# Natural for Ajax

## Natural Pages Development

Version 9.3.2

February 2025

ADABAS & NATURAL

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

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



**Document ID: NJX-NATNJX-DEVELOPMENT-932-20250213**

# Table of Contents

# Natural Pages Development

| | |
|---|---|
| **Developing the User Interface** | How to develop the user interface using Application Designer. |
| **Developing the Application Code** | How to develop the application code using Natural Studio. |
| **Deploying the Application** | How to unload and install the Natural modules and user interface components. |
| **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

| 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:<br><br>Variables for which you must supply values specific to your own situation or environment.<br>New terms the first time they occur in the text.<br>References to other documentation sources. |
| `Monospace font` | Identifies:<br><br>Text you must type in.<br>Messages displayed by the system.<br>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://techcommunity.softwareag.com**. From here you can, for example:

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

**Product Support**

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

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

## Data Protection

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

# 2 Developing the User Interface

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 Application Designer 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. Use Natural Studio to 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.

For detailed information on how to use the Application Designer development workplace, see *Development Workplace* in the Application Designer documentation. The latest version of the Application Designer documentation is available at ***https://documentation.softwareag.com/webmethods/application_designer.htm***. The information which is provided below describes the most important differences which pertain to Natural for Ajax.

## Starting the Development Workplace

The Application Designer development workplace is the central point for starting tools for layout development.

### To start the development workplace

1 Make sure that your application server is running.

2 Invoke your browser and enter the following URL:

```
http://<host>:<port>/cisnatural/HTMLBasedGUI/workplace/ide.html
```

where *<host>* is the name of the machine on which your application server is installed and *<port>* is the port number of your application server.

**Note:** If you have not defined another port number during installation, the default port number is "8080".

## Creating an Application Designer Project

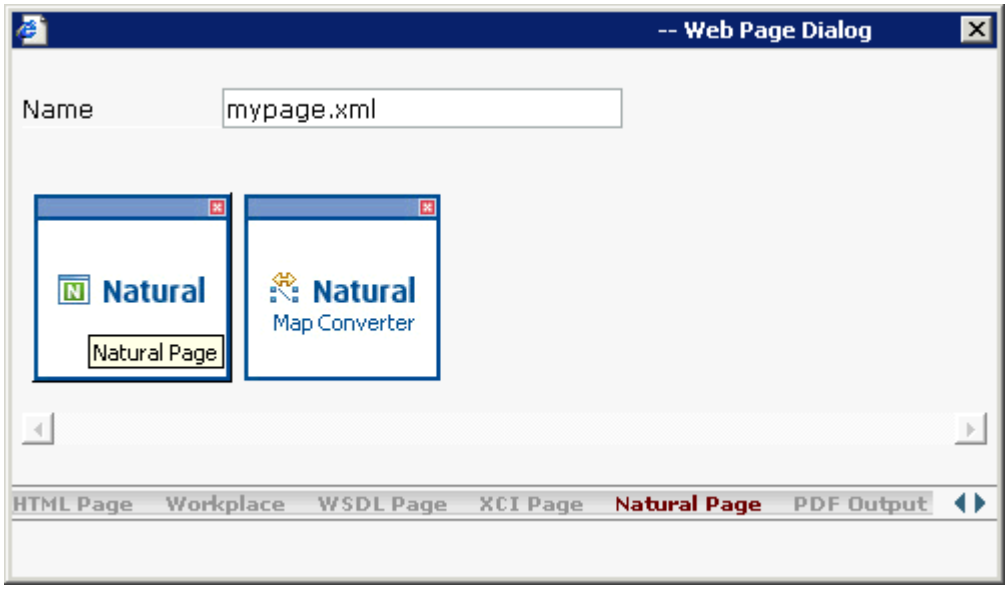
First you create an Application Designer project using the Project Manager. The project contains the layouts of the web pages you design, the files that are generated from the layouts and are required to run your application and additional files that make your application multi language capable and supply help information. See also *Creating a Project* in the tutorial.

All files in your Application Designer project are stored in one directory on the application server where Natural for Ajax is installed. The name of the directory corresponds to the project name you have chosen. The location of the directory depends on the application server.

## Creating a Natural Page

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

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

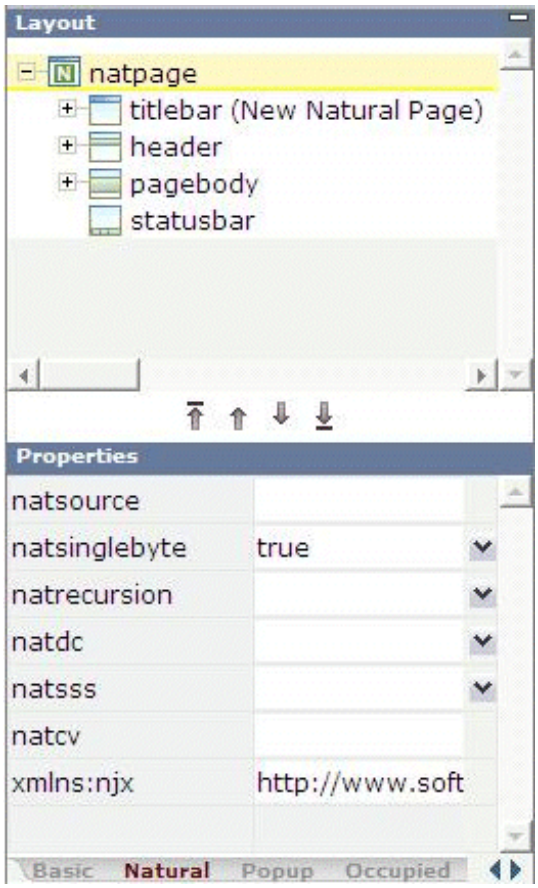



**Note:** More detailed information on creating a layout is provided in the Application Designer documentation at *https://documentation.softwareag.com/webmethods/application_designer.htm*.

## 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 area at the bottom. When you select the **Natural** tab in the properties area, you can see the Natural-specific properties.




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 HTML preview. This has already been explained in the section *Writing the GUI Layout* of the tutorial.

**Note:** More detailed information on defining the layout is provided in the Application Designer documentation at ***https://documentation.softwareag.com/webmethods/application_designer.htm***.

## 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 open the Event Editor as described in the Application Designer documentation at ***https://documentation.softwareag.com/webmethods/application_designer.htm***.

The Event Editor displays only those properties of controls that can be bound to application variables and events. It indicates also which properties are mandatory and must be bound. The usage and meaning of the properties and events is described for each control in *Natural Pages Development*,

As an example for property and event binding, see the following sections in the *First Steps* tutorial:

- *Using the Property Editor*
- *Specifying a Name and Method for the Button*

## Previewing the Layout

To find out how the current layout definitions are rendered on the page, preview the layout as described in the Application Designer documentation at ***https://documentation.softwareag.com/webmethods/application_designer.htm***.

## Viewing the Protocol

The protocol contains warnings and error messages that might occur while you design and preview your page. For further information, see the Application Designer documentation at ***https://documentation.softwareag.com/webmethods/application_designer.htm***.

## Saving the Layout

Save the page layout as described in *Saving Your Layout* in the tutorial.

Other than with Java adapters (which are described in the Application Designer documentation), you do not use the Code Assistant (which is part of the Layout Painter) to generate adapter code interactively. For Natural pages the adapter code is generated completely from the page properties and the property and event bindings that you specified previously. An adapter is generated automatically when you save the layout for the first time. It 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 subdirectory *nat* of your project directory.<br><br>The name of the project directory corresponds to the project name. The location of the directory depends on the application server. See ***Creating an Application Designer Project***. |
| **Name** | The name of the adapter is determined by the properties you have set. See ***Specifying Properties for the Natural Page***. |
| **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.<br><br>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.<br>For these events, no handler skeleton can be generated. The control reference contains information about these additional events.<br>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 following problems can be configured:

- 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 in all browsers or no longer supported in the current browsers at all. | ACTIVEX control. |

### 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.` |

**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

Natural for Ajax Tools, which is an optional plug-in for Natural Studio, allows you to use some of the Natural for Ajax functionality which is described in this chapter directly from within Natural Studio. For further information, see *Natural for Ajax Tools* in the *Natural Studio Extensions* documentation which is provided for Natural for Windows.

## Importing the Adapter

After having generated the adapter, the next step is making it available to your Natural development project.

As described previously, the adapter code is generated into a directory in your application server environment.

The following topics are covered below:

- Importing the Adapter Using Natural Studio

### Importing the Adapter Using Natural Studio

It is assumed that your development library is located on a Natural development server and that you have mapped this development server in Natural Studio.

**To import the adapter from a remote environment**

- Use drag-and-drop.

  Or:

  Remote Linux environment only: Use the import function of `SYSMAIN`.

## Creating the Main Program

After you have imported the adapter, you create a program that calls the adapter to display the page and handles the events that the user raises on the page. This program can be a Natural program, subprogram, subroutine or function. We use a Natural program as example.

The adapter already contains the data structure that is required to fill the page. It contains also a skeleton with the necessary event handlers. You can therefore create a program with event handlers from an adapter in a few steps.

Open or list the adapter in Natural Studio.

```
* PAGE1: PROTOTYPE       --- CREATED BY Application Designer ---
* PROCESS PAGE USING 'XXXXXXXX' WITH
* FIELD1 FIELD2
DEFINE DATA PARAMETER
1 FIELD1 (U) DYNAMIC
1 FIELD2 (U) DYNAMIC
END-DEFINE
*
PROCESS PAGE U'/MyProject/mypage' WITH
PARAMETERS
 NAME U'field1'
  VALUE FIELD1
 NAME U'field2'
  VALUE FIELD2
END-PARAMETERS
*
*  TODO: Copy to your calling program and implement.
/*/*( DEFINE EVENT HANDLER
* DECIDE ON FIRST *PAGE-EVENT
*  VALUE U'nat:page.end',U'nat:browser.end'
*   /* Page closed.
*   IGNORE
*  VALUE U'onExit'
*   /* TODO: Implement event code.
*   PROCESS PAGE UPDATE FULL
*  NONE VALUE
*   /* Unhandled events.
*   PROCESS PAGE UPDATE
* END-DECIDE
/*/*) END-HANDLER
*
END
```

Create a new program, copy the adapter source into the program and then proceed as follows:

- Remove the comment lines in the header.
- Change `DEFINE DATA PARAMETER` into `DEFINE DATA LOCAL`.
- Replace the `PROCESS PAGE` statement with a `PROCESS PAGE USING` *operand4* statement, where *operand4* stands for the name of your adapter.
- Remove the comment lines that surround the `DECIDE` block.
- Uncomment the `DECIDE` block.

Your program should now look as follows:

```
DEFINE DATA LOCAL
1 FIELD1 (U) DYNAMIC
1 FIELD2 (U) DYNAMIC
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'onExit'
  /* TODO: Implement event code.
  PROCESS PAGE UPDATE FULL
 NONE VALUE
  /* Unhandled events.
  PROCESS PAGE UPDATE
END-DECIDE
*
END
```

Stow the program with a name of your choice. 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.

## 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
```

## 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
```

## Starting a Natural Application from the Logon Page

See *Starting a Natural Application from the Logon Page* in the *Configuration and Administration* documentation.

## Starting a Natural Application with a URL

See *Starting a Natural Application with a URL* and *Wrapping a Natural for Ajax Application as a Servlet* in the *Client Configuration* documentation.

# 4 Deploying the Application

## Components of a Natural for Ajax Application

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 related files that are installed in a Natural for Ajax environment on an application server.

## Unloading the Natural Modules

The Natural modules that belong to your application are contained in one or several Natural libraries in your Natural development environment. Unload them into a file, using the Object Handler.

## Installing the Natural Modules

In order to install the Natural modules in the production environment, load them with the Object Handler.

## Packaging the User Interface Components

Your web application might contain one or more user interface components.

In production environments it is deeply recommended to always deploy/refresh the whole web application for consistency.

In development or test environments you sometimes might want to deploy single user interface components into an already deployed web application. To deploy global files such as custom controls, which are used by several user interface components or configuration files like the *ciscon-fig.xml* file (which is used for the whole web application), you have to use web application deployment/refreshment as described in the following sections.

User interface components are stored in subdirectories of your web application.

You only need to package those files of your user interface component which are not generation results. All files which are generation results will be generated by the Natural for Ajax runtime

during deployment. If you also package files which are generation results, the Natural for Ajax runtime system will ignore these files.

If you are using NaturalONE, use the Ant war deployment wizard to create an Ant file which will package your user interface component(s). To package a user interface component for deployment without NaturalONE, add all files and subdirectories to an archive using an archiving tool like WinZip or tar. Do not include the following files and folders:

| File | Description |
|---|---|
| *<use interfacedir>/*.html* | Generated HTML pages. |
| *<use interfacedir>/wsdl/*.** | Generated data schemas. |
| *<use interfacedir>/nat* | Generated Natural code |
| *<use interfacedir>/protocol* | Generated protocol files |
| *<use interfacedir>/styles/<mystylesheet>.css* | Style sheet files that are generated from a Natural for Ajax **.info* file |

Provide a unique name for the created zip file. This can for example be done by appending date or timestamp to the file name. Example: *<myui>20170501.zip*.

## Deploying the User Interface Components

In order to deploy the user interface components, simply copy the zip file which you created as described previously into the *_uiupdates* folder of your web application, for example: *<tomcatfolder>/webapps/<mywebapp>/_uiupdates/<myui>20170501.zip*.

By default the Natural Ajax runtime system will pick up the file every 5 seconds. This value can be customized with the `monitoringthreadinterval` parameter. It will deploy it and refresh all internal caches of the Natural Ajax runtime system. For the example above, deployment and refreshing is finished when a file with the name *<tomcatfolder>/webapps/<mywebapp>/_uiupdates/<myui>20170501/update.result* exists. It is important to check the *update.result* file for errors: Open the file and look for "Update finished Successfully". If you cannot find this, check the **.protocol* files in the *protocol* sub folder for errors and/or exceptions.

In case you cannot solve the generation problems via the Layout Painter error marking, you can switch on the creation of additional log files. See the `htmlgeneratorlog` attribute in the *Ajax Configuration* section.

## Packaging and Deployment as a Web Application

Natural for Ajax is delivered as a web application (*.war* file). This allows for packaging and deploying also your own applications (more exactly: the user interface components thereof) as self-contained web applications. The preferred way to create a **.war* for your application is to use the Ant war deployment wizard of NaturalONE.

If you are not using NaturalONE:

### To package your application as a web application

1 Invoke the Application Designer development workplace.

2 In the **Development Tools** node of the navigation frame (which is visible when the **Tools & Documentation** button has previously been chosen), choose **WAR Packager**.

3 In the resulting dialog, make sure that the **Deployment Scenario** tab is selected.

4 Define the generation type by selecting one of the following option buttons: **with file system reference** or **fully clusterable**. See the Application Designer documentation for detailed information on these generation types.

**Note:** The option **fully clusterable** applies only for web applications written in Java, not for those written in Natural. This is because a Natural-written application runs on a Natural server and therefore needs to keep a TCP/IP connection to the server, while Java applications are executed on the web container itself.

5 If you selected with **file system reference**, enable the **Switch off Design Time** check box.

6 Select the **Project Selection** tab.

7 Select the project directories that you want to include in your web archive. These must be at least the following:

| Directory | Description |
|---|---|
| *cis* | Application Designer configuration files. |
| *cisnatural* | Natural for Ajax logon page and related pages. |
| *HTMLBasedGUI* | HTML user interface. |
| *images* | Application Designer image files. |
| *META-INF* | Standard directory in a web application. |
| *resources* | Natural Web I/O Interface style sheets and related files. |
| *scripts* | Natural Web I/O Interface JavaScript files. |
| *WEB-INF* | Standard directory in a web application. |

In addition, you have to select your own project directories.

8 In the text box **WAR File to be created**, specify a path and name for the web application to be created.

9 Choose the **Create WAR** button.

The web application (*.war* file) is created.

### To deploy your application

1 You deploy your web application in the same way as you deployed Natural for Ajax itself (see *Installation*).

2 After you have deployed your web application, you can use the configuration tool to specify the configuration for this specific application. For further information, see *Natural Client Configuration Tool*.

Start the configuration tool with the following URL:

```
http://<host>:<port>/<webcontext>/conf_index.jsp
```

The logon page of the application can be found here:

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

**Note:** `<webcontext>` denotes the web context of your application. On Apache Tomcat, this is the name of the *.war* file, without the extension *.war*. On IBM WebSphere, this is the value you specified as the web context during the deployment.

## Generating HTML Pages Using the Command Line

You can generate HTML pages using the command line (either single pages or entire projects). If you do this, you have to reload your web application afterwards or - in a development environment - use the monitoring tool to refresh the internal caches.

An Ant file named *generate.xml* is available for this purpose. After the installation, you can find it in the *support/ant* directory.

The Ant file has the following major targets:

- `info`
  Shows the syntax of this Ant task.

- `project`
  Generates all HTML pages for a given project.
- `page`
  Generates a single HTML file for a given page in a project.
- `pages`
  Generates HTML files for given pages in a project.
- `style`
  Generates CSS files for info files in a project

The following call explains the targets with their mandatory and optional parameters:

```
ant -f generate.xml info
```

By default, the following log files are written during the HTML generation:

- <layout>.protocol files
- HTMLGeneratorWholeDirectory.log
- <layout>.log files

### <layout>.protocol files

For a layout named `mylayout.xml` a file named *mylayout.protocol* is created in the *protocol* subdirectory of the user interface component. It contains the error messages, warning messages and information messages for the controls in the layout. Open the layout with the Layout Painter Editor to position at the erroneous controls.

**Example**

```
<?xml version="1.0" encoding="utf-8"?>
<pro:protocol xmlns:pro="http://www.softwareag.com/cis/protocol">
<pro:lineitem>
 <pro:id>6</pro:id>
 <pro:tag>field</pro:tag>
 <pro:message>
  <pro:severity>Error</pro:severity>
  <pro:mtext>Property VALUEPROP is not set</pro:mtext>
 </pro:message>
 <pro:message>
  <pro:severity>Warning</pro:severity>
  <pro:mtext>One of the properties LENGTH or WIDTH should be set</pro:mtext>
 </pro:message>
</pro:lineitem>
<pro:summary errors='1' warnings='1' infos='0' ></pro:summary>
</pro:protocol>
```

### HTMLGeneratorWholeDirectory.log

When generating all layouts of a user interface component, additionally a file named *HTMLGeneratorWholeDirectory.log* is created in the *log* subdirectory of the user interface component. It contains the names of the generated layouts and the information, whether an error occurred.

**Example**

```
...
Starting generation of wpworkplacelan1.xml...
...finished
Starting generation of wpworkplacelan2.xml...
...finished
Starting generation of xmldatamode.xml...
...finished
Starting generation of xmldatamode2.xml...
...finished
========================================
 227 layouts generated
 1 layouts with ERROR
```

### <layout>.log files

In addition to the above, you can activate the creation of an individual *.log* file for each layout. This file contains generation details.

To switch on the log file creation, set `htmlgeneratorlog="true"` in the *cisconfig.xml* file. Only activate this option, if you need to analyze generation problems, as it reduces generation performance. The additional *.log* files are created in the *log* subdirectory of the user interface component. The `htmlgeneratorlog` option is described in *General cisconfig.xml Parameters* of the *Ajax Configuration* section.

# 5 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***.

# 6 Usage of Edit Masks

## 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.

# 7 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 (see *Starting a Natural Application from the 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 Client Configuration Tool* in the *Client Configuration* documentation, .

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*.

# 8 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. |

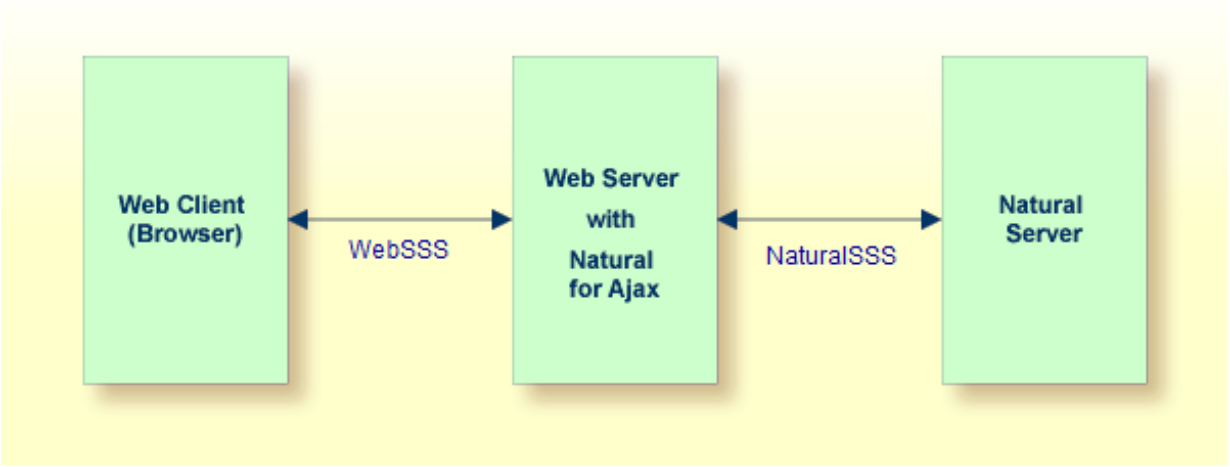
# 9 Server-Side Scrolling and Sorting

## 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.

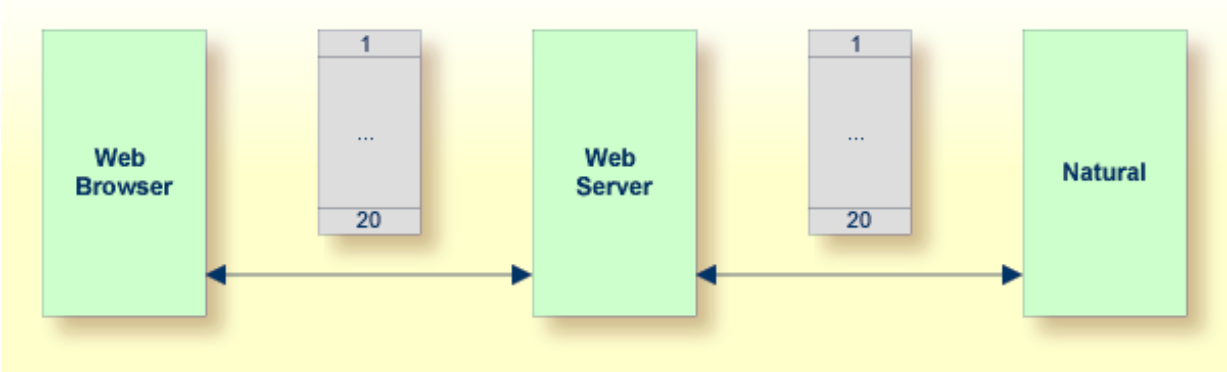
The decision between these options will often depend on the expected data volume. The application can decide dynamically at runtime which option to use.

The following topics show the difference between these three options

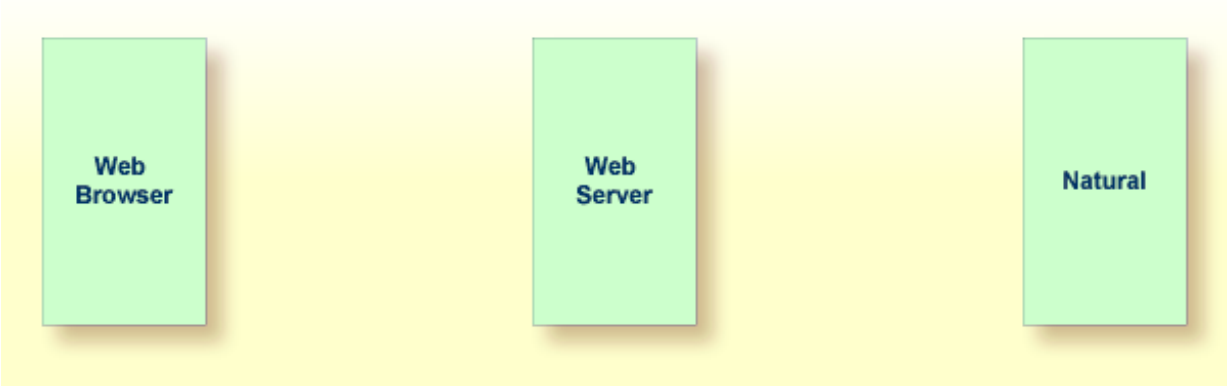
- No Server-Side Scrolling and Sorting
- Web Server-Side Scrolling and Sorting
- Natural Server-Side Scrolling and Sorting

### 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`).

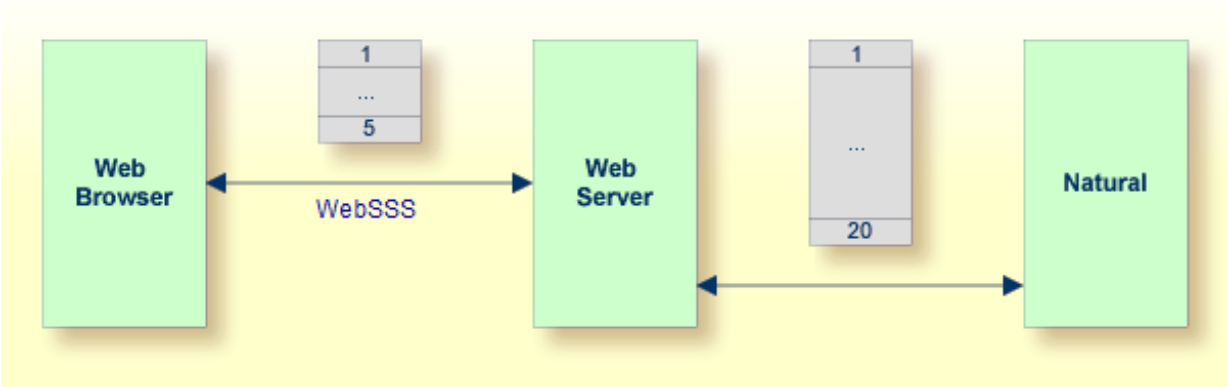


Step 2: When you scroll up and down, no server round trips to the web server or to the Natural application are performed.

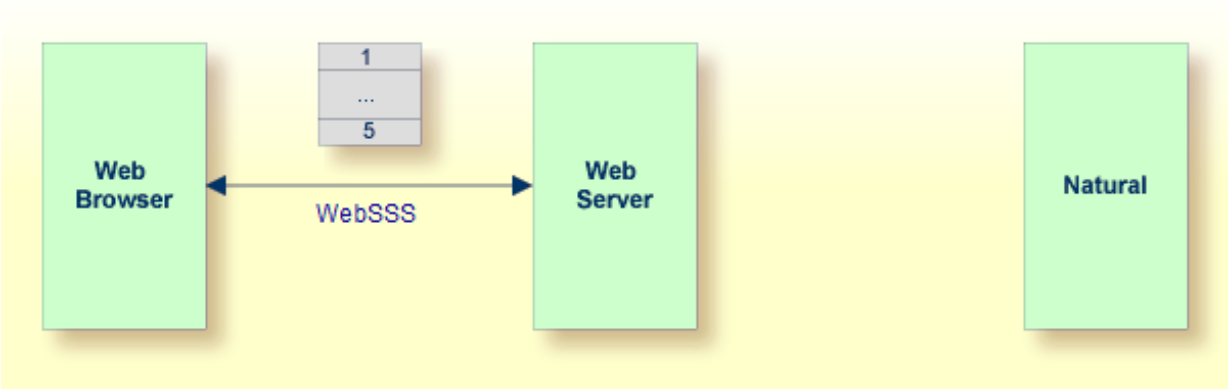



### Web Server-Side Scrolling and Sorting

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`).

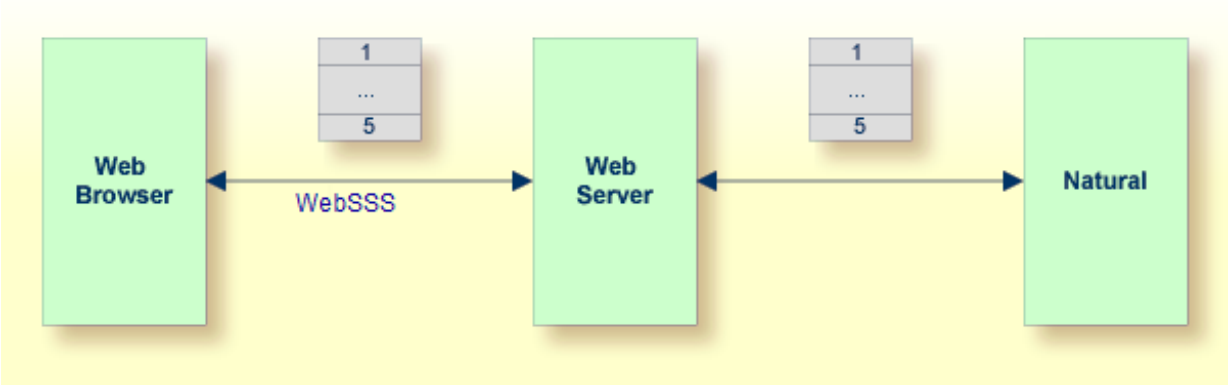



Step 2: When you scroll up and down, the web browser requests additional records from the web server There are no server round trips to Natural.

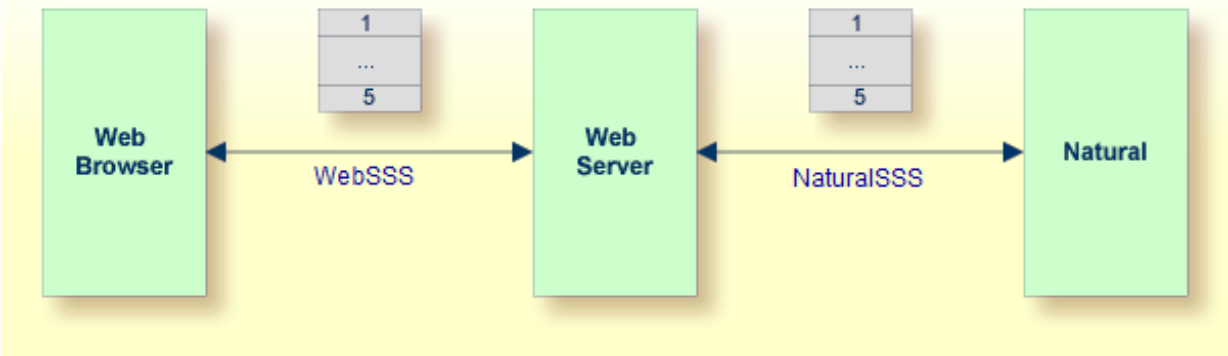



### Natural Server-Side Scrolling and Sorting

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**


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


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.

# 10 Accessibility

## 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.

# 11 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
```

# 12 Test Automation of Natural for Ajax Applications

## 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.

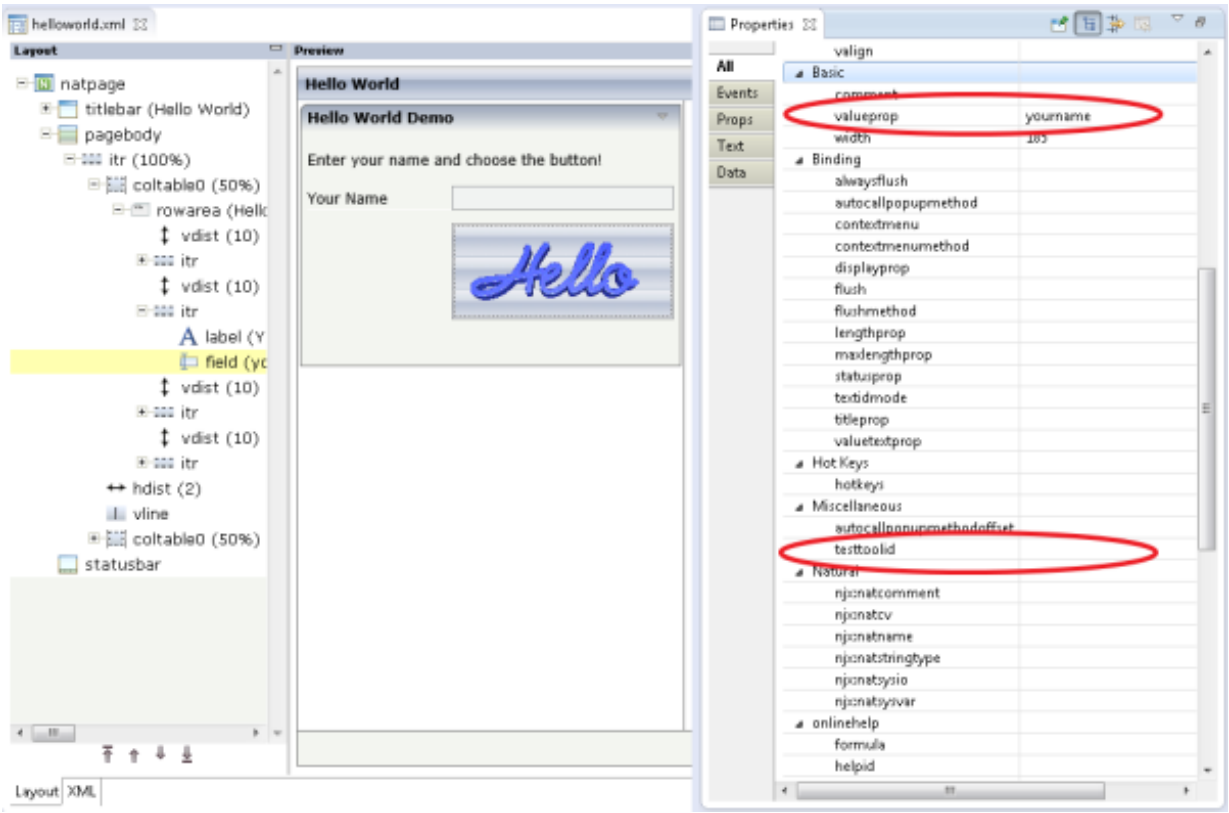
## Enabling the Applications for Test Automation

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.



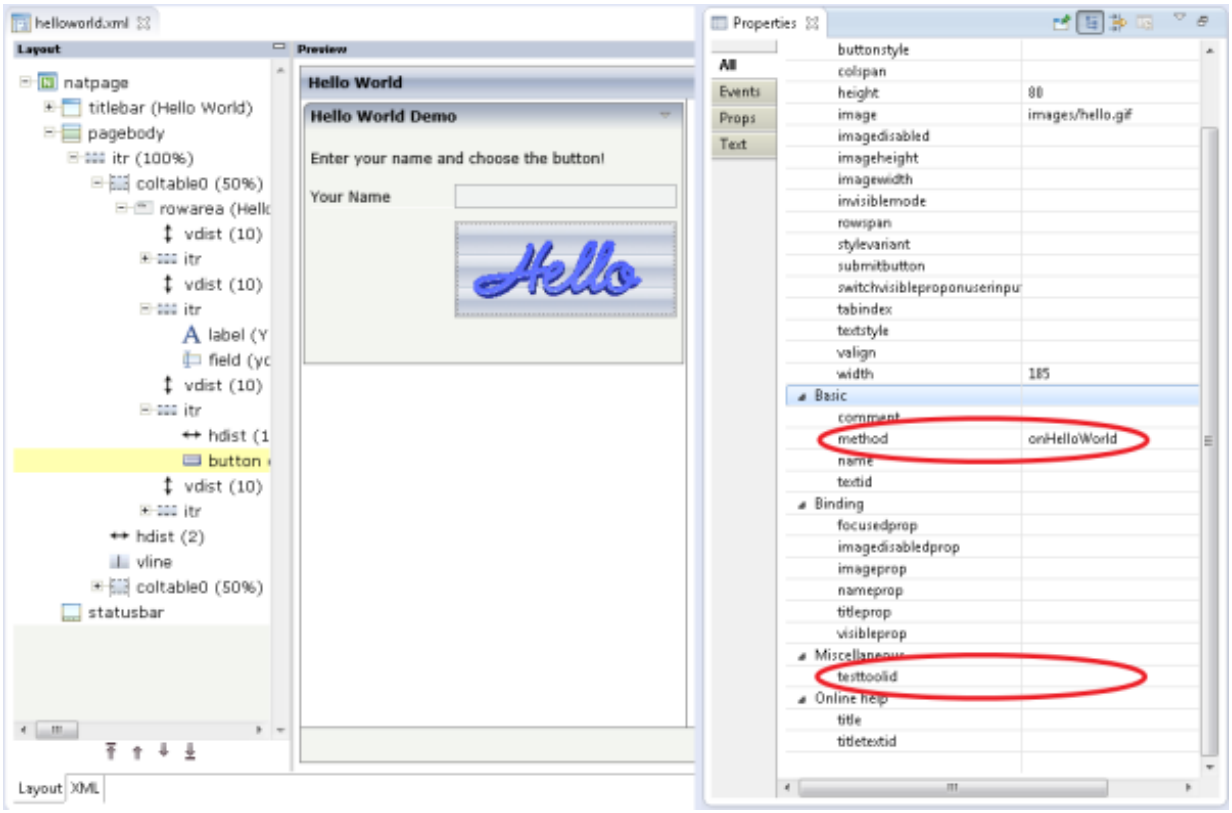

If a value for the `testtoolid` property is not explicitly set in a FIELD control, the HTML will contain a `data-testtoolid` attribute with the value of the `valueprop` property. This is shown in the HTML snippet below. You do not need to understand all the HTML details. The snippet just shows that a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

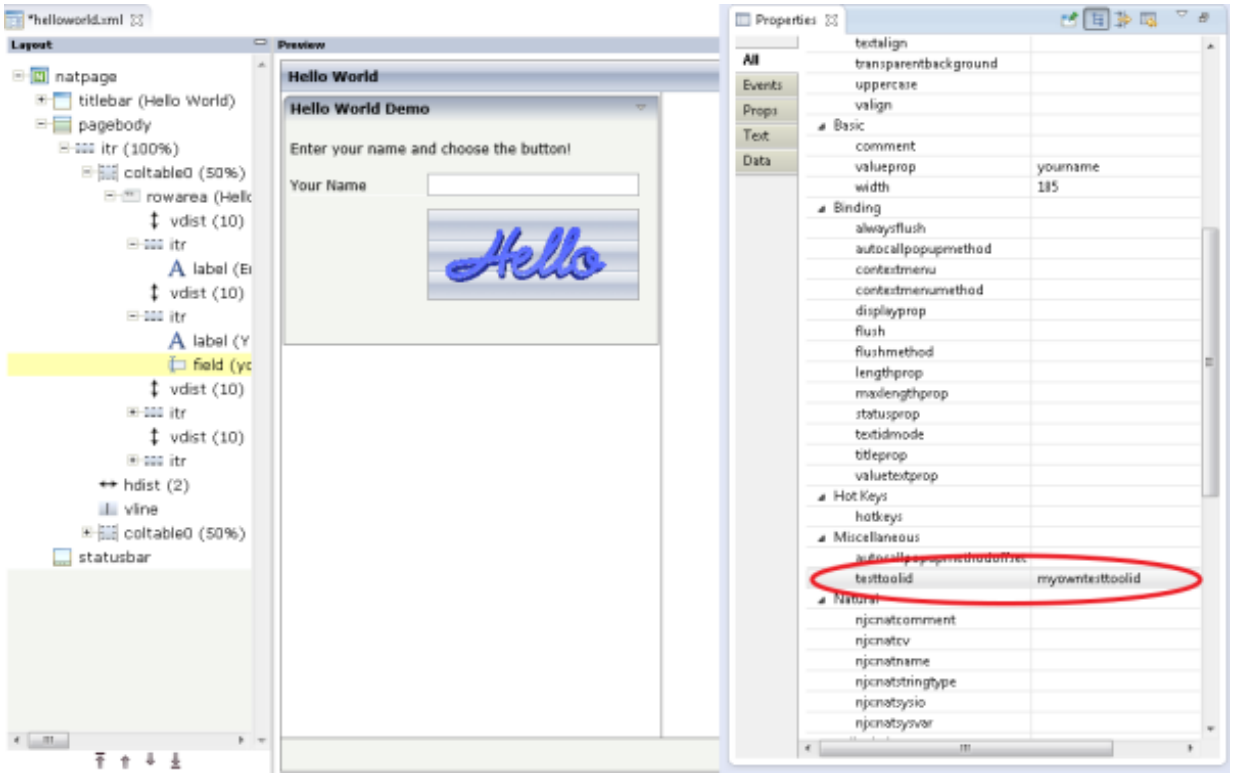
With the Selenium tool, for example, you can locate the FIELD and BUTTON controls using an XPATH expression which contains the `data-testtoolid` value. This XPATH expression can be passed to the Selenium locator `org.openqa.selenium.By.ByXPath`:

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

| Control | testtoolid | XPATH |
|---|---|---|
| | | //*[@data-testtoolid='lines']//tr[1]/td[1]<br>Use the className 'SIMPLEGRIDEditbutton' to select th<br>Selenium example:<br>driver.findElement(By.xpath(".//*[@data-testto<br>Input field of editable column:<br>.//*[@data-testtoolid='SIMPLEGRIDColinput']<br>OK Button editable column:<br>.//*[@data-testtoolid='SIMPLEGRIDColinputok']<br>Cancel Button editable column:<br>.//*[@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'],<br>.//*[@data-testtoolid='myiconlist1'],... |
| BUTTONLIST | buttonlistprop="mybuttonlist" | .//*[@data-testtoolid='mybuttonlist0'],<br>.//*[@data-testtoolid='mybuttonlist1'],... |
| ROWTABLEAREA2 | griddataprop="lines" | .//*[@data-testtoolid='lines_table']<br>Rows/columns:<br>.//*[@data-testtoolid='lines.items[0].<col<br>.//*[@data-testtoolid='lines.items[0].<col<br>.//*[@data-testtoolid='lines.items[1].<col<br>.//*[@data-testtoolid='lines.items[1].<col |

| Control | *prop | XPATH |
|---|---|---|
| ROWTABSUBPAGES | `pagesprop="mytabs"` | `.//*[@data-testtoolid='mytabs0'], .//*[@data-testtoolid='mytabs1'],...` |
| MULTISELECT | `valueprop="mychoice"` | XPATH for entries:<br>`.//*[@data-testtoolid='mychoice0'], .//*[@data-testtoolid='mychoice1'],...`<br>XPATH for buttons:<br>`.//*[@data-testtoolid='mychoicebutton0'], .//*[@data-testtoolid='mychoicebutton1'], .//*[@data-testtoolid='mychoicebutton2'], .//*[@data-testtoolid='mychoicebutton3']...` |
| BMOBILE:SIMPLEGRID | `gridprop="lines"` | `.//*[@data-testtoolid='lines']`<br>Column text:<br>`.//*[@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`<br>Button to enable editing<br>Select the row/column:<br>`//*[@data-testtoolid='lines']//tr[1]/td[1]`<br>Use the className ‘`SIMPLEGRIDEditbutton`’ to select the b<br>Selenium example:<br>`driver.findElement(By.xpath(".//*[@data-testtool`<br>Input field of editable column:<br>`.//*[@data-testtoolid='SIMPLEGRIDColinput’]`<br>OK Button editable column: |

| Control | *prop | XPATH |
|---|---|---|
| | | .//*[@data-testtoolid='SIMPLEGRIDColinputc<br>Cancel Button editable column:<br>.//*[@data-testtoolid='SIMPLEGRIDColinputc |