Natural for Ajax

Natural Pages Development

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: ONE-NATNJX-DEVELOPMENT-934-20261004

Table of Contents

Natural Pages Development v
1 About this Documentation 1
Document Conventions 2
Online Information and Support 2
Data Protection 3
2 Developing the User Interface 5
Enabling a Natural Project for Ajax Developer 6
Creating a User Interface Component 6
Creating a Natural Page 7
Specifying Properties for the Natural Page 8
Designing the Page 8
Binding Properties and Methods 9
Previewing the Layout 9
Viewing the Protocol 9
Saving the Layout 9
Generating the Adapter 10
Data Type Mapping 10
Configuration of Page Layout Errors/Warnings 11
3 Developing the Application Code 15
Generating the Main Program 16
Switching between the Natural Program and the Layout 17
Executing the Main Program 19
Debugging the Main Program 20
Structure of the Main Program 20
Handling Page Events 21
Built-in Events and User-defined Events 21
Sending Events to the User Interface 22
Using Pop-Up Windows 23
Debugging Modal Pop-up Windows 24
Using Natural Maps 25
Navigating between Pages and Maps 25
Using Pages and Maps Alternatively 26
4 Developing Customized Logon and Disconnect Pages 29
General Information 30
Adding the NatLogon and/or NatDisconnect Page Layout to a NaturalONE Project 30
What Can be Customized? 31
What Cannot be Customized? 32
Advanced Customizations 32
Deployment 33
5 Styling the User Interface 35
6 Deploying the Application 37
General Information 38

Content of a Natural for Ajax Web Application 38
Content of the Sample webconfig Directory 39
Using the Deployment Wizard for Web Applications 42
Starting the Deployment from Eclipse 50
Starting the Deployment from the Command Line 51
Status Code Handling 52
Deploying the Web Application Archive (.war) 52
7 Natural Parameters and System Variables 53
8 Usage of Edit Masks 55
General Information 56
Data Types with Edit Masks 56
Natural Profile Parameters 58
Specifying Edit Masks in Layouts 58
Edit Masks at Runtime 59
9 Multi-Language Management in Ajax 61
10 Support of Right-to-Left Languages 63
11 Server-Side Scrolling and Sorting 65
General Information 66
Variants of Server-Side Scrolling and Sorting 66
Controls that Support Server-Side Scrolling and Sorting 70
Data Structures for Server-Side Scrolling and Sorting 70
Server-Side Scrolling and Sorting in Trees 72
Events for Server-Side Scrolling and Sorting 73
12 Accessibility 75
Accessibility Non Responsive Pages 76
Accessibility Responsive Pages 76
13 Code Pages 77
14 Test Automation of Natural for Ajax Applications 79
General Information 80
Enabling the Applications for Test Automation 80
Advanced testtoolid Settings in Complex Controls 83

Natural Pages Development

Developing the User Interface	How to develop the user interface.
Developing the Application Code	How to develop the application code.
Developing Customized Logon and Disconnect Pages	How to add the NatLogon and NatDisconnect pages to a NaturalONE project, and how to customize these pages.
Styling the User Interface	How to develop and apply application-specific style settings.
Deploying the Application	How to deploy a rich GUI application from a version control system to a WAR file.
Natural Parameters and System Variables	Gives an overview of the Natural parameters and system variables that are evaluated in Natural for Ajax applications and sent to Application Designer.
Usage of Edit Masks	Describes how Natural for Ajax supports the Natural edit mask concept.
Multi Language Management in Ajax	Describes aspects to be considered for internationalization.
Support of Right-to-Left Languages	Describes how Natural for Ajax supports right-to-left languages and bidirectional text.
Server-Side Scrolling and Sorting	Describes how Natural for Ajax supports the concept of server-side scrolling and sorting.
Accessibility	Describes how Natural for Ajax supports features pertaining to accessibility.
Code Pages	
Test Automation of Natural for Ajax Applications	

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.

2 Developing the User Interface

- Enabling a Natural Project for Ajax Developer 6
- Creating a User Interface Component 6
- Creating a Natural Page 7
- Specifying Properties for the Natural Page 8
- Designing the Page 8
- Binding Properties and Methods 9
- Previewing the Layout 9
- Viewing the Protocol 9
- Saving the Layout 9
- Generating the Adapter 10
- Data Type Mapping 10
- Configuration of Page Layout Errors/Warnings 11

In the *First Steps* tutorial, you have developed a small rich internet program step by step. In this tutorial, you have already performed most of the steps required to develop a rich internet application.

The general procedure to develop a rich internet application with Natural for Ajax is as follows:

1. Use Ajax Developer to design the web pages that form the user interface of your application.
2. Generate a Natural adapter for each page (by saving the page). The adapter is a Natural object that forms the interface between the application code and the web page.
3. Write the Natural application programs that contain the business logic and use adapters to exchange data with the web pages.

In this chapter, the first two steps (design and adapter) are explained in more detail. Step 3 (business logic) is described in the section ***Developing the Application Code*** which also addresses advanced topics that are not covered in the tutorial.

Enabling a Natural Project for Ajax Developer

Before you can define layout pages for a Natural project, you have to enable the project for Ajax Developer. For detailed information, see *Enabling a Project for Ajax Developer* in the *Ajax Developer* documentation.

If you skip this step, you are asked to enable the project when you create the first user interface component in the project.

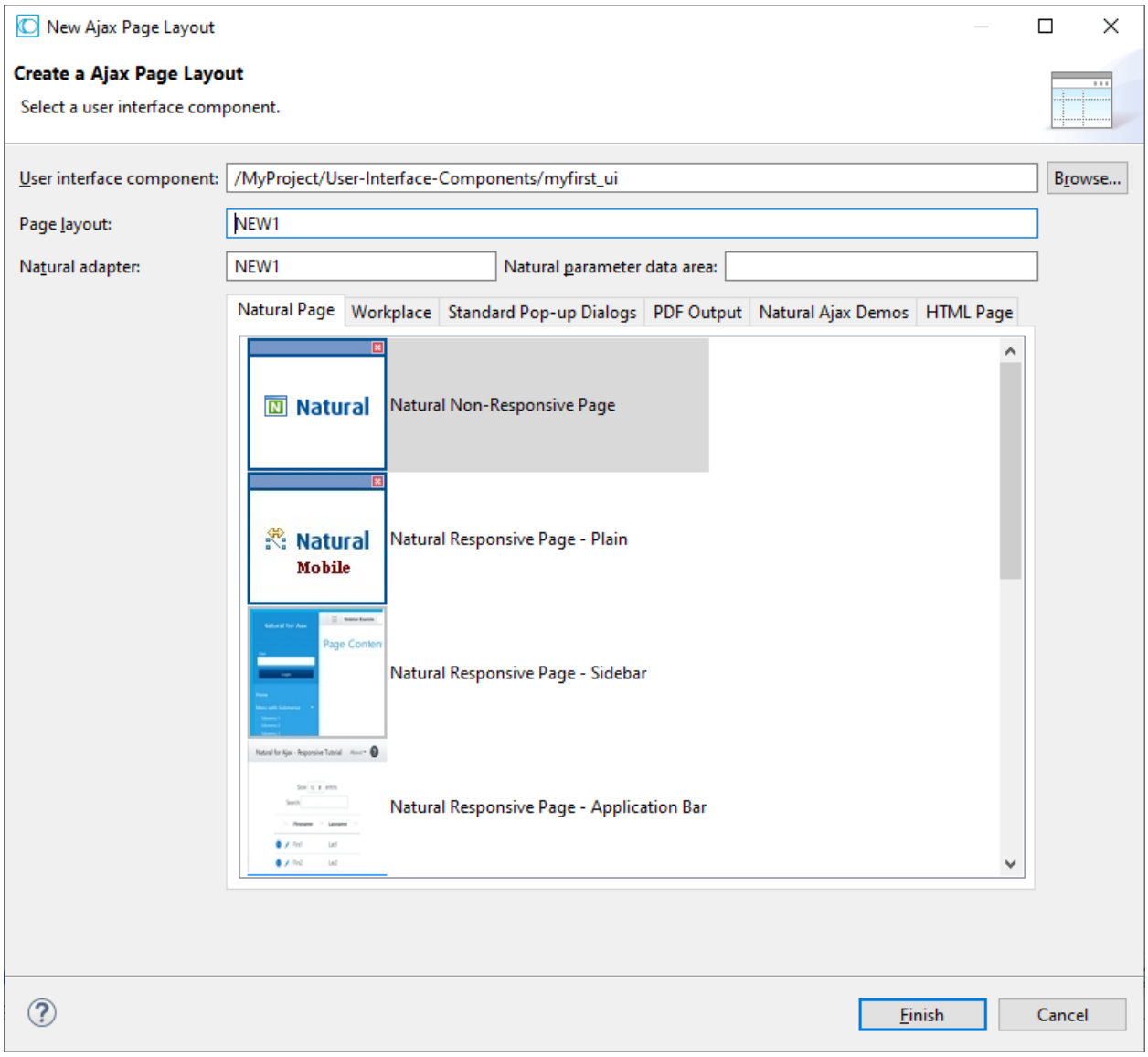
Creating a User Interface Component

After the project has been enabled for Ajax Developer, you have to define the folder that is to contain the Natural adapters that will be generated from your page layouts. This is usually the *SRC* folder of a Natural library. For detailed information, see *Creating User Interface Components* in the *Ajax Developer* documentation.

Creating a Natural Page

In order to create the layout of your web pages, you use Ajax Developer's Layout Painter.

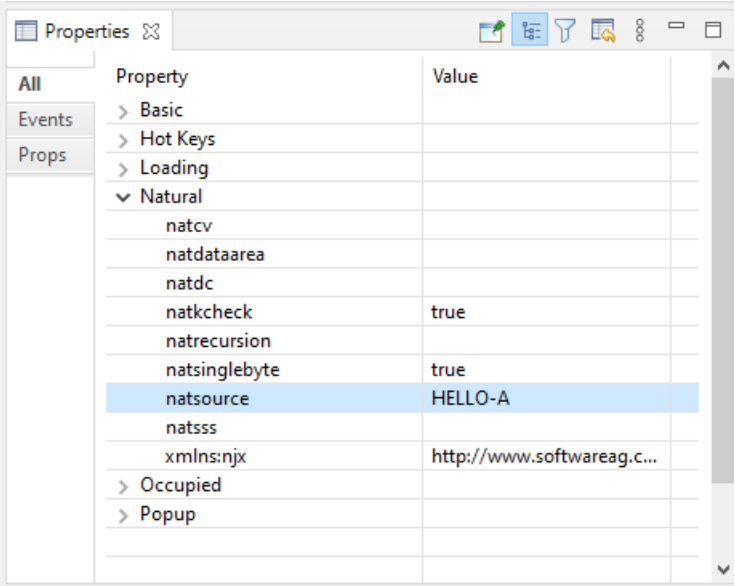
Add a page layout to your project as described in *Creating a Layout* in the *Ajax Developer* documentation (select the template for the Natural page).



Specifying Properties for the Natural Page

In order to specify generation options for the new page, you specify values for certain properties that are specific for Natural pages.

To define properties, you select the node **natpage** in the layout tree of the Layout Painter. The properties for this control are then shown in the **Properties** view. All Natural-specific properties are provided in the **Natural** node.



For information on the properties that are available for a Natural page, see NATPAGE.

Designing the Page

Design your Natural page by dragging controls and containers from the controls palette onto the corresponding node in the layout tree or to the preview area. This has already been explained in the section *Writing the GUI Layout* of the tutorial.

Note: For more detailed information on defining the layout, see *Defining the XML Layout* in the *Ajax Developer* documentation.

Binding Properties and Methods

Many of the controls you use on your page have properties that can be controlled by the application. Also the controls can raise events that your application may wish to handle. The next step is therefore assigning identifiers to each of these properties and events under which your application can later address them. This procedure is called “binding”.

To get an overview which properties and events are bindable to application variables and events, select a control in the layout tree and check the properties in the **Properties** view.

The usage and meaning of the properties and events is described for each control in *Natural Pages Development*,

Previewing the Layout

To find out how the current layout definitions are rendered on the page, preview the layout as described in *Previewing the Layout* of the *Ajax Developer* documentation.

Viewing the Protocol

The protocol contains warnings and error messages that might occur while you design and preview your page. For further information, see *Viewing the Layout Painter Protocol* in the *Ajax Developer* documentation.

Saving the Layout

Save the page layout as described in *Saving the Layout* in the *Ajax Developer* documentation. A Natural adapter is then generated into the folder that you specified when creating a user interface component. The adapter is updated each time you save the layout.

Generating the Adapter

When you save the layout, a Natural adapter is generated according to the following rules:

Location	The adapter is generated into the folder that you specified when creating a user interface component. See *Creating User Interface Components* in the *Ajax Developer* documentation.
Name	The name of the adapter is determined by the name you specified when creating a new layout. See *Creating a Layout* in the *Ajax Developer* documentation.
Property identifiers	For each control property that has been bound to an identifier (as described in ***Binding Properties and Methods***) a parameter in the parameter data area of the adapter is generated. The identifier is therefore validated against the Natural naming conventions for user-defined variables and translated to upper-case. If an identifier does not comply to these rules, a warning is generated into the protocol and as a comment into the adapter code. Additionally, the name must comply to the naming conventions for XML entities. This means especially that the name must start with a character. To achieve uniqueness within 32 characters, the last four characters are (if necessary) replaced by an underscore, followed by a three-digit number.
Event identifiers	For each event that can be raised by a control on the page, an event handler skeleton is generated as a comment into the adapter. **Caution:** Some controls raise events whose names are dynamically constructed at runtime. For these events, no handler skeleton can be generated. The control reference contains information about these additional events. The event identifiers are not validated.

When you specify a value for the property natdataarea, then also a Natural Parameter Data Area (PDA) is generated.

Data Type Mapping

Several Application Designer controls have properties for which a data type can be specified. An example is the FIELD control. It has a `valueprop` property which can be restricted to a certain data type. The data type is used at runtime to validate user input. At generation time (that is, when a Natural adapter is generated for the page), the data type determines the Natural data format of the corresponding adapter parameter.

The following table lists the data types used in Application Designer and the corresponding Natural data formats.

Application Designer	Natural
color	A or U (depending on the NATPAGE property natsinglebyte). The string must contain an RGB value, for instance "#FF0000" for the color red.
date	D (YYYYMMDD)
float	F4
int	I4
long	P19
time	T (HHIISS)
timestamp	T (YYYYMMDDHHIISST)
N *n.n*	N*n.n*
P *n.n*	P*n.n*
string (default)	A or U dynamic (depending on the NATPAGE property natsinglebyte).
string *n*	A*n* or U*n* (depending on the NATPAGE property natsinglebyte).
xs:double	F8
xs:byte	I1
xs:short	I2

Configuration of Page Layout Errors/Warnings

The layout protocol contains the information whether a page layout contains errors or warnings. What is considered as error or warning can be configured. An invalid XML file or an XML file from which valid HTML cannot be generated is always treated as an error.

Depending on the problem it often depends on the browser or even the browser version whether the rendering is still as intended or not. The following default settings are a recommendation. In case these settings are lowered, it may happen that in some browser versions the rendering is not as intended.

To provide a better overview, the recommended settings are grouped by the type of problem that is protocoled. The problems that can be configured are described below.

- Problem Group: Valid Value of Properties
- Problem Group: Data and Event Binding
- Problem Group: Height and Width Properties
- Problem Group: Obligatory and Recommended Properties and Controls
- Problem Group: Container - Control Hierarchy
- Problem Group: Property Combination
- Problem Group: Deprecated Controls and Properties
- Problem Group: FOP Layout Definitions

- Problem Group: Runtime Behavior

Problem Group: Valid Value of Properties

Setting	Explanation
Invalid or missing integer value	The specified value is not a valid integer value.
Invalid edit mask value	The specified edit mask is invalid.
Invalid usage of timestamp type	Datatype timestamp is not supported for all property combinations.
Invalid comma separated list	The property value must be a valid comma separated list.
Invalid hot key value	The specified hotkey is not valid.
Invalid property value	The specified value is not a valid value for this property.

Problem Group: Data and Event Binding

Setting	Explanation	Sample
Missing obligatory data or event property	A valueprop, method or other mandatory data property is missing. The impact is that no Natural fields are generated in the adapter.	VALUEPROP in FIELD control missing.
Missing recommended data property	A recommended data property is missing. This can have major impacts on the control at runtime.	-/-
Possible unintended usage of default event	When a method is not specified, in many cases a default event will be triggered at runtime. It might be the intended event or not.	FLUSHMETHOD in FIELD control missing.

Problem Group: Height and Width Properties

Setting	Explanation
Missing HEIGHT, WIDTH or LENGTH properties	HEIGHT, WIDTH or LENGTH property is obligatory for proper rendering.
Missing recommended HEIGHT, WIDTH or LENGTH properties	HEIGHT, WIDTH or LENGTH property is recommended for proper rendering. Missing values might have impacts on the rendering.
Missing ROWCOUNT in GRIDS	ROWCOUNT is missing in a grid control. This usually has major impacts on the sizing of the grids.
Recommended TAKEFULLWIDTH or TAKEFULLHEIGHT missing	Depending on specific property combinations or nesting of controls, the specification of TAKEFULLWIDTH or TAKEFULLHEIGHT is recommended.

Problem Group: Obligatory and Recommended Properties and Controls

Setting	Explanation	Sample
Missing obligatory property	An obligatory property is missing.	HELPICON control: property HELPID missing.
Missing recommended control definition	Sometimes the proper rendering of a control requires the specification of another control.	HSPLIT control: 2 SPLITCELL controls required.
Missing recommended property	A recommended property is missing. This might have major rendering impacts.	COLAREA: no NAME, TEXTID or VALUEPROP specified.

Problem Group: Container - Control Hierarchy

Setting	Explanation
Invalid sub node	A control or container has been specified as sub node of another control or controller. But this hierarchy is not supported. This problem can only happen if you are not using Layout Painter as editor.

Problem Group: Property Combination

Setting	Explanation	Sample
Duplicated definition - design time and runtime	Some properties are supported as design time properties and as runtime properties. But specifying the same property as runtime and as design time property leads to undefined rendering.	AREA controls: NAME and TEXTID are set.
Invalid property combination	The specified combination of properties is not supported.	FIELD control: POPUPPROP is set, but POPUPMETHOD is not set.
Incomplete property combination	Sometimes a property is only supported if also other properties are specified: Supply either all or none.	TEXTGRID* controls: WITHGRIDCOLHEADER is specified, but PROPREFSPROP is not.

Problem Group: Deprecated Controls and Properties

Setting	Explanation	Sample
Deprecated properties	A deprecated property has been specified. One impact might be that it is simply ignored in the currently supported browsers.	PAGEHEIGHTMINUS in ROWTABSUBPAGES.
Deprecated controls	A deprecated control has been specified. One impact might be that the corresponding HTML is not supported	ACTIVEX control.

Setting	Explanation	Sample
	in all browsers or no longer supported in the current browsers at all.	

Problem Group: FOP Layout Definitions

Setting	Explanation
Missing obligatory properties	An obligatory property in the FOP controls is missing. May have impacts on the *.pdf generation.
Invalid property value	Invalid property value in FOP controls.

Problem Group: Runtime Behavior

Setting	Explanation	Example
Possibly reduced performance	All single controls may be specified correctly and still the layout might cause performance issues.	Layouts too big.
Invalid TABINDEX values	Specifying invalid tabindex values confuses the browser and leads to unexpected behavior at runtime.	`TABINDEX=-10`.

In NaturalONE you can customize the settings in the *Ajax Developer Preferences* of your workspace.

Important: Export your settings and commit them in your version control system together with the other workspace settings. When creating a new workspace, import your settings. When upgrading your workspace to a new Natural for Ajax runtime version, your settings will be taken over automatically.

3 Developing the Application Code

- Generating the Main Program 16
- Switching between the Natural Program and the Layout 17
- Executing the Main Program 19
- Debugging the Main Program 20
- Structure of the Main Program 20
- Handling Page Events 21
- Built-in Events and User-defined Events 21
- Sending Events to the User Interface 22
- Using Pop-Up Windows 23
- Debugging Modal Pop-up Windows 24
- Using Natural Maps 25
- Navigating between Pages and Maps 25
- Using Pages and Maps Alternatively 26

Generating the Main Program

You can generate the main program using the information provided in the adapter. This main program will call the adapter to display the page and handles the events that the user raises on the page.

The resulting program can be executed in a browser where it displays the page. However, it does not yet do anything useful, because it handles the incoming events only in a default way and contains no real application logic.

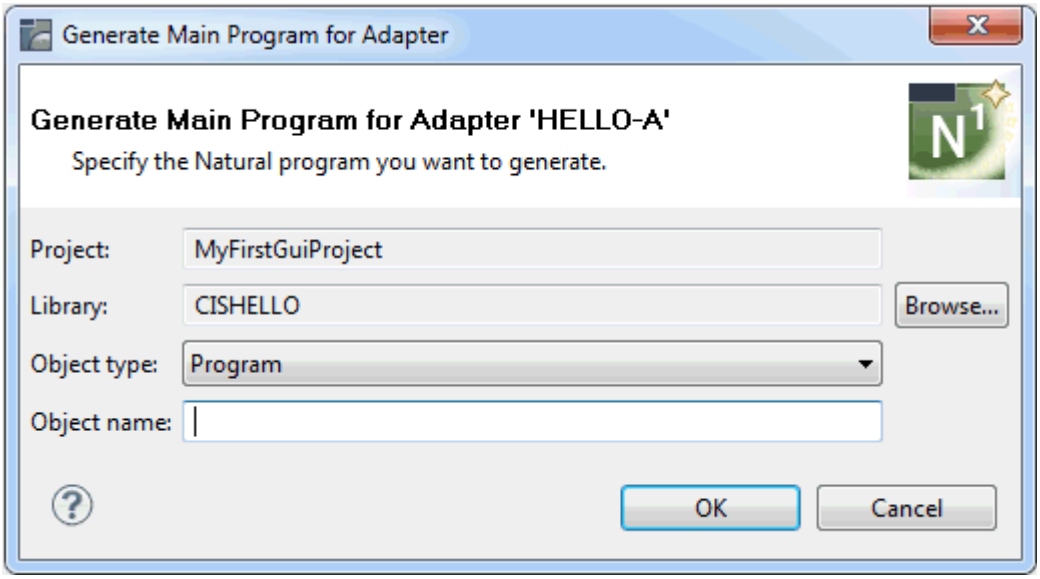
To generate the main program

1 Open the NaturalONE perspective.

2 In the **Project Explorer** view or in the **Natural Navigator** view, select the adapter for which you want to generate the main program.

3 Invoke the context menu and choose **Generate Main Program**.

The following dialog box appears.



4 Use the **Object type** drop-down list box to specify whether you want to generate a program or a subprogram.

5 In the **Object name** text box, enter the name for the main program.

6 Choose the **OK** button.

The main program with the specified name is now created.

7 Handle the events that can occur on the page.

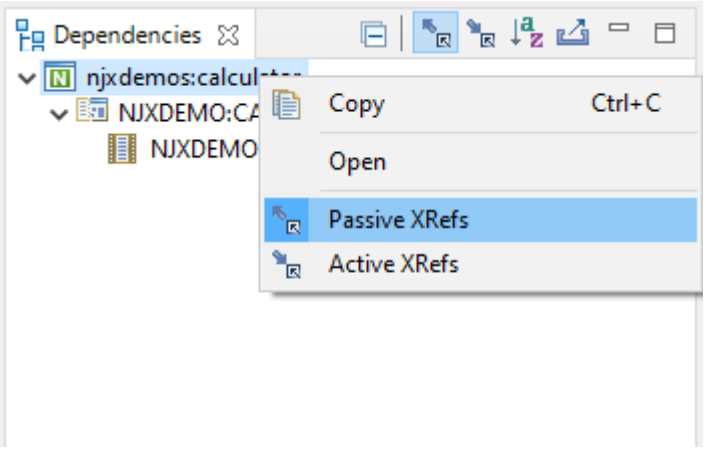
8 Save your changes and use the **Build Natural Project** command to update the Natural server and catalog the sources of the current project.

Switching between the Natural Program and the Layout

You can easily switch back and forth between your Natural program in Natural Editor and its corresponding Layout in the Layout Painter Editor.

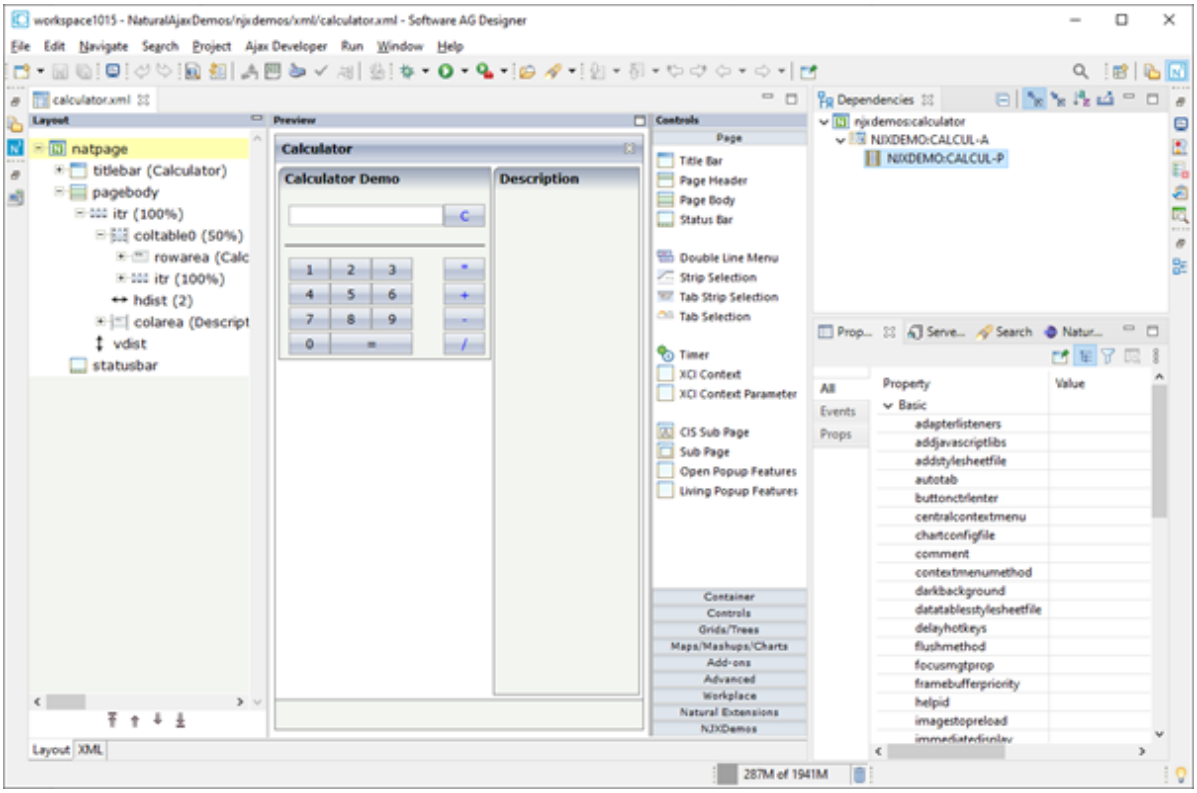
To switch from the Layout to the Natural program

1 Select "Passive XRefs" via the Dependencies View:



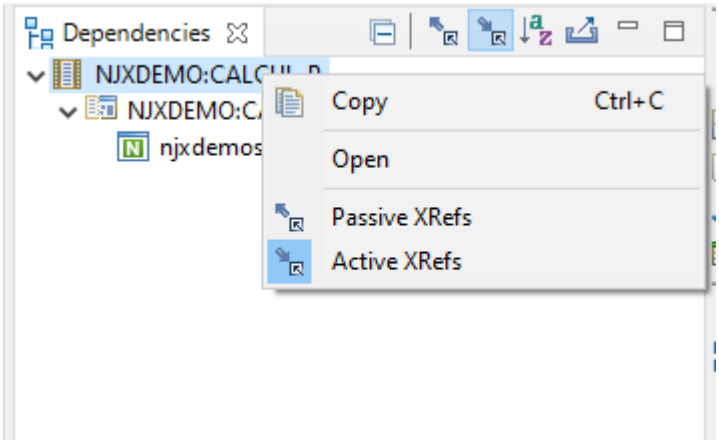
2 Double-click your program.

This will open your program ("CALCUL-P" in the example below) in the Natural Editor:



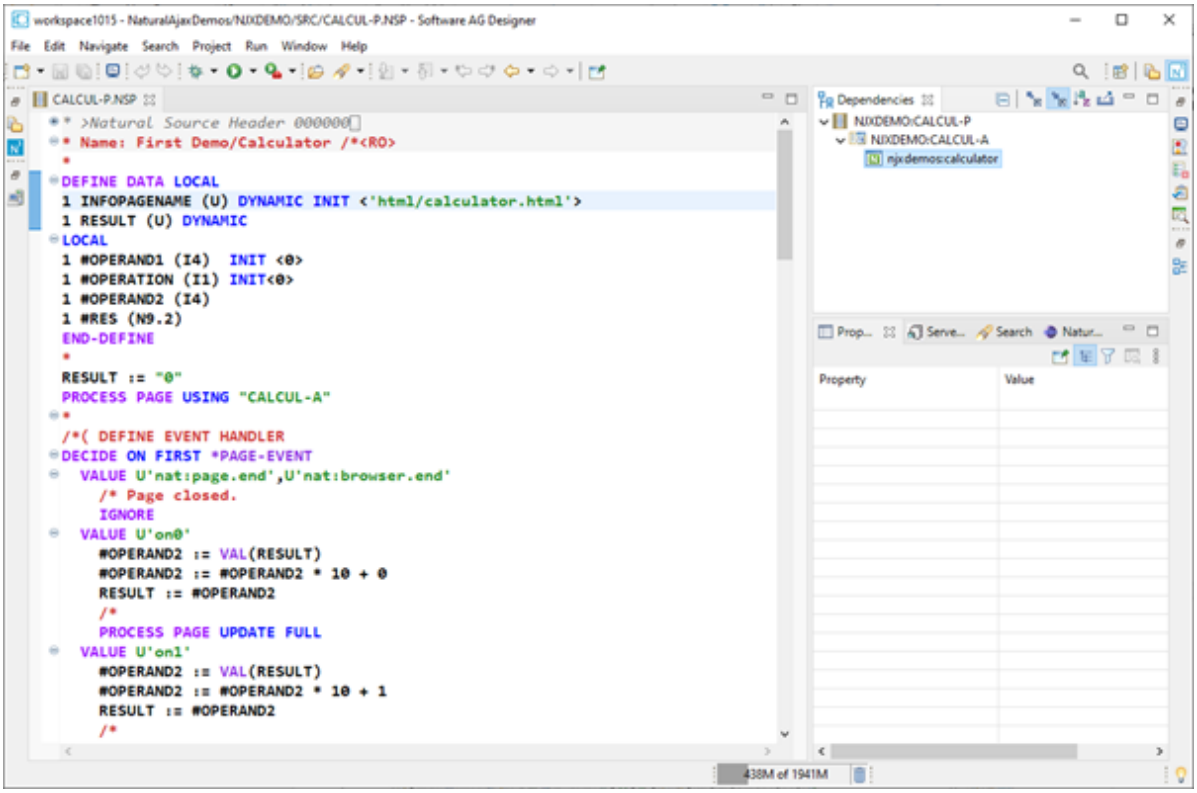
To switch from the Natural Program to the Layout

1 Select "Active XRefs":



2 Double-click your Layout.

This will open your Layout ("calculator" in the example below) in the Layout Painter Editor:



Executing the Main Program

If you have generated a program, you can execute and test it immediately as described below. If you have generated a subprogram, you can test it using the **Test Subprogram** command which is part of the Business Services functionality.

To execute the main program

1 Make sure that the NaturalONE perspective is active.

2 In the **Project Explorer** view or in the **Natural Navigator** view, select the main program.

3 Invoke the context menu and choose **Run As > Natural Application**.

See also *Executing Objects* in *Using NaturalONE*.

Debugging the Main Program

To see how the program works in detail, you can start it in the debugger.

To debug the main program

1 Make sure that the NaturalONE perspective is active.
2 In the **Project Explorer** view or in the **Natural Navigator** view, select the main program.
3 Invoke the context menu and choose **Debug As > Natural Application**.

See also *Debugging Natural Applications* in *Using NaturalONE*.

Structure of the Main Program

The main program that displays the page and handles its events has the following general structure:

- A `PROCESS PAGE USING` statement with the page adapter. The `PROCESS PAGE` statement displays the page in the user's web browser and fills it with data. Then, it waits for the user to modify the data and to raise an event.
- A `DECIDE` block with a `VALUE` clause for each event that shall be explicitly handled.
- A default event handler for all events that shall not be explicitly handled.

Each event handler does the following:

- It processes the data the has been returned from the page in the user's web browser.
- It performs a `PROCESS PAGE UPDATE FULL` statement to re-execute the previous `PROCESS PAGE USING` statement with the modified data and to wait for the next event.

The default event handler does not modify the data. It does the following:

- It performs a `PROCESS PAGE UPDATE` statement to re-execute the previous `PROCESS PAGE USING` statement and to wait for the next event.

Handling Page Events

When the `PROCESS PAGE` statement receives an event, the data structure that was passed to the adapter is filled with the modified data from the page and the system variable `*PAGE-EVENT` is filled with the name of the event. Now, the corresponding `VALUE` clause in the `DECIDE` statement is met and the code in the clause is executed.

The application handles the event by processing and modifying the data and resending it to the page with a `PROCESS PAGE UPDATE FULL` statement. Alternatively, it uses the `PROCESS PAGE UPDATE` statement without the `FULL` clause in order to resend the original (not modified) data.

Built-in Events and User-defined Events

There are built-in events and user-defined events.

Built-in Events

The following built-in events can be received:

nat:browser.end

This is event is raised whenever the session is terminated by a browser action:

- Closing of the browser.
- Navigation to another page in the browser.
- Programmatic close in a workplace (for example, close all session functions).

In addition, this event is raised in the following cases:

- Timeout of the session.
- Removal of the session with the monitoring tool.

After the event is raised, the Natural session terminates.

nat:page.end

This event is raised when the user closes the page with the Close button in the upper right corner of the page.

nat:popup.end

This event can be raised when the user closes the pop-up window with the Close button in the upper right corner of the pop-up window. To activate this event for the current pop-up window, the property `popupendmethod` of the NATPAGE control has to be set to "true". The default of this property is "false". When the property `popupendmethod` is set to false, the event `nat:page.end` is raised when the user closes the pop-up window with the Close button in the upper right corner of the pop-up window.

Note: When the user closes a pop-up window using the Close button of the TITLEBAR control, the built-in event `nat:page.end` is always raised, no matter whether `popupendmethod` is set to "true" or not. With the `nat:popup.end` event, it is possible to find out that the Close button of the actual pop-up window was clicked (and not the Close button of a page within the pop-up window).

nat:page.default
This event is sent if the Natural for Ajax client needs to synchronize the data displayed on the page with the data held in the application. It is usually handled in the default event handler and just responded with a `PROCESS PAGE UPDATE`.

Other built-in events can be sent by specific controls. These events are described in the control reference.

User-defined Events

User-defined events are those events that the user has assigned to controls while designing the page layout with the Layout Painter. The names of these events are freely chosen by the user. The meaning of the events is described in the control reference.

Sending Events to the User Interface

The `PROCESS PAGE UPDATE` statement can be accompanied by a `SEND EVENT` clause. With the `SEND EVENT` clause, the application can trigger certain events on the page when resending the modified data.

The following events can be sent to the page:

nat:page.message
This event is sent to display a text in the status bar of the page. It has the following parameters:

Name	Format	Value
`type`	A or U	Sets the icon in the status bar ("S"=success icon, "W"=warning icon, "E"=error icon).
`short`	A or U	Short text.
`long`	A or U	Long text.

nat:page.valueList
This event is sent to pass values to a FIELD control with value help on request (see also the description of the FIELD control in the control reference). It has the following parameters:

Name	Format	Value
id	A or U	A list of unique text identifiers displayed in the FIELD control with value help. The list must be separated by semicolon characters.
text	A or U	A list of texts displayed in the FIELD control with value help. The list must be separated by semicolon characters.

nat:page.xmlDataMode

This event is sent to switch several properties of controls on the page in one call to a predefined state. The state must be defined in an XML file that is expected at a specific place. See the information on XML property binding in the Application Designer documentation for further information.

Name	Format	Value
data	A or U	Name of the property file to be used.

Using Pop-Up Windows

A rich GUI page can be displayed as a modal pop-up in a separate window. A modal pop-up window can open another modal pop-up window, thus building a window hierarchy. If a `PROCESS PAGE` statement and its corresponding event handlers are enclosed within a `PROCESS PAGE MODAL` block, the corresponding page is opened as a modal pop-up window.

The application can check the current modal pop-up window level with the system variable `*PAGE-LEVEL`. `*PAGE-LEVEL = 0` indicates that the application code is currently dealing with the main window. `*PAGE-LEVEL > 0` indicates that the application code is dealing with a modal pop-up window and indicates the number of currently stacked pop-up windows.

In order to modularize the application code, it makes sense to place the code for the handling of a modal pop-up window and the enclosing `PROCESS PAGE MODAL` block in a separate Natural module, for instance, a subprogram. Then the pop-up window can be opened with a `CALLNAT` statement and can thus be reused in several places in the application.

Example program `MYPAGE-P`:

```
DEFINE DATA LOCAL
1 FIELD1 (U) DYNAMIC
1 FIELD2 (U) DYNAMIC
END-DEFINE
*
PROCESS PAGE USING 'MYPAGE-A'
*
DECIDE ON FIRST *PAGE-EVENT
 VALUE U'nat:page.end',U'nat:browser.end'
  /* Page closed.
```

```
  IGNORE
 VALUE U'onPopup'
  /* Open a pop-up window with the same fields.
  CALLNAT 'MYPOP-N' FIELD1 FIELD2
  PROCESS PAGE UPDATE FULL
 NONE VALUE
  /* Unhandled events.
  PROCESS PAGE UPDATE
END-DECIDE
*
END
```

Example subprogram `MYPOP-N`:

```
DEFINE DATA PARAMETER
1 FIELD1 (U) DYNAMIC
1 FIELD2 (U) DYNAMIC
END-DEFINE
*
/* The following page will be opened as pop-up.
PROCESS PAGE MODAL
*
 PROCESS PAGE USING 'MYPOP-A'
*
 DECIDE ON FIRST *PAGE-EVENT
  VALUE U'nat:page.end',U'nat:browser.end'
   /* Page closed.
   IGNORE
  NONE VALUE
   /* Unhandled events.
   PROCESS PAGE UPDATE
 END-DECIDE
*
END-PROCESS
*
END
```

Debugging Modal Pop-up Windows

When debugging Natural for Ajax applications with modal pop-up windows, it is recommended to use an external web browser instead of the internal web browser of Eclipse. Only then it is possible to step into the modal pop-up handling code.

See also *Debugging Natural Applications* in *Using NaturalONE*.

To define the use an external web browser

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

2 Expand the **General** node and select **Web Browser**.

3 Select the **Use external web browser** option button and choose the **OK** button.

Using Natural Maps

Rich internet applications written with Natural for Ajax need not only consist of rich GUI pages, but may also use classical maps. This is especially useful when an application that was originally written with maps shall only be partly changed to provide a rich GUI. In this case the application can run under Natural for Ajax from the very beginning and can then be "GUIfied" step by step.

Navigating between Pages and Maps

Due to the similar structure of programs that use maps and programs that use adapters, it is easy for an application to leave a page and open a map, and vice versa. For each rich GUI page, you write a program that displays the page and handles its events. For each map, you write a program that displays the map and handles its events. In an event handler of the page, you call the program that handles the map. In an "event handler" of the map, you call the program that handles the page.

Example for program `MYPAGE-P`:

```
DEFINE DATA LOCAL
1 FIELD1 (U20)
1 FIELD2 (U20)
END-DEFINE
*
PROCESS PAGE USING 'MYPAGE'
*
DECIDE ON FIRST *PAGE-EVENT
 VALUE U'nat:page.end',U'nat:browser.end'
  /* Page closed.
  IGNORE
 VALUE U'onDisplayMap'
  /* Display a Map.
  FETCH 'MYMAP-P'
 NONE VALUE
  /* Unhandled events.
  PROCESS PAGE UPDATE
END-DECIDE
*
END
```

Example for program `MYMAP-P`:

```
DEFINE DATA LOCAL
1 FIELD1 (U20)
1 FIELD2 (U20)
END-DEFINE
*
SET KEY ALL
INPUT USING MAP 'MYMAP'
*
DECIDE ON FIRST *PF-KEY
 VALUE 'PF1'
  /* Display a rich GUI page.
  FETCH 'MYPAGE-P'
 NONE VALUE
  REINPUT WITH TEXT
  'Press PF1 to display rich GUI page.'
END-DECIDE
*
END
```

Using Pages and Maps Alternatively

An application can also decide at runtime whether to use maps or rich GUI pages, depending on the capabilities of the user interface. The system variable `*BROWSER-IO` lets the application decide if it is running in a web browser at all. If this is the case, the system variable tells whether the application has been started under Natural for Ajax and may thus use both maps and pages, or whether it has been started under the Natural Web I/O Interface and may thus use only maps.

Example:

```
DEFINE DATA LOCAL
1 FIELD1 (U20)
1 FIELD2 (U20)
END-DEFINE
*
IF *BROWSER-IO = 'RICHGUI'
  /* If we are running under Natural for Ajax,
  /* we display a rich GUI page.
  PROCESS PAGE USING 'MYPAGE'
  DECIDE ON FIRST *PAGE-EVENT
    VALUE U'nat:page.end',U'nat:browser.end'
      /* Page closed.
      IGNORE
    NONE VALUE
      /* Unhandled events.
      PROCESS PAGE UPDATE
  END-DECIDE
ELSE
  /* Otherwise we display a map.
```

```
  SET KEY ALL
  INPUT USING MAP 'MYMAP'
  DECIDE ON FIRST *PF-KEY
    VALUE 'PF1'
      /* Map closed.
      IGNORE
    NONE VALUE
      REINPUT WITH TEXT
      'Press PF1 to terminate.'
END-DECIDE
END-IF
*
END
```

4 Developing Customized Logon and Disconnect Pages

- General Information 30
- Adding the NatLogon and/or NatDisconnect Page Layout to a NaturalONE Project 30
- What Can be Customized? 31
- What Cannot be Customized? 32
- Advanced Customizations 32
- Deployment 33

General Information

In a production environment, a Natural for Ajax application starts with a logon page and ends with a disconnect page. Many applications do not show these pages directly. For more information, see *Client Configuration* in the documentation for the standalone version of Natural for Ajax, and especially these topics:

- *Using the Configuration Tool*
- *Starting a Natural Application from the Logon Page*
- *Starting a Natural Application with a URL*
- *Wrapping a Natural for Ajax Application as a Servlet*

See also *Creating Your Own Workplace Application* in this documentation.

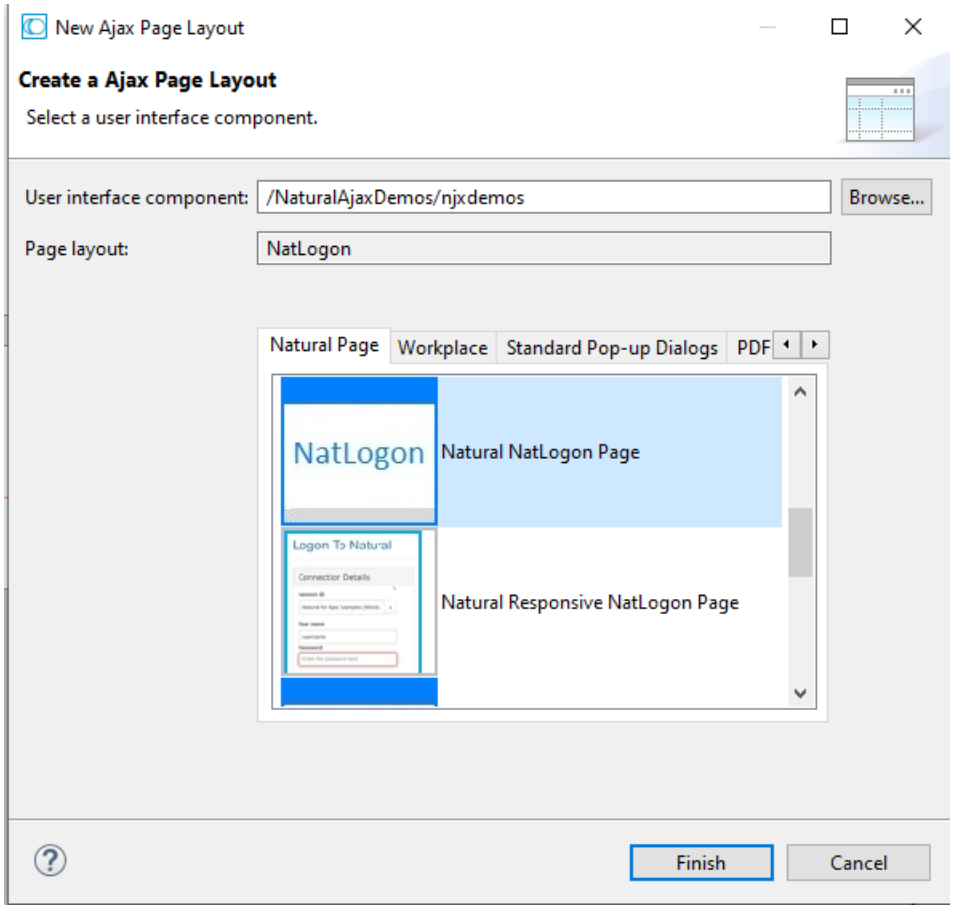
If you prefer that your applications are to show the logon or disconnect page directly, you will most likely customize the page layouts of these pages in order to adapt the rendering, the images, the style and/or the shown information. Natural for Ajax provides the following page layouts:

- **NatLogon**
  This is the page layout for the logon page.
- **NatDisconnect**
  This is the page layout for the disconnect page.

The topics below describe how to customize these page layouts.

Adding the NatLogon and/or NatDisconnect Page Layout to a NaturalONE Project

Open the 'New Ajax Page Layout' and select either 'Natural NatLogon Page' or 'Natural Responsive NatLogon Page' as shown in the screenshot below. Select the Page Layout depending on whether your application uses the responsive or the non-responsive control set.



Now you can customize the NatLogon and/or NatDisconnect page layout with the Layout Painter in the usual way, but you must follow the rules described below.

What Can be Customized?

You can customize rendering and styling. For example, you can modify style settings and rendering properties such as the height and width. You can rearrange the existing controls. You can add, modify and remove images. You can also add your own controls if they do not have data bindings to Natural.

For example, if you want to add your own application image to the page, you can define the ICON control as follows:

```
<icon image="../my_ui/images/MyImage.png" ></icon>
```

What Cannot be Customized?

You must leave all data bindings of the `*prop` properties as they are. You must not remove any bindings.

You cannot add your own custom data bindings. For example, you cannot add your own FIELD, TEXTOUT or other controls with properties bound to Natural data.

You cannot add your own events. This means, you cannot trigger your own Natural events on the server.

Advanced Customizations

The NatDisconnect page layout, for example, contains the following control:

```
<textout valueprop="disconnectMessage" fgcolorprop="colorProp">
</textout>
```

Often, applications do not want to show the disconnect message. As stated above, you cannot remove the data bindings "disconnectMessage" and "colorProp". But you can do the following: Add XCIDATADEF controls and move the data bindings from the TEXTOUT control to the XCIDATADEF controls:

```
<xcidatadef dataprop="disconnectMessage"></xcidatadef>
<xcidatadef dataprop="colorProp"></xcidatadef>
```

The XCIDATADEF control is not visible in a rendered HTML page. Therefore, the corresponding message is no longer visible. The data binding, however, has not changed. This way you can remove controls from the NatLogon and NatDisconnect pages without changing the data bindings.

If you use a customized NatDisconnect page with SUBCISPAGE controls, we recommend that you remove all code which triggers the `onLogon` event (for buttons, hot keys, etc.). For example: The original NatDisconnect page contains a hot key to navigate to the NatLogon page on pressing ENTER:

```
<natpage hotkeys="13;onLogon" ...>
```

You can simply remove the hot key definition. This will prevent the navigation to the NatLogon page from within a SUBCISPAGE:

```
<natpage ...>
```

Deployment

When you generate a WAR deployment file (the default name is *wardeploy.xml*) for your project, this file will automatically contain all required rules for deploying your customized NatLogon and NatDisconnect page layouts. You don't need to edit the WAR deployment file manually and you don't need to copy any additional files to your production environment. For more information on the WAR deployment file, see ***Deploying the Application***.

5 Styling the User Interface

Use the Style Sheet Editor tool to develop your own application-specific style sheet. Follow the recommendations given under Style Sheet Editor of the Ajax Developer documentation to add the style sheet files to your user interface, so that they will automatically be packaged with the WAR deployment file. For more information on the WAR deployment file, see ***Deploying the Application***.

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

6 Deploying the Application

- General Information 38
- Content of a Natural for Ajax Web Application 38
- Content of the Sample webconfig Directory 39
- Using the Deployment Wizard for Web Applications 42
- Starting the Deployment from Eclipse 50
- Starting the Deployment from the Command Line 51
- Status Code Handling 52
- Deploying the Web Application Archive (.war) 52

General Information

A Natural for Ajax application consists of two parts that are usually installed on two different machines. On one hand, there are Natural modules (adapters, programs, subprograms and other Natural objects) that are installed on a Natural server. On the other hand, there are page layouts of rich GUI pages and other user interface related files that are installed in a Natural for Ajax environment on an application server or web container.

The Natural modules are deployed to the Natural server with the Natural Ant deployment wizard (see *Deploying Natural Applications* in the *Using NaturalONE* documentation).

The user interface part of an application is packaged as a standard web application that can be deployed to an application server or web container. For this purpose, NaturalONE offers an additional deployment wizard, which is described below. This wizard collects all required information needed to package the user interface components of a Natural for Ajax application to a web application. It creates an Ant script which performs the packaging process. This process creates either a web archive file (*.war*) which can be deployed to one of the supported web containers (such as Apache Tomcat) or application servers (such as IBM WebSphere), or it creates single user interface component packages which can be deployed into an already existing web application.

The deployment process can either be started from within the NaturalONE Eclipse environment or via the Ant command line utility.

For more detailed information about Natural for Ajax deployment see also *Deploying the Application* in the *Natural for Ajax* documentation.

Content of a Natural for Ajax Web Application

A deployed Natural for Ajax web application consists of the following elements:

- **User Interface Components**
  These are the elements which constitute the user interface of the application, such as layout pages, style sheets, images and language files.
- **Configuration Tool**
  This tool enables you to modify the session configuration (*sessions.xml*) after the deployment.
- **Runtime License File**
  To execute a Natural for Ajax application outside the development environment of NaturalONE, a Natural for Ajax runtime license is required. You received this license file with your Natural for Ajax product kit.

- **Special Files for the Web Application**
  You can customize your web application further by adding specific configuration files and web pages. To do this, you can add a directory named *webconfig* to your project. The files contained in this directory are packaged to specific, well-defined places in your web application:
  - **Root Directory**
    The files from your project's *webconfig/root* directory are placed in the root directory of the web application. This directory may contain, for example, your own application-specific *index.html* file.
  - ***WEB-INF* Directory**
    The files from your project's *webconfig/web-inf* directory are placed in the *WEB-INF* directory of the web application. Usually, this contains your own application-specific *web.xml* and *sessions.xml* files.
  - ***resources* Directory**
    The files from your project's *webconfig/resources* directory are placed in the *resources* directory of the web application. Usually, this contains your own application-specific style sheets.

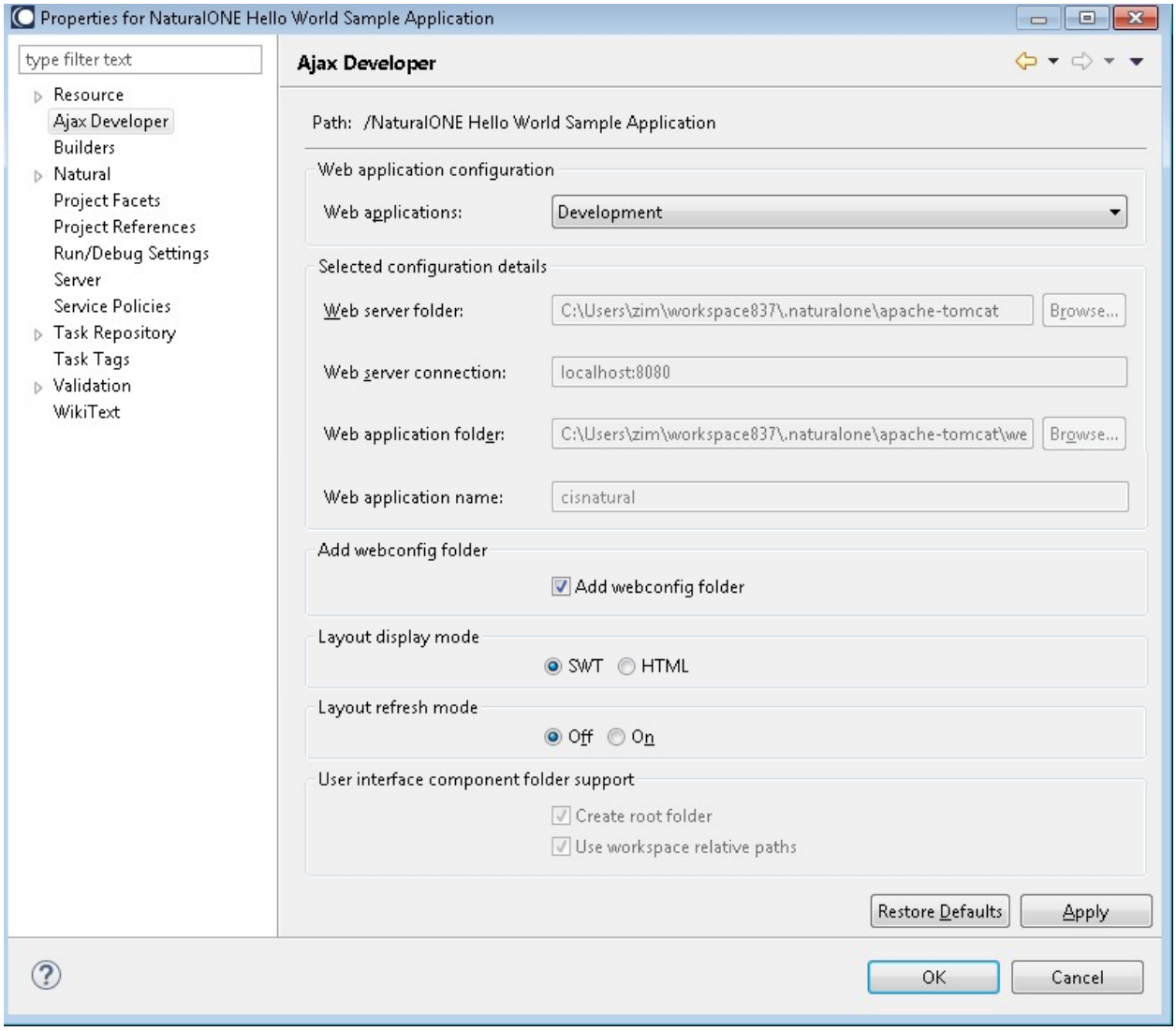
To make it simple, a sample *webconfig* directory is provided (see **below**) that you can import into your project before deployment. If you are an experienced developer of web applications, you can create and modify the files in this directory.

Important: If the deployment mode **Deploy as UI update** is selected in the deployment wizard, only the user interface components are deployed. In this case, it is assumed that all other elements mentioned above have already been installed on the application server.

Content of the Sample webconfig Directory

Before packaging the application with the deployment wizard, it is useful to import the sample *webconfig* directory into your project. The files contained in this sample directory can be customized as needed, or left as they are. In their default state, they provide a comfortable approach to launch the application.

You can import the sample *webconfig* directory into your project by clicking the box **Add webconfig folder** on the **Ajax Developer** tab of the project properties:



The sample *webconfig* directory contains the following subdirectories:

- ***web-inf***
  If your project contains the files listed below, the packaging script adds them to the web application and replaces certain variables with the values you have specified in the deployment wizard. The files will be placed into the *WEB-INF* directory of the web application.
  - ***sessions.xml***
    This configuration file defines the sessions that can be invoked from the logon page. After the deployment, the configuration file can be modified using the configuration tool.
  - ***web.xml***
    This is the central configuration file for a web application. Modify this file only if you have experience with web applications.

If you need to support different application servers which require different versions of the files *sessions.xml* and *web.xml*, you can place these files in specific subdirectories of the *WEB-INF* directory of the web application. For example:

```
- WEB-INF
    - Tomcat
      - sessions.xml
      - web.xml
    - WebSphere
      - sessions.xml
      - web.xml
    - Wildfly
      - sessions.xml
      - web.xml
```

- ***root***
  This directory contains the following files which will be placed into the root directory of the web application:
  - ***start.html***
    This file is used to start the web application in the browser. You start the application with the following URL:

    ```
    http://<host>:<port>/<webcontext>/start.html
    ```

  - ***index.html***
    This file provides links for starting the web application in the browser and in the SWT client. These correspond to the above URLs. In addition, it provides links for invoking the configuration tool and the tracing and monitoring tools for the application.
- ***resources***
  This directory contains the following file which will be placed in the *resources* directory of the web application:
  - ***sample.css***
    This is an example style sheet which is used to control the font, the color and the representation of the PF keys of classic I/O pages that your application might use. You can adapt this style sheet as needed or create more style sheets.

Using the Deployment Wizard for Web Applications

The deployment wizard creates a WAR deployment file in your project root. This is an Ant script. You can create one or more WAR deployment files for a project, and you can also load an existing WAR deployment file and modify the current settings.

To use the deployment wizard

1. In the **Project Explorer** view or in the **Natural Navigator** view, select the project for which you want to create the deployment file.

   Or:

   If you want to load the settings of an existing WAR deployment file, select this file in the **Project Explorer** view or in the **Natural Navigator** view.
2. From the **File** menu or from the context menu, choose **New > Other**.
3. In the resulting **New** dialog box, expand the **Natural** node, select **Deploy Natural War Ant** and then choose the **Next** button.

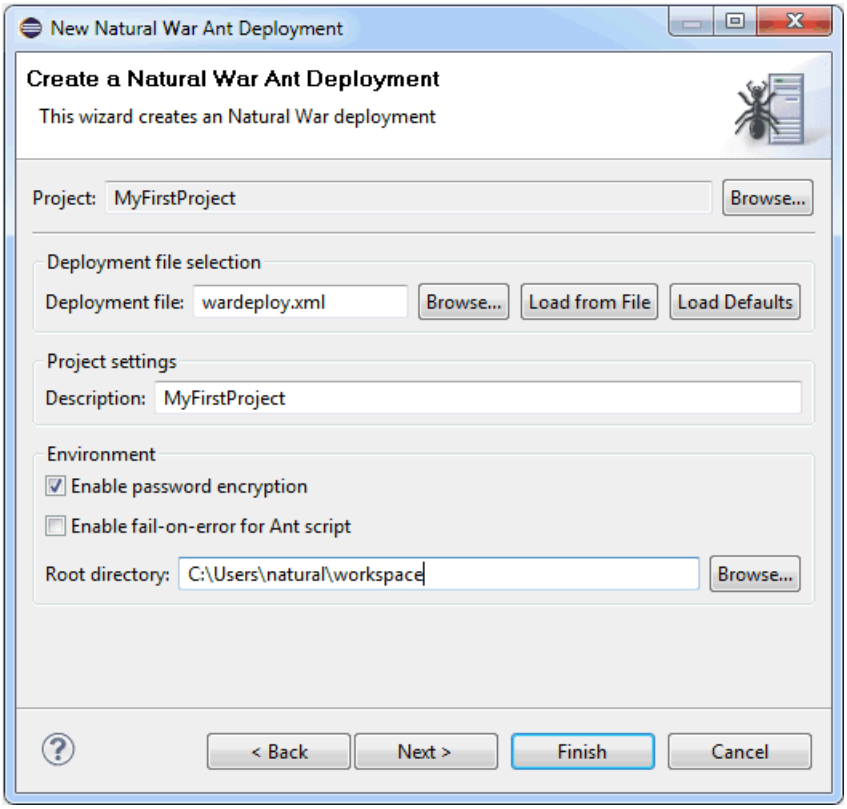
   The first page of the wizard appears (see below).
4. Specify all required information as described in the topics below. Use the **Next** button repeatedly to proceed from the first page of the wizard to the last page.
5. When all required information has been provided, choose the **Finish** button.

The different pages of the deployment wizard are described in the topics below.

- General Settings
- User Interface Components
- Repository
- Runtime and Web Settings
- Session Settings

General Settings

On the first page of the wizard, you define general settings for the deployment.



Deployment file

The default name for the WAR deployment file is *wardeploy.xml*. This name is shown in this text box when an existing WAR deployment file was not selected while invoking the wizard. However, when an existing WAR deployment file was selected, the name of the selected file is shown and the settings from this file are automatically loaded.

You can enter any other name for your new deployment file. It is recommended that your new deployment file also has the extension ".xml".

Note: If you keep the name *wardeploy.xml*, the settings from an existing WAR deployment file with the same name are loaded the next time you select the project and invoke the wizard.

If you want to load an existing WAR deployment file, choose the **Browse** button. A dialog appears, providing for selection all WAR deployment files in the current project. Next, you have to choose the **Load from File** button. Otherwise, the settings in this file are not shown in the wizard and may thus be overwritten unintentionally.

If you want to return to the default settings of the deployment wizard (this also includes the information that can be specified on the other pages of the wizard), choose the **Load Defaults** button.

Description

This descriptive text is displayed in the administration tool of the application server or web container after the web application has been deployed.

Enable password encryption

When enabled, all passwords that are used in the deployment file are stored in an encrypted format.

Enable fail-on-error for Ant script

When enabled, the Ant script reports errors and terminates in the case of a build failure.

When disabled, the Ant script still reports errors but build failures are only triggered in severe situations (see also ***Status Code Handling***).

Root directory

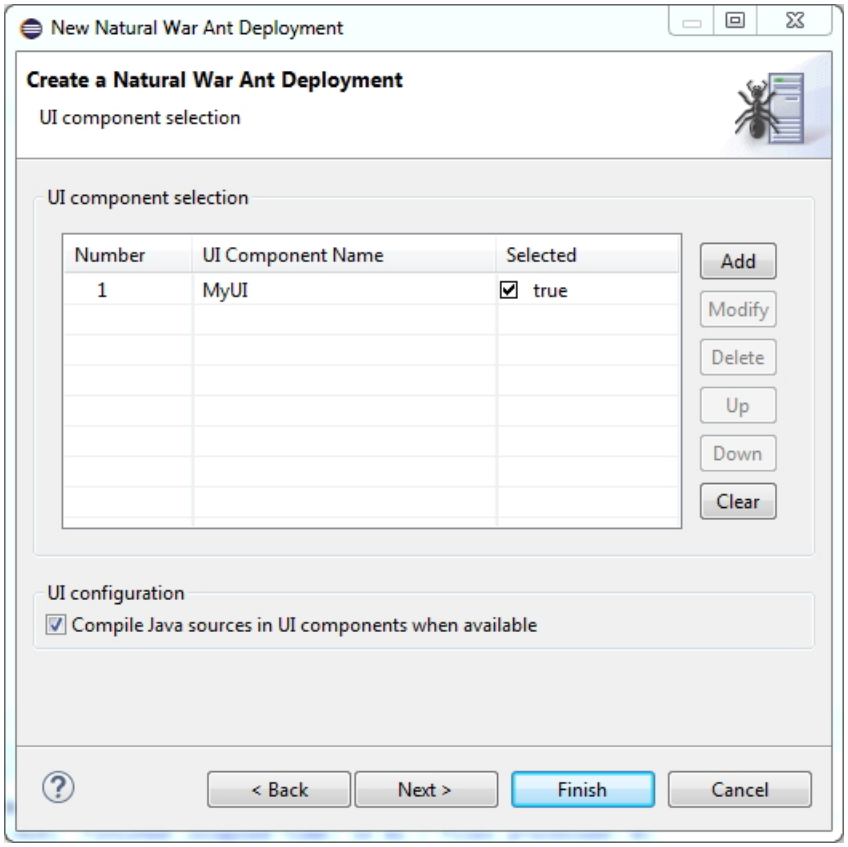
This path should only be changed when you intend to start the deployment from the command line (that is, when the deployment is not to be started from Eclipse).

Specify the working directory in which the selected project is to be checked out and where the processing takes place. When the deployment is supposed to run on a different machine, you can insert the desired root path via copy-and-paste.

User Interface Components

On the second page of the wizard, you specify the user interface components that are to be processed.

By default, all user interface components in the project are selected. If you want to exclude a user interface component from processing, remove the check mark in the **Selected** column.



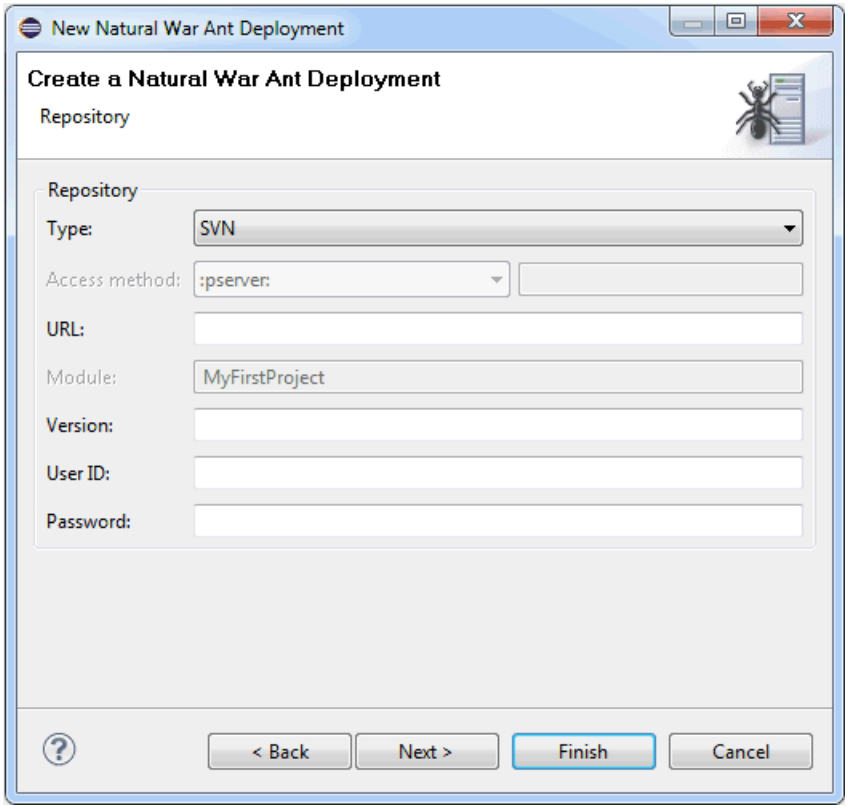
The following applies if **Deploy as war archive** has been selected:

Style sheets for the user interface should be contained in a subfolder *styles* of your user interface component. In this case, the *.css* files are automatically regenerated from the *.info* files. If your style sheets are in a different folder, you need to take care of the regeneration yourself. To package your own *cisconfig.xml* file with the *.war* file, put the *cisconfig.xml* file into a subfolder *cisconfig* of your user interface.

Select **Compile Java sources in UI components when available** if the user interface components contain Java sources that need to be compiled for deployment. The Java sources must reside in the predefined sub folder *src* of your user interface component, for example: *<myui>/src*. Please note that a Java compiler (i.e. a JDK kit) must be accessible for the Ant script in that case.

Repository

On the third page of the wizard, you define all settings related to the versioning repository. This can be either Subversion (SVN), GIT or CVS. The settings are used to check out the entire project.

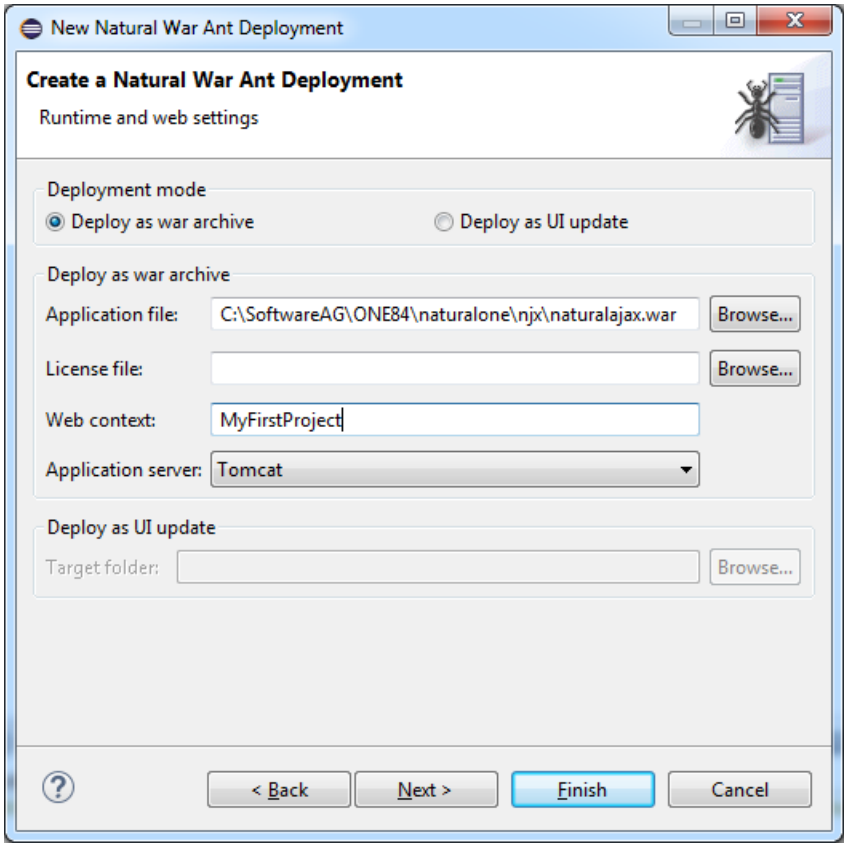


From the **Type** drop-down list box, select the type of versioning repository that you are using, and then specify all required information. The names of the text boxes and their availability changes according to the selected type.

The wizard usually collects a set of default information as given for the selected project. In most cases, only minor corrections have to be made to the defaults, for example, user ID and password may have to be provided.

Runtime and Web Settings

The fourth page of the wizard shows information which applies to the web application that is to be created.



Deployment mode

The deployment wizard can either create a *.war* archive or deploy single user interface components as updates. Depending on your choice (**Deploy as war archive** or **Deploy as UI update**), different options are available on this page of the wizard.

Application file

Only available when **Deploy as war archive** is selected.

Your web application is created based on a basic Natural for Ajax web application that is provided with your NaturalONE installation. This text box is preset with the path to this web application. Normally, you should not change this setting.

License file

Only available when **Deploy as war archive** is selected.

In a runtime environment, a Natural for Ajax runtime license is required. Specify the path to this license file. You received a runtime license file with your Natural for Ajax product kit.

Web context

Only available when **Deploy as war archive** is selected.

The web context is the root of all URLs of your web application. When you later deploy the application to an application server or web container, you will start it with the following URL:

```
http://<host>:<port>/<webcontext>/start.html
```

where `<host>` and `<port>` are the host name and port number of your application server or web container.

Application server

Only available when **Deploy as war archive** is selected.

Select the type of application server for which the *.war* archive is to be created. Different application servers might require different sets of configuration files. This option allows you to select the application server-specific files as discussed under ***Content of the Sample webconfig Directory***.

Target folder

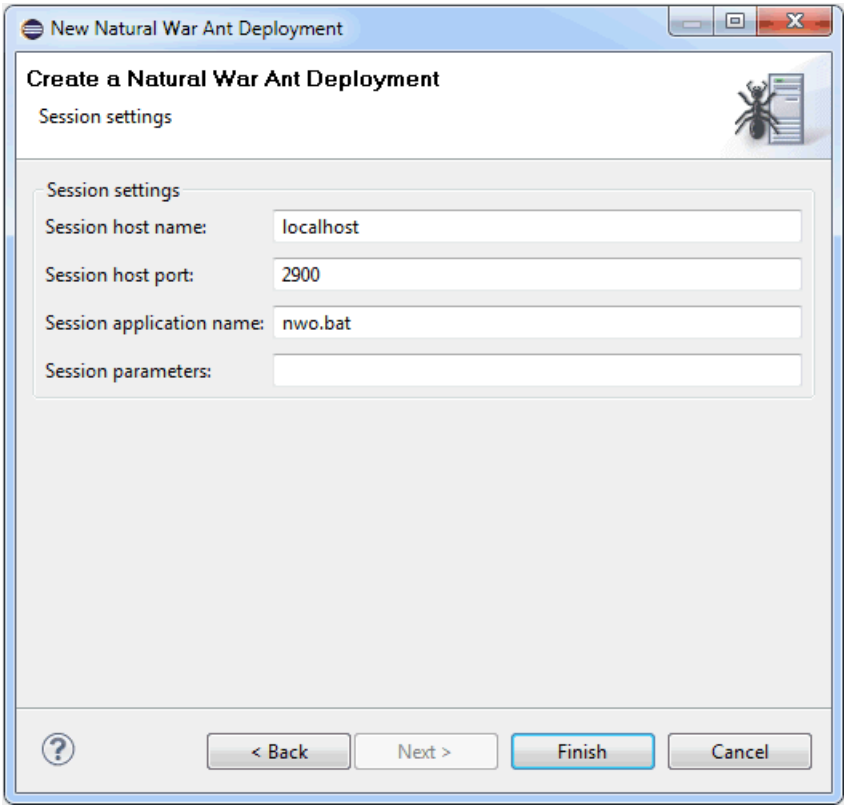
Only available when **Deploy as UI update** is selected.

Specify the destination folder for the UI components. When that destination folder is a valid web application folder then an automatic deployment is performed. The UI component is copied into the web application, the deployment Ant script waits for the web application update to finish and collects and evaluates the results.

When the destination folder is not a valid web application folder then the UI component is simply copied to that location.

Session Settings

The fifth (and last) page of the wizard shows the options that your web application will use by default to connect to the Natural server. The packaging script places the values you specify here into the *sessions.xml* file of your web application. After you have deployed the application, you can use the configuration tool to modify these settings or to add alternative connection parameters.



Session host name

The name or TCP/IP address of the server on which Natural and the Natural Web I/O Interface server are running.

Session host port

The TCP/IP port number on which the Natural Web I/O Interface server is listening.

Session application name

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

  ```
  TEST01 data1,data2
  ```

- **Natural on Linux and Cloud**
  The name of the Linux shell script for starting the Natural application (a file similar to *nwo.sh*).
- **Natural for Windows**
  The name of the Windows command file (*.bat*) for starting the Natural application.

Session parameters

Optional. Parameters for starting the Natural application. This can be stack parameters, a parameter file/module or other Natural-specific information.

- **Natural for Mainframes**
  Used to pass dynamic Natural profile parameters to the session, for example:

```
SYSPARM=(MYPARMS) STACK=(LOGON MYAPPL)
```

  Note: It is recommended to specify the Natural program that starts the application with the option **Session application name** instead of passing it with the profile parameter `STACK`.

- **Natural on Linux and Cloud and Natural for Windows**
  Used when the above shell script (Linux) or command file (Windows) uses the parameter `$5` after "natural", for example:

```
PARM=MYPARM STACK=(LOGON MYLIB;MENU)
```

Starting the Deployment from Eclipse

When you start the deployment process from Eclipse, it is not possible to execute the `checkout` and `update` targets of the deployment file since these targets would access the versioning repository, and this is not feasible from within an Eclipse environment. If you want to check out a specific revision from the versioning repository or if you want to update your project with sources from the versioning repository, you have to start the deployment from the command line as described below.

For testing purposes, for example, it is helpful to start the deployment process from Eclipse. Since the WAR deployment file is an Ant script, the built-in Eclipse functionality of starting Ant scripts is used here. The `build` target of the Ant script will then be executed.

To start the deployment from Eclipse

- In the **Project Explorer** view or in the **Natural Navigator** view, select your WAR deployment file, invoke the context menu and choose **Run As > Ant Build**.

  The deployment process is started, and the output of the deployment file is written to the **Console** view.

  When **Deploy as war archive** has been selected as the deployment mode, the web application (*.war* file) is created in the project directory. To see the file in the **Project Explorer** view or in the **Natural Navigator** view, a refresh may be required.

  When **Deploy as file export** has been selected as the deployment mode, the directories and files are generated within the specified destination folder.

Starting the Deployment from the Command Line

You can start the deployment process from a Windows command line such as the Command Prompt (*cmd.exe*) or from a shell command line on a Linux system. When you start the deployment from the command line, special requirements must be met.

The following topics are covered below:

- Prerequisites
- Starting the Deployment

Prerequisites

The prerequisites for deploying a web application are the same as for deploying Natural applications. See *Prerequisites* in *Deploying Natural Applications*, which is part of *Using NaturalONE*. However, you have to copy the WAR deployment file to the root directory (instead of the Natural deployment file).

Starting the Deployment

When all prerequisites are in place, the deployment can be started by issuing specific Ant calls. This section just provides some examples (where the default name *wardeploy.xml* is used).

- Print the help screen of the Ant script:

```
ant -lib path-to-mylib -f wardeploy.xml help
```

- Perform an initial checkout of the project sources from the versioning repository:

```
ant -lib path-to-mylib -f wardeploy.xml checkout
```

- Perform an update of the project sources from the versioning repository:

```
ant -lib path-to-mylib -f wardeploy.xml update
```

- With a single call, perform an update of the project sources from the versioning repository first, and then package the web application:

```
ant -lib path-to-mylib -f wardeploy.xml update build
```

- In the above examples, the logging information is written to standard output. If logging information is to be written to a file, use a call such as the following:

```
ant -lib path-to-mylib -f wardeploy.xml update build -logfile mylogfile.txt
```

Status Code Handling

The Ant deployment script for Ajax can run in two status code modes. The mode can be toggled by specifying the command line parameter `-Dnatural.ant.ajax.failonerror` as described in the following table.

Command Line Option	Description
`-Dnatural.ant.ajax.failonerror=no`	Only severe errors such as missing project directories will lead to a build failure with a status code other than 0. This is the default mode.
`-Dnatural.ant.ajax.failonerror=yes`	In addition to the severe errors described above, errors occurring during checkout or update will also lead to a status code other than 0 and hence will lead to a build failure.

Note: The default mode can also be changed on the **first page** of the deployment wizard.

When the additional status code handling has been enabled, the Ant tool as well as the internally used tools such as SVN, GIT or CVS clients may issue specific status codes. In case the status codes are unclear, refer to the documentation of these tools.

Deploying the Web Application Archive (.war)

The way you actually deploy the web application to a production application server or web container depends on the server you are using. Usually, an application server or web container provides an Administration Console for deploying and configuring an application (Apache Tomcat, IBM WebSphere, Wildfly Application Server). Or you can simply copy the web application into an application directory on the server (Apache Tomcat, Wildfly Application Server).

Note: If you are using open source products like Apache Tomcat, sometimes the Administration Consoles of these products have platform or browser-specific differences. It is always good to check the downloads of these open source products for updates.

7 Natural Parameters and System Variables

The following Natural parameters and system variables are evaluated in Natural for Ajax applications and sent to Application Designer:

- `DC`

  The character assigned to the `DC` parameter is used in the representation of decimal fields in Application Designer.

- `DTFORM`

  This parameter is used for all date fields in Application Designer pages. In your application, the date is shown according to the setting of the `DTFORM` parameter.

- `EMFM`

  The value of the `EMFM` parameter is evaluated for fields in Application Designer pages for which a dynamic edit mask has been assigned. See also ***Usage of Edit Masks***.

- `*CURS-FIELD`

  Identify the operand that represents the value of the control that has the input focus. When the Natural system function `POS` is applied to a Natural operand that represents the value of a control, it yields the identifier of that operand.

- `*LANGUAGE`

  Change the language while an application is running. See also ***Multi-Language Management***.

8 Usage of Edit Masks

- General Information 56
- Data Types with Edit Masks 56
- Natural Profile Parameters 58
- Specifying Edit Masks in Layouts 58
- Edit Masks at Runtime 59

General Information

Natural for Ajax supports a subset of the Natural edit mask concept in order to support output formatting for most of the commonly used fields.

If edit mask support is specified for a field, the field content is

- rendered according to the edit mask during output, and
- checked for validity against the edit mask during user input.

Due to the nature of data being handled with a Natural for Ajax client, not all of the different Natural edit mask types make sense. Therefore, only a subset of edit mask types is available for Natural for Ajax.

Data Types with Edit Masks

In all controls that support the property `datatype`, edit masks can be specified for the data types listed in the topics below.

- Edit Masks for Numeric Fields
- Edit Masks for Alphanumeric Fields
- Edit Masks for Date and Time Fields
- Edit Masks for Logical Fields

For detailed information on edit masks, see the Natural documentation for the appropriate platform.

Edit Masks for Numeric Fields

Edit masks for numeric fields can be specified for the following data types:

- `N` *`n.n`*
- `P` *`n.n`*
- `int`
- `long`
- `float`
- `xs:double`
- `xs:byte`
- `xs:short`
- `xs:decimal`

The full set of Natural numeric edit masks can be applied for these data types.

Edit Masks for Alphanumeric Fields

Edit masks for alphanumeric fields can be specified for the following data type:

- `string` *n*

The full set of Natural alphanumeric edit masks can be applied for this data type.

Edit Masks for Date and Time Fields

Edit masks for date and time fields can be specified for the following data types:

- `date`
- `time`
- `timestamp` (can only be displayed)
- `xs:date`
- `xs:time`
- `xs:dateTime` (can only be displayed)

A subset of the Natural edit masks can be applied for these data types.

Edit masks for date fields may contain the following characters:

Character	Usage
`DD`	Day.
`ZD`	Day, with zero suppression.
`MM`	Month.
`ZM`	Month, with zero suppression.
`YYYY`	Year, 4 digits.
`YY`	Year, 2 digits.
`Y`	Year, 1 digit. Must not be used for input fields.

The time in a date/time edit mask may contain the following characters:

Character	Usage
T	Tenths of a second.
SS	Seconds.
ZS	Seconds, with zero suppression.
II	Minutes.
ZI	Minutes, with zero suppression.
HH	Hours.
ZH	Hours, with zero suppression.

Edit Masks for Logical Fields

Edit masks for logical fields can be specified for the following data types:

- L
- xs:boolean

The full set of Natural logical edit masks can be applied for these data types.

Natural Profile Parameters

The following Natural profile parameters are evaluated for the edit mask processing of Natural for Ajax:

- DC
- EMFM

For detailed information on these profile parameters, see the Natural documentation for the appropriate platform.

Specifying Edit Masks in Layouts

An edit mask is added to a specific data type in the following way:

◢ Validation	
datatype	N4.2
decimaldigits	
decimaldigitsprop	
digits	
digitsprop	
editmask	*EURZZ9.9
spinrangemax	
spinrangemin	
validation	
validationprop	

The `datatype` property of a field is specified (here the numeric type `N4.2`) and the `editmask` property is filled with the proper (here numeric) edit mask.

Edit Masks at Runtime

At runtime, fields with edit masks are processed as follows:

- When a field has an edit mask and when a value is to be displayed in that field, the value is processed and formatted according to the edit mask and is displayed afterwards.
- When a user enters a value into a field which has an edit mask, the value is validated against that edit mask and the real value is extracted from the entered value by stripping the irrelevant portions of the edit mask.

9 Multi-Language Management in Ajax

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

There are two translation aspects:

- All literals in the GUI definitions of a layout are replaced by strings which are language-specific. This is based on the multi-language management of Application Designer.

  Note: Detailed information on the multi-language management is provided in the Application Designer documentation at *Multi-Language Management*.

- Literals that are contained in your application code are handled with the language management of Natural.

In a Natural for Ajax application, both language management systems are related by common language codes. The language codes used are those that are defined for the Natural profile parameter `ULANG` and the system variable `*LANGUAGE`.

The Application Designer documentation describes how the text files containing the language-dependent texts are created and maintained (see the information on writing multi language layouts at the above URL). For a multi-lingual Natural for Ajax application, the names of the directories that contain the text files should be chosen according to the Natural language codes, for instance */multilanguage/4* for Spanish texts.

When an application is started from the Natural logon page, the user can select the language to be used. Depending on the selected language, the same (Natural) language code is set up both in Application Designer and in the Natural session, so that both language management systems are then configured to use the same language.

Note: The language for a session can also be defined in the configuration file *sessions.xml*, using the Natural for Ajax configuration tool. See *Natural for Ajax Runtime Tools* in the

Configuration and Administration documentation , which is included in the Natural for Ajax documentation for the standalone version of this product.

It is also possible to change the language while an application is running. This is done by setting the Natural system variable `*LANGUAGE` in the Natural program. Each time this system variable is changed, Natural for Ajax changes the language code for the web pages when the next update of the page occurs.

For compatibility with the predefined multi language directories in Application Designer, the English and German texts need not be stored in */multilanguage/1* and */multilanguage/2*, but can be contained in */multilanguage/en* and */multilanguage/de*.

See also: *Multi-Language Management in Workplace Applications*.

10 Support of Right-to-Left Languages

Natural for Ajax supports right-to-left languages and bidirectional text without specific actions taken by the application. The browser displays and accepts bidirectional text always in the expected order.

Applications can use the same page layouts both in left-to-right and in right-to-left screen direction. To switch the screen direction, the statement `SET CONTROL` is used as follows:

Statement	Description
`SET CONTROL 'VON'`	Sets the screen direction to right-to-left.
`SET CONTROL 'VOFF'`	Sets the screen direction to left-to-right.
`SET CONTROL 'V'`	Switches from left-to-right to right-to-left screen direction and vice versa.

11 Server-Side Scrolling and Sorting

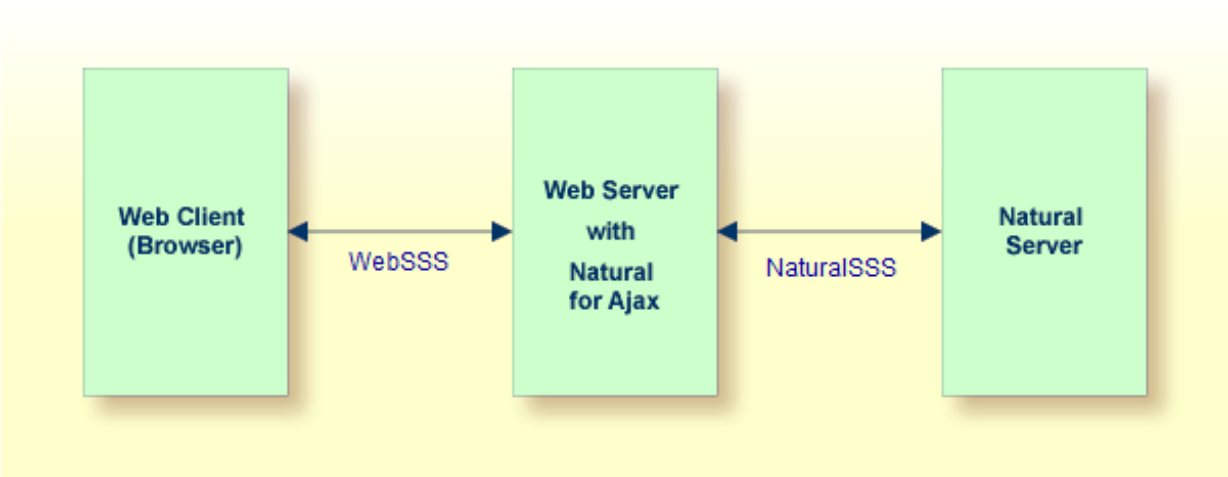
- General Information 66
- Variants of Server-Side Scrolling and Sorting 66
- Controls that Support Server-Side Scrolling and Sorting 70
- Data Structures for Server-Side Scrolling and Sorting 70
- Server-Side Scrolling and Sorting in Trees 72
- Events for Server-Side Scrolling and Sorting 73

General Information

It is often the case that a web application has to display an arbitrary amount of data in a grid control, for instance, the records from a database table. In these cases, it is mostly not efficient to send all data as a whole to the web client. Instead, it will be intended to display a certain amount of data to begin with and to send more data as the user scrolls through the page. To support this, the grid controls in Natural for Ajax support the concept of server-side scrolling and sorting.

Variants of Server-Side Scrolling and Sorting

The following graphic illustrates the different types of server-side scrolling and sorting that are supported by Natural for Ajax.



With respect to server-side scrolling and sorting, the following options can be used:

- **No Server-Side Scrolling and Sorting**
  The Natural application sends the grid data to the web server as a whole. The web server sends the grid data to the web client (browser) as a whole.

  Advantage: Neither the web server nor the Natural application are involved in the process of scrolling and sorting. As long as the user only scrolls and sorts, no round trip from the web client to the web server or to the Natural server is necessary.

  Disadvantage: A round trip between web server and Natural server that is triggered by other user actions transports the entire grid data.

- **Web Server-Side Scrolling and Sorting (WebSSS)**
  The Natural application sends the grid data to the web server as a whole. The web server sends the grid data to the web client (browser) in portions.

  Advantage: The Natural application is not involved in the process of scrolling and sorting. As long as the user only scrolls and sorts, no round trip from the web server to the Natural server is necessary.

  Disadvantage: A round trip between web server and Natural server that is triggered by other user actions transports the entire grid data.
- **Natural Server-Side Scrolling and Sorting (SSS_N)**
  The Natural application sends the grid data to the web server in portions. The web server sends the grid data to the web client (browser) in portions.

  Advantage: A round trip between web server and Natural application passes only the visible data portion.

  Disadvantage: The Natural application must support the process of scrolling and sorting with a specific application logic.

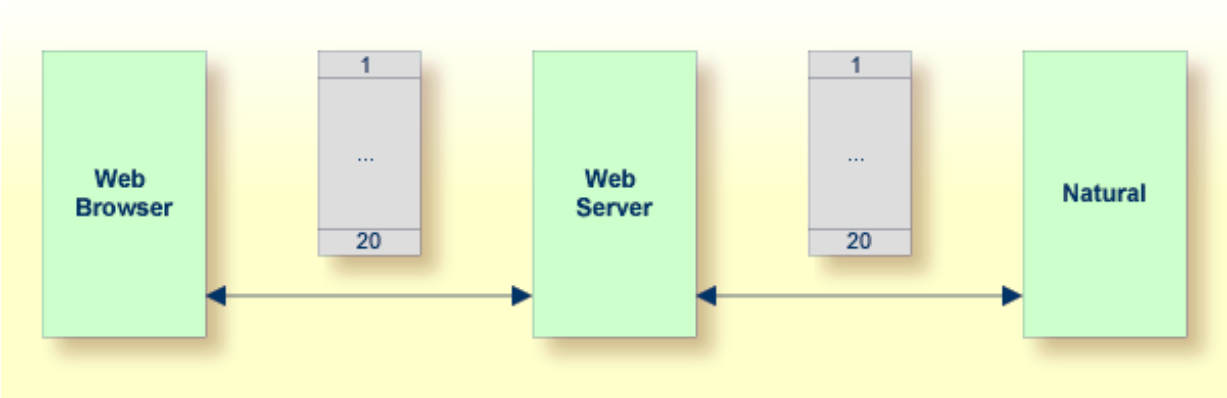
The decision between these options will often depend on the expected data volume. The application can decide dynamically at runtime which option to use.

The topics below show the difference between these three options.

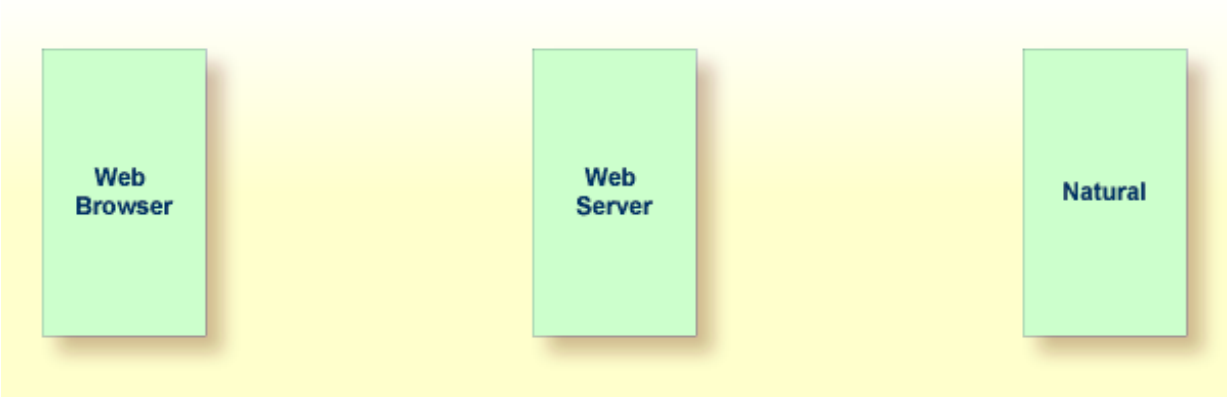
- No Server-Side Scrolling and Sorting
- Web Server-Side Scrolling and Sorting
- Natural Server-Side Scrolling and Sorting

No Server-Side Scrolling and Sorting

Step 1: The grid is configured at design time to a row count of twenty. The Natural application sends twenty rows and indicates that no further rows are to be expected (`SIZE=0`).

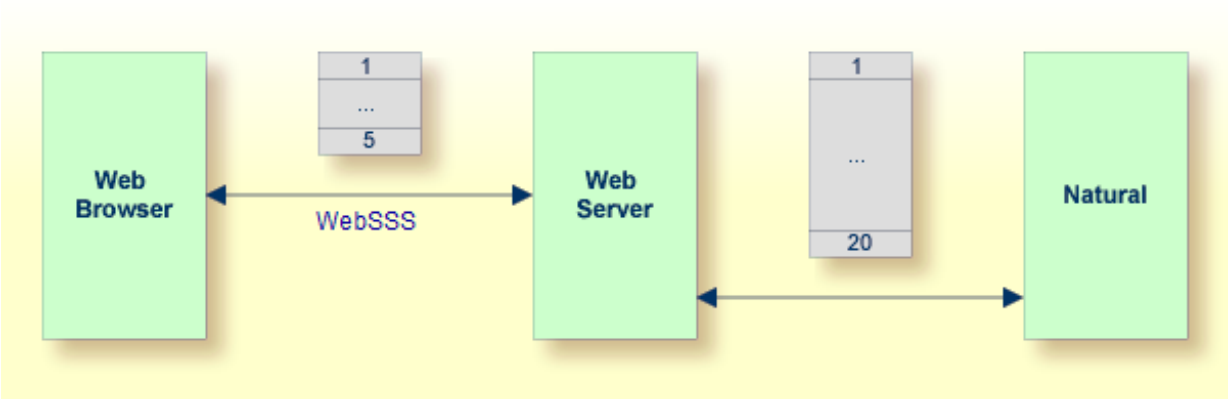


Step 2: When you scroll up and down, no server round trips to the web server or to the Natural application are performed.

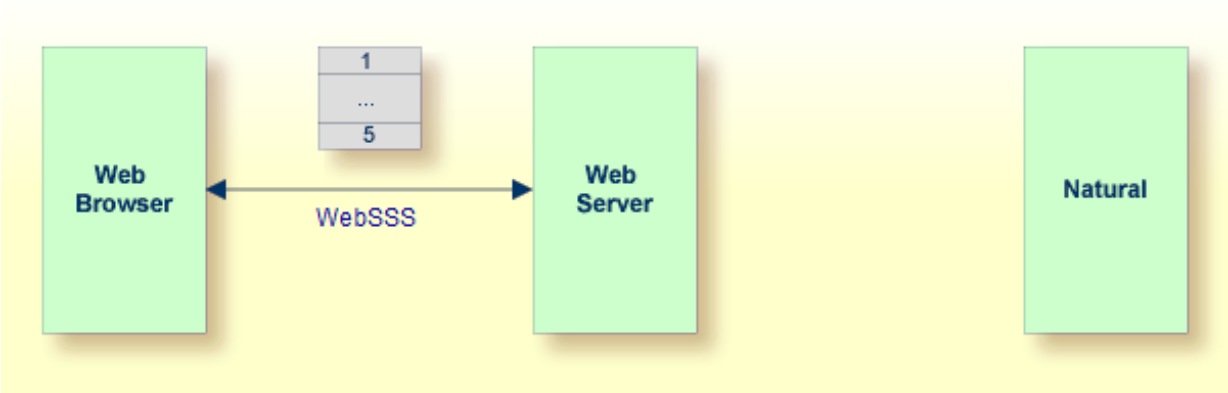


Web Server-Side Scrolling and Sorting

Step 1: The grid is configured at design time to a row count of five. The Natural application sends twenty rows and indicates that no further rows are to be expected (`SIZE=0`).

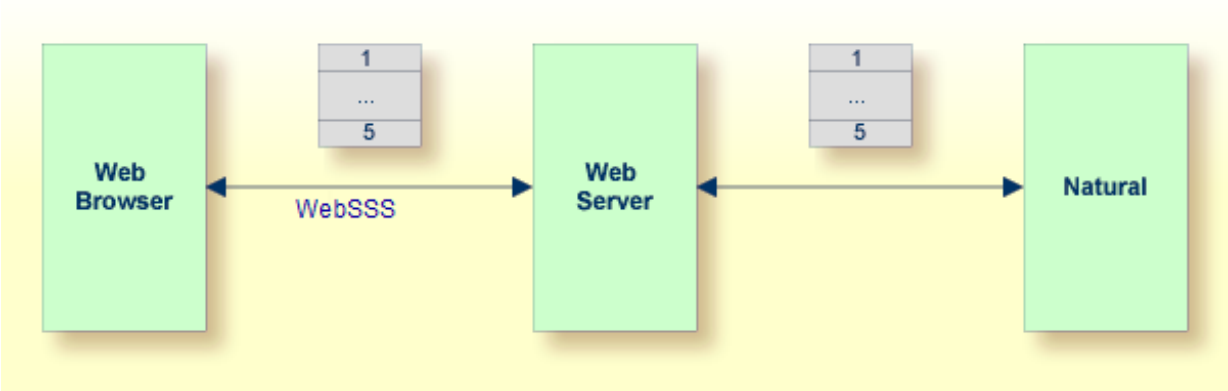


Step 2: When you scroll up and down, the web browser requests additional records from the web server There are no server round trips to Natural.

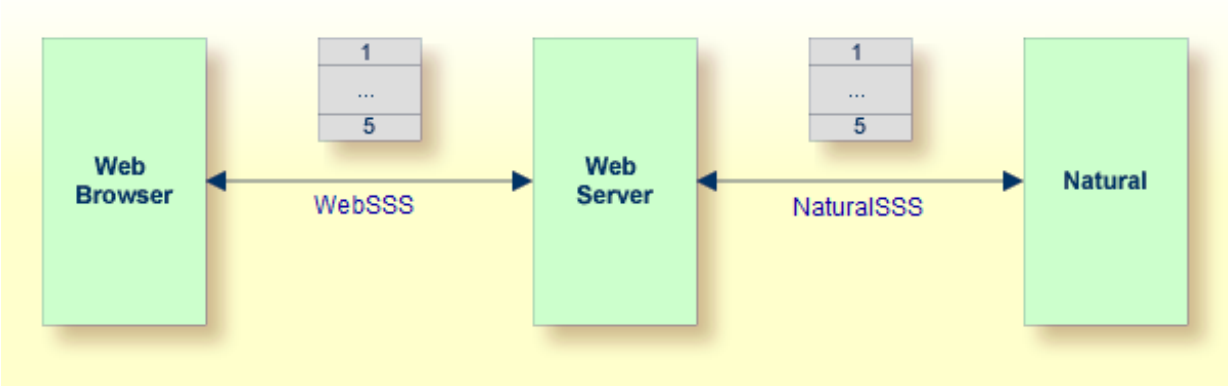


Natural Server-Side Scrolling and Sorting

Step 1: The grid is configured at design time to a row count of five. The Natural application sends five rows and indicates that further rows are to be expected (`SIZE=20`).



Step 2: When you scroll up and down, the web browser requests additional records from the web server. The web server requests additional records from the Natural application.



The Natural application can dynamically decide at runtime which option of server-side scrolling and sorting it wants to use. This can depend on the number of records contained in a search result.

- If the application does not want to use server-side scrolling and sorting at all, it sends as many rows to the web browser as the grid is configured to hold, or it sends fewer rows.
- If the application wants to use web server-side scrolling and sorting, it sends all available rows and sets the `SIZE` parameter to zero in the data structure that represents the grid in the application.
- If the application wants to use Natural server-side scrolling and sorting, it sends only part of the available rows and indicates in the `SIZE` parameter how many rows are to be expected altogether.

Controls that Support Server-Side Scrolling and Sorting

The following controls support server-side scrolling and sorting:

- TEXTGRIDSSS2
- ROWTABLEAREA2
- MGDGRID
- BMOBILE:SIMPLEGRID

Note: For compatibility reasons with earlier versions of Natural for Ajax, you have to set the `natsss` property of NATPAGE to true in order to activate server-side scrolling and sorting for the controls ROWTABLEAREA2 and MGDGRID. If this property is set to true, for all instances of these grid controls on a page, the necessary data structures are generated into the Natural adapter interface.

Data Structures for Server-Side Scrolling and Sorting

If you use the TEXTGRIDSSS2 control or if you use the ROWTABLEAREA2 or MGDGRID control and have set the property `natsss` to true for the page, the following additional data structure is generated into the adapter interface for each instance of these controls. This data structure is used to control the scroll and sort behavior at runtime.

```
1 LINESINFO
2 ROWCOUNT (I4)
2 SIZE (I4)
2 SORTPROPS (1:*)
3 ASCENDING (L)
3 PROPNAME (U) DYNAMIC
2 TOPINDEX (I4)
```

The name of the data structure is derived from the name of the variable that is bound to the grid. In this example, the variable `LINES` had been bound to the grid. Therefore, the name `LINESINFO` was generated.

With each event that is related to scrolling and sorting, the application receives the information how many rows it should deliver at least (`ROWCOUNT`) and the index of the first record to be delivered (`TOPINDEX`).

In `SORTPROPS`, the application receives the information in which sort sequence the records should be delivered and by which columns the records should be sorted.

On the other hand, the application itself can specify a sort sequence (also using multiple sort criteria) and indicate this sort sequence by filling the structure with the desired sort criteria.

- If web server-side scrolling and sorting is used, the specified sort sequence is automatically created on the web server.
- If Natural server-side scrolling and sorting is used, the application itself must provide the records in the specified sort sequence.
- With the TEXTGRIDSSS2 control, the first three specified sort criteria are automatically indicated in the column headers of the grid.
- With the ROWTABLEAREA2 control, the first specified sort criterion is automatically indicated in the column headers of the grid. If more sort criteria are to be indicated, the application should provide custom grid headers.

In `SIZE`, the application can indicate whether the delivered amount of rows represents all available data (`SIZE=0`, no Natural server-side scrolling), or whether there are more rows to come (`SIZE=`*`total-number-of-records`*, Natural server-side scrolling).

When Natural server-side scrolling is used, the application will, for instance, hold the available rows (mostly the result of a database search) in an X-array, sort this X-array as requested and deliver the requested portion of rows. However, other implementations and optimizations are possible, depending on the needs and possibilities of the application.

When the application supports the selection of grid items, the application must take care to remember the selected state of each item when the `TOPINDEX` changes.

Server-Side Scrolling and Sorting in Trees

The ROWTABLEAREA2 control can also be configured as a tree control, where each row represents a tree node. In this case, the data structure that supports server-side scrolling contains one more field, DSPINDEXFIRST.

```
1 LINESINFO
2 DSPINDEXFIRST (I4)
2 ROWCOUNT (I4)
2 SIZE (I4)
2 SORTPROPS (1:*)
3 ASCENDING (L)
3 PROPNAME (U) DYNAMIC
2 TOPINDEX (I4)
```

The need for this additional control field comes from the fact that a tree can contain hidden items.

The rows sent by the Natural application must always start with an item at level one. The additional field DSPINDEXFIRST is provided because the visible part of the tree can start at a node with a level greater than one (a subnode). In DSPINDEXFIRST, the application must indicate the index of the first visible row within the rows sent from Natural.

Example

LN	Tree Nodes	Label
1	toptext_0	lineinfo_0
2	childtext_0.0	childlineinfo_
3	childtext_0.1	childlineinfo_
4	childtext_0.2	childlineinfo_
5	childtext_0.3	childlineinfo_

The top nodes of the tree are open and the user scrolls down as shown below:

LN	Tree Nodes	Label
4	childtext_0.2	childlineinfo_
5	childtext_0.3	childlineinfo_
6	childtext_0.4	childlineinfo_
7	toptext_1	lineinfo_1
8	childtext_1.0	childlineinfo_

The Natural application is supposed to send data starting with a top node. In our example, this is the node named **toptext_0**. But the first visible child node would be **childtext_0.2**. This means that among the sent items, the first three items are hidden. The application sets the value for `DSPINDEXFIRST` to "3" when sending the data.

Events for Server-Side Scrolling and Sorting

In order to support server-side scrolling and sorting, an application must handle a number of related events properly. The events are described with the corresponding controls. Examples on how to handle the events are provided in the Natural for Ajax demos.

12 Accessibility

- Accessibility Non Responsive Pages 76
- Accessibility Responsive Pages 76

Accessibility Non Responsive Pages

The Non Responsive Layout Pages support many accessibility requirements and recommendations, such as page structures and names, labels and headings, dynamic language settings, keyboard handling, flexible color settings via CSS stylesheets.

Accessibility requires an appropriate design of the application, using suitable containers and controls, as well as a conducive page structure and coloring. The setting of correct properties, provides pages with detailed information for Screen Readers.

The NaturalAjaxDemos contain running accessibility samples with corresponding guidelines. Although all major accessibility requirements can be met with appropriate control settings, the newer Responsive Pages of Natural for Ajax provide better accessibility support for Screen Readers.

Accessibility Responsive Pages

The Responsive Layout Pages support the major accessibility requirements and recommendations, Level A and AA.

All responsive samples have seen tests with two different accessibility tools.

Please mind that even with responsive pages, accessibility requires an appropriate design of the application, using suitable containers and controls, as well as a conducive page structure and coloring. The setting of correct properties provides pages with detailed information for Screen Readers.

For details, please see the responsive samples in the NaturalAjaxDemos. The NaturalAjaxDemos also contain a Responsive Accessibility Guide.

13 Code Pages

The built-in event names in your Natural for Ajax main program (such as `nat:page.end` and `nat:browser.end`) are usually written in lower case or mixed case. The URL values in your Natural programs (in controls such as SUBCISPAGE2 and ROWTABSUBPAGES) are usually written in mixed case. If you have an environment, however, in which you are bound to a code page which only allows Latin upper-case characters, you need to set the parameter `natuppercase="true"` in the *cisconfig.xml* file. In this case, the built-in events are generated in upper case, and URLs to Natural for Ajax pages are handled correctly even if they are specified completely in upper case.

Limitations: Since browsers and URLs to web pages are usually case-sensitive, you cannot integrate all kinds of URLs into your application. For example, it is not possible to integrate an HTML page which is not a Natural for Ajax page into a Natural for Ajax workplace application using the NJX:XCIWPACCESS2 control.

Important: Set the parameter `natuppercase="true"` *before* you implement your main program with Natural for Ajax. If you set this parameter after the implemention, you will have to change all Latin lower-case characters to upper-case manually.

The following shows an implementation of the sample program `CTRSUB-P` from the Natural for Ajax demos which runs with `natuppercase="true"`.

Tip: You often need to use an ampersand (&) as a separator between the parameters in a URL. The Hebrew code page CP803 does not support the character "&". Therefore, you need to specify the ampersand in your Natural code as a Unicode character, as shown below.

```
DEFINE DATA LOCAL
  1 ARTICLE (U) DYNAMIC
  1 INNERPAGE
    2 CHANGEINDEX (I4)
    2 PAGE (U) DYNAMIC
    2 PAGEID (U) DYNAMIC
  1 MYCONTEXT
    2 SELECTEDARTICLE (U) DYNAMIC
  1 MYTITLEPROP (U) DYNAMIC
END-DEFINE
*
INNERPAGE.CHANGEINDEX := 0
*
COMPRESS '/CISNATURAL/NATLOGON.HTML'
  UH'0026' 'XCIPARAMETERS.NATSESSION=WORKPLACE'
  UH'0026' 'XCIPARAMETERS.NATPARAMEXT=STACK%3D%28LOGON+SYSEXNJX%3BCTRSBI-P%29'
  TO INNERPAGE.PAGE LEAVING NO
INNERPAGE.PAGEID := 'MYID'
INNERPAGE.CHANGEINDEX := INNERPAGE.CHANGEINDEX+1
*
PROCESS PAGE USING "CTRSUB-A"
*
DECIDE ON FIRST *PAGE-EVENT
  VALUE U'NAT:PAGE.END', U'NAT:BROWSER.END'
    IGNORE
  VALUE U'SHOWDETAILS'
    MYCONTEXT.SELECTEDARTICLE := ARTICLE
    INNERPAGE.CHANGEINDEX := INNERPAGE.CHANGEINDEX + 1
    PROCESS PAGE UPDATE FULL
  NONE VALUE
    PROCESS PAGE UPDATE
END-DECIDE
*
END
```

14 Test Automation of Natural for Ajax Applications

- General Information 80
- Enabling the Applications for Test Automation 80
- Advanced testtoolid Settings in Complex Controls 83

General Information

Natural for Ajax is based on running HTML pages in a browser. These pages are designed as XML page layouts.

Test automation tools like Selenium (see ***http://docs.seleniumhq.org/***) need to locate specific HTML elements in an HTML page to either check or adapt the content or to trigger corresponding events. In a Selenium test program, the developer usually passes identifiers using the Selenium Java API which enable Selenium to locate the elements for testing.

For stable automated tests, it is extremely important to use stable identifiers. For instance, rearranging controls in a layout or adding an additional control must not change the identifiers. For the most common controls, Natural for Ajax automatically generates stable identifiers, the so-called "test tool IDs". They are generated as `data-testtoolid` attributes into the HTML page. Test tools like Selenium can use this `data-testtoolid` attribute to locate the element.

The following gives a brief introduction for using stable identifiers in Natural for Ajax applications.

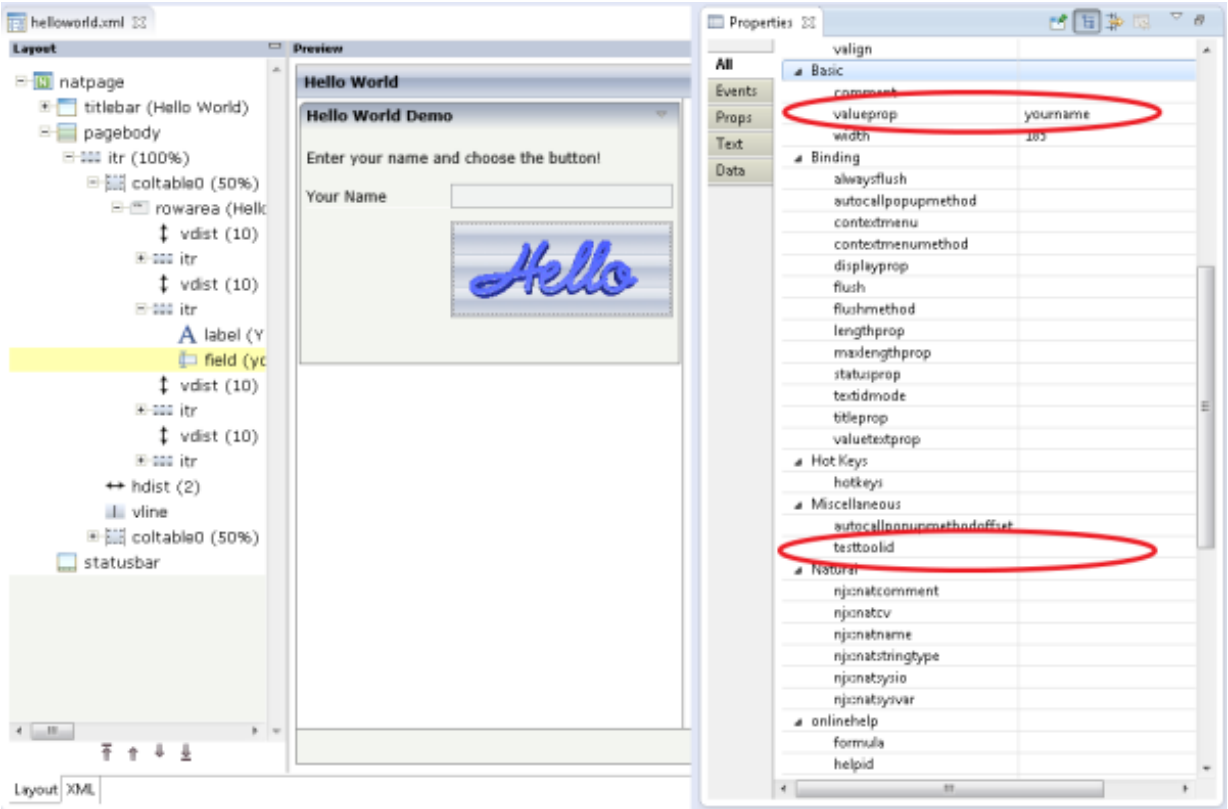
Enabling the Applications for Test Automation

All Natural for Ajax applications automatically generate stable identifiers for the most common controls. So a developer need not do anything to set them.

Let us have a look at the *helloworld.xml* page layout of the njxdemos. The most interesting controls for automated tests are the FIELD and BUTTON controls.

FIELD Control

In the following example, you see that the `valueprop` property is set in the FIELD control, but the `testtoolid` property is not explicitly set.



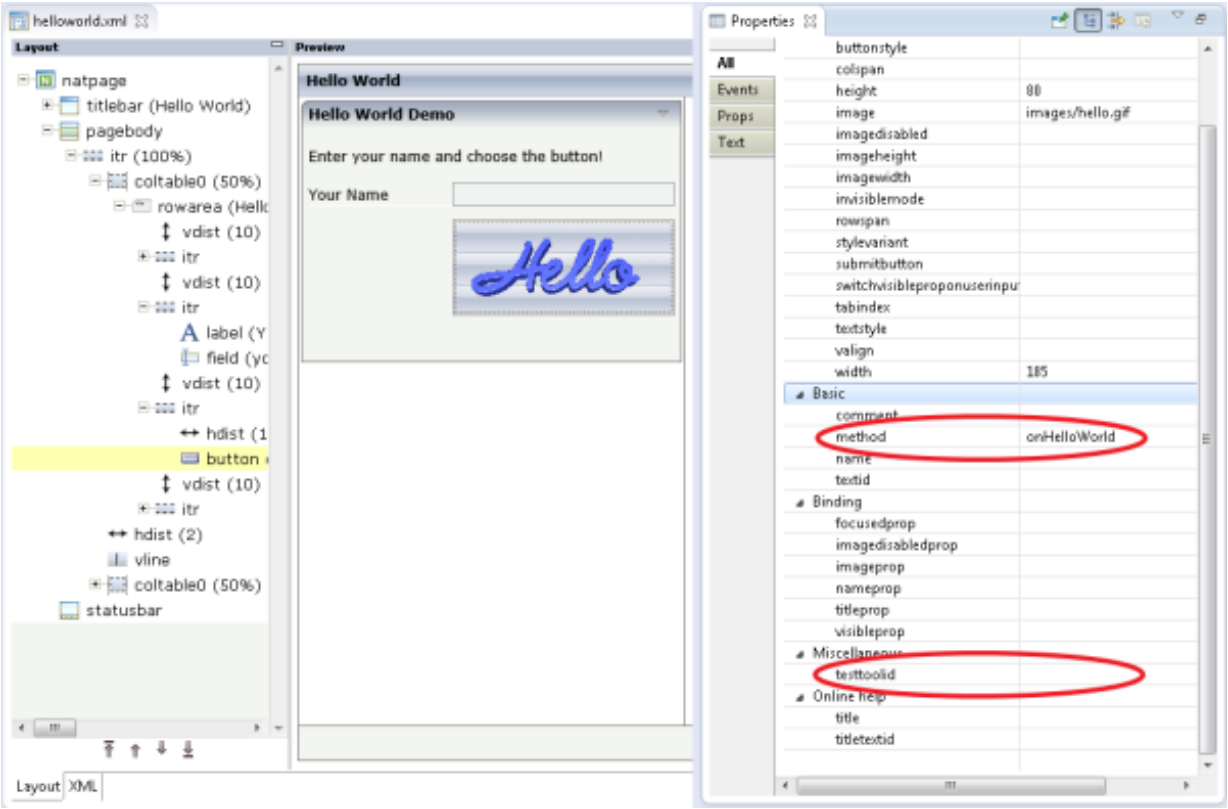
If a value for the `testtoolid` property is not explicitly set in a FIELD control, the HTML will contain a `data-testtoolid` attribute with the value of the `valueprop` property. This is shown in the HTML snippet below. You do not need to understand all the HTML details. The snippet just shows that a `data-testtoolid` attribute is automatically generated for a FIELD control; you do not have to do anything.

```
...
<input id="F_13" name="CC" class='FIELDInputEdit'
data-testtoolid='yourname' type="text" style="width: 185px;">
...
```

Caution: The above HTML code contains the `id` attribute with the value `F_13`. Do not use this in your test tool. It will break your tests sooner or later because it is not stable. For example, if you add another FIELD control in front of the `yourname` FIELD control, the `id` of the `yourname` FIELD control will change its value to `F_14`.

BUTTON Control

In the BUTTON control, the `method` property is set. Again, the `testtoolid` property is not explicitly set.



For a BUTTON control, a `data-testtoolid` attribute with the value of the `method` property is automatically generated as shown in the HTML snippet below. Again, you need not understand all the HTML details, just look at the `data-testtoolid` attribute.

```
...
<button type="button" id="B_17" data-testtoolid='onHelloWorld'
        style="width: 185px; height: 80px;" name="CC"
        class="BUTTONInput">
...
```

XPATH Expression

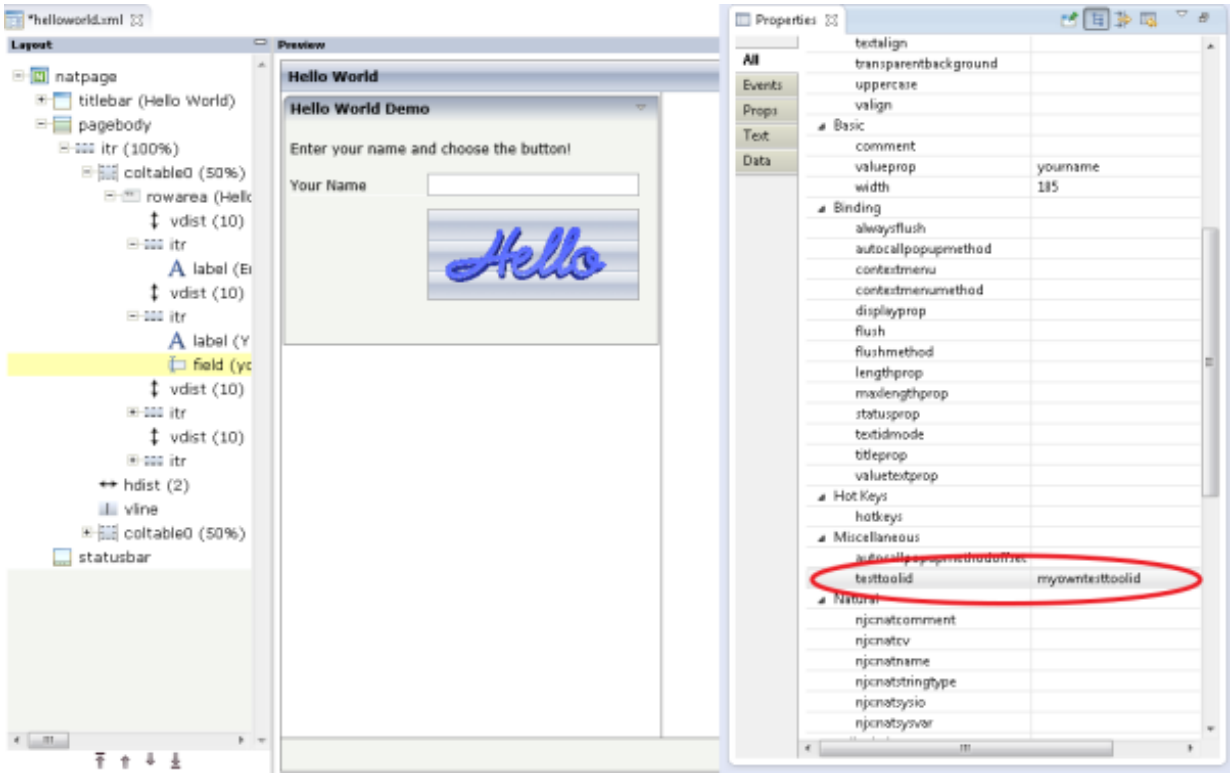
With the Selenium tool, for example, you can locate the FIELD and BUTTON controls using an XPATH expression which contains the `data-testtoolid` value. This XPATH expression can be passed to the Selenium locator `org.openqa.selenium.By.ByXPath`:

```
By myfieldlocator = new ByXPath(".//*[@data-testtoolid='yourname"]");
By mymemthodlocator = new ByXPath(".//*[@data-testtoolid='onHelloWorld"]");
```

See ***http://docs.seleniumhq.org/*** for more information about the Selenium Java API.

Explicit `testtoolid`

In some cases, you may not want to use the `valueprop` property of a control as the `testtoolid`. Instead, you want to specify your own `testtoolid`. Examples for this are layouts in which several controls are bound to the same Natural data field. You can then simply set an explicit `testtoolid` property for each of these controls.




```
...
<input id="F_13" name="CC" class='FIELDInputEdit'
testtoolid='myowntesttoolid' type="text" style="width: 185px;">
...
```

Advanced testtoolid Settings in Complex Controls

For complex controls, a single `testtoolid` is not enough to locate the individual parts of the control. The following table provides examples for the most common XPATH expressions for some complex controls.

Control	testtoolid	XPATH
ICONLIST	testtoolid="myiconlist"	.//*[@data-testtoolid='myiconlist0'], .//*[@data-testtoolid='myiconlist1'],...
BUTTONLIST	testtoolid="mybuttonlist"	.//*[@data-testtoolid='mybuttonlist0'], .//*[@data-testtoolid='mybuttonlist1'],...
ROWTABLEAREA2	testtoolid="lines"	.//*[@data-testtoolid='lines_table'] Rows/columns: .//*[@data-testtoolid=’lines.items[0].<col1testtooli .//*[@data-testtoolid=’lines.items[0].<col2testtooli .//*[@data-testtoolid=’lines.items[1].<col1testtooli .//*[@data-testtoolid=’lines.items[1].<col2testtooli
ROWTABSUBPAGES	testtoolid="mytabs"	.//*[@data-testtoolid='mytabs0'], .//*[@data-testtoolid='mytabs1'],...
MULTISELECT	testtoolid="mychoice"	XPATH for entries: .//*[@data-testtoolid='mychoice0'], .//*[@data-testtoolid='mychoice1'],... XPATH for buttons: .//*[@data-testtoolid='mychoicebutton0'], .//*[@data-testtoolid='mychoicebutton1'], .//*[@data-testtoolid='mychoicebutton2'], .//*[@data-testtoolid='mychoicebutton3']...
BMOBILE:SIMPLEGRID	testtoolid="lines"	.//*[@data-testtoolid='lines'] Column text: .//*[@data-testtoolid='lines'] 1.row/1.column //*[@data-testtoolid='lines']//tr[1]/ 1.row/2.column.//*[@data-testtoolid='lines']//tr[1]/ 2.row/1.column .//*[@data-testtoolid='lines']//tr[2] Button to enable editing Select the row/column:

Control	testtoolid	XPATH

		`//*[@data-testtoolid='lines']//tr[1]/td[1]`
		
		Use the className ‘`SIMPLEGRIDEditbutton`’ to select th
		
		Selenium example:
		
		`driver.findElement(By.xpath(".//*[@data-testto`
		
		Input field of editable column:
		
		`.//*[@data-testtoolid='SIMPLEGRIDColinput’]`
		
		OK Button editable column:
		
		`.//*[@data-testtoolid='SIMPLEGRIDColinputok’]`
		
		Cancel Button editable column:
		
		`.//*[@data-testtoolid='SIMPLEGRIDColinputok’]`

In complex controls, you need not explicitly set the `testtoolid` property in the page layout. If you do not specify any `testtoolid`, the corresponding `*prop` properties such as `valueprop`, `griddataprop`, `iconlistprop` or `pagesprop` will be used.

Here is the table from above when not specifying a `testtoolid` explicitly:

Control	*prop	XPATH
ICONLIST	`iconlistprop="myiconlist"`	`.//*[@data-testtoolid='myiconlist0'],` `.//*[@data-testtoolid='myiconlist1'],...`
BUTTONLIST	`buttonlistprop="mybuttonlist"`	`.//*[@data-testtoolid='mybuttonlist0'],` `.//*[@data-testtoolid='mybuttonlist1'],...`

ROWTABLEAREA2	`griddataprop="lines"`	`.//*[@data-testtoolid='lines_table']`
		
		Rows/columns:
		
		`.//*[@data-testtoolid=’lines.items[0].<col`
		`.//*[@data-testtoolid=’lines.items[0].<col`
		
		`.//*[@data-testtoolid=’lines.items[1].<col`
		`.//*[@data-testtoolid=’lines.items[1].<col`

Control	*prop	XPATH
ROWTABSUBPAGES	`pagesprop="mytabs"`	`.//*[@data-testtoolid='mytabs0'],` `.//*[@data-testtoolid='mytabs1'],...`

MULTISELECT	`valueprop="mychoice"`	XPATH for entries:
		`.//*[@data-testtoolid='mychoice0'],`
		`.//*[@data-testtoolid='mychoice1'],...`
		XPATH for buttons:
		`.//*[@data-testtoolid='mychoicebutton0'],`
		`.//*[@data-testtoolid='mychoicebutton1'],`
		`.//*[@data-testtoolid='mychoicebutton2'],`
		`.//*[@data-testtoolid='mychoicebutton3']...`

BMOBILE:SIMPLEGRID	`gridprop="lines"`	`.//*[@data-testtoolid='lines']`
		Column text:
		`.//*[@data-testtoolid='lines']`
		`1.row/1.column //*[@data-testtoolid='lines']//tr`
		`1.row/2.column.//*[@data-testtoolid='lines']//tr`
		`2.row/1.column .//*[@data-testtoolid='lines']//t`
		Button to enable editing
		Select the row/column:
		`//*[@data-testtoolid='lines']//tr[1]/td[1]`
		Use the className ‘`SIMPLEGRIDEditbutton`’ to select the b
		Selenium example:
		`driver.findElement(By.xpath(".//*[@data-testtool`
		Input field of editable column:
		`.//*[@data-testtoolid='SIMPLEGRIDColinput’]`
		OK Button editable column:

Control	*prop	XPATH
		`.//*[@data-testtoolid='SIMPLEGRIDColinputc` Cancel Button editable column: `.//*[@data-testtoolid='SIMPLEGRIDColinputc`