configuration file.

To duplicate a session

1 Choose the **Duplicate** icon that is shown next to the session that you want to duplicate.

A new entry bearing the name `Copy of session-ID` appears at the bottom of the list of sessions. However, this duplicate session is not yet available in the configuration file.

2 Edit and save the duplicated session as described above.

Deleting a Session

You can delete any existing session from the configuration file.

To delete a session

1 Choose the **Delete** icon that is shown next to the session that you want to delete.

The session is deleted from the list of sessions. However, it is not yet deleted in the configuration file.

2 Choose the **Save** button to delete the session from the configuration file.

Adding a New User

You can add predefined Natural users and their passwords in the configuration file.

When a Natural page is opened with a URL that specifies a user in the URL parameter `natuser`, the specified user is matched against the list of users in the configuration file. If the specified user is defined in the configuration file, the corresponding password is used to authenticate the user when the Natural session is started. See also ***Starting a Natural Application with a URL***.

Example

The following URL applies the password as defined for `user1`:

http://myhost:8080/cisnatural/servlet/StartCISPage?PAGEURL=/cisnatural/NatLogon.html&xciParameters.natuser=user1 ...

To add a new user

1 Choose the **Create** button in the Users panel

The **Create User** dialog appears.

2 Specify a username and password.

Choose the **SAVE** button to write the new user to the configuration file.

Note: You edit, duplicate and delete a user in the same way as a session (see the corresponding descriptions above).

Saving the Configuration

When you choose the **Save** button, all of your changes are written to the configuration file. The server picks up the new settings automatically the next time it reads data from the configuration file.

Caution: The new settings are NOT written to the configuration file if you do not select **Save** before you log out or leave the configuration tool for another URL.

Natural Client Logging Configuration

Logging is managed from the ***Logging Configuration*** page.

On this page, you can determine the settings recorded in the configuration file for logging, *natlogger.xml*.

Ajax Configuration

Opens the Ajax Configuration Editor.

The Ajax Configuration settings are described in ***Ajax Configuration***.

Monitoring Tool

- About the Monitoring Tool
- Invoking the Monitoring Tool
- Java Memory
- JavaScript
- Ajax Server Log File
- System Management (Caches)

- Active Ajax Sessions

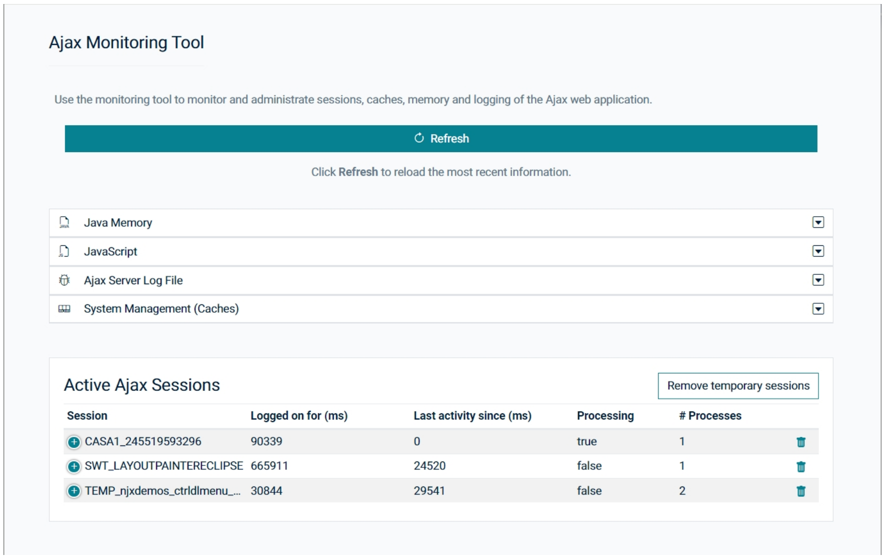
About the Monitoring Tool

The monitoring tool is used to monitor the sessions that are currently running in the deployed web application.

You refresh internally used cached sessions or delete sessions, which are no longer required. This way you can free memory. For more information on sessions, see *Details on Session Management*.

Invoking the Monitoring Tool

The monitoring tool is part of the Runtime Tools. When you invoke the monitoring tool, the following editor appears:



Ajax Monitoring Tool

Use the monitoring tool to monitor and administrate sessions, caches, memory and logging of the Ajax web application.

Click **Refresh** to reload the most recent information.

Session	Logged on for (ms)	Last activity since (ms)	Processing	# Processes
CASA1_245519593296	90339	0	true	1
SWT_LAYOUTPAINTERECLIPSE	665911	24520	false	1
TEMP_njxdemos_ctrldlmenu_...	30844	29541	false	2

The information and functions are grouped in the following way:

- Java Memory
- JavaScript
- Ajax Server Log File
- System Management (Caches)
- Active Ajax Sessions

Using the **Refresh** button at the very top, you can reload the information in the editor and show the most recent information.

To invoke the monitoring tool in a deployed web application

- Click the button **Runtime Tools** of the default index page. This opens the Runtime Tools.

To invoke the monitoring tool in NaturalONE

1. In the **ProjectExplorer** view, select the project for which you want to invoke the monitoring tool.
2. Invoke the context menu and from the **Ajax Developer** menu, choose **Ajax Runtime Administration**. This opens the Runtime Tools.

Java Memory

The top of the editor provides the following information:

Option	Description
Used memory	This text box indicates the currently used memory.
Process memory	This text box indicates the available memory.
Collect garbage	When you choose this command button, memory which is no longer used on the server is set free.

JavaScript

You can use this section to dynamically switch on or off the writing of JavaScript logging to the browser console.

Ajax Server Log File

The server log file contains a protocol of all interactions with the application server/web container. You switch **Debug Mode** on or off. In debug mode, the log contains more detailed information.

When you choose the **Open Today's Log File** command button, a dialog appears showing the content of the log file.

System Management (Caches)

You can use this section to clear or refresh specific internal caches.

Option	Description
Class Loader	When you choose the **Use latest Version of Applications for new Sessions** command button, a new class loader instance is generated. The most recent versions of your applications are then loaded. For example, when an adapter class has been modified, the modified class is loaded and any changes are immediately in effect. See also *Class Loader Concepts*.
Text Buffer	When you choose the **Refresh Text Buffer** command button, the most recent versions of your language files are loaded and any changes are immediately in effect. When you choose the **Refresh Layout Repository Buffer** command button, the most recent versions of your layouts are loaded and any changes are immediately in effect.

Active Ajax Sessions

This section provides a list with all sessions, which are currently active.

The monitoring process itself is also considered as a session. When no other sessions are active, the monitoring process is the only session that is listed in the editor.

The following command buttons and icons are available:

Command Button/Icon	Description
Remove	The trashcan icon next to each active session. When you click this icon, the selected session is removed from the list of active sessions.
Remove temporary sessions	When you click this button, all temporary sessions, for instance Layout Painter Preview sessions, are removed from the list of active sessions. Temporary sessions start with “TEMP”.

4 Ajax Configuration

The following topics are covered below:

Ajax Configuration Editor

The Ajax Configuration Editor is part of the *Natural for Ajax Runtime Tools*. It's available in every Natural for Ajax web archive. When started, it opens the active ajax configuration file *user_cisconfig.xml*. If the configuration file doesn't exist, it opens the *cisconfig.xml* file in the <subfolder>/cis/config.

In NaturalONE, to open a *cisconfig.xml* file of your User Interface Component with the Ajax Configuraton Editor: Right-click on the *cisconfig.xml* file and choose **Open with** > **Ajax Configuration Editor**.

General cisconfig.xml 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>
```

Control Behavior – All Controlsets

Parameter	Description
`buttonctrlenter`	Default false. If set to true <ctrl><enter> on a focused button will trigger the button method.
`completedateinput`	Default: true. By default, partial input in the `DATEINPUT` control is automatically completed. When you set this parameter to "false", no automatic completion will be done, thus forcing end-users to always enter the complete date. Values: true/false.
`completedateinputrange`	Default: 999 If the setting `completedateinput` is set to false, the `completedateinputrange` is ignored. Valid values: 999 : Compatibility with versions before 932. For a two-digit year value 00 - 49, the current century is set. For a two-digit year value 50 - 99, the previous century is set. 0 : For a two-digit year value, the current century is set. 1 : 99 Same behavior as the Natural `YSLW` profile parameter. For details, see *Parameter Reference* > *YSLW - Year Sliding or Fixed Window* in the Natural documentation. 1582-2600 : Same behavior as the Natural YSLW profile parameter. For details, see *Parameter Reference* > *YSLW - Year Sliding or Fixed Window* in the Natural documentation.
`customdates`	You can specify several `customdates` (*`YYYY-MM-DD`*) separated by a semi-colon. Entering a custom date in the browser doesn't display a validation error.
`displayallowtab`	Default: true Defines if tabbing into DISPLAY input controls is possible.
`accessibilityroles`	Default: true Defines for controls and container that corresponding role attributes for accessibility are generated.
`fieldnumerictypesrightaligned`	Default: false.

Parameter	Description
	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.
`flushreceivespreviousfocused`	Default: false. By default, during a flush event the adapter gets as focus information the input control that *received* the focus. Set this parameter to "true" if during a flush event your application relies on getting as focus information the input control that *lost* the focus. 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. Values: true/false.
`suppressfocusmanagement`	Default: false. 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. 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 `focusmgtprop` in the NATPAGE control. Only set this parameter to "true" if your application needs to do it vice versa: Suppress focus mangement for nearly all server round trips and only explicitly activate focus management for some specific server round trips from within your adapter code. Values: true/false.

Control Behavior – Non-Responsive Controlset

Parameter	Description
`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.

Parameter	Description
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.
takeoutfieldpopupicon	Default: false. 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. Values: true/false.
valuehelpkeys	You can specify your own keys to open the value help pop-up and/or combo box in a FIELD control. The keys are specified in the same way as hot keys. Example: `valuehelpkeys = "ctrl-65;ctrl-alt-66"`
resetstatusbarbefore	Default: false. When set to true, the status bar messages are reset in the browser before a server roundtrip is done. Values: true/false.
resetstatuspropfocus	Default: false. If set to "true", statusprop values FOCUS and FOCUS_NO_SELECT will automatically be reset to the default value; statusprop value ERROR will be reset to ERROR_NO_FOCUS. The reset is done before the next server round-trip to Natural, so that the Natural program will receive the changed value. This way, the focus is set only once without having to actively reset the statusprop value in the next server round-trip. The Natural for Ajax runtime will take care of the reset.
clientsideerrorinstatusbar	Default: false. By default, client-side error messages are displayed as pop-ups. When you set this parameter to "true", client-side error messages are displayed in the status bar. Values: true/false.
usemessagepopup	Default: false.

Parameter	Description
	Set this parameter to "true" in order to show status messages as message pop-ups. Values: true/false.
`itrinlinedisplay`	Only set this if you notice rounding issues with pixel-sizing in ITRs while zooming the page in Google Chrome and/or Edge Chromium.

Logging and Error Handling

Parameter	Description
`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.
`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.
`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.

Parameter	Description
	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.
`loglevel`	Default: EWI. Defines the message types that are to be logged. Values: E (error) W (warning) I (information) D (debug) N (no logging) 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. Specify "N" (no logging) to switch off writing log messages to a logfile. **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.
`maxserverlogage`	Default: -1 (log files are not automatically deleted). When setting `maxserverlogage` to a value > 0, *ServerLog*.log* files, which are older than the set number of days, are automatically deleted. For example, if `maxserverlogage` is set to 3, all *ServerLog*.log* files, which are older than 3 days are automatically deleted. If `startmonitoringthread` is set to false, this parameter has no effect.
`monitoringthreadinterval`	Default: 5000. The interval in milliseconds for the wake-up of the monitoring thread. If `startmonitoringthread` is set to false, this parameter has no effect.
`startmonitoringthread`	Default: true.

Parameter	Description
	If set to "true", a monitoring thread is opened which by default wakes up every 5 seconds. You can customize this value by setting the parameter `monitoringthreadinterval`. 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 *n* milliseconds. Where *n* represents the interval length defined in the `monitoringthreadinterval` parameter). 3. It calls the clean up of sessions which are timed out (every two minutes) 4. It checks for user interface component updates, which need to be deployed. 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. 4. No user interface deployment will be done. **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.
`licwarningsfor`	Semicolon seperated list of hostnames. When the web application is called with an URL containing one of these hostnames and the license is in the expiration period of 40 days, an alert box with a license warning is shown once per day.

Sessions, Buffers and Caches

Parameter	Description
`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.

Parameter	Description
`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.
`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.
`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.
`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.
`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.
`maxworkplaceactivities`	Default: -1 (unlimited). The maximum number of workplace activities in a workplace application.

Styles, Languages and Encoding - All Pages

Parameter	Description
`defaultcss`	You can set your own default style sheet for your entire application. For example: `../cis/styles/MY_STYLE.css`
`defaultlanguage`	Default: en (English). Defines the language that is to be used by default when starting Application Designer. If not set, "en" is used.
`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. 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.
`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.

Styles, Languages and Encoding - Responsive Pages

Parameter	Description
`uselatestbootstrap`	If set to true Bootstrap 4 is used. If set to false Bootstrap 3 is used. When changing this setting you need to regenerate you page layouts. Default: true. Bootstrap 4 is used. Values: true/false.
`defaultbmobilecss`	Set your Natural for Ajax specific responsive stylesheet for the whole application (Bootstrap 3). See *Styling a Responsive Page* .

Parameter	Description
`defaultbmobilecss4`	Set your Natural for Ajax specific responsive stylesheet for the whole application (Bootstrap 4). See *Styling a Responsive Page* .
`defaultresponsivecss`	Set your Bootstrap stylesheet for the whole application (Bootstrap 3). See *Styling a Responsive Page* .
`defaultresponsivecss`	Set your Bootstrap stylesheet for the whole application (Bootstrap 4). See *Styling a Responsive Page* .
`defaultresponsivecss`	Set your DataTables Theme (Bootstrap 3). See *Styling a Responsive Page* .
`defaultdatatablescss4`	Set your DataTables Theme (Bootstrap 4). See *Styling a Responsive Page* .

Page Behavior - Non-Responsive Pop-Ups

Parameter	Description
`usepagepopup`	Default: false. Set this parameter to "true" in order to open Natural for Ajax pop-ups as page pop-ups instead of browser pop-ups. Values: true/false.
`pagepopupenterhotkey`	Default: false. By default, the `reactOnPagePopupEnterKey` event is not triggered when ENTER is pressed in the page pop-up. When setting this parameter to "true", the event reactOnPagePopupEnterKey is triggered when ENTER is pressed in the page pop-up. This event can be processed in the Natural program. Values: true/false.
`pagepopuphorizontal`	Use this to automatically adapt the size of a page pop-up if it does not fit to its parent because the parent width is not big enough. Supported values are "zoom" and "resize". If set to "resize" the width of a page pop-up is reduced. If set to "zoom" the page pop-up will automatically be zoomed in. Values: resize/zoom.
`pagepopupvertical`	Use this to automatically adapt the size of a page pop-up if it does not fit to its parent because the parent height is not big enough. Supported values are "zoom" and "resize". If set to "resize" the height of a page pop-up is reduced. If set to "zoom" the page pop-up will automatically be zoomed in.

Parameter	Description
	Values: resize/zoom.
pagepopuponresize	Default: false. If set to true an open page pop-up is resized when it's parent is resized. Values: true/false.
popupparentdisabled	Default: false. When setting this parameter to "true", the parent page of a page pop-up is rendered disabled while the pop-up is open. It only applies to page pop-ups. Values: true/false.

Page Behavior - Natural

Parameter	Description
natuppercase	Default: false. 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. **Important:** Set the parameter `natuppercase="true"` *before* you implement your main program with Natural for Ajax. If you set this parameter after the implemention, you will have to change all Latin lower-case characters to upper-case manually. Values: true/false.
notifyparentonpopupclosed	Default: true. If this parameter is set to "false", no `nat:page.default` will be sent to the parent Natural program after a pop-up is closed. Otherwise the parent Natural program will receive a `nat:page.default` event after a pop-up is closed. Values: true/false.

Special Functions

Parameter	Description
`urlopenstreetmapgeocoder`	URL used to access the third party geocoder for the OPENSTREETMAP controls. You usually do not have to specify this parameter. However, if the URL of the server of this third party geocoder changes, you can adapt the URL here correspondingly.
`urlbackbuttonpressed`	When the browser back button is pressed, in some cases the page is not synchronized with the server anymore and the session has to be closed. In these cases a default page is displayed. Instead of this default page you can define a URL to a custom page. Value: the URL of the page that is to be shown instead of the default page.
`reloadpageonbackbutton`	Default: false. If set to true, the Ajax framework tries to reload the page when the back button is pressed. A corresponding message box is displayed to inform the end-user about the reload.
`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. Value: name of a class that implements the interface `com.softwareag.cis.server.IRequestDataConverter`.
`sdofullpath`	Default: false The default setting enables the product's XSLT processor implementation. If switched to true, the 3rd party Xalan implementation is used instead of the product's functionality. Should you decide to use Xalan, download Xalan 2.7.2 from the Apache download sites. **Note:** Xalan 2.7.2 contains a vulnerable.

Parameter	Description
`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.
`workplacehotkeys`	You can specify hot keys with which you can switch back and forth between the activities in a workplace. The first entry defines the key for forward switching and the second entry defines the key for backward switching. The following example defines CTRL page up and CTRL page down as corresponding hotkeys: `workplacehotkeys = "ctrl-34;ctrl-33"`

Developing and Testing

Parameter	Description
`htmlgeneratorlog`	Defaut: false. By default *.protocol* files are written during the HTML generation. If set to "true", an additional *.log* file is written for each layout. Only set this to "true", if you cannot resolve generation errors via the Layout Painter error marking. It reduces generation performance.
`designtimeclassloader`	By default, Application Designer uses an own class loader for accessing adapter classes at design time. (You can switch this off by specifying `useownclassloader="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.
`useownclassloader`	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.

Parameter	Description
	Values: true/false.
`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.
`testtoolidhtml4`	Default: false. If set to "true", the HTML attribute generated for the test tool IDs has the name "testtoolid". Otherwise, the name is "data-testtoolid". See also the information on standards mode and HTML5 in the Natural for Ajax documentation.

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>
```

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.

5 Design Time Mode and Runtime Mode

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 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. Both, "own" and standard runtime class loaders 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.

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”.

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.

6 Natural Web I/O Style Sheets

Name and Location of the Style Sheets

Several aspects on a page (such as font, font style or color) are controlled by a style sheet (CSS file).

Natural for Ajax is delivered with a number of predefined style sheets. The default style sheet is *natural.css* . If you would like to see the same colors in the output window as in the map editor, you can use the style sheet *natural_mapeditor.css* instead of the default style sheet.

The location of the style sheet s depends on the application server that you are using.

Note: For more information on style sheets, see ***http://www.w3.org/Style/CSS/*** .

Editing the Style Sheets

It is recommended that you have a basic understanding of CSS files.

You can edit the predefined style sheets or create your own style sheets.

It is recommended that you work with backup copies. When a problem occurs with your style sheet, you can thus always revert to the original state.

To see your changes in the browser, you have to

1. delete the browser's cache, and
2. restart the session.

Modifying the Position of the Main Output and of the PF Keys

Applies when only the named PF keys are displayed. This feature cannot be used when all PF keys are displayed, since they are always displayed at the same position. See also ***Overview of Session Options*** .

The following elements are available:

Element Name	Description
`.mainlayer`	Controls the position of the main output in the output window. Used for languages that are written from left-to-right (LTR).
`.mainlayer_rtl`	Controls the position of the main output in the output window. Used for languages that are written from right-to-left (RTL).
`.pfkeydiv`	Controls the position of the PF keys in the output window. Used for languages that are written from left-to-right (LTR).
`.pfkeydiv_rtl`	Controls the position of the PF keys in the output window. Used for languages that are written from right-to-left (RTL).

The `*_rtl` elements are only used if Natural sends the web I/O screen with a right-to-left flag (`SET CONTROL 'VON'`). In the browser, the screen elements are then shown on the right side (instead of the left side).

For web I/O in applications where only the left-to-right orientation is used, the `*_rtl` elements are not required.

If the PF keys are to appear at the bottom, define the elements as shown in the following example:

```
/* Defines the main screen position */ .mainlayer { top: 5px;
     left: 0px;
     height: 550px; } /* Defines the main screen position for right-to-left */ ↩
.mainlayer_rtl{ top: 5px;
     right: 30px;
    height: 550px; } /* Defines the PF keys screen position */ .pfkeydiv { height: ↩
70px;
     left: 0px;
     top: 580px; width: 100%; } /* Defines the PF keys screen position for ↩
right-to-left */ .pfkeydiv_rtl { height: 70px;
     right: 30px;
     top: 580px; width: 100%; }
```

Modifying the Font Size

Depending on the screen resolution, one of the following style sheets for defining the font size is used in addition to the default style sheet:

- *model2.css*
- *model3.css*
- *model4.css*
- *model5.css*

These style sheets are located in the *tmodels* subdirectory of the *resources* directory in which all style sheets are located.

Depending on what comes closest to the standard 3270 screen model, the corresponding style sheet from the *tmodels* subdirectory is automatically used. It is selected according to the following criteria:

Standard 3270 Screen Model	Criteria	Style Sheet
Model 2 (80x24)	30 rows or less.	*model2.css*
Model 3 (80x32)	Between 31 and 40 rows.	*model3.css*
Model 4 (80x43)	41 rows or more.	*model4.css*
Model 5 (132x27)	30 rows or less, and more than 100 columns.	*model5.css*

The font sizes in the above style sheets can be adjusted. Example for *model4.css* :

```
body { font-size: 10px; }
```

The default font sizes for the above 3270 screen models are:

Standard 3270 Screen Model	Default Font Size
Model 2	16px
Model 3	14px
Model 4	10px
Model 5	12px

Modifying the Font Type

As a rule, you should only use monospace fonts such as Courier New or Lucida Console. With these fonts, all characters have the same width. Otherwise, when using variable-width fonts, the output will appear deformed.

If you want to define a different font type, you should define the same font type for the body, the output fields and the input fields as shown in the following example:

```
body {
    background-color: #F3F5F0;
    font-family: Lucida Console;
    }

.OutputField {
    white-space:pre;
    border-width:0;
    font-family: Lucida Console;
    font-size: 100%;
}

.InputField {
    background-color: white;
    font-family: Lucida Console;
    border-width: 1px;
    font-size: 100%;
    border-color: #A7A9AB;
}
```

Use the CSS at-rule `@font-face` to specify a custom font with which to display text.

For example, Arabic fonts to be used with `InputFields` for the presentation of "Arabic-Indic numerals" instead of "European numerals":

```
@font-face {
font-family: CustomFont;
src: url( CustomFont.woff );
}
```

Defining Underlined and Blinking Text

The following elements are available:

Element Name	Description
`.natTextDecoUnderline`	Defines underlined text.
`.natTextDecoBlinking`	Defines blinking text.
`.natTextDecoNormal`	Defines normal text (no underline, no blinking).

Example:

```
/* Text decoration */
.natTextDecoUnderline { text-decoration:underline; }
.natTextDecoBlinking {text-decoration:blink; }
.natTextDecoNormal {text-decoration:normal;}
```

Blinking text is not supported by the Internet Explorer.

Defining Italic Text

The following elements are available:

Element Name	Description
`.natFontStyleItalic`	Defines italic text.
`.natFontStyleNormal`	Defines normal text (no italics).

Example:

```
/* font style */
.natFontStyleItalic {font-style:italic;}
.natFontStyleNormal {font-style:normal;}
```

Defining Bold Text

The following elements are available:

Element Name	Description
`.natFontWeightBold`	Defines bold text.
`.natFontWeightNormal`	Defines normal text (not bold).

```
/* Font weight */
.natFontWeightBold {font-weight:bolder;}
.natFontWeightNormal {font-weight:normal;}
```

When you define bold text (`{font-weight:bolder;}`) for the default font Courier New, your text always has the same width as with normal text (`{font-weight:normal;}`).

However, when you define bold text for Courier or Lucida Console, the bold text will be wider than the normal text and your output may thus appear deformed. It is therefore recommended that you switch off bold text for Courier and Lucida Console:

```
.natFontWeightBold {font-weight:normal;}
```

Defining Different Styles for Output Fields

The following elements are available:

Element Name	Description
.FieldVariableBased	Defines the style for output fields that are based on a variable.
.FieldLiteralBased	Defines the style for output fields that are based on a literal.

Example:

```
.FieldVariableBased {
   /* font-style:italic; */
}

.FieldLiteralBased {
    /* font-style:normal; */
}
```

Note: In the above example, as well as in the standard CSS files delivered by Software AG, the variable-based output fields are defined as italic, but are commented out.

Modifying the Natural Windows

The following elements are available:

Element Name	Description
.naturalwindow	Controls the rendering of the Natural windows.
.wintitle	Controls the rendering of the titles of the Natural windows.

Example:

```
.naturalwindow {
    border-style: solid;
    border-width: 1px;
    border-color: white;
    background-color: black;
}

.wintitle {
    left: 0px;
    top: 1px;
    height: 17px;
    width: 100%;
    color: black;
    font-size: 100%;
    font-weight: bold;
    background-color: white;
    text-align: center;
    font-family: Verdana;
    border-bottom-style: solid;
    border-bottom-width: 2px;
}
```

Note: In a mainframe environment, you have to set the Natural profile parameter `WEBIO` accordingly to enable this feature.

Modifying the Message Line

The rendering of the message line is controlled by the `.MessageLine` element.

Example:

```
.MessageLine {
    color: blue;
}
```

Note: In a mainframe environment, you have to set the Natural profile parameter `WEBIO` accordingly to enable this feature.

Modifying the Background Color

The background color is defined in the `body` element.

Example:

```
body {
      background-color: #F3F5F0;
      font-family: Lucida Console;
}
```

Modifying the Color Attributes

You can define different colors for all Natural color attributes. These are:

Red
Green
Blue
Yellow
White
Black
Pink
Turquoise
Transparent

You can define these color attributes for input fields and output fields, and for normal output and reverse video.

The following examples show how to define the color attribute “Red” .

Define the color for a normal output field:

```
.natOutputRed {color: darkred;}
```

Define the foreground and background colors for an output field with reverse video:

```
.reverseOutputRed {background-color: darkred; color:#F3F5F0;}
```

Define the color for a normal input field:

```
.natInputRed {color: darkred;}
```

Define the foreground and background colors for an input field with reverse video:

```
.reverseInputRed {background-color: darkred; color:#F3F5F0;}
```

Modifying the Style of the PF Key Buttons

The following elements are available:

Element Name	Description
`.PFButton`	Controls the style for normal rendering.
`.PFButton:hover`	Controls the style that is used when the mouse hovers over a PF key button.

Example:

```
.PFButton {
    text-align: center;
    width: 90px;
    border-style: ridge;
    border-width: 3px;
    padding: 2px;
    text-decoration: none;
    font-family: Verdana;
    font-size: 12px;
    height: 22px;
}

.PFButton:hover {
    color: #FFFF00;
    background-color: #222222;
}
```

Note: In a mainframe environment, you have to set the Natural profile parameter `WEBIO` accordingly to enable this feature.

JavaScript and XSLT Files

In addition to the CSS files described above, Natural for Ajax uses XSLT files with specific names for the conversion of the Natural Web I/O Interface screens from the internal XML format to HTML. The HTML also contains calls to JavaScript. For Web I/O screens the following conversion is done:

- Input text is placed into the HTML element `<input>` .
- Output text is placed into the HTML element `<input>` (with attribute `readonly="readonly"`).
- A message line is placed into the HTML element `<span>` .
- PF keys are embedded in an XML island and then rendered with JavaScript.
- Window elements are embedded in an XML island and then rendered with JavaScript.

The conversion is done at runtime by the following product files of your web application:

1. *<mywebapp>/WEB-INF/***transuni.xsl**
2. *<mywebapp>/scripts/***natuniscript.js**

The XSLT file *transuni.xsl* is only read once when the server is started.

Important: Do not change the above files. Software AG may change the functionality of these files in new versions or service packs of the product, which would overwrite your changes.

Customizing JavaScript

You can copy your own JavaScript file with extended JavaScript functionality into the directory *<mywebapp>/scripts*. This file must have the name ***usernatunicscript.js***. Define your new functionality in this JavaScript file.

If a *usernatunicscript.js* is found when the server is started, it is read additionally to the JavaScript file *natunicscript.js*.

For the new JavaScript functionality to be executed, you may also need to **customize the XSLT file**.

Customizing XSLT

Make a copy of *<mywebapp>/WEB-INF/transuni.xsl* and save it as *<mywebapp>/WEB-INF/****usertransuni.xsl***. Modify the XSL elements to your needs.

After making changes to *usertransuni.xsl* user file, you have to restart the server so that your changes become effective. If a *usertransuni.xsl* file is found when the server is started, it is read instead of the default XSLT file.

Customizing Overwrite / Insert mode in Input fields

Up to NJX 9.1.1, Internet Explorer always used “overwrite” mode whereas all other browsers used “insert” mode. With NJX 9.1.2 this inconsistency was fixed. The default behavior is now “overwrite” mode for all browsers. This reflects the default behavior of Natural applications.

If you prefer "insert" mode, do the following:

1. Copy the file *WEB-INF/transuni.xsl* to *WEB-INF/usertransuni.xsl* – refer to the section **customize the XSLT file** above.
2. Open *usertransuni.xsl* in a text editor. Search for the following line:

   <xsl:attribute name="onkeypress">handleKeyPress(this.name, this);</xsl:attribute>
3. Remove this line and save the file.

Packaging Customized Files in Natural for Ajax WAR Files

When using the ***Deployment Wizard for Web Applications*** of NaturalONE to deploy your Natural for Ajax *.war* file, you can automatically package a custom *usertransuni.xsl* in the *.war* file for deployment:

- Add a webconfig directory to your NaturalONE project as described in the documentation *NaturalONE > Natural for Ajax > Deploying the Application >* ***Content of the Sample webconfig Directory***.
- Copy your *usertransuni.xsl* to the *webconfig/web-inf* folder of your NaturalONE project.

7 Ajax Pages Style Sheets

Use the Style Sheet Editor tool to develop your own application-specific style sheet.

You can find further hints on how to apply a specific styling to the different controls in the respective control description chapters, for example *Working with Containers* for Natural and Java, and *Working with Grids* for Natural and Java.

8 Multi-Language Management

The multi-language management is responsible for changing the text IDs into strings that are presented to the user.

There are two translation aspects:

- All literals in the GUI definitions of a layout are replaced by strings which are language-specific.
- Literals you output within your adapter code (e.g. status messages) must be translated.

The multi-language management is internally kept cleanly behind an internal interface. This means that in the future a different implementation will be available to provide a solution to find a string for a given text ID. In this part, the default implementation which simply uses comma separated value files is described.

The information provided in this part is organized under the following headings:

Writing Multi-Language Layouts

When defining properties of controls inside a layout definition, there are always two options to specify fix labels: either use property `name` or property `textid`. In case your pages support multi-language ability, you only have to use the `textid` property. At runtime, the corresponding labels are found in the following way:

- Each PAGE has the property `translationreference`. This property may be the name of the HTML file - or it may be a logical name, used by different HTML pages.
- Inside Application Designer, there are defined directories and files in which the text information is stored: each application project is represented by a directory under the web application directory of Application Designer. Inside the project directory, there is a directory */multilanguage/*. Under this directory, each language is represented by its own directory, e.g. by the directory */multilanguage/de/* for German translations.
- Inside each language directory, there is one comma separated value (CSV) file for each page name. The name of the file is *<pagename>.csv* (for example, *Login.csv*).
- Inside the CSV file, each line contains the text ID, a semicolon and the label text, e.g. "Label1;Login name".

Example

Let us assume you have defined an application project "accountmgmt". Inside the application project, there is a layout definition *account.xml* that points via the `translationreference` property of PAGE to "account". The file structure inside your application project directory now looks as follows:

```
<webapp-directory>/
  accountmgmt/
    account.html                        // generated HTML file
    multilanguage/
      de/
        account.csv                     // German text
      en/
        account.csv                     // English text
    xml/
       account.xml                      // layout definition
```

Page Name Strategy

The previous section explained how a translation file is found for a certain HTML page. Basically, the translation reference is used to link the layout definition and the Application Designer multi-language management.

In general, there are two strategies for using this translation reference, and a mixture of both:

- Specify one central page name for a couple of pages. Therefore, all pages share the same multi-language information (i.e. the same *.csv* file).
- Specify one page name for each page. Therefore, every page has its own *.csv* file.

For larger projects, it makes sense to combine different literal information into one file - in order to keep consistency and to avoid redundancy. Of course, you have to synchronize the naming of text IDs for each page.

Creating the Translation File

The translation file (*account.csv* in the **example** of the previous section) is a simple comma separated file with the following format:

```
textid1;text1
textid2;text2
textid3,text3
```

If your text itself contains a semicolon, then write "\;".

You can either create the file by using a text editor or you can use Application Designer's Literal Assistant which is integrated in the Layout Painter.

Pay attention: when using text editors of your own, you must configure your editor to store the text using UTF-8 character encoding. Otherwise, any characters that are not "ASCII characters < 128" will not be properly displayed. Make sure that your editor is UTF-8 capable.

Tools for Translating Text IDs

There are two tools. One is the Literal Assistant that is part of the Layout Painter. The other is the Literal Translator.

Tool for Creating Languages

Application Designer comes with two languages: "en" for English and "de" for German. When creating a new language abbreviation, you have to take care of the following:

- You have to create language directories in your projects.
- You have to copy certain files in which Application Designer holds text information that is language dependent.

The Language Manager automates the creation of language abbreviations.

Unicode

Pay attention to the fact that Application Designer is fully based on Unicode and its UTF-8 format. All multi-language files must be in UTF-8 format. Especially pay attention when maintaining CSV files with programs like MS Excel.

Interface IMLManager

Application Designer uses CSV files as the default implementation for storing translation information. In some scenarios, you do not want Application Designer to be responsible for storing this information, but want to store translation information on your own.

Example: you may already have a text database within your existing application, and you may already have defined procedures on top of this: translation tools, automated translation etc.

Application Designer provides an interface `com.softwareag.cis.multilanguage.IMLManager` that is internally used for accessing translation information:

```
public interface IMLManager
    public void refreshBuffers();
    public String getString(String language,
                            String project,
                            String application,
                            String literal);
    public boolean checkIfExists(String language,
                                 String project,
                                 String application,
                                 String literal);
    public String getString(String language,
                            String project,
                            String application,
                            String literal,
                            String[] embeddedStrings);
    public String[] getApplicationStrings(String language,
                                          String project,
                                          String application,
                                          String[] textIds);

}
```

See the JavaDoc API documentation for detailed information.

You can implement this interface by a class of your own and register this class in the Application Designer runtime environment. The registration is done inside the *cisconfig.xml* file:

```
<cisconfig ...
            multilanguagemanager="<your implementation>"
            ...>

            ↩
```

Be aware of the fact that the interface is a pure “access interface” that is used at runtime; i.e. whenever Application Designer requires information about a certain text ID, the interface is called. The interface currently does not provide methods for writing text information back, i.e. there is no coupling of Application Designer's multi-language tools (e.g. translation inside the Layout Painter).

9 Starting a Natural Application with a URL

The connection parameters available in the configuration file for the session and on the logon page can also be specified as URL parameters of the logon page URL. This allows bookmarking the startup URL of a Natural application or starting an application by clicking a hyperlink in a document.

The URL parameters overrule the definitions in the configuration file, with the exception described in the table below.

The following URL parameters are available for the logon page:

URL Parameter	Corresponding Option in the Session Configuration
`xciParameters.natsession`	**Session ID**
`xciParameters.natserver`	**Host name**
`xciParameters.natport`	**Port number**
`xciParameters.natuser`	**User name**
`xciParameters.natpassword`	**Password**
`xciParameters.natprog`	**Application**
`xciParameters.natparam`	**Natural parameters**
`xciParameters.natparamext`	**Natural parameters** The URL parameter `xciParameters.natparamext` extends an existing Natural parameter definition in the configuration file. The extension works in the following way: the Natural parameters defined in the configuration file come first. Then, the Natural parameters defined in the URL parameter `xciParameters.natparamext` are added, separated by a space character. If you want to overrule the definition in the configuration file, use the URL parameter `xciParameters.natparam` instead.
`xciParameters.nattimeout`	**Timeout (n seconds)**

URL Parameter	Corresponding Option in the Session Configuration
`xciParameters.savesessionuser`	**Save user credentials**
`xciParameters.sharesessionuser`	**Share session user**

⚠ **Important:** All parameter values must be URL-encoded.

Example: In order to start the Natural program `MENU-NJX` from the library `SYSEXNJX` , while your application server is running on *myappserver:4711* , your Natural Web I/O Interface server is running on *mywebio:4712* , and the name of the Natural startup script is *nwo.sh* , you can use the following URL:

http://myappserver:4711/cisnatural/servlet/StartCISPage?PAGEURL=%2Fcisnatural%2FNatLogon.html&xciParameters.natserver=mywebio&xciParameters.natprog=nwo.sh&xciParameters.natport=4712&xciParameters.natparam=stack%3D%28logon+SYSEXNJX%3BMENU-NJX%3Bfin%29

Instead of adding the parameters to the URL, you can also use the HTTP method `POST` to set the parameters in an HTML form. Example:

```
<html>
<head>
<title>Start Natural for Ajax Session</title>
<script type="text/javascript">
function submitStart() {
document.forms["myform"].submit();
}
</script>
</head>
<body>
 <form id="myform" name="myform" action="servlet/StartCISPage" method="post">
  <input type="hidden" name="PAGEURL" value="/cisnatural/NatLogon.html" />
  Host: <input type="input" size="20" name="xciParameters.natserver" value="mywebio" ↩
/><br/>
   Port: <input type="input" size="10" name="xciParameters.natport" value="4712" ↩
/><br/>
   Natural program: <input type="input" size="20" name="xciParameters.natprog" ↩
value="nwo.sh" /><br/>
  Natural parameter: <input type="input" size="50" name="xciParameters.natparam" ↩
value="stack=(logon sysexnjx;menu)" />
 </form>
 <a href="#" onclick="submitStart()">Start Session</a>
 <div id="status">Click on Start Session</div>
</body>
</html>
```

When you write the above example code to a file named *startWithPost.html* which is located in the *cisnatural* main directory, you can start the form with the following URL:

http://myappserver:4711/cisnatural/startWithPost.html

10 Configuring Container-Managed Security

General Information

Natural for Ajax comes as a Java EE-based application. For the ease of installation, the access to this application is by default not secured. You might, however, wish to restrict the access to certain parts of the application to certain users. An important example is the **configuration tool** , which enables you to modify the Natural session definitions and the logging configuration of Natural for Ajax . Other examples are the Application Designer development workplace contained in Natural for Ajax or the Natural logon page.

This section does not cover the concepts of JAAS-based security in full extent. It provides, however, sufficient information to activate the preconfigured security settings of Natural for Ajax and to adapt them to your requirements. More information on the topics described in this section can be found, for instance, at ***http://www.wildfly.org/*** .

Notes:

1. The recommended security method is application-managed authentication. For further information, see *Configuring Application-Managed Authentication* .
2. Container-managed security is not supported for IBM WebSphere Application Server. Use application-managed authentication instead.

Name and Location of the Configuration File

Security is configured in the file *web.xml* . The path to this file depends on the application server.

- **Wildfly Application Server**
  `<application-server-install-dir>`*/server/default/deploy/njx* `<nn>`*.ear/cisnatural.war/WEB-INF*
- **Apache Tomcat**
  `<tomcat-install-dir>`*/webapps/cisnatural/WEB-INF*

Activating Security

Great care must be taken when editing and changing the configuration file *web.xml* . After a change, the application server must be restarted.

Edit the file *web.xml* and look for the section that is commented with "Uncomment the next lines to add security constraints and roles." . Uncomment this section by removing the comment marks shown in boldface below:

```
<!-- Uncomment the next lines to add security constraints and roles. -->
<!--
<security-constraint>
    <web-resource-collection>
    <web-resource-name>Configuration Tool</web-resource-name>
        <url-pattern>/conf_index.jsp</url-pattern>
        <url-pattern>/faces/*</url-pattern>
    </web-resource-collection>
...
<security-role>
    <description>Administrator</description>
    <role-name>nwoadmin</role-name>
</security-role>
-->
```

Defining Security Constraints

The security constraints defined by default are just examples. A `<security-constraint>` element contains of a number of `<web-resource-collection>` elements combined with an `<auth-constraint>` element. The `<auth-constraint>` element contains a `<role-name>`. The whole `<security-constraint>` element describes which roles have access to the specified resources.

Example - the following definition specifies that only users in the role "nwoadmin" have access to the configuration tool:

```
<security-constraint>
    <web-resource-collection>
    <web-resource-name>Configuration Tool</web-resource-name>
        <url-pattern>/conf_index.jsp</url-pattern>
        <url-pattern>/faces/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>nwoadmin</role-name>
    </auth-constraint>
</security-constraint>
```

In the following section, you will see where and how the roles are defined.

Defining Roles

A few lines below in the file *web.xml* , there is a section `<security-role>` . Here, the roles that can be used in `<security-constraint>` elements are defined. You can define additional roles as needed. The assignment of users to roles is done outside this file and will often be done in a user management that is already established at your site.

Example:

```
<security-role>
    <description>Administrator</description>
    <role-name>nwoadmin</role-name>
</security-role>
```

Selecting the Authentication Method

In the file *web.xml* , there is a section `<login-config>` . The only element that should possibly be adapted here is `<auth-method>` . You can choose between the authentication methods "FORM" and "BASIC" . Form-based authentication displays a specific page on which users who try to access a restricted resource can authenticate themselves. Basic authentication advises the web browser to retrieve the user credentials with its own dialog box.

Example:

```
<login-config>
    <auth-method>FORM</auth-method>
...
</login-config>
```

Choosing the Login Module (WildFly)

On WildFly, Natural for Ajax is installed as a web application (WAR file). See *Installing Natural for Ajax on WildFly* .

The configuration of WildFly as a so-called standalone server is described here.

All configuration (especially the security configuration) is centralized in the file *`<application-server-install-dir>`/standalone/configuration/standalone.xml* .

The following describes how to create a sample JAAS-based security configuration for WildFly 26:

1. **njxnwo-roles.properties and njxnwo-users.properties**

   Move the following sample configuration files from `<application-server-install-dir>` */standalone/deployments/cisnatural.war/WEB-INF* to their appropriate location as described below:

   njxnwo-roles.properties* and*njxnwo-users.properties
   Move these two files to `<application-server-install-dir>` */standalone/configuration* .

2. Add the following security domain definition in the file *standalone.xml* , under `<security-domains>`:

```
<security-domains>
  ...
  <security-domain name="NaturalWebIOAndAjaxRealm" ↩
default-realm="NaturalWebIOAndAjaxRealm" ↩
permission-mapper="default-permission-mapper">
    <realm name="NaturalWebIOAndAjaxRealm"/>
  </security-domain>
  ...
</security-domains>

<security-realms>
  ...
  <properties-realm name="NaturalWebIOAndAjaxRealm" groups-attribute="Roles">
    <users-properties path="njxnwo-users.properties" ↩
relative-to="jboss.server.config.dir" digest-realm-name="NaturalWebIOAndAjaxRealm" ↩
/>
    <groups-properties path="njxnwo-roles.properties" ↩
relative-to="jboss.server.config.dir"/>
  </properties-realm>
  ...
</security-realms>
```

3. Under `<http>` add the following:

```
<http>
  ...
  <http-authentication-factory name="NaturalWebIOAndAjaxRealm-http" ↩
http-server-mechanism-factory="global" security-domain="NaturalWebIOAndAjaxRealm">
    <mechanism-configuration>
      <mechanism mechanism-name="FORM"/>
    </mechanism-configuration>
  </http-authentication-factory>
     ...
</http>
```

4. Under `<subsystem xmlns="urn:jboss:domain:undertow:12.0" ...>`, add the following:

```
<subsystem xmlns="urn:jboss:domain:undertow:12.0" ...>
  ...
  <application-security-domains>
    ...
    <application-security-domain name="NaturalWebIOAndAjaxRealm" ↩
http-authentication-factory="NaturalWebIOAndAjaxRealm-http"/>
    ...
  </application-security-domains>
  ...
</subsystem>
```

This sample configuration uses the login module `UsersRoles`. The login module `UsersRoles` expects the role definitions in one file (*njxnwo-roles.properties*) and the user definitions (password and assignment to roles) in another file (*njxnwo-users.properties*). An example user "admin" with the password "adminadmin" and the role "nwoadmin" is defined to begin with.

You can choose and configure a different login module (for example, one that expects the user and role definitions in a database or in an LDAP directory), or you can even write a custom login module.

Configuring the UserDatabaseRealm (Apache Tomcat only)

In the *tomcat-users.xml* file (which is located in the *conf* directory), specify the role "nwoadmin" for any desired user name and password. For example:

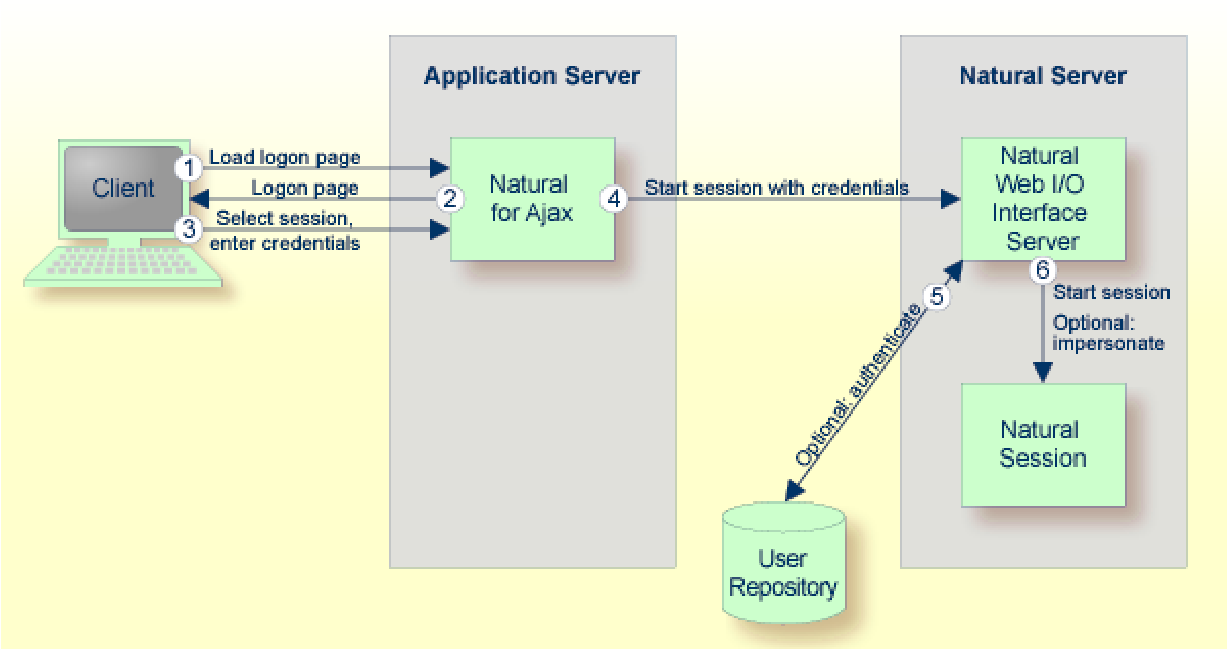
```
<user username="pepe" password="pepe123" roles="nwoadmin"/>
```

For detailed information on the necessary realm configuration for Tomcat, see ***http://tomcat.apache.org/tomcat-6.0-doc/realm-howto.html#UserDatabaseRealm*** .

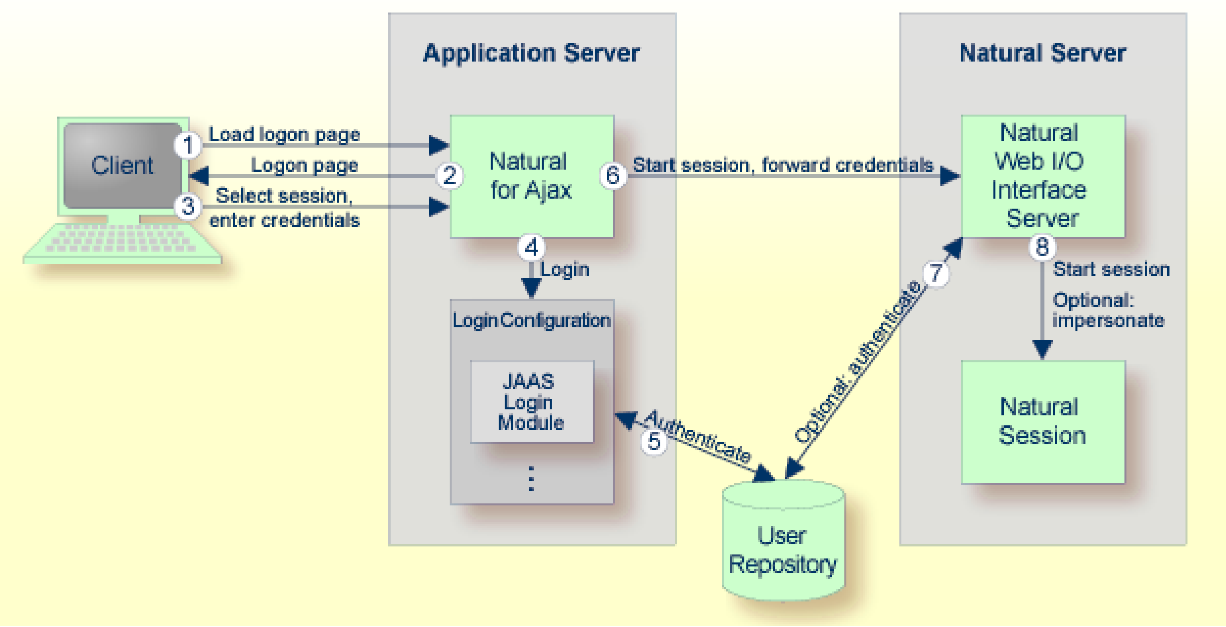
11 Configuring Application-Managed Authentication

General Information

Natural for Ajax is a Java EE-based application that runs on an application server or web container. By default, the access to this application is not secured. The credentials that users enter in the Natural for Ajax logon page (see the graphic below) are used to authenticate them in the selected Natural environment. They are not used to authenticate them on the application server or web container.



You might, however, wish to authenticate the users already in the application server or web container, before they even attempt to access a Natural session. This can be achieved with container-managed security (see *Configuring Container-Managed Security*), but only for a subset of the supported application servers. With application-managed authentication, this can be achieved for all supported application servers and web containers.



When application-managed authentication has been activated, the user is first authenticated on the application server or web container. For the authentication, JAAS-based (Java Authentication and Authorization Service) login modules are used. The login modules to be used and their parameters are configured in a login configuration. The user credentials entered on the Natural for Ajax logon page are authenticated against a user repository that is defined by the configured login modules.

A login configuration can consist of several login modules that are executed one after the other. A specific login module is responsible to forward the entered credentials from the application server or web container to the Natural Web I/O Interface server, so that they can be reused to authenticate the user on the Natural server side.

Activating Application-Managed Authentication

Application-managed authentication is activated on a per-session basis in the **configuration tool**
.

To activate-application managed authentication for a session

1 **Invoke** the configuration tool.

2 In the frame on the left, choose the **Session Configuration** link.

3 **Add** a new session or **edit** an existing session.

4 Select the **Use JAAS-based authentication** check box.

The **Forward credentials** check box is then selected automatically. This makes sure that the credentials entered for the authentication on the application server or web container are forwarded to the Natural server.

5 Choose the **OK** button.

6 Choose the **Save Configuration** button.

When a user opens the Natural for Ajax logon page and selects the session for which you made the above changes, the credentials entered on the logon page are now used to authenticate the user on the application server or web container and are then forwarded to the Natural server.

This applies also when a user does not explicitly open the logon page in order to select a session manually, but instead passes the session name as an URL parameter to the logon page as described in ***Starting a Natural Application with a URL*** .

Securing the Logon Page

You may wish to authenticate the users before they even access the Natural for Ajax logon page. This is activated globally in the **configuration tool** .

To secure the logon page

1 **Invoke** the configuration tool.

2 In the frame on the left, choose the **Session Configuration** link.

3 Select the **Use secure logon page** check box.

4 Choose the **Save Configuration** button.

When a user opens the Natural for Ajax logon page without specifying a session name beforehand, the user will now be prompted to enter the credentials in order to be authenticated on the application server or web container.

The Login Configuration

With JAAS, authentication is always done against a so-called realm. A realm defines the scope of security definitions. There can be several distinct realms. The user "George" in realm A, for example, is considered to be different from the user "George" in realm B. The realms are usually defined in a login configuration file. The location of this file depends on the application server or web container. A typical realm definition contains a set of login modules that are executed in a specific order to authenticate a user within this realm. The login modules are responsible for the actual authentication.

Natural for Ajax authenticates users against a realm named "NaturalWebIOAndAjaxRealm" . Therefore, the login configuration of the application server or web container must contain a realm definition with this name.

Defining the Login Configuration on Wildfly Application Server

The login configuration depends on the Wildfly Application Server version. To define a sample configuration, proceed as described in one of the following sections, depending on the version that you are using:

- *Choosing the Login Module (WildFly)*

Further configuration is described in the version-specific topics below:

- Wildfly Application Server

Wildfly Application Server

If you use other login modules than in the sample configuration, copy the JAR files with these login modules into the *WEB-INF/lib* directory of the Natural for Ajax web application, which is called *cisnatural.war* by default.

In order to prepare for the step ***Forwarding the User Credentials to Natural*** , you need to provide the Natural for Ajax login module *com.softwareag.njx.loginmodule.NJXLoginModule* in the right place. This login module is contained in the *JBoss7-WildFly8* directory of the installation medium, in the file *njxlogin<nn>.jar* . Copy this file also into the *WEB-INF/lib* directory of of the Natural for Ajax web application.

Defining the Login Configuration on IBM WebSphere Application Server

Copy the JAR files with the login modules to be used into the *lib/ext* directory of your IBM WebSphere installation. The Natural for Ajax login module *com.softwareag.njx.loginmodule.NJXLoginModule* mentioned below is contained in the file *njxlogin<nn>.jar* , which can be found in the WebSphere-specific directory of the installation medium.

To configure the login module

1 Make sure the application server is running.

2 Open your web browser and enter the following URL:

```
http://<host>:<adminport>/ibm/console
```

This opens the Administration Console.

3 Open the tree node **Security > Global security** .

4 On the right side of the screen, open **Java Authentication and Authorization Service** .

5 Choose **Application logins** .

6 Choose **New** .

7 Enter "NaturalWebIOAndAjaxRealm" as an alias.

8 Choose **OK** .

9 Choose **Save** .

10 Select **NaturalWebIOAndAjaxRealm** .

11 Choose **New** .

12 Enter the class name of your login module.

13 Configure the authentication strategy and custom properties of your login module.

14 Choose **OK** .

15 Choose **Save** .

16 Select **NaturalWebIOAndAjaxRealm** once more.

17 Choose **New** .

18 Enter the class name "com.softwareag.njx.loginmodule.NJXLoginModule" .

19 Choose **OPTIONAL** as the authentication strategy.

20 Enter "useFirstPass" as the property name.

21 Enter "true" as the property value.

22 Select the **Select** check box.

23 Choose **New** .

24 Enter "storePass" as the property name.

25 Enter "true" as the property value.

26 Select the **Select** check box.

27 Choose **OK** .

28 Choose **Save** .

Defining the Login Configuration on Apache Tomcat

Copy the JAR files with the login modules to be used into the *lib* directory of your Apache Tomcat installation.

The Natural for Ajax login module *com.softwareag.njx.loginmodule.NJXLoginModule* mentioned below is contained in the file *njxlogin<nn>.jar* , which can be found in the Tomcat-specific directory of the installation medium. Copy the file *njxlogin<nn>.jar* into the *WEB-INF/lib* directory of the Natural for Ajax web application, which is called *cisnatural* by default.

In the *conf* directory of your Apache Tomcat installation, add a new properties file named *njxjaas_config.properties* . Within this file, configure the login modules in the following way:

```
NaturalWebIOAndAjaxRealm {
  your-login-module-class required
    param1="value1"
    param2="value2";

  com.softwareag.njx.loginmodule.NJXLoginModule optional
    useFirstPass=true
    storePass=true;
};
```

On Windows, edit the file *startup.bat* in the Apache Tomcat *bin* directory and add the following line:

```
set JAVA_OPTS=%JAVA_OPTS% ↩
-Djava.security.auth.login.config=%CATALINA_HOME%/conf/njxjaas_config.properties
```

Or, if you have installed Apache Tomcat as a Windows service, specify the above Java option in the Apache Tomcat **Properties** dialog.

On Linux, edit the file *startup.sh* in the Apache Tomcat *bin* directory and add the following line:

```
JAVA_OPTS=$JAVA_OPTS ↩
-Djava.security.auth.login.config=$CATALINA_HOME/conf/njxjaas_config.properties
```

Forwarding the User Credentials to Natural

When the user has been authenticated on the application server or web container, the authenticated user with the credentials can be forwarded directly to the Natural Web I/O Interface server. Optionally, the user can be authenticated the Natural Web I/O Interface server again. Also optionally, the started Natural session can be started under the user ID of the client (impersonation).

However, this works only if both the authentication on the application server and the authentication on the Natural Web I/O Interface server are done with the same credentials against the same authentication system. This will be the case, for example, if the Natural Web I/O Interface server is configured to authenticate with RACF and you have configured a login module on the application server or web container that authenticates the user against the same system.

To forward the user credentials to Natural

1. **Invoke** the configuration tool.
2. In the frame on the left, choose the **Session Configuration** link.
3. **Add** a new session or **edit** an existing session.
4. Make sure that the **Forward credentials** check box is selected.
5. Choose the **OK** button.
6. Choose the **Save Configuration** button.

Using Software AG Security Infrastructure (Deprecated)

Software AG Security Infrastructure (SIN) has originally been designed as a common authentication infrastructure for the webMethods product suite. SIN is based on the above mentioned JAAS (Java Authentication and Authorization Service) framework and provides a number of predefined JAAS-based login modules. For detailed information on these login modules and their configuration, see the Software AG Security Infrastructure documentation at ***http://documentation.softwareag.com/webmethods/security_infrastructure.htm*** .

With this Natural for Ajax distribution, the SIN login modules are delivered in the top level directory */Support/SIN* as a zip file with name *SIN_* `<version>` *.zip* . Unzip this file to a temporary directory. The login modules are then contained in the directory with the name *jars* . The JAR files of interest are:

- *sin-common.jar*
- *sin-misc.jar*
- *sin-ssx.jar*

- *sin-xmlserver.jar*

If you want to use the SIN login modules with Natural for Ajax, you have to copy the above JAR files into the proper directory of your application server or web container (for information on the proper directory, see the above sections which describe how to define the login configuration).

Natural for Ajax currently supports the following predefined login modules from the SIN package:

- `SSXLoginModule`
- `SAMLArtifactLoginModule`

SAMLArtifactLoginModule

When this login module is installed and configured for Natural for Ajax, the end user is not prompted for a user name or password during the login.

Make sure that the `opensaml` library and the Apache common libraries are in the class path of the login module (for detailed information, see the description of the `SAMLArtifactLoginModule` in the Software AG Security Infrastructure documentation) and then configure the use of SAML as described below.

To configure the use of SAML

1. **Invoke** the configuration tool.
2. In the frame on the left, choose the **Session Configuration** link.
3. Under **Global Settings** , make sure that the **Use SAML for JAAS-based authentication** check box is selected.
4. **Add** a new session or **edit** an existing session.
5. Make sure that the **Use JAAS-based authentication** check box is selected.

   When both **Use SAML for JAAS-based authentication** and **Use JAAS-based authentication** are selected, the user name and password fields are not visible in the Natural for Ajax logon page for the used session.
6. Choose the **OK** button.
7. Optional. Under **Global Settings** , select the **Secure logon page** check box.

   Now, the secure logon page will only be displayed in the case of an error in the SAML artifact validation.
8. Choose the **Save Configuration** button.

For more information on the above options, see *Session Configuration* .

In both logon pages (Natural for Ajax logon page and secure logon page), the end user is no longer prompted to enter user name and password. Instead, the SAML artefact ticket, which is transported in the HTTP header, is validated. The `SAMLArtifactLoginModule` can extract the user name from the SAML artefact. Therefore, it is not necessary to enter a user name. A valid SAML artefact inside the HTTP header provides the authentication for the Natural for Ajax server.

Important: The Natural for Ajax application does not send a password. Therefore, you have to disable the password check on the Natural Web I/O Interface server. Make sure that only selected Natural for Ajax application servers have access to a Natural server which has been configured in this way.

Using Integrated Authentication Framework (IAF) (Deprecated)

With Natural for Ajax, you can authenticate against a Software AG IAF server. Natural for Ajax delivers an IAF login module for this purpose. After you have successfully entered your login credentials, the IAF server creates an IAF token. This token will be sent to Natural for enhanced authentication.

The IAF login module needs a running IAF server. For detailed information on how to set up an IAF server, see the Software AG Security Infrastructure documentation at *http://documentation.softwareag.com/webmethods/security_infrastructure.htm* .

The name of the IAF login module is `com.softwareag.njx.loginmodule.iaf.IAFLoginModule`. It supports the following parameter:

Parameter	Description
`iafConnection`	Mandatory. The address of the IAF server connection. This address must also include the location of the tru `iafConnection="ssl://myiafserver:11958?TRUST_STORE=C:\\SoftwareAG\\IAF\\iaf\\etc\\I` The default port is 11958. `TRUST_STORE` must define a valid path on the PC on which the web server is running.

The Natural for Ajax login modules `com.softwareag.njx.loginmodule.NJXLoginModule` and `com.softwareag.njx.loginmodule.iaf.IAFLoginModule` are contained in the following files:

- *njxlogin<nn>.jar*
- *njxiaflogin<nn>.jar*

The above JAR files are contained in the application server-specific directory of the installation medium. You have to copy these files into the proper directory of your application server or web container (see below).

The IAF login module uses parts of the Software AG Security Infrastructure. Therefore, the following additional JAR files are required:

- *IafClient.jar*
- *entirex.jar*
- *sin-common.jar*

The above JAR files are delivered in the top level directory */Support/SIN* of the installation medium, in a zip file with name *SIN_* `<version>` *.zip* . Unzip this file to a temporary directory. The JAR files are then contained in the directory with the name *jars* . Note that the file *entirex.jar* is contained in the file *IafClient.jar* and needs to be extracted from there. You have to copy these files into the proper directory of your application server or web container (see below).

Further configuration information is provided in the topics below:

- Configuration on Wildfly Application Server
- Configuration on IBM WebSphere Application Server
- Configuration on Apache Tomcat

Configuration on Wildfly Application Server

Prepare the login modules as described under ***Choosing the Login Module (Wildfly)*** .

Copy all required JAR files (see above) from the installation medium to *standalone\deployments\cisnatural.war\WEB-INF\lib* .

Edit the file *standalone.xml* and add the definition for the IAF login module under `<security-domains>`:

```
<security-domain name="NaturalWebIOAndAjaxRealm" cache-type="default">
 <authentication>
  <login-module code="com.softwareag.njx.loginmodule.iaf.IAFLoginModule" ↩
flag="required">
   <module-option name="iafConnection" ↩
value="ssl://myiafserver:11958?TRUST_STORE=C:\\SoftwareAG\\IAF\\iaf\\etc\\IAFAppCert.jks&amp;VERIFY_SERVER=N"/>
  </login-module>
  <login-module code="com.softwareag.njx.loginmodule.NJXLoginModule" flag="optional">
   <module-option name="password-stacking" value="useFirstPass"/>
   <module-option name="useFirstPass" value="true"/>
  </login-module>
 </authentication>
</security-domain>
```

Configuration on IBM WebSphere Application Server

Prepare the login modules as described above under *Defining the Login Configuration on IBM WebSphere Application Server* .

Copy all required JAR files (see above) from the installation medium to *lib/ext* .

Configure the IAF login module as described below.

To configure the IAF login module

1 Configure the login module in the same way as described under *Defining the Login Configuration on IBM WebSphere Application Server* .

2 For the step "Enter the class name of your login module" , enter the following:

```
com.softwareag.njx.loginmodule.iaf.IAFLoginModule
```

3 For the step "Configure the authentication strategy and custom properties of your login module" proceed as follows:

- Enter the class name of your login module: "com.softwareag.njx.loginmodule.iaf.IAFLoginModule" .
- Choose **REQUIRED** as the authentication strategy.
- Enter "iafConnection" as the property name.
- Enter the IAF connection string as the property value. Example:

```
iafConnection="ssl://myiafserver:11958?TRUST_STORE=C:\\SoftwareAG\\IAF\\iaf\\etc\\IAFAppCert.jks&VERIFY_SERVER=N";
```

4 Proceed with the remaining configuration steps described under *Defining the Login Configuration on IBM WebSphere Application Server* .

Configuration on Apache Tomcat

Prepare the login modules as described above under *Defining the Login Configuration on Apache Tomcat* .

Copy all required JAR files (see above) from the installation medium to *WEB-INF/lib* .

Edit the file *njxjaas_config.properties* and add the definition for the IAF login module:

```
NaturalWebIOAndAjaxRealm {
  com.softwareag.njx.loginmodule.iaf.IAFLoginModule required
    ↩
iafConnection="ssl://myiafserver:11958?TRUST_STORE=C:\\SoftwareAG\\IAF\\iaf\\etc\\IAFAppCert.jks&VERIFY_SERVER=N";

  com.softwareag.njx.loginmodule.NJXLoginModule optional
    useFirstPass="true"
    tomcatMode="true"
    storePass="true";
};
```

12 Wrapping a Natural for Ajax Application as a Servlet

In a production environment, it is inconvenient to start an application with a URL such as the following:

http://myappserver:4711/cisnatural/servlet/StartCISPage?PAGEURL=%2Fcisnatural%2FNatLogon.html&xciParameters.natserver=mywebio&xciParameters.natprog=nwo.sh&xciParameters.natport=4712&xciParameters.natparam=stack%3D%28logon+SYSEXNJX%3BMENU-NJX%3BFIN%29

The URL can be shortened by defining a corresponding session profile in the **configuration tool** . For example:

http://myappserver:4711/cisnatural/servlet/StartCISPage?PAGEURL=%2Fcisnatural%2FNatLogon.html&xciParameters.natsession=DemoApplication

However, this shortened URL is still not practical for security reasons because end users should not be allowed to access the generic servlet `StartCISPage` .

When you define a dedicated servlet for each application, you can easily define the security constraints for each application in the file *web.xml* (for further information on this file, see ***Configuring Container-Managed Security***). With the servlet `com.softwareag.cis.server.StartCISPageWithParams`, you define a wrapper servlet for a given Natural for Ajax application so that the application can later be invoked with a URL such as the following:

http://myappserver:4711/cisnatural/servlet/DemoApplication

Note: The servlet `com.softwareag.cis.server.StartCISPageWithParams` can also be used with the HTTP method `POST` . See ***Starting a Natural Application with a URL*** .

The following example shows how you define an application as a servlet in the file *web.xml* .

```
<servlet id="DemoApplication">
  <servlet-name>DemoApplication</servlet-name>
  <display-name>DemoApplication</display-name>
  <servlet-class>com.softwareag.cis.server.StartCISPageWithParams</servlet-class>
  <load-on-startup>1</load-on-startup>
  <init-param id="OVERWRITE">
    <param-name>OVERWRITE</param-name>
    <param-value>false</param-value>
  </init-param>
  <init-param id="PAGEURL">
    <param-name>PAGEURL</param-name>
    <param-value>/cisnatural/NatLogon.html</param-value>
  </init-param>
  <init-param id="xciParameters.natsession">
    <param-name>xciParameters.natsession</param-name>
    <param-value>Local</param-value>
  </init-param>
  <init-param id="xciParameters.natserver">
    <param-name>xciParameters.natserver</param-name>
    <param-value>localhost</param-value>
  </init-param>
  <init-param id="xciParameters.natport">
    <param-name>xciParameters.natport</param-name>
    <param-value>2900</param-value>
  </init-param>
  <init-param id="xciParameters.natparamext">
    <param-name>xciParameters.natparamext</param-name>
    <param-value>STACK=(LOGON SYSEXNJX;MENU-NJX;FIN)</param-value>
  </init-param>
</servlet>
```

You can omit the parameters `xciParameters.natserver`, `xciParameters.natport` and `xciParameters.natparamext` if the corresponding values are defined in the session definition that is referenced in `xciParameters.natsession` . This the recommended way, because the settings can thus be changed in the configuration tool at any time without the need to adapt the file *web.xml* .

To complete the definition, you define a corresponding servlet mapping in the file *web.xml* :

```
<servlet-mapping>
  <servlet-name>DemoApplication</servlet-name>
  <url-pattern>/servlet/DemoApplication</url-pattern>
</servlet-mapping>
```

With this servlet mapping, you can now start your application by URL. Example: *http://myhost:myport/mywebapp/servlet/DemoApplication*

If you additionally want to prevent users from starting other NJX applications you can do the following:

Exchange the servlet class `com.softwareag.cis.server.StartCISPage` with a different servlet class, namely `com.softwareag.cis.server.StartCISPageInSession` . This servlet class cannot be called directly; it can only be called in an Application Designer session which is already active. Therefore, each attempt to start an arbitrary - not wrapped - application by just building a URL based on `StartCISPage` will result in an error message.

To exchange the servlet class, you change the following in the file *web.xml*

```
<servlet id="StartCISPage">
  <servlet-name>StartCISPage</servlet-name>
  <display-name>StartCISPage</display-name>
  <servlet-class>com.softwareag.cis.server.StartCISPage</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
```

to

```
<servlet id="StartCISPage">
  <servlet-name>StartCISPage</servlet-name>
  <display-name>StartCISPage</display-name>
  <servlet-class>com.softwareag.cis.server.StartCISPageInSession</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
```

If your application uses pop-ups, subcispage controls or web I/O pages you have to add the parameter `ALLOWSUBPAGES` to the servlet definition in your *web.xml* :

```
<servlet id="StartCISPage">
  <servlet-name>StartCISPage</servlet-name>
  <display-name>StartCISPage</display-name>
  <servlet-class>com.softwareag.cis.server.StartCISPageInSession</servlet-class>
  <load-on-startup>1</load-on-startup>
  <init-param id="ALLOWSUBPAGES">
    <param-name>ALLOWSUBPAGES</param-name>
    <param-value>true</param-value>
  </init-param>
</servlet>
```

For a workplace application , the definition of the `com.softwareag.cis.server.StartCISPageWithParams` in the *web.xml* is slightly different. You do not define the start-up page of the workplace directly in the `PAGEURL` parameter. Instead, you define an intermediate navigation page */HTMLBasedGUI/com.softwareag.cis.util.navigatetopage.html* . The start-up page of the workplace is specified in the `navPage` parameter. The navigation page makes sure that an Applicaton Designer session is created before it navigates to the start-up page of the workplace.

```
<servlet id="WorkplaceDemo">
  <servlet-name>WorkplaceDemo</servlet-name>
  <display-name>WorkplaceDemo</display-name>
  <servlet-class>com.softwareag.cis.server.StartCISPageWithParams</servlet-class>
  <load-on-startup>1</load-on-startup>
  <init-param id="OVERWRITE">
    <param-name>OVERWRITE</param-name>
    <param-value>false</param-value>
  </init-param>
  <init-param id="PAGEURL">
    <param-name>PAGEURL</param-name>
    ↩
<param-value>/HTMLBasedGUI/com.softwareag.cis.util.navigatetopage.html</param-value>
  </init-param>
  <init-param id="navPage">
    <param-name>navPage</param-name>
    <param-value>/njxdemos/wpdynworkplace.html</param-value>
  </init-param>
</servlet>
```

In this case, the corresponding servlet mapping is defined as follows:

```
<servlet-mapping>
  <servlet-name>WorkplaceDemo</servlet-name>
  <url-pattern>/servlet/WorkplaceDemo</url-pattern>
</servlet-mapping>
```

Note: For further information on the above mentioned servlets, see the Java API documentation which is provided with Application Designer.

13 Customizing Error Pages

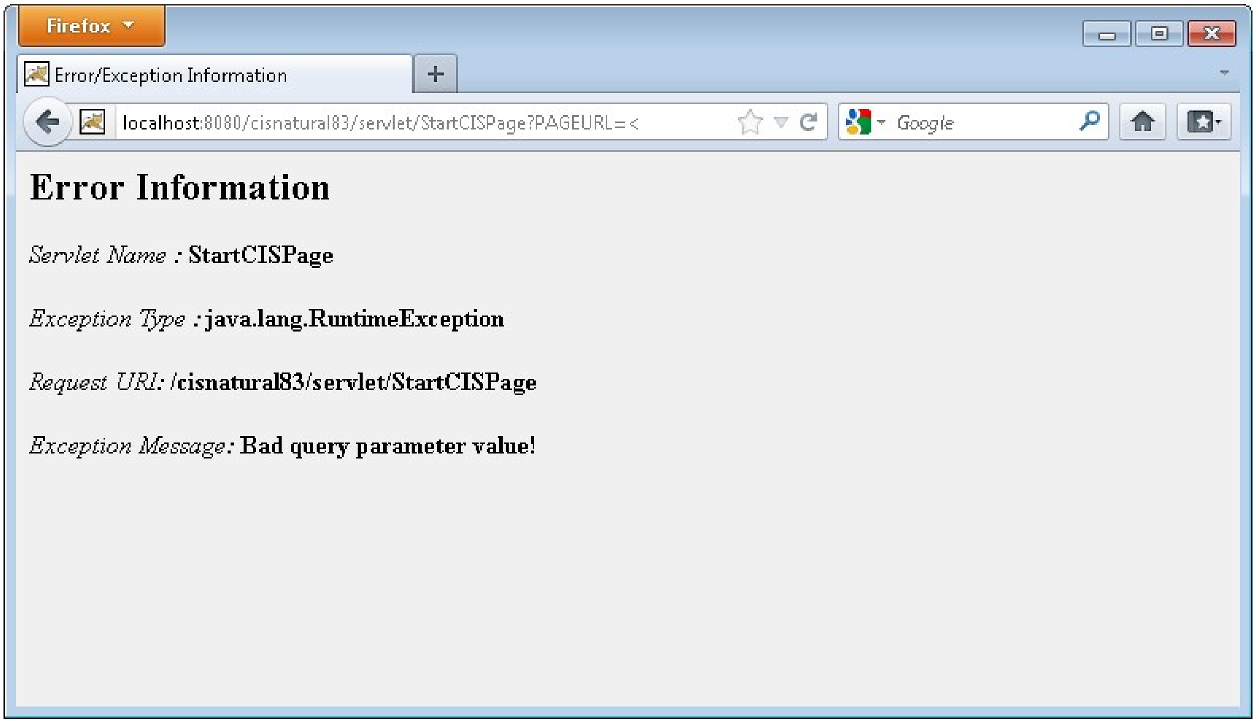
In the case of unexpected errors, most application servers show a default error page. This default error page mostly contains information such as stack traces. Showing full stack traces in a production environment is regarded as a security risk. To avoid this vulnerability, you can configure your own error pages in the *web.xml* file of your web application. For your convenience, the product contains a ready-to-use error handling servlet. The following example shows how to configure this servlet in the *web.xml* file:

```
<servlet id="DefaultErrorHandler">
         <servlet-name>DefaultErrorHandler</servlet-name>
        <servlet-class>com.softwareag.cis.server.DefaultErrorServlet</servlet-class>
</servlet>

<servlet-mapping>
     <servlet-name>DefaultErrorHandler</servlet-name>
     <url-pattern>/DefaultErrorHandler</url-pattern>
</servlet-mapping>

<error-page>
      <exception-type>java.lang.Throwable</exception-type >
      <location>/DefaultErrorHandler</location>
</error-page>
```

The following is a sample error page that has been generated by the `com.softwareag.cis.server.DefaultErrorServlet`:


Firefox
Error/Exception Information
localhost:8080/cisnatural83/servlet/StartCISPage?PAGEURL=<
Google

Error Information

Servlet Name : **StartCISPage**

Exception Type : **java.lang.RuntimeException**

Request URI: **/cisnatural83/servlet/StartCISPage**

Exception Message: **Bad query parameter value!**

As an alternative to this default error handling servlet, you can add your own error handling servlets and/or error pages.

14 License Checks

The following describes the license check support.

License Check during Deployment

When you deploy a Natural for Ajax web application into an Application Server or a Web Container, license checks are done. In case of an invalid license, you'll find an error in the Natural for Ajax server log file or the log file of your Application Server/Web Container, depending on the configuration. Each access to the deployed web application shows a corresponding license error page in the browser.

License Check for Deployed Applications

Whenever a new session is started in the browser, the license is checked.

- License Errors Before Expiration Date
- License Errors After Expiration Date

License Errors Before Expiration Date

Starting 40 days before the license expires, corresponding warnings are written to the Natural for Ajax server log file or the log file of your Application Server/Web Container, depending on the configuration. Additionally, you will see a corresponding warning text when opening the Monitoring Tool.

Optionally, in the cisconfig.xml file, you can define a list of hostnames. When the web application is called with an URL containing this hostname, an alert box with a license warning is shown once per day.

Example:

```
<cisconfig licwarningsfor=”localhost” ...>
```

This will show an alert box for the URL *http(s)://localhost:<port>/mywebapp/**. It will not show any alert boxes when the web application is accessed from outside, for example using the URL *http(s)://myhost:<port>/mywebapp/**.

License Errors After Expiration Date

When the license expires, an error page shows when trying to start a new browser session. Sessions that are already running do not have any impacts. The Configuration Tool can still be used without any constraints.

15 Configuring SSL

General Information

Trust files are used for a secure connection between the Natural Web I/O Interface server and Natural for Ajax . Server authentication cannot be switched off. A trust file is always required.

A trust file contains the certificates that you trust. These can be certificates of a CA (Certificate Authority) such as VeriSign, or self-signed certificates.

For information on the steps that are required on the Natural Web I/O Interface server and how to generate a self-signed certificate which needs to be imported to the client, see *SSL Support* in *Installing and Configuring the Natural Web I/O Interface Server* which is part of the *Natural Web I/O Interface* documentation for your Natural platform .

To establish a secure connection, you have to proceed as described in the topics below.

Creating Your Own Trust File

To create your own trust file, you can use, for example, Sun's keytool utility which can be found in the *bin* directory of the Java Runtime Environment (JRE). Here are some helpful examples:

- Create an empty, password-protected trust file:

```
keytool -genkey -alias foo -keystore truststore.jks -storepass "your-passwort"
keytool -delete -alias foo -keystore truststore.jks
```

- Import a certificate:

```
keytool -import -alias "name-for-ca" -keystore truststore.jks -storepass ↩
"your-passwort" -file server.cert.crt
```

  You should use a meaningful name for the alias.

- List the certificates in a trust file:

```
keytool -list -v -keystore truststore.jks
```

- Delete a certificate from a trust file:

```
keytool -delete -alias "name-for-ca" -keystore truststore.jks
```

When you modify the trust file or its password, you have to restart the application server so that your modification takes effect.

Defining SSL Usage in the Configuration File

Invoke the **configuration tool** and proceed as follows:

1. In the global settings for all defined sessions, define the **SSL trust file path** and, if required, the **SSL trust file password** . See also ***Global Settings*** in *Natural Client Configuration Tool* .

   With the server authentication, Natural for Ajax checks whether the certificate of the Natural Web I/O Interface server is known. If it is not known, the connection is rejected.

   When a trust file is not defined in the configuration tool, Natural for Ajax tries to read the file *calist* from the *lib/security* directory of the Java Runtime Environment (JRE). The default password for this file is "changeit" .
2. Define a session and set the session option **Use SSL** to **Yes** . See also ***Overview of Session Options*** in *Natural Client Configuration Tool* .

16 Configuring Single Sign-On (SSO)

General Information

Natural for Ajax supports SSO with OpenID Connect. OpenID Connect is now the leading standard for single sign-on on the Internet. The user authentication is delegated to a central Identity Provider (IdP) service. This can be a public IdP like Google and Microsoft or an enterprise internal IdP.

When starting a Natural for Ajax application, Natural for Ajax will redirect the user to the IdP. In case the user is already logged in at the IdP, no login screen will appear. If the user is not yet logged in at the IdP, the IdP will open a corresponding login dialog. For Natural for Ajax it is absolutely transparent how the users are authenticated, it is all up to the IdP. In case of a successful authentication, Natural for Ajax will receive a valid token from the IdP. This token is used for the login to the Natural server.

Prerequisite is a Natural server environment, which supports SSO with OpenID Connect. For the corresponding Natural versions and configuration requirements see the Natural documentation.

As described above, a token is used for the login to the Natural server. Among other information, the token contains claims about the user such as email. In the Natural for Ajax **Session Configuration** you need to specify which of the claims should be used as a Natural username. This claim must be included in the token. Therefore, you may also want to configure which claims should be part of the token. These include the claims Natural for Ajax should request from the IdP and the details for how Natural for Ajax should request the claims. You can configure all this in the global settings of the session configuration. For details about claims please see the list of **Standart Claims** and the documentation of your IdP.

SSL/TLS Configuration

OpenID Connect requires the usage of SSL/TLS. You must provide a trustfile with a valid certificate for the connection between the Natural for Ajax client and the Natural Web I/O Interface server as described in the ***Configuring SSL*** documentation.

For OpenID Connect, you must also put the certificate of your IdP into the same trustfile.

Configuring OpenID Connect Usage

To configure the OpenID Connect:

1. Invoke the Natural for Ajax runtime tools and choose **Session Configuration**.
2. Define the OpenID Connect settings for your IdP in the global settings for all defined sessions.

You can only configure one IdP per Natural for Ajax application but the single sessions may be a mix of sessions which use OpenID Connect and sessions which don't use OpenID Connect.

To enable OpenID Connect for a single session:

1. Switch on **SSL** in the **Natural Connection** tab.
2. Switch on **OpenID Connect (OIDC)** in the **Authentication** tab.

Configuring your Identity Provider (IdP)

Your IdP requires you to set valid redirect URIs for your application. These are the allowed URIs to which a browser can redirect after a successful login at the IdP. In case the URI for your web application is https://mywebapplication you need to set the following URIs as valid redirect URIs in your IdP:

- https://mywebapplication/servlet/OpenidConnectRequest
- https://mywebapplication/servlet/OpenidConnectRequest?subpage=true

When using the Natural for Ajax subpage or workplace functionality, you also need to allow the following CORS origin in your IdP:

https://mywebapplication

17 Logging

General Information

Natural for Ajax uses the Java Logging API. In case of problems with Natural for Ajax , you can enable logging and thus write the logging information to an output file. This should only be done when requested by Software AG support.

You configure logging using the **configuration tool** .

Note: Some logging information is also written to the console, regardless of the settings in the configuration file. The console shows the information which is normally provided by the logging levels `SEVERE` , `WARNING` and `INFO` .

Name and Location of the Configuration File

The name of the configuration file is *natlogger.xml* . The path to this file depends on the application server. Example for JBoss Application Server:

`<application-server-install-dir>` */server/default/deploy/njx* `<nn>` *ra.rar/log*

Logging on JBoss Application Server

JBoss Application Server uses a different logging API (log4j). In this case, we recommend that you enable the file handler in the configuration file *natlogger.xml* .

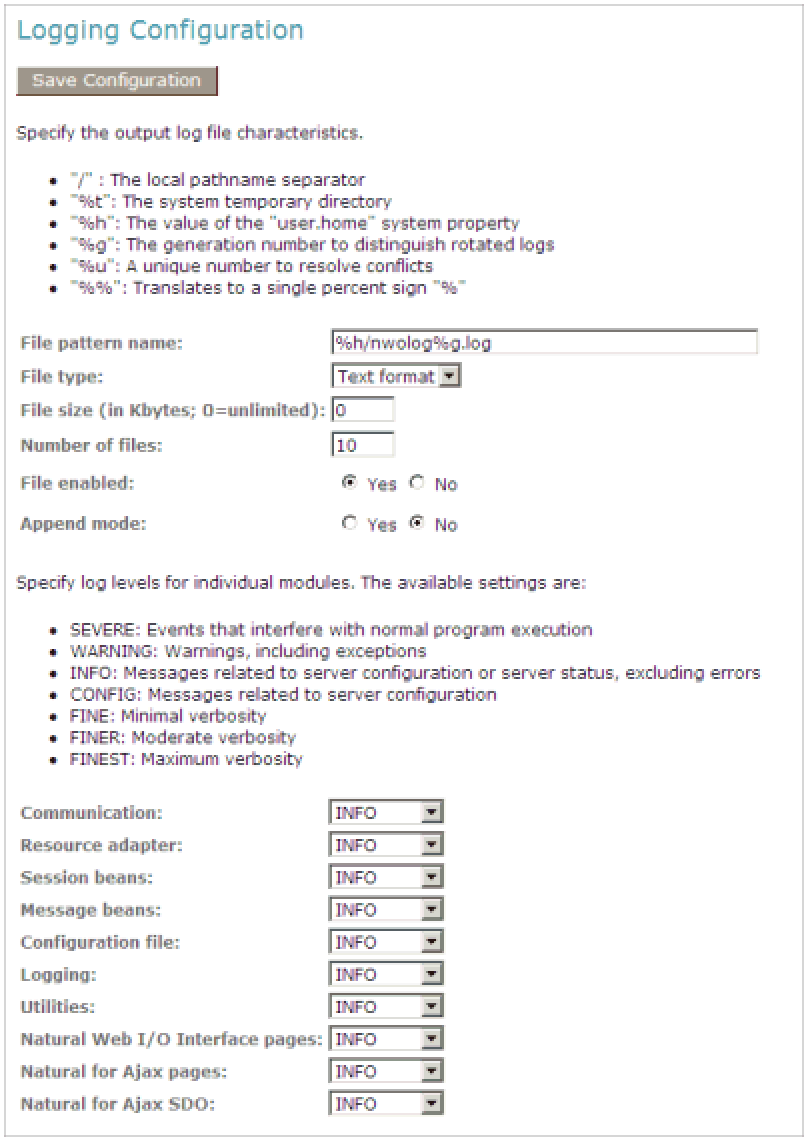
Invoking the Logging Configuration Page

The content of the configuration file *natlogger.xml* is managed using the **Logging Configuration** page of the **configuration tool** .

To invoke the Logging Configuration page

1 In the frame on the left, choose the **Logging Configuration** link.

The **Logging Configuration** page appears in the right frame. Example:



Logging Configuration

Specify the output log file characteristics.

- "/" : The local pathname separator
- "%t": The system temporary directory
- "%h": The value of the "user.home" system property
- "%g": The generation number to distinguish rotated logs
- "%u": A unique number to resolve conflicts
- "%%": Translates to a single percent sign "%"

File pattern name:	%h/nwolog%g.log
File type:	Text format
File size (in Kbytes; 0=unlimited):	0
Number of files:	10
File enabled:	● Yes ○ No
Append mode:	○ Yes ● No

Specify log levels for individual modules. The available settings are:

- SEVERE: Events that interfere with normal program execution
- WARNING: Warnings, including exceptions
- INFO: Messages related to server configuration or server status, excluding errors
- CONFIG: Messages related to server configuration
- FINE: Minimal verbosity
- FINER: Moderate verbosity
- FINEST: Maximum verbosity

Communication:	INFO
Resource adapter:	INFO
Session beans:	INFO
Message beans:	INFO
Configuration file:	INFO
Logging:	INFO
Utilities:	INFO
Natural Web I/O Interface pages:	INFO
Natural for Ajax pages:	INFO
Natural for Ajax SDO:	INFO

2 Specify the characteristics of the output file as described below in the section *Overview of Options for the Output File* .

3 Specify the log levels for individual modules by selecting the log level from the corresponding drop-down list box.

A brief description for each log level is provided on the **Logging Configuration** page.

4 Choose the **Save Configuration** button to write the modifications to the configuration file.

Caution: When you do not choose the **Save Configuration** button but log out instead or leave the configuration tool by entering another URL, your modifications are not written to the configuration file.

Overview of Options for the Output File

The following options are provided for specifying the characteristics of the output file:

Option	Description
File pattern name	The pattern for generating the output file name. Default: "%h/nwolog%g.log" . The default value means that an output file with the name *nwolog* `<number>` *.log* will be created in the home directory of the user who has started the application server. For detailed information on how to specify the pattern, see the Java API documentation .
File type	The format of the output file. Select one of the following entries from the drop-down list box: ■ **Text format** Output in simple text format (default). ■ **XML format** Output in XML format. The corresponding formatter class is then used.
File size	The maximum number of bytes that is to be written to an output file. Zero (0) means that there is no limit. Default: "0" .
Number of files	The number of output files to be used. This value must be at least "1" . Default: "10" .
File enabled	If set to **Yes** (default), the file handler is enabled. If set to **No** , the file handler is disabled.
Append mode	If set to **Yes** , the logging information is appended to the existing output file. If set to **No** (default), the logging information is written to a new output file.

18 Browser Configuration

JavaScript Enabling

Ajax 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.

Browser Caching

When working with Ajax 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 Ajax HTML pages remain stable in the browser. They do not contain any application data and are more comparable to small programs. The actual application data is filled into the pages dynamically at runtime. Also other files like JavaScript files are stable for each Natural Ajax version. If the browser is not allowed to cache them, the JavaScript files will be reloaded with each access of an Ajax HTML page.

The following sections contain recommendations on how to optimize browser caching:

Natural Ajax JavaScript Files

The Natural Ajax JavaScript framework only loads a few JavaScript files. In previous versions of Natural for Ajax, for different controls single JavaScript files were loaded. Especially with slow connection lines, the number of files influences the loading time of an Ajax page.

For the Natural JavaScript files it is important to have the correct version of the JavaScript files in the cache of the browser. The JavaScript files usually change with different Natural for Ajax versions. It is important to run the pages exactly with the JavaScript file version for which it was generated. In previous versions of Natural for Ajax the only solutions were:

1. Letting the browser cache the files and then make sure that the browser cache is cleared when the Natural for Ajax version is upgraded.
2. Instructing the browser to always ask the server if the JavaScript file has changed before using the cached file – by setting a corresponding HTTP header in the server or the web application.

Both solutions were not optimal. The first solution required the end-users to always make sure that their cache is cleared. The second solution does not allow for optimal performance because at least a corresponding server request for the last modified date of the file had to be done.

Instead of the previous approach, Natural for Ajax uses versioned JavaScript for the JavaScript of the frequently used controls. These files now have the build version of the *cisversion.xml*.

Example

- `ciscentralCIS_V`*`vrs_YYYYMMDD_HHMM`*`_NJX.js`
- `cisbasicCIS_V`*`vrs_YYYYMMDD_HHMM`*`_NJX.js`
- `cisadvancedCIS_V`*`vrs_YYYYMMDD_HHMM`*`_NJX.js`

Where *`vrs`* is the Natural for Ajax version used (e.g. 841) and *`YYYYMMDD`* and *`HHMM`* represent the date and timestamp.

It is not required to add corresponding HTTP headers to check for newer versions of *ciscentral*.js*, *cisbasic*.js* and *cisadvance*.js*. Every Ajax page will request exactly the JavaScript file for which it is generated. Browser and server can be configured so that these JavaScript files are cached and always accessed from the cache without any additional request to the server.

Ajax HTML Files and Styles

The Ajax HTML files and styles for an Ajax page are generated files from corresponding source files (**.xml*, **.info*). During development you may want to reload them whenever the sources are modified, which can be every few seconds. In a production environment, you usually want to reload them whenever a new version of your web application is released in your production environment.

In a development environment we recommend to set the HTTP header to:

```
Cache-Control: max-age=0, must-revalidate
```

Alternatively, you can opt for not allowing any caching at all:

```
Cache-Control: no-cache
```

In a production environment we recommend to set an HTTP header such that the client only checks the age of these files every hour:

```
Cache-Control: max-age=3600
```

This results in all clients being forced to load the new version with a maximum delay of an hour after upgrading your production environment. You can of course still force reload earlier by clearing the browser cache manually for test purposes.

Images

In contrast to the Ajax HTML files and styles, images are not generated and usually do not change so frequently. You might think about allowing an age higher than that granted for the generated files but in most applications this does not have advantages. The recommendation is to use the same settings as for the generated files described above.

Setting HTTP Headers in web.xml

The HTTP headers recommended above can be set at different places. They can be set in the configuration files of web applications and web application servers. They can also be set in the *web.xml* file and then apply exactly to this web application. The Natural Ajax framework contains a ready to use `HttpHeader` filter which you can easily configure in your *web.xml* file. Search for `HttpHeader` in the *web.xml* file and you should find commented configuration tags, which you can use.

Example

```
<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=3600</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>
<filter-mapping>
    <filter-name>HttpHeader</filter-name>
    <url-pattern>*.html</url-pattern>
</filter-mapping>
...
```

Internet Explorer

Internet Explorer supports to switch off caching via settings in the browser itself. You need to make sure that the browser is configured so that it allows caching at all.

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**.

Pop-Up Blocker

The default browser setting in most browsers is to block pop-ups.

By default, Natural Ajax pop-ups are opened as page pop-ups, instead of browser pop-ups. See the attribute `usepagepopups` in the ***cisconfig.xml***.

Only some controls like the `DATEINPUT` controls use browser pop-ups. There is a newer `DATEINPUT2` control, which does not use browser pop-ups. If you use controls requiring browser pop-ups in your application, you need to switch off the browser's pop-up blocker. You can find the corresponding setting in the **Security** tab of the browser options.

Browser Standards Mode and HTML5

General Information

Natural for Ajax applications up to version 8.3.4 were running in quirks mode in the browsers, which does not support HTML5 and CSS3 and which behaves differently in the different browsers. With Internet Explorer 11, Firefox and Chrome, Natural for Ajax applications now run in standards mode, which supports HTML5 and CSS3. In standards mode, the browsers should behave as described by the W3C HTML and CSS specifications.

The following sections explain what you need to do when switching to standards mode and HTML5 in the browsers.

Enabling Standards Mode in the Browser

The HTML pages that are generated with Natural for Ajax contain the following declaration:

```
<!DOCTYPE html>
```

This tells the browser to run in standards mode. If you have not defined specific configurations settings in your browser, you do not have to do anything. The browser will automatically use the correct mode.

Upgrading an Application to HTML5

There are differences between HTML4 and HTML5. Page layouts are written in XML and Natural for Ajax takes care of the correct generation into HTML5. Therefore, you do not need to adapt anything in your layouts in most cases. You only have to do the following: Regenerate the HTML of your layout pages and the **.css* files of your application.

When using NaturalONE this is automatically done when you rebuild your projects: When packaging your application as a *.war* file using a NaturalONE *wardeploy.xml* file, the HTML and CSS files are automatically regenerated. The prerequisite for this is a *wardeploy.xml* file that has been generated with version 8.3.4 or above.

If you are not using the NaturalONE *wardeploy.xml* files you can do the regeneration in your Natural Ajax production or test environments using command line jobs.

You need to check whether your implementation is HTML5/CSS3-compliant if you are using the following advanced features:

Style Sheet Settings

Some style settings have changed in CSS3. One major change is that for attributes such as `height`, `width` and `padding`, a number only is no longer a valid value. The value must now also include a unit such as "px", "cm" or "%", or it must be one of the predefined values.

For your **.info* files and if your application is using its own **.css* files, we recommend that you check at least whether "px" is properly applied to the corresponding attribute values.

IHTML Controls

In IHTML controls, the Natural programs provide plain HTML at runtime. We recommend that you check whether this plain HTML is HTML5/CSS3-compliant.

*style and *styleprop Properties

Many controls support properties such as `textstyle` and `textstyleprop` to directly set CSS attributes at design time and/or runtime. In attributes such as `height`, `width` and `padding`, any missing "px" units are automatically applied by the Natural for Ajax framework. You do not need to take care of this. In rare cases, you might want to check for attributes which are no longer supported with CSS3.

Custom Controls

If you have built your own macro controls, changes are usually not required. For custom controls, however, for which you generate your own HTML, you need to check whether the generated HTML is HTML5-compliant.

Upgrading a Test Environment to HTML5

According to the HTML5 standard, all custom attributes must start with the string "data-". For this reason, the Natural for Ajax framework generates the attribute `data-testtoolid` into the HTML files by default.

In earlier versions, this attribute was called `testtoolid`. In the layout XML, the property name `testtoolid` is kept - you need not change any layouts. This is just the default for the attribute in the HTML which is changed to `data-testtoolid`. If you are using this attribute in automated tests, you need to change your tests accordingly.

As an alternative solution, the Natural for Ajax framework still supports the `testtoolid` attribute. This allows you to perform the upgrade step by step: You can first upgrade your application without touching the test suite. When this is stable, you can adapt your test suite. The attribute `testtoolid` does not adhere to the naming convention of the HTML5 specification. Therefore, the resulting HTML will not be 100% valid HTML5. But the currently supported browsers still accept this attribute.

If you want the Natural for Ajax framework to generate the attribute `testtoolid` in the HTML files instead of the default `data-testtoolid`, do the following:

1. In the *cisconfig.xml* file, set the parameter `testtoolidhtml4` to "true".
2. Regenerate the HTML files for your layouts.
3. Make sure that your test environment also contains a *cisconfig.xml* file in which `testtoolidhtml4` is set to "true".

Note: There is no guarantee that future browser versions will still tolerate the `testtoolid` attribute. You may have to switch to `data-testtoolid` sooner or later, but you do not have to do it immediately.

Upgrading a Production Environment to HTML5

If your application needs to support Internet Explorer 10, or if you need to do any compatibility settings in Internet Explorer to run other applications, you have to add the following entries to the *web.xml* file in your production environment:

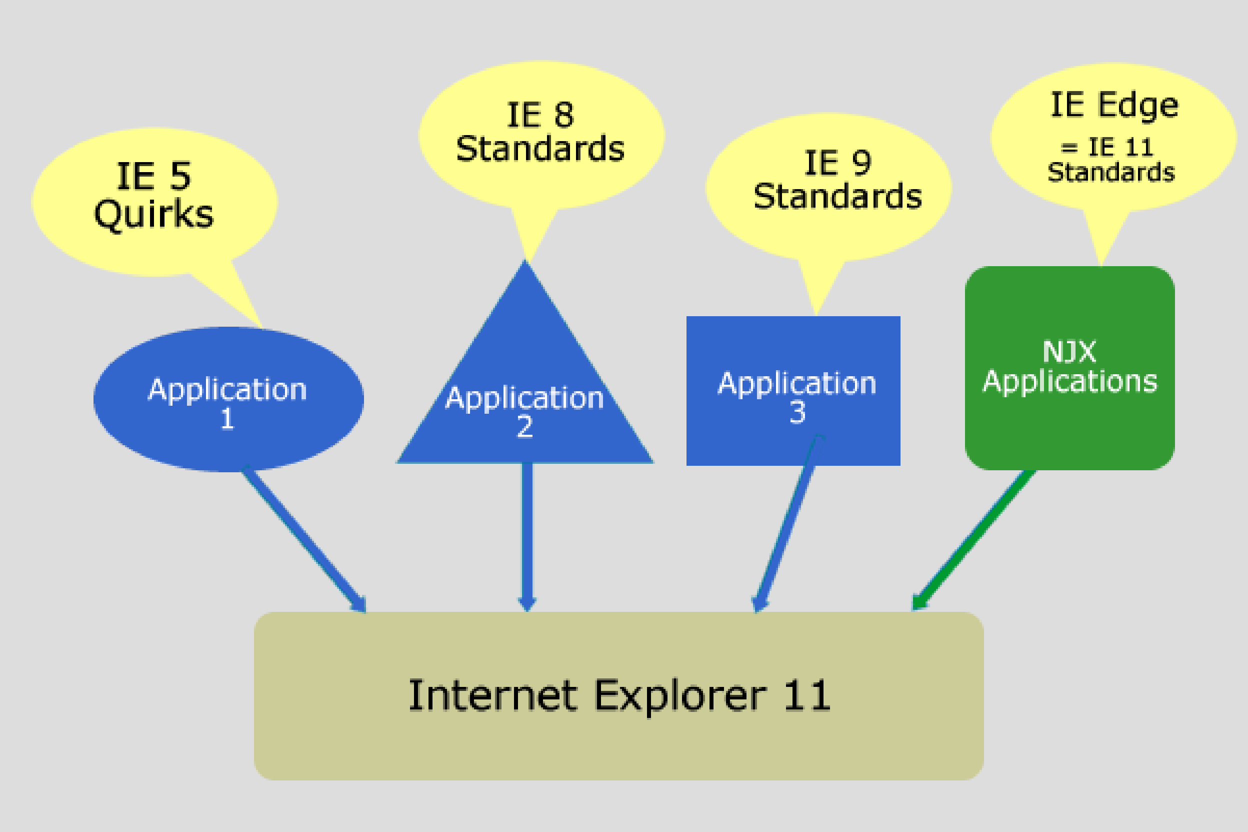
```
<filter>
   <filter-name>BrowserCompatibility</filter-name>
   <filter-class>
      com.softwareag.cis.server.filter.BrowserCompatibilityFilter
   </filter-class>
</filter>
<filter-mapping>
   <filter-name>BrowserCompatibility</filter-name>
   <url-pattern>*.html</url-pattern>
   <servlet-name>StartCISPage</servlet-name>
</filter-mapping>
<filter-mapping>
    <filter-name>BrowserCompatibility</filter-name>
    <servlet-name>StartDynamicPage</servlet-name>
</filter-mapping>
```

For an example, see the *web.xml* file that is shipped with Natural for Ajax.

Mastering Internet Explorer Browser Modes

- Applications Setting the Required Browser Mode
- Applications not Setting the Required Browser Mode

You are running Internet Explorer 11 (IE11) because it provides up-to-date security. But you have different applications with different needs regarding the browser mode. IE11 supports all these browser modes, but someone has to tell the browser which application should run in which mode.



Applications Setting the Required Browser Mode

In an ideal world all applications tell IE11 which mode they require and all applications are rendered correctly. In this case: Do not configure any compatibility settings in your IE11.

Natural for Ajax applications tell the browser, which mode they require. You only have to add the `BrowserCompatibility` filter in your production or test environments as described in *Upgrading a Test Environment to HTML5* and *Upgrading a Production Environment to HTML5* above.

Applications not Setting the Required Browser Mode

If some of your applications do not set the required browser mode automatically but expect some specific mode:, you have the following options.

Configuring your Application and/or Web Server/Web Container

Applications can tell IE11 in which mode they require to run by setting the HTTP header "X - UA - Compatible". For more information see **https://msdn.microsoft.com/en - us/library/ff955275%28v=vs.85%29.aspx**.

If these applications are deployed in a web server/web container that is under your control, it is possible to configure the HTTP response header, which is sent for these applications.

In this case: Do not configure any compatibility settings in your IE11.

Configuring your Internet Explorer 11

In case you do not have a chance to configure the application and/or its web server/web container, IE11 supports an "Enterprise Modus", see **https://msdn.microsoft.com/de - de/library/dn640687.aspx**.

19 Timeout Configuration

In Natural for Ajax applications, timeouts between the browser and the web application as well as timeouts between the web application and the Natural server can be configured.

Timeout Between the Browser and the Web Application

Most users keep their browser open with several applications running in multiple browser tabs for many hours or even days. More often than not, they do not remember which of the applications are still running in one of the browser tabs. For security reasons and to avoid resource bottlenecks, web applications should therefore timeout after a certain period of inactivity of the user.

By setting `sessiontimeout` in the *Ajax Configuration* you can configure such a timeout.

Timeout Between the Web Application and the Natural Server

Between the web application and the Natural server each user of your application usually has one or several Natural connections active. There are two situations in which you would like to close these connections automatically:

- Situation A
- Situation B

Situation A

When the web application does not receive an appropriate answer for a request from the Natural server in a defined time, thereby indicating that something could be wrong with

- either the server side execution,
- the Natural server or
- the connection to the Natural server.

Using the parameter **Timeout (in Seconds)** the timeout for this situation can be configured for each session individually. For details, refer to the description of this parameter in the table under *Overview of Session Options* in section *Session Configuration*.

Situation B

When the Natural server does not receive any more requests from the web application for a defined time, thereby indicating that something could be wrong with

- either the web application or
- with the connection from the web application to the Natural server
- or the user unintendedly did not close/end the application.

Using the global setting **Last activity timeout (in seconds)** the timeout for this situation can be configured for all sessions globally. For details, refer to the description of this setting in *Global Settings* under *Session Configuration*.

Dependencies Between Timeouts

We recommend, that you use the same value for the **Last activity timeout** (as in **Situation B**) and the `sessiontimeout` in the Ajax configuration.

The **Last activity timeout** covers a subset of situations to which the `sessiontimeout` applies. The `sessiontimeout` also applies to "non-Natural" Ajax pages such as workplace applications.