Natural for Ajax

Ajax Developer

Version 9.3.4

October 2026

ADABAS & NATURAL

This document applies to Natural for Ajax Version 9.3.4 and all subsequent releases.

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

Copyright © 2009-2026 Software GmbH, Darmstadt, Germany and/or its subsidiaries and/or its affiliates and/or their licensors.

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

Detailed information on trademarks and patents owned by Software GmbH and/or its subsidiaries is located at https://softwareag.com/licenses.

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

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software GmbH Products / Copyright and Trademark Notices of Software GmbH Products". These documents are part of the product documentation, located at https://softwareag.com/licenses and/or in the root installation directory of the licensed product(s).

Use, reproduction, transfer, publication or disclosure is prohibited except as specifically provided for in your License Agreement with Software GmbH.

Document ID: AT-CIT-AJAXDEV-934-20261004

Table of Contents

Preface vii
1 About this Documentation 1
Document Conventions 2
Online Information and Support 2
Data Protection 3
I Getting Started with Ajax Developer 5
2 Getting Started with Ajax Developer 7
About the Runtime Environment 8
Runtime Environment Updates 8
Setting the Ajax Developer Preferences 10
Working with Projects 12
Working with User Interface Components 17
Using Ajax Configuration Files 25
Working With Version Control Systems 26
Using the Ajax Developer Perspective 28
Developing Java Extensions 28
II Layout Painter 31
3 Getting Started with the Layout Painter 33
General Information 34
Creating a Layout 34
Layout Templates 35
Opening an Existing Layout 36
Elements of the Layout Painter 36
Creating Custom Layout Templates 38
4 Defining the XML Layout 43
Previewing the Layout 44
Setting the Properties for a Control 46
Changing the Appearance of Text 48
Adding Controls to the Layout 49
Creating a Folder for Images 51
Defining a Background Image 51
Defining a Style Sheet 52
Defining Hot Keys 53
Managing the Nodes in the Layout Area 54
Managing the Controls in the Preview Area 57
Viewing the XML Code 59
Saving the Layout 60
Layout Tester 61
5 Using the Code Assistant 63
About the Code Assistant 64
Code Template 64
Opening the Code Assistant 64
Code Introspection 65

Generating the Code ... 66
Editing the Adapter Code ... 67
Saving Changes ... 68
Changes from the Outside ... 69
6 Using the Literal Assistant ... 71
About the Literal Assistant ... 72
Opening the Literal Assistant ... 72
Maintaining Literals ... 73
Selecting Another Language ... 74
Displaying a Comparison Language ... 74
7 Using the Validation Rules Editor ... 77
About the Validation Rules Editor ... 78
Opening the Validation Rules Editor ... 78
Adding a New Validation Rule ... 79
Displaying an Overview of All Defined Validation Rules ... 81
8 Using the Formula Editor ... 83
About the Formula Editor ... 84
Opening the Formula Editor ... 84
Adding a New Formula ... 85
Displaying an Overview of All Defined Formulae ... 87
III ... 89
9 Language Manager ... 91
About the Language Manager ... 92
Invoking the Language Manager ... 92
Defining a New Language ... 93
Opening an Existing Language ... 95
10 Literal Translator ... 97
About the Literal Translator ... 98
Invoking the Literal Translator ... 98
Loading the Literals of a Layout ... 100
Editing the Literals ... 101
Adding a New Text ID ... 101
Removing Text IDs ... 102
11 Style Sheet Editor ... 103
About the Style Sheet Editor ... 104
Developing Style Sheets in Your User Interface Component ... 104
Creating a New Style Sheet ... 104
Opening an Existing Style Sheet ... 106
Changing a Style Sheet ... 107
Overview of Variables ... 109
Applying your Stylesheet to Pages ... 110
Regenerating Your Own Style Sheet from the Style Sheet Template ... 111
Using a Version Control Tool ... 111
IV Control Editor ... 113
12 Using the Control Editor ... 115

Invoking the Control Editor 116
Creating an Editor Extension 117
Adding a Control to an Editor Extension 119
Adding a Data Type to an Editor Extension 120
Deleting a Control or Data Type 123
Saving an Editor Extension 123
Opening an Editor Extension 124
13 Defining a Control 125
Attributes 126
Positioning 127
XML 129
XML Defaults 130
Protocol Extension 131
14 Examples 133
About the Examples 134
Defining a Control with a Corresponding Tag Handler 134
Defining a Macro Control 137
Additional Information 146
V Runtime Tools 147
15 Runtime Tools 149
VI Server Logs Viewer 151
16 Server Logs Viewer 153
About the Server Logs Viewer 154
Preconfigured Server Logs 154
Advanced Usage 156
VII Bootstrap Icons 163
17 Bootstrap Icons 165

Preface

This documentation is organized under the following headings:

Getting Started with Ajax Developer	About the runtime environment. How to enable Ajax Developer for a project and how to create user interface components (that is, the folders containing the layouts). An overview of the files and folders that need to be checked in when using a version control system.
Layout Painter	How to define page layouts with the Layout Painter. This also includes the tools which are available when the Layout Painter is active: Code Assistant, Literal Assistant, Validation Rules Editor and Formula Editor.
Language Manager	How to define additional languages.
Literal Translator	How to translate text IDs.
Style Sheet Editor	How to create style sheets.
Control Editor	How to build your own controls.
Runtime Tools	How to administrate the Ajax Runtime in NaturalONE.
Server Logs Viewer	How to use the server logs viewer.
Bootstrap Icons	How to use the Bootstrap Icon library.

Note: For information on the Conversion Rules tool and the Conversion Logs tool, see the *Natural for Ajax* documentation.

1 About this Documentation

- Document Conventions 2
- Online Information and Support 2
- Data Protection 3

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://hub.docker.com/publishers/softwareag** 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 AG 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.

I Getting Started with Ajax Developer

2 Getting Started with Ajax Developer

- About the Runtime Environment 8
- Runtime Environment Updates 8
- Setting the Ajax Developer Preferences 10
- Working with Projects 12
- Working with User Interface Components 17
- Using Ajax Configuration Files 25
- Working With Version Control Systems 26
- Using the Ajax Developer Perspective 28
- Developing Java Extensions 28

This documentation describes the Ajax Developer tools that are available in the Eclipse environment for creating and maintaining complex graphical user interfaces.

About the Runtime Environment

Ajax Developer requires a web container such as Tomcat as its runtime environment.

If NaturalONE is installed, an Apache Tomcat is already included and all required configuration has automatically been done. In this case, you can immediately enable a project for Ajax Developer (see below).

If NaturalONE is not installed, you must first enter the settings for your own local web container environment in the Ajax Developer preferences. For further information, see ***Setting the Ajax Developer Preferences***.

You can customize memory and port settings of the Tomcat in the file *<Installation Directory>/naturalone/njx/tomcat.properties.*

Property name	Value	Explanation
host	localhost	Do not change this.
minconnectorport	28080	The Tomcat tries to find a free port starting with this value. You can change this vaue. If you change it, you usually must also change the **minshutdownport**.
minshutdownport	28005	Tomcat shutdown port corresponding to **minconnectorport**.
numconnectorports	20	Number of ports the Tomcat will try to find a free port.
numshutdownports	20	Corresponding to **numconnectorports**.
webcontext	cisnatural	Do not change this.
javaopts	Example value: -Xmn128m -Xms512m -Xmx1024m	Java Options for the Java VM running in the Tomcat. If you have really huge Ajax layouts you may want to increase the memory settings.

Runtime Environment Updates

NaturalONE updates may include updates to the runtime environment. Runtime environment updates can be:

- Tomcat updates: the folder *<Installation Directory>/naturalone/apache-tomcat* will be updated.
- Natural Ajax Runtime updates: the file *<Installation Directory>/naturalone/njx/naturalajax.war* will be updated.

- Tomcat Updates
- Natural Ajax Runtime Updates

Tomcat Updates

The Tomcat version of the NaturalONE installation must always match the Tomcat version of an Eclipse workspace.

When you start NaturalONE after an update, the workspace will be updated correspondingly. Per default a dialog will pop up to confirm the workspace update. You must confirm the update if you want to work with the workspace.

If you never want to be prompted to confirm the workspace update:

- Open the file *<Installation Directory>/naturalone/eclipse/configuration/.settings/org.eclipse.ui.ide.prefs* in a text editor
- Add as the last line: WARN_ABOUT_WORKSPACE_INCOMPATIBILITY=false

Now after every NaturalONE upgrade, your workspace will also be automatically updated without bothering you with dialogs.

Natural Ajax Runtime Updates

In the same NaturalONE installation you can have multiple workspaces with different Natural Ajax Runtime versions. When you start NaturalONE after an update, a dialog will pop-up to ask you if the Natural Ajax Runtime should be updated in the workspace. Making this upgrade is optional. However, we strongly recommend updating the workspace because newer Natural Ajax Runtime versions also contain fixes for security vulnerabilities.

You can customize the update behavior by setting the property **upgradenjx** in the file *<Installation Directory>/naturalone/njx/tomcat.properties:*

Value	Explanation
always	The Natural Ajax Runtime in the workspace will automatically be updated if required without bothering you with dialogs
never	The Natural Ajax Runtime in the workspace will not be updated and no dialog will pop up
ifnewer	The Natural Ajax Runtime in the workspace will only be updated if the version in NaturalONE is newer than the version in the workspace.

Note: If a workspace has been updated, you need to rebuild your projects.

Setting the Ajax Developer Preferences

The web application configuration in the Ajax Developer preferences is normally only required if you use your own local web container environment, that is, when NaturalONE is not installed.

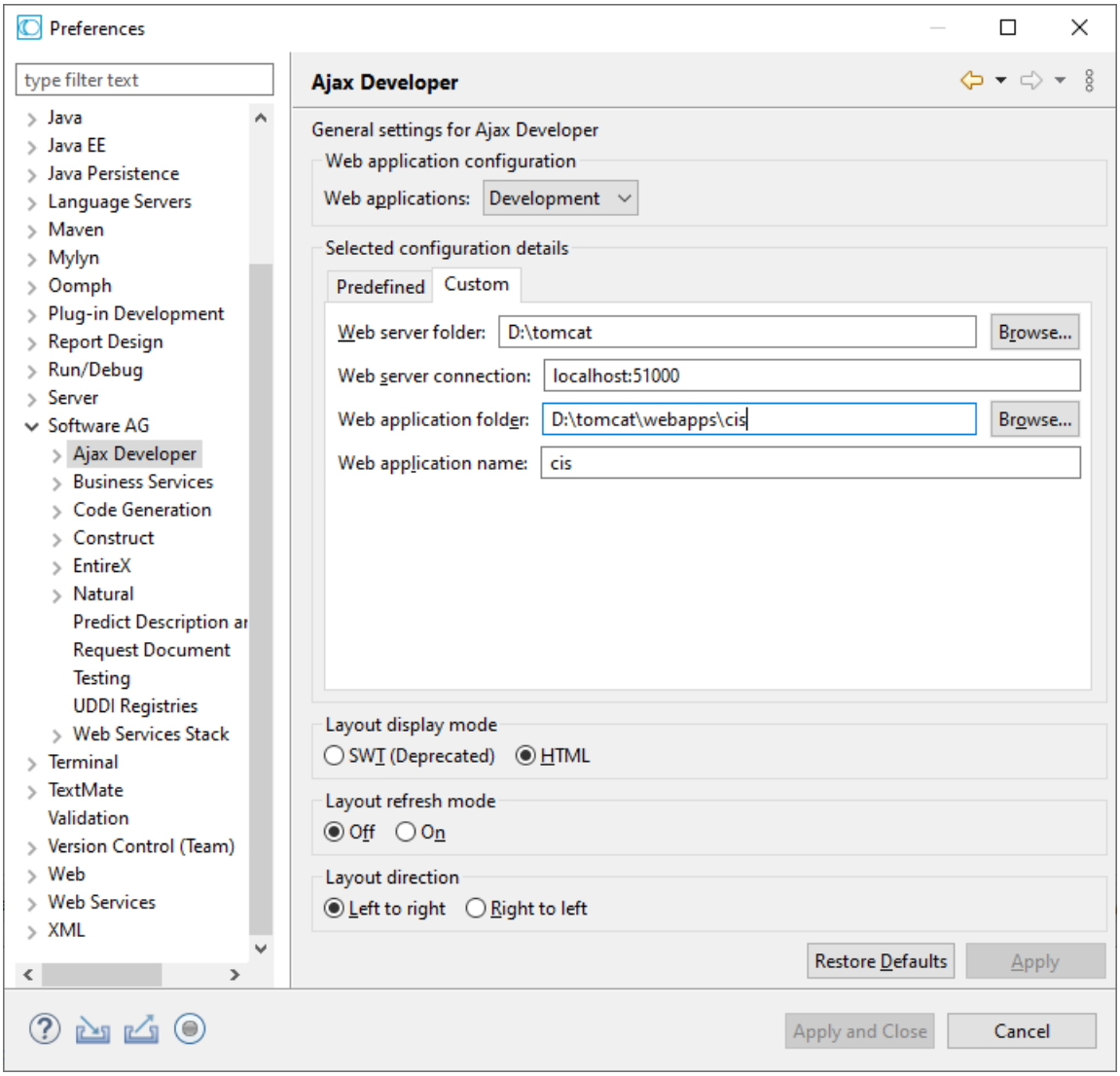
To set the Ajax Developer preferences

1 From the **Window** menu, choose **Preferences**.

2 In the tree of the resulting dialog box, expand the **Software AG** node and then select the **Ajax Developer** node.

3 If you want to use your own web container environment, choose the **Custom** tab on the resulting **Ajax Developer** page and specify all required information. For information on the available options, see *Enabling a Project for Ajax Developer*.

   It is important that you set the web application folder to `<webcontainer-installdir>`/*webapps*/*cis*. Example:



4 The options under **Layout display mode** allow you to control how the page layouts are rendered by default in the Layout Painter preview window. If you choose **SWT**, the layout will be rendered by default as SWT (Standard Widget Toolkit) pages, that is, with Java means. If you choose **HTML**, the layout will be rendered by default as HTML pages.

> **Note:** You can also change the display mode directly in the Layout Painter, using a menu command. See *Changing the Display Mode*.

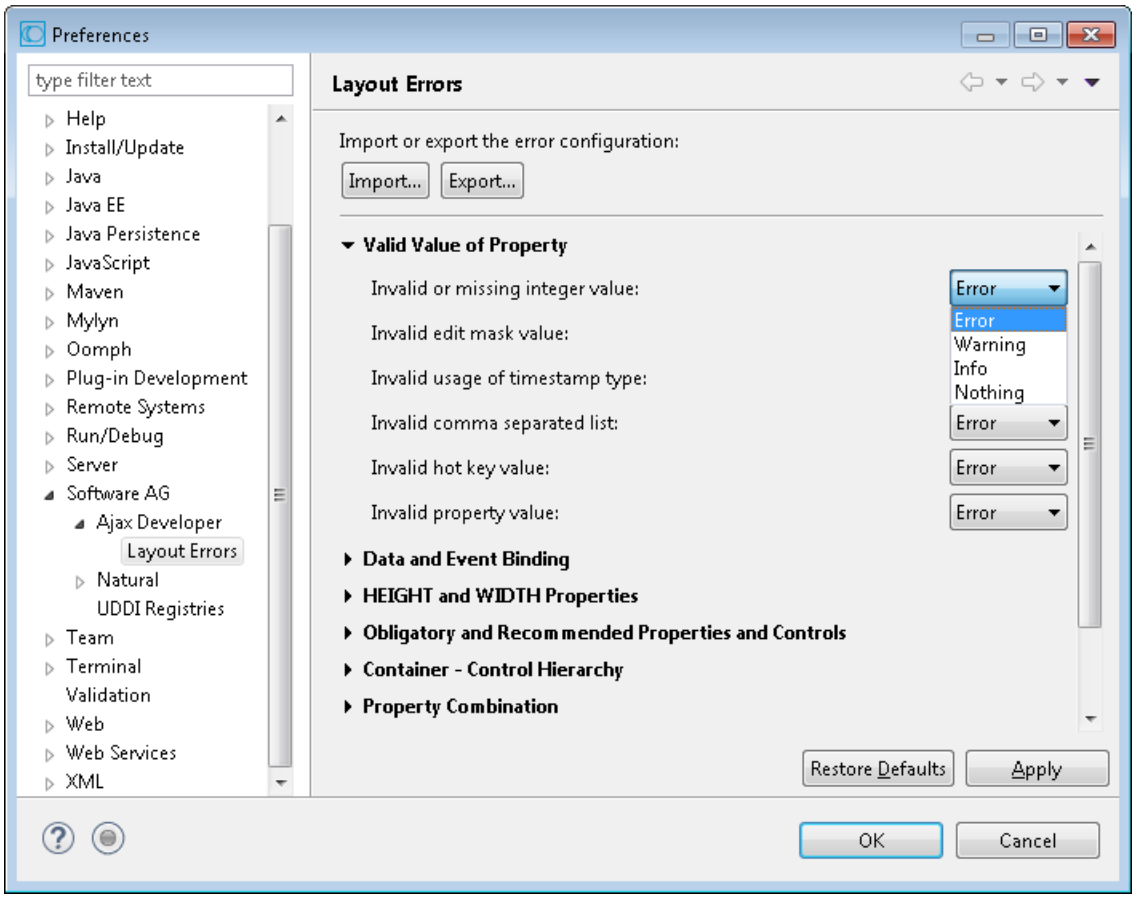
5 The options under **Layout refresh mode** allow you to enable or disable automatic refreshing of the Layout Painter's preview area.

6 The options under **Layout direction** allow you to design your page layout in either left-to-right (LTR) or right-to-left (RTL) direction. The specified direction is then used in the preview area of the Layout Painter.

7 Choose the **OK** button to save your changes and to close the dialog box.

Setting the Ajax Developer Preferences for Layout Errors

You can customize the layout errors and warnings in the following preference page.



For more information on the problem groups see *Configuration of Page Layout Errors/Warnings*.

To keep your own settings when creating a new workspace you need to export the settings and import them in the new workspace.

Working with Projects

The following topics are covered below:

- Types of Projects
- Enabling a Project for Ajax Developer
- Changing the Ajax Developer Properties for a Project

- Disabling a Project for Ajax Developer

Types of Projects

With Ajax Developer, page layouts can be developed in Natural or Java projects. You create your Natural or Java project as usual and then enable this project for Ajax Developer. When a project has been enabled for Ajax Developer, the settings for a specific web container environment are applied to the project. These settings are used to develop and run page layouts.

You can have multiple projects which are enabled for different web container environments.

In a NaturalONE installation, you will usually always enable a project for the preconfigured Tomcat environment. This is the so-called "Development" environment which is automatically installed and configured with NaturalONE.

When NaturalONE is not installed, you have the following choices for the location of your Java projects:

- Choose a project location outside the web container environment. This is recommended for development, especially for working with version control systems such as CVS or Subversion.
- Choose the web application root folder named *cis* as the project location. This is only recommended for testing and debugging layouts in a production environment or test environment.

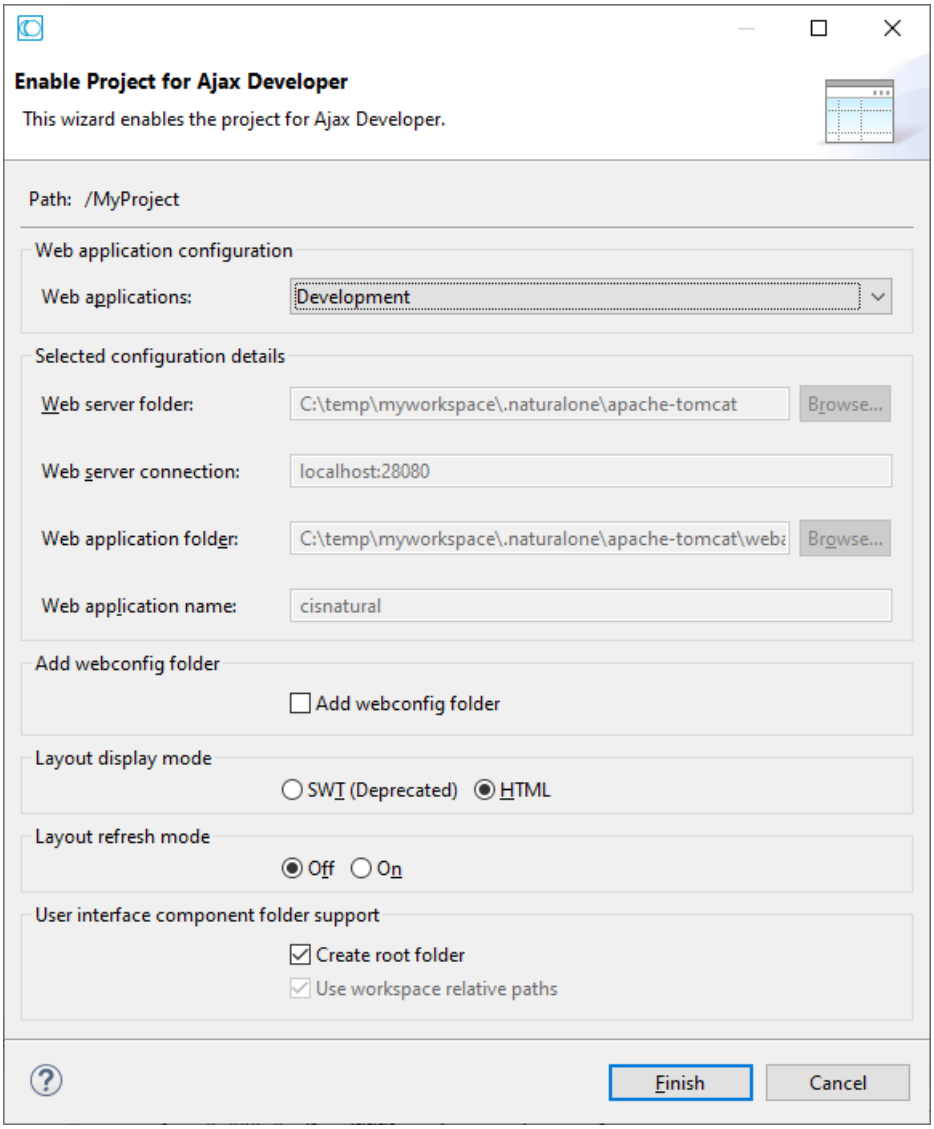
Enabling a Project for Ajax Developer

Before you can define layout pages for a project, you have to enable the project for Ajax Developer.

To enable a project for Ajax Developer

1. In the **Project Explorer** view, select the project that you want to enable.
2. Invoke the context menu and choose **Enable for Ajax Developer**.

   The following dialog box appears:



3 Specify the following information:

Web applications

Select one of the following options from the drop-down list box:

- **Development**
  If NaturalONE is installed, select this option. No further steps are required (the configuration options in this dialog box are disabled in this case).

  Note: This option is only available when NaturalONE is installed.

- **Custom**
  If NaturalONE is not installed, select this option. In this case, the configuration options in the dialog box are enabled and you can define your own settings for your web container environment. See below.

Web server folder
The root folder of the local web container environment.

Web server connection
Host name and port number of the local web container environment.

Web application folder
The folder of the Application Designer web application. For NaturalONE installations, this is the root folder of the *cisnatural* web application. If NaturalONE is not installed, this is the root folder of the *cis* web application (`<webcontainerinstalldir>`/*webapps*/*cis*).

Web application name
For NaturalONE, this is *cisnatural*. If NaturalONE is not installed, this is *cis*.

Layout display mode
Select the display mode for the Layout Painter and Layout Tester:

- **HTML**
  This mode makes use of the ActiveX plug-in of Eclipse in which Internet Explorer is running.
- **SWT**
  This mode makes use of SWT controls which are shown in an SWT client.

Layout refresh mode
Enable or disable automatic refreshing of the Layout Painter's preview area:

- **Off**
  The preview area is not automatically refreshed when the layout is changed. It is only refreshed when the changes are saved. It is recommended that you use this mode if you are working with complex controls such as ROWTABAREA.
- **On**
  The preview area is automatically refreshed when the layout changes, even if the changes have not yet been saved.

Create root folder
When selected, all user interface components of the project are created within a folder which has the fixed name "User-Interface-Components".

When not selected, all user interface components of the project are created directly in the project directory.

When NaturalONE is installed, the default for this setting can be defined under **Window > Preferences > Software AG > Natural**.

Use workspace relative paths
This option is read-only.

In a NaturalONE installation, the **Development** web application is part of your Eclipse workspace. Ajax Developer automatically uses relative paths to access this web application. When relative paths are used, the workspace can be renamed or moved on the file system.

If you are not working with NaturalONE and you are using the **Custom** web application, the web application is usually not located within your Eclipse workspace. In this case, Ajax Developer uses absolute paths. If you move your web application or your Tomcat on the file system, you must adapt the settings.

4 Choose the **Finish** button.

The icon for the project changes to indicate that Ajax Developer has been enabled for the project.

The context menu for the project now provides the additional cascading menu **Ajax Developer** which provides for selection a number of tools which are helpful when defining layouts.

Changing the Ajax Developer Properties for a Project

If you are working with NaturalONE, you can change the layout display mode that is defined for the project, for example, from SWT mode to HTML mode. If NaturalONE is not installed, you can also change the settings for your web container environment.

To change the properties

1 In the **Project Explorer** view or in the **Natural Navigator** view, select the project which has been enabled for Ajax Developer.

2 Invoke the context menu and choose **Properties**.

3 In the tree of the resulting dialog box, select **Ajax Developer**.

The resulting page contains the same information as the dialog box which appears when you enable a project for Ajax Developer. See *Enabling a Project for Ajax Developer* for further information.

Disabling a Project for Ajax Developer

If you decided to **remove all user interface components** from a project, it makes sense to entirely disable this project for Ajax Developer. Then, the Ajax Developer functionality is no longer available for the project.

The user interface components are not deleted physically from the disk. Thus, it is possible to re-enable the same project for Ajax Developer later.

To disable a project

1 In the **Project Explorer** view, select the project that you want to disable.

2 Invoke the context menu and choose **Ajax Developer > Disable**.

Working with User Interface Components

The following topics are covered below:

- Creating User Interface Components
- File Structure in a User Interface Component
- Changing the Properties of a User Interface Component
- Global User Interface Components
- Exporting User Interface Components
- Removing User Interface Components
- Cleaning User Interface Components
- Building User Interface Components
- Refreshing User Interface Components
- Executing and Debugging User Interface Components

Creating User Interface Components

After a project has been enabled for Ajax Developer, you can add user interface components. A user interface component is a container for a set of layout pages and additional artifacts such style sheets, images or language files.

A user interface component is also connected to the corresponding Natural or Java sources which implement the business logic. A single Java or Natural project may contain several user interface components.

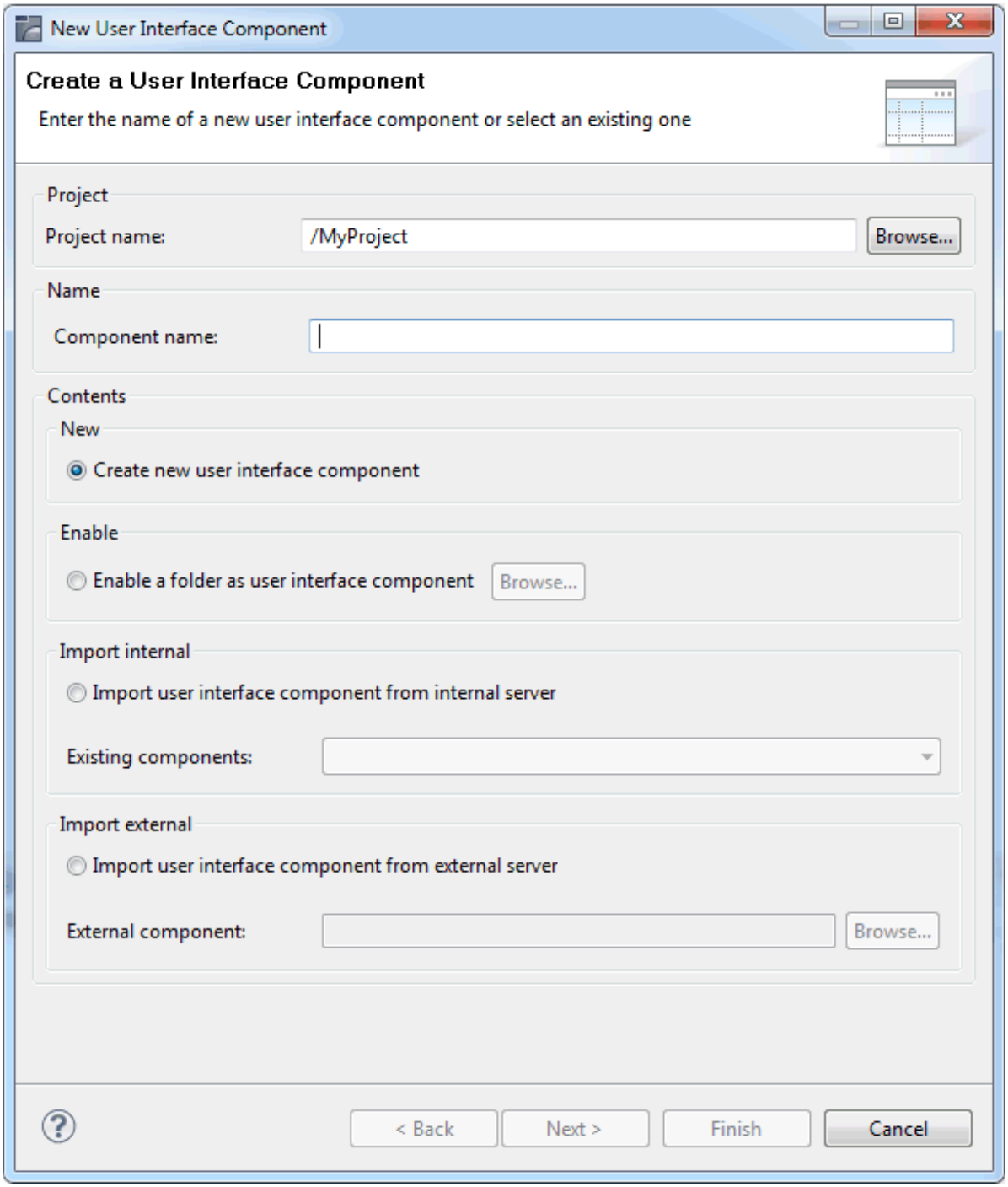
The name of the user interface component must be unique across the Eclipse workspace.

To create a user interface component

1 In the **Project Explorer** view or in the **Natural Navigator** view, select the project in which you want to create a user interface component.

2 From the **File** menu or context menu, choose **New > User Interface Component**.

The following dialog box appears:



3 In the **Component name** text box, enter a name for your user interface component (for example "MyFirstUI").

4 Select one of the following option buttons:

Create new user interface component

This option creates a new user interface component from scratch. The location of the new user interface component is a subfolder of your Natural or Java project folder.

Enable a folder as user interface component

This option converts an existing subfolder of the project to a user interface component folder. In this case, you have to choose the **Browse** button to select the folder to be converted from a dialog box.

The internal web container environment is then used for further development.

Import user interface component from internal server

This option moves an existing Application Designer project from the internal web container environment to your Eclipse workspace. The internal web container environment is the environment configured for this Natural or Java project. In this case, you have to select a component from the **Existing components** drop-down list box.

In a NaturalONE environment, the components "njxdemos" and "njxmapconverter" are provided in the drop-down list box.

If NaturalONE is not installed, the component "cisdemos" is provided, and any existing Application Designer projects in your *cis* web application. If you have chosen the *cis* web application folder as your project location, the selected Application Designer projects will not be moved. If you have chosen a different project location, the selected Application Designer projects will be physically moved to a subfolder of your Java project folder.

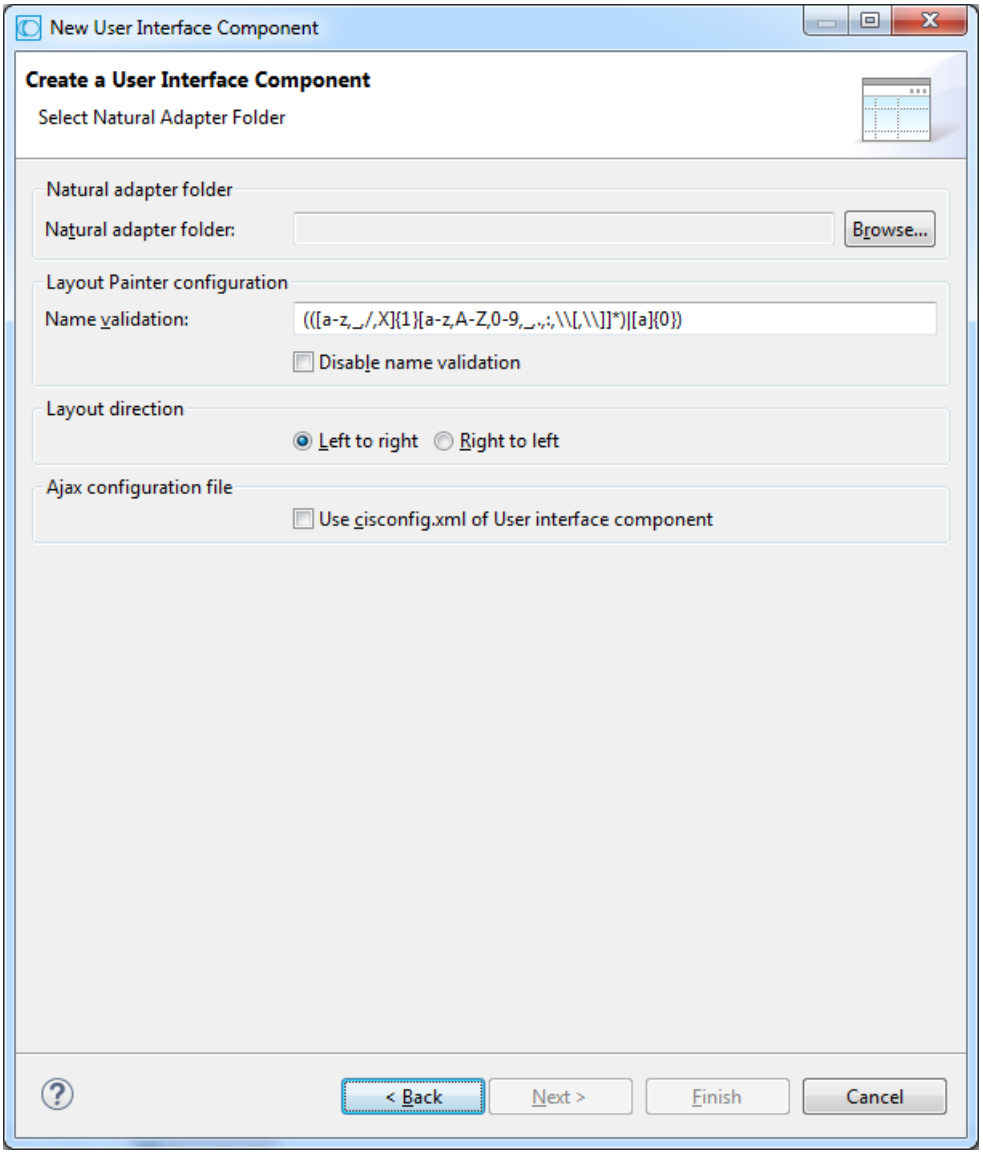
Import user interface component from external server

This option copies an existing Application Designer project from an external web container environment to your Eclipse workspace. An external web container environment is an environment other than the environment which is configured for the Natural or Java project. In this case, you have to choose the **Browse** button to select an external component from a dialog box.

The internal web container environment is then used for further development. The original sources in the external web container environment are no longer updated.

5 Choose the **Next** button.

The following page appears:



6 Choose the **Browse** button to select from a dialog box the folder into which the adapters are to be generated. Usually, you will select the *SRC* folder of a Natural library.

7 The **Name validation** text box shows all regular expressions that are used for the validation of properties and methods.

To switch off the validation, select the **Disable name validation** check box.

8 Select either the **Left to right** or **Right to left** option button. The specified layout direction is then used in the preview area of the Layout Painter. The selected layout direction applies for all layouts in this user interface component.

Note: The default layout direction for new user interface components is specified in the Ajax Developer **preferences**.

9 Choose the **Finish** button.

A new folder with the component name that you have defined is now shown in the **Project Explorer** view or **Natural Navigator** view.

File Structure in a User Interface Component

After the creation of a user interface component, the following folders are visible in the **Project Explorer** view:

Folder Name	Description
xml	Contains the page layouts.
multilanguage	Contains text files for the language support. These files are maintained with the **Literal Translator**.

In addition, a user interface component contains folders such as *accesspath*, *wsdl* or *protocol*. These folders contain generation results which are automatically changed when a layout is modified. The files in these folders are not intended to be viewed or modified directly, and are therefore filtered in the **Project Explorer** view.

Changing the Properties of a User Interface Component

In the properties of a user interface component, you can change the path to the Natural adapter folder, you can change the regular expressions that are used for the validation of properties and methods, you can disable the name validation, or you can change the layout direction.

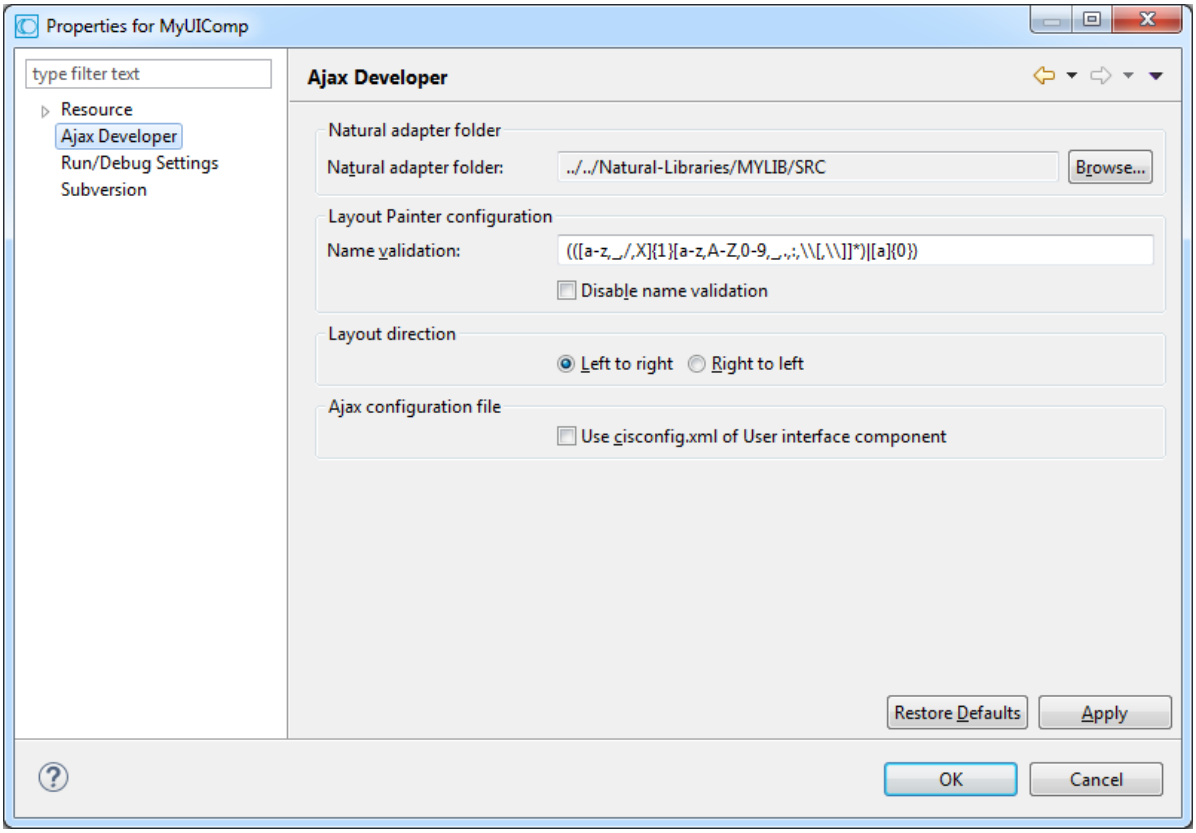
To change the properties of a user interface component

1 Select the user interface component in the **Project Explorer** view or in the **Natural Navigator** view.

2 Invoke the context menu and choose **Properties**.

3 In the tree of the resulting dialog box, expand the **Ajax Developer** node.

This displays the same information which is also shown on the second page when you **create** a user interface component.



4 Change all required information.

5 Choose the **OK** button to save your changes and to close the dialog box.

Global User Interface Components

It is often required that you often put certain components such as

- layout templates
- configuration files
- custom controls
- style sheets
- Java extensions (such as adapter listeners)
- images
- etc.

into a global user interface component and just use them from within your other user interface component.

These global user interface components will not contain page layouts for which Natural adapters are generated. Still, when creating a user interface component you must specify a Natural adapter folder.

For global user interface components we recommend the following:

1. Create a folder in your project and give this folder a name which you might want to filter from version control check-ins. Example: *zzztemp*.
2. When creating your global user interface component, specify this *zzztemp* folder as Natural adapter folder.

Exporting User Interface Components

You can export a user interface component to an external web container environment. This is useful if a user interface component is to be tested in an environment other than the development environment.

To export a user interface component

1 In the **Project Explorer** view, select the user interface component that you want to export

2 Invoke the context menu and choose **Ajax Developer > Export**.

The standard Eclipse functionality for exporting to the file system is used; see the Eclipse online help for further information.

3 Specify all required information.

4 Choose the **Finish** button.

Removing User Interface Components

When a user interface component is no longer required, you can remove it from the internal server.

After removing a user interface component, the files still exist physically on the disc and are still accessible in the **ProjectExplorer** view or **Natural Navigator** view, but the files are no longer accessible from within the web application. If you also want to delete the files, use the **Delete** command which is available in the context menu.

To remove a user interface component

1 In the **Project Explorer** view, select the user interface component that you want to remove.

2 Invoke the context menu and choose **Ajax Developer > Remove**.

A dialog appears, indicating that the user interface component has been removed from the web application.

Cleaning User Interface Components

When you clean a user interface component, files such as *.*html*, *.*access* and *.*NS8*, which have been generated for the layouts in the selected user interface component, are removed.

To clean a user interface component

1 In the **Project Explorer** view, select the user interface component that you want to clean.

2 Invoke the context menu and choose **Ajax Developer > Clean**.

Building User Interface Components

When you build a user interface component, HTML files are generated for the XML layout definitions. For pages of type NATPAGE, the adapters (*.*NS8* files) are also generated. This is helpful, for example, if you have previously cleaned the user interface component.

When an HTML file does not exist, it is generated. When an HTML file already exists, it is regenerated using the latest information from the XML layout definition.

Note: When you save a layout with the Layout Painter for the first time, an HTML file is automatically generated (in addition to the XML file).

To build a user interface component

1 In the **Project Explorer** view, select the user interface component that you want to build.

2 Invoke the context menu and choose **Ajax Developer > Build**.

Refreshing User Interface Components

In rare situations, it may happen that the configuration files of the web application are not in sync with the existing user interface components in your project. This may happen, for example, when a user interface component is created in a new workspace before the initialization of the **Development** web application is finished.

In these cases, you can refresh the user interface components to synchronize the configuration files of the web application with the user interface components. If the user interface components are already in sync, the commands described below will not apply any changes.

To refresh all user interface components in a project

1 In the **Project Explorer** view, select the project in which you want to refresh the user interface components.

2 Invoke the context menu and choose **Ajax Developer > Refresh User Interface Components**.

To refresh a single user interface component

1 In the **Project Explorer** view, select the user interface component that you want to refresh.

2 Invoke the context menu and choose **Ajax Developer > Refresh**.

Executing and Debugging User Interface Components

You can execute pages of type PAGE and MFPAGE (workplace). Execution means that the page with the real application (Natural or Java) is started in the browser.

For workplace applications (MFPAGE) which contain pages implemented with Natural, you can debug the corresponding Natural programs.

To execute a page

1 In the **Project Explorer** view, select a page of type PAGE or MFPAGE.

2 Invoke the context menu and choose **Ajax Developer > Execute**.

To debug Natural programs in a workplace application (MFPAGE)

1 In the **Project Explorer** view, select a page of type MFPAGE.

2 Invoke the context menu and choose **Ajax Developer > Debug**.

Notes:

1. For pages of type MFPAGE, the **Debug** command is only enabled when the debug attach server has been enabled in the Natural preferences. See also *Debug Attach Settings* in *Using NaturalONE > Setting the Preferences*.
2. The commands **Execute** and **Debug** are only available for the above mentioned page types. For pages of type NATPAGE, you have to start the application by executing or debugging the corresponding Natural main program. See also *Executing the Main Program* and *Debugging the Main Program* in the *Natural for Ajax* documentation.

Using Ajax Configuration Files

The Ajax configuration file *cisconfig.xml* contains settings for building an application and settings for runtime configuration. Put your specific *cisconfig.xml* into the predefined subfolder *cisconfig* **of exactly one user interface component of your workspace**. If you switch on the setting **Use cisconfig.xml of user interface component** in the properties of a user interface component, the *cisconfig.xml* file will be used for

- building all user interface components of your workspace and
- running all Natural for Ajax applications in your workspace.

The *cisconfig.xml* file will also be automatically packaged when using the War Ant deployment.

- How Can I Create a cisconfig.xml File in My User Interface Component?
- How Can I Modify the cisconfig.xml in My User Interface Component?
- How Can I Switch Back to the Default cisconfig.xml File?

How Can I Create a cisconfig.xml File in My User Interface Component?

Activate the setting **Use cisconfig.xml of user interface component** in the properties of your user interface component. This will create a subfolder *cisconfig* and copy a default *cisconfig.xml* to this subfolder.

How Can I Modify the cisconfig.xml in My User Interface Component?

Use an XML editor to open the file. Adapt the settings and save it.

Then click on **Refresh** in the **Ajax Developer** context menu of your user interface component.

How Can I Switch Back to the Default cisconfig.xml File?

Deactivate the setting **Use cisconfig.xml of user interface component** in the properties of your user interface component. Now the default *cisconfig.xml* will be used again.

Working With Version Control Systems

Eclipse provides fully integrated support for version control systems such as CVS or Subversion (SVN) via corresponding plug-ins.

The following topics are covered below:

- Checking In Projects and User Interface Components

- Checking Out Projects and User Interface Components

Checking In Projects and User Interface Components

You have to check in the following Ajax Developer-specific files and folders of a project:

- The file *com.softwareag.cis.ide.prefs* which is stored in the *.settings* directory of your project.

  This file contains the Ajax Developer-specific settings for this project (for example, whether this project has a root folder for the user interface components). To be able to simply check out a project into a new workspace and be able to work with the project immediately without having to reenter any configuration settings, you have to check in this file.
- The following files in the individual user interface components:

File Type	Description
Page Layouts	The page layouts are stored in the *xml* subdirectory of your user interface component.
Translation Information	If using the standard multi language management, literal translations are stored in comma-separated files in the *multilanguage* subdirectory of your user interface component.
Help Texts	If using the standard online help management, help texts are stored in the *help* subdirectory of your user interface component.
Other Web Resources	Images, style sheets or additional HTML pages are often used in user interface components. For example, your user interface component may have an *images* subdirectory which contains all GIF and JPG files.
Configuration Files	The file *.cisapplication* and, if it exists, the file *ciseditorconfig.xml*.

Do not check in the following files and folders (you can see them, for example, when you switch to the Resource perspective):

- Do not check in the **.html* files which are stored in the root directory of your user interface component.
- Do not check in any ZZZZZZ* files (that is, any files which start with "ZZZZZZ"). These are temporary files that are used by the Layout Painter.
- Do not check in derived folders. Some folders of a user interface component such as *accesspath*, *wsdl* and *protocol* are marked as "derived" in the folder properties. Version control systems such as CVS and Subversion automatically ignore derived folders during check-in. This is important because these files are automatically generated when you build your project.

To ignore the first two file categories listed above, you can either add them to *.cvsignore* or *svn:ignore* (depending on the version control system you are using), or you can simply **clean the user interface component** before you synchronize with your repository.

Checking Out Projects and User Interface Components

If you have checked in all files and folders described above, you can simply check out the project, including the user interface components, into a new workspace. The user interface components are automatically added to the internal Tomcat of this workspace.

After a check-out, it is recommended that you rebuild the project using the **Clean** command from the **Project** menu. If the **Build Automatically** command in the **Project** menu is currently not active, make sure that **Start a build immediately** is switched on in the **Clean** dialog box.

Using the Ajax Developer Perspective

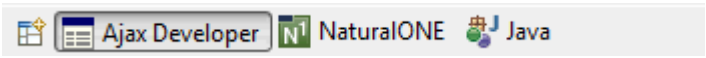
Ajax Developer provides its own perspective which contains the views which are important for editing layouts. The positions and sizes of the views are optimized for developing and testing the layout pages.

Note: When NaturalONE is installed, it is not mandatory to use the Ajax Developer perspective. You can also use the NaturalONE perspective. However, with the Ajax Developer perspective, the **Properties** view for modifying the properties of a page layout, for example, is shown with a convenient size and position.

To open the Ajax Developer perspective

- From the **Window** menu, choose **Open Perspective > Ajax Developer**.

  The corresponding icon is then shown on the perspective bar.



Note: By clicking the corresponding icon in the perspective bar, you can quickly switch between the NaturalONE perspective and the Ajax Developer perspective.

Developing Java Extensions

For some advanced functionality such as writing adapter listeners, advanced custom controls or advanced workplace enhancements, the development of Java extensions is supported.

If you want to develop Java extensions in NaturalONE, proceed as follows:

- Enable your Natural project for Java. For detailed information on how to enable projects for Java, see the Eclipse documentation.
- Add the files *cis.jar* and *sdo.jar* to the build path of the project. These files are located in the folder *cisnatural/WEB-INF/libs*.
- Create a subfolder *src* in the user interface component. Example: *<myui>/src*. Be sure to use this predefined folder name as Java source folder. Only then your Java files will be automatically compiled and packaged into a *.war* file for deployment.
- As output folder for the *.class* files use the subfolder *appclasses/classes* of your user interface component. Example: *<myui>/appclasses/classes*.
- Add the Javadoc location to the *cis.jar*. A corresponding zip file with the name *cisjavadoc.zip* can be found in the folder *cisnatural*.

Note: The folder *cisnatural* is located in your Eclipse workspace under *\.naturalone\apache-tomcat\webapps*.

II Layout Painter

The Layout Painter is the central tool for layout development. You use it to specify the layouts of your HTML pages. The Layout Painter includes an active WYSIWYG preview in which you can test your applications.

The description of the Layout Painter is organized under the following headings:

Getting Started with the Layout Painter

Defining the XML Layout

Using the Code Assistant

Using the Literal Assistant

Using the Validation Rules Editor

Using the Formula Editor

Note: If you are new to Ajax Developer, it is recommended that you read the *First Steps* tutorial before reading the information in this part. The tutorial provides step-by-step instructions on how to create a layout with the Layout Painter.

3 Getting Started with the Layout Painter

- General Information 34
- Creating a Layout 34
- Layout Templates 35
- Opening an Existing Layout 36
- Elements of the Layout Painter 36
- Creating Custom Layout Templates 38

General Information

Ajax Developer follows the commonly accepted paradigm of separating the layout (view) as much as possible from the application logic (model). You design the layout using the WYSIWYG Layout Painter. The result is kept in an XML page layout file. When you save the XML page layout, an HTML page is generated.

On the one hand, the XML layout contains the used controls of a page; on the other hand, it keeps information about how these controls are bound to the application code.

If you implement your application in Natural, the controls are bound to a Natural adapter. The Natural adapter is generated as soon as you save the page layout. The Natural adapter becomes part of the Natural application and forms the interface between the Natural application code and the user interface.

If you implement in Java, the controls are bound to the properties and methods of a Java adapter. This is a Java class that you can generate with the Code Assistant (which is part of the Layout Painter).

In addition, the XML layout may optionally contain so called “literals”. Literals are text identifiers (text IDs) that are replaced at runtime with language-specific texts. For each language you want to support, there must be a comma-separated value (CSV) file that contains the language-specific text per literal. You maintain these translation files using the Literal Assistant (which is part of the Layout Painter).

Creating a Layout

The layouts that you create as described below are stored in the *xml* folder of the user interface component.

To create a layout

1. In the **Project Explorer** view or in the **Natural Navigator** view, select the user interface component or the *xml* folder.
2. From the **File** menu or context menu, select **New > Page Layout**.
3. In the **Page layout** text box, enter a name for the file that will contain your layout definition. Do not specify an extension - the extension ".xml" is added automatically.
4. Select a layout template for your page from one of the tabs. See also ***Layout Templates***.

If you select a layout template from the **Natural Page** tab, enter a name for the Natural adapter that will be generated when you save the page layout in the **Natural adapter** text box. This name must match the naming conventions for Natural objects.

5 Click **Finish**.

The Layout Painter shows, so you can design the newly created layout (see **below**).

Layout Templates

When you **create** a layout, you have to select a layout template in the **New Ajax Page Layout** dialog box. The following tabs, which contain different types of layout templates, are offered by default:

- **Natural Page**
  The templates on this tab are used to create pages for Natural for Ajax. They have the NATPAGE control as the top node.
- **Workplace**
  The templates on this tab are used for arranging Application Designer pages in a frameset. They have the MFPAGE control as the top node. See Multi Frame Pages for further information.
- **Standard Pop-up Dialogs**
  Several standard pop-up dialogs are supported. See in *Working with Pages* for further information.

  These standard pop-up dialogs are also used within the workplace framework. Sometimes, you would like to slightly modify these standard pop-up dialogs, for example, when writing a workplace application. The templates on this tab are used to create a copy of the corresponding standard pop-up dialog in your project and adapt the appearance to your needs. Be sure not to change the names of the layouts, the corresponding adapter classes or the properties/methods. See also *Using Your Own Standard Pop-up Dialogs and Messages*
- **PDF Output**
  The templates on this tab are used for integrating a PDF document into an Application Designer page. They have the CISFO:FOPPAGE2 control as the top node. See Working with PDF Documents for further information.
- **HTML Page**
  The templates on this tab have the PAGE control as the top node. See PAGE for further information.

If you have created your own custom layout templates, they are also offered for selection. For further information, see ***Creating Custom Layout Templates***.

Opening an Existing Layout

Layouts are stored in the *xml* folder of the user interface component. They have the extension *.xml*.

To open a layout

- In the **Project Explorer** view or in the **Natural Navigator** view, double-click the layout that you want to open.

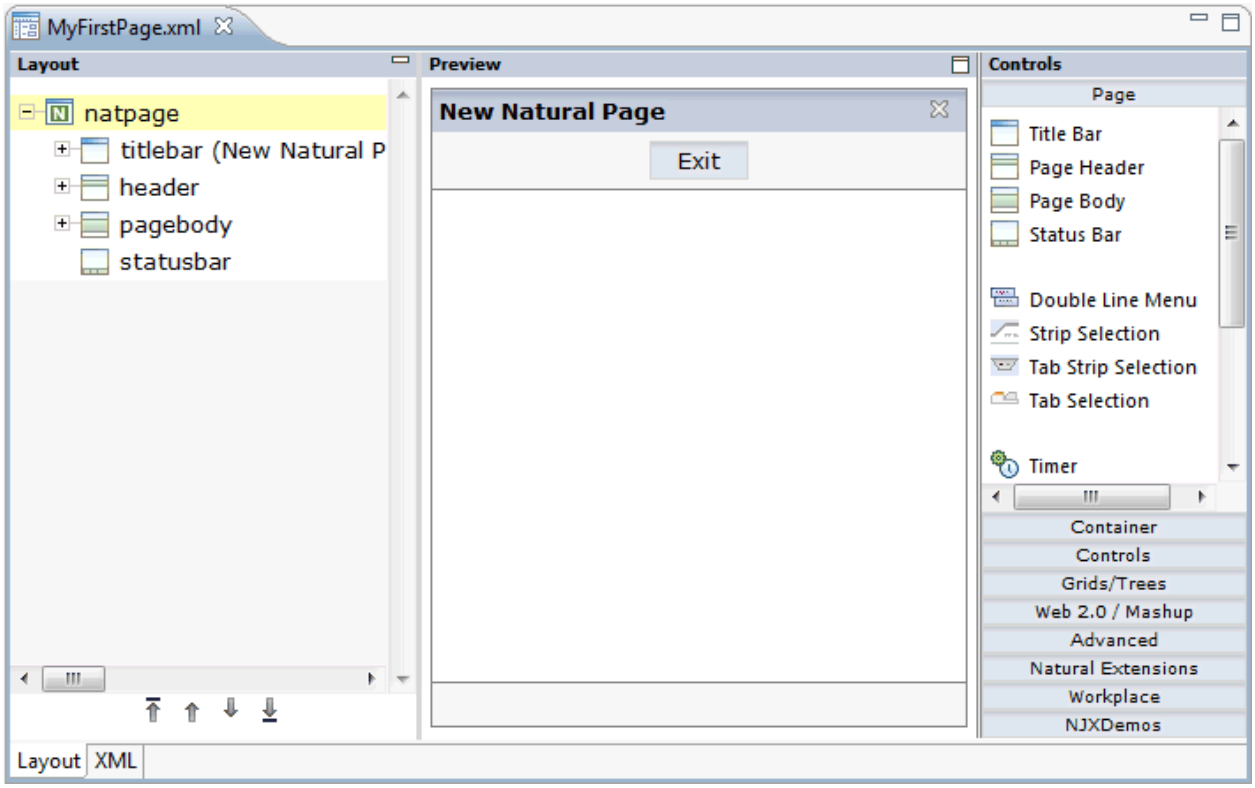
  Or:

  Invoke the context menu for the layout and choose **Open With > Layout Painter**

  The Layout Painter appears (see **below**).

Elements of the Layout Painter

The Layout Painter appears when you **create** a new layout or when you **open** an existing layout.



The Layout Painter is divided into several areas:

- **Layout Area (left side)**
  This area shows the layout tree which contains the controls that represent the XML layout definition. You drag these controls from the controls palette into the layout tree. Each node in the layout tree represents an XML tag.

  Using the button which is located to the right of the title bar, you can hide the layout area. This enlarges the preview area. To redisplay the layout tree, choose the small arrow which is shown in the left of the preview area's title bar.
- **Preview Area (middle)**
  The preview area shows the page which is created using the controls in the layout area. This page is refreshed each time, you choose the **Refresh Preview** command (see below).

  Using the toggle button which is located to the right of the preview area's title bar, you can enlarge the preview area so that it takes up the space of the layout area and controls palette, and reduce the enlarged preview area so that the layout area and controls palette are shown again.
- **Controls Palette (right side)**
  Each control is represented by an icon. A tool tip is provided which appears when you move the mouse pointer over the control. This tool tip also displays the XML tag which will be used in the XML layout.

  The controls palette is structured into sections (for example, **Page** and **Container**), where each section represents certain types of controls.

  Note: The sections that are shown in the controls palette and the content of a section depend on the layout template that is used for the current layout.

When you select a control in the layout tree or in the preview area, the properties for this control are shown in the **Properties** view. You can modify the properties as required.

When the Layout Painter is active, the additional menu **Ajax Developer** is shown in the menu bar. Using this menu, you can invoke additional tools which are helpful for the creation of layouts:

- **Code Assistant**
  Only used for developing page layouts in Java. Helps you to generate code for all properties and methods which are explicitly referenced in the page layout. See *Code Assistant* for further information.
- **Literal Assistant**
  Helps you to prepare your layout for multiple languages. You can maintain texts of different languages for the literals that are referenced within the XML layout. See *Literal Assistant* for further information.

- **Validation Rules Editor**
  You can define that the value that is entered in a field must meet certain conditions. See ***Using the Validation Rules Editor*** for further information.
- **Formula Editor**
  You can define a formula for a field (for example, the sum of the fields A and B is to be shown in field C). See ***Using the Formula Editor*** for further information.

Some commands of the **Ajax Developer** menu (such as **Refresh Preview**) are also available as buttons in a toolbar.

Creating Custom Layout Templates

You can create your own custom layout templates. When you **create** a layout, your custom layout templates are offered for selection together with the default **layout templates** in the **New Ajax Page Layout** dialog box.

You can create your custom layout templates in any Natural project. Your custom layout templates can always be used from all other Natural projects in the current workspace.

Tip: It is helpful to create a single project which contains all of your custom layout templates. Other developers can then import this project into their own workspaces. Thus, they can also use your custom layout templates.

To create a custom layout template

1. In the user interface component of your project, create a folder with the name "cisconfig".
2. In the *cisconfig* folder, create one or more layout template files. Or copy your template files into this folder.

   The layout template files have the same XML format as the page layout files (for an example, see ***Viewing the XML Code***). Therefore, you can also copy an existing layout file into the *cisconfig* folder and rename it to, for example, "editorTemplateNATPage1.xml".
3. In the *cisconfig* folder, create an XML configuration file with the name "editortemplates.xml". This file defines all custom layout templates that are to be used. Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<editortemplates>
    <group name="Natural Page">
         <xmltemplate filename="editorTemplateNATPage1.xml"
         image="templatenatural.jpg"
         info="My Natural Page Template"/>
    </group>
</editortemplates>
```

The XML configuration file uses the following tags:

`<editortemplates>`
: Each XML configuration file must have exactly one `<editortemplates>` tag.

`<group>`
: You can define one or more `<group>` tags inside the `<editortemplates>` tag.

  Each `<group>` tag has a mandatory `name` attribute which defines the tab in the **New Ajax Page Layout** dialog box on which the custom layout template is to be shown (see ***Creating a Layout***).

  You can also add a new tab to the **New Ajax Page Layout** dialog box by specifying a new group name (for example, "My Templates"). However, keep in mind that such a new tab is not suited for Natural pages because the text box in which you specify the adapter name will not be shown for the templates on your new tab.

`<xmltemplate>`
: You can define one or more `<xmltemplate>` tags inside a `<group>` tag.

  Each `<xmltemplate>` tag has the following mandatory attributes:

  - `filename`

    The name of an existing custom layout file in the *cisconfig* folder.

    Tip: If you want to create a custom layout template for a Natural page (that is, a page which has the NATPAGE control as the top node), it is recommended that the name of your layout file starts with either "editorTemplateNAT" or "editor-TemplateNJX". Otherwise, if you define a different file name, the text box **Natural adapter**, in which you specify the name of the adapter that is to be generated, will not be shown for your custom layout template in the **New Ajax Page Layout** dialog box.

  - `image`

    The name of an image file. This image will be shown in the **New Ajax Page Layout** dialog box. If the image file is not stored in the root of the *cisconfig* folder, you have to specify a relative path with the file name.

If you want, you can copy the standard image file for Natural pages. Its name is *templatenatural.jpg* and you can find it in your Eclipse workspace, under *.naturalone\apache-tomcat\webapps\cisnatural\HTMLBasedGUI\images*. You can copy it, for example, to your *cisconfig* folder.

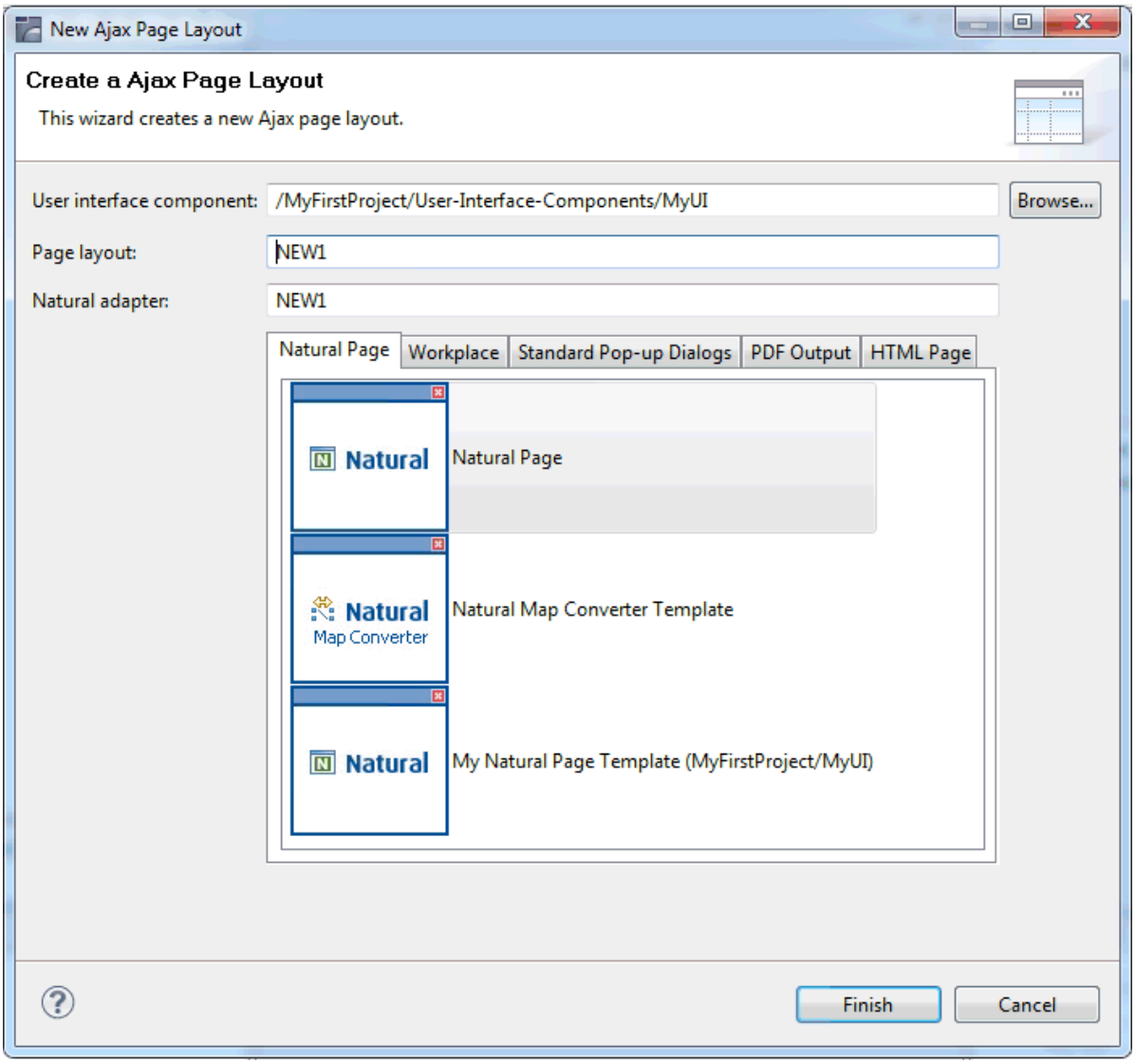
You can also add subfolders to your *cisconfig* folder. For example, you can add a *templates* folder which is to contain all of your custom layout templates, and an *images* folder which is to contain all of your images. It is important that the *editortemplates.xml* configuration file can always be found in the root of the *cisconfig* folder - you must not move this file to a subfolder.

- `info`

  A brief text describing the custom layout template. For example, "My Natural Page Template". This info text will be shown in the **New Ajax Page Layout** dialog box, next to the image.

4 To check whether your custom layout template is shown in the **New Ajax Page Layout** dialog box, **create** a new layout.

The following example shows a custom layout template with the info text "My Natural Page Template". This template has been defined with the above *editortemplates.xml* file.



The information in parentheses behind the info text indicates the location of the template (project name and user interface component).

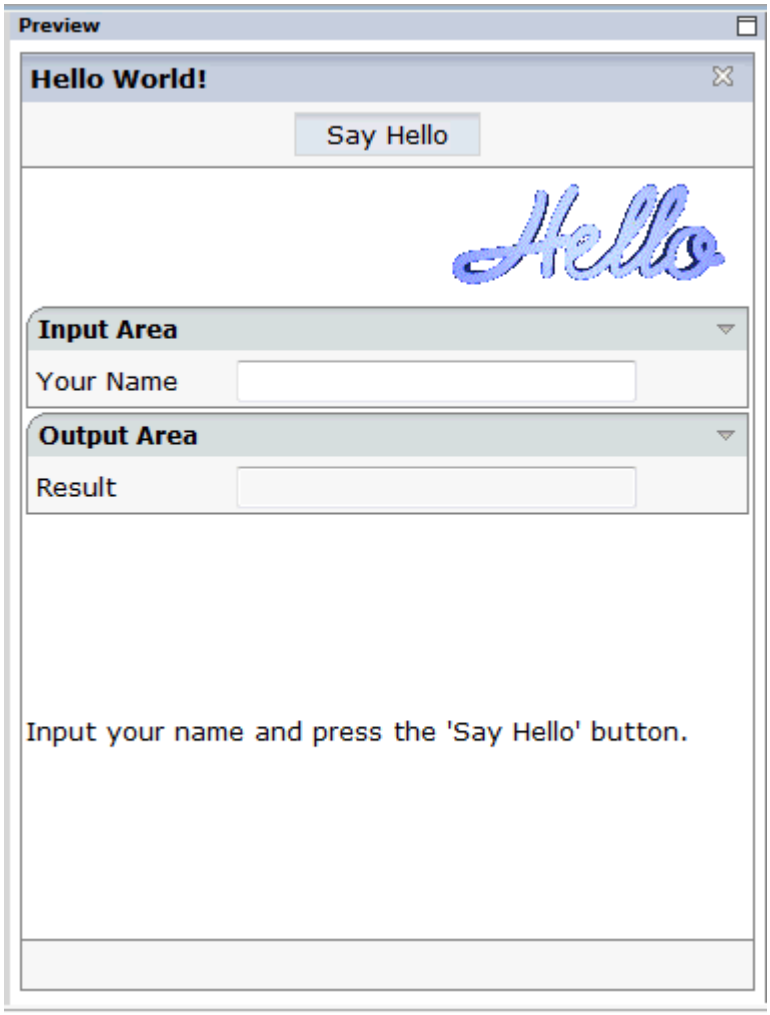
Your custom templates are always shown below the default layout templates. They are shown in the same sequence as defined in the *editortemplates.xml* file.

4 Defining the XML Layout

- Previewing the Layout 44
- Setting the Properties for a Control 46
- Changing the Appearance of Text 48
- Adding Controls to the Layout 49
- Creating a Folder for Images 51
- Defining a Background Image 51
- Defining a Style Sheet 52
- Defining Hot Keys 53
- Managing the Nodes in the Layout Area 54
- Managing the Controls in the Preview Area 57
- Viewing the XML Code 59
- Saving the Layout 60
- Layout Tester 61

Previewing the Layout

If you want to find out how the current layout definitions are rendered on the page, you can preview the layout. The preview of the layout is shown in the preview area. For example:



The preview area is a sensitive area. When you select a control in the preview area (for example, the title bar), this control is automatically selected in the layout tree.

Note: If you want to design your page layouts with a right-to-left (RTL) layout direction, you can set this in the Ajax Developer **preferences** (as a default for new user interface components), when **creating** a new user interface component, or when changing the **properties** of an existing user interface component.

To preview the layout

- From the **Ajax Developer** menu, choose **Refresh Preview**.

  Or:

  Choose the following button from the Ajax Developer toolbar:

  The preview area is updated according to the defined preview mode (see **below**).

  If the system finds wrong definitions, a message is displayed in the **Problems** view. To select the wrong control definition, double click the message line in the Problems view.

  - Changing the Preview Mode
  - Changing the Display Mode

Changing the Preview Mode

There are two different preview modes:

Run Application
This is the default setting.

For page layouts other than NATPAGE layouts, the latest changes to the adapter class are used when you preview your layout.

For NATPAGE layouts, only generic Java-side classes, which are part of the Application Designer framework, are used for the preview. The Natural adapter itself is not run. To run a NATPAGE layout with the Natural adapter, you use the **Run As > Natural Application** command with the corresponding Natural program.

Screen Test Only
For page layouts other than NATPAGE layouts: When the name of an adapter class has not been defined in the `model` property of the page, or when an adapter class has been defined which does not yet exist, an error screen is shown when you try to preview your layout. In this case, you have the possibility to activate **Screen Test Only**. Your layout will then be loaded with an empty adapter. This means that you will not have any functionality. Any changes that you make in your Java development environment will not be reflected in the preview.

For NATPAGE layouts: Instead of the generic Java-side classes of the Application Designer framework, which are executed when you choose **Run Application**, the layout is loaded with the same empty adapter as described above for non-NATPAGE layouts. For NATPAGE layouts, **Screen Test Only** is only used for troubleshooting. If **Run Application** fails, you can use **Screen Test Only**, for example, to verify whether Java Scripting and the basic Application Designer framework are able to load the page layout properly.

To change the preview mode

- From the **Ajax Developer** menu, choose **Preview Mode >** *mode*, where *mode* is one of the preview modes described above.

Changing the Display Mode

When you change the display mode, this is immediately considered in the preview area.

By default, the layout display mode as defined in the **project properties** is used. However, by choosing a different display mode (HTML or SWT) from the **Ajax Developer** menu, you can temporarily change the preview without having to change the settings in the project properties.

The menu provides for selection the following display modes:

HTML
When this display mode is active, the preview is shown as in the browser. The layout is rendered in the preview area with the actual CSS style sheet.

SWT
When this display mode is active, the preview is rendered as SWT controls. The actual CSS style sheets are ignored. Only a subset of the supported controls can be rendered as SWT controls. Responsive controls cannot be rendered as SWT controls.

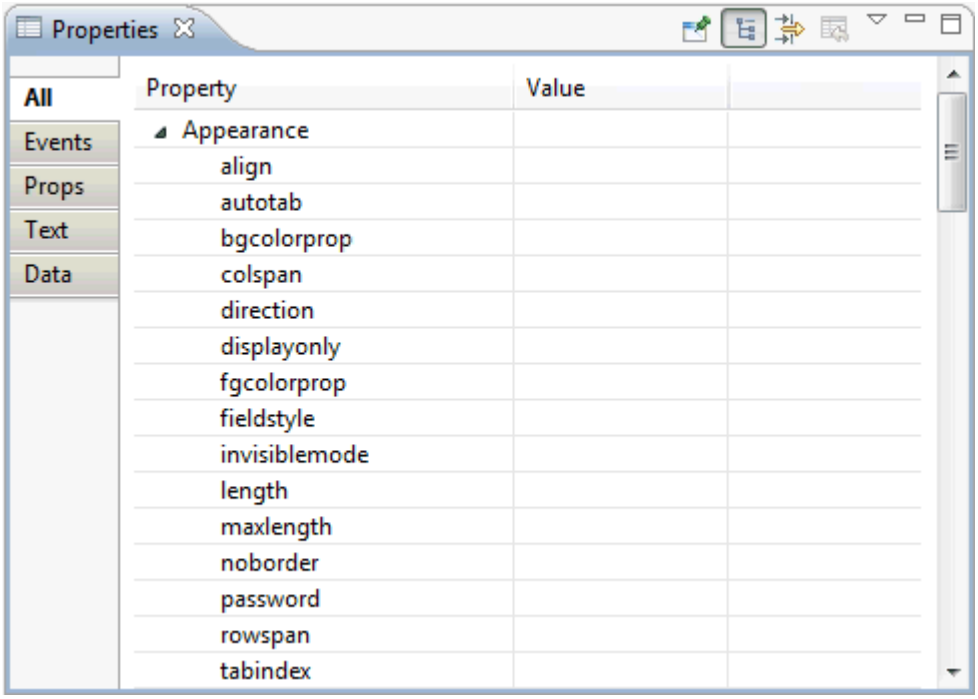
Project Settings
This is the default setting. When this display mode is active, the corresponding settings from the project properties are used.

To change the display mode

- From the **Ajax Developer** menu, choose **Display Mode >** *mode*, where *mode* is one of the display modes described above.

Setting the Properties for a Control

When you select a control in the layout area or in the preview area, its properties are shown in the **Properties** view and you can edit them immediately. Example:



The **All** tab is always provided for all controls. The availability of other tabs depends on the control which is currently selected.

The **Props** tab is only shown when the control has `*prop` properties (such as `visibleprop`), and the **Events** tab is only shown when the control has `*method` properties (such as `contextmenumethod`). These tabs are provided for your convenience. They show the properties which bear a reference to the program code. However, the properties that are shown on these tabs are also shown on the **All** tab.

When it is possible to change, for example, the font for text on a control, the **Text** tab is available. For further information, see *Changing the Appearance of Text*.

The **Data** tab is only shown for field controls. It shows any validation rules or formulae that have been defined for the field. Using the corresponding **Edit** button, you can invoke either the validation rules editor or the formula editor. For further information, see *Using the Validation Rules Editor* or *Using the Formula Editor*.

To set a property using the Properties view

1 Select the control in the layout area or in the preview area.

   The properties for the selected control are now shown in the **Properties** view. For some properties, default values are provided.

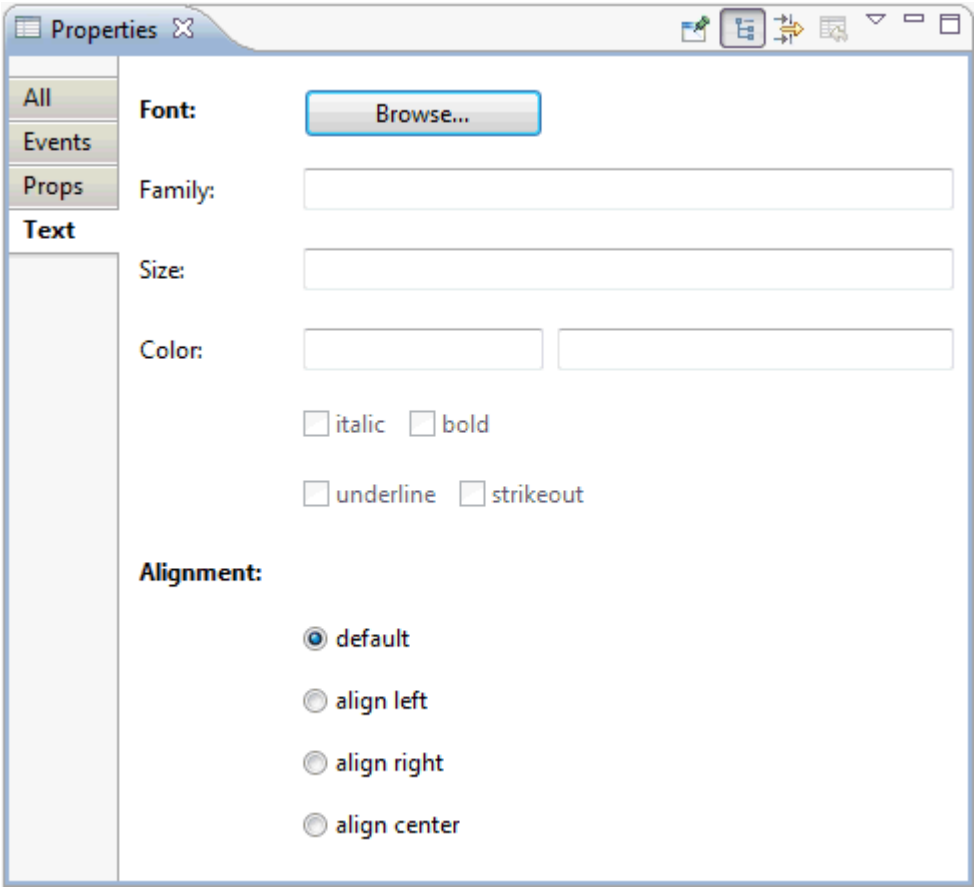
2 Specify the value for the required property.

Note: The values of some important properties (for example, the `name` or `textid` of the **Title Bar** control) are shown in the layout area, in round brackets after the XML tag. If you modify the value of such a property, the changed value will be shown in the layout area.

Changing the Appearance of Text

Instead of setting the corresponding property for a control, you can change the appearance of text (for example, on a button) using the **Text** tab of the **Properties** view. This tab is available for all controls for which the appearance of text can be changed.

The **Text** tab provides for selection only those options which make sense with the selected control. Example for a button:



To change the appearance of text

1 Select the control in the layout area or in the preview area.

2 Select the **Text** tab of the **Properties** view.

3 Choose the **Browse** button.

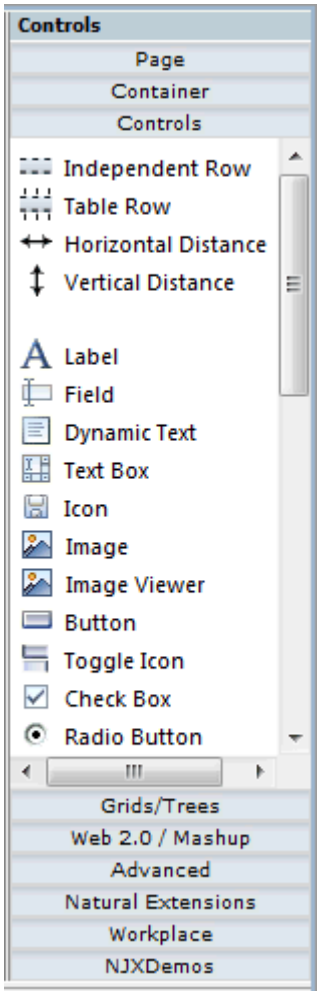
A dialog appears in which you can specify all required settings (such font family, font color or italics). These settings are then automatically entered on the **Text** tab.

4 If you want to change the alignment, select the corresponding option button.

Your changes are immediately shown in the preview area.

Adding Controls to the Layout

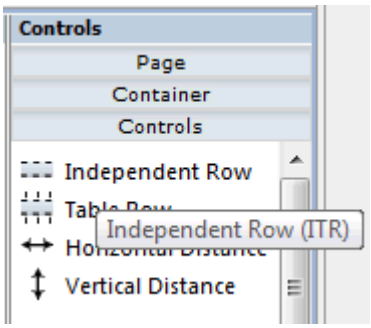
The controls that you can add to the layout can be found in the various sections of the controls palette. You simply drag them from the controls palette onto the corresponding node in the layout area or to the preview area. The preview area is immediately refreshed.



To add a control to the layout

1 Open the required section of the controls palette by choosing the corresponding button (for example, the **Controls** button).

When you move the mouse over a control, a tool tip appears which also displays the control name which will be used in the XML layout. For example:



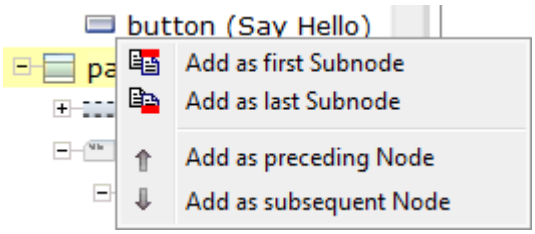
2 Drag the control from the controls palette onto the required node in the layout area.

Or:

Drag the control from the controls palette and drop it in the preview area.

Note: A control can only be inserted at a valid position. Thus, it may happen that the control is not inserted in the node which was highlighted when you dropped the control, but at a different position.

When you drop information, the system will sometimes respond by offering a context menu with certain options about where to place the control. Depending on the current context, different commands are available. For example:



The new node is automatically selected in the layout area so that you can maintain its properties directly in the **Properties** view

Creating a Folder for Images

All resources that your user interface component needs (such as images) must be contained in your project directory in the Eclipse workspace. It is good practice to create a specific folder for these resources.

To create a folder for images

1. In the **Project Explorer** view or in the **Natural Navigator** view, select the user interface component for which you want to create a folder.
2. Invoke the context menu and choose **New > Other**.
3. In the resulting dialog box, expand the **General** node, select **Folder** and choose the **Next** button.
4. Specify a folder name (for example, "images") and then choose the **Finish** button.
5. Place all images that you want to use in the images folder that you have just created.

Defining a Background Image

You can define a background image for your page. The image must be a file with the extension *gif*, *jpeg* or *jpg*.

To define a background image

1. Select a control for which a background image can be set.

   Note: When a background image cannot be set for a control, the **Insert Background Image** command is disabled.

2. From the **Ajax Developer** menu, choose **Insert Background Image**.

   Or:

   Choose the following button from the Ajax Developer toolbar:

A dialog appears.

3 Select the image file from your images folder (see also *Creating a Folder for Images*).

The path to the image will automatically be converted to a relative path.

4 Choose the **Open** button.

The correct relative URL is automatically set in the corresponding property. In different controls, different properties are used to support background images. Background images are supported, for example, by the `pagebodystyle` property of the PAGEBODY control and the `titlestyle` property of the TITLEBAR control.

The correct style definition including the relative URL for the background image file is automatically set in the corresponding property. For example:

```
background-image:url(images/absbackground_small.gif)
```

Defining a Style Sheet

You can assign a style sheet to your page. This can either be a predefined style sheet or one of your own style sheets that you have created using the **Style Sheet Editor**. The style sheet must be a file with the extension *css*.

Instead of setting the corresponding property for the page (see Static Selection of the Style Sheet File), you can define a style sheet as described below.

To define a style sheet

1 From the **Ajax Developer** menu, choose **Insert Style Sheet**.

Or:

Choose the following button from the Ajax Developer toolbar:

A dialog appears.

2 Specify the path to your style sheet.

You have to specify a style sheet from your local file system. The file must be located in the same directory as your web application (or any subdirectory). When you specify an absolute path, this will automatically be converted to a relative path.

Defining Hot Keys

You can assign hot keys to pages and to the controls FIELD and ROWTABLEAREA2. For example, you can define the hot key CTRL+S for calling the method `onSave`.

See also *Extended Hot Key Management* in the *Natural for Ajax*.

To define hot keys

1 Select the control in the layout area or in the preview area.

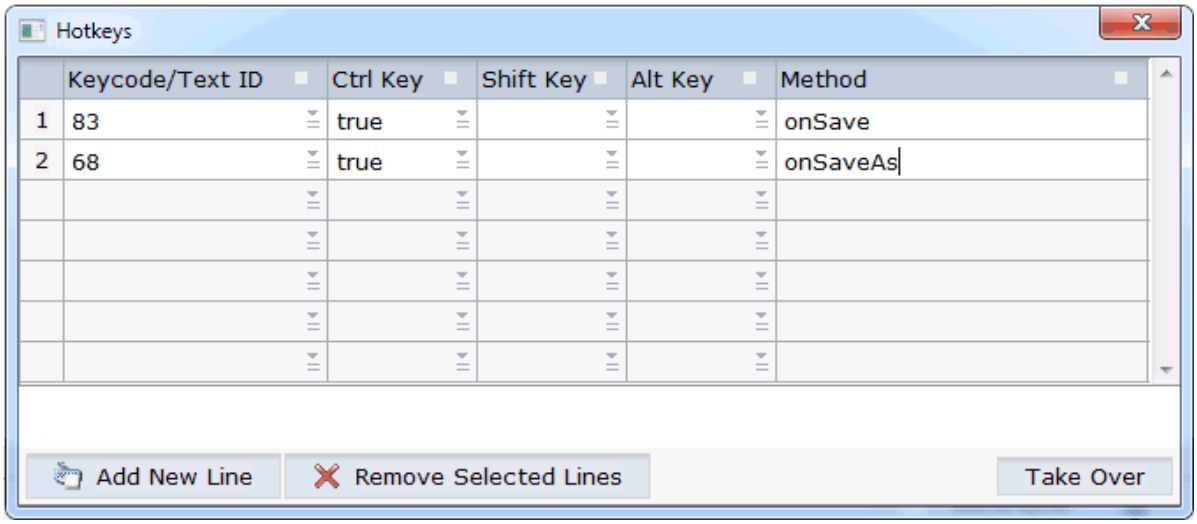
Note: When a control is selected for which hot keys cannot be defined, your hot key definition will apply to the page.

2 From the **Ajax Developer** menu, choose **Insert Hotkeys**.

Or:

Choose the following button from the Ajax Developer toolbar:

A dialog appears. When hot keys have already been defined, they are shown. Example:



3 When hot keys have already been defined, choose the **Add New Line** button to add an additional hot key.

> **Note:** To remove a hot key, select the corresponding row and choose the **Remove Selected Lines** button.

4 From the drop-down list box in the first column, select the key that you want to use as a hot key. For example, "83" represents the S key and "120" represents the F9 key.

Or:

In the text box of the drop-down list box in the first column, enter a text ID that is to be translated by the multi language management.

5 Define each key (CTRL, SHIFT and/or ALT) which is to be used for the hot key by selecting "true" from the corresponding drop-down list box.

> **Note:** If neither "true" nor "false" is selected for a key, the default setting "false" is used.

6 In the **Method** column, specify the name of the method that is to be invoked when the user presses the hot key.

7 When all hot keys have been defined, choose the **Take Over** button.

Managing the Nodes in the Layout Area

The following topics are covered below:

- Moving Nodes
- Using the Clipboard
- Drag-and-Drop
- Context Menus

Moving Nodes

You can manage the nodes in the layout area using the following buttons which are shown below the layout tree:

Button	Description
	Moves the selected node up to first position in the tree.
	Moves the selected node up to the previous position in the tree.
	Moves the selected node down to the next position in the tree.
	Moves the selected node down to last position in the tree.

When an operation is not allowed, clicking on the corresponding button has no effect.

Using the Clipboard

You can manage the nodes in the layout tree using the following commands from the **Edit** menu of Eclipse:

Command	Description
Cut	Cuts the selected node.
Copy	Copies the selected node.
Delete	Deletes the selected node.
Undo	Undoes the previous action. The previous action can be, for example, a move operation or a changed property value.
Redo	Redoes the previous undo action.

Note: To copy and paste information in the **Properties** view, use the key combinations CTRL+C and CTRL+V.

Drag-and-Drop

In the layout area, you copy or move the nodes using drag-and-drop.

To copy or move a node using drag-and-drop

1. In the layout area, select the node you want to move so that it is highlighted.

   Press SHIFT or CTRL, if you want to select more than one node.
2. Press the left mouse button and keep it pressed.
3. Drag the mouse to the target node.

   While you drag the mouse, a tool tip in light colors is shown for the node(s) you are currently dragging.
4. Release the mouse button.

A context menu appears and you can choose the location where you want to place the copy or the moved item (for example, in front of the target node).

Note: When the copy or move operation is not allowed, the context menu does not appear.

5 Choose the required command from the context menu.

Note: To cancel the drag process, move the mouse outside the context menu or press ESC. The context menu will disappear and any selected nodes will keep their current position.

Context Menus

You can manage the nodes in the layout tree using context menus. The following commands may appear in a context menu:

Command	Description
Add as first Subnode	Adds a control as the first subnode of the selected node. A cascading menu is available for this command, containing all controls that can be added as a subnode of the selected node.
Add as last Subnode	Adds a control as the last subnode of the selected node. A cascading menu is available for this command, containing all controls that can be added as a subnode of the selected node.
Add as preceding Sidenode	Adds a control before the selected node, on the same level as the selected node. A cascading menu is available for this command, containing all controls that can be added as a sidenode of the selected node.
Add as subsequent Sidenode	Adds a control after the selected node, on the same level as the selected node. A cascading menu is available for this command, containing all controls that can be added as a sidenode of the selected node.
Cut	Cuts the selected node.
Copy	Copies the selected node.
Paste as first Subnode	Pastes the cut or copied node as the first subnode of the selected node.
Paste as last Subnode	Pastes the cut or copied node as the last subnode of the selected node.
Paste as preceding Sidenode	Pastes the cut or copied node before the selected node, on the same level as the selected node.
Paste as subsequent Sidenode	Pastes the cut or copied node after the selected node, on the same level as the selected node.
Remove	Deletes the selected node.
Open all Subnodes	Opens all subnodes of the selected node.
Close all Subnodes	Closes all subnodes of the selected node.

Managing the Controls in the Preview Area

You can manage controls directly in the preview area.

When you select a control in the preview area, it is highlighted with a different color in the layout area and vice versa. Pressing CTRL+SHIFT while selecting a control will select its parent control. You can copy or move the control. In SWT mode you can also resize the control.

The following topics are covered below:

- Moving and Copying Controls in the Preview Area
- Controls with visibleprop
- Customizing the Selection Highlighting
- Resizing Controls in the Preview Area

Moving and Copying Controls in the Preview Area

You can move or copy a control or groups of controls directly in the preview area.

Note: To select more than one control, press CTRL or SHIFT while selecting the controls.

To move controls in the preview area

1. Select the controls that you want to move.
2. Drag the controls to a new valid position.

To copy controls in the preview area

1. Select the controls that you want to copy.
2. Press CTRL and drag the controls to a valid position at which it is possible to insert the copies.

The drop position will be highlighted. A control can only be inserted at a valid position. Thus, it may happen that the control is not inserted in the node which was highlighted when you dropped the control. When you drop controls, the system will sometimes respond by offering a context menu with certain options about where to place the control.

To copy/move controls in the preview area via context menu

1. Right-click with the mouse on the control you want to copy. If you want to copy several controls, select them first before right-clicking with the mouse.
2. Select **Copy** or **Cut**.

To paste controls in the preview area via context menu

1 Right-click with the mouse at a valid position.

2 Select your preferred **Paste** operation from the context menu.

Controls with visibleprop

When defining a `visibleprop` for controls, this means that the control is only visible when at runtime the value is set to TRUE. Responsive controls are shown in the HTML preview even if they have a `visibleprop`.

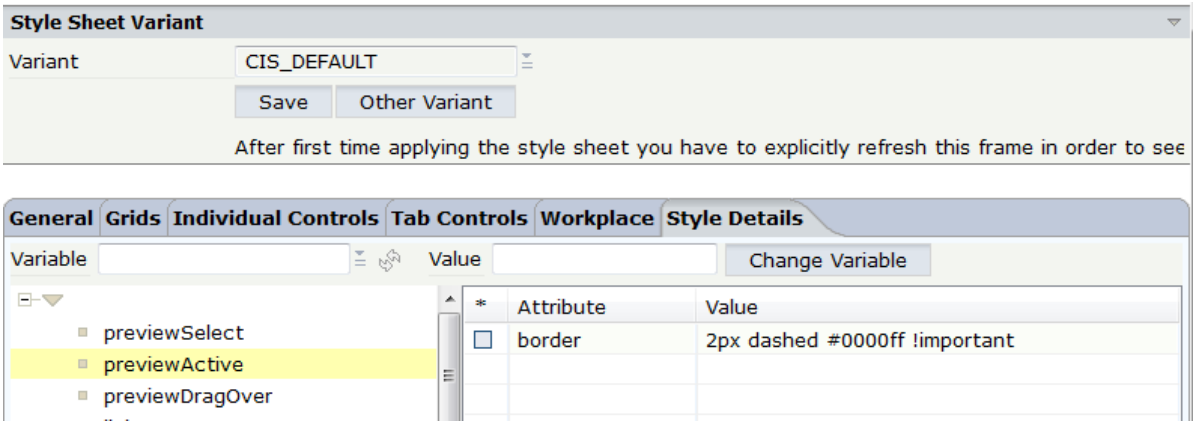
Customizing the Selection Highlighting

When a control is selected, it is highlighted with a specific style in the preview area. For the HTML Preview you can customize the highlighting style.

To customize the highlighting style

1 Open the style sheet that you use for your page layouts in the **Style Sheet Editor**. The default style sheet is the *CIS_DEFAULT.css*.

2 Select the **Style Details** tab as shown below.



3 Customize the `preview*` styles according to your needs.

Preview Variable	Description
`previewSelect`	Style for selected controls. **Note:** The selected active control has the style `previewActive`.
`previewActive`	The active control, for which the properties are shown in the property view.
`previewDragOver`	During drag and drop, controls are highlighted when the mouse is dragged over the control.

Resizing Controls in the Preview Area

In SWT Preview mode you can set the width and/or height of a control directly in the preview area.

To resize a control in the preview area

1. Select the control that you want to resize.
2. Move the mouse pointer over the dashed border of the selected control.

   The mouse pointer changes, indicating that you can start the resizing.
3. Press the mouse button and drag the border until the control has the desired width or height.
4. Release the mouse button.

Viewing the XML Code

When creating the layout, you can view the currently defined XML code at any point of time.

To view the XML code

- Choose the **XML** tab which is shown at the bottom of the Layout Painter.

  Information such as the following is shown:

```
NEW1.xml
1  <?xml version="1.0" encoding="UTF-8"?>
2  <natpage natkcheck="true" natsinglebyte="true" xmlns:njx="http://www.soft
3      <titlebar name="New Natural Page">
4      </titlebar>
5      <header withdistance="false">
6          <button name="Exit" method="onExit">
7          </button>
8      </header>
9      <pagebody>
10     </pagebody>
11     <statusbar withdistance="false">
12     </statusbar>
13 </natpage>
14
15
16
Layout  XML
```

Note: The properties which you leave blank in the **Properties** view are not shown in the XML code.

Saving the Layout

When you save the layout, all of your changes in the Layout Painter are saved.

When you save the page layout or build your project, errors and warnings are shown in the **Problems** view, which is a standard Eclipse view. An error occurs, for example, when a value has not been set for a mandatory property. When you click an error or warning, the control containing the error is selected in Layout Painter.

When you save a layout for the first time, an HTML file is generated (in addition to the XML file) which is placed into the root directory of your application project. This HTML file is updated each time you save the layout.

For NATPAGE layouts, the corresponding Natural adapter is also generated.

Note: When you save the page, the **preview area** is automatically updated.

To save the page

- Use the standard Eclipse functionality for saving (for example, CTRL+S).

Layout Tester

The layout tester can be used to test the currently defined layout according to the display mode which is defined in the properties of the project: it is either shown as in the browser or as in the SWT client.

To invoke the layout tester

1 Select the layout in the **Project Explorer** view or in the **Natural Navigator** view.

2 Invoke the context menu and choose **Open With > Layout Tester**.

5 Using the Code Assistant

- About the Code Assistant 64
- Code Template 64
- Opening the Code Assistant 64
- Code Introspection 65
- Generating the Code 66
- Editing the Adapter Code 67
- Saving Changes 68
- Changes from the Outside 69

About the Code Assistant

The Code Assistant only applies to PAGE layouts.

When developing page layouts in Java, automatic code modifications do not occur. For example, when you set the `valueprop` property in a FIELD control, the `set()` and `get()` methods are not automatically created in the underlying Java adapter class. Using the Code Assistant, however, you can generate code for all properties and methods which are explicitly referenced in the page layout. The compilation of the modified adapter class itself is done in your Java development environment (for example, in Eclipse).

When developing page layouts in Natural, the corresponding Natural code for the properties and methods is automatically generated when you **build a user interface component**, when you build your project in Eclipse, or when you simply **save** a modified layout in Eclipse and the **Build Automatically** command in the **Project** menu is activated. Therefore, the Code Assistant is not required for NATPAGE layouts and the corresponding menu commands are disabled.

Code Template

Before you open the Code Assistant, make sure that the `model` property of the page contains the correct adapter class. The default entry for this property is "DummyAdapter".

The Code Assistant looks in the source directory for the adapter class you have set with the `model` property. If the class cannot be found, the Code Assistant loads a code template.

The code template is stored in the directory `<your-webapplication>`/*cis/config* . For NaturalONE, this is usually `<webcontainerinstalldir>`/*webapps/cisnatural/cis/config*. Otherwise, it is usually `<webcontainerinstalldir>`/*webapps/cis/cis/config*.

The name of the code template is *adapterTemplate.java*. You can change this template according to your needs.

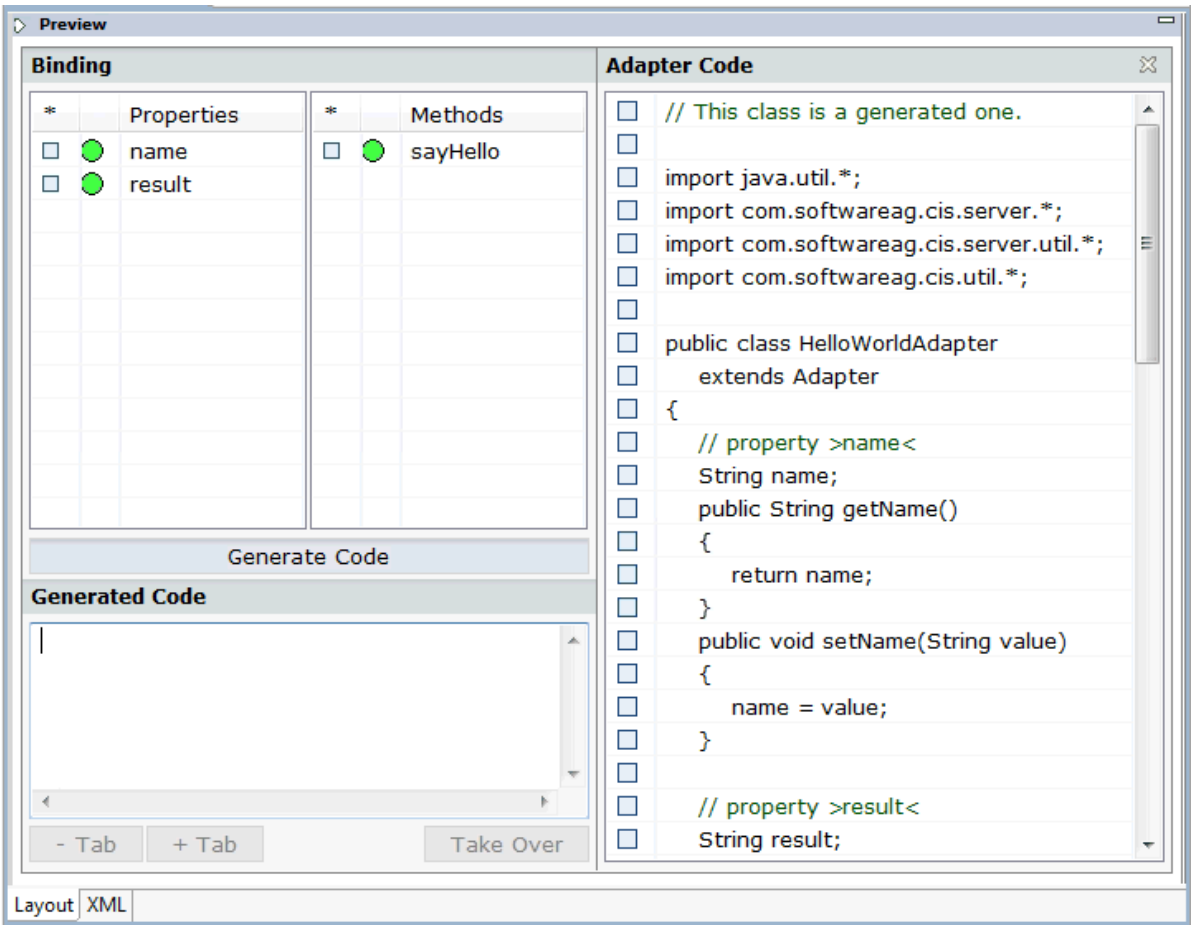
Opening the Code Assistant

In order to read and write source code, the Code Assistant needs to know where the Java sources are kept. This information is stored in the file *ciseditorconfig.xml* of each user interface component. When working with Eclipse, a *src* subdirectory is automatically created in each user interface component and applied as a Java source directory to your Eclipse project. The corresponding settings are automatically saved to the file *ciseditorconfig.xml*.

To open the Code Assistant

- From the **Ajax Developer** menu, choose **Open Code Assistant**.

  When the Java source directory can be found, the Code Assistant is shown in the preview area. For example:



Code Introspection

The binding area at the top left of the Code Assistant lists all properties and methods that are used in the XML layout. Colored dots are shown in front of each property and method. The color of the dot indicates whether the Code Assistant could find the corresponding coding within your source code.

Introspection is done by reading and interpreting the code. Comments that highlight properties or methods are not necessary.

Possible dot colors for properties:

Green	The property's `get` and `set` methods are available.
Yellow	The `set` method is required but could not be located. The `get` method is fine.
Red	The `get` method could not be found.

Possible dot colors for methods:

Green	The method is available.
Red	The method could not be found.

When you select a property or method in the binding area, the properties are shown in the **Properties** view.

In addition, when code has already been generated for the selected property or method (see below), the corresponding line is automatically selected in the adapter code which is shown at the right of the Code Assistant.

Generating the Code

The properties and methods which cannot be found in the adapter code are indicated by red dots.

When you generate code, a framework for the property or method is generated in the adapter code.

To generate code

1. Select the property or method for which you want to generate code.

   You can also use the SHIFT or CTRL key to select multiple entries.
2. Choose the **Generate Code** button.

   The generated code appears in the area below this button.

   Check whether you really want to insert these statements into your source code. You can modify or delete the code. Any code that you manually enter here will be taken over.
3. Optional. Choose the **+Tab** or **-Tab** button to add or remove a tab at the beginning of the code. This applies to all code which is currently shown in the **Generated Code** area.
4. In the **Adapter Code** area on the right, select the line below which you want to insert the code.
5. Choose the **Take Over** button to insert the code below the selected line.

The color of the dot changes from red to green. This indicates that the corresponding methods are now available in the adapter code.

6 Save your changes.

Note: It is recommended that you save your changes each time you switch from the Code Assistant to another tool which may be shown in the preview area, and vice versa.

Editing the Adapter Code

You can edit the adapter code in different ways as described below.

- Editing the Adapter Code in the Java Editor
- Editing the Adapter Code Directly in the Code Assistant

Editing the Adapter Code in the Java Editor

If you want to apply complex changes to the adapter code, it is recommended that you do this in the Java editor.

Important: Be sure to close the Code Assistant before you open the adapter code in the Java editor.

To invoke the Java editor

1 In the **Project Explorer** view, select the page layout (that is, the XML file) for which you want to change the adapter code.

2 Invoke the context menu and choose **Ajax Developer > Open Java Adapter**.

Or:

When the Layout Painter is still active, open the **Ajax Developer** menu and choose **Open Java Adapter**.

Editing the Adapter Code Directly in the Code Assistant

Instead of invoking the Java editor, you can also apply smaller changes to the adapter code directly in the Code Assistant, while generating the code for the properties and methods.

To edit the adapter code in the Code Assistant

1 Select one or more lines in the **Adapter Code** area.

To select more than one line, use CRTL or SHIFT.

2 Invoke the context menu and choose one of the following commands:

Command	Description
-Tab	Removes a tab from the beginning of the selected line(s).
+Tab	Adds a tab to the beginning of the selected line(s).
Cut	Cuts the selected line(s).
Copy	Copies the selected line(s).
Paste	Only available after a cut or copy operation. Pastes the cut or copied line(s) below the currently selected line.
Remove	Removes the selected line(s).

Saving Changes

You save the adapter code using the standard Eclipse functionality for saving.

Notes:

1. When editing the adapter code using the Code Assistant, it is recommended that you first close the Code Assistant before you close the Layout Painter.
2. After saving your changes to the adapter code, choose the **Refresh Preview** command so that you can see the changes in the preview area.

Changes from the Outside

Each time you **preview** the layout, the Code Assistant checks whether the file has changed on the file system. Therefore, the file attribute "last modified" is examined (not the file content). If the Code Assistant notices a file change, it checks whether it is holding own unsaved changes. If not, the class code is reloaded from the file system. If you have done changes in parallel (within your Java development environment and the Code Assistant) a dialog appears. Choose either to reload the file from disk (and lose the changes you have done within the Code Assistant) or to keep the Code Assistant changes (and lose the outside changes on the next save).

Note: You have to take care that the following case never occurs: the file has changed on file system and the Code Assistant has its own unsaved changes. Of course, it is reasonable to open the class both in your Java development environment and in the Code Assistant. It is recommended that you save or discard changes before you switch from the Code Assistant to your Java development environment (or vice versa). Otherwise, you will lose one of your changes.

6 Using the Literal Assistant

- About the Literal Assistant 72
- Opening the Literal Assistant 72
- Maintaining Literals 73
- Selecting Another Language 74
- Displaying a Comparison Language 74

About the Literal Assistant

The Literal Assistant can be used it to maintain texts of different languages for the literals that are referenced within the XML layout.

The concept of the multi language management is described in *Multi-Language Management in Java Applications* . It is recommended that you read this information before you proceed with the information below.

Note: Text can also be translated using the **Literal Translator**.

Opening the Literal Assistant

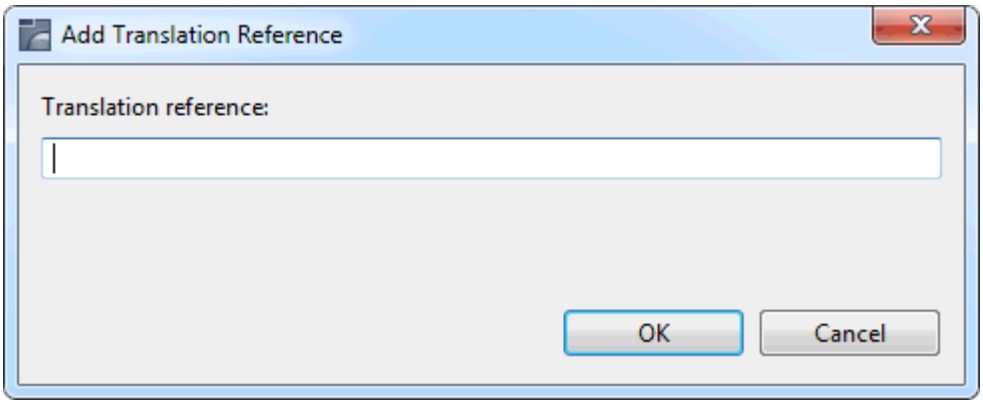
The Literal Assistant needs to know in which file the language-specific texts are kept. The name of this translation file is stored in the `translationreference` property of the page.

See also *Multi-Language Management in Java Applications*.

To open the Literal Assistant

1 From the **Ajax Developer** menu, choose **Literal Assistant**.

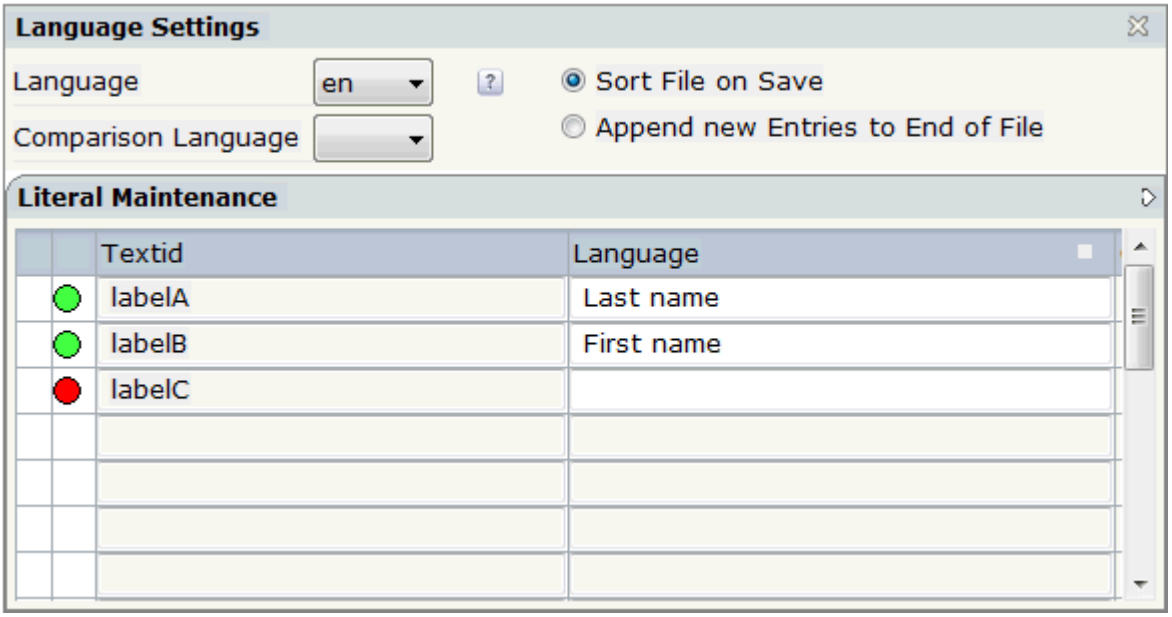
When a translation file has not yet been specified, the following dialog box appears:



2 Specify the name of the translation file and choose the **OK** button.

The name you specify is entered in the `translationreference` property.

The Literal Assistant appears in the editor area.



Note: For multi language support, you use the `textid` property for a label (instead of the `name` property).

Maintaining Literals

All referenced literals of the currently selected language are provided in the **Literal Maintenance** area.

Colored dots are shown in front of each literal. The color of the dot indicates whether the Literal Assistant could find the corresponding text in the translation file for the currently selected language.

Possible dot colors:

Green	A text is available in the translation file.
Yellow	The translation file contains a line for the literal - but the text is empty.
Red	The literal (and text) does not yet exist in the translation file.

When you select the first column (that is, the column to the left of the colored dot), the corresponding node is automatically selected in the layout tree and the properties for this node are shown in the **Properties** view.

To add/modify text for the current language

1 Enter all required text in the **Language** column.

2 Define the sequence of the entries in the translation file by selecting one of the following option buttons:

- **Sort File on Save** (ascending order)
- **Append new Entries to End of File**

3 Choose the standard Eclipse **Save** command to save your text input.

Your changes will also be available in the preview of the HTML page.

Selecting Another Language

The valid languages are determined by the existing subdirectories in the *multilanguage* directory which belongs to your user interface component. An entry for each subdirectory is provided in the **Language** drop-down list box.

Note: You may have to choose the **Refresh Preview** command to refresh the Literal Assistant so that any subdirectories you have just defined in your file system are shown in the drop-down list box.

To select another language

- From the **Language** drop-down list box, select the required language.

  When you have not yet saved your previous changes, a dialog appears asking whether you want to save your changes.

  Important: The Literal Assistant keeps changes for one language only. If you want to maintain texts for several languages, save the changes before you switch to another language. Otherwise, you will lose input.

  The selected language is shown in the **Language** column of the table.

Displaying a Comparison Language

For comparison purposes, you can display the texts of another translation file in the **Comparison Language** column. The **Comparison Language** drop-down list box contains an entry for each subdirectory in the *multilanguage* directory which belongs to your user interface component.

To display another language for comparison

- From the **Comparison Language** drop-down list box, select the required language.

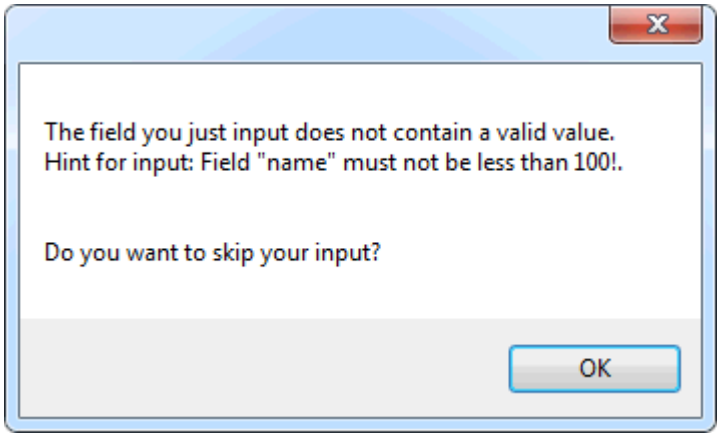
  The selected language is shown in the **Comparison Language** column of the table.

7 Using the Validation Rules Editor

- About the Validation Rules Editor 78
- Opening the Validation Rules Editor 78
- Adding a New Validation Rule 79
- Displaying an Overview of All Defined Validation Rules 81

About the Validation Rules Editor

The validation rules editor can be used to define that the value that is entered in a field must meet certain conditions. You can test your validation rule in the preview area. If an invalid value is entered, a corresponding message appears. Example:



Important: The validation is done on the client side.

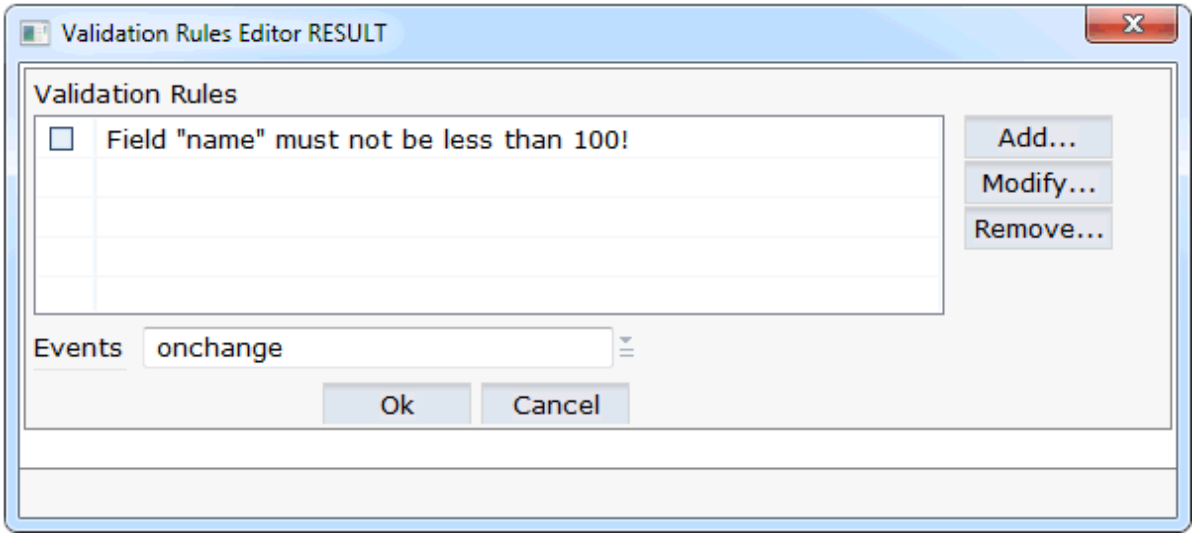
Opening the Validation Rules Editor

When you open the validation rules editor, you can add, modify and remove validation rules.

To open the validation rules editor

1. In the layout area or preview area, select the field control that you want to validate.
2. From the **Ajax Developer** menu, choose **Insert/Edit Validation Rules**.

   A dialog appears. If validation rules have already been defined, the messages that have been defined for these rules are shown in the text box. Example:



The following command buttons are provided:

Command Button	Description
Add	Adds a new validation rule. See below.
Modify	Modifies the selected validation rule.
Remove	Removes the selected validation rule.

Adding a New Validation Rule

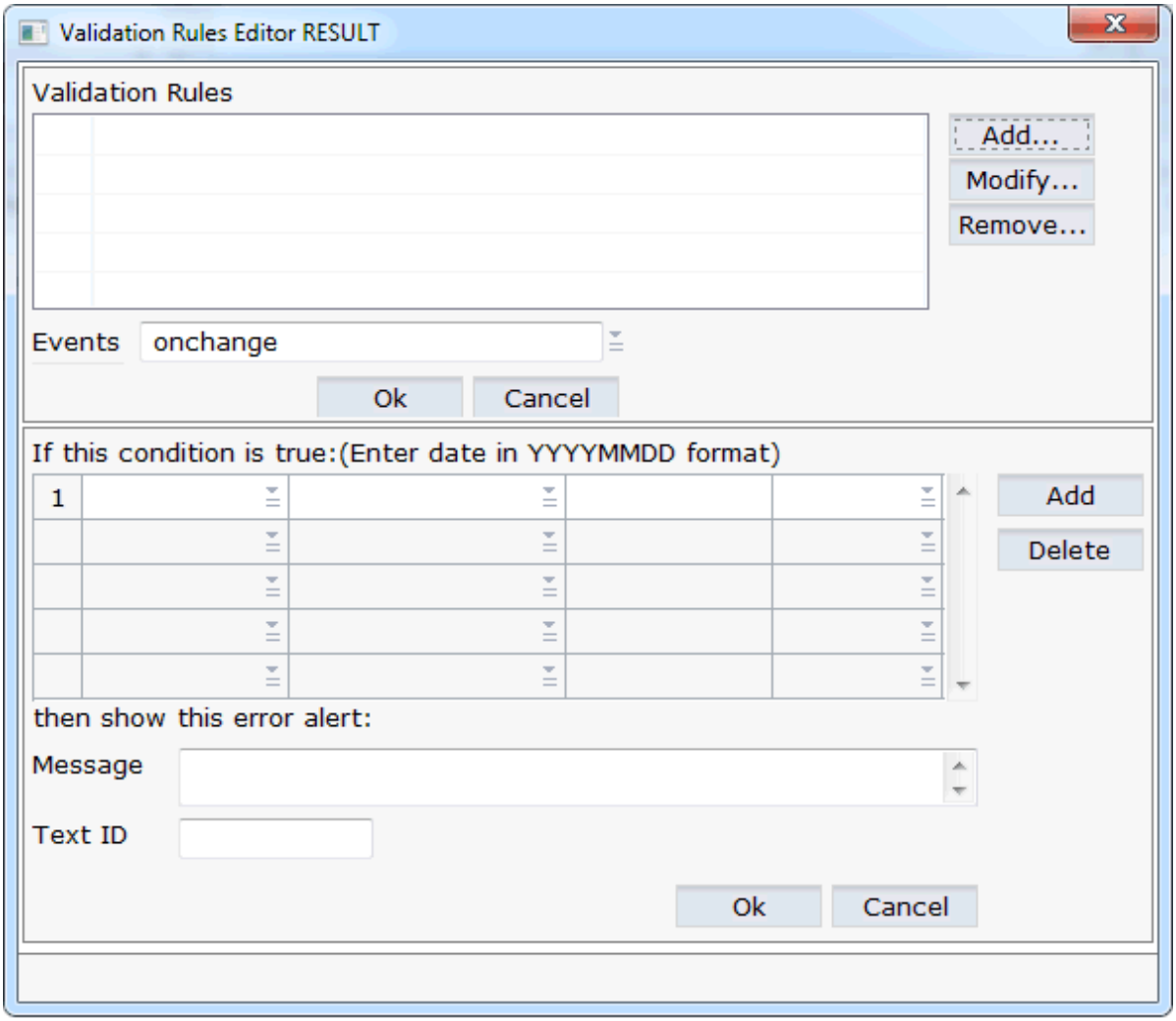
The prerequisite for adding a validation rule for a field is that an adapter class or Natural adapter with the corresponding code exists for the field. Otherwise, the field does not appear in the validation rules editor.

You can add more than one validation rule.

To add a new validation rule

1 To add a validation rule, choose the **Add** button.

Additional information is now shown at the bottom of the dialog.



2 From the first drop-down list box, select the name of the field for which you want to add a condition.

3 From the second drop-down list box, select the condition (for example, "less than").

4 In the third box, enter the value for the condition (for example, "100").

5 Optional. From the fourth drop-down list box, select the operator **And** or **Or** if you want to define an additional condition. In this case, you have to choose the **Add** button which is shown in the lower part of the dialog and specify the required values in the resulting line (as described above).

6 In the **Message** text box, enter the message that is to appear in a dialog box when an invalid value is entered (for example: "The value in the field must not be less than 100!").

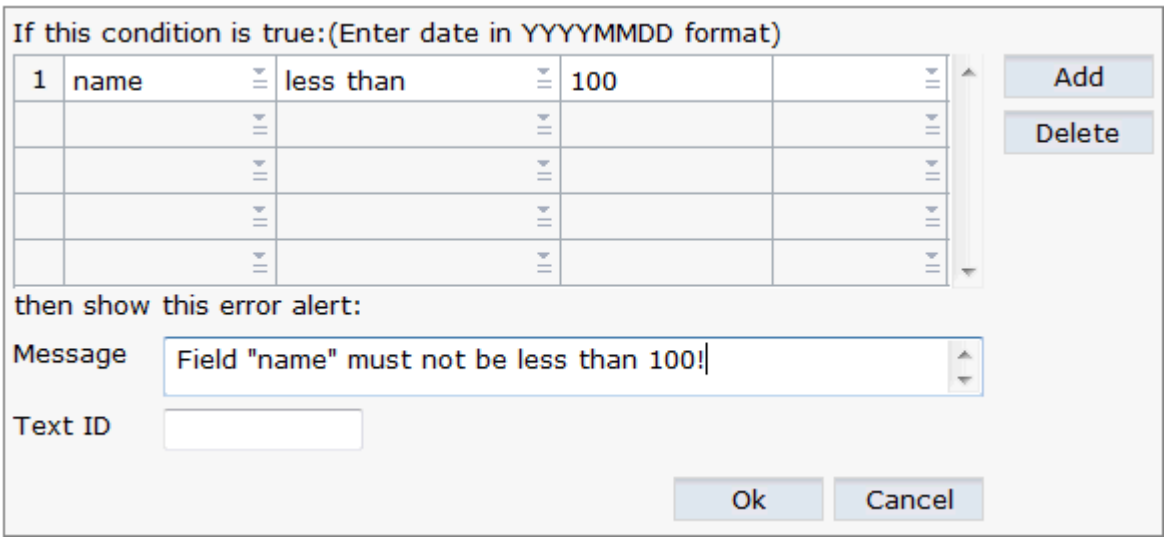
Or:

If you are working with several languages, enter a text ID in the **Text ID** text box. See also *Multi Language Management* in the *Natural for Ajax* documentation.

7 From the **Events** drop-down list box, select one of the following values. This defines the behavior in the case of an error.

Event	Description
onchange	Your message will appear when the user leaves the input field.
onsubmit	Your message will appear when the information is about to be transferred to the server (for example, by choosing a **Save** button).

The lower part of the dialog may now look as follows:



8 Optional. If you want to delete a condition, select the corresponding row and choose the **Delete** button.

9 To confirm the new validation rule, choose the **OK** button in the lower part of the window.

The dialog is not closed and you can add further validation rules.

10 To confirm all validation rules and to close the window, choose the **OK** button in the upper part of the window.

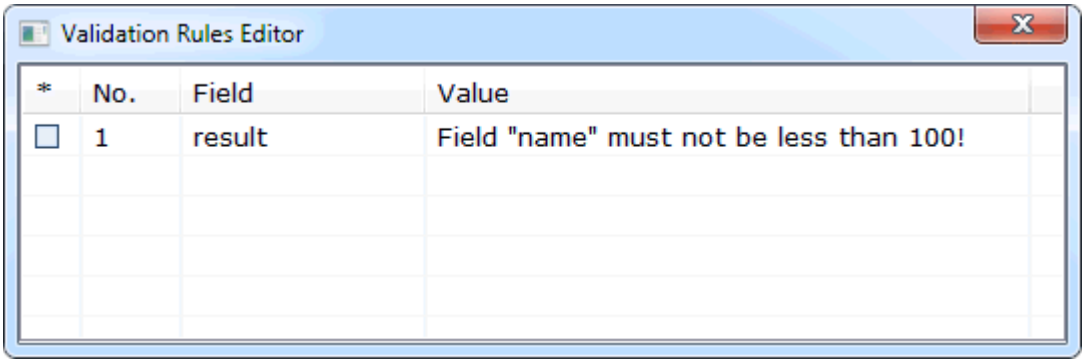
Displaying an Overview of All Defined Validation Rules

You can display a list of all validation rules which are defined in the current layout.

To display an overview

- From the **Ajax Developer** menu, choose **Validation Rules Overview**.

A dialog appears. When validation rules have been defined in the current layout, they are listed in this dialog.



When you click on a validation rule, the validation rules editor appears and you can change the validation rule as described above.

8 Using the Formula Editor

- About the Formula Editor 84
- Opening the Formula Editor 84
- Adding a New Formula 85
- Displaying an Overview of All Defined Formulae 87

About the Formula Editor

The formula editor can be used, for example, to define that the sum of the fields A and B is to be shown in field C. In this case, the formula has to added to field C.

Important: The formula is processed on the client side.

Opening the Formula Editor

When you open the formula editor, you can add or remove a formula.

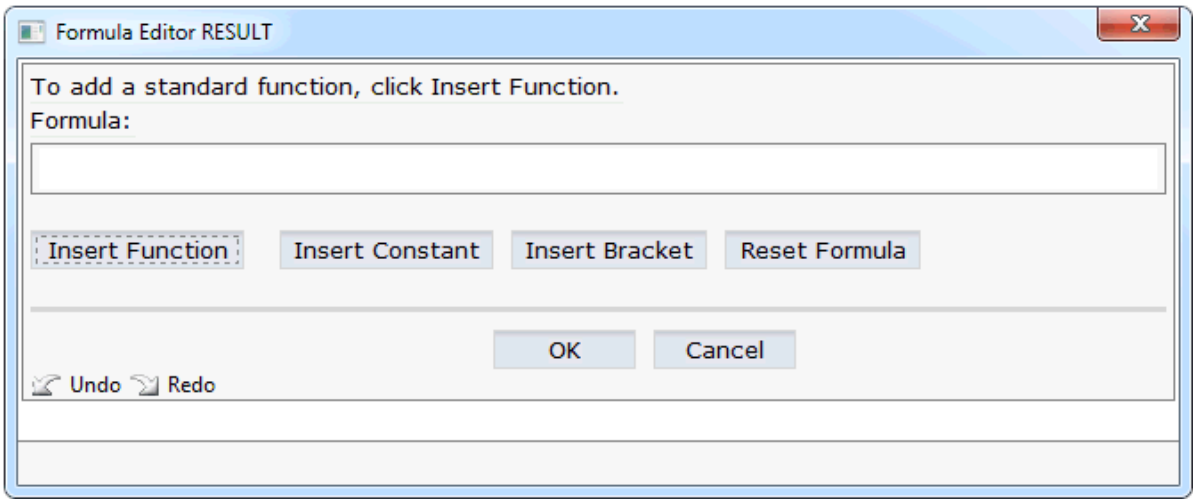
The prerequisite for adding a formula for a field is that an adapter class or Natural adapter with the corresponding code exists. The formula editor only works with fields that have valid declared properties on the adapter side.

To open the formula editor

1 In the layout area or preview area, select the field control for which you want to add a formula.

2 From the **Ajax Developer** menu, choose **Insert/Edit Formula**.

A dialog appears. If a formula has already been defined for the selected field, it is shown in the dialog.



Important: A function is always defined from the left to the right. It is not possible to go back in the function, for example, in order to insert a missing operator. Therefore,

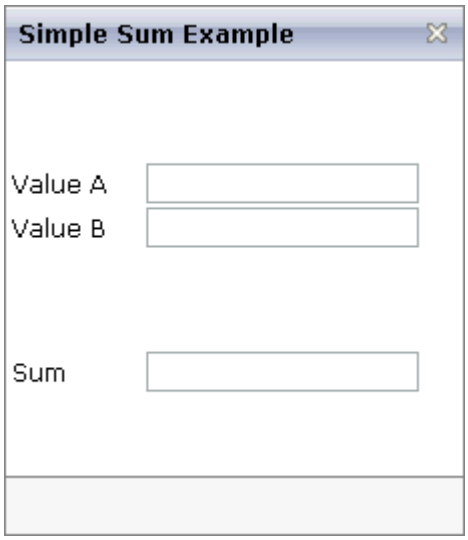
you should already have a plan of your function in mind before you compose the function using the controls in the formula editor.

The following command buttons are provided:

Command Button	Description
Insert Function	Allows you to select a function (such as **sum**) from a drop-down list box. After you have selected a function, you define the required fields using an "insert field" link. A simple example is provided below; see *Adding a New Formula*.
Insert Operator	This button is only shown when applicable, that is, if at least one expression has been entered. Allows you to select an operator (such as the plus sign) from a drop-down list box.
Insert Constant	Allows you to define a constant (for example, the number "77") in a text box.
Insert Bracket	Allows you to select brackets from a drop-down list box.
Reset Formula	Deletes the formula.

Adding a New Formula

This section explains how to define a simple formula: the sum of the fields **Value A** and **Value B** is to be shown in the **Sum** field.



To add a simple formula

It is assumed that the formula editor has been invoked as described **above** (the field was selected in which the sum is to be shown).

1 Choose the **Insert Function** button.

The following information is now shown at the bottom of the dialog.

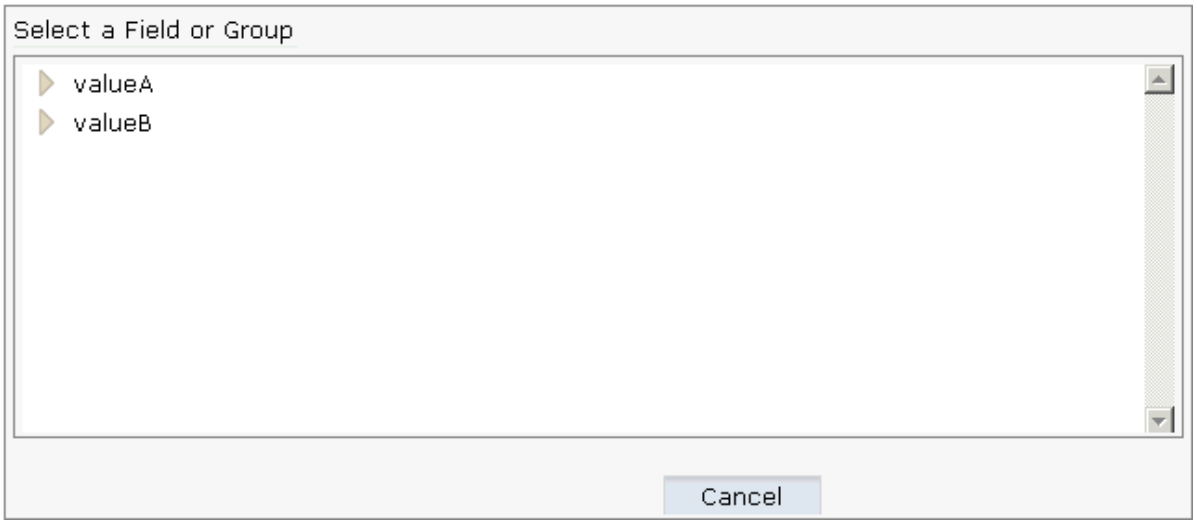
2 Select the required function (for this example, select **sum**) from the **Functions** drop-down list box and choose the **OK** button.

The information at the bottom of the dialog is no longer shown. The selected function is now shown in the **Formula** text box, together with the link "insert field".

Note: It is not possible to edit the formula in the **Formula** text box.

3 Choose the "insert field" link in the **Formula** text box.

The fields that are currently defined in the layout are now shown at the bottom of the dialog. Example:



4 Choose a field (for this example, choose **valueA**).

The selected field is now shown in the **Formula** text box. The link "insert field" is still shown in case you want to insert more fields.

5 Insert all required fields as described above (for this example, choose **valueB**).

For this example, the formula should look as follows:

```
sum(valueA,valueB)
```

6 When the formula is complete, choose the **Cancel** button.

7 To close the formula editor, choose the **OK** button.

You can now test the formula in the preview area.

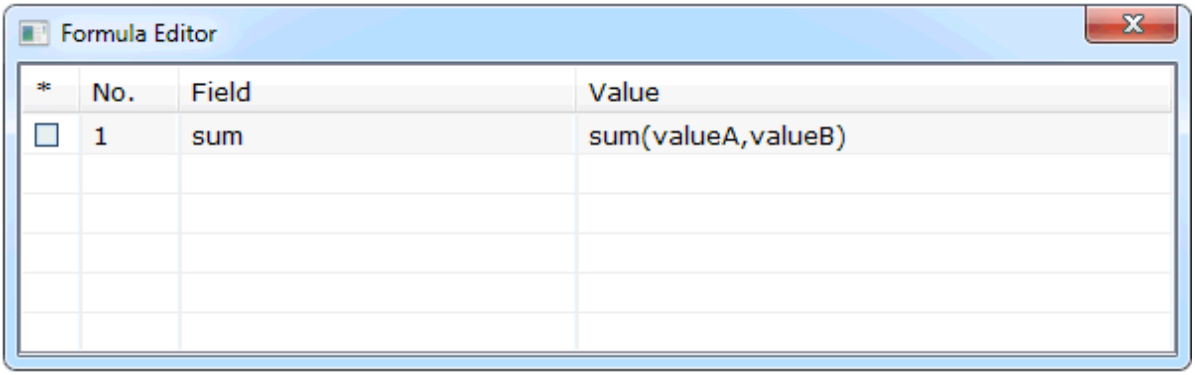
Displaying an Overview of All Defined Formulae

You can display a list of all formulae which are defined in the current layout.

To display an overview

- From the **Ajax Developer** menu, choose **Formula Overview**.

  A dialog appears. When formulae have been defined in the current layout, they are listed in this dialog.



  When you click on a formula, the formula editor appears and you can change the formula as described above.

III

- 9 Language Manager 91
- 10 Literal Translator 97
- 11 Style Sheet Editor 103

9 Language Manager

- About the Language Manager 92
- Invoking the Language Manager 92
- Defining a New Language 93
- Opening an Existing Language 95

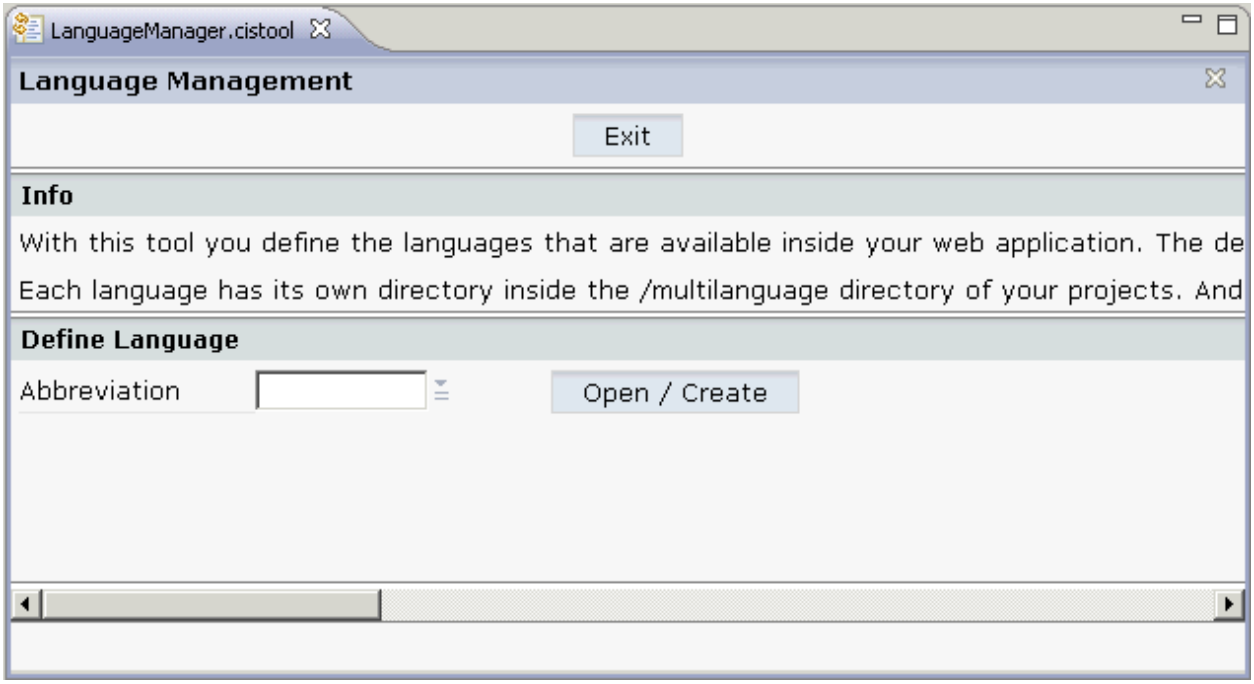
About the Language Manager

Two languages are supported by default: "en" for English and "de" for German. The Language Manager can be used to define additional languages.

The concept of the multi language management is described in *Multi Language Management* in the *Natural for Ajax* documentation. It is recommended that you read this information before you proceed with the information below.

Invoking the Language Manager

When you invoke the Language Manager, the following dialog appears.



To invoke the Language Manager

1. In the **Project Explorer** view, select the project for which you want to invoke the Language Manager.
2. Invoke the context menu and from the **Ajax Developer** menu, choose **Language Manager**.

Defining a New Language

When you define a new language, the required directories and certain files holding textual information will automatically be created in all existing projects.

To define a new language

1 In the text box of the **Abbreviation** drop-down list box, enter the abbreviation for your new language.

2 Choose the **Open / Create** button.

The following information is shown.

Translate the literals and message texts listed below and press "Save" afterwards.

Literals **Messages**

Date Input		Today	
Time Input		Hour	
Text Input		Minute	
Number Input		Second	
OK		File Upload	
		Upload	
January		Mo	
February		Tu	
March		We	
April		Th	
May		Fri	
June		Sa	
July		Su	
August		Ctrl	
September		Alt	
October		Shift	
November			
December		No match for	

3 Translate all literals that are shown on the **Literals** tab.

4 Choose the **Messages** tab.

5 Translate all strings that are shown on the **Messages** tab.

Each "\n" in a string stands for a line break.

"REPLACE" is a placeholder for a variable. It must not be deleted. During runtime, the corresponding value will be used. Example:

Language 1: \nHint for input: REPLACE.\n\n\n

Language 2: \nHinweis für die Eingabe: REPLACE.\n\n\n

6 Choose the **Save Language** button at the bottom of the dialog.

The directories and files for the specified language abbreviation are created. A message appears in the status bar of the Language Manager.

If you want to find out which directories and files were created, click the message in the status bar.

Opening an Existing Language

You can modify the text for literals and messages that you have specified when you have defined a new language (see above).

To open an existing language

1. In the text box of the **Abbreviation** drop-down list box, enter the abbreviation of an existing language.
2. Choose the **Open / Create** button.

   A dialog appears showing the currently defined texts for the literals and messages.
3. Edit the required texts on the **Literals** and/or **Messages** tab.
4. Choose the **Save Language** button at the bottom of the dialog.

   The texts are written to the files which have been created for the specified language abbreviation. A message appears in the status bar of the Language Manager.

   If you want to find out which files were affected, click the message in the status bar to display a dialog.

10 Literal Translator

- About the Literal Translator 98
- Invoking the Literal Translator 98
- Loading the Literals of a Layout 100
- Editing the Literals 101
- Adding a New Text ID 101
- Removing Text IDs 102

About the Literal Translator

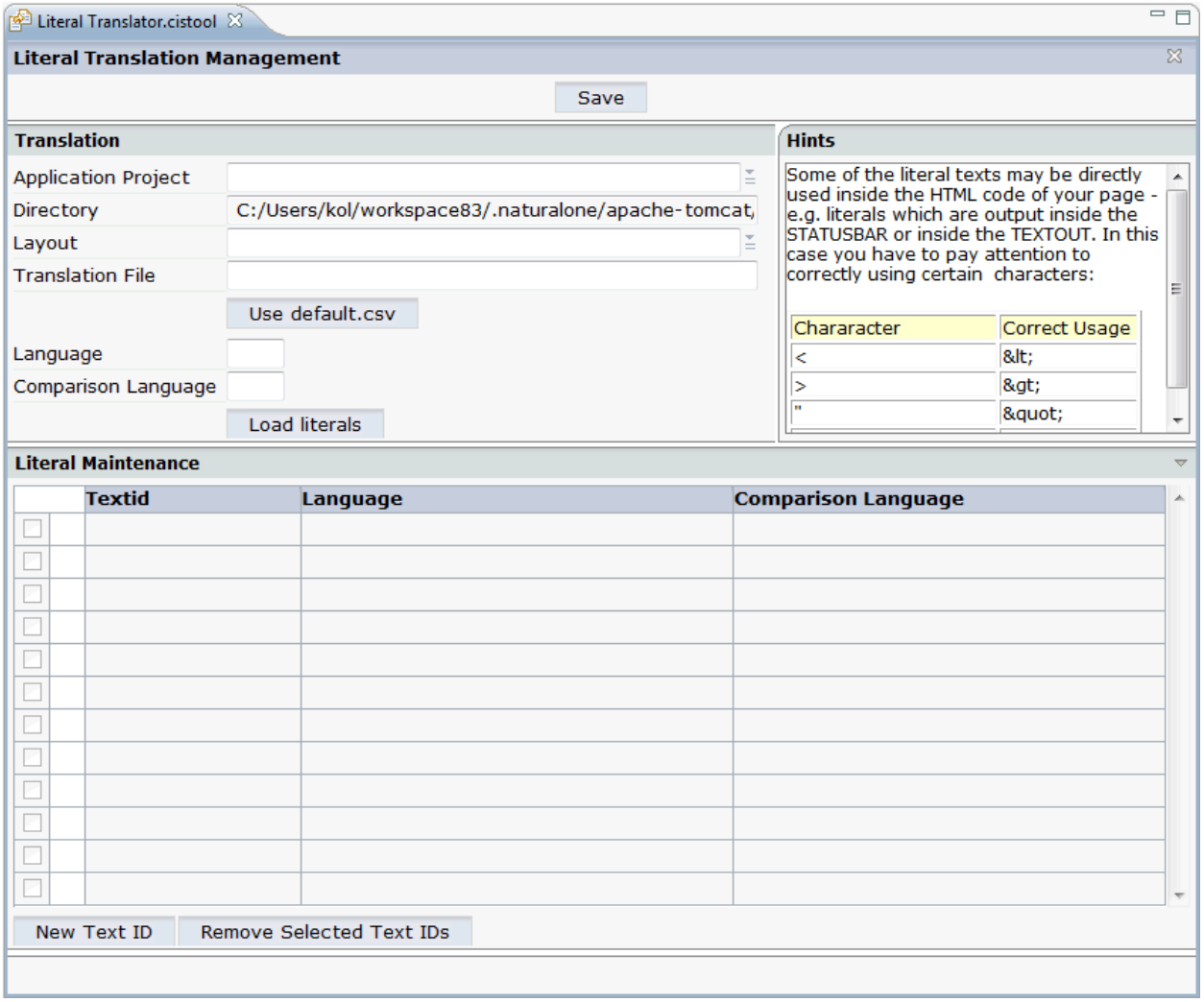
The Literal Translator can be used to translate text IDs. You can also use it to add and remove text IDs.

Note: Text IDs can also be translated using the **Literal Assistant**.

The concept of the multi language management is described in *Multi Language Management* in the *Natural for Ajax* documentation. It is recommended that you read this information before you proceed with the information below.

Invoking the Literal Translator

When you invoke the Literal Translator, the following is shown:



To invoke the Literal Translator

1 In the **Project Explorer** view, select the project for which you want to invoke the Literal Translator.

2 Invoke the context menu and from the **Ajax Developer** menu, choose **Literal Translator**.

Loading the Literals of a Layout

You can load and modify the texts for a specific layout in one language. Optionally, can also load the literals of a second language for comparison purposes.

Note: If you want to open the literals for a different layout or project, you have to close the Literal Translator and start it once more. Otherwise, the required options are not enabled.

To load the literals of a layout

1 In the text box of the **Application Project** drop-down list box, enter the name of the project which contains your layout.

Or:

Open the drop-down list box and choose the project from the resulting dialog.

2 In the text box of the **Layout** drop-down list box, enter the name of the layout for which you want to translate the literals.

Or:

Open the drop-down list box and choose the layout from the resulting dialog.

3 In the **Translation File** text box, enter the name and extension of your translation file.

Notes:

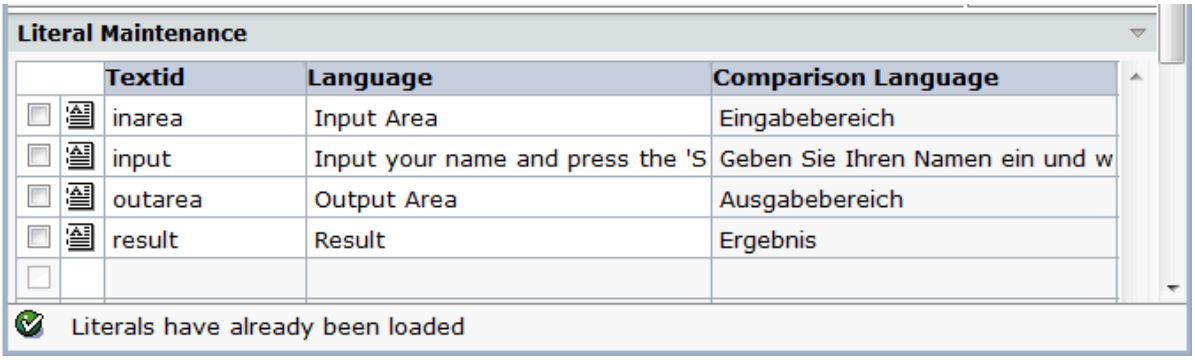
1. It is not required that the specified translation file already exists. You can **add** new text IDs as described below. When you choose the **Save** button, the translation file will be created and all of your definitions are written to it. Literals that have been created in this way, still have to be assigned to the corresponding controls, for example, using the **Literal Assistant**.
2. If you do not enter the name of a translation file, *default.csv* is automatically provided in this text box.
3. You can use the **Use default.csv** button to enter *default.csv* in this text box.

4 In the **Language** text box, enter the abbreviation for the language that you want to use. See also *Language Manager*.

5 Optional. If you want to display the literal of a second language for comparison purposes, enter the abbreviation for this language in the **Comparison Language** text box.

6 Choose the **Load literals** button.

The contents of the translation file(s) are loaded and the literals are shown at the bottom. Example:



Editing the Literals

You can edit the literals that are shown in the **Language** column. It is not possible to edit the literals of the comparison language.

To edit the literals

1 Enter the new text directly in the **Language** column.

Or:

To edit a literal in a separate dialog, choose the following icon which is shown next to the literal (this is helpful with long literals):

If you have opened a separate dialog, you have to choose the **Take Over** button in order to confirm your changes and to close the dialog.

2 Choose the **Save** button to write your changes to the translation file.

Adding a New Text ID

You can add a new text ID to the translation file and specify a literal for it.

To add a new text ID

1 Choose the **New Text ID** button which is shown below the list of literals.

An empty line is added to the list.

2 Enter the literal in the **Language** column.

3 Choose the **Save** button to write the new text ID and literal to the translation file.

Removing Text IDs

You can remove one or more text IDs from the translation file.

To remove text IDs

1 Select the check boxes in front of the text IDs that you want to remove.

2 Choose the **Remove Selected Text ID(s)** button which is shown below the list of literals.

3 Choose the **Save** button to remove the text IDs from the translation file.

11 Style Sheet Editor

- About the Style Sheet Editor 104
- Developing Style Sheets in Your User Interface Component 104
- Creating a New Style Sheet 104
- Opening an Existing Style Sheet 106
- Changing a Style Sheet 107
- Overview of Variables 109
- Applying your Stylesheet to Pages 110
- Regenerating Your Own Style Sheet from the Style Sheet Template 111
- Using a Version Control Tool 111

About the Style Sheet Editor

All controls that are contained inside the control library are rendered using a style sheet. Ajax Developer delivers a variety of predefined styles but also allows to you to create your own styles sheets.

The Style Sheet Editor simplifies the creation of your own style sheets: on the one hand, you can define the very basic style elements (main colors to be used), and on the other hand, you can change a controls' style definition on the lowest level.

For information on how to define a style sheet in the Layout Painter, see ***Defining a Style Sheet***.

For further information on style sheets, see *Adapting the Look & Feel*.

Developing Style Sheets in Your User Interface Component

Style sheets are part of your application's user interface and should therefore be maintained in a user interface component. When a user interface component is created, a subfolder named *styles* is automatically created in your user interface component. If you already have your own style sheets, copy them into this folder.

Remember to always use the Style Sheet Editor tool to modify the style. The file you need to check in into your version control systems is the **.info* file. The **.css* file is simply a build result of the **.info* file.

Be sure to put your style sheet into this predefined folder *styles*. Then the styles will automatically be built when building your project and they will automatically be packaged with your application when creating a deployment file.

Creating a New Style Sheet

When you create and save a new style sheet, it is written to the *styles* subfolder of your user interface component.

Note: By default, your pages are rendered with the *CIS_DEFAULT* style sheet which is one of Ajax Developers' predefined style sheets. Do not modify this style sheet because your modifications would be lost when you update to a new version. Instead, create your own style sheet as described below.

To create a new style sheet using the wizard

1 Use the **New Style Sheet** wizard to create a new style sheet.

From the context menu of the UI component, chose **New** > **Style Sheet**.

In the dialog of the wizard, enter a name for your style, for example "MyStyle" and click **Finish**.

2 The Style Sheet Editor is opened with "MyStyle" as **Variant** name.

Click **Open**.

3 Modify the values on the tabs according to your requirements. See *Changing a Style Sheet* for information on the **Style Details** tab.

4 To save your changes, click the **Save** button.

Note: Do not forget to click **Save** before closing the Style Sheet Editor or clicking the **Other Variant** button. Otherwise all changes will be lost without warning. When you choose the **Other Variant** button, you can open another style sheet or create another new style sheet. If you have not saved your input previously, it is lost.

When you save a new style sheet for the first time, it is created in your style sheet directory with the extension *.css*.

To create a new style sheet manually, based on CIS_DEFAULT

Alternatively, a new style sheet can be created manually based on CIS_DEFAULT following these steps:

1 In the */styles* folder of the ONE user interface component create a subfolder */images*

2 From the file system, copy *cisnatural/cis/styles/images/sag23*.** to *<myoneui>/styles/images*

3 From the file system, copy *cisnatural/cis/styles/CIS_DEFAULT.info* to *<myoneui>/styles*

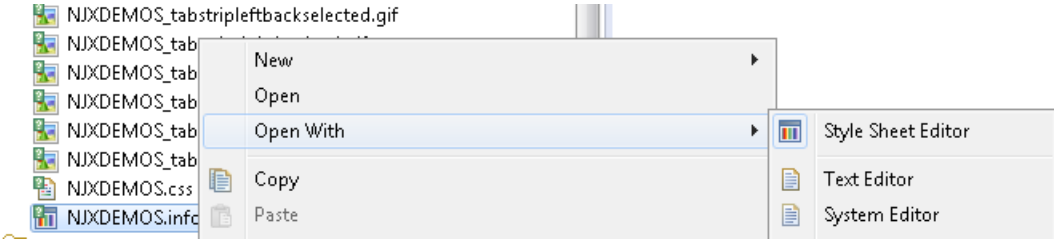
4 Rename *<myoneui>/styles/CIS_DEFAULT.info* to, for instance, *<myoneui>/styles/MYSTYLE.info*

5 Open and edit *MYSTYLE.info* with the style sheet editor tool and edit to requirements

6 Click **save** and close the style sheet editor tool

Opening an Existing Style Sheet

Right-click the *.*info* file in the *style* subfolder of your user interface component and open it with the Style Sheet Editor.



To open an existing style sheet

1 When the Style Sheet Editor is open you can also select a style name via the **Variant** drop-down list box. The style sheets of this user interface component and the central style sheets of the Natural Ajax framework are shown in the drop-down list.

> **Note:** In addition to the style sheet names, the drop-down list box also shows the paths to the corresponding files in the file system.

2 Choose the **Open** button.

The dialog with the style sheet definitions appears.

3 Modify the values on the different tabs according to your requirements. See *Changing a Style Sheet* for information on the **Style Details** tab.

> **Note:** When you choose the **Other Variant** button, the dialog is shown again which appears when you invoke the Style Sheet Editor. You can then open another style sheet or create a new style sheet. Any changes that you have applied after the last save are lost.

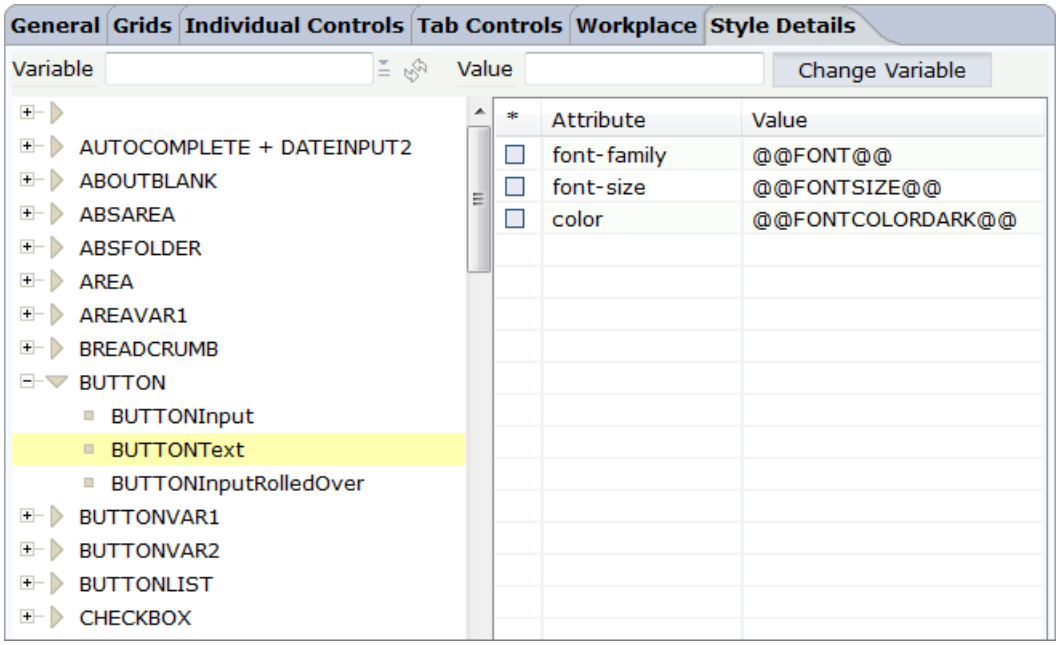
4 To save your changes, click the **Save** button.

Changing a Style Sheet

Ajax Developer provides an internal style template in which all the default style information is kept. A newly created style sheet is identical to this template. All style updates that you apply to your style sheet are stored within a separate file which has the extension *info*. For example, if you named your style sheet "MyStyle", your style sheet folder contains the file *MyStyle.info*.

Note: The style sheet folder also contains GIF files. They are generated according to your specifications for background, font, border color, etc. Each style sheet has thus individual GIF files. When you modify, for example, a color setting, new GIF files are generated which overwrite the previous GIF files.

When you edit a style sheet, you define the standard settings such as the font size on the **General** tab. All of the settings defined on the **General** tab are then assigned to individual controls on the **Style Details** tab via the corresponding variable (for example, `@@FONTSIZE@@`). See *Overview of Variables*.



To change an attribute value in a single style sheet class

1 Go to the **Style Details** tab.

2 Expand the tree on the left and select the name of a style sheet class.

Or:

In the **Variable** field select a specific variable from the list of variables.

The tree in the left pane will be rebuilt if a variable is selected from the list. The tree then shows only style sheet classes that contain the selected variable.

Expand the tree on the left and select the name of a style sheet class.

The attributes for the selected class are shown on the right.

3 Select the attribute that you want to change.

The line is marked as selected and the attribute name and value are now shown in the text boxes at the bottom.

4 Specify another value for the attribute in the text box at the bottom. For example, if you want to define another color, you can define a color value such as `#FF0000` (instead of `@@FONTCOLORDARK@@`).

You can also define a different name for a property. As soon as you choose the **Update** button, a new entry is created in the style sheet. This entry must be a valid CSS style sheet definition. For example, when you change the attribute name from `color` to `background-color`, this is a valid definition. When you specify an invalid attribute name, this will not have any effect. For example, when you change the attribute name from `color` to `hello`, this will not have any effect even though the new entry `hello @@FONTCOLORDARK@@` will be created.

5 Choose the **Update** button.

When you have changed a default entry of the style template, the changed information is now shown in bold in an additional line. For example:

*	Attribute	Value
☐	**color**	**#FF0000**
☐	font-family	@@FONT@@
☐	font-size	@@FONTSIZE@@
☐	color	@@FONTCOLORDARK@@

Notes:

1. If you do not want to take over a property from the style sheet template, specify "DELETE" as the attribute value and choose the **Update** button. All attributes that are marked with "DELETE" are not included in your style sheet.
2. If you want to take back a style update (that is: a change which is indicated by a bold line), delete the corresponding attribute value (that is: set it to blank) and choose the **Update** button. The bold line will disappear.

To change an attribute value in multiple classes

1 Go to the **Style Details** tab.

2 In the **Variable** field select a specific variable from the list of variables.

The tree in the left pane will be rebuilt if a variable is selected from the list. The tree then shows only style sheet classes that contain the selected variable.

To end the variable search you can choose the blank entry in the variable list.

3 Enter a new value.

The currently used value of the selected variable is displayed in the **Value** field. You can change the value of the selected variable by entering a new value.

4 Choose the **Change Variable** button to apply your changes.

The new attribute value will be applied to all style sheet classes using that variable

5 If a variable is updated by choosing the **Update** button, the icon (**Refresh list**) is activated to indicate that the search result that is currently displayed can be refreshed to display the most current results.

Overview of Variables

The following variables can be defined on the **Style Details** tab:

@@FONTCOLORDARK@@
@@FONTSIZE@@
@@FONT@@
@@LIGHTBACKGROUND@@
@@HEADLINEBACKGROUND@@
@@BORDERCOLOR@@
@@DARKBACKGROUND@@
@@TITLEBARBACKGROUND@@
@@FONTCOLORLIGHT@@
@@BUTTONHEIGHT@@
@@BUTTONIMAGE@@
@@BUTTONCOLOR@@
@@FONTCOLORINACTIVE@@
@@CONTROLERRORBACKGROUND@@
@@CONTROLEDITBACKGROUND@@
@@CONTROLDISPLAYBACKGROUND@@
@@LIGHTTITLEBARBACKGROUND@@
@@VARIANT@@

@@FIELDHEIGHT@@
@@FIELDBORDERCOLOR@@
@@CONTROLPOPUPINPUTONLYBACKGROUND@@
@@SHADEDDARKBACKGROUND@@
@@SELECTEDCELLBACKGROUND@@
@@SELECTEDBACKGROUND@@
@@EVENCELLBACKGROUND@@
@@ODDCELLBACKGROUND@@
@@TABAREALEFTPADDINGFIRST@@
@@TABAREALEFTPADDINGSECOND@@
@@EMPTYCELLBACKGROUND@@
@@SHADEDSELECTEDBACKGROUND@@
@@ODDCELLBACKGROUNDVAR1@@
@@EVENCELLBACKGROUNDVAR1@@
@@SELECTEDCELLBACKGROUNDVAR1@@
@@EMPTYCELLBACKGROUNDVAR1@@
@@ODDCELLBACKGROUNDVAR2@@
@@EVENCELLBACKGROUNDVAR2@@
@@SELECTEDCELLBACKGROUNDVAR2@@
@@EMPTYCELLBACKGROUNDVAR2@@
@@TITLEBARHEIGHT@@
@@COLORTOPIC1@@
@@COLORTOPIC2@@
@@COLORTOPIC3@@
@@COLORTOPIC4@@
@@COLORTOPIC5@@
@@COLORTOPIC6@@
@@COLORTOPIC7@@
@@COLORTOPIC8@@
@@COLORTOPIC9@@
@@COLORTOPIC10@@

Applying your Stylesheet to Pages

There are three options when applying own stylesheets:

- **Set a stylesheet for the whole application**

  To do so, set the attribute defaultcss in the *cisconfig.xml* as described in *Natural for Ajax > Client Configuration > Client Configuration > Central Class Path Extensions for Development*.

- **Apply a specific stylesheet at design time to a page**

  To do so, set the property stylesheetfile in the NATPAGE tag as described in section *Natural for Ajax > Natural for Ajax > Typical Page Layout > NATPAGE*.

- **Apply a specific stylesheet dynamically at runtime**

  To do so, use the NJX:SESSIONPARAMS control as described in section *Natural for Ajax > Natural for Ajax > Non-Visual Controls and Hot Keys > NJX:SESSIONPARAMS*.

Regenerating Your Own Style Sheet from the Style Sheet Template

From release to release, new controls are added to the control library. As a consequence, the style sheet template is typically enhanced with every new control (for example, new style classes are added). As your style changes are kept in a separate file which has the extension *info*, your style sheet can easily include the enhancements. If your styles are located in a subfolder *styles* of one of your user interface components, you simply have to rebuild your project to regenerate the style.

If you want to manually regenerate your style without doing a "build all" on the project level, do the following:

To regenerate your own style sheet manually

1. Open your style sheet in the Style Sheet Editor as described **above**.
2. Choose the **Refresh** button.

   The following command buttons are now shown: **Selected Variant** and **All Variants**.
3. Choose the **Selected Variant** button.

   Note: Choose the **All Variants** button if you want to regenerate all existing style sheets.

Using a Version Control Tool

If you want to secure your style sheets with a version control tool, it is sufficient to keep the files with the extension *info* under version control. The actual style sheet file (with the extension *css*) is just a generation result out of the INFO file.

IV Control Editor

You use the Control Editor to build your own custom controls.

The description of the Control Editor is organized under the following headings:

Using the Control Editor
Defining a Control
Examples

Note: Before you proceed with the information in this documentation, it is recommended that you first become familiar with the control development with Ajax Developer. For detailed information, see *Custom Controls*.

12 Using the Control Editor

- Invoking the Control Editor 116
- Creating an Editor Extension 117
- Adding a Control to an Editor Extension 119
- Adding a Data Type to an Editor Extension 120
- Deleting a Control or Data Type 123
- Saving an Editor Extension 123
- Opening an Editor Extension 124

Invoking the Control Editor

When you invoke the Control Editor, a dialog appears which shows a list of all editor extensions. These are the XML files with the prefix "editor_" which are stored in the central directory *<your-webapplication>/cis/config* and in the subdirectories *cisconfig* of your user interface components. A single editor extension may contain multiple controls. Ajax Developer comes along with a set of predefined editor extensions.

If you start the Control Editor for the first time, cancel the dialog and create a new editor extension. See *Creating an Editor Extension*.

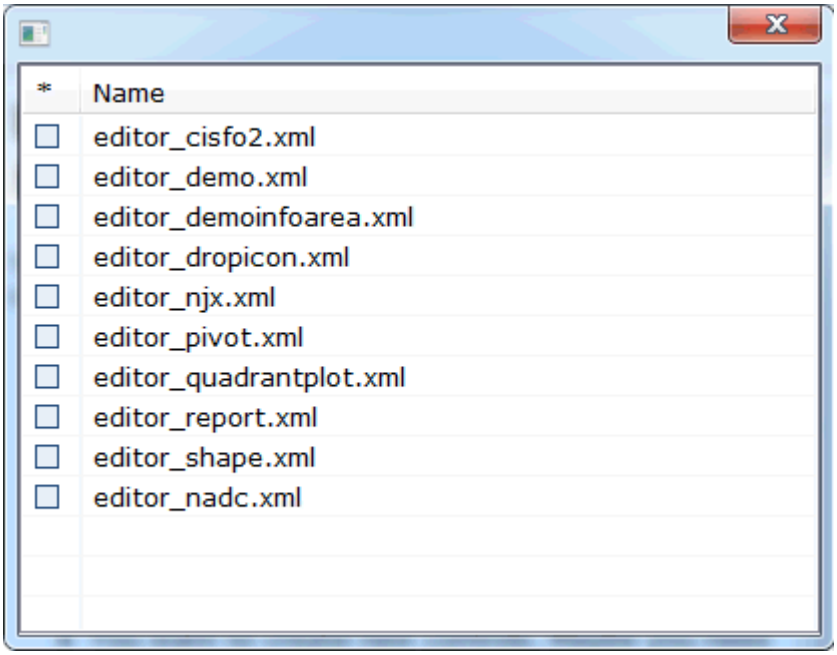
Important: Never touch an editor extension file you do not own. Develop your controls within your own extension.

To invoke the Control Editor

1 In the **Project Explorer** view, select the project for which you want to invoke the Control Editor.

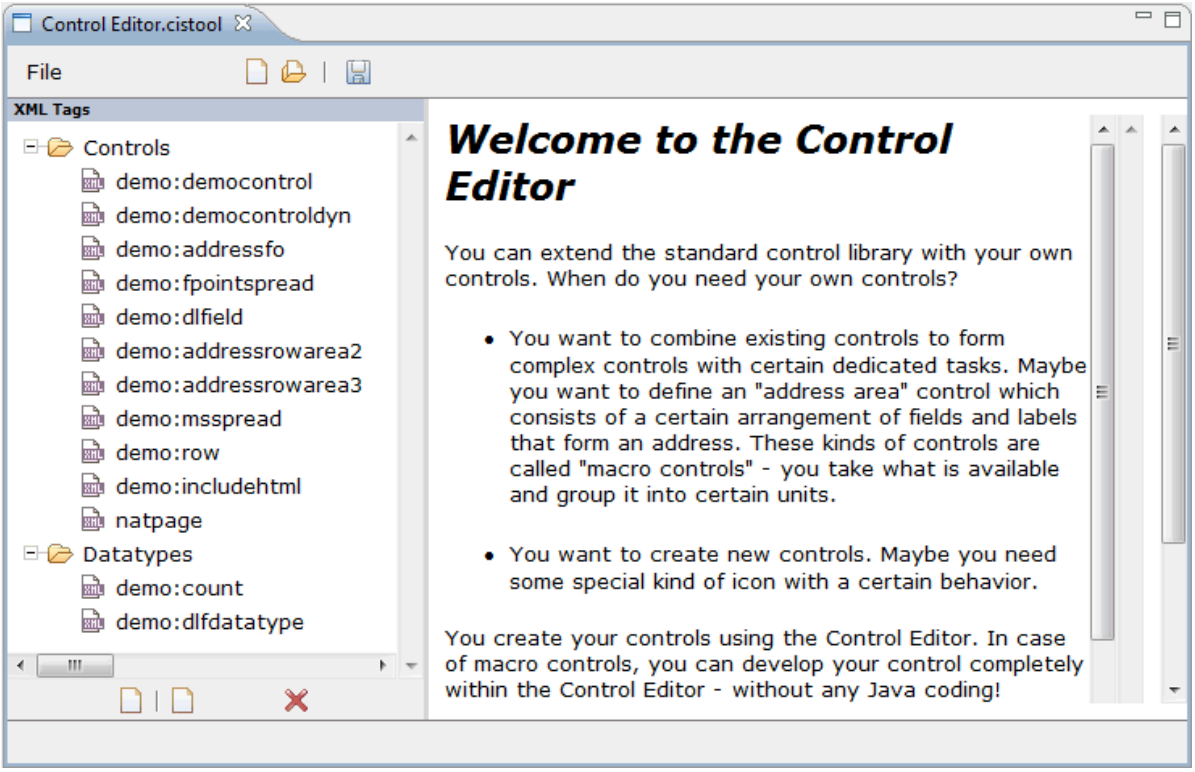
2 Invoke the context menu and from the **Ajax Developer** menu, choose **Control Editor**.

A dialog appears, listing all available editor extensions.



3 Choose the editor extension that you want to open.

The contents of the editor extension are loaded into the Control Editor. Example:



You can now edit your editor extension as described in the remainder of this section.

Note: You can also open an editor extension as described in *Opening an Editor Extension*.

Creating an Editor Extension

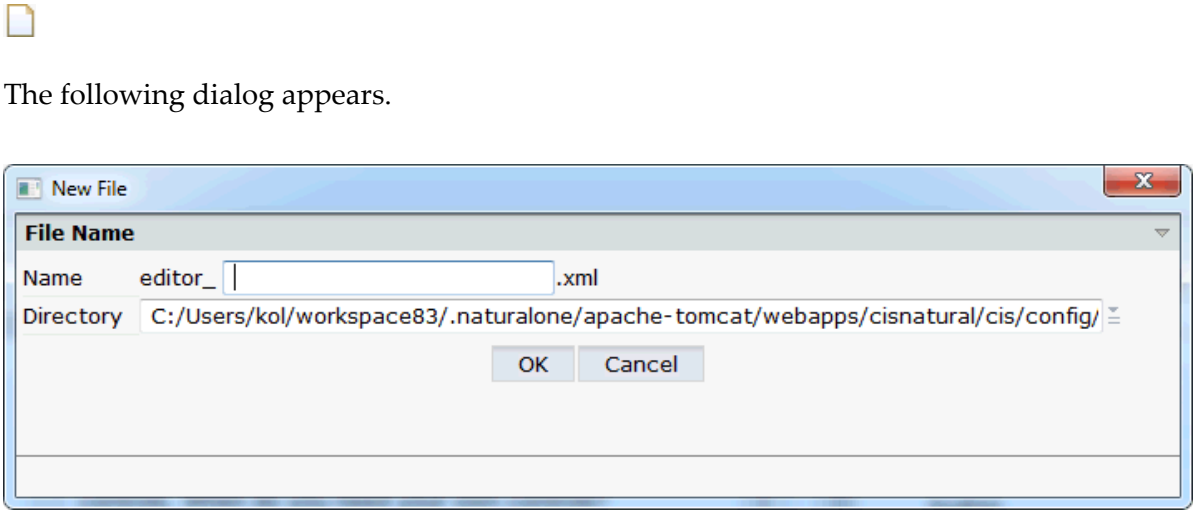
When you create a new editor extension, you can choose whether to stored it in the central directory *<your-webapplication>/cis/config/* or in the subdirectory *cisconfig* of your user interface component.

To create an editor extension

1 From the **File** menu of the Control Editor, choose **New**.

Or:

Choose the following button from the toolbar of the Control Editor.



2 Enter the name of your editor extension (for example, "mycontrols").

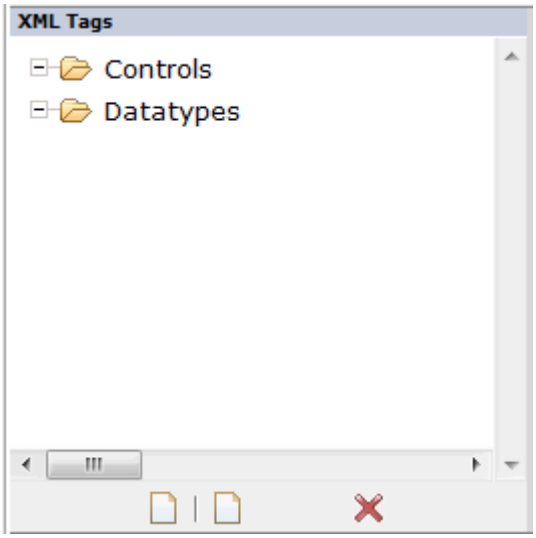
3 From the **Directory** drop-down list box, select the directory in which the new editor extension is to be stored.

The drop-down list box shows the full path names. The names of your user interface components are shown in the path names.

4 Choose the **OK** button.

The name of your new editor extension is composed of the prefix "editor_" and the name you have specified (for example, *editor_mycontrols.xml*). The extension is always *xml*.

The following empty nodes are now shown.



You can now add **controls** and **data types** as described below.

Adding a Control to an Editor Extension

A control consists of the control's definition (attributes and positioning) and its corresponding tag handler (Java class).

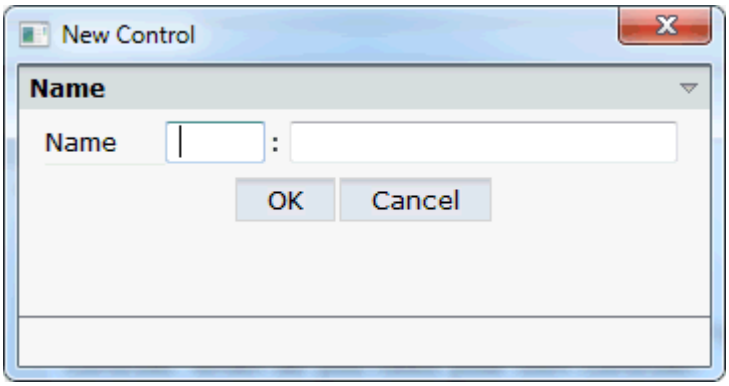
When defining macro controls, you do not have to write your own tag handler class. You can define the inner structure directly within XML (XML/protocol extension).

When defining completely new controls which implement their own HTML/JavaScript, you must also implement a corresponding tag handler in order to generate your own HTML/JavaScript code.

To add a control

1 Choose the first button which is shown below the tree of XML tags (when you move the mouse over this button, the tooltip “New Control” appears):

The following dialog appears.



2 Enter a name for the new control.

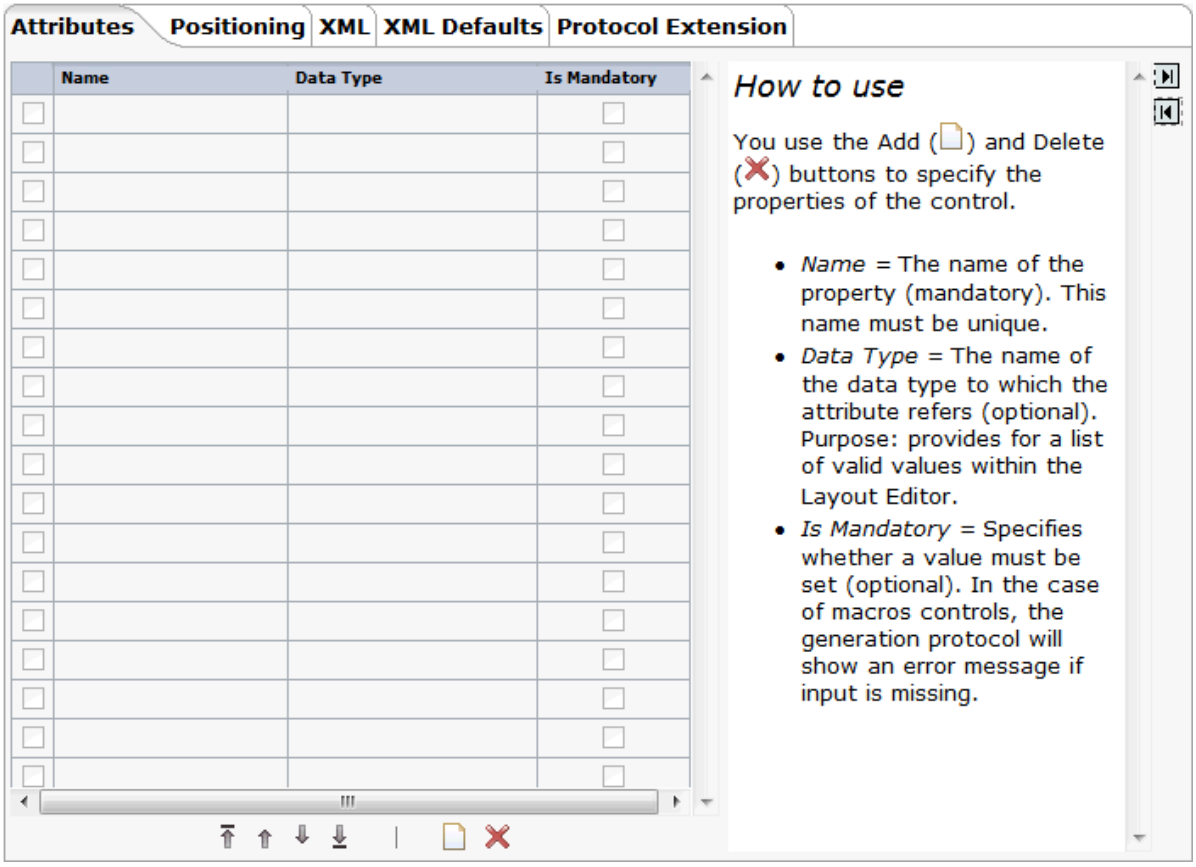
The name of a control must be unique within your control library. Therefore, you have to prefix the control name with the name of the control library: "`<control-library-name>:<control-name>`". See also *Library Concept* in *Custom Controls*.

3 Choose the **OK** button.

The new control is inserted in the **Controls** node. If the **Controls** node already contains controls, each new control is automatically inserted at the bottom of the list.

4 Select the new control in the tree.

The following information is shown.



Using the buttons ⏭ and ⏮ you can hide and show the help information in this dialog.

5 Specify all required information on the different tabs. See *Defining a Control* for detailed information on these tabs.

See also *Examples*.

Adding a Data Type to an Editor Extension

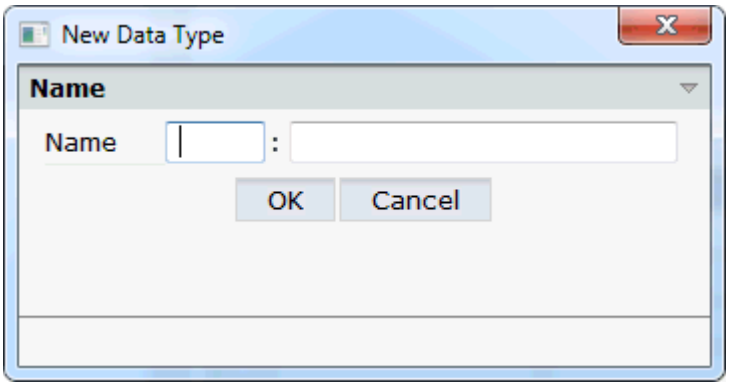
A data type defines a list of valid values. The properties of your controls can have data types. This is optional.

A set of predefined data types (such as `align` or `boolean`) is provided. You can use them within your control's definition. They are defined in the editor configuration file *editor.xml*. If there is no appropriate data type for your purpose, you can create your own data type.

To add a data type

1 Choose the second button which is shown below the tree of XML tags (when you move the mouse over this button, the tooltip “New Datatype” appears):

The following dialog appears.



2 Enter a name for the new data type.

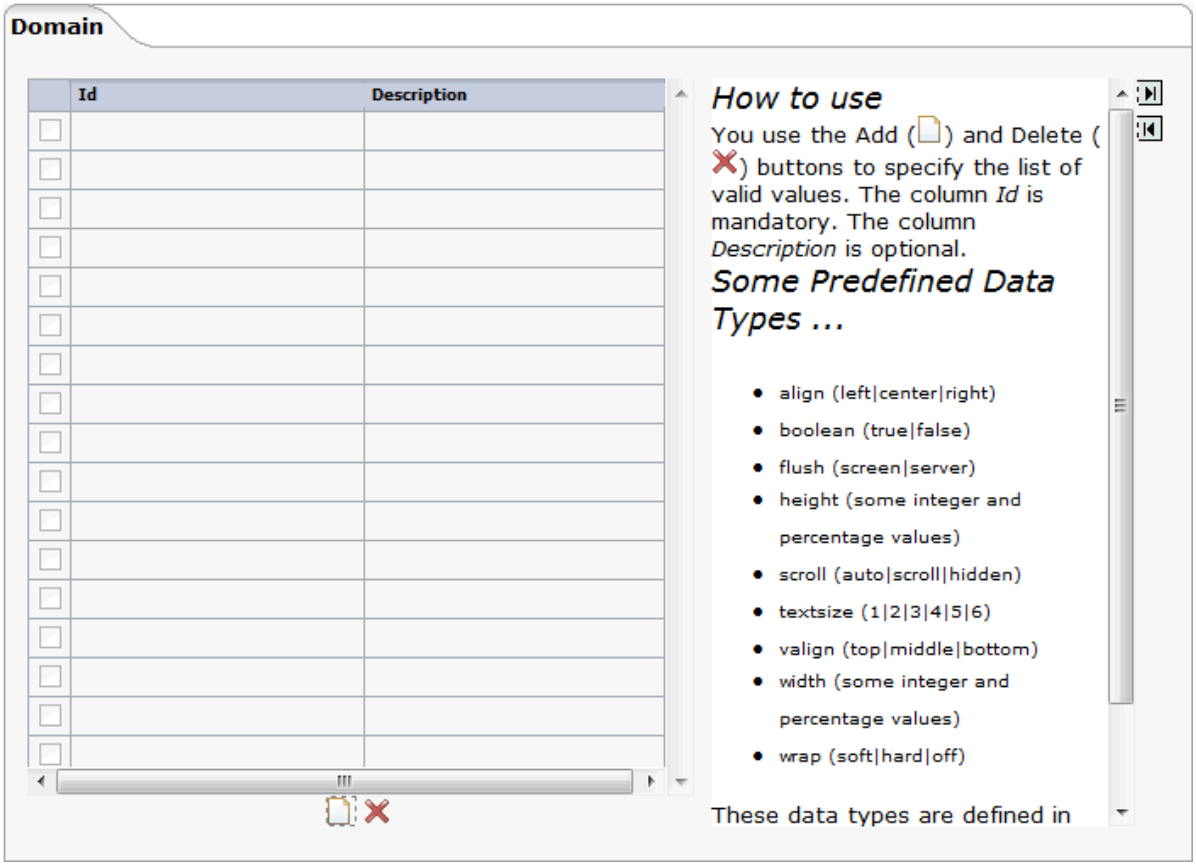
The name of a data type must be unique within your control library. Therefore, you have to prefix the data type name with the name of the control library: "`<control-library-name>:<data-type-name>`".

3 Choose the **OK** button.

The new data type is inserted in the **Datatypes** node. If the **Datatypes** node already contains data types, each new data type is automatically inserted at the bottom of the list.

4 Select the new data type in the tree.

The following information is shown.



Using the buttons and you can hide and show the help information in this dialog.

The following buttons are provided on this tab:

Button	Description
	Adds an empty line in which you can specify a data type.
	Deletes the selected data type(s).

5 Use the button to add one or more empty lines and specify the following:

Id

The name of the data type.

Description

Optional. A short description of your data type.

See also *Examples*.

Deleting a Control or Data Type

You can delete any controls or data types that are shown in the tree of XML tags.

To delete a control or data type

1 In the tree of XML tags, select the control or data type that you want to delete.

2 Choose the following button which is shown below the tree of XML tags:

You are not asked to confirm the deletion.

Saving an Editor Extension

When you save an editor extension, all of your changes in the Control Editor are saved. This includes all controls and data types that you have added or changed.

To save an editor extension

- From the **File** menu of the Control Editor, choose **Save**.

  Or:

  Choose the following button from the toolbar of the Control Editor.

  The status bar of the Control Editor shows the name of the file (including the path) to which the information has been written.

  Tip: If the path is longer than can be shown in the status bar, click on the status bar. You can then find out the complete path and file name in a separate dialog.

Opening an Editor Extension

You can open any editor extension that is stored in the central directory *<your-webapplication>/cis/config/* or in the subdirectories *cisconfig* of your user interface components.

To open an editor extension

1 From the **File** menu of the Control Editor, choose **Open**.

Or:

Choose the following button from the toolbar of the Control Editor.

Note: When your latest changes in the Control Editor have not yet been saved, you are asked whether you want to save them.

The **Open File** dialog appears.

2 Choose the editor extension that you want to open.

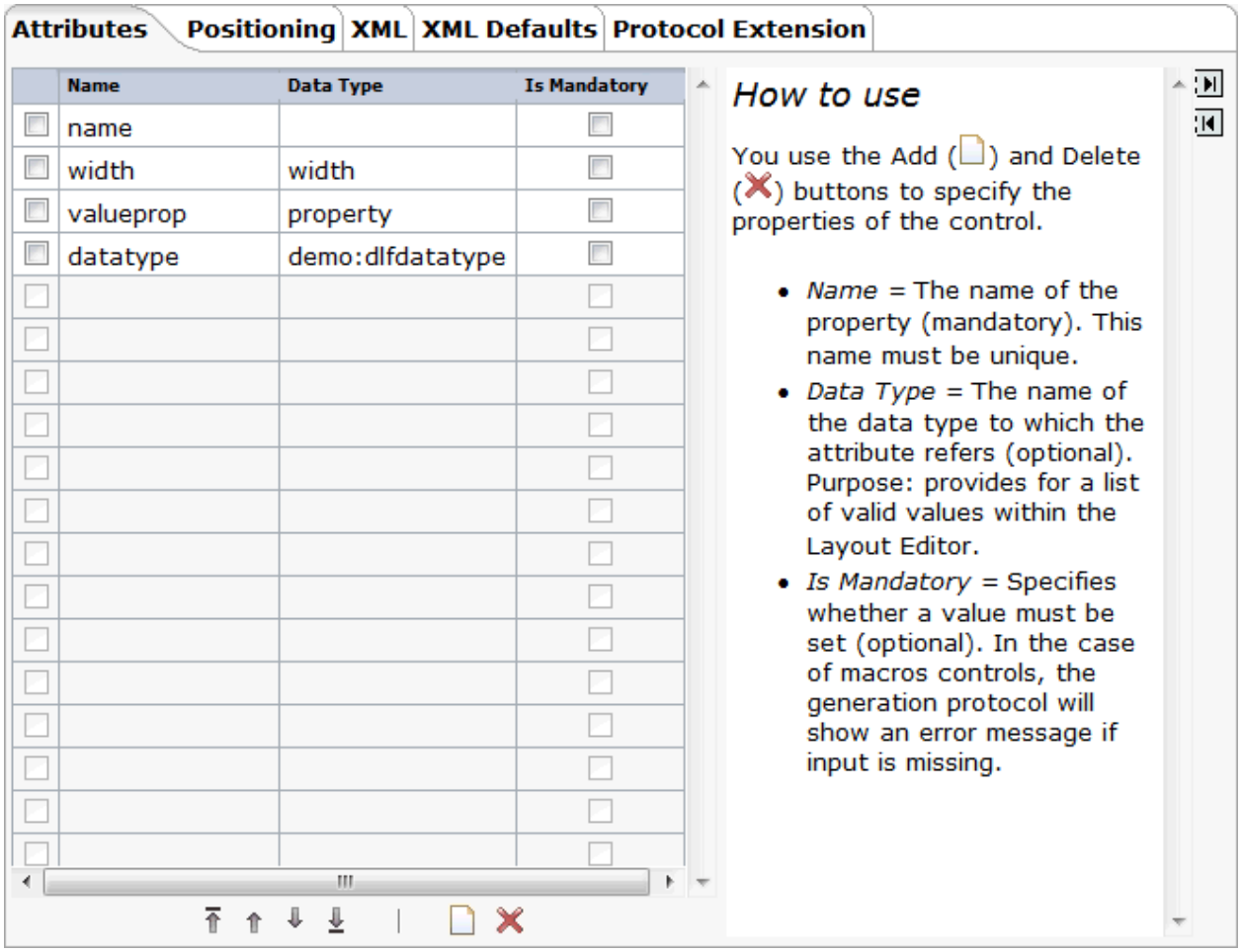
The contents of the editor extension are loaded into the Control Editor.

13 Defining a Control

- Attributes 126
- Positioning 127
- XML 129
- XML Defaults 130
- Protocol Extension 131

Attributes

The **Attributes** tab is used to specify the list of attributes.



The following buttons are provided below the list of attributes:

Button	Description
	Moves the selected attribute(s) up to first position in the list.
	Moves the selected attribute(s) up to the previous position in the list.
	Moves the selected attribute(s) down to the next position in the list.
	Moves the selected attribute(s) down to last position in the list.
	Adds an empty line in which you can specify a property.
	Deletes the selected attribute(s).

To define a property, use the button to add an empty line and specify the following information in this line:

Name
The name of a property. The name must be unique within this attribute list.

Data Type
Optional. The name of a data type to which the attribute refers. The data type can be an Application Designer data type (see the data type definitions in the file *editor.xml*) or a user-defined data type.

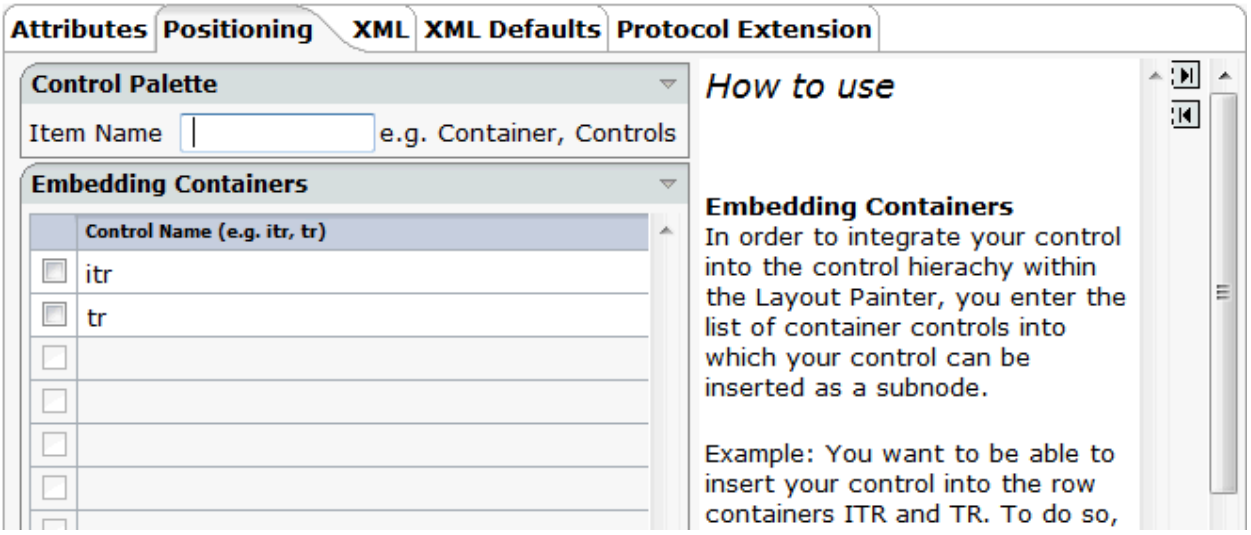
Is Mandatory
Optional. When this check box is selected, a value for the corresponding attribute must be set.

In the case of macro controls, the generation protocol will show an error message if input is missing.

Positioning

The **Positioning** tab is used to specify the following:

- the section of the controls palette which is to contain the control,
- the containers in which the user will be able to insert the control, and
- the subcontrols which the user can insert into the control.



The following buttons are provided on this tab:

Button	Description
	Adds an empty line in the corresponding list.
	Deletes the selected control(s) from the corresponding list.

Specify the following information:

Controls Palette

The section of the controls palette in which your new controls are to be included. You can either specify an existing section such as **Controls**, or you can specify a new section.

For example, when you specify the name "MyControls", you have to choose the **MyControls** button in the controls palette to display your controls.

Embedding Containers

This list determines the availability in the Layout Painter.

Use the button to add one or more empty lines and specify the names of the containers (such as ITR or TR) into which your new control can be inserted as a subnode. If specify your own controls, do not forget to use the library prefix.

In the Layout Painter, the new control can then be selected from:

- **Context Menu**
  The new control will be offered for selection in the Layout Painter when you invoke the context menu for an embedding container control.

  For example, when you define a new control with the name "test:mycontrol" and define the ITR control as an embedding container, this control will be available as follows:

  - **Add as first Subnode > Extensions > test:mycontrol**
  - **Add as last Subnode > Extensions > test:mycontrol**

- **Controls Palette**
  The new control will be available in the controls palette of the Layout Painter. It is shown when you open the corresponding section of the controls palette. For example, when you have defined that control is to appear in the **MyControls** section, you have to choose the **MyControls** button.

  The Layout Painter will use a default image in the controls palette. However, you can assign your own image; in this case, you have to observe the following rules:

  - The image must be a GIF file.
  - The name of the image must have the following structure:

    ctrl<*LIBRARY-NAME*>_<*CONTROL-NAME*>.gif

    For example:

ctrlTEST_MYCONTROL.gif

- The image must be stored in the directory *<your-webapplication>/cis/config/controlimages*.
- The preferred width is 16 pixels and the preferred height is 16 pixels.

Sub Controls

Your control can be a container itself (for example, you want to be able to insert the standard ICON control into your container). In this case, you enter the list of the subcontrols here.

Use the button to add one or more empty lines and specify the names of the subcontrols (such as ICON). If you specify your own controls, do not forget to use the library prefix.

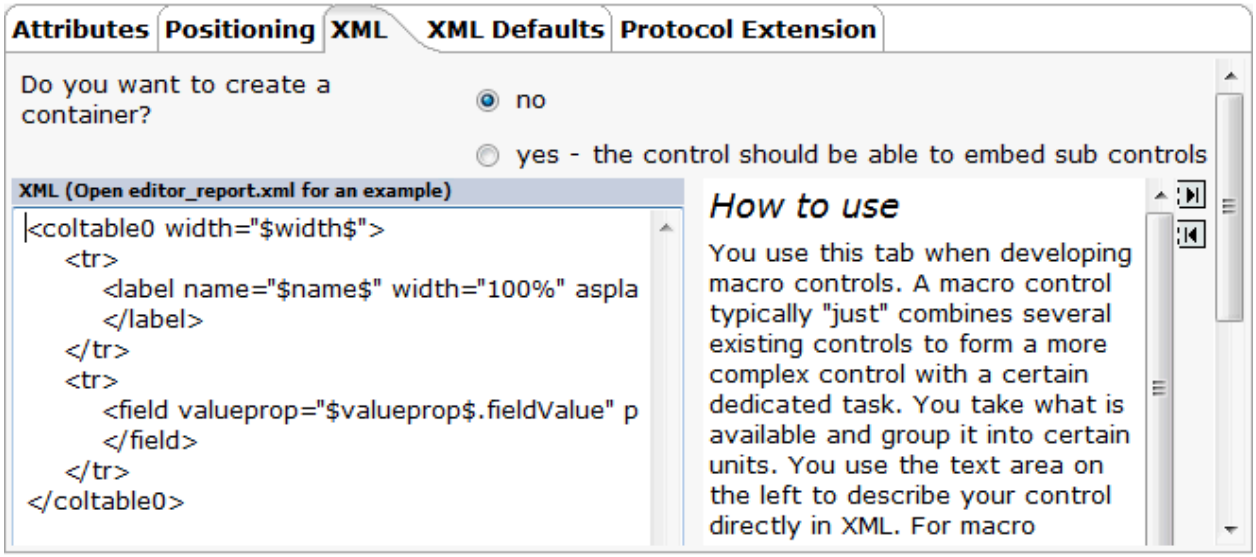
XML

The **XML** tab is used to specify a macro control.

Typically, your control combines existing controls to form a more complex control with a certain dedicated task. You take what is available and group it into certain units.

With a macro control, you compose your control directly in XML and therefore do not need to provide for a tag handler class.

See also *Defining a Macro Control* in the *Examples* section.



Specify the following information:

Do you want to create a container?

Select one of the following option buttons:

- **no**
  When this button is selected, your control cannot embed other controls. BUTTON is an example of such a control.
- **yes**
  When this button is selected, your control is a container control which can embed other controls. ITR and ROWAREA are examples of container controls.

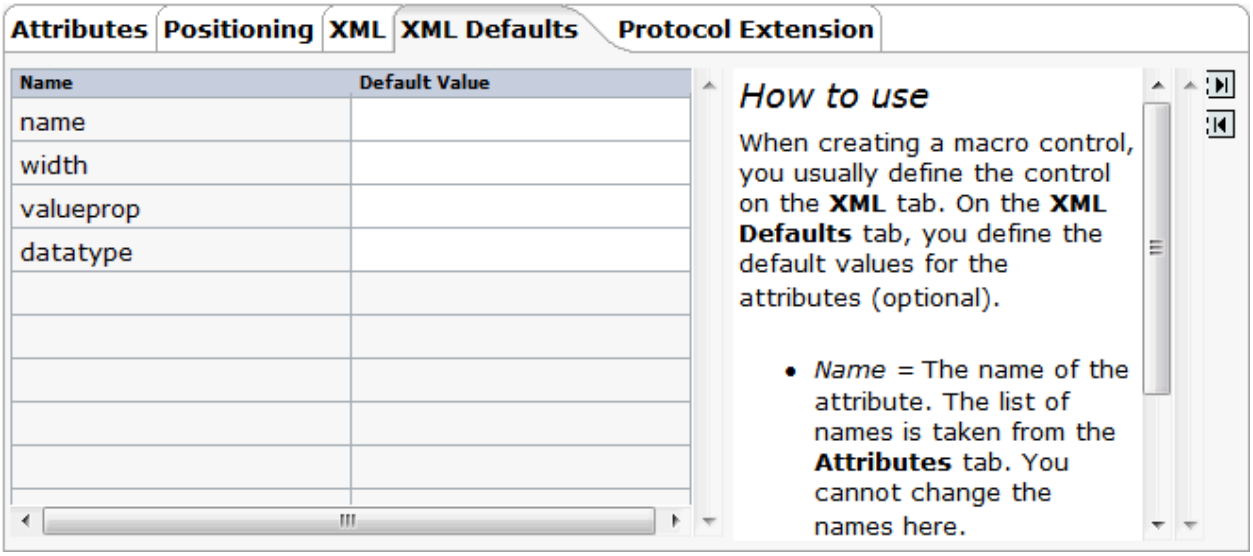
XML
Enter your macro in this text box.

XML Defaults

Only applies to a macro control.

When you have entered a macro control on the XML tab, you can use the **XML Defaults** tab to define the default values for the attributes (optional).

All attributes that are currently defined on the **Attributes** tab are automatically provided on the **XML Defaults** tab.



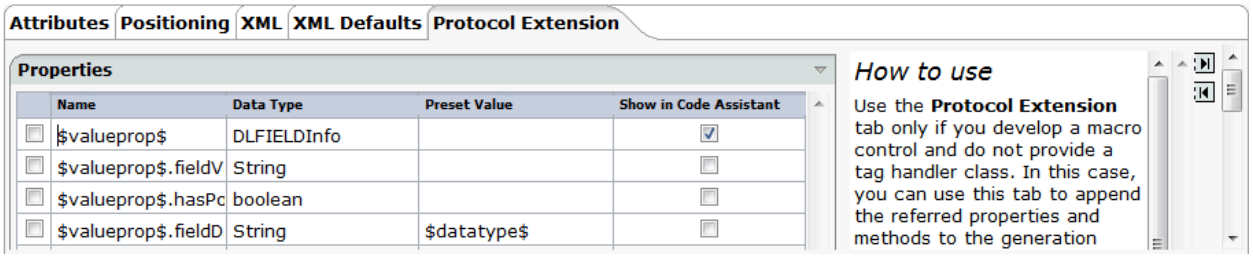
Specify the following information:

Default Value
The default value will be used if the user does not specify a value.

Protocol Extension

Use the **Protocol Extension** tab only if you develop a macro control and do not provide a tag handler class. In this case, you can use this tab to append the referred properties and methods to the generation protocol. If you provide a tag handler class, you implement the addition of properties and methods to the generation protocol in the tag handler class instead.

See also *XML Definition for a Macro Control with a Server-Side Representative* in the *Examples* section.



The following buttons are provided on this tab:

Button	Description
	Adds an empty line in the corresponding list.
	Deletes the selected line(s) from the corresponding list.

Specify the following information:

Properties

Use the button to add one or more empty lines and specify the following:

Name
The name of the adapter property that is used in your control.

Data Type
Optional. The data type of the property. This information is used in the Code Assistant (which is part of the Layout Painter) to generate appropriate property coding.

Note: This has no meaning for custom controls in NATPAGE layouts.

Preset Value
Optional. The value that is set in your adapter when the page is loading.

Show in Code Assistant

When this check box is selected, the property appears in the Code Assistant. By default, this check box is selected.

You should only select this check box for the properties that must be provided by the adapter.

Note: This has no meaning for custom controls in NATPAGE layouts.

Methods

Use the button to add one or more empty lines and specify the following:

Method Name

The name of the adapter method.

Show in Code Assistant

When this check box is selected, the method appears in the Code Assistant. By default, this check box is selected.

You should only select this check box for the methods that must be provided by the adapter.

Note: This has no meaning for custom controls in NATPAGE layouts.

JavaScript Libraries

Use the button to add one or more empty lines and specify the following:

Source

The URL that points to your JavaScript library.

14 Examples

- About the Examples 134
- Defining a Control with a Corresponding Tag Handler 134
- Defining a Macro Control 137
- Additional Information 146

About the Examples

These examples assume that you expect the label of an input field not at the left of the field but above the field (two separate rows). Furthermore, you want to have several such fields within a single row (inline control).

For these examples, you will proceed as follows:

1. You create your own editor extension with the name *editor_test.xml*.

   Important: Do not use the Application Designer editor extensions for your own controls. Place your own controls within your own extension file. Thus, there will be no conflicts when upgrading your installation to a newer build.

2. You build a control named DLFIELD (double line field) which will be available in the control library "test". Therefore, you have to name your control "test:dlfield". See also *Library Concept* in *Natural Custom Controls*.
3. You define a macro control which determines that the control consists of two rows, one for the label and another for the field. The rows themselves are placed into a column container.

The composition of controls looks roughly like this:

```
COLTABLEO
  TR
    LABEL
  TR
    FIELD
```

Defining a Control with a Corresponding Tag Handler

The definition of a control with a corresponding tag handler is quite simple. It consists of

- the name of the control,
- the list of attributes, and
- the list of embedding container controls.

You will now define a control with the following properties: `name` (label name), `width` (width of column container) and `valueprop` (name of the adapter property to which the field is bound). The control will be available within the row containers TR and ITR.

Note: For further information on the tag handler, see *Control Concept* in *Natural Custom Controls*.

To define a new control

1 Invoke the Control Editor.

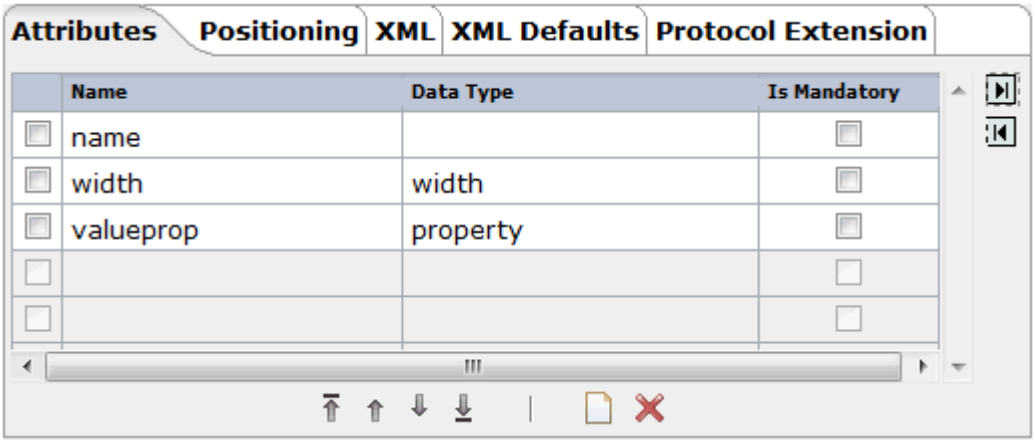
2 **Create your own editor extension** with the name *editor_test.xml*

3 **Add a new control** with the name "test:dlfield".

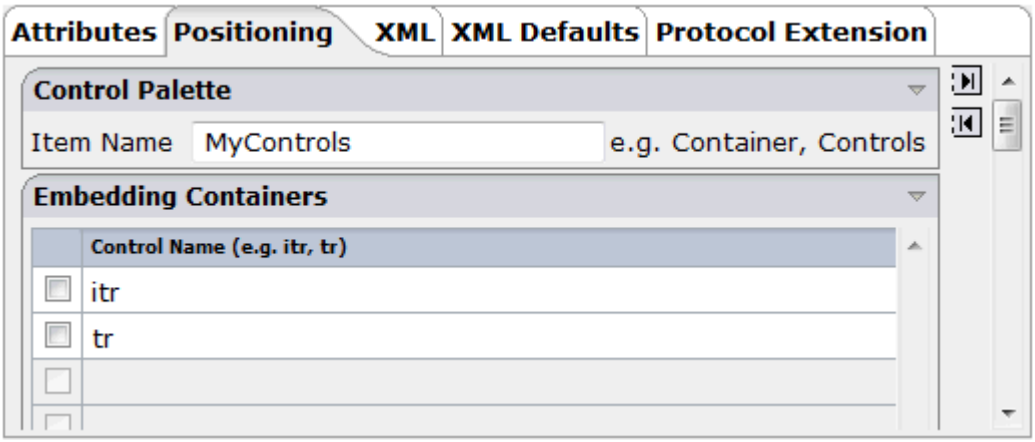
> **Note:** The editor extension *editor_demo.xml* contains a control with the name "demo:dlfield". When you have completed the example, your new "test:dlfield" extension should be similar to "demo:dlfield".

4 Select your new control in the tree.

5 To create the list of attributes, specify the following information on the **Attributes** tab:



6 Specify the following information on the **Positioning** tab:



This determines that your control can be selected from the **MyControls** section of the controls palette and that it can be inserted into the row containers TR and ITR.

7 Save your changes.

8 Have a look at the generated XML file (*editor_test.xml*). It should look as follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
Dynamic extension of editor.xml file.
-->

<controllibrary>
  <editor>

  <!--
  ****************************************************************************
  * DATATYPEs
  ****************************************************************************
  -->


  <!--
  ****************************************************************************
  * TAGs
  ****************************************************************************
  -->

  <!-- TEST:DLFIELD -->
  <tag name="test:dlfield">
    <attribute name="name"/>
    <attribute name="width" datatype="width"/>
    <attribute name="valueprop" datatype="property"/>
    <taginstance>
    </taginstance>
    <protocolitem>
    </protocolitem>
  </tag>
  <tagsubnodeextension control="itr" newsubnode="test:dlfield"/>
  <tagsubnodeextension control="tr" newsubnode="test:dlfield"/>

  <taggroupsubnodeextension group="MyControls" newsubnode="test:dlfield"/>

  </editor>
</controllibrary>
```

Defining a Macro Control

You can define a macro control as XML without any additional coding. The basic specification for a macro control defined as XML is the same as the specification for a control to which also a tag handler is applied. It consists of:

- the name of the control,
- the list of attributes, and
- the list of embedding container controls.

In addition, you define the following:

- the XML for the control,
- additional properties and methods (optional), and
- additional JavaScript libraries (optional).

With the definition of a macro control, you describe how the control is composed out of other controls. An attribute of the macro control can be referenced by enclosing the attribute name in "$" characters (for example, "$myattribute$").

With a simple macro control, no further information is required.

Additional data (additional properties, methods, libraries) is only required if you want to bind the control to a so-called “server-side representative”. The control's properties and methods are bound to a dedicated Java class. Within this class, you typically encapsulate certain tasks/functionality used by the control.

The following topics are covered below:

- XML Definition for a Simple Macro Control
- XML Definition for a Macro Control with a Server-Side Representative

XML Definition for a Simple Macro Control

You will now continue with the "test:dlfield" control from the previous example.

You will add an XML definition to the "test:dlfield" control which defines the following:

- The control consists of a row for the label and a second row for the field.
- The rows themselves are placed into a column container.
- The property `width` is delegated to the column container. The label and field have a width of 100%.
- The property `name` is delegated to the corresponding property of the label.

- The property `valueprop` is delegated to the corresponding property of the field.

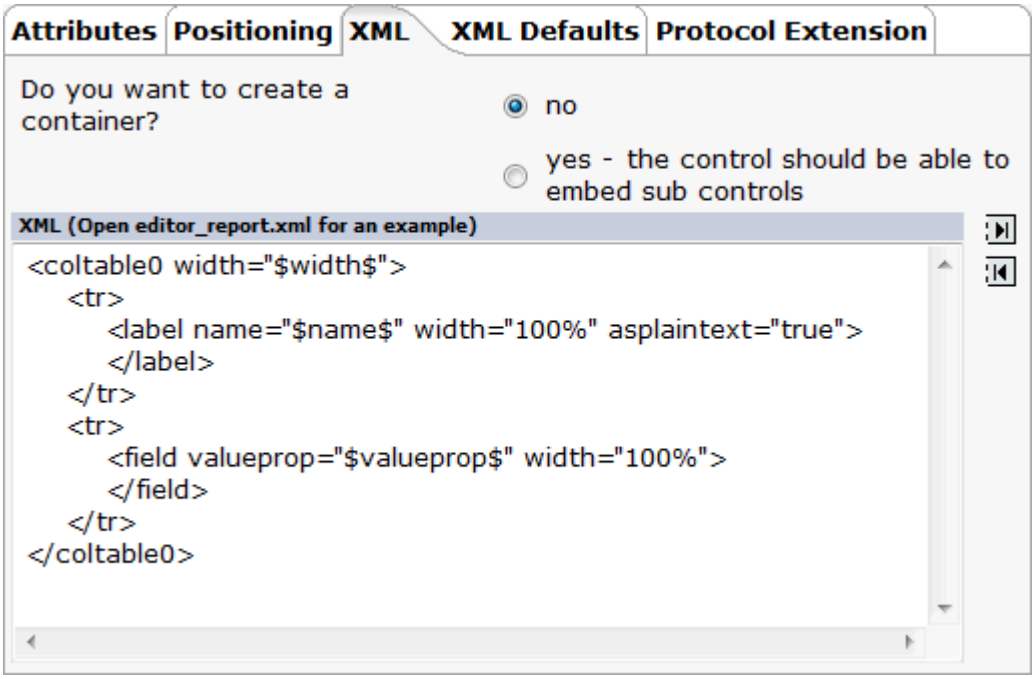
The XML for the macro control looks like this:

```
<coltable0 width="$width$">
    <tr>
        <label name="$name$" width="100%" asplaintext="true">
        </label>
    </tr>
    <tr>
        <field valueprop="$valueprop$" width="100%">
        </field>
    </tr>
</coltable0>
```

To define the XML of the macro control

1 Specify the above XML on the **XML** tab of the "test:dlfield" control.

The tab should now look as follows:



2 Save your changes.

3 Have a look at the generated XML file (*editor_test.xml*). It should now look as follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
Dynamic extension of editor.xml file.
-->

<controllibrary>
  <editor>

  <!--
  ******************************************************************************
  * DATATYPEs
  ******************************************************************************
  -->


  <!--
  ******************************************************************************
  * TAGs
  ******************************************************************************
  -->

  <!-- TEST:DLFIELD -->
  <tag name="test:dlfield">
    <attribute name="name"/>
    <attribute name="width" datatype="width"/>
    <attribute name="valueprop" datatype="property"/>
    <taginstance>
        <coltable0 width="$width$">
        <tr>
        <label name="$name$" width="100%" asplaintext="true">
        </label>
        </tr>
        <tr>
        <field valueprop="$valueprop$" width="100%">
        </field>
        </tr>
        </coltable0>
    </taginstance>
    <protocolitem>
    </protocolitem>
  </tag>
  <tagsubnodeextension control="itr" newsubnode="test:dlfield"/>
  <tagsubnodeextension control="tr" newsubnode="test:dlfield"/>

  <taggroupsubnodeextension group="MyControls" newsubnode="test:dlfield"/>

  </editor>
</controllibrary>
```

The control is now ready for use. The Layout Painter will offer the control for the containers ITR and TR.

XML Definition for a Macro Control with a Server-Side Representative

This example demonstrates how to build a macro control that refers to a server-side representative.

The server-side representative class `DLFIELDInfo` described below is specific for custom controls used within PAGE layouts. In addition, the Code Assistant support described below is only available in PAGE layouts.

For NATPAGE layouts, you can also implement server-side representative classes. These are the so-called “binding classes” (see also *Binding Concept*.

It is important that the server-side representative classes for controls in PAGE layouts are different from the binding classes of custom controls in NATPAGE layouts. For binding classes of custom controls used within NATPAGE layouts, see the examples in the Natural for Ajax demos.

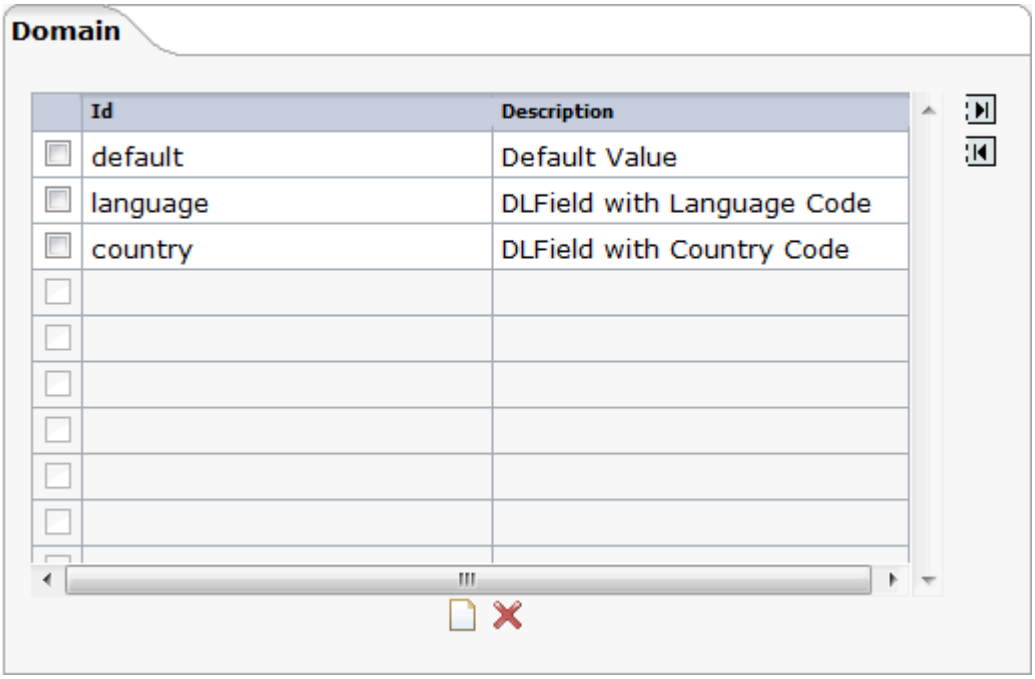
You will now extend the "test:dlfield" control from the previous example as follows:

- If the field is used to enter a country or language, it provides a value help.
- For reuse, the computation of valid values is encapsulated within a server-side representative (Java class `DLFIELDInfo`).
- The field (i.e. the properties `valueprop` and `popupprop`) is bound to properties of class `DLFIELDInfo`.

Thus, the page adapter just has to provide for a `DLFIELDInfo` object; it does not have to compute the value help by itself.

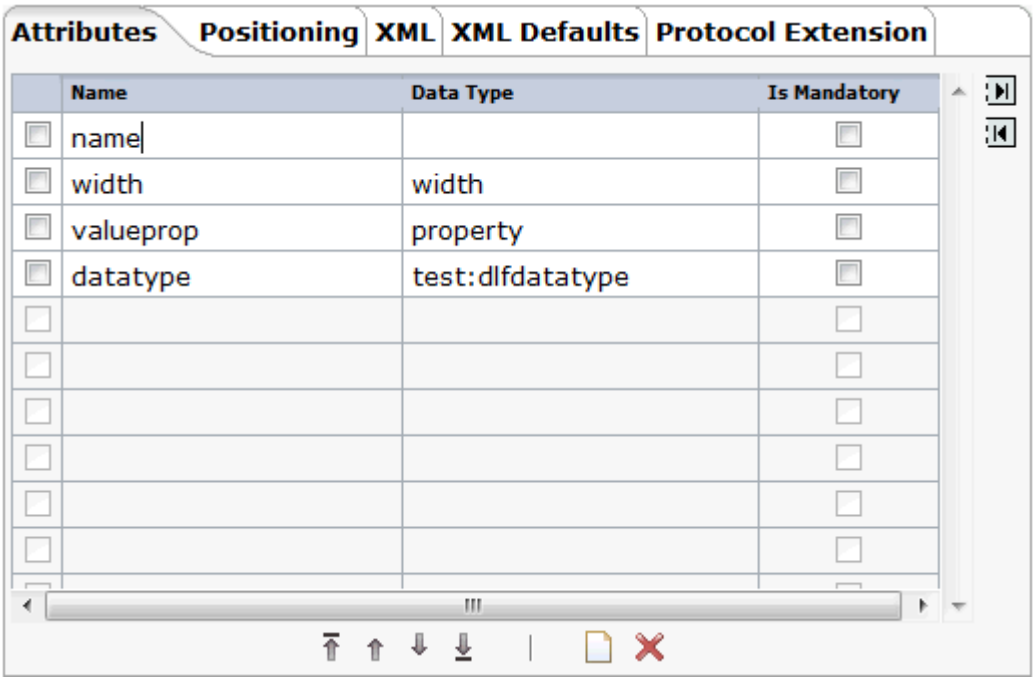
To define a macro control with a server-side representative

1. **Add a new data type** with the name "test:dlfdatatype".
2. Select the new data type in the tree.
3. Specify the following values on the **Domain** tab:



4 Select the control "test:dlfield" in the tree.

5 On the **Attributes** tab, add the attribute `datatype` and define "test:dlfdatatype" as the data type.



Until now, you have used the attribute `valueprop` to bind the field's input value to an adapter property.

Using the server-side representative concept, the adapter now provides a property (referenced by the property `valueprop`) that returns an instance of such a representative (`DLFIELDInfo`). The field value and the value help are handled within that class. The following shows the coding of the class `DLFIELDInfo`:

```
package com.softwareag.cis.test35;

import com.softwareag.cis.server.util.ValidValueLine;
import java.util.*;
import com.softwareag.cis.server.*;
import com.softwareag.cis.server.util.*;
import com.softwareag.cis.util.*;

public class DLFIELDInfo
{
 private final static String ML_APP = "release35";
 private Adapter m_adapter;

 //Constructor

 public DLFIELDInfo(Adapter adapter)
 {
  m_adapter = adapter;
 }

   // ------------------------------------------------------------------------
    // properties
   // ------------------------------------------------------------------------

    // property >fieldValue<
    String m_fieldValue;
    public String getFieldValue() { return m_fieldValue; }
    public void setFieldValue(String value) { m_fieldValue = value; }

    // property >hasPopupHelp<
    public boolean getHasPopupHelp()
    {
        return m_fieldDatatype != null &&
               m_fieldDatatype.length() != 0;
    }

    // property >fieldDatatype<
    String m_fieldDatatype;
    public String getFieldDatatype() { return m_fieldDatatype; }
    public void setFieldDatatype(String value) { m_fieldDatatype = value; }

   // ------------------------------------------------------------------------
    // methods
   // ------------------------------------------------------------------------

    /** */
```

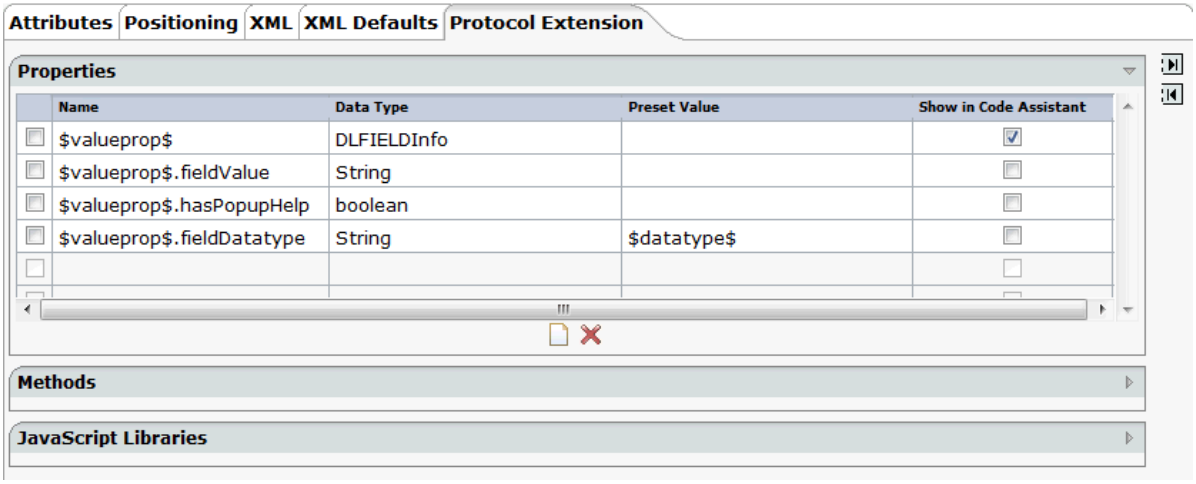
```
    public ValidValueLine[] findValidValuesForFieldValue()
    {
     String text1 = m_adapter.replaceLiteral(ML_APP, "dlfi_code_1");
     String text2 = m_adapter.replaceLiteral(ML_APP, "dlfi_code_2");
     String text3 = m_adapter.replaceLiteral(ML_APP, "dlfi_code_3");
     String text4 = m_adapter.replaceLiteral(ML_APP, "dlfi_code_4");
     String text5 = m_adapter.replaceLiteral(ML_APP, "dlfi_code_5");
     String text6 = m_adapter.replaceLiteral(ML_APP, "dlfi_code_6");
     String text7 = m_adapter.replaceLiteral(ML_APP, "dlfi_code_7");
     String text8 = m_adapter.replaceLiteral(ML_APP, "dlfi_code_8");
     String text9 = m_adapter.replaceLiteral(ML_APP, "dlfi_code_9");
     String text10 = m_adapter.replaceLiteral(ML_APP, "dlfi_code_10");

        if (m_fieldDatatype != null &&
            m_fieldDatatype.equalsIgnoreCase(text1))
        {
            return new ValidValueLine[]
            {
                  new ValidValueLine(text2, text3),
                  new ValidValueLine(text4, text5)
            };
        }
        if (m_fieldDatatype != null &&
                m_fieldDatatype.equalsIgnoreCase(text6))
        {
            return new ValidValueLine[]
            {
                  new ValidValueLine(text2, text7),
                  new ValidValueLine(text8, text9)
            };
        }
        throw new Error(text10);
    }
}
```

6 Select the **XML** tab and change the existing macro as follows:

```
<coltable0 width="$width$">
    <tr>
        <label name="$name$" width="100%" asplaintext="true">
        </label>
    </tr>
    <tr>
       <field valueprop="$valueprop$.fieldValue" popupmethod="openIdValueCombo" ↩
popupprop="$valueprop$.hasPopupHelp" width="100%">
        </field>
    </tr>
</coltable0>
```

7 Add the used properties to the **Protocol Extension** tab so that it looks as follows.



The properties `$valueprop$`, `$valueprop$.fieldValue` and `$valueprop$.hasPopupHelp` will be included in the access path of the layout.

You can specify whether a property is to be shown within the Code Assistant of the Layout Painter. Select only the properties that must be provided by the adapter itself (in this example, this is property `$valueprop$`). Do not select the properties that are provided by the control representative. This class is written once and used multiple times within your adapters.

The property `$valueprop$.fieldDatatype` is used to pass the value of the attribute `datatype` from the control to the class `DLFIELDInfo`. The value is set once when the page is loaded. With this, the `DLFIELDInfo` class determines whether pop-up help is available and computes the list of valid values for the pop-up help.

8 Save your changes.

9 Have a look at the generated XML file (*editor_test.xml*). It should now look as follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
Dynamic extension of editor.xml file.
-->

<controllibrary>
  <editor>

  <!--
  ******************************************************************************
  * DATATYPEs
  ******************************************************************************
  -->

  <!-- TEST:DLFDATATYPE -->
  <datatype name="test:dlfdatatype">
    <value id="default" name="Default Value."/>
```

```
    <value id="language" name="DLFIELD with Language Code."/>
    <value id="country" name="DLFIELD with Country Code."/>
  </datatype>


  <!--
  ****************************************************************************
  * TAGs
  ****************************************************************************
  -->

  <!-- TEST:DLFIELD -->
  <tag name="test:dlfield">
    <attribute name="name"/>
    <attribute name="width" datatype="width"/>
    <attribute name="valueprop" datatype="property"/>
    <attribute name="datatype" datatype="test:dlfdatatype"/>
    <taginstance>
        <coltable0 width="$width$">
        <tr>
        <label name="$name$" width="100%" asplaintext="true">
        </label>
        </tr>
        <tr>
       <field valueprop="$valueprop$.fieldValue" popupmethod="openIdValueCombo" ↩
popupprop="$valueprop$.hasPopupHelp" width="100%">
        </field>
        </tr>
        </coltable0>
    </taginstance>
    <protocolitem>
        <addproperty name="$valueprop$" datatype="DLFIELDInfo" ↩
useincodegenerator="true"/>
        <addproperty name="$valueprop$.fieldValue" datatype="String"/>
        <addproperty name="$valueprop$.hasPopupHelp" datatype="boolean"/>
        <addproperty name="$valueprop$.fieldDatatype" datatype="String" ↩
presetvalue="$datatype$"/>
    </protocolitem>
  </tag>
  <tagsubnodeextension control="itr" newsubnode="test:dlfield"/>
  <tagsubnodeextension control="tr" newsubnode="test:dlfield"/>

  <taggroupsubnodeextension group="MyControls" newsubnode="test:dlfield"/>

  </editor>
</controllibrary>
```

The control is now ready for use. The Layout Painter will offer the control for the containers ITR and TR.

Additional Information

If you are working with NaturalONE, see the macro control examples of the Natural for Ajax demos.

If you have also installed the standalone version of Natural for Ajax, you can find the sources of the above described DLFIELD example in the **cisdemos**:

- `<your-webapplication>`/*cisdemos/xml/35_dlfield.xml*
- `<your-webapplication>`/*cisdemos/src/com/softwareag/cis/test35/DLFIELDAdapter.java*
- `<your-webapplication>`/*cisdemos/src/com/softwareag/cis/test35/DLFIELDInfo.java*

V Runtime Tools

15 Runtime Tools

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

In NaturalONE you can access these tools via the Ajax Developer context menu on an Ajax-enabled Natural project.

The runtime tools cover the following capabilities:

- Session Configuration
- Natural Client Logging Configuration
- Ajax Configuration
- Monitoring Tool

All tools and procedures are described in the section *Natural for Ajax > Configuration and Administration > Natural for Ajax Runtime Tools* of this documentation.

VI Server Logs Viewer

16 Server Logs Viewer

- About the Server Logs Viewer 154
- Preconfigured Server Logs 154
- Advanced Usage 156

About the Server Logs Viewer

The server logs viewer is preconfigured to view the server logs of the user interface components. All user interface components of a project which has been enabled for Ajax Developer share the same server log. This is described under ***Preconfigured Server Logs***.

In addition, the server logs viewer can be used to view all kind of other log files. This is described under ***Advanced Usage***.

Preconfigured Server Logs

The **Server Logs Viewer** view is automatically opened when a project is enabled for Ajax Developer. A predefined log viewer is automatically available for this project. To display this predefined log viewer in the **Server Logs Viewer** view, you have to create an entry for it, as described below.

To create an entry for a predefined log viewer

1 In the **Project Explorer** view or in the **Natural Navigator** view, select the project which has been enabled for Ajax Developer.

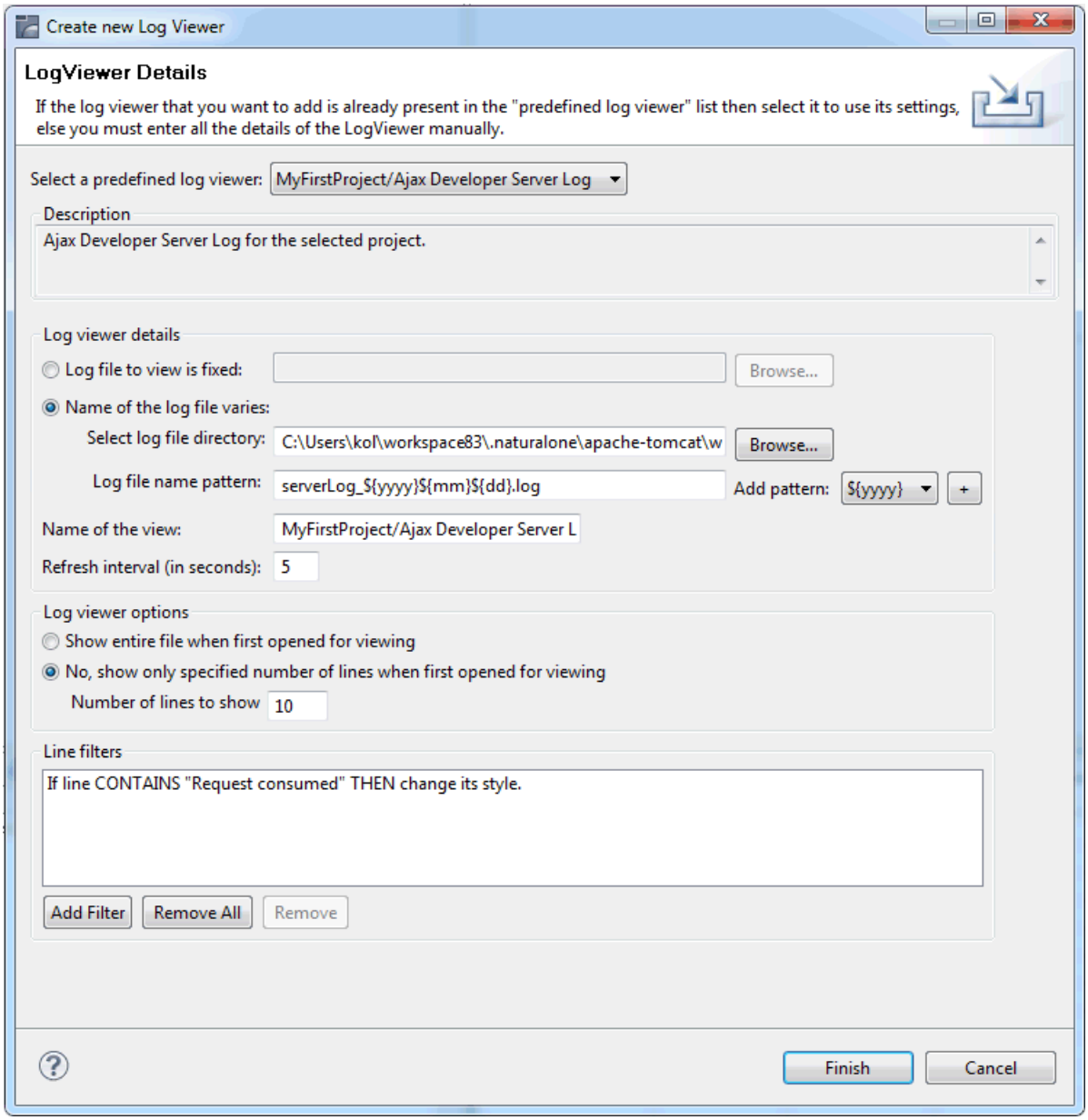
2 Go to the **Server Logs Viewer** view.

If this view is not active, you can open it as described under ***Opening the Server Logs View***.

3 Choose the following icon in the local toolbar:

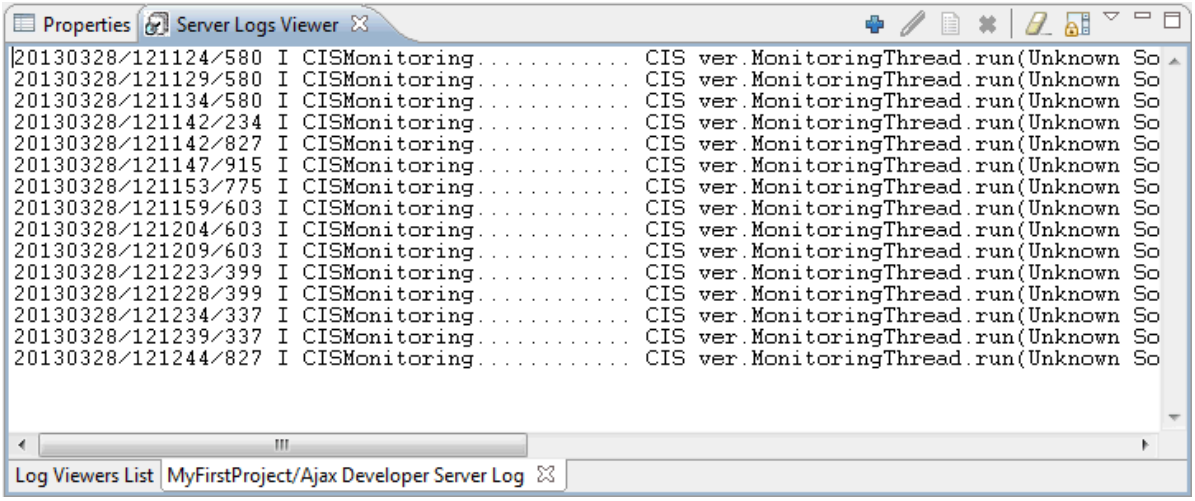
4 In the resulting dialog box, open the **Select a predefined log viewer** drop-down list box and select the predefined log viewer. The name of the predefined log viewer is `<project-name>`/*Ajax Developer Server Log*.

When you select the predefined log viewer, Ajax Developer automatically sets the correct log location for the project. A predefined filter is used. If you want, you can customize the filter settings. See ***Creating a Log Viewer*** for further information.



5 Choose the **Finish** button.

An entry for the log viewer is now shown in the **Server Logs Viewer** view. When you double-click this entry, the log file is opened. See also *Opening a Log Viewer*.



Advanced Usage

The server logs viewer can also be used to view other log files, for example, the server logs of the Tomcat server. The server logs viewer can be used to view single or multiple log files. A log file records activities and operations that occurred on a server/computer, maintaining an operational history of these activities. A log file is identified by its *.log* extension.

A log viewer allows the user to:

- Specify refresh intervals.
- Specify the number of lines to be displayed from a file.
- Customize the way to view files by specifying filter conditions.
- Save the log files added in the viewer so that the user is able access them easily the next time.
- Provide extension points so as to enable other Eclipse plug-ins to contribute their log files for viewing.

The following topics are covered below:

- Opening the Server Logs Viewer
- Creating a Log Viewer
- Editing a Log Viewer
- Opening a Log Viewer

- Removing a Log Viewer

Opening the Server Logs Viewer

The **Server Logs Viewer** view is automatically opened when a project is enabled for Ajax Developer.

If this view is not active, you can open it as described below.

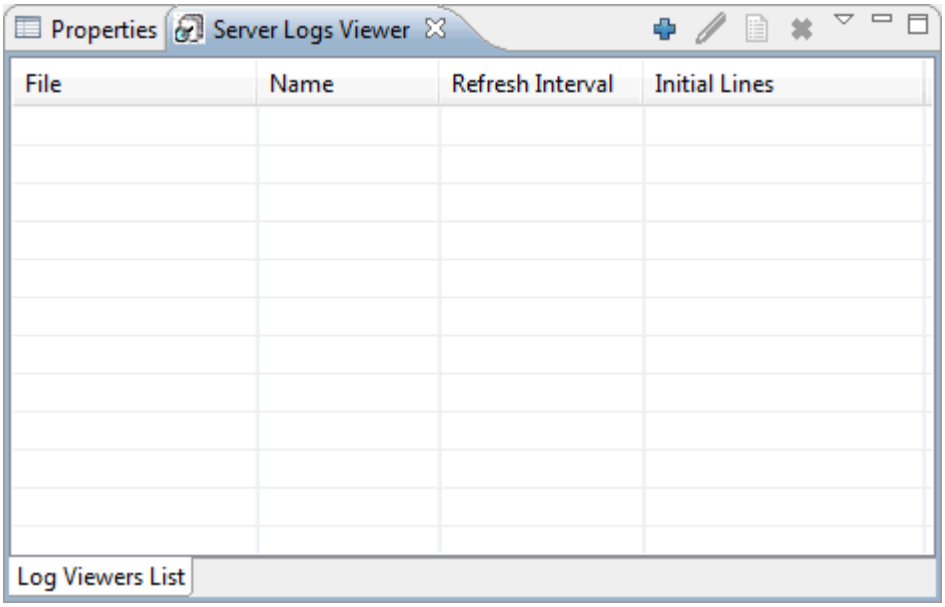
To open the server logs viewer

1 From the **Window** menu, choose **Show View > Other**.

2 In the resulting **Show View** dialog box, expand the **Software AG NaturalONE** node and select **Server Logs Viewer**.

3 Choose the **OK** button.

The following view appears.



Creating a Log Viewer

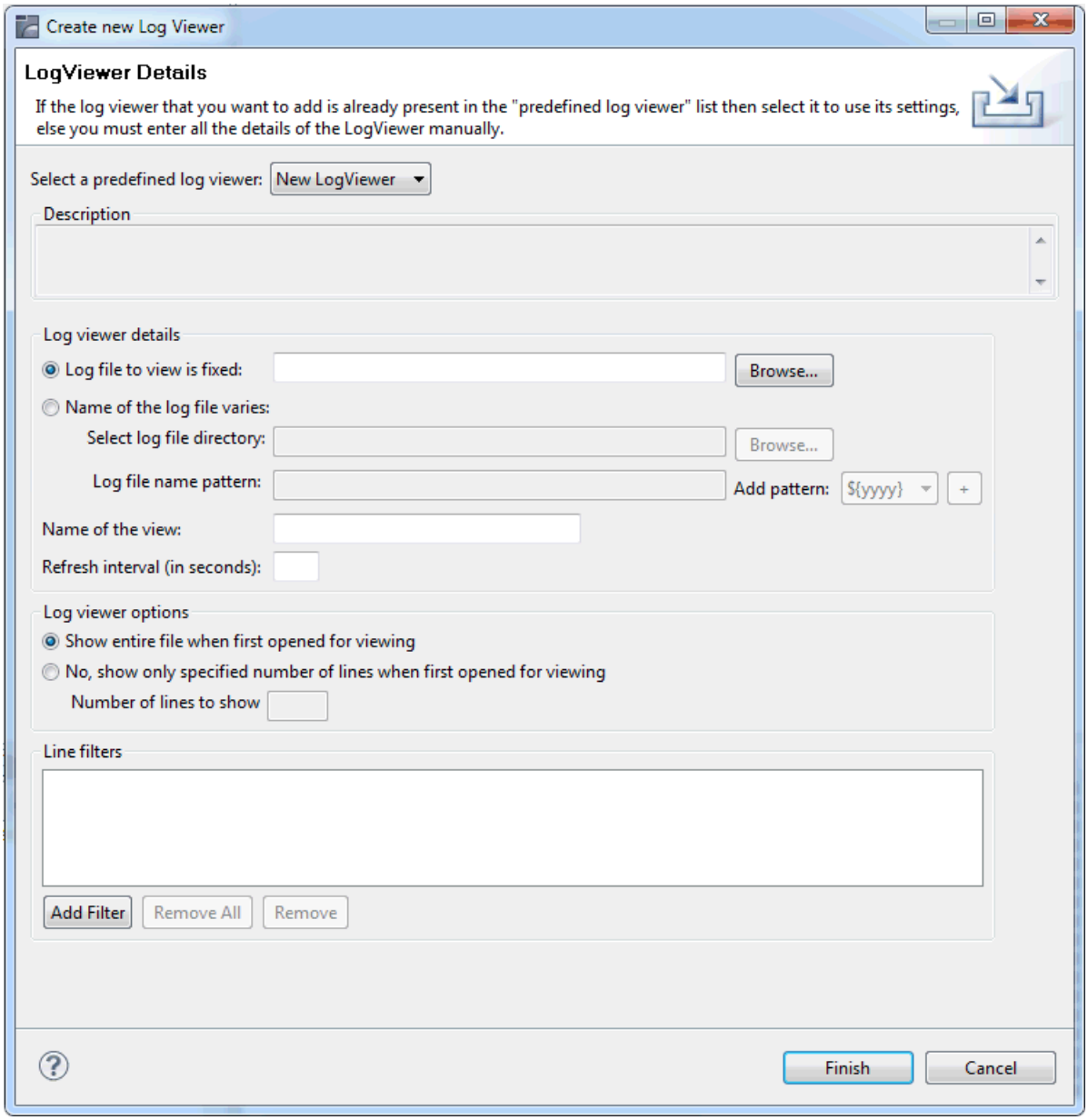
You can create an entry for a new log viewer by using the settings of an existing log viewer (for example, of the predefined log viewer) and then modifying the settings according to your requirements.

To create an entry for a new log viewer

1 Go to the **Server Logs Viewer** view.

2 Choose the following icon in the local toolbar:

The following dialog box appears.



3 Specify the following information:

Select a predefined log viewer

To create a new log viewer, keep the default value **New Log Viewer**.

If you want to use an existing log viewer as the basis for the new log viewer, select it from this drop-down list box. The description for the selected log viewer is then shown in the **Description** text box.

Log file to view is fixed

Select this option if the log file that must be viewed is fixed, that is, if its name does not change. The log viewer will view/read this file every time. Use the **Browse** button to select the log file.

Name of the log file varies

Select this option if the name of the log file changes on some criteria. For example, if the name of the log file contains a timestamp which is changed daily, the name of the log file also varies daily. Specify the following information in this case:

Option	Description
Select log file directory	Use the **Browse** button to select the directory containing the log files. For example, *<myworkspace>\.naturalone\apache-tomcat\logs* for the log files of your local Apache Tomcat.
Log file name pattern	Specify the date pattern of your log files. For example, *catalina.${yyyy}-${mm}-${dd}.log* for the log files of your local Apache Tomcat. **Note:** Ensure that you know the log file naming conventions used in your application before specifying this information.
Add pattern	Select the date format from the available options in the drop-down list box. Choose the button containing the plus sign to add each pattern to the **Log file name pattern** text box.

Name of the view

Enter a name to identify this viewer.

Refresh interval (in seconds)

The log viewer refreshes the view as per the interval you specify here.

Log viewer options

Choose the option to either view the entire log file or only a specified number lines when first opened for viewing. If you select the latter option, you need to specify the number of lines to be displayed.

Line filters

Choose the **Add Filter** button if you wish to add filters to the log files you view. Using these features, you can filter lines based on some conditions such as skipping the entire line or changing the display style in the view.

You add the line filters in the resulting dialog box. For example, to view a line highlighted in black and the font color as green, specify the following:

1. From the **If line** drop-down list box, select **Contains text** and enter "Info" in the text box.
2. Select the **Change the style of the line** option button.
3. Choose the **Foreground color** button and select the color green.

4. Choose the **Background color** button and select the color black.
5. Choose the **OK** button.

   The following information is then shown:

   ```
   If line CONTAINS "Info" THEN change its style.
   ```

To remove the selected line filter or all line filters, choose either the **Remove** or **Remove All** button.

4 Choose the **Finish** button.

Editing a Log Viewer

When you edit a log viewer, the same information is shown as when adding a new log viewer.

To edit a log viewer

1 In the **Server Logs Viewer** view, select the log viewer that you want to edit.

2 Invoke the context menu and choose **Edit Log Viewer**.

Or:

Choose the following icon in the local toolbar:

A dialog appears.

3 Modify all required information as described in *Creating a Log Viewer*.

In edit mode, it is not possible to change the log file to be viewed. The corresponding options are disabled.

4 Choose the **OK** button.

Opening a Log Viewer

When you open a log viewer, it is shown on a separate tab of the **Server Logs Viewer** view. It shows the contents of the defined log file.

To open one or more log viewers

1 In the **Server Logs Viewer** view, select the log viewers that you want to open.

2 Invoke the context menu and choose **Open Log Viewer(s)**.

Or:

Choose the following icon in the local toolbar:

Or:

If you want to to open a single log viewer, you can simply double-click it.

A separate tab for each selected log viewer is provided in the **Server Logs Viewer** view.

If you want to remove such a tab, choose the close button that is shown next to the tab name.

Removing a Log Viewer

When you remove a log viewer, it is no longer shown in the **Server Logs Viewer** view.

To remove one or more log viewers

1 In the **Server Logs Viewer** view, select the log viewers that you want to remove.

2 Invoke the context menu and choose **Remove Log Viewer(s)**.

Or:

Choose the following icon in the local toolbar:

A dialog appears, asking whether you really want to remove the selected log viewers.

3 Choose the **OK** button.

VII Bootstrap Icons

17 Bootstrap Icons

The Bootstrap Icon library is included in Natural for Ajax and offers over 2,000 high quality icons without requiring the user to download anything. Access the Bootstrap Icon library in your internet browser through this **link**.

To use an icon for your program:

1. Open the Bootstrap Icons library.
2. Search the icon by name in the filter field.
3. Copy the icon font html code to your clipboard.
4. Paste the html code as a property value into your layout xml or as a field value into your Natural program.