Natural for Ajax

Version 9.3.1

February 2025

ADABAS & NATURAL

This document applies to Natural for Ajax Version 9.3.1 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: ONE-NATNJX-INTRO-931-20250213

Table of Contents

Preface

This documentation explains how to create rich internet applications which use the Ajax (Asynchronous JavaScript and XML) technology. It should be used together with the *Ajax Developer* documentation.

This documentation is organized under the following headings:

Using Natural for Ajax	
Introduction	What is Natural for Ajax?
First Steps	How to create a "Hello World!" application.
First Steps Responsive Pages	How to build a responsive application.
First Steps Java Pages	How to build a "Hello World" application with Natural for Ajax, based on Java.

You can install the Natural for Ajax demos from the welcome page. A collection of small Natural for Ajax samples will then be available as a new Natural project in the **Project Explorer** view. These samples demonstrate how to use the different controls and how to bind them to Natural server-side processing. Corresponding samples are available in the NaturalAjaxDemos.

Using the runtime version of Natural for Ajax, which is not part of NaturalONE, you can

- start a Natural application from the logon page or with a URL,
- manage the configuration file for the session using the configuration tool,
- modify the style sheet which controls the font, the color and the representation of the PF keys,
- activate the preconfigured security settings of Natural for Ajax and adapt them to your requirements,
- create your own trust files for a secure connection between the Natural Web I/O Interface server and Natural for Ajax,
- enable logging in the case of problems with Natural for Ajax.

For detailed information, see *Configuration and Administration* in the documentation for the standalone version of Natural for Ajax.

1 About this Documentation

Document Conventions

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

Online Information and Support

Product Documentation

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

Product Training

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

Tech Community

You can collaborate with Software GmbH experts on our Tech Community website at **https://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.

I Introduction

2 Introduction

Using Natural for Ajax, you can create rich internet applications using Ajax (Asynchronous JavaScript and XML) technology.

Natural for Ajax enables Natural users on Windows, Linux and mainframe platforms to develop and operate Natural applications with a browser-based user experience, similar to GUI desktop applications.

Adding to flexibility, Natural for Ajax also allows building browser-based user interfaces driven by Java applications or a mix of Natural and Java.

In contrast to many other tools for building browser-based applications, Natural for Ajax requires no in-depth knowledge of HTML and/or JavaScript.

Natural for Ajax can convert existing Natural maps to Natural for Ajax pages. Or, just run existing Natural maps in the browser without conversion.

What is a Rich Internet Application?

Classical HTML- and browser-based applications suffer from known disadvantages: The server's response time increases with each user interaction, rendering a new page in the browser, thus discontinuing the user's workflow. In contrast, classical GUI applications suffer from the prerequisite of installing an application's client component on each client machine.

Rich internet applications based on Ajax technology overcome such disadvantages as they combine the browser-based application's accessibility with the rich user interface of GUI applications.

Rich Internet Applications with Natural

At runtime, a rich internet application with Natural has the following structure:

- A Natural host session on a Windows, Linux, or mainframe server runs the application code. Other than with a map application, the application does not deal with user interface issues. It contains only the application logic and communicates with the user interface layer by sending and receiving data. The data displays as a page in a web browser. Events, such as button clicks raised by the user in the web browser, are passed back to the application code. The application code receives the data modified by the user in the web browser along with such an event. The event and the data are processed and returned to the web browser page.
- Natural for Ajax, running on an application server, receives data from the Natural application and merges it into an HTML page, delivered to the web browser. In the inverse direction, Natural for Ajax forwards events that the user raised in the web browser along with the modified data to the Natural application.

- A web browser renders the HTML page. JavaScript code on the page processes local user interaction and exchanges data with Natural for Ajax as needed. It uses Ajax technology to exchange data with the Natural application in the background without re-rendering the complete page.

At development time, creating a rich internet application with Natural proceeds as follows:

- Ajax Developer of NaturalONE is used to develop the user interface layout of a web page and to bind the controls on the page to data elements in the application.
- When the user saves the page layout, a Natural module of type "Adapter" is generated. The adapter serves as an interface between the application code and the page layout. It contains:
  - a data structure that describes the data that the Natural application has to deliver to the application server to populate the web page.
  - the Natural code necessary to transfer the data structure to the user interface and to receive modified data back.
  - a code skeleton in the form of comment lines that provide handlers for the expected events. The application programmer can copy this code skeleton into the main program to implement the event handlers.
- Then a main program is implemented that exchanges data with the web page using the adapter and handles the events. The event handler code is not aware of the web page's layout and operates exclusively on the page data sent and received through the adapter.
- Finally, the navigation between different pages is implemented: A rich internet application navigates between pages just as a map application would navigate between maps.

Natural Maps in Rich Internet Applications

With the support of Unicode, Natural has introduced the Natural Web I/O Interface that renders Natural maps in a web browser. Rendering is based on HTML elements. However, rich internet pages contain much more advanced HTML elements that provide a much better user experience.

When coming from map-oriented applications and planning to switch to rich internet pages, such projects happen gradually: In certain parts of an application, rich internet pages may replace maps, and other parts of an application will possibly be left unchanged. To support such gradually conducted projects, Natural can run "mixed" applications consisting of both, maps as well as rich internet pages. Seamless navigation between Natural maps and rich internet pages is supported.

II First Steps

This part is organized under the following headings:

It is important that you work through the exercises in the same sequence as they appear in this tutorial. Problems may occur if you skip an exercise.

3 About this Tutorial

This tutorial provides an introduction to working with Natural for Ajax. It explains how to create a “Hello World!” application. This covers all basic steps you have to perform when creating pages with Natural for Ajax: you create a layout file, you create an adapter and a main program, and you run the application.

When you have completed all steps of this tutorial, the page for your “Hello World!” application will look as follows:

Your application will act in the following way: When you enter a name in the **Your Name** field and choose the **Say Hello** button, the **Result** field displays "Hello World" and the name you have entered.

To reach this goal, you will proceed as follows:

1. You work with a Natural project that has been enabled for Ajax Developer.
2. You add a new user interface component to your Natural project. This is a folder which will contain all of your page layouts.
3. You will then create the following page layout:

This corresponds to the following XML layout:

```
<?xml version="1.0" encoding="UTF-8"?>
<natpage natsource="HELLO-A" natsinglebyte="true" natkcheck="true"
xmlns:njx="http://www.softwareag.com/njx/njxMapConverter">
<titlebar name="Hello World!">
</titlebar>
<header withdistance="false">
<button name="Say Hello" method="sayHello">
</button>
</header>
<pagebody>
<itr takefullwidth="true">
<hdist width="100%">
</hdist>
<icon image="./images/hello.gif">
</icon>
</itr>
<rowarea name="Input Area">
<itr>
<label name="Your Name" width="100">
</label>
<field valueprop="name" width="200">
</field>
```

```
</itr>
</rowarea>
<rowarea name="Output Area">
<itr>
<label name="Result" width="100">
</label>
<field valueprop="result" width="200" displayonly="true">
</field>
</itr>
</rowarea>
<vdist height="100">
</vdist>
<itr>
<label name="Input your name and press the &apos;Say Hello&apos; button." ↩
asplaintext="true">
</label>
</itr>
</pagebody>
<statusbar withdistance="false">
</statusbar>
</natpage>
```

4. When you save your layout for the first time, an intelligent HTML page and the Natural adapter for this page are generated.
5. You will then create the main program which will use the adapter to display the page and which will handle the events that occur on the page, for example, when you choose the **Say Hello** button of your application.

You can now proceed with your first exercise: *Enabling a Natural Project for Ajax Developer*.

4 Enabling a Natural Project for Ajax Developer

This tutorial assumes that a Natural project containing a library has already been created. The library name that is used in this tutorial is `CISHELLO`.

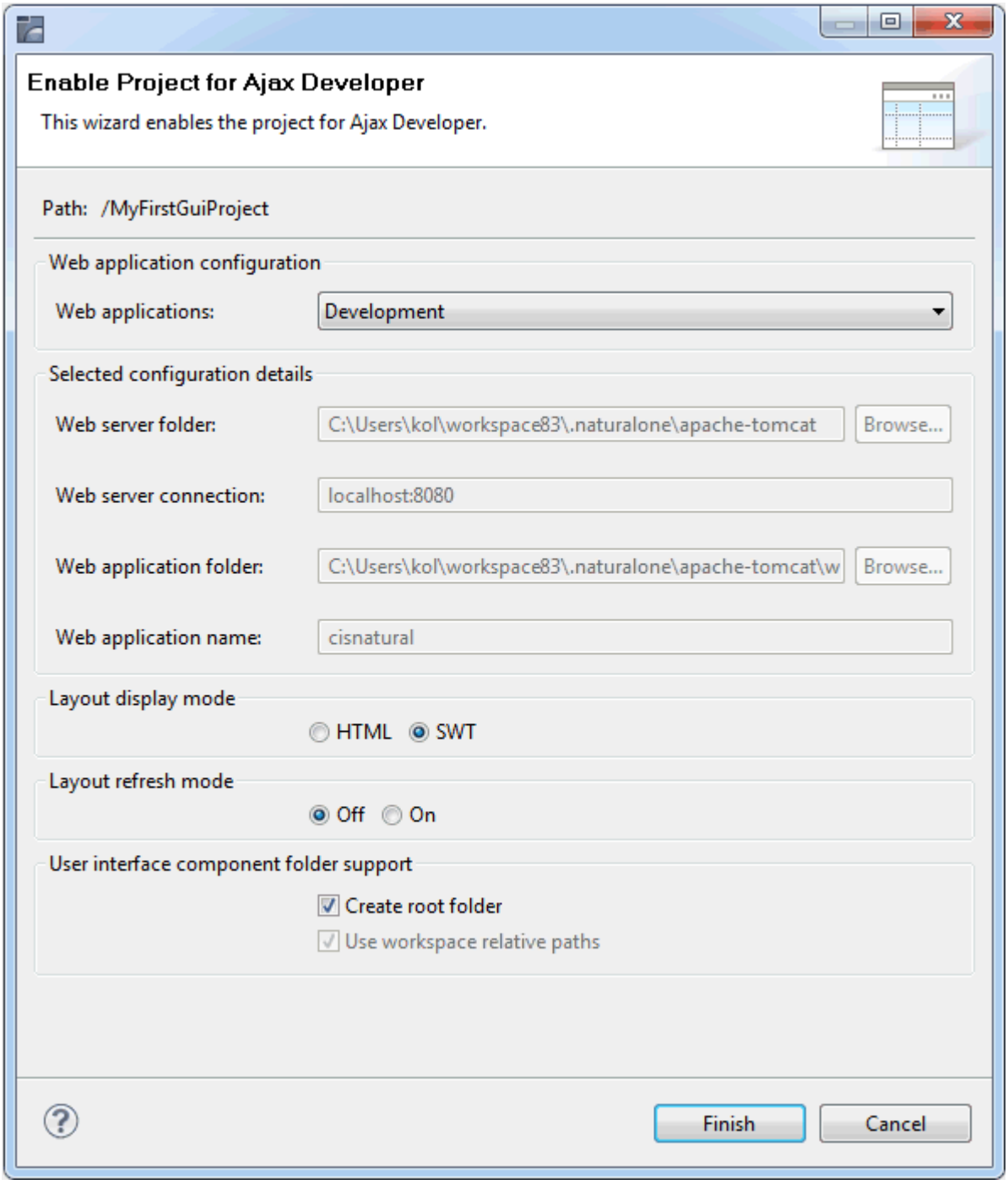
Before you can define layout pages for a Natural project, you have to enable this project for Ajax Developer.

To enable a project for Ajax Developer

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

2 Invoke the context menu and choose **Enable for Ajax Developer**.

The following dialog box appears:



3 Use the default settings, that is, leave the **Web applications** drop-down list box with the default value **Development**.

4 Choose the **Finish** button.

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

You can now proceed with the next exercise: *Creating a User Interface Component*.

5 Creating a User Interface Component

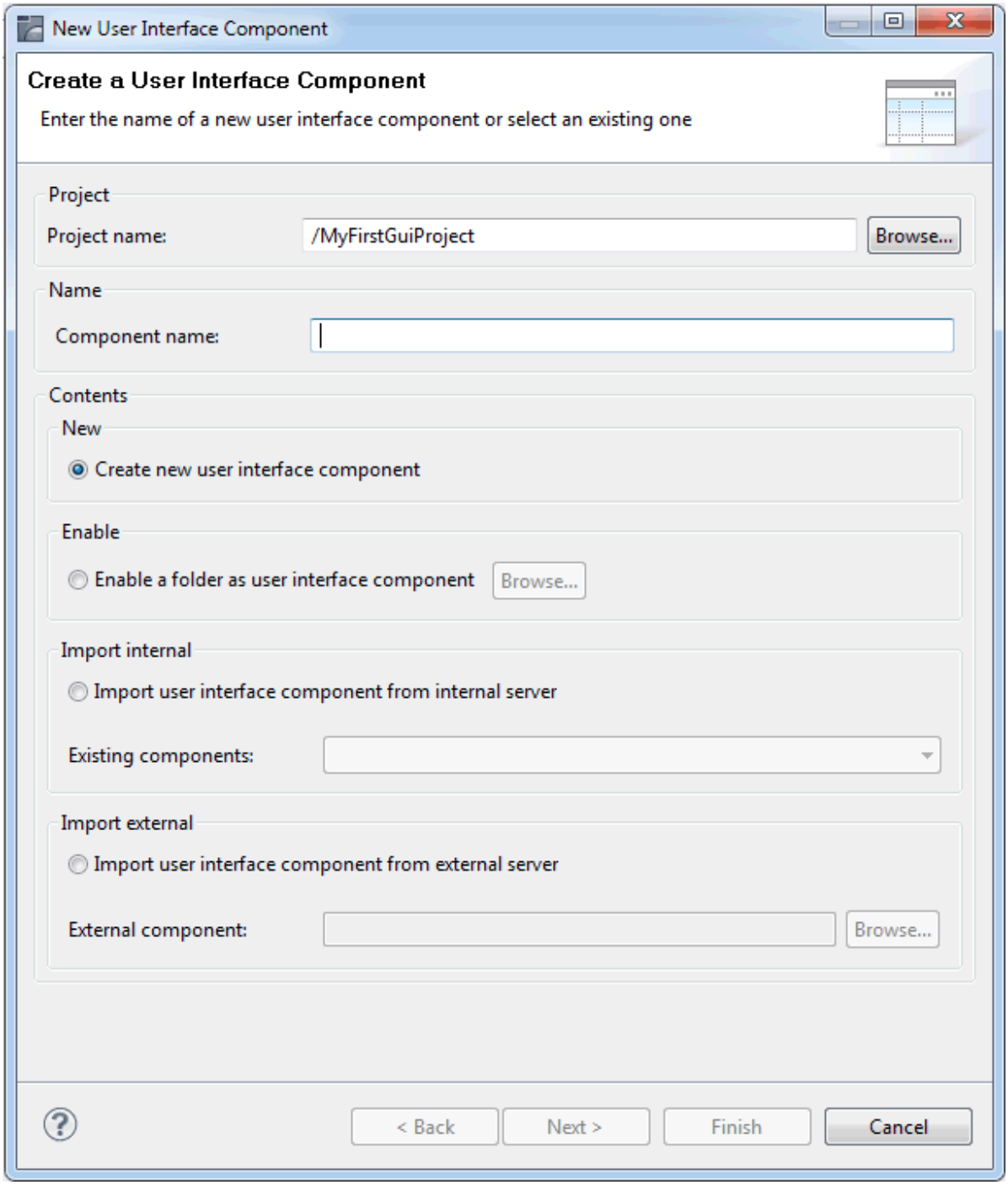
Page layouts are stored in so-called “user interface component” folders. You have to create such a folder for each Natural project that is to contain page layouts.

To create a user interface component

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

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

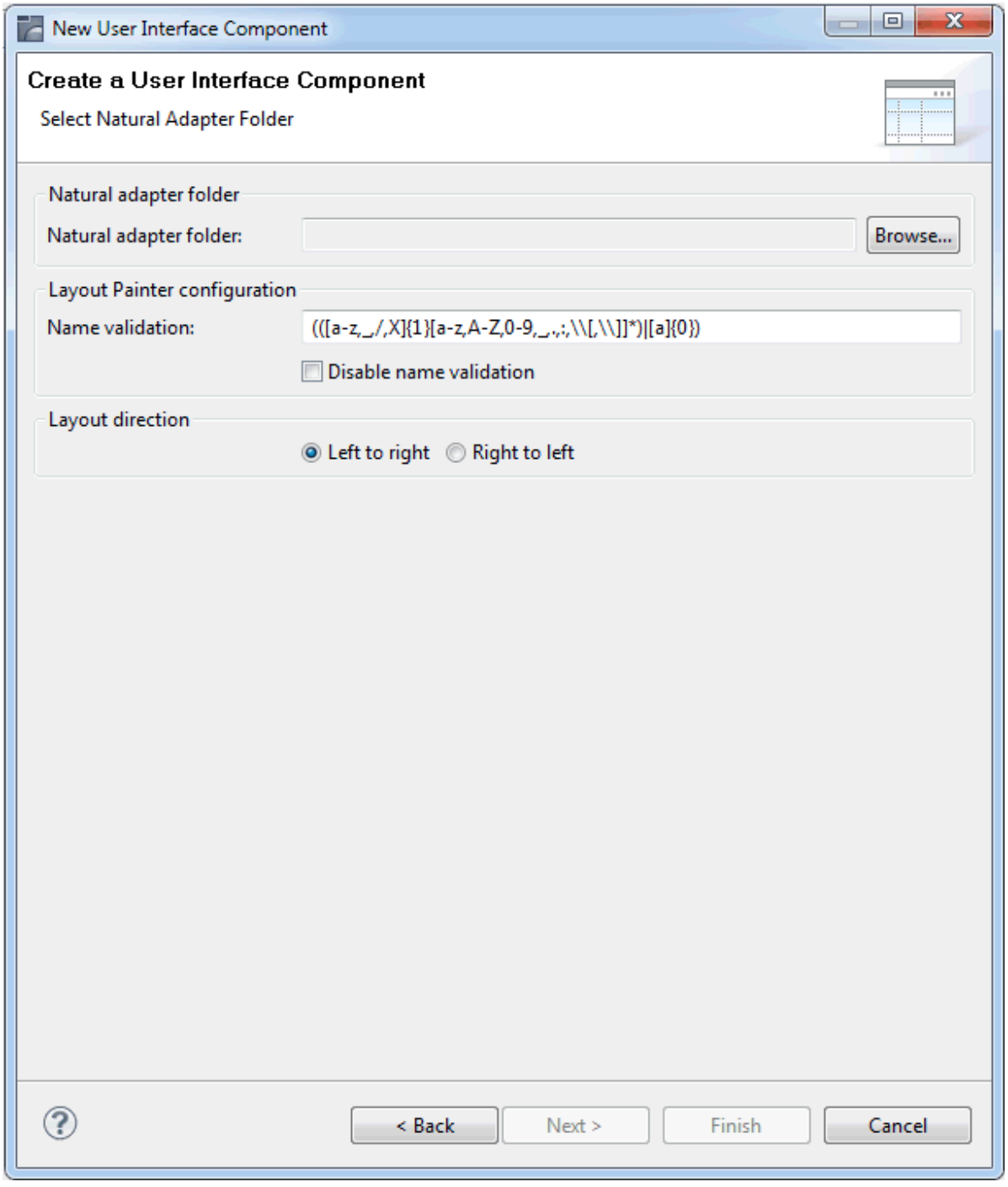
The following dialog box appears:



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

4 Choose the **Next** button.

The following page appears:



5 Choose the **Browse** button.

6 In the resulting dialog box, select the folder into which the adapters are to be generated and choose the **OK** button. Usually, you will select the *SRC* folder of a Natural library.

For this tutorial, you select the *SRC* folder in the library `CISHELLO`.

Note: If you want to use a layout direction different than the one used in this tutorial, select the **Right to left** option button. This tutorial, however, explains how to design your layout in the left-to-right direction.

7 Choose the **Finish** button.

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

You can now proceed with the next exercise: *Getting Started with the Layout Painter*.

6 Getting Started with the Layout Painter

The Layout Painter is used to write the page layout.

Creating a New Layout

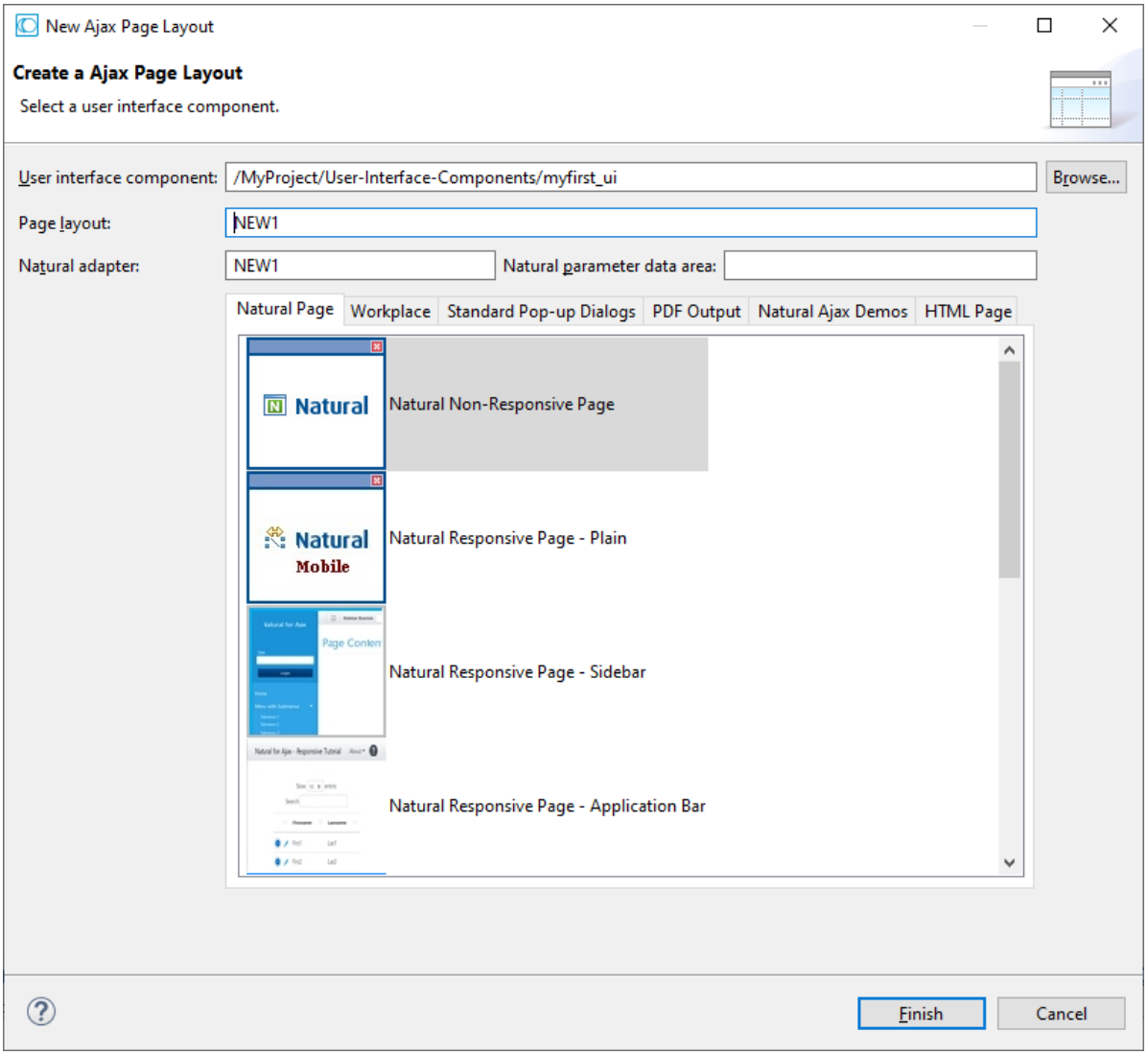
You will now create a layout which is stored in the user interface component you have previously created.

To create a new layout

1 In the **Project Explorer** view or in the **Natural Navigator** view, select the user interface component.

2 From the **File** menu, choose **New > Page Layout**.

The following dialog box appears:



3 Enter "helloworld" in the **Page layout** text box. This is the name of your layout definition.

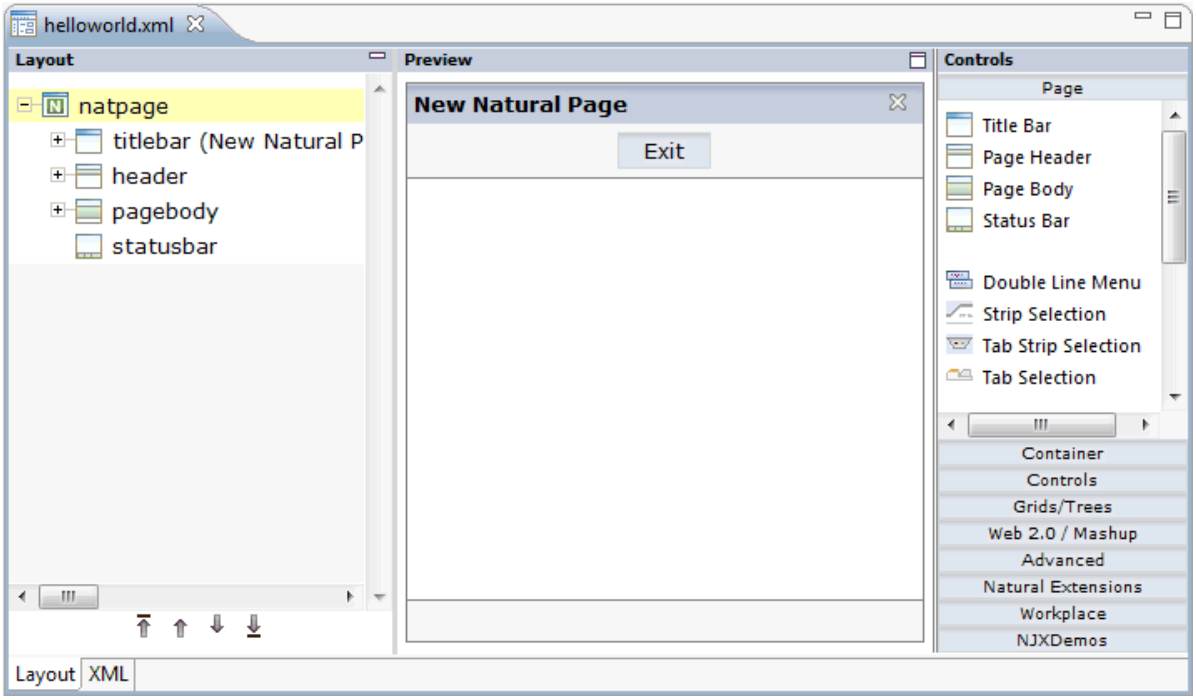
4 Enter a valid Natural object name in the **Natural adapter** text box. The name you specify here is used to create a Natural adapter object when you save the layout.

In this tutorial, the name `HELLO-A` is used.

5 On the **Natural Page** tab, select the template for the Natural page.

6 Choose the **Finish** button.

The Layout Painter appears.



Note: The file *helloworld.xml* is stored in the *xml* folder of the user interface component.

Elements of the Layout Painter

The Layout Painter is divided into several areas:

- **Layout Area (left side)**
  This area shows the layout tree which contains the controls that represent the XML layout definition. You drag these controls from the controls palette into the layout tree. Each node in the layout tree represents an XML tag.
- **Preview Area (middle)**
  The preview area shows the HTML page which is created using the controls in the layout area. This page is refreshed each time, you choose the **Refresh Preview** command (see below).
- **Controls Palette (right side)**
  Each control is represented by an icon. A tool tip is provided which appears when you move the mouse pointer over the control. This tool tip displays the XML tag which will be used in the XML layout. The palette is structured into sections, where each section represents certain types of controls.

When the Layout Painter is active, the additional menu **Ajax Developer** is shown in the menu bar.

Previewing the Layout

The layout tree inside the Layout Painter already contains some nodes that were copied from the template that you chose in the dialog in which you specified the name of the page. When you modify the layout and want to find out how the current layout definitions are rendered on the page, you can preview the layout as described below.

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

To preview the layout

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

Viewing the XML Code

When creating the layout, you can view the currently defined XML code.

To view the XML code

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

  At this stage of the tutorial, the resulting page contains the following XML layout definition for the nodes which were copied from the template.

```
<?xml version="1.0" encoding="UTF-8"?>
<natpage natsource="HELLO-A" natkcheck="true" natsinglebyte="true"
xmlns:njx="http://www.softwareag.com/njx/njxMapConverter">
<titlebar name="New Natural Page">
</titlebar>
<header withdistance="false">
<button name="Exit" method="onExit">
</button>
</header>
<pagebody>
</pagebody>
<statusbar withdistance="false">
</statusbar>
</natpage>
```

Opening the Ajax Developer Perspective

Ajax Developer provides its own perspective which provides just the views which are important for editing layouts.

Note: It is not mandatory to use the Ajax Developer perspective; the same views are provided in the NaturalONE perspective.

To open the Ajax Developer perspective

1 From the **Window** menu, choose **Open Perspective > Other**.

2 In the resulting **Open Perspective** dialog box, select **Ajax Developer** and choose the **OK** button.

You can now proceed with the next exercise: *Writing the GUI Layout*.

7 Writing the GUI Layout

You will now create the layout for your “Hello World!” application. When you have completed all exercises in this chapter, the layout should look as shown below and the **XML code** should be the same as shown in the section *About this Tutorial*.

Tip: **Preview the layout** and **view the XML code** each time you have completed an exercise. When important properties are not set, this is indicated in the **Problems** view. To get more information about problems, you can also choose **Show Protocol** from the **Ajax Developer** menu.

Specifying the Properties for the Natural Page

You will now specify the following for the Natural page:

- **Name for the Natural Adapter (`natsource`)**
  The value in the property `natsource` defines the name of the adapter. The adapter is a Natural object that your application will use to communicate with the page. It will be generated when you save the page layout.

If you do not specify a value for `natsource`, the name that you have specified for the layout (without the extension ".xml") will be used as the name for the Natural adapter. If you want to use the adapter in a development environment other than NaturalONE, you must make sure that the resulting name matches the naming conventions for Natural object names.

- **Handling of Strings (`natsinglebyte`)**
  Using the property `natsinglebyte`, you can specify how the strings displayed on this page are to be handled in the Natural application. Natural knows two types of strings: Unicode strings (format U) and code page strings (format A). By default, the strings displayed in web pages are mapped to Unicode strings in Natural. For this tutorial, you will specify that code page strings are to be used. Therefore, you will set the property `natsinglebyte` to "true".

  If you do not specify a value for `natsinglebyte` or when you set it to "false", Unicode strings will be used.

To specify the properties for the Natural page

1 In the layout tree, select the node **natpage**.

  The properties for this control are now shown in the **Properties** view.

2 Make sure that the following properties are specified:

Property	Value
`natsinglebyte`	true
`natsource`	HELLO-A

Specifying a Name for the Title Bar

You will now specify the string "Hello World!" which is to appear in the title bar of your application.

To specify the name for the title bar

1 In the layout tree, select the node **titlebar (New Natural Page)**.

  The properties for this control are now shown in the **Properties** view. You can see the default entry "New Natural Page" for the `name` property.

2 Specify the following property:

Property	Value
name	Hello World!

When you press ENTER, the node in the layout tree changes to **titlebar (Hello World!)**.

Note: Properties that are left blank are not shown in the XML code.

Specifying a Name and Method for the Button

You will now specify the string "Say Hello" which is to appear on the button. And you will specify the name of the method that is to be invoked when the user chooses this button.

To specify the name and the method for the button

1 In the layout tree, open the **header** node.

Note: By clicking the icon of a node, you hide or expand the node's subnodes.

You can now see the entry for the button with the default name "Exit".

2 Select the node **button (Exit)**.

3 Specify the following properties:

Property	Value
method	sayHello
name	Say Hello

The method needs to be programmed in the adapter. This will be explained later in this tutorial.

Adding the Input and Output Areas

The input and output areas in this tutorial are created using **Row Area** controls. These controls can be found in the **Container** section of the controls palette.

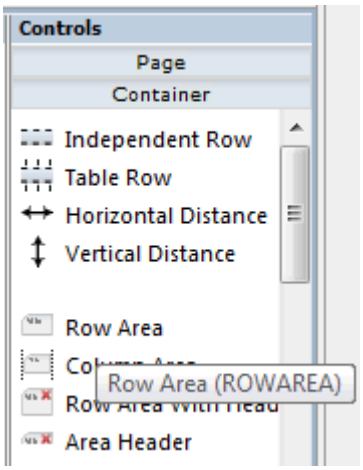
Each row area will contain an **Independent Row** control which in turn contains a **Label** and a **Field** control. These controls can be found in the **Controls** section of the controls palette.

For adding controls to your layout, you drag them from the controls palette onto the corresponding tree node in the layout tree. This is explained below.

To create the input area

1 Open the **Container** section of the controls palette.

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



2 Drag the **Row Area** control from the controls palette onto the **pagebody** node in the layout tree.

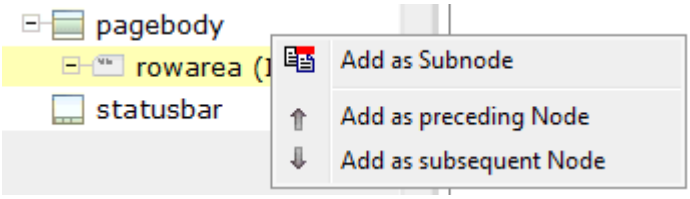
The row area is added as a subnode of the **pagebody** node. The new subnode is automatically selected so that you can maintain the properties of the row area directly in the **Properties** view.

3 Specify the following property:

Property	Value
name	Input Area

4 Drag the **Independent Row** control from the controls palette onto the **rowarea (Input Area)** node in the layout tree.

When you drop information into the tree, the system will sometimes respond by offering a context menu with certain options about where to place the control. In this case, the following context menu appears.



5 Choose the **Add as Subnode** command.

The control is now inserted below the **rowarea (Input Area)** node. The new node is shown as **itr**.

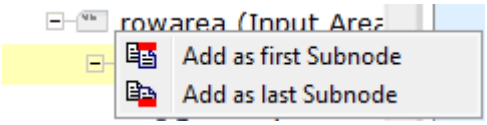
6 Open the **Controls** section of the controls palette.

7 Drag the **Label** control from the controls palette onto the **itr** node you have just inserted and specify the following properties:

Property	Value
`name`	Your Name
`width`	100

8 Drag the **Field** control from the controls palette onto the **itr** node you have previously inserted.

A context menu appears and you have to specify where to place the control.



9 From the context menu, choose the **Add as last Subnode** command.

10 Specify the following properties for the field:

Property	Value
`valueprop`	name
`width`	200

To create the output area

- Create the output area in the same way as the input area (add it as the last subnode of the **pagebody** node), with the following exceptions:

  Row Area

  Specify a different value for the following property:

Property	Value
name	Output Area

Label

Specify a different value for the following property:

Property	Value
name	Result

Field

Specify different values for the following properties:

Property	Value
valueprop	result
displayonly	true

Adding the Image

You will now add the image which is to be shown above the input area. To do so, you will use the **Icon** control which can be found in the **Controls** section of the controls palette.

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

To create a folder for images

1. In the **Project Explorer** view or in the **Natural Navigator** view, select the user interface component for which you want to create a folder.
2. Invoke the context menu and choose **New > Other**.
3. In the resulting dialog box, expand the **General** node, select **Folder** and choose the **Next** button.
4. Specify a folder name (for example, "images") and then choose the **Finish** button.
5. The image that you need for this tutorial (*hello.gif*) is provided in the */.naturalone/apache-tomcat/webapps/cisnatural/njxdemos/images* directory in your Eclipse workspace. Copy it to the images folder that you have just created.

To add the image

1 Drag the **Icon** control from the controls palette onto the **pagebody** node in the layout tree.

The icon is added as the last subnode of the **pagebody** node. It is automatically placed into an **itr** (independent row) node.

2 Specify the following property for the icon:

Property	Value
image	images/hello.gif

Note: You can also choose the icon using the browse button which is shown next to the property name.

3 Select the **itr** node containing the icon and choose the following button below the layout tree:

The selected node is now moved up so that it appears as the first subnode of the **pagebody** node.

4 Specify the following property for the **itr** node containing the icon:

Property	Value
takefullwidth	true

Adding a Horizontal Distance

You will now move the image to the right side of the page. To do so, you will use the **Horizontal Distance** control which can be found in both the **Controls** section and the **Container** section of the controls palette.

To add the horizontal distance

1 Drag the **Horizontal Distance** control from the controls palette onto the **itr** node containing the icon.

2 From the resulting context menu, choose the **Add as first Subnode** command.

The node **hdist** is inserted into the tree.

3 Specify the following property:

Property	Value
width	100%

Adding an Instructional Text

You will now enter a text which is to appear below the output area and which tells the user what to do.

To do so, you will once again use the **Independent Row** control into which you will insert a **Label** control.

Note: The **Independent Row** control can be found in both the **Controls** section and the **Container** section of the controls palette.

To add the independent row with the label

1 Drag the **Independent Row** control from the controls palette onto the **pagebody** node in the layout tree.

2 From the resulting context menu, choose the **Add as last Subnode** command.

The node **itr** is inserted into the tree.

3 Drag the **Label** control from the controls palette onto the **itr** node you have just created.

4 Specify the following properties for the label:

Property	Value
name	Input your name and press the 'Say Hello' button.
asplaintext	true

Adding a Vertical Distance

When you preview the layout, you will see that the text you have just added appears directly below the output area. You will now move the text 100 pixels to the bottom.

To do so, you will use the **Vertical Distance** control which can be found in both the **Controls** section and the **Container** section of the controls palette.

To add the vertical distance

1 Drag the **Vertical Distance** control from the controls palette onto the **itr** node containing the label.

2 From the resulting context menu, choose the **Add as preceding Node** command.

The node **vdist** is inserted into the tree.

3 Specify the following property:

Properties	Value
`height`	100

Saving Your Layout

If you have not already done so, you should now save your layout. Layouts are saved using the standard Eclipse functionality.

When you save a layout for the first time, an HTML file is generated (in addition to the XML file) which is placed into your user interface component (however, this HTML file can only be seen in the file system and not in Eclipse itself). This HTML file is updated each time you save the layout.

The Natural adapter is also created when you save your layout for the first time. The adapter has the name that you have specified in the `natsource` property of the Natural page, plus the extension that is used for adapters (*HELLO-A.NS8*) and is located in the *SRC* folder of your Natural library. Your application program will use the adapter to communicate with the page.

To save the layout

- From the **File** menu, choose **Save**.

  Or:

  Press CTRL+S.

  Or:

  Choose the following button from the Eclipse toolbar:

You can now proceed with the next exercise: ***Creating the Natural Code***.

8 Creating the Natural Code

Content of the Adapter

When you saved your page layout, the Natural adapter `HELLO-A` was created for your page. This is the name that you have specified earlier in this tutorial. Your application program will use the adapter to communicate with the page.

The adapter code looks as follows:

```
* PAGE1: PROTOTYPE       --- CREATED BY Application Designer --- /*<RO>>
* PROCESS PAGE USING 'XXXXXXXX' WITH
* NAME RESULT
DEFINE DATA PARAMETER
/*( PARAMETER
1 NAME (A) DYNAMIC
1 RESULT (A) DYNAMIC
/*) END-PARAMETER
END-DEFINE
*
/*( PROCESS PAGE
PROCESS PAGE U'/MyFirstUI/helloworld' WITH
PARAMETERS
NAME U'name'
VALUE NAME
NAME U'result'
VALUE RESULT
END-PARAMETERS
/*) END-PROCESS
*
*  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'sayHello'
*   /* TODO: Implement event code.
*   PROCESS PAGE UPDATE FULL
*  NONE VALUE
*   /* Unhandled events.
*   PROCESS PAGE UPDATE
* END-DECIDE
/*/*) END-HANDLER
*
END /*<<RO>
```

Creating the Main Program

You will now create the main program which uses the adapter to display the page and which handles its events. The name of the program will be `HELLO-P` and you will store it in the library `CISHELLO`.

To create the main program

1 Open the NaturalONE perspective.

2 In the **Project Explorer** view or in the **Natural Navigator** view, go to the *SRC* folder of the library `CISHELLO`.

3 Invoke the context menu for the adapter `HELLO-A` and choose **Generate Main Program**.

4 In the resulting dialog box, enter "HELLO-P" as the name of the new program and choose the **OK** button.

The program with the specified name is now created.

5 Initialize the page data. In the page layout definition, the property `name` has been bound to the FIELD control with the label **Your Name**. For the property `name`, a parameter `NAME` has been generated into the parameter data area of the adapter. Thus, in order to preset the FIELD control, we will preset the variable `NAME` with the value "Ajax Developer".

```
DEFINE DATA LOCAL
1 NAME (A) DYNAMIC
1 RESULT (A) DYNAMIC
END-DEFINE
*
NAME := 'Ajax Developer'
PROCESS PAGE USING 'HELLO-A'
WITH NAME RESULT
```

6 Handle the events that can occur on the page. A template for the event handler code has been generated as a comment block into the page adapter `HELLO-A`.

```
DEFINE DATA LOCAL
1 NAME (A) DYNAMIC
1 RESULT (A) DYNAMIC
END-DEFINE
*
NAME := 'Ajax Developer'
PROCESS PAGE USING 'HELLO-A'
WITH NAME RESULT
*
DECIDE ON FIRST *PAGE-EVENT
VALUE 'nat:page.end',U'nat:browser.end'
```

```
/* Page closed.
IGNORE
VALUE 'sayHello'
/* TODO: Implement event code.
PROCESS PAGE UPDATE FULL
NONE VALUE
/* Unhandled events.
PROCESS PAGE UPDATE
END-DECIDE
*
END
```

After the page has been displayed, the user raises events on the page by using the controls. The name of the raised event is then contained in the system variable `*PAGE-EVENT`. Depending on the event, the program modifies the page data, resends it to browser with a `PROCESS PAGE UPDATE FULL` statement and waits for the next event to occur.

The predefined events `nat:page.end` and `U'nat:browser.end'` are raised when the user closes the page or closes the browser. The event `sayHello` is raised when the user chooses the **Say Hello** button. Previously in this tutorial, you have bound the event `sayHello` to this button while designing the page. The `NONE VALUE` block should always be defined as above. It contains the default handling of all events that are not handled explicitly.

7 When the event `sayHello` occurs, we want to display a greeting in the FIELD control with the label **Result**. Therefore, we modify the variable `RESULT` (which is bound to the corresponding FIELD control in the page layout) accordingly before we resend the page data.

```
DEFINE DATA LOCAL
1 NAME (A) DYNAMIC
1 RESULT (A) DYNAMIC
END-DEFINE
*
NAME := 'Ajax Developer'
PROCESS PAGE USING 'HELLO-A'
WITH NAME RESULT
*
DECIDE ON FIRST *PAGE-EVENT
VALUE 'nat:page.end',U'nat:browser.end'
/* Page closed.
IGNORE
VALUE 'sayHello'
/* TODO: Implement event code.
COMPRESS 'Hello, ' NAME '!' TO RESULT
PROCESS PAGE UPDATE FULL
NONE VALUE
/* Unhandled events.
PROCESS PAGE UPDATE
END-DECIDE
*
END
```

The main program is now complete.

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

Note: When **Build Natural projects automatically** is selected in the Natural preferences, the Natural server is automatically updated each time you save a source or add a new source. The source is uploaded to the Natural server and is cataloged there.

Testing the Completed Application

You will now execute the program and check whether it provides the desired result. If it contains errors, you can debug it.

To execute the program

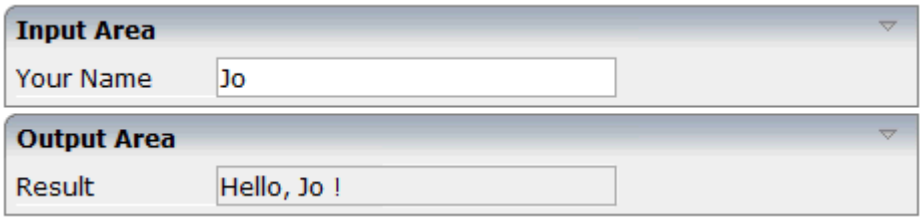
1 In the **Project Explorer** view or in the **Natural Navigator** view, select the program `HELLO-P`.

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

The program is executed and the output is shown in the browser.

3 Enter your name and choose the **Say Hello** button.

The page should now successfully "talk" to your adapter.



To debug the program

1 In the **Project Explorer** view or in the **Natural Navigator** view, select the program `HELLO-P`.

2 Invoke the context menu and choose **Debug As > Natural Application**.

The Debug perspective is opened. In the editor window, the debugger waits at the first executable source code line. You can now use the standard Eclipse functionality to debug the Natural program.

You have now completed this tutorial. See the remaining section of these *First Steps* for **some background information**.

9 Some Background Information

Name Binding between Controls and Adapter

Which are the critical parts when building the “Hello World!” application?

- The NATPAGE control in the layout points to the name of the adapter object (property `natsource`).
- The FIELD control in the layout points to the property name of the adapter (property `valueprop`).
- The BUTTON control in the layout points to the event `sayHello()` of the adapter (property `method`).

There is a name binding between the layout definition and its corresponding adapter. This is the simple and effective approach of the development process: The adapter represents a logical abstraction of what the page displays. All layout definitions are kept in the page - all the logic is kept in the adapter. (Or better: behind the adapter. The adapter itself should only be a facade to the “real” application logic.)

Data Exchange at Runtime

What happens at runtime?

- When the user starts a Natural session from the logon page, the Natural program that the user specified in the command line is started.
- The Natural program executes a `PROCESS PAGE` statement, using an adapter.
- The `PROCESS PAGE` statement passes the name of the HTML page to be used and the initial page data to the browser.
- The browser displays the page. JavaScript code on the page distributes the initial data to the controls.
- The user provides some input, for example, enters the name. The content change is stored inside the page. The Natural program is not yet involved.
- The user does something which causes a flush of the changes (for example, the user chooses a button). Therefore, all registered data changes are packaged and are sent through the adapter to the Natural program, including the information which event has been raised.
- The Natural program receives the modified data.
- The system variable `*PAGE-EVENT` receives the name of the raised event.
- The event handler in the Natural program modifies the data and resends it to the page using a `PROCESS PAGE UPDATE` statement.
- And so forth.

With a standard HTTP connection, only the changed content of the screen is passed when operating on one page. The layout is kept stable in the browser. Consequently, there is no flickering of the page due to page reloading.

All steps described in the list above are done completely transparent to your adapter; i.e. you do not have to cope with session management, stream parsing, error management, building up HTML on the server, etc. You just have to provide an intelligent HTML page by defining it in the Layout Painter and an adapter object.

Files and their Locations

Have a look at the files created for your “Hello World!” application and take notice of the directory in which they are located.

- The XML layout definition is kept in the *MyFirstUI/xml* directory.
- The generated HTML page is kept directly in the *MyFirstUI* directory of the user interface component. There are possibly also some other files inside this directory that start with "ZZZZ". These files are temporary files used when previewing pages inside the Layout Painter.
- The generated Natural adapters are kept in the directory that you specified while creating the user interface component.
- In the directory *MyFirstUI/accesspath*, “access restriction” files are generated. If you view these files inside the text editor, you see that one file is maintained for each page; it holds the information about which properties are accessed by the page.

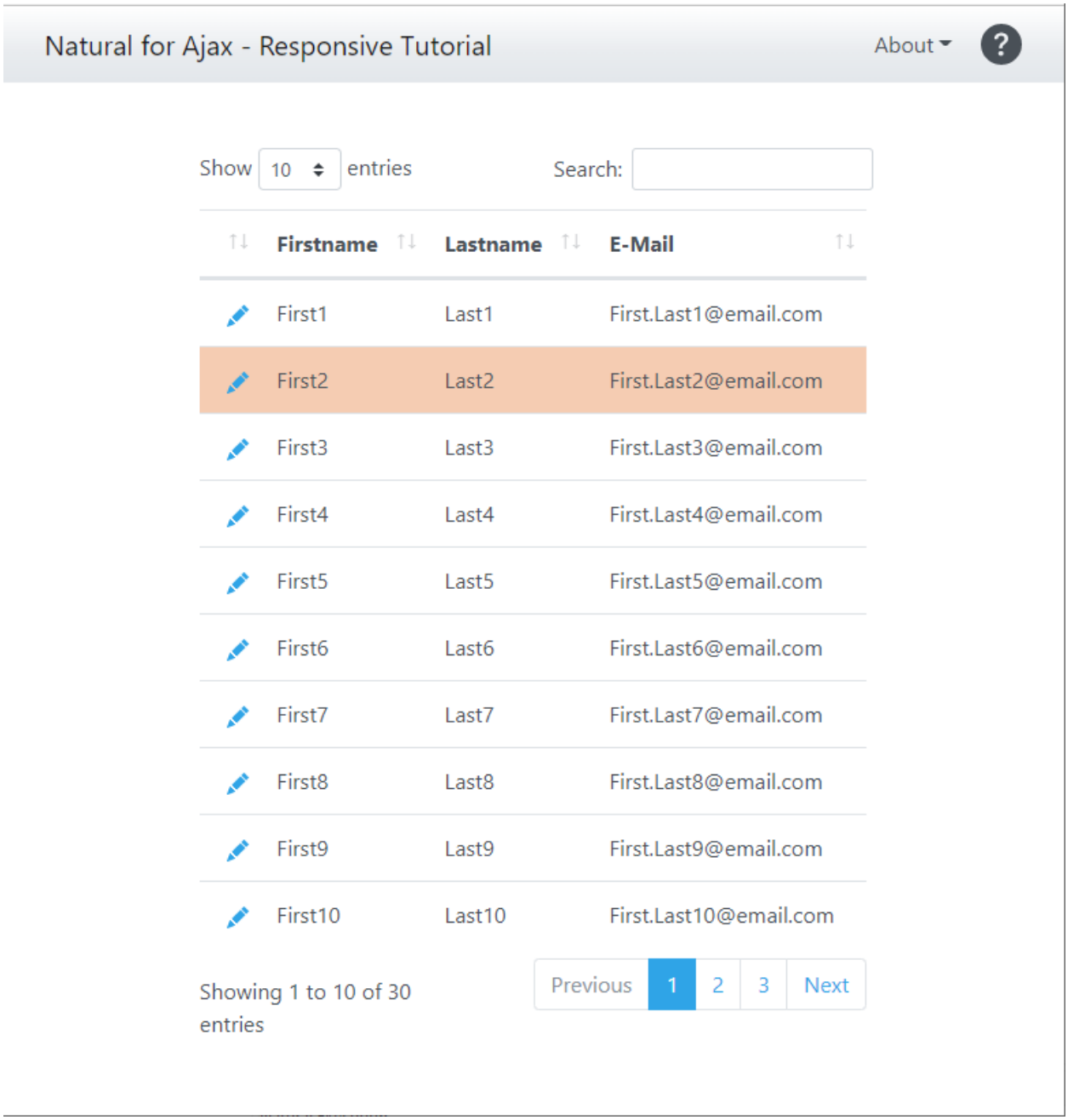
III First Steps Responsive Pages

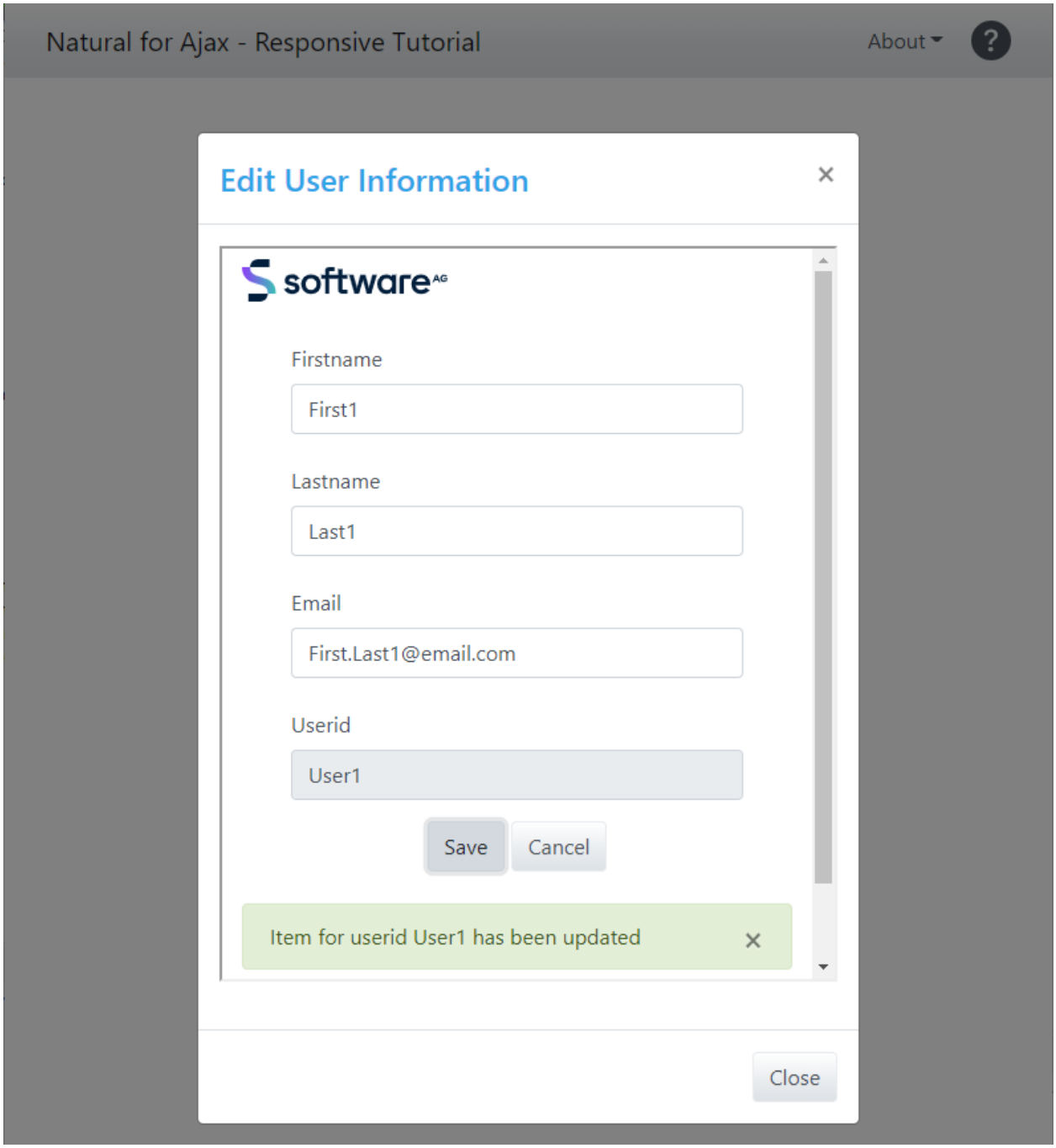
This tutorial explains how to build a responsive application and is organized under the following headings:

10 About this Tutorial

In this tutorial you will build a responsive application with two pages. You will use NaturalONE and Natural for Ajax. It shows you how to use the responsive control set of Natural for Ajax.

Here you can see what your application will look like when you have completed all the steps in this tutorial:


Natural for Ajax - Responsive Tutorial
About
Show 10 entries
Search:
Firstname
Lastname
E-Mail
First1 Last1 First.Last1@email.com
First2 Last2 First.Last2@email.com
First3 Last3 First.Last3@email.com
First4 Last4 First.Last4@email.com
First5 Last5 First.Last5@email.com
First6 Last6 First.Last6@email.com
First7 Last7 First.Last7@email.com
First8 Last8 First.Last8@email.com
First9 Last9 First.Last9@email.com
First10 Last10 First.Last10@email.com
Showing 1 to 10 of 30 entries
Previous
1
2
3
Next



Your application will work as follows: When you click the pencil icon in the grid, a pop-up window will open to modify the item. When resizing the browser window, the grid page as well as the pop-up page will adapt to the size of the browser window in a responsive way.

The Tutorial is split into three parts. You can do Part 1 and/or Part 2 independently of the other parts. Part 3 requires that you have done Part 1 + 2.

	Learning	Estimated Time
Part 1	Develop a responsive page with an application bar and a grid.	20 min
Part 2	Develop a responsive page with input controls including validation, displayonly settings and submit buttons.	15 min
Part 3	Page navigation between two pages. Integrate a page as pop-up.	15 min

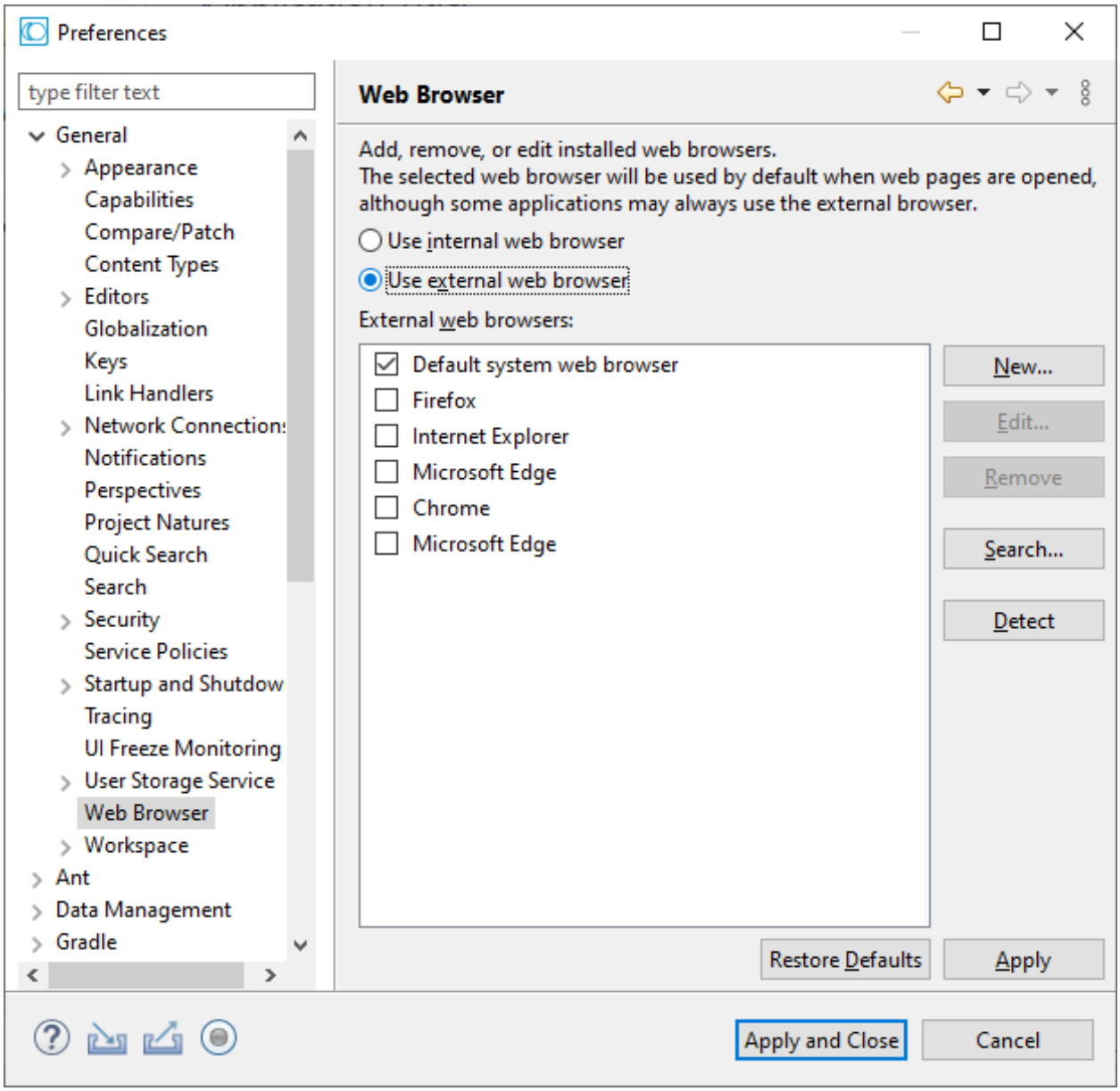
This tutorial assumes that you have started NaturalONE and switched to *The NaturalONE Perspective*. This Tutorial uses the **Project Explorer** view.

11 Prepare your NaturalONE Workspace

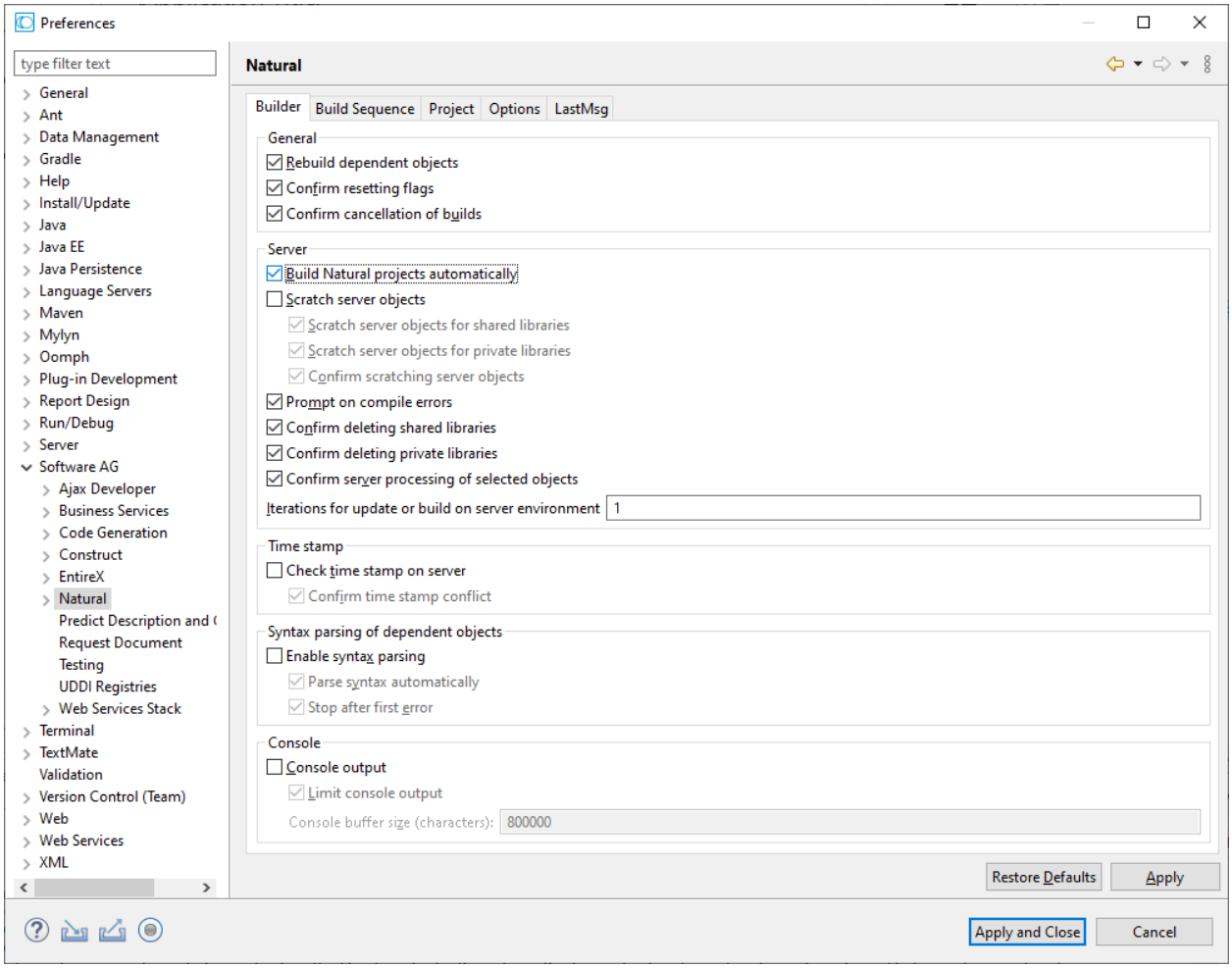
Perspective Settings

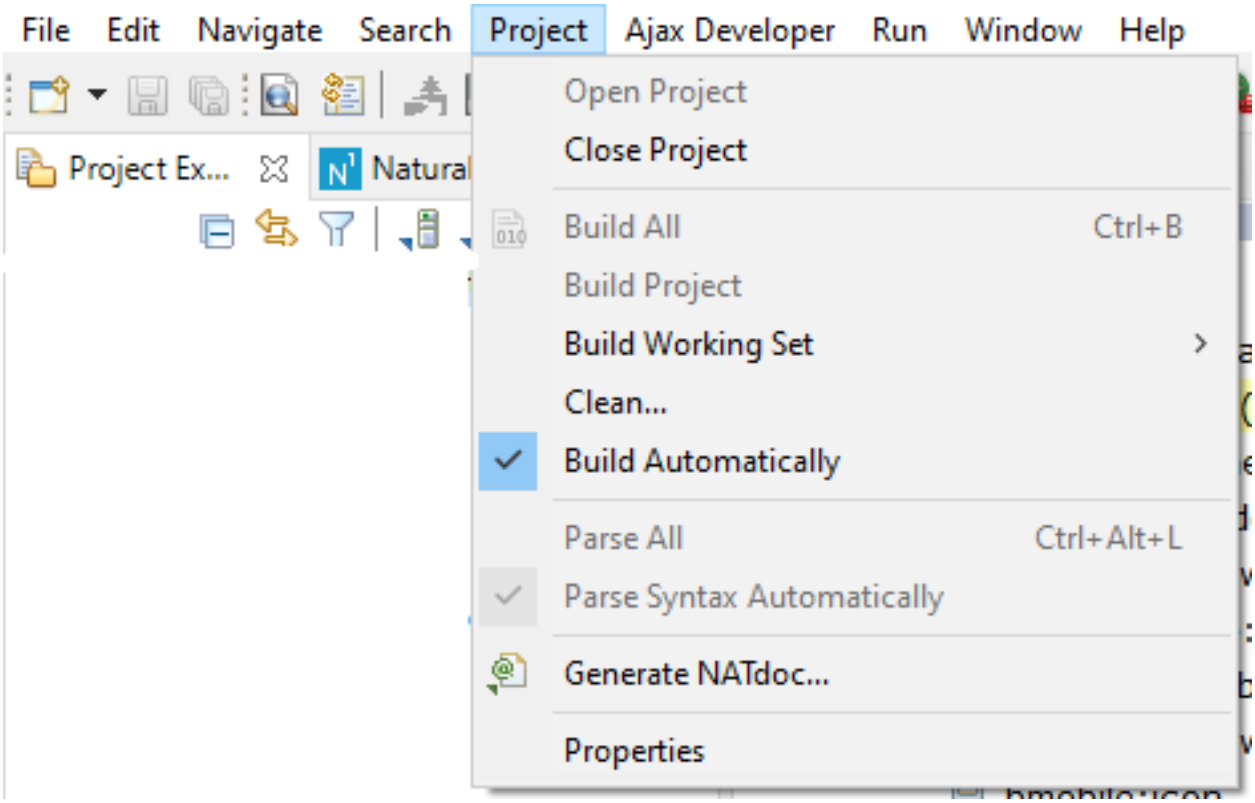
Click Window > Preferences to apply the settings described below.

To run the responsive application, it is required to use an external web browser. The internal web browser of Eclipse does not fully support responsive rendering.



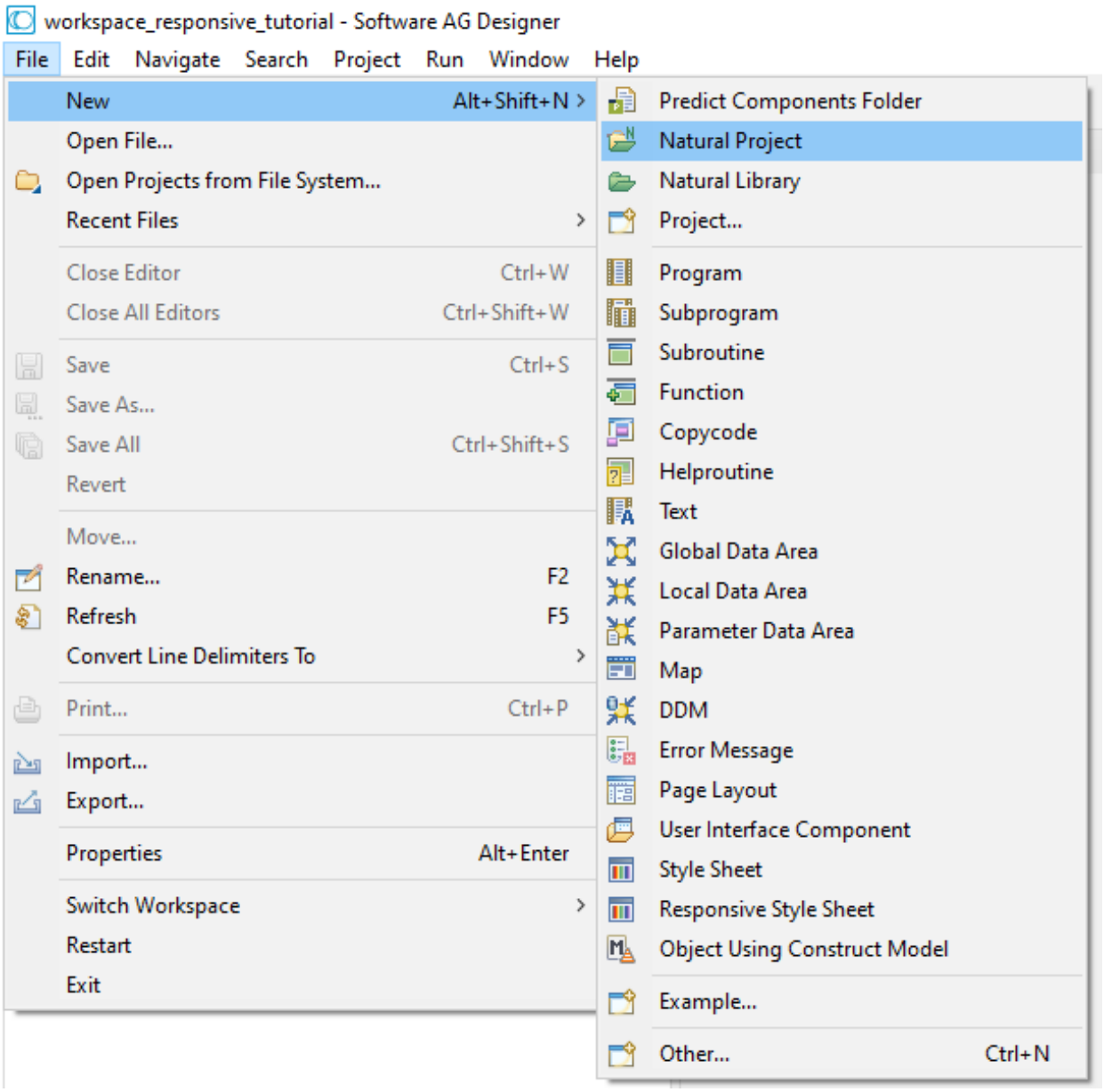
We want that the Natural adapters are always regenerated and compiled automatically whenever we change a responsive layout.


Preferences
type filter text
General
Ant
Data Management
Gradle
Help
Install/Update
Java
Java EE
Java Persistence
Language Servers
Maven
Mylyn
Oomph
Plug-in Development
Report Design
Run/Debug
Server
Software AG
Ajax Developer
Business Services
Code Generation
Construct
EntireX
Natural
Predict Description and (
Request Document
Testing
UDDI Registries
Web Services Stack
Terminal
TextMate
Validation
Version Control (Team)
Web
Web Services
XML
Natural
Builder
Build Sequence
Project
Options
LastMsg
General
Rebuild dependent objects
Confirm resetting flags
Confirm cancellation of builds
Server
Build Natural projects automatically
Scratch server objects
Scratch server objects for shared libraries
Scratch server objects for private libraries
Confirm scratching server objects
Prompt on compile errors
Confirm deleting shared libraries
Confirm deleting private libraries
Confirm server processing of selected objects
Iterations for update or build on server environment
1
Time stamp
Check time stamp on server
Confirm time stamp conflict
Syntax parsing of dependent objects
Enable syntax parsing
Parse syntax automatically
Stop after first error
Console
Console output
Limit console output
Console buffer size (characters):
800000
Restore Defaults
Apply
Apply and Close
Cancel

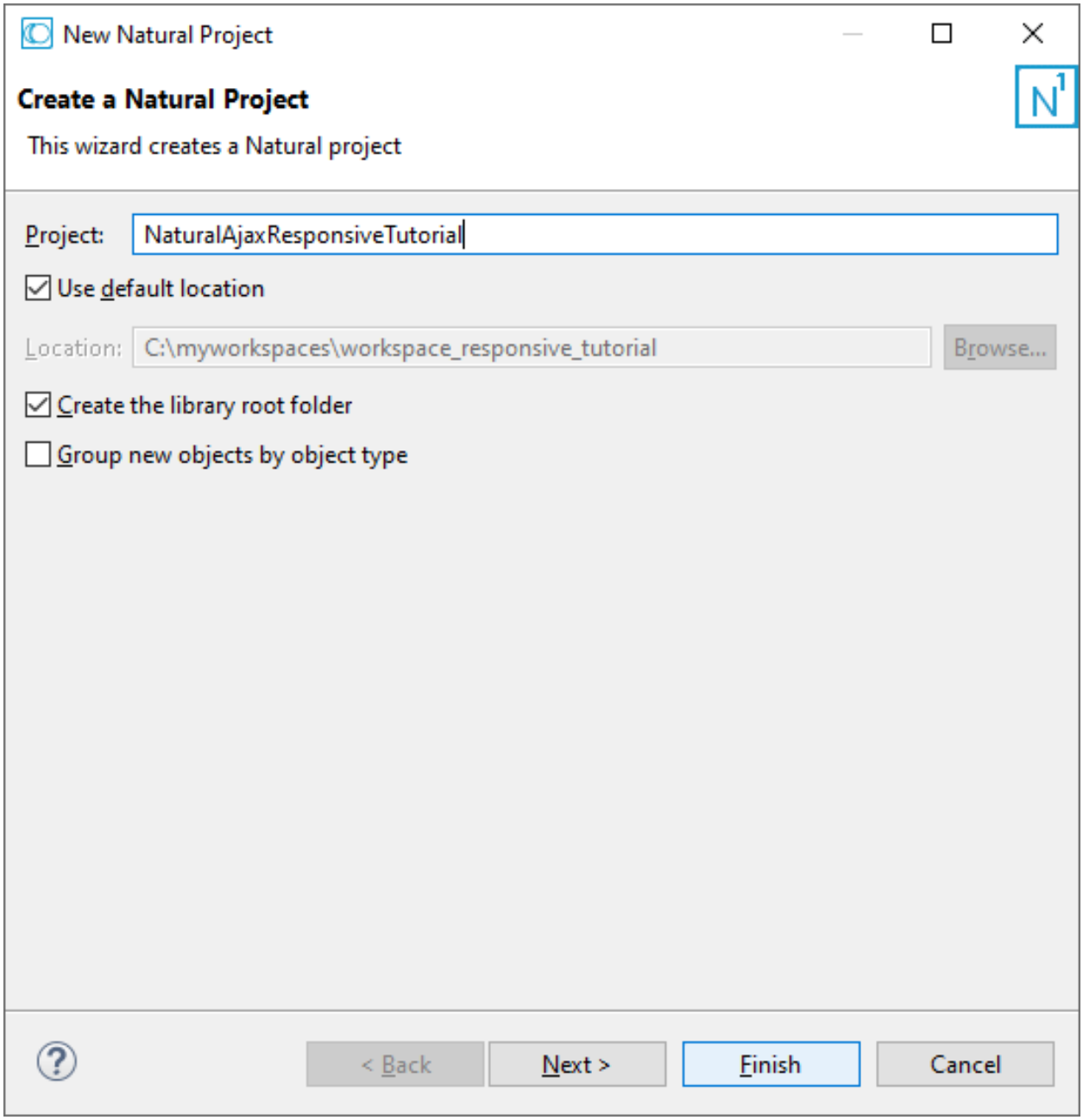


Create a Natural Project with a User Interface Component

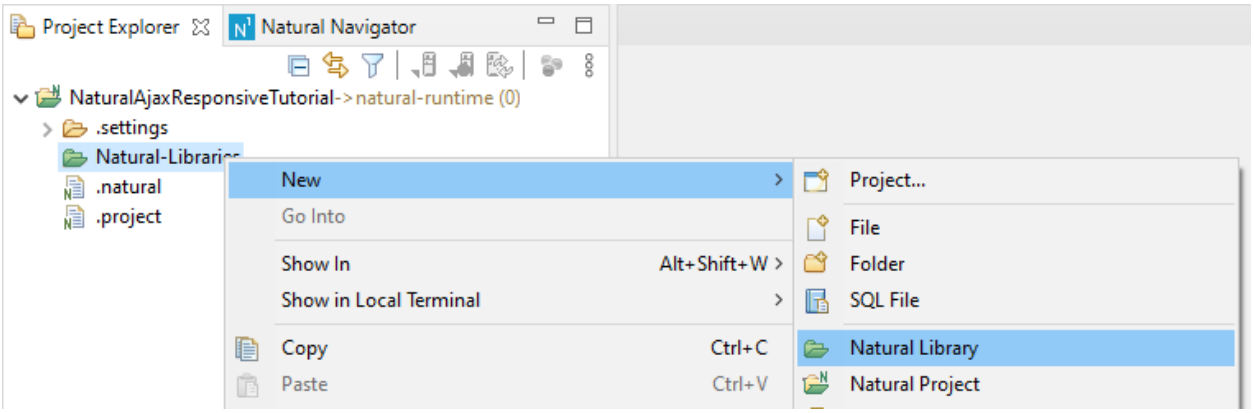
In the File menu select **New** > **Natural Project**.



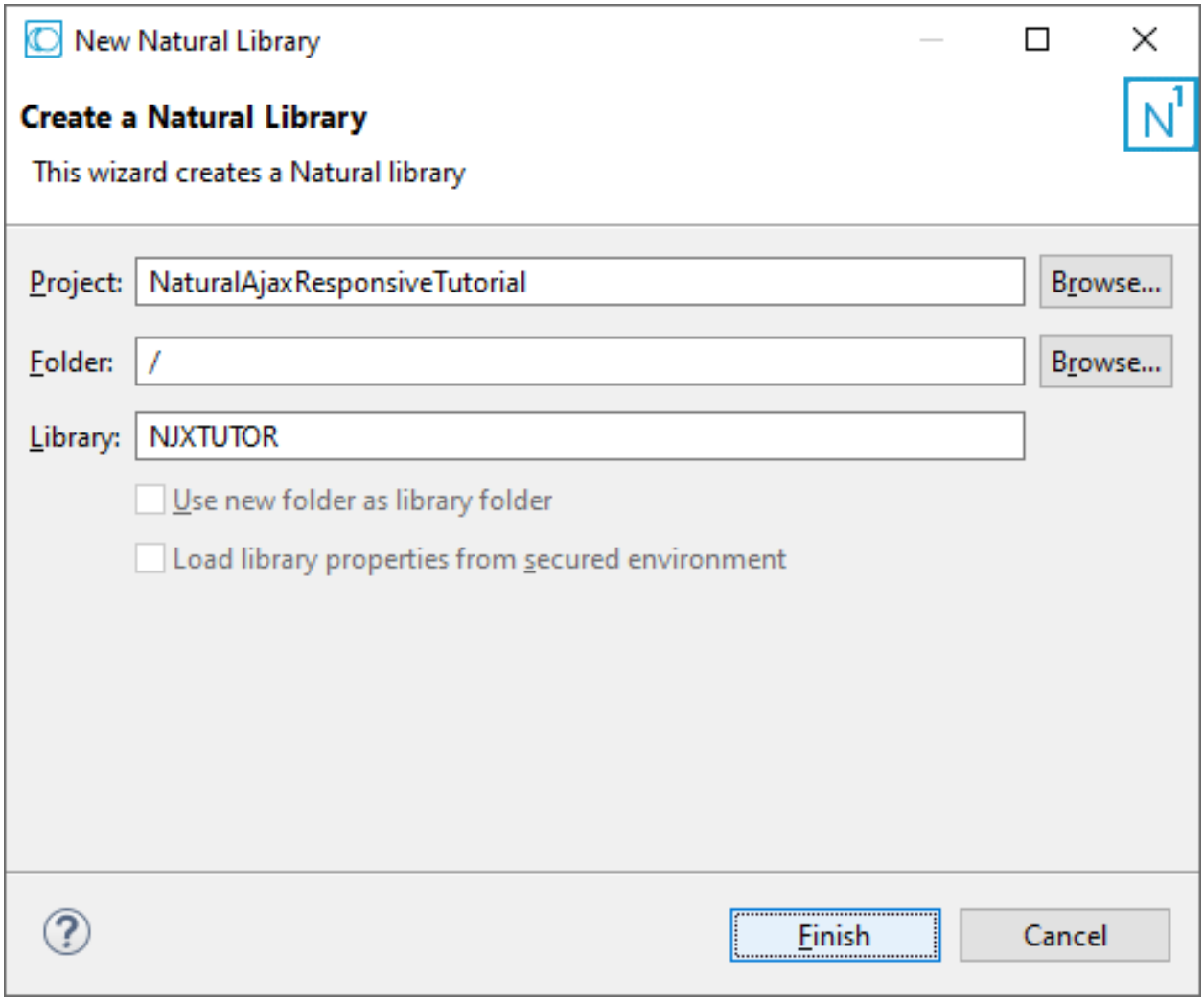
Set **NaturalAjaxResponsiveTutorial** as project and click the **Finish** button.



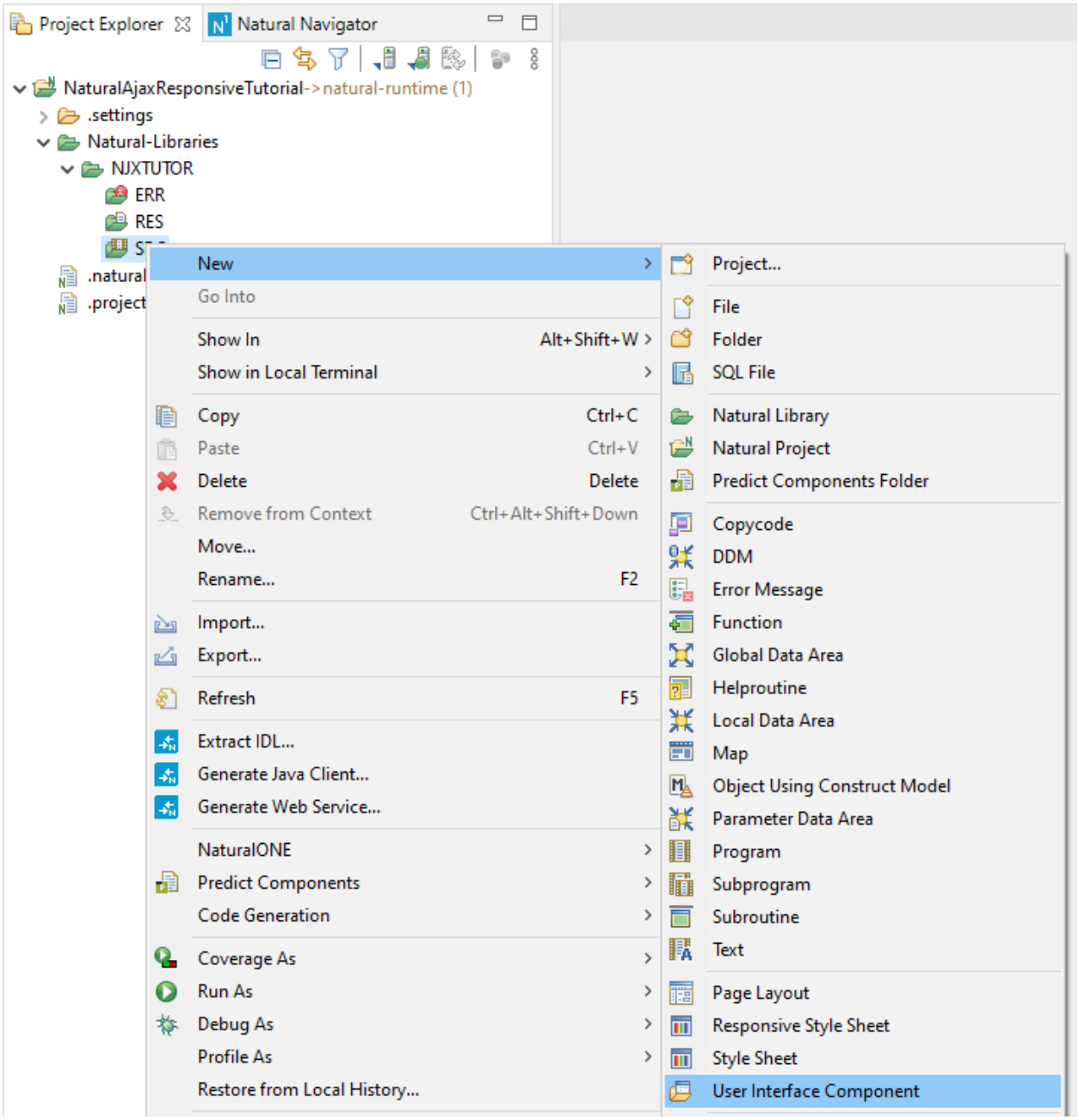
In the Project Explorer right click on the **Natural-Libraries** folder of the created project and select **New** > **Natural Library**.



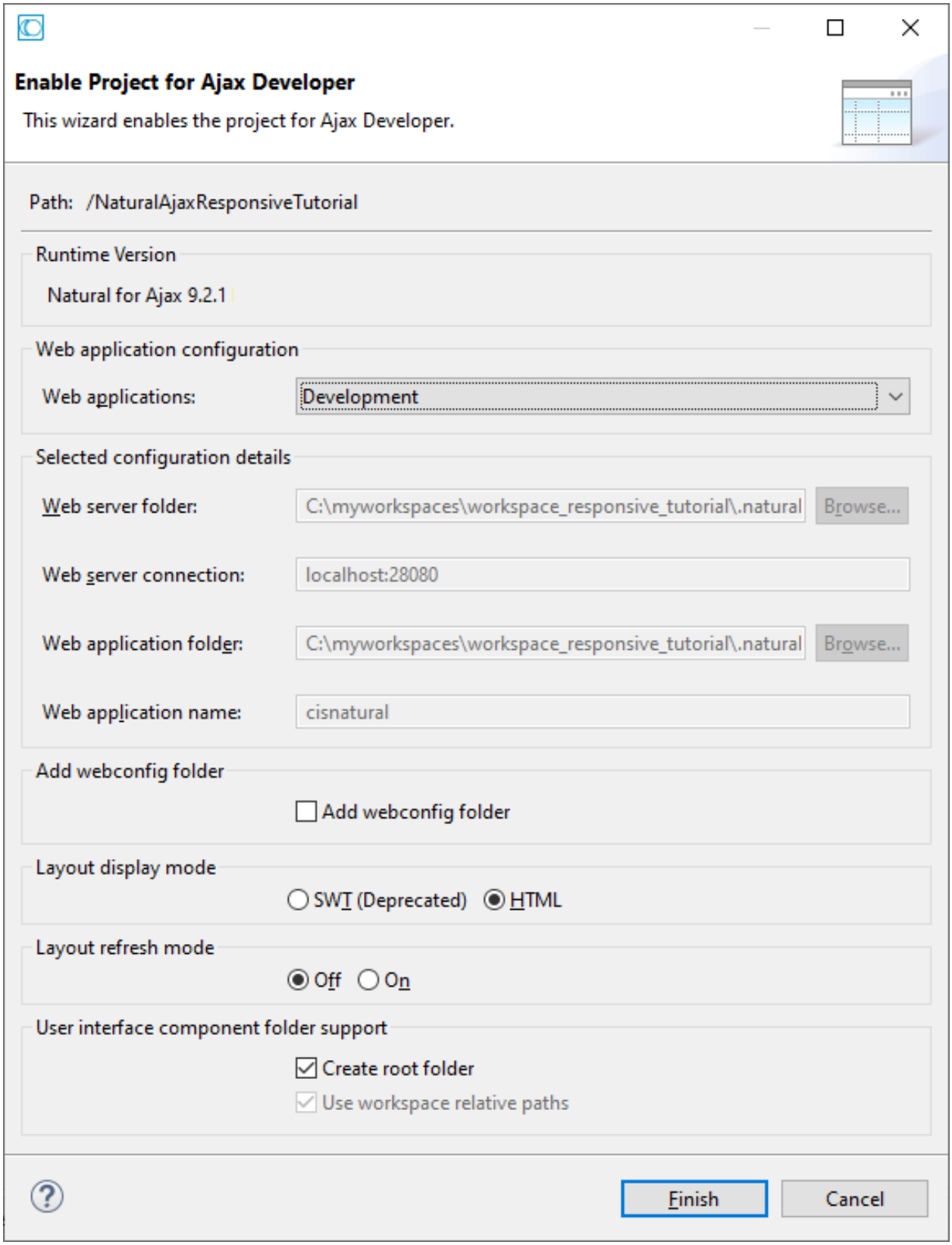
Set **NJXTUTOR** as Library and click the **Finish** button.



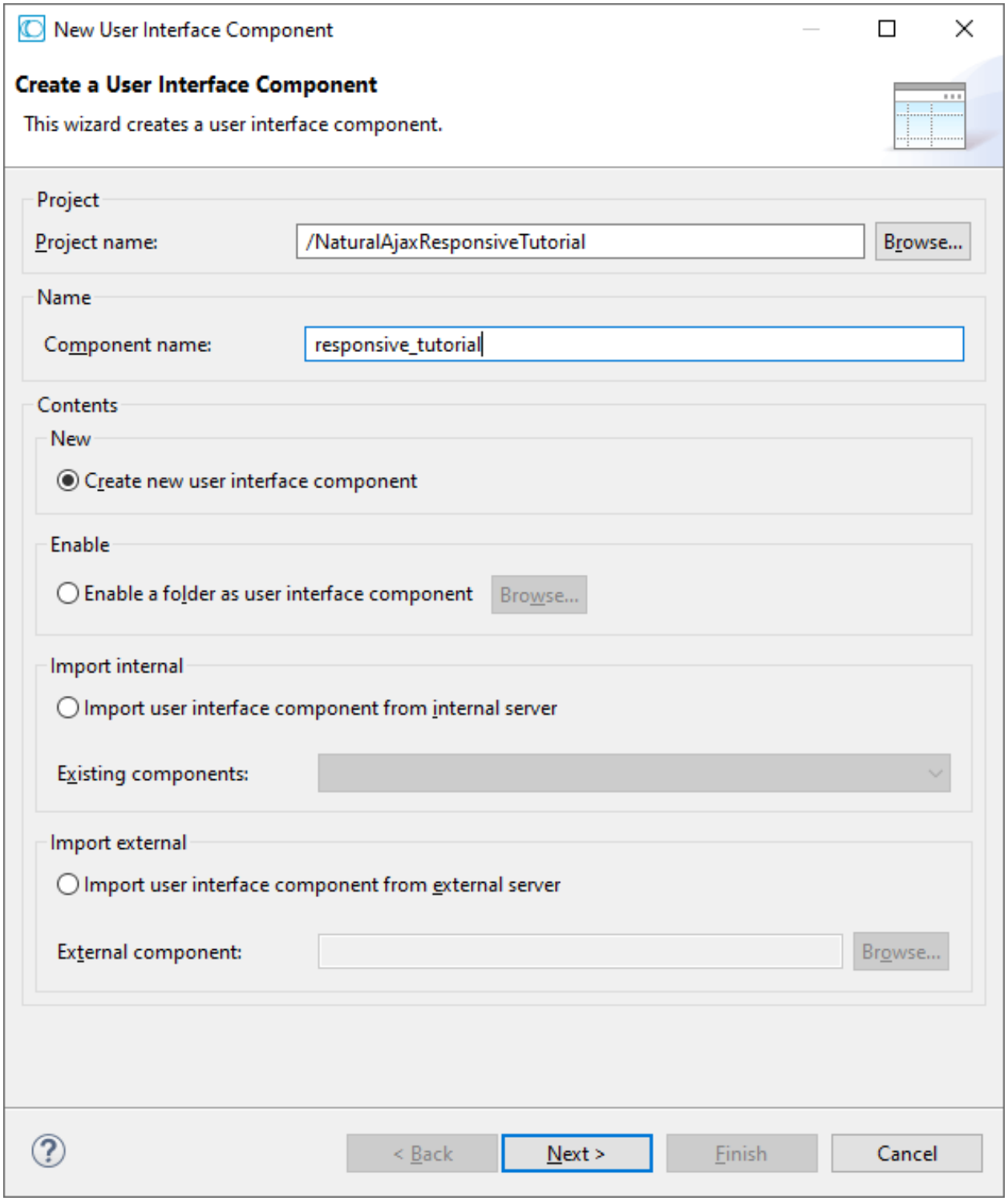
In the Project Explorer right click on the **SRC** folder of the created library and select **New** > **User Interface Component**.



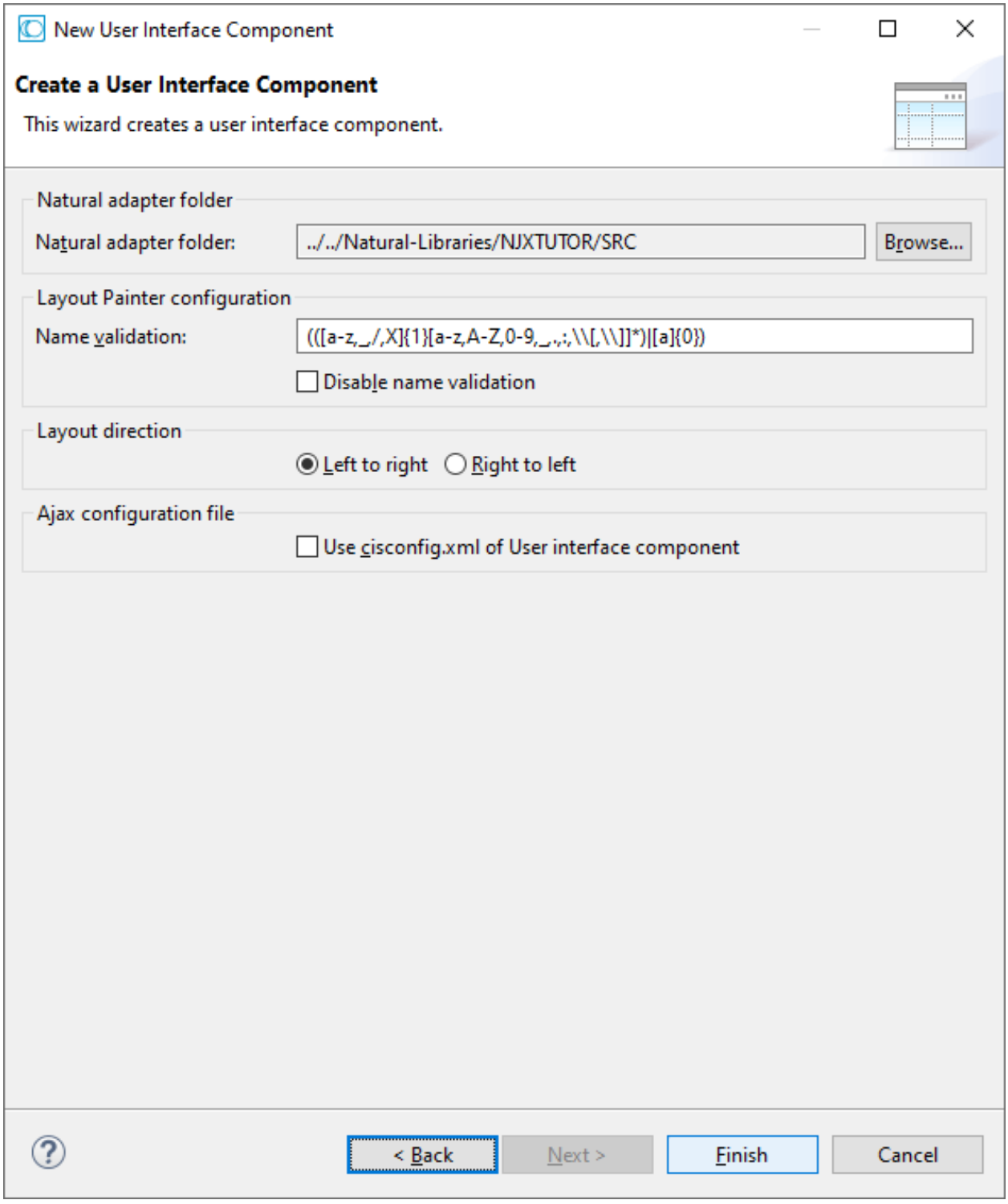
The dialog **Enable Project for Ajax Developer** is opened. Click the **Finish** button.


Enable Project for Ajax Developer
This wizard enables the project for Ajax Developer.
Path: /NaturalAjaxResponsiveTutorial
Runtime Version
Natural for Ajax 9.2.1
Web application configuration
Web applications:
Development
Selected configuration details
Web server folder:
C:\myworkspaces\workspace_responsive_tutorial\.natural
Browse...
Web server connection:
localhost:28080
Web application folder:
C:\myworkspaces\workspace_responsive_tutorial\.natural
Browse...
Web application name:
cisnatural
Add webconfig folder
Add webconfig folder
Layout display mode
SWT (Deprecated)
HTML
Layout refresh mode
Off
On
User interface component folder support
Create root folder
Use workspace relative paths
Finish
Cancel

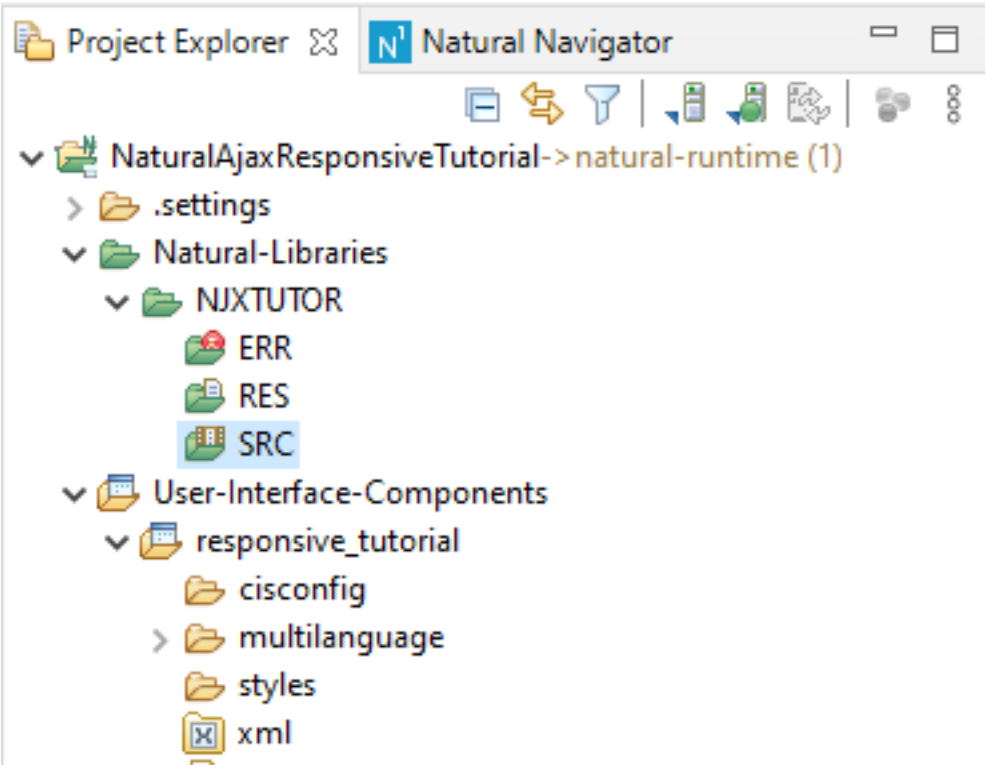
The wizard **Create a User Interface Component** is opened. Set **responsive_tutorial** as Component name and click the **Next** button.



The next wizard page appears. You can see that the correct Natural source folder for your Natural adapters is automatically selected. Click the **Finish** button.



You have successfully created an Ajax enabled Natural project and a User Interface Component for this tutorial. In Project Explorer you will see:



12 PART1: Develope the Main Application Page

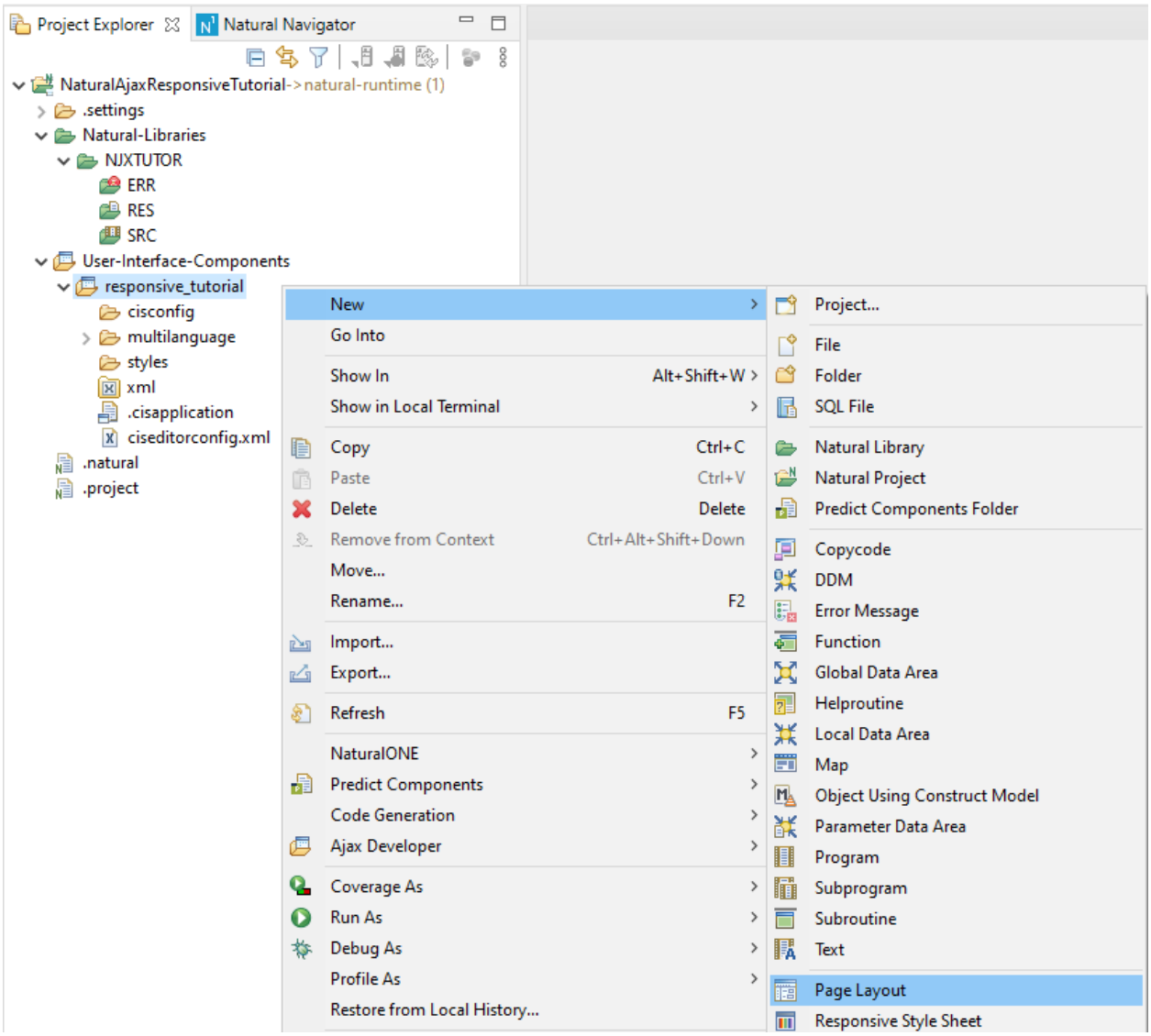
Our main application page will contain:

- A responsive application bar
- A responsive data grid.

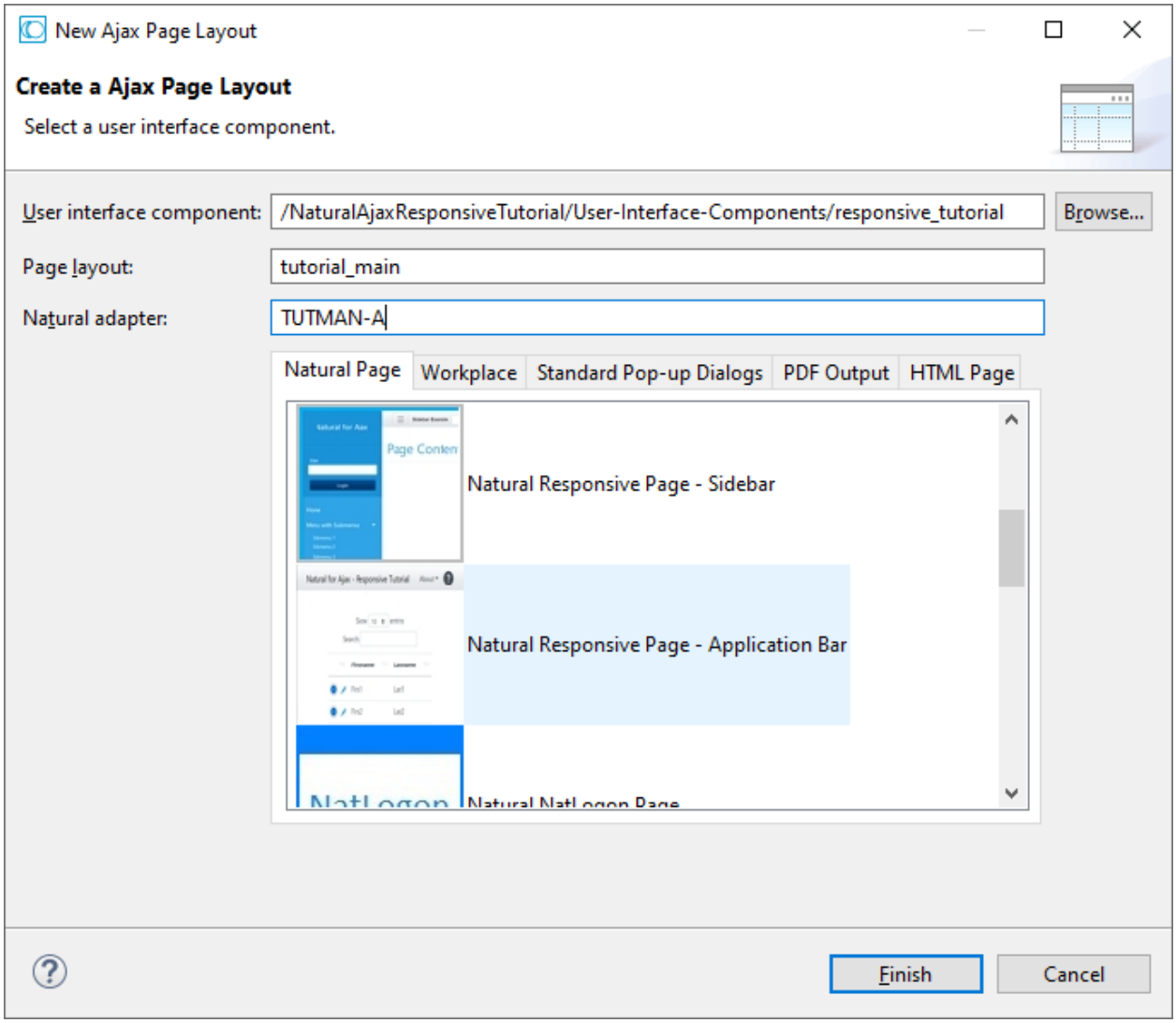
We will use a template which already contains a basic application bar.

The Application Template

Right click the folder responsive_tutorial in Project Explorer and select New > Page Layout



In the **Natural Page** tab scroll a little bit down and select the template **Natural Responsive Page - Application Bar**. Set **tutorial_main** in the **Page layout** field. Set **TUTMAIN-A** in the **Natural adapter** field. Click the **Finish** button.



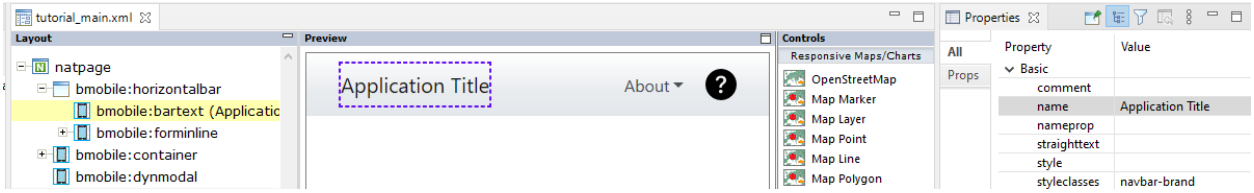
The **tutorial_main** layout is opened in **Layout Painter Editor**. It contains an application bar with a text at the left-hand side and a drop-down button and an icon at the right.

The Application Bar

In this step the controls in the application bar will be explained and we will adapt the application bar.

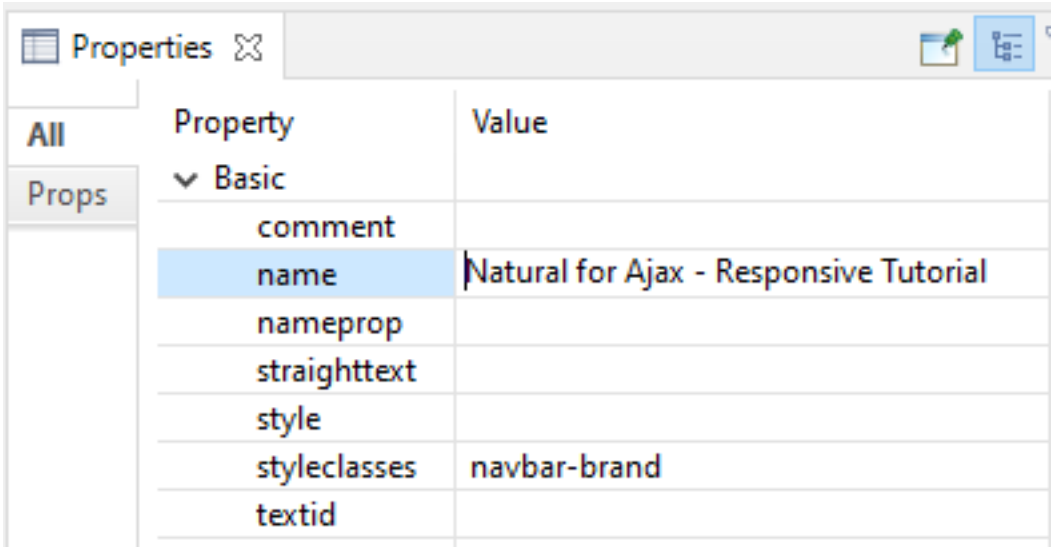
You see a **BMOBILE:HORIZONTALBAR** control, which contains a **BMOBILE:BARTEXT**, a **BMOBILE:FORMINLINE** and a **BMOBILE:ICON** control.

Click on the text **Application Title** in the **Preview Area (middle)**. This will select the **BMOBILE:BARTEXT** control. You can now see and modify the properties of this control.



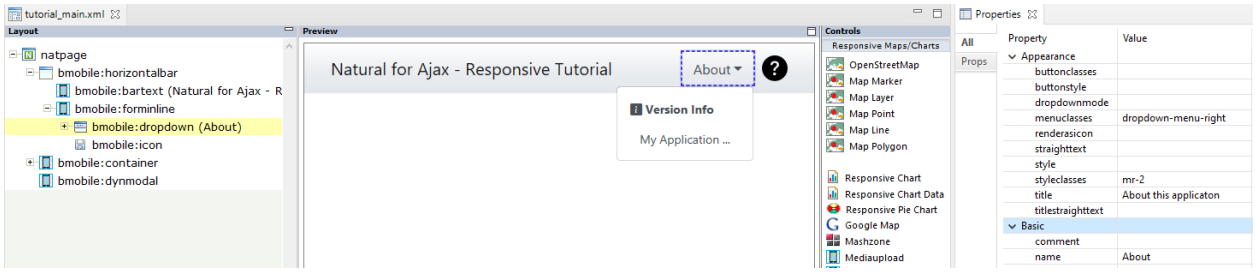
The **BMOBILE:BARTEXT** is a control to render texts in bars. In our example we want to add our application title as text here. You see that as **styleclasses** the css class **navbar-brand** is set. This increases the font-size and adds some shadow and padding. We keep this setting.

Our application name is "Natural for Ajax - Responsive Tutorial". Change the **name** property to **Natural for Ajax - Responsive Tutorial**.



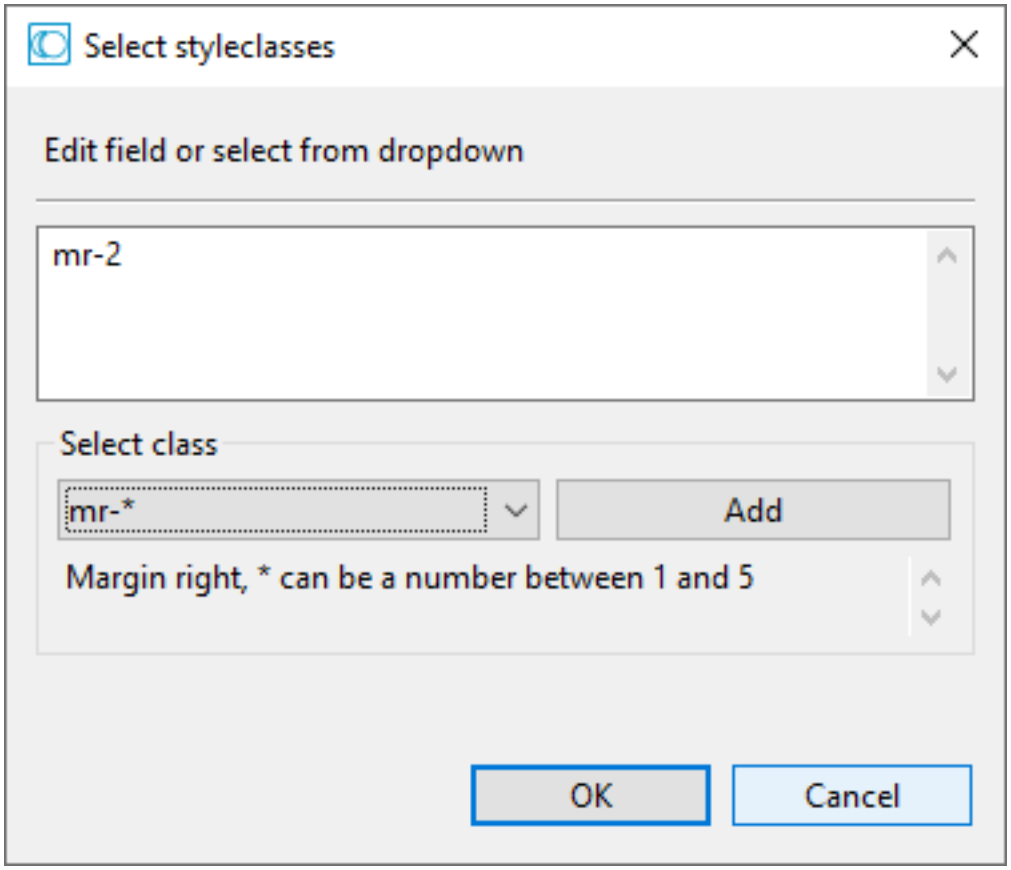
Press **<ctrl s>** to save the change. This will refresh the **Preview Area.**

Click on the **About** button in the **Preview Area (middle)**. In the **Layout Area (left)** a **BMOBILE:DROPDOWN** control is selected and you can now see and modify the properties of this control in the **Properties v** .



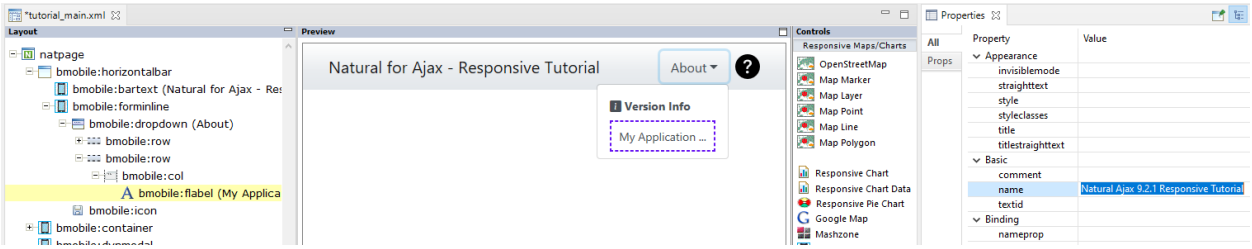
The **BMOBILE :DROPDOWN** and the **BMOBILE:ICON** control are grouped in a **BMOBILE:FORMINLINE** container control. Use a **BMOBILE:FORMINLINE** container whenever you want to render several form controls – like buttons, texts, fields, icons,... – side by side on a single horizontal row or bar.

Have a look at the properties of the **BMOBILE:DROPDOWN**: As **menuclasses** the css class **dropdown-menu-right** is set. This aligns the opened dropdown on the right side of the dropdown button. As **buttonclasses** property you can set classes to change the rendering of the dropdown button itself. We keep the default rendering. As **styleclasses** the css class **mr-2** is set. Click on the small button right to the text “mr-2”. This will open the “Select styleclasses” dialog. In the lower half of this dialog there is a dropdown box. This dropdown contains the most frequently used responsive classes for this control. Select mr-*. Now you can see a help text below the dropdown which explains the class.

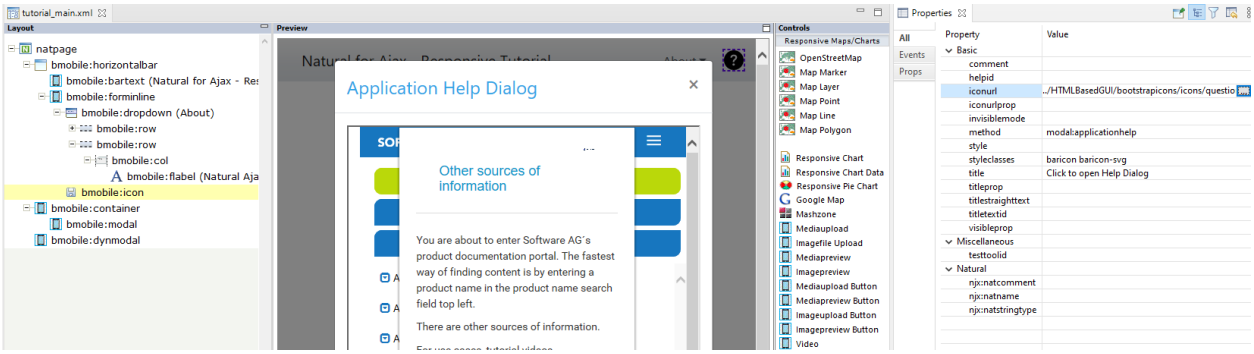


Click the **Cancel** button to close the dialog. In our example the mr-2 class puts a vertical distance between the dropdown button and the icon.

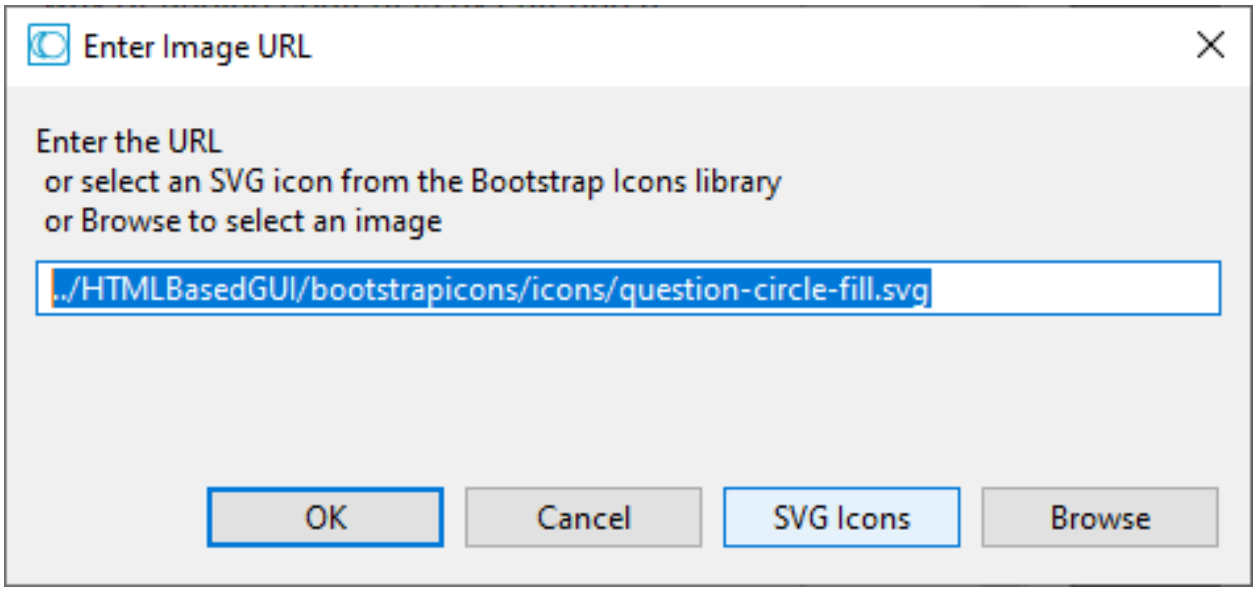
In the **Preview Area** click on the text **My Application** of the opened dropdown. This will select a **BMOBILE:FLABEL** control. In a **BMOBILE:DROPDOWN** control you can have rows with all kinds of controls. We simply apply a different text to the name attribute.



Press **<ctrl s>**. In the **Preview Area** click on the icon to the right of the dropdown. This will open a modal dialog in the **Preview Area** and will show the properties of the **BMOBILE:ICON** control in the **Properties** view.

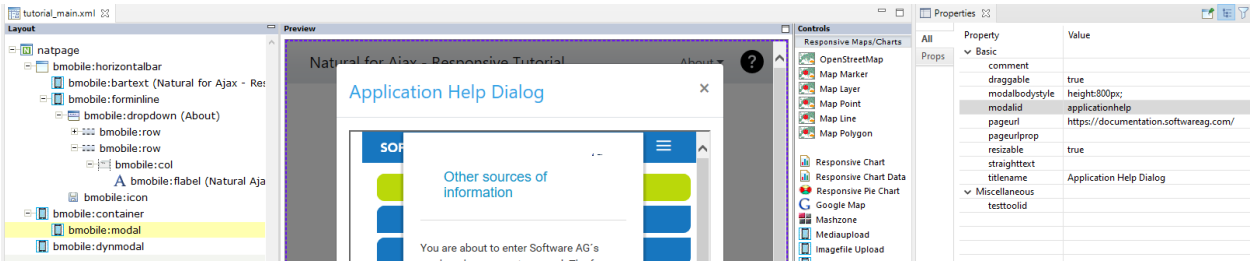


The **iconurl** property sets the icon image. Natural for Ajax contains an icon library with SVG icons and icon fonts. Click on the small button next to the property value "../HTMLBasedGUI/bootstrapicons/icons/question-circle-fill.svg". The dialog **Enter Image URL** opens.



When clicking the **SVG Icons** button you can select among more than 1300 SVG icons. Click the **Cancel** button.

The value of the **method** property defines which modal is opened. Each modal has a **modalid**. The method to open the modal is always "modal:<themodalid>". To see the corresponding **BMOBILE:MODAL** control with its properties, select it in the **Layout Area**.



Underneath the modalid is the **pageurl** property. Here you can set any URL to a valid page accessible as URL in your browser.

SUMMARY

We have built a simple application bar showing:

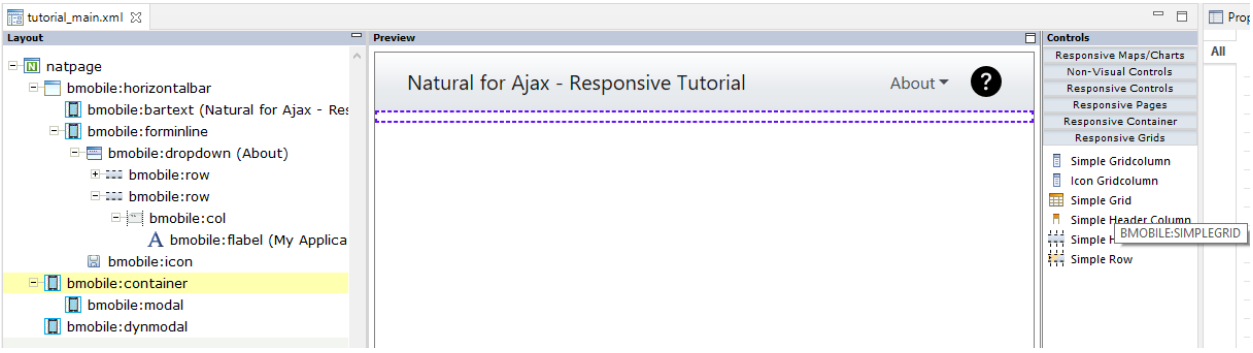
- Application Name
- Version Information (Dropdown)
- Help Icon with a Help Pop-Up.

We only had to adapt an existing template for this.

The Data Grid

We will use the controls palette to add a data grid with several different columns and behavior to our page.

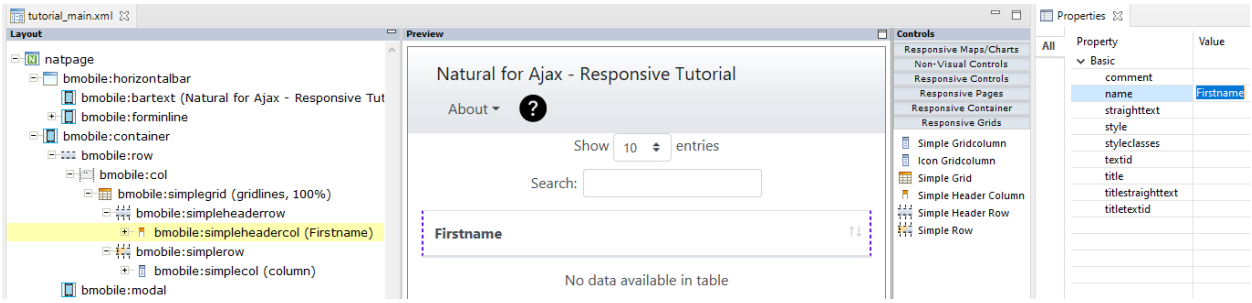
In the **Controls Palette (right)** open the **Responsive Grids** section. Drag and drop the **Simple Grid** to the bmobile:container node in the **Layout Area (left)**.



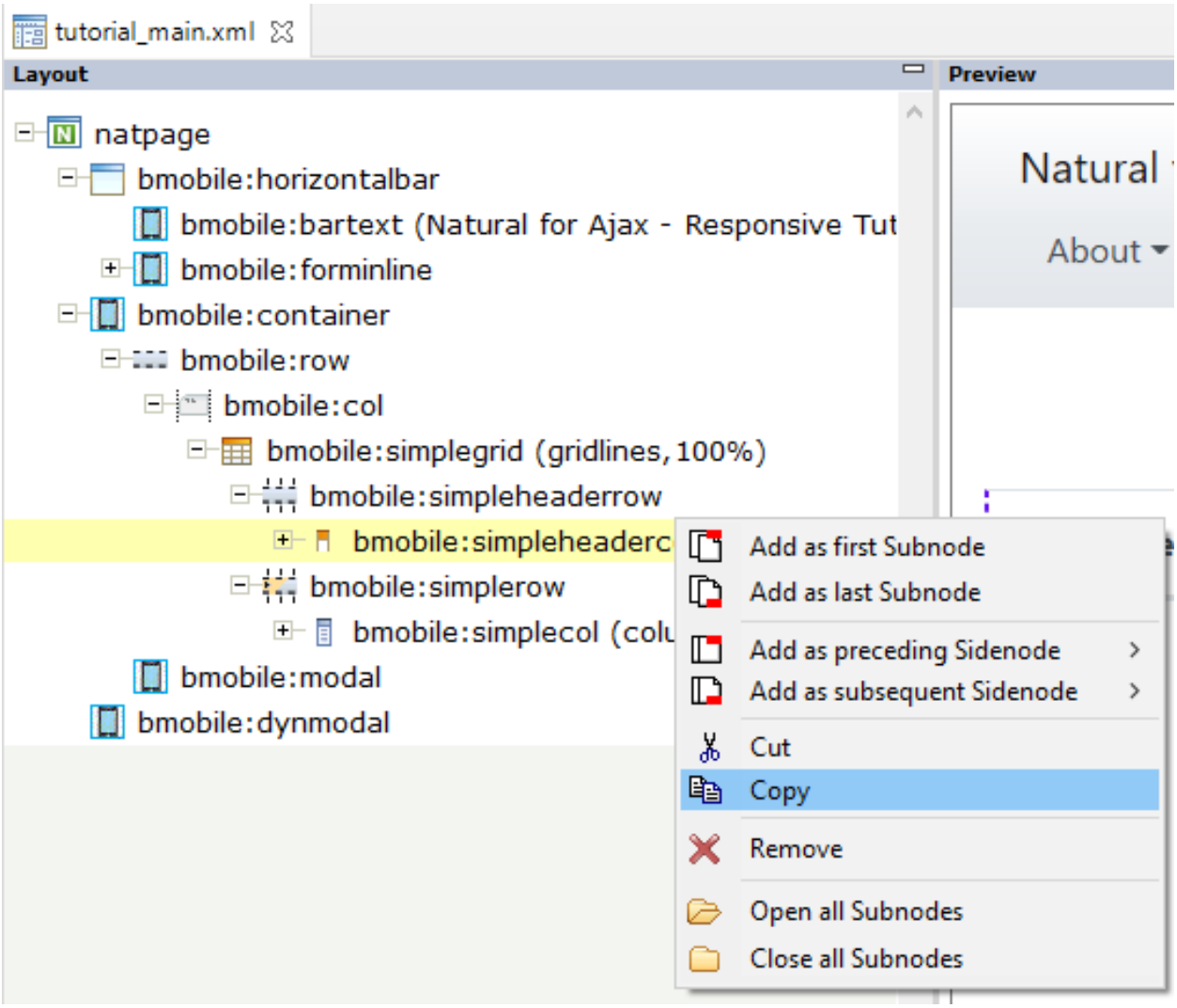
Press **<ctrl s>**. Open all child nodes of the bmobile:simplegrid node.

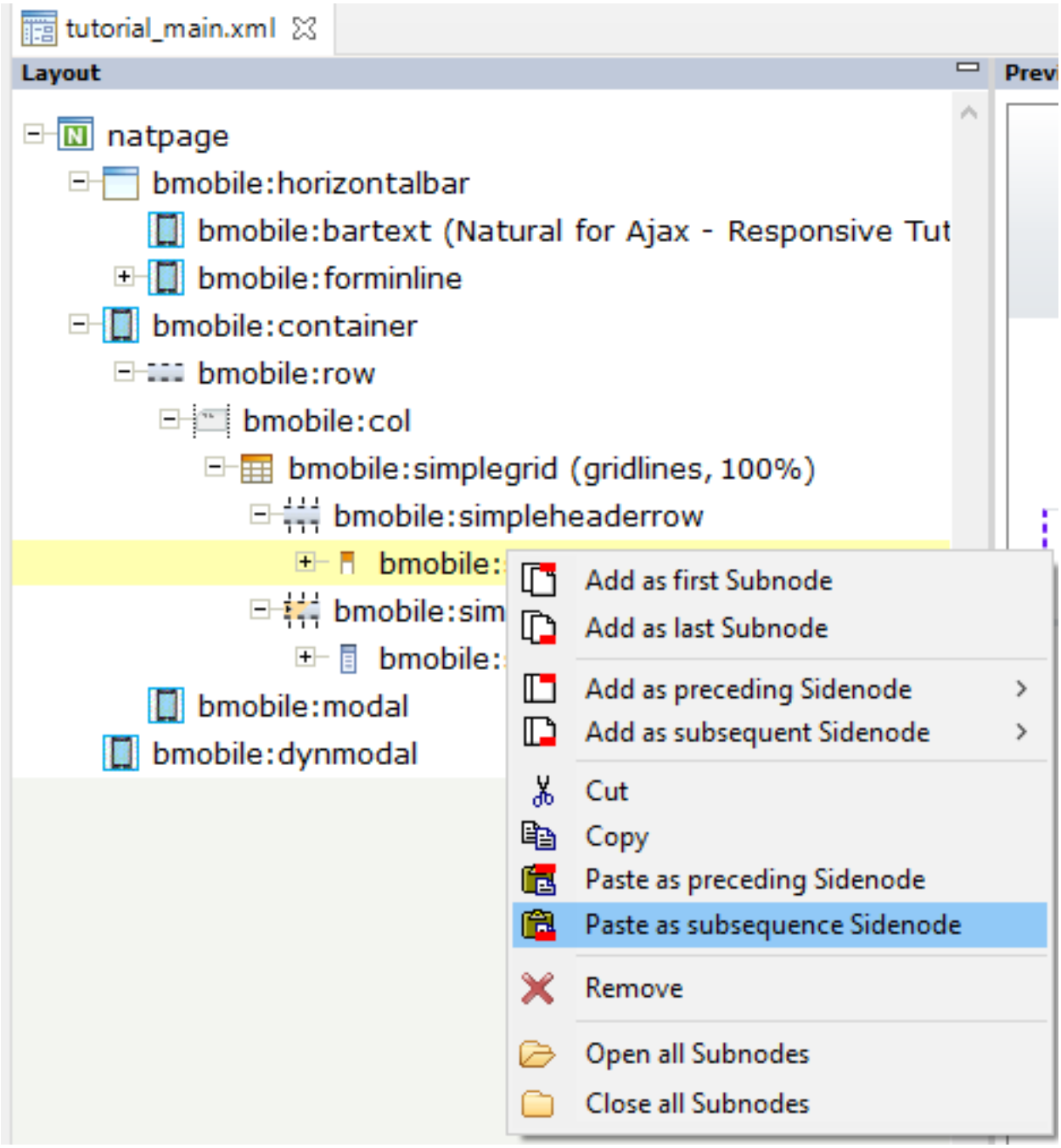
The BMOBILE :SIMPLEGRID control contains a row with grid header columns (**BMOBILE:SIMPLEHEADERROW**) and a row with data columns (**BMOBILE:SIMPLEROW**). Click on the

bmobile:simpleheadercol node and change the value of the property name to Firstname. Press **<ctrl s>**.

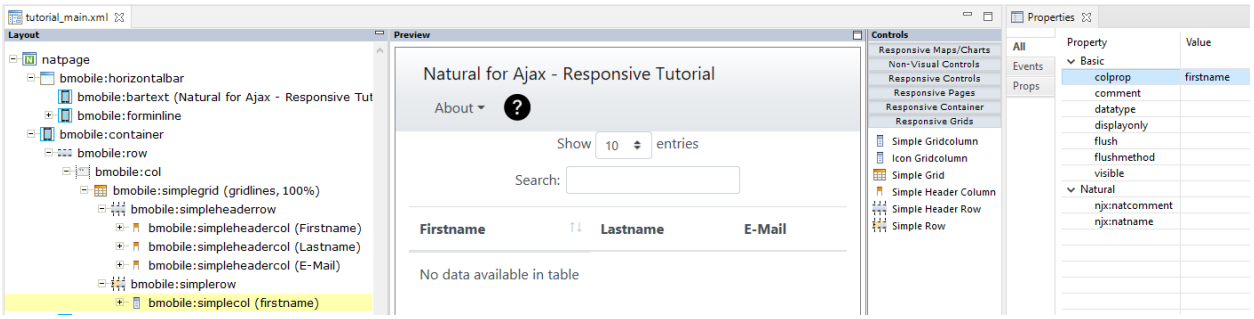


Copy and paste the bmobile:simpleheadercol twice

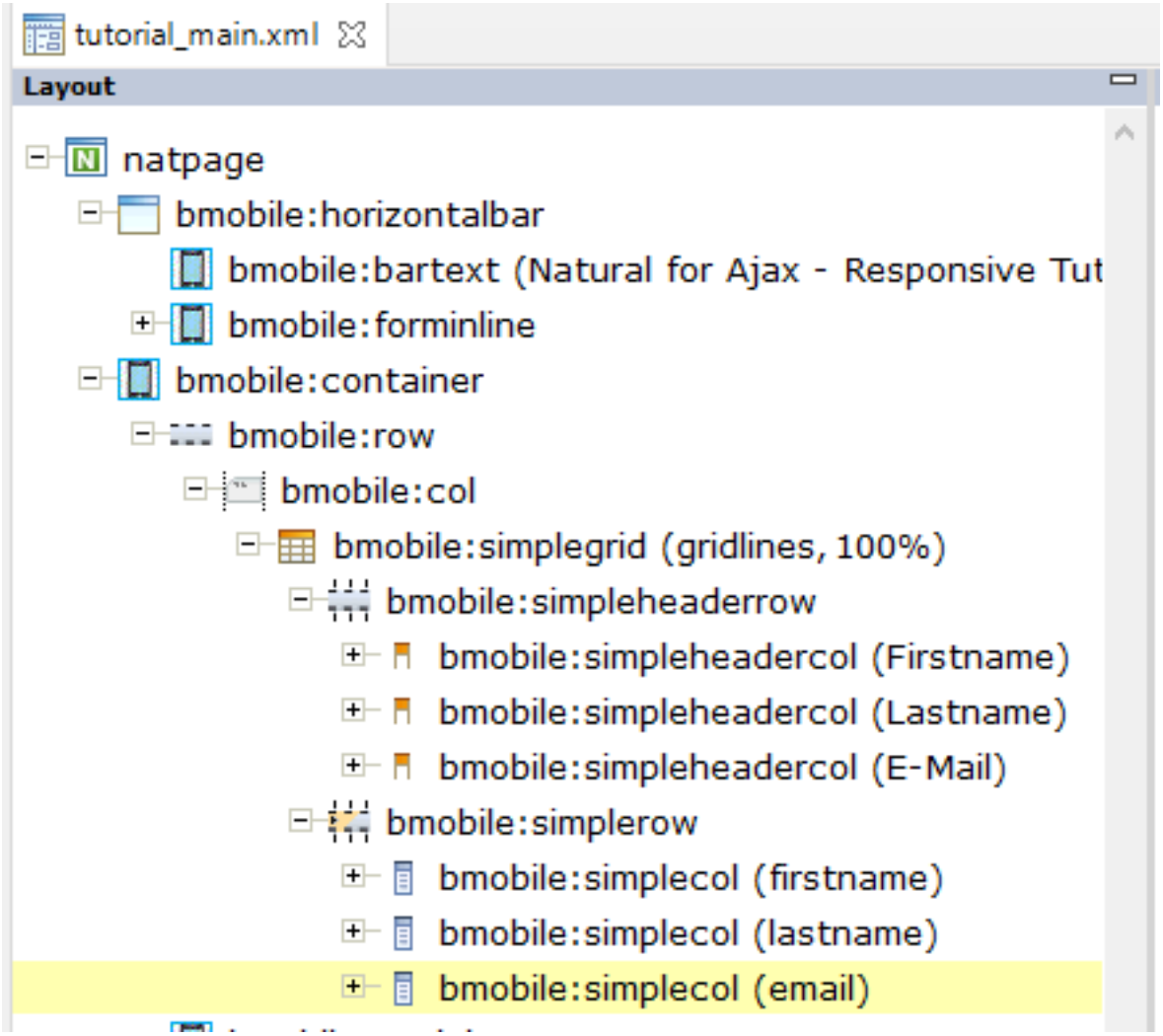




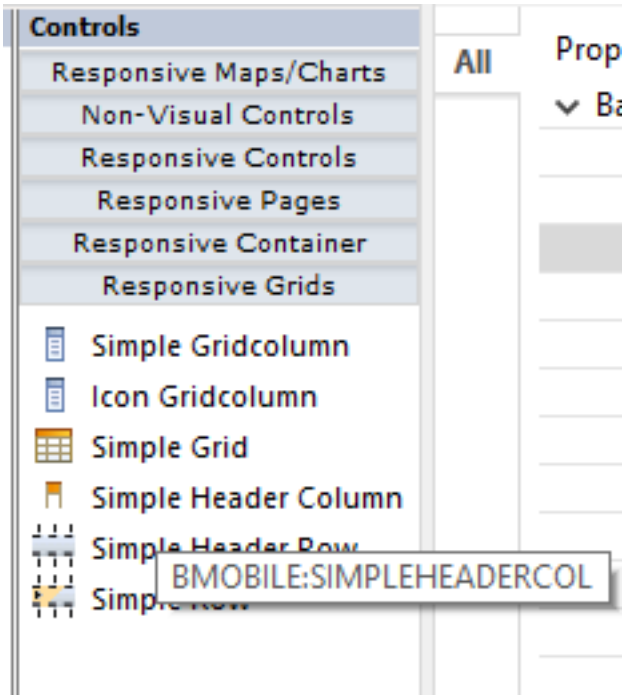
Modify the property name so that you have three header columns: **Firstname**, **Lastname**, **E-Mail**. Press **<ctrl s>**. In the **Layout Area** click on the node bmobile:simplecol. The **colprop** property is the Natural fieldname for your data column. Change it to **firstname**.



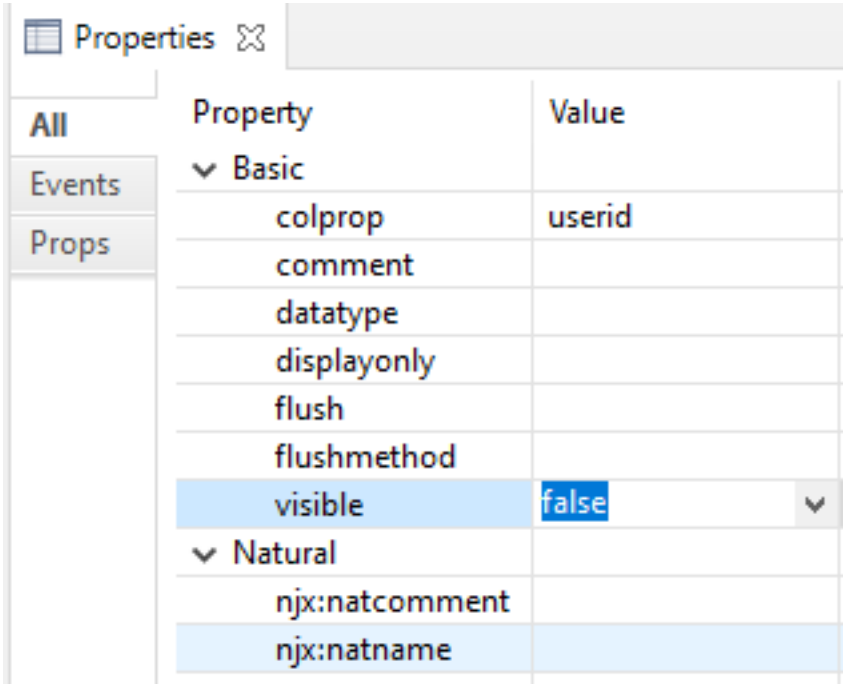
Copy and paste the bmobile:simplecol twice and change the colprop correspondingly so that you see:



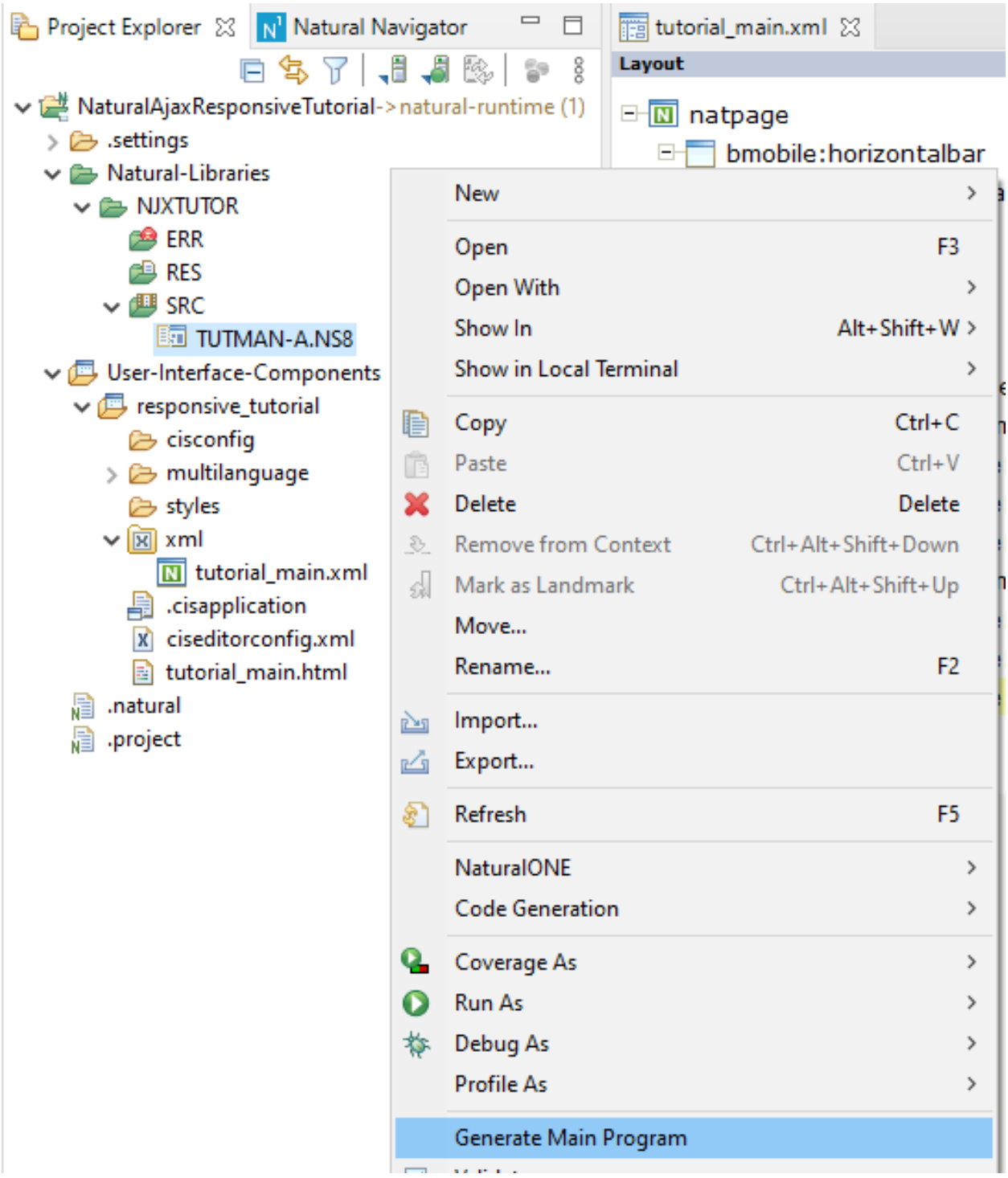
We now add an additional invisible id column. Drag and drop a **Simple Header Column** from the **Controls palette** and leave the **name** empty.



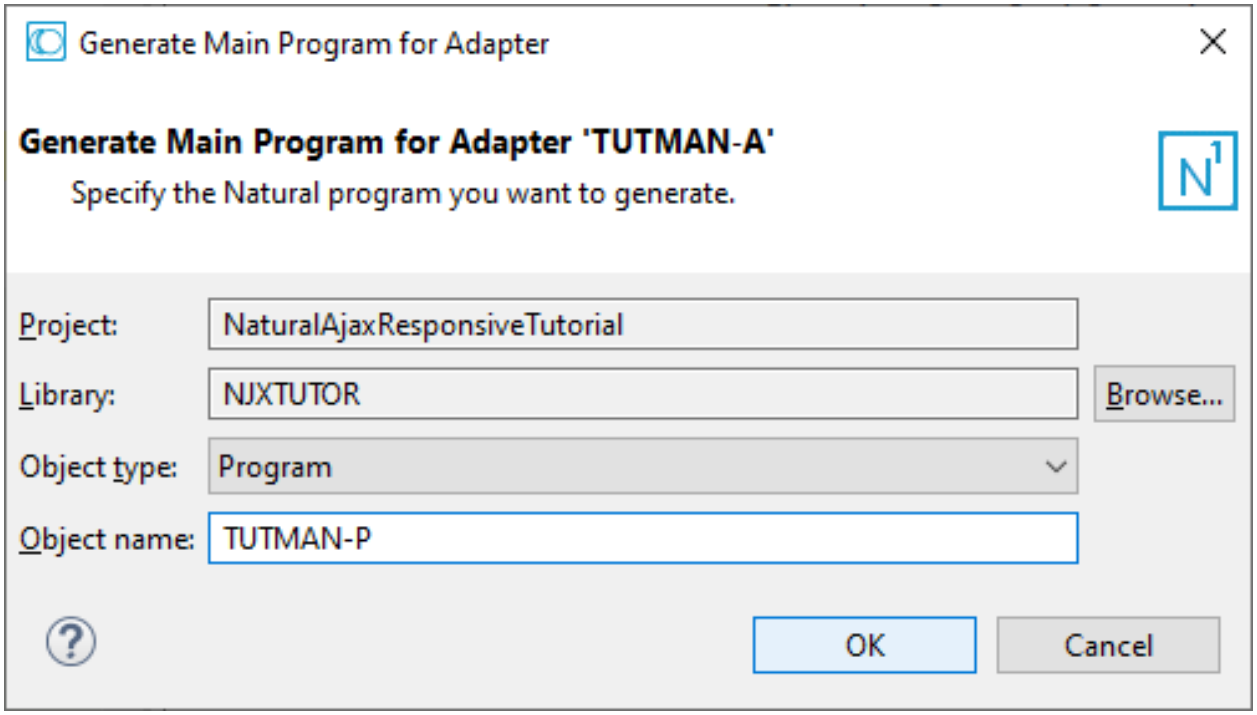
Drag and drop a **Simple Gridcolumn** from the **Controls palette** and apply the following properties:



Press **<ctrl s>**. Whenever a layout is saved, the corresponding Natural adapter is updated. We will now generate a Natural program from the adapter. In **Project Explorer** right click on TUTMAN-A.NS8 and select **Generate Main Program**.



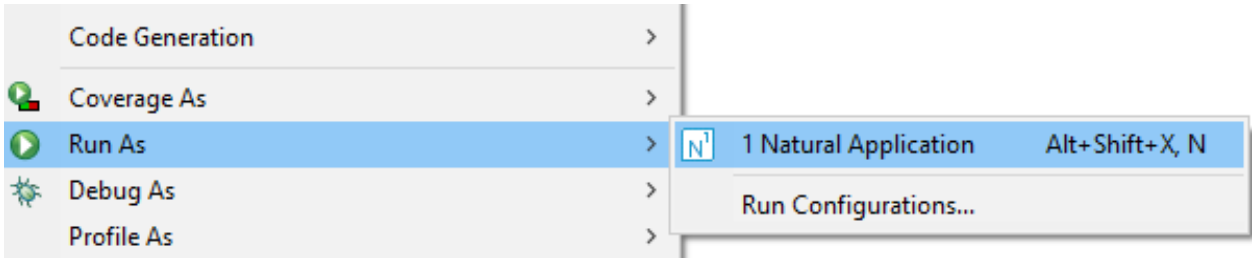
In the dialog add **TUTMAN-P** as **Object name** and click the **OK** button.



Fill some items into your grid. Example code:

```
…
1 #II (I2)
END-DEFINE
EXPAND ARRAY GRIDLINES TO (1:30)
FOR #II:=1 TO 30
COMPRESS 'First ' #II INTO GRIDLINES.FIRSTNAME(#II) LEAVING NO
COMPRESS 'Last ' #II INTO GRIDLINES.LASTNAME(#II) LEAVING NO
COMPRESS 'First.Last' #II '@email.com' INTO GRIDLINES.EMAIL(#II) LEAVING NO
COMPRESS 'User' #II INTO GRIDLINES.USERID(#II) LEAVING NO
END-FOR
PROCESS PAGE USING "TUTMAN-A"
…
```

Right click and select **Run As** > **Natural Application**



SUMMARY

We have developed a data grid with visible and invisible data columns and corresponding header columns, added the Natural code to fill the grid and ran our main application page.

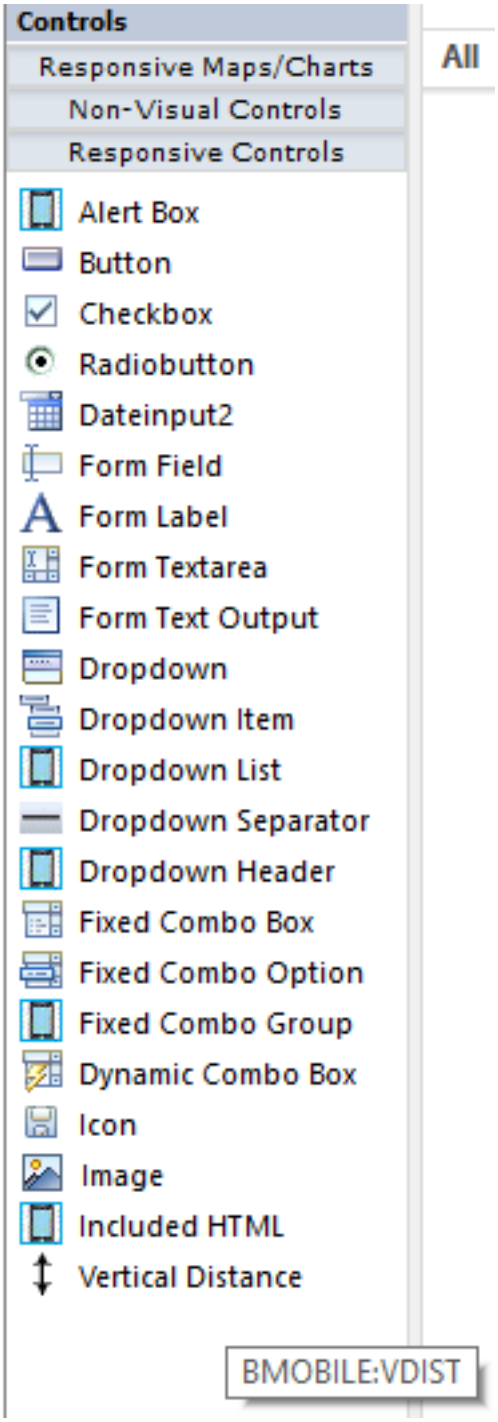
Adding Responsive Spacing and Visibility

If you look at your running page in the browser, you will notice that some parts of the rendering do not look good. For instance, there is no space between the application bar and your grid. If you are not familiar with the Bootstrap grid system, take some minutes to read the chapter *Responsive Pages Basics*.

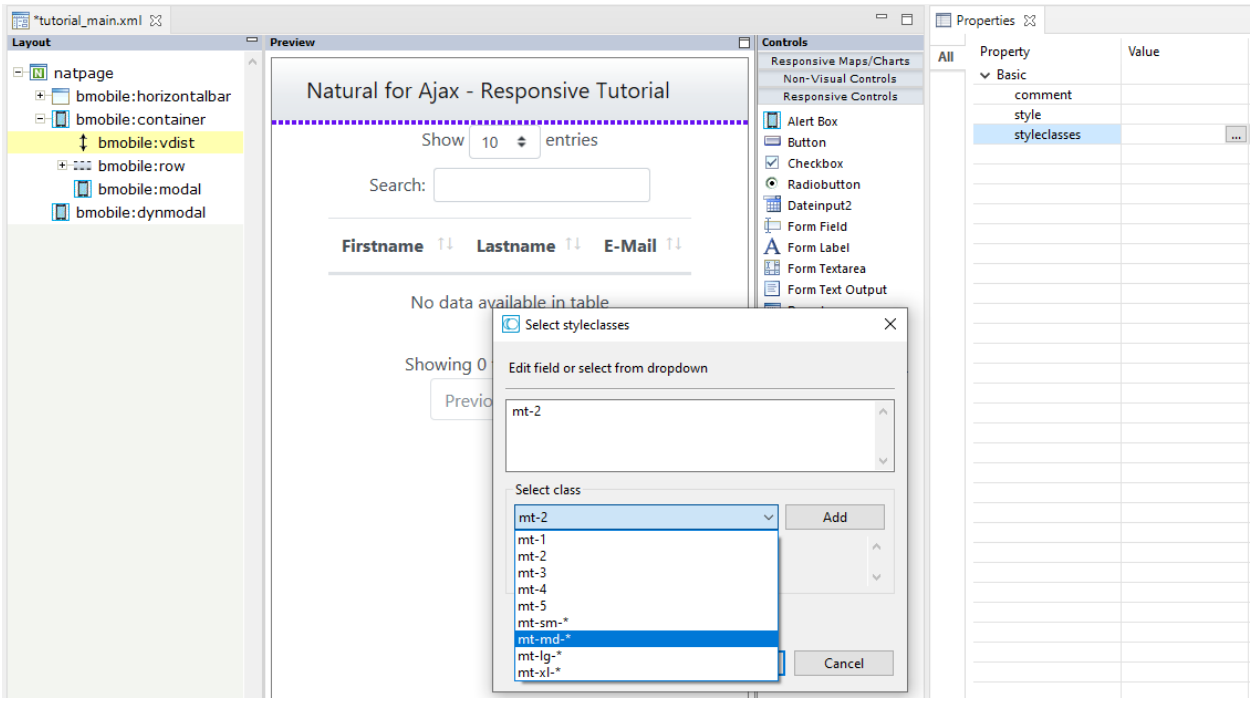
- Adding Vertical Distance
- Adding Horizontal Distance
- Device Dependent Showing/Hiding of Controls

Adding Vertical Distance

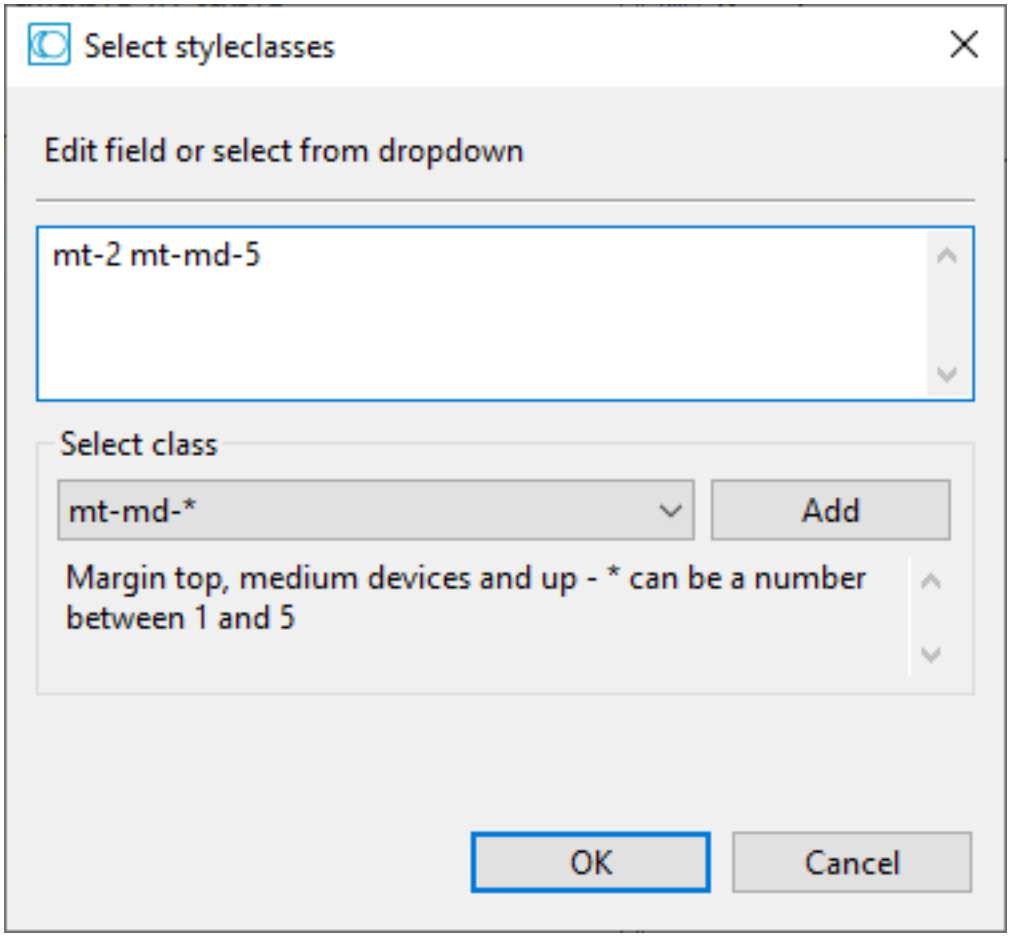
Open **tutorial_main.xml** in **Layout Painter**. In the **Layout Area (left)** select the **bmobile:container** node. Drag and drop a **Vertical Distance** control from the **Controls palette (right)** under the bmobile:container node as first sub node.



In the **Properties** view click the small button right to the styleclasses property. We will now add a device dependent vertical distance: For smaller devices we add a smaller distance than for larger devices. In the dropdown select **mt-2** and click the **Add** button. Now select **mt-md-*** and press the **Add** button.



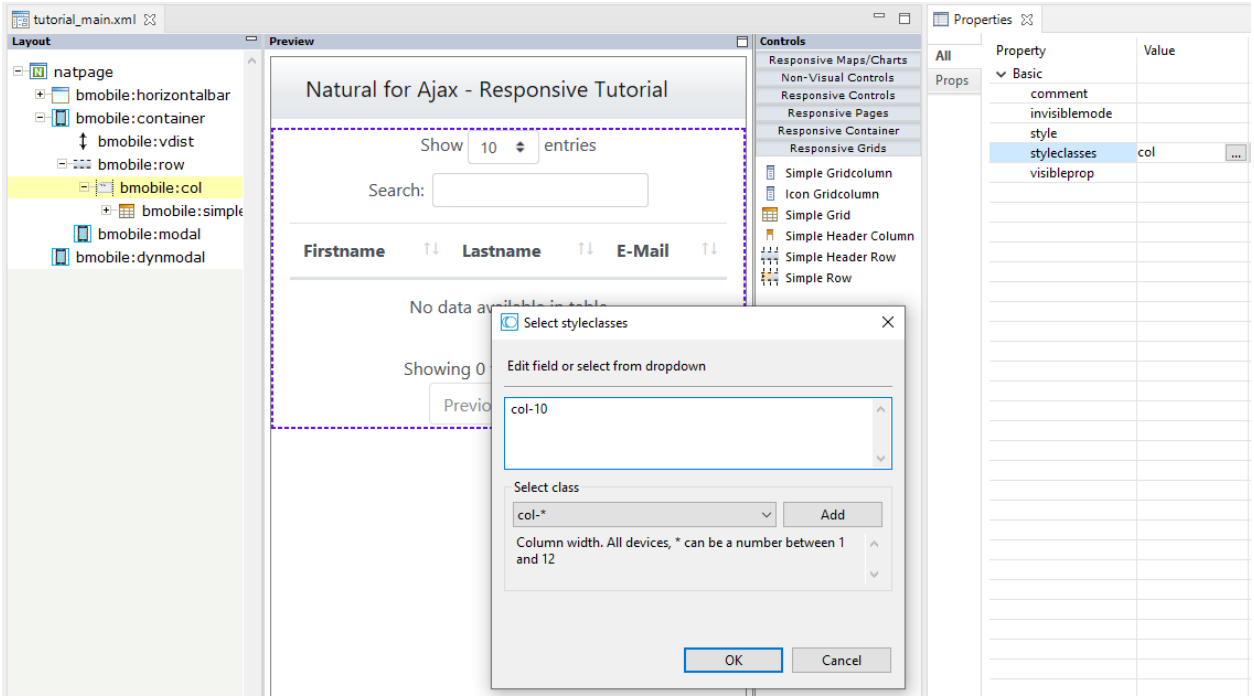
In the field change **mt-md-*** to **mt-md-5** and click the **OK** button.



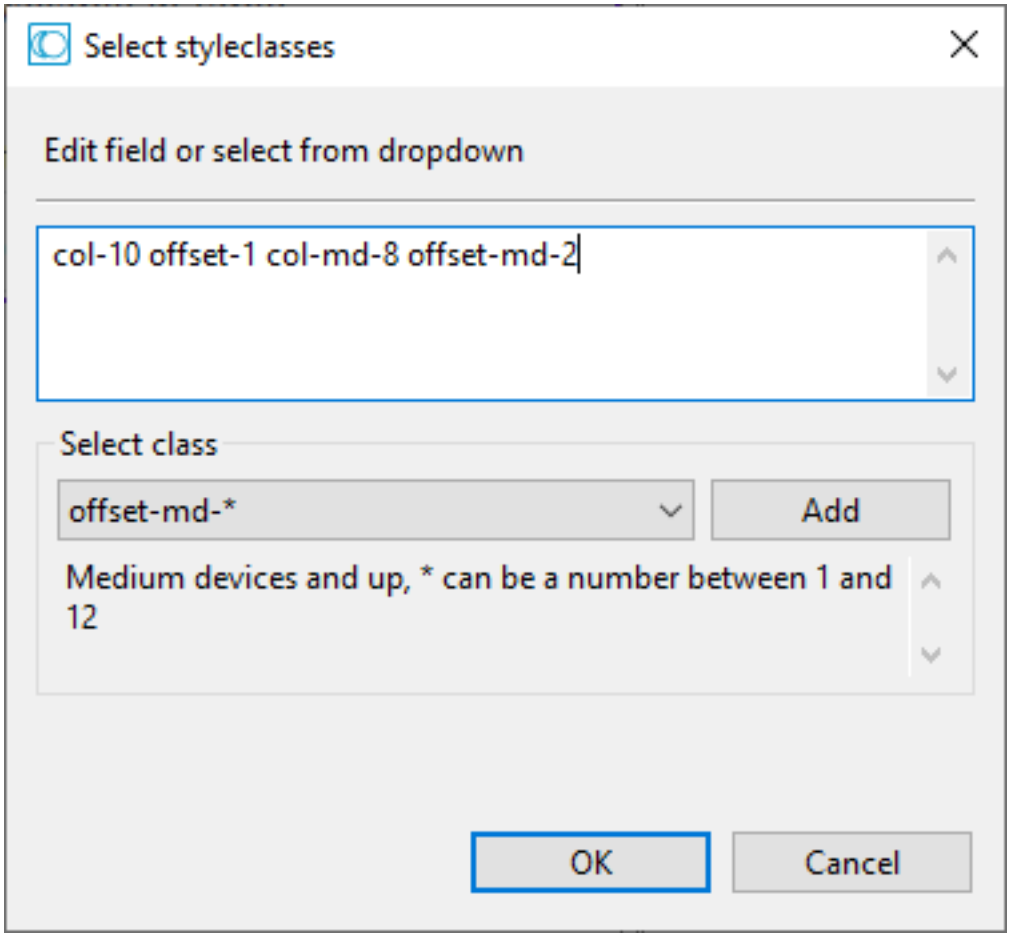
What we have done means the following: For all devices we set mt-2 sets a distance of about 8px. For medium devices and up we overwrite this default setting with mt-5 which sets a distance of about 48px.

Adding Horizontal Distance

In the **Layout Area (left)** select the bmobile:col node above the bmobile:simplegrid node. In **Properties view** click the small button right to the styleclasses property. We will now apply a device dependent width and horizontal distance for our grid. Remove the col in the field. In the dropdown select **col-*** and click the **Add** button. In the field modify **col-*** to **col-10**.



In the same way add **col-offset-1**, **col-md-8** and **col-offset-md-2**.



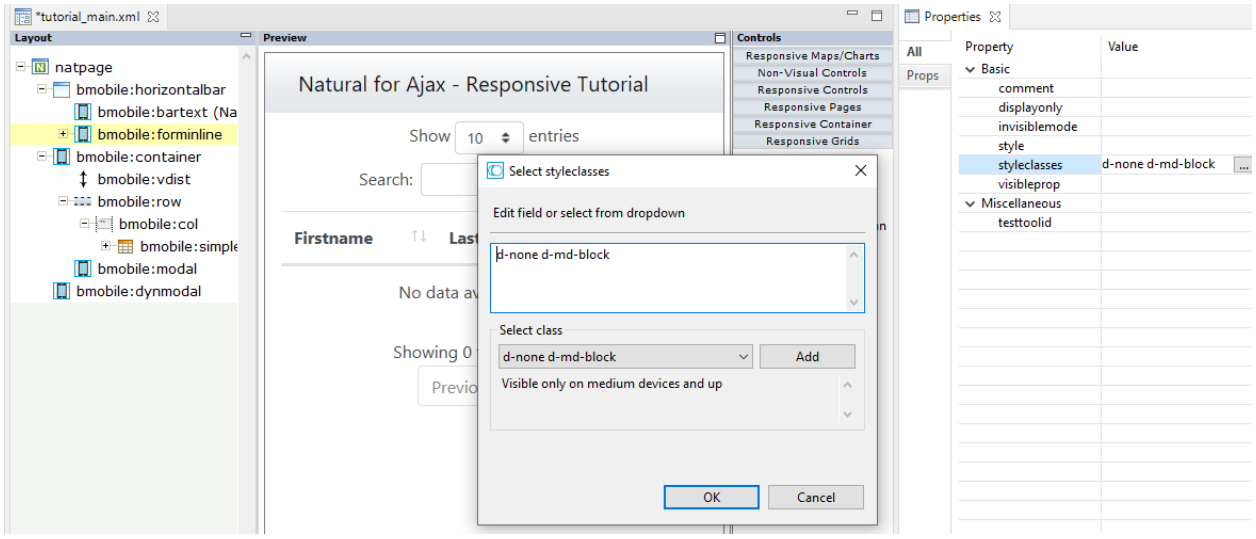
Click the **OK** button and press **<ctrl s>**.

What we have done means the following: For all devices our BMOBILE:SIMPLEGRID takes 10 of the 12 Bootstrap grid columns. We set a horizontal distance of 1 Bootstrap grid column. For medium devices and up we overwrite this default setting so that our BMOBILE:SIMPLEGRID takes 8 of the 12 Bootstrap grid columns and has a horizontal distance of 2 Bootstrap grid columns.

Device Dependent Showing/Hiding of Controls

On larger devices you can show more information than on smaller devices. For our application bar we only want to show the About button and the Help icon for larger devices. This makes sure that our application bar is not “wrapped”.

In the **Layout Area (left)** select the bmobile:forminline node. Apply the **styleclasses d-none d-md-block**.



Now the BMOBILE :FORMINLINE container with its sub controls is only shown for medium devices and up.

Be sure you pressed **<ctrl s>**. Run your **TUTMAN-P** program, resize your browser window and see how the rendering adapts to the size of the browser window.

SUMMARY

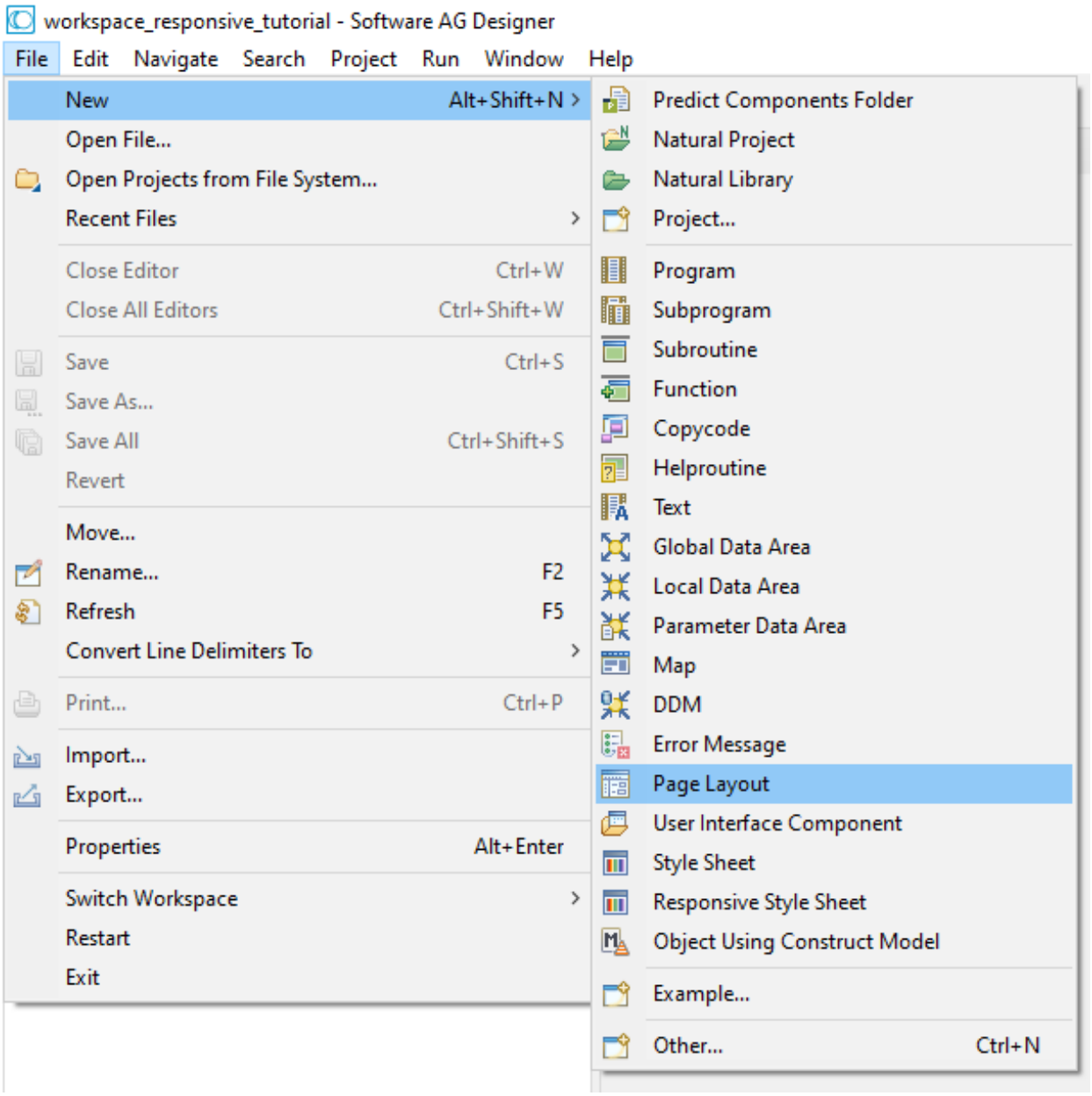
We have applied device dependent spaces (vertical and horizontal) and device dependent showing/hiding of controls by simply applying css styleclasses to our layout.

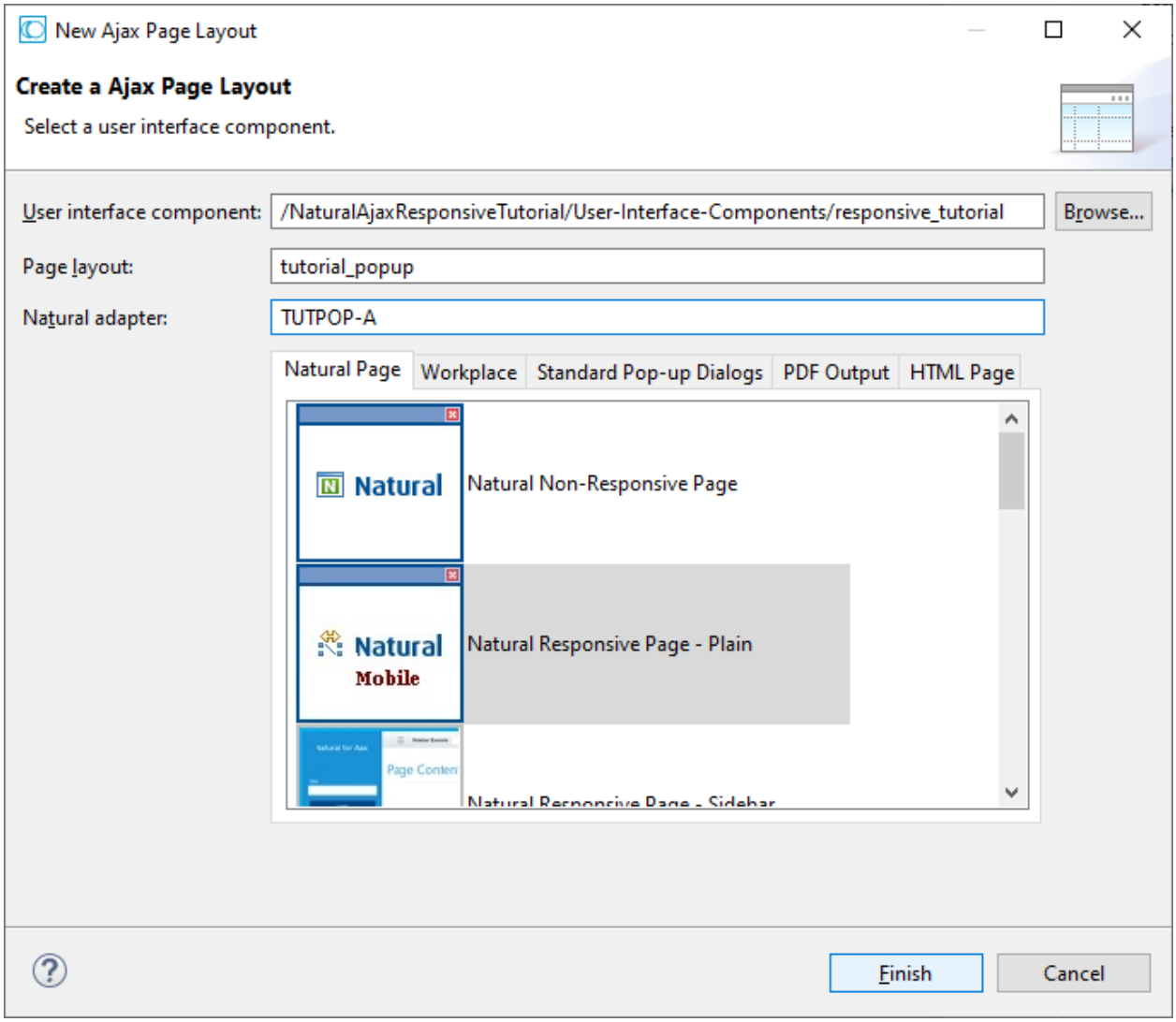
13 PART2: Develop the Pop-Up Page

In this part we will develop a page with a responsive form. We will prepare everything so that we can open the page also as a pop-up from our main application page.

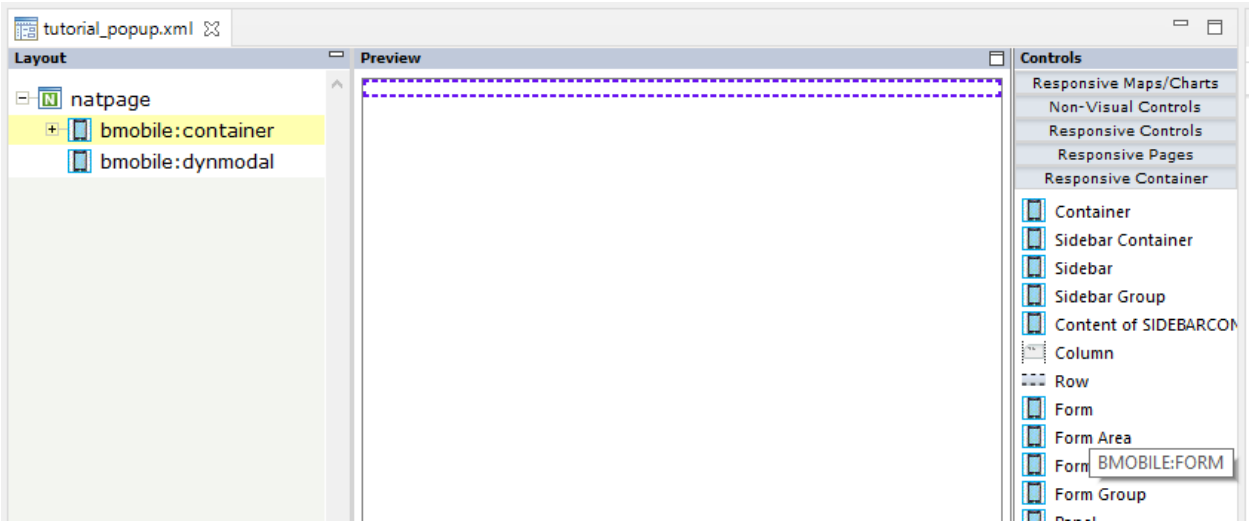
Create a Responsive Form

In the File menu click New > User Interface Component, select the **Natural Responsive Page - Plain** template and enter layout and adapter name as shown below.



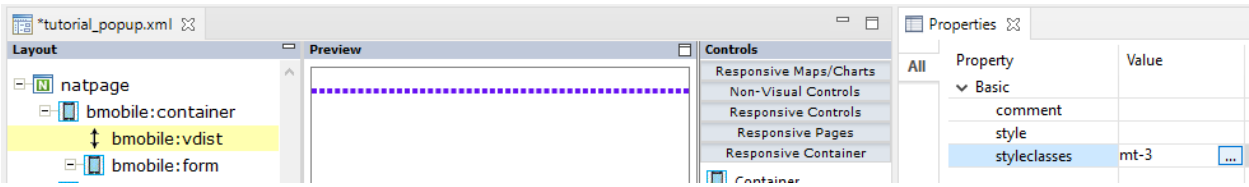


Click the **Finish** button. Drag and drop a **Form** container from the **Controls palette (right)** under the bmobile:container node.



We want a little bit spacing left and right of our form. In the **Properties** view set mx-3 as **styleclasses** as described in PART1 of this tutorial. The "x" stands for "left and right".

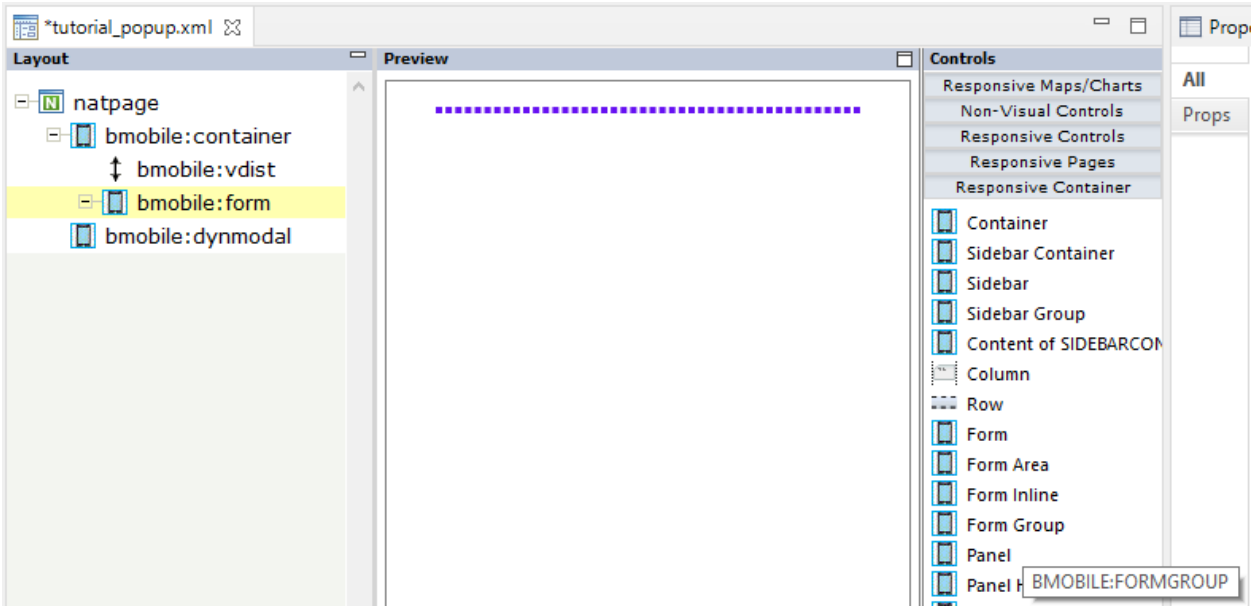
Drag and drop a **Vertical Distance** control from the **Responsive Grids** section of the **Controls palette (right)** under the bmobile:container node as first sub node. In the **Properties** view set mt-3 as **styleclasses**.



Tip: Don't forget to press **<ctrl s>** after changes to refresh the Preview Area (middle).

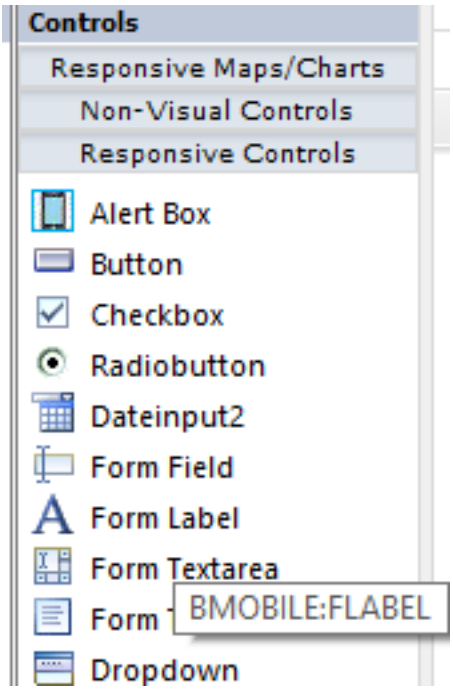
A **Form** is a container for different types of input elements, such as: text fields, checkboxes, radio buttons, submit buttons, etc. Use Form Group containers inside a form to get some grouping of labels, inputs, validation etc.

Drag and drop a **Form Group** container from the **Controls palette (right)** under the bmobile:form node.

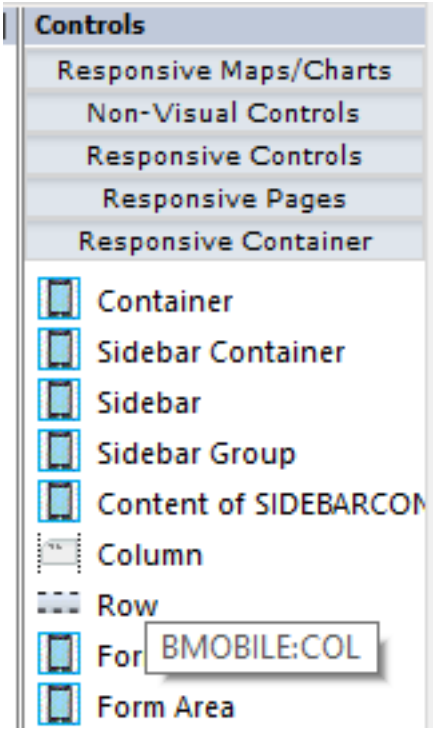


On larger devices we want to render the label and the input control for a value side by side. For smaller devices we want to render the label above the input control. To do so we set the property **renderasrow** to **true**. Then we can use the Bootstrap grid system with its **col*** classes to define when to render side-by-side and when to break the row.

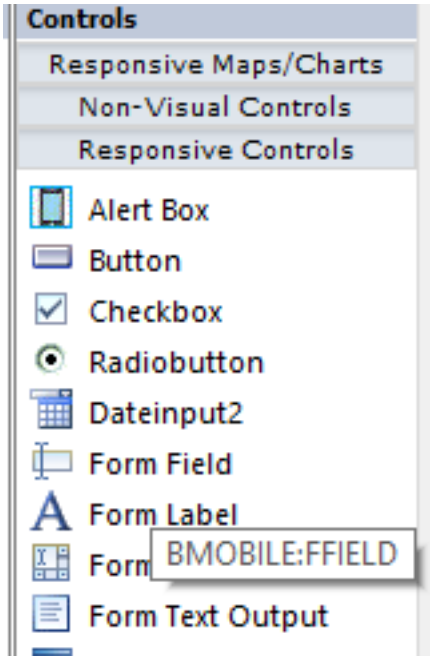
Drag and drop a **Form Label** control from the **Controls palette (right)** under the bmobile:formgroup node.



Drag and drop a **Column** container from the **Controls palette (right)** under the bmobile:formgroup node – after the bmobile:flabel node.



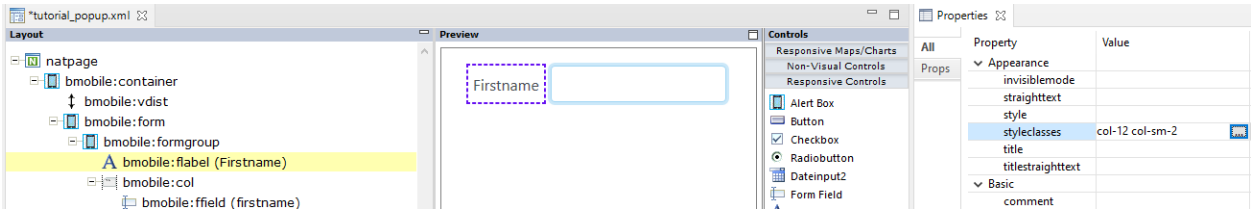
Drag and drop a **Form Field** control from the **Controls palette (right)** under the bmobile:col node.



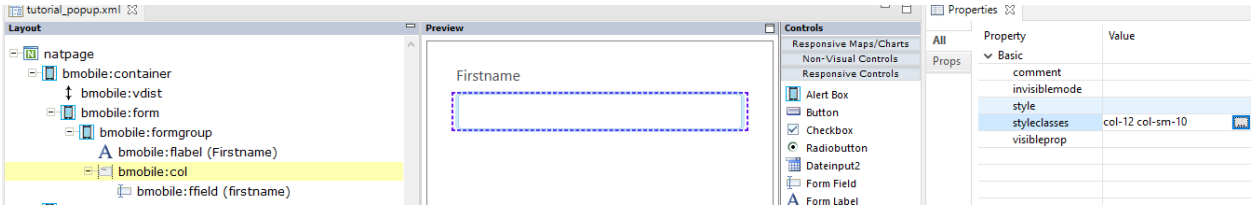
Select the bmobile:flabel node. In the **Properties** view apply **Firstname** for the **name** property.

Select the bmobile:ffield node. In the **Properties** view apply **firstname** for the **valueprop** property. This will be the name used for the corresponding Natural field. It must start with a lower-case character, but in your Natural program the field will be upper-case.

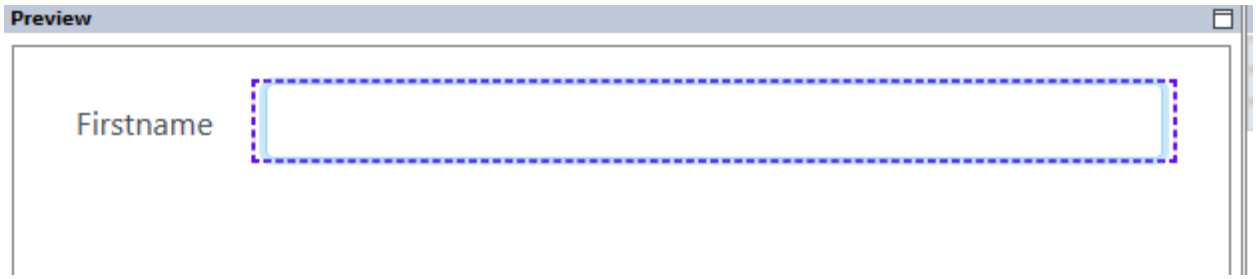
Select the bmobile:flabel node. In the **Properties** view apply **col-12 col-sm-2** for the styleclasses property. This means: It should span the complete row on very small devices. But it should only span 2 columns on a little bit larger devices and up.

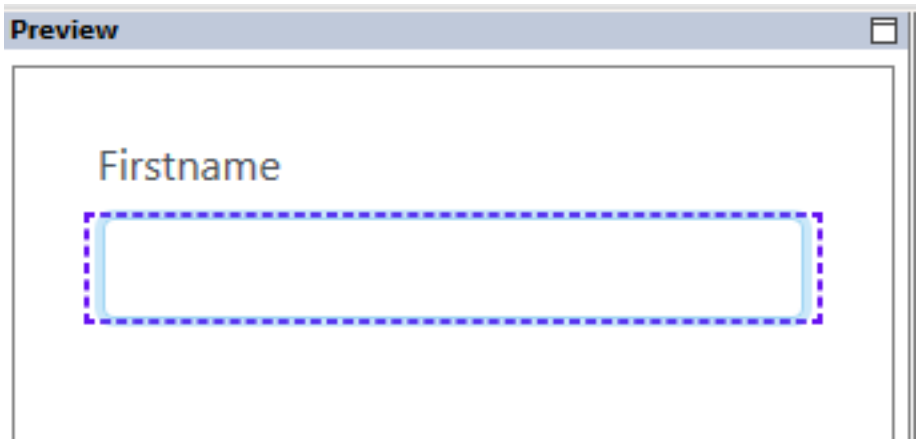


Select the bmobile:flabel node. In the **Properties** view apply **col-12 col-sm-10** for the styleclasses property. This means: It should span the complete row on very small devices. But it should only span 10 columns on a little bit larger devices and up.

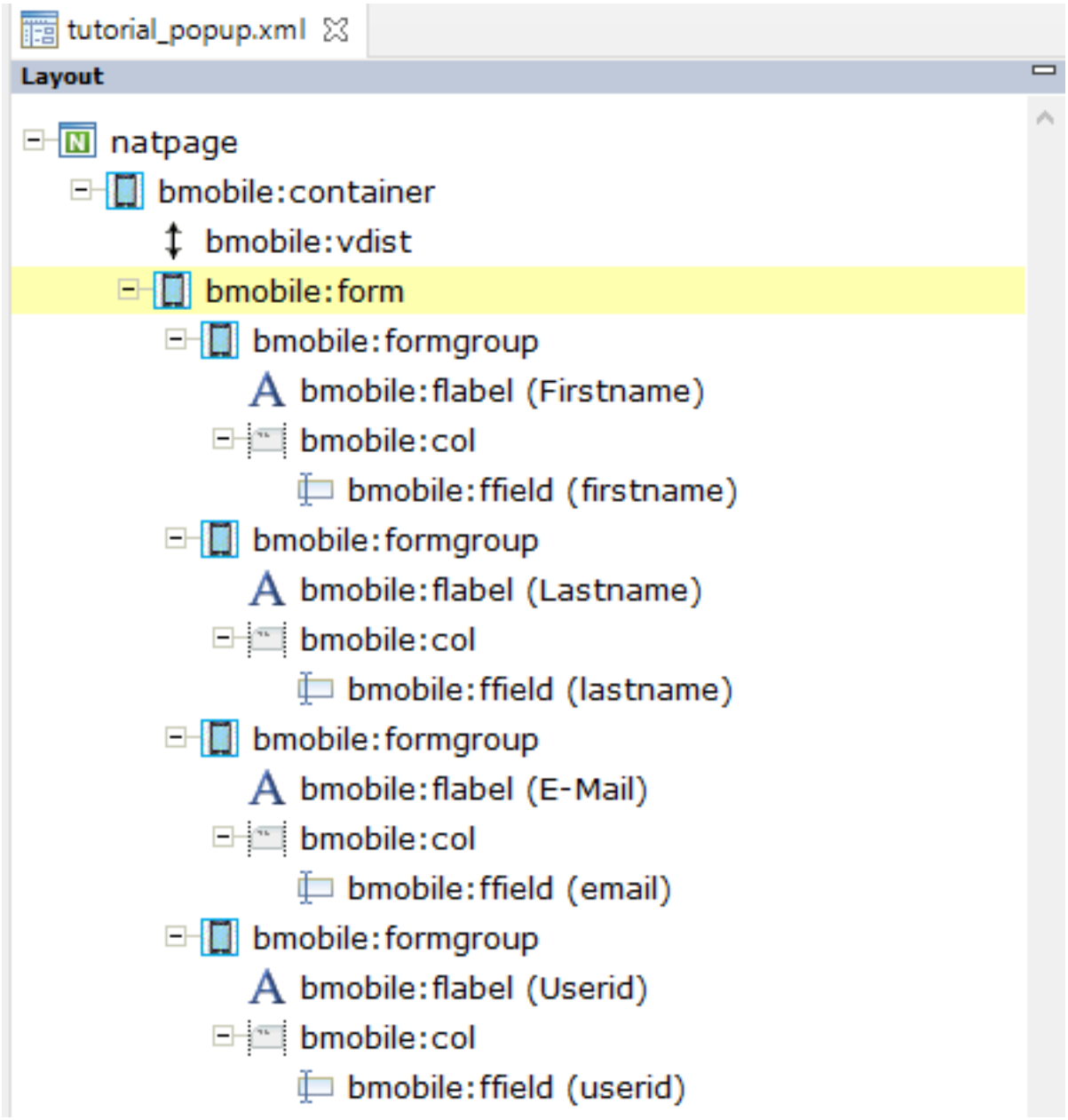


With this on a little bit larger devices, our form-group will be rendered in one row and on very small devices our form-group will break into two rows. Resize your Preview Area to see this effect.





In the same way add three more form groups. You can copy and paste the complete bmobile:formgroup node and adapt it correspondingly.



Add Placeholders, Validators and Displayonly Behavior

We want our firstname and lastname to be mandatory. Select the corresponding bmobile:ffield nodes. In **Properties** view set the property **required** to **true**.

Our userid should be displayonly. Select the corresponding bmobile:ffield node. In **Properties** view set the property **displayonly** to **true**.

We want that our email is validated for valid email values. Select the corresponding bmobile:ffield node. In **Properties** view set the property **email** to **true**.

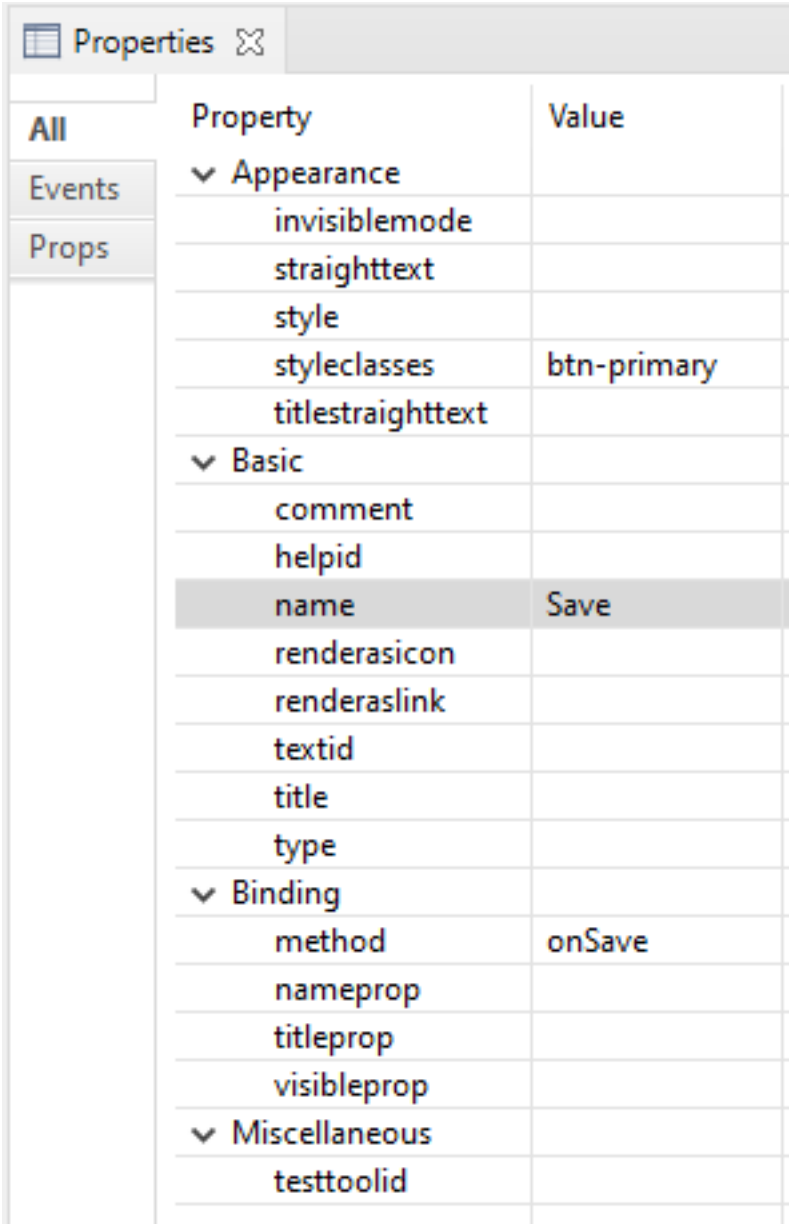
For empty fields we want to show a placeholder text. For the bmobile:ffield nodes of firstname, lastname and email, set the property **placeholder** to **Your firstname**, **Your lastname**, **Optional: Your E-Mail**.

Add Submit and Cancel Button

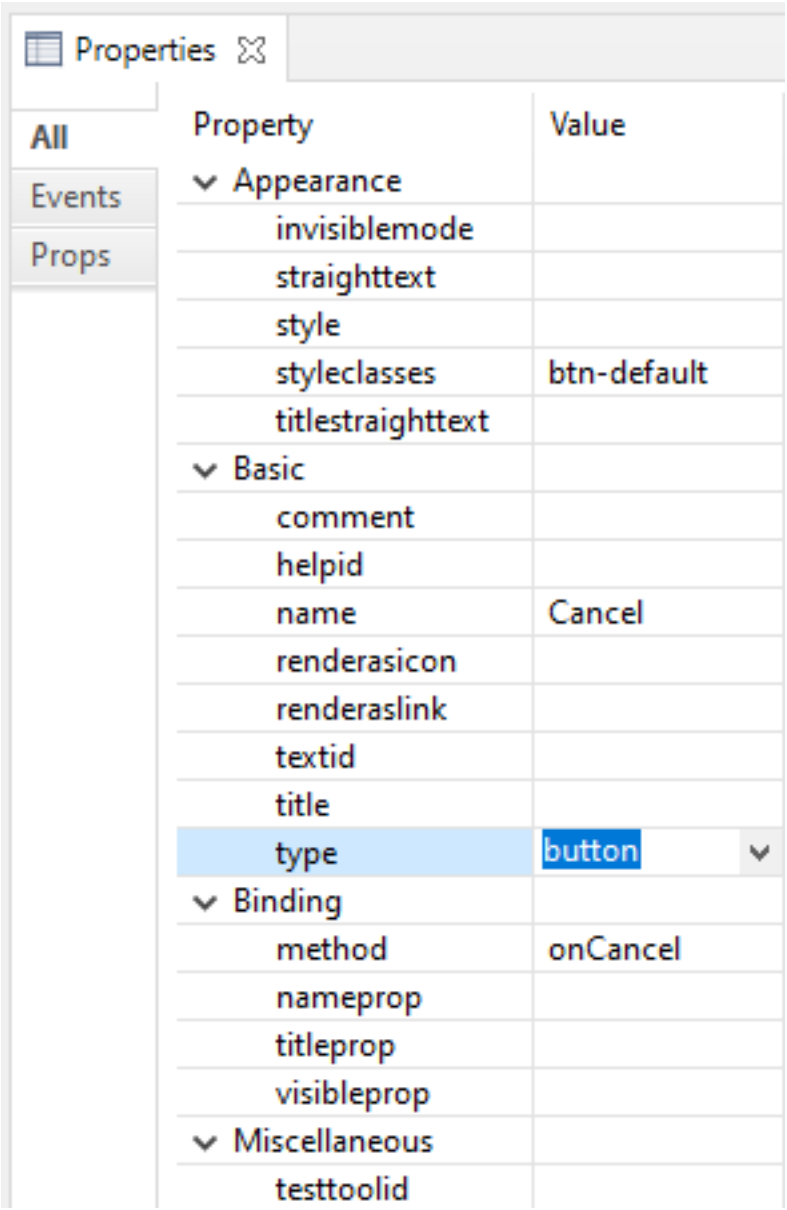
Add another **Form Group** as last sub node of our **bmobile:form** node. This will contain our buttons. Add a Vertical Distance before our new bmobile:formgroup node and set the **styleclasses** property to **mt-1**. We want just a little bit space between our fields and the buttons.

Add a **Column** as sub node of the new bmobile:formgroup and set the **styleclasses** property to mx-auto. We want our buttons to be centered.

Add a **Button** under the bmobile:col node and set the following properties:



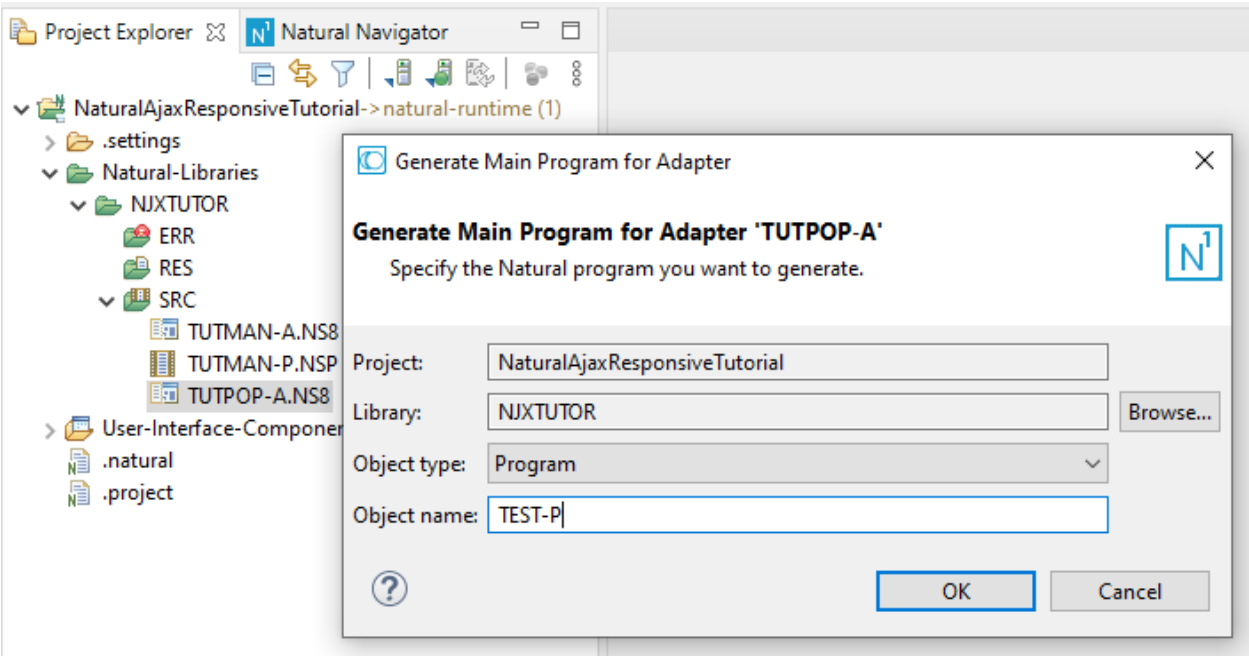
The **method** is the name of the event, which will be forwarded to your Natural program when the button is clicked. Add a second Button under the bmobile:col node as last sub node and set the following properties:



Be sure to set the property **type** to **button**. In difference to **submit** buttons (=default) this button will not validate the form. This allows you to close the pop-up even in case of invalid field values.

Running a Quick Test

From the generated adapter TUTPOP-A you can simply generate a Natural program and run the generated program – without any modifications. In PART3 of this tutorial we will integrate the form as pop-up into our main application.



SUMMARY

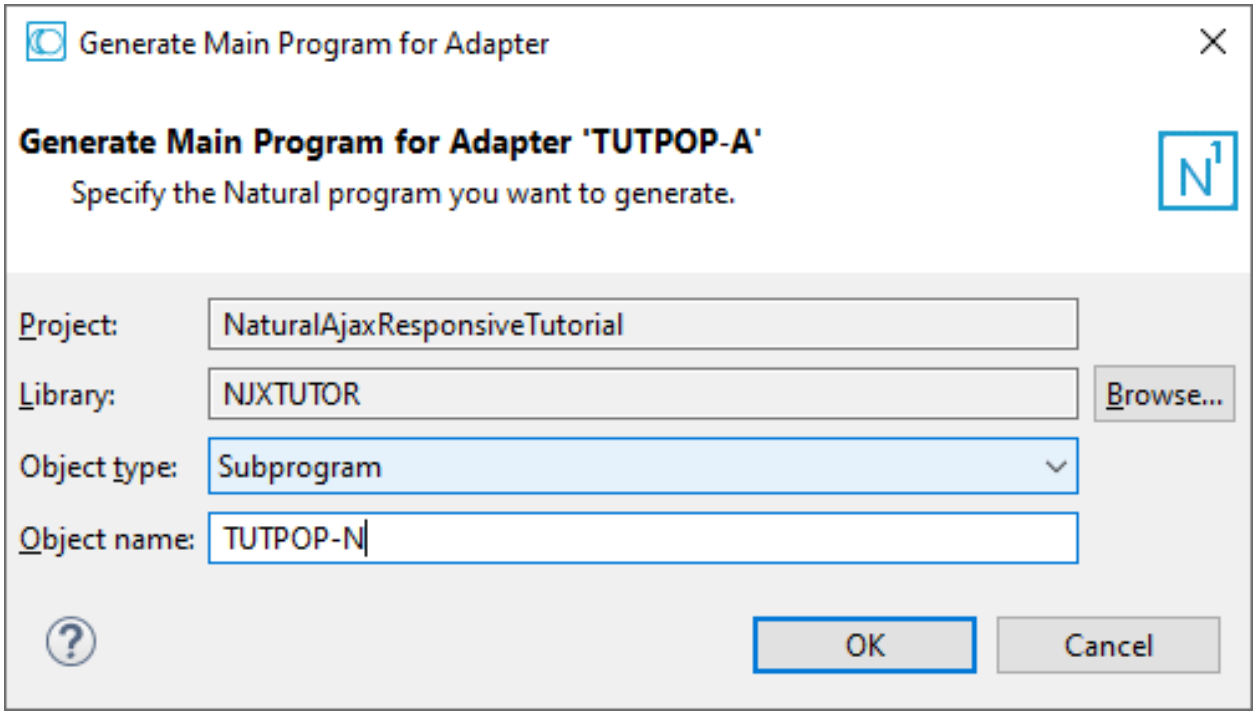
We have created a responsive form with several fields, added validation and responsive rendering and we added a submit and a cancel button to the form. We generated a Natural program for it and could run it immediately without any programming.

14 PART3: Put it All Together

We will now open our **tutorial_popup** page (PART2) from our **tutorial_main** page (PART1). We will create a Natural subprogram for our pop-up. When our TUTMAIN-P receives a specific event – an icon is clicked - it will call the Natural subprogram. This will open the pop-up window.

Create the Natural Subprogram for the tutorial_popup Page

Right click on the TUTPOP-A adapter and select **Generate Main Program**. In the opened dialog select **Subprogram** and enter **TUTPOP-N** as Object name.



We want to pass all values from our TUTMAIN-P program. To do so, apply the code in bold:

```
DEFINE DATA PARAMETER
1 P_EMAIL (A) DYNAMIC
1 P_FIRSTNAME (A) DYNAMIC
1 P_LASTNAME (A) DYNAMIC
1 P_USERID (A) DYNAMIC
LOCAL
1 ALERTTEXT (A) DYNAMIC
1 ALERTTYPE (A) DYNAMIC
1 EMAIL (A) DYNAMIC
1 FIRSTNAME (A) DYNAMIC
1 LASTNAME (A) DYNAMIC
1 USERID (A) DYNAMIC
END-DEFINE
*
```

```
EMAIL:=P_EMAIL
FIRSTNAME:=P_FIRSTNAME
LASTNAME:=P_LASTNAME
USERID:=P_USERID
*
…
```

We want that our **tutorial_popup** page is opened as pop-up (modal). To do so, apply the code in bold:

```
PROCESS PAGE MODAL
PROCESS PAGE USING "TUTPOP-A"
….
END-PROCESS
END
```

We want to close the pop-up when our **Cancel** button is clicked. When our **Save** button is clicked, we want to pass back the changed values. To do so, apply the code in bold:

```
VALUE U'onCancel'
IGNORE
VALUE U'onSave'
P_EMAIL:=EMAIL
P_FIRSTNAME:=FIRSTNAME
P_LASTNAME:=LASTNAME
P_USERID:=USERID
PROCESS PAGE UPDATE FULL
```

Add Controls to Open the Pop-Up (tutorial_main)

Open the **tutorial_main.xml** layout in Layout Painter. We will now add an icon column to our grid. When the icon is clicked, we want to pass an event to our Natural program so that it can open the pop-up.

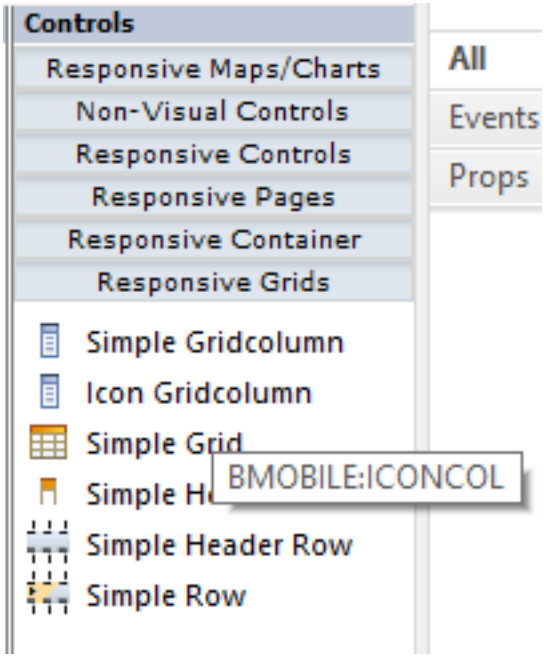
- Add an Icon Grid Column Control (BMOBILE:ICONCOL)
- Add a Natural Event Data Control (NJX:EVENTDATA)

- Add an Open Popup Features Control (XCIEVENTDATA.XCIINDEX)

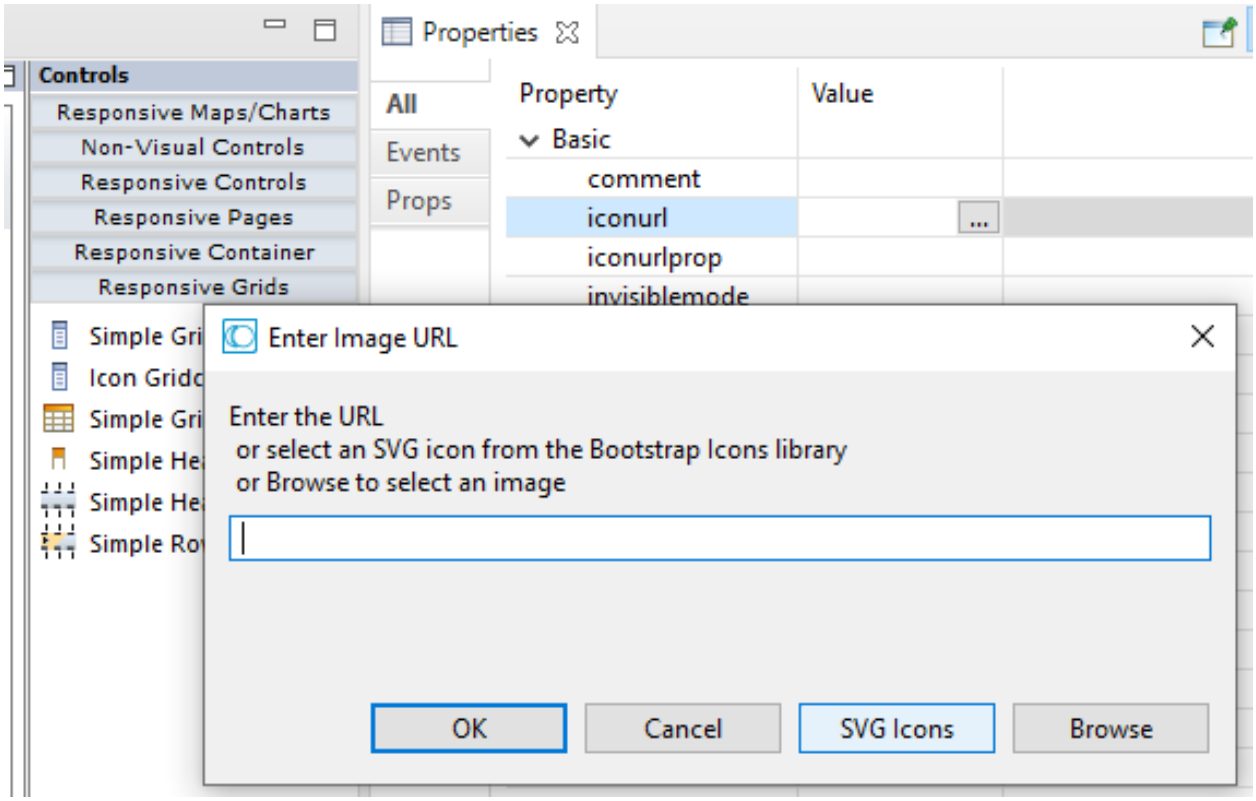
Add an Icon Grid Column Control (BMOBILE:ICONCOL)

In the **Layout Area (left)** select the **bmobile:simpleheaderrow** node. From the **Controls palette (right)** drag and drop a **Simple Header Column** as first sub node of the **bmobile:simpleheaderrow**.

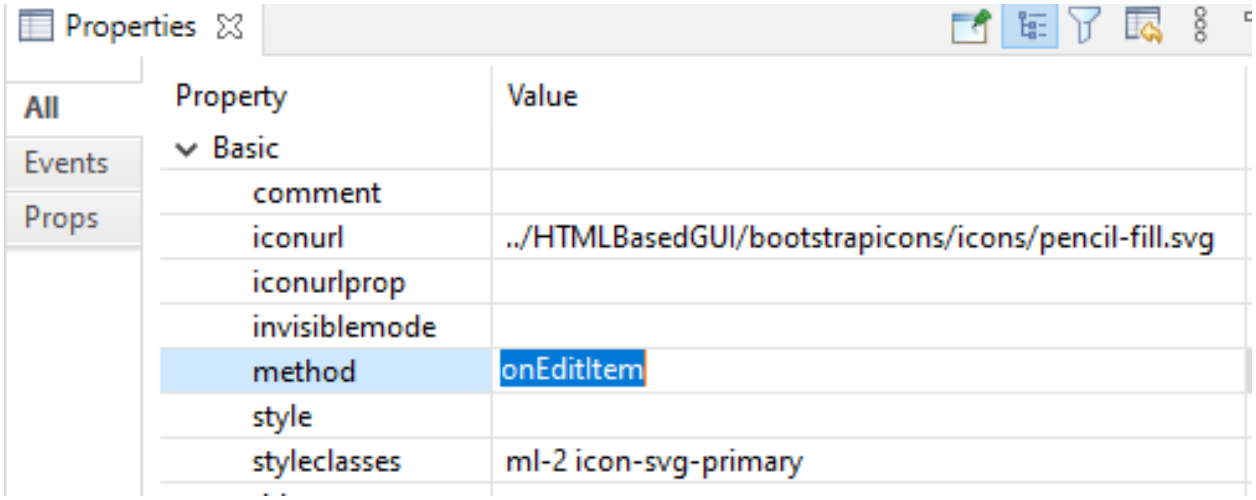
From the **Controls palette (right)** drag and drop an **Icon Grid Column** as first sub node of the **bmobile:simplerow**.



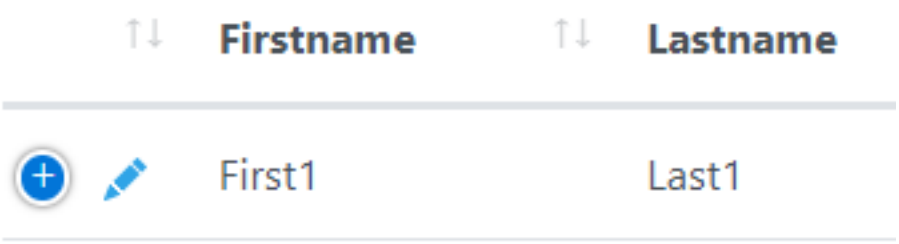
In **Properties** view click on the small button right of the **iconurl** property. In the opened dialog click the **SVG Icons** button and select the **pencil-fill.svg** icon.



Set the following **method** and **styleclasses** property values:



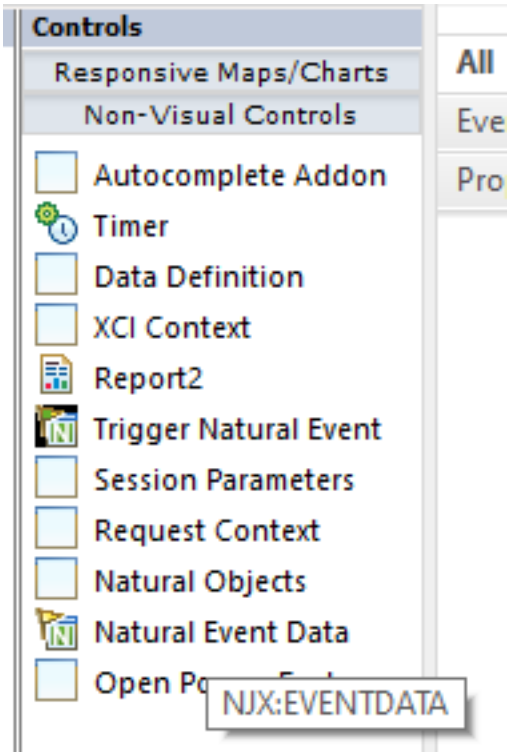
The event **onEditItem** will be sent to our Natural program when the icon is clicked. The styleclass **ml-2** adds some space left of the icon. This is important for small devices because on small devices a plus icon to see all columna is automatically shown. The **ml-2** styleclass makes sure that there is enough space between our pencil icon and the plus icon.



The **icon-svg-primary** styleclass renders our pencil in our primary color (blue).

Add a Natural Event Data Control (NJX:EVENTDATA)

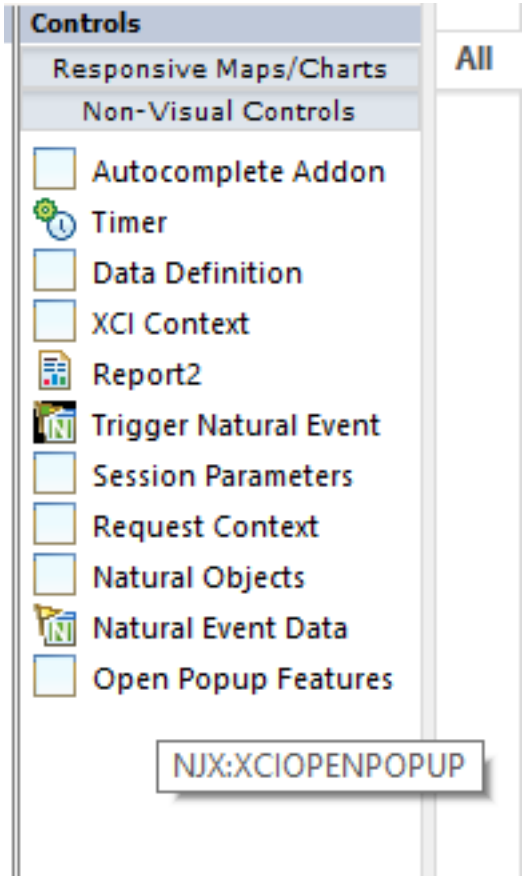
When our icon is clicked, we want to automatically know the index of the item for which the icon is clicked. To get this support, drag and drop a **Natural Event Data** control as sub node of the **natpage** node.



This is an invisible control. It generates Natural fields which will tell us the index of the item for which the icon is clicked.

Add an Open Popup Features Control (XCIEVENTDATA.XCIINDEX)

Drag and drop an **Open Popup Features** control as sub node of the **natpage** node.



This is an invisible control. It generates Natural fields which allow us to set the title, height, width, etc. of the pop-up dynamically at runtime. Be sure that you saved your layout (<ctrl-s>).

Adapt the Natural code (TUTMAIN-P)

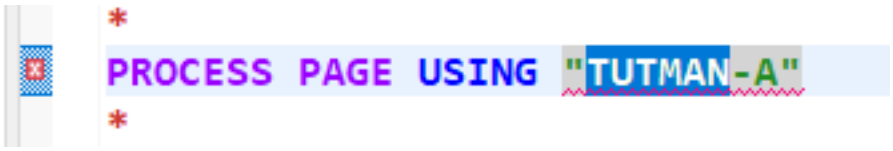
We now adapt our TUTMAN-P to initialize the pop-up and call our subprogram TUTPOP-N.

- Adapt the TUTMAN-P to the Added Controls
- Initialize the Pop-Up
- Call the Natural Subprogram (TUTPOP-N)

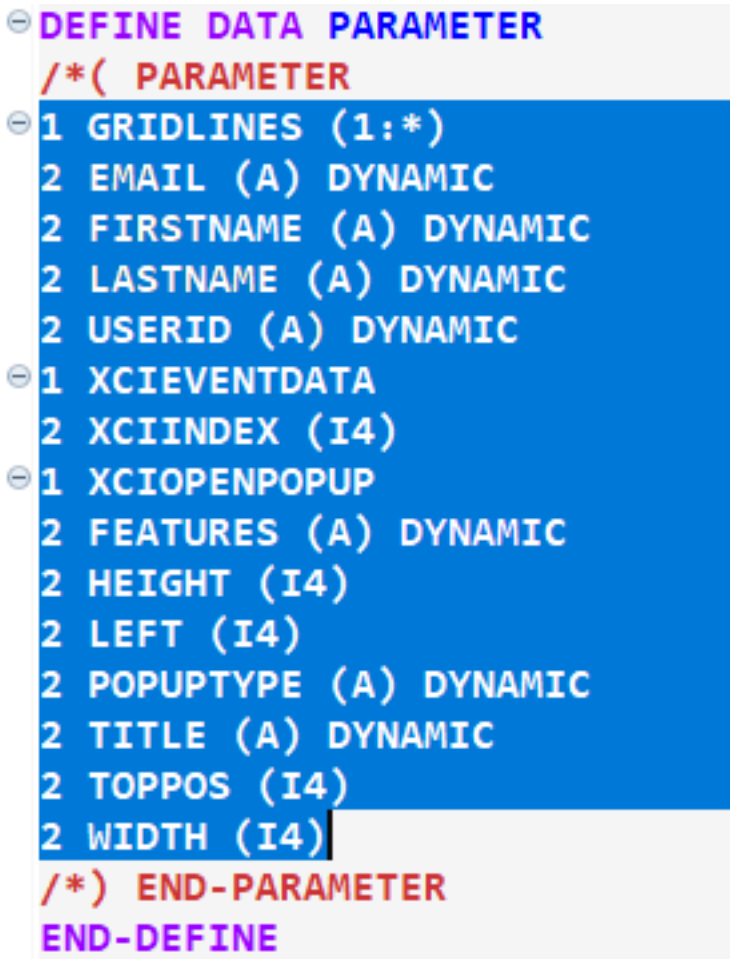
- Run your Application

Adapt the TUTMAN-P to the Added Controls

Double-click the file **TUTMAN-P**. It will open in the Natural editor. You see that it has errors. In the editor select **TUTMAN-A** and press **F3**.



This will open the TUTMAN-A adapter in the Natural editor. You see that in the adapter some additional fields exist. The reason is that for the Natural Event Data and the Open Popup Features controls additional Natural fields were generated. We need to adapt our program. Select all the fields in the adapter.



In the Natural program replace the old fields with these new fields. Also copy the lines for the **onEditItem** event. Your code should look like this:

```
DEFINE DATA LOCAL
1 GRIDLINES (1:*)
2 EMAIL (A) DYNAMIC
2 FIRSTNAME (A) DYNAMIC
2 LASTNAME (A) DYNAMIC
2 USERID (A) DYNAMIC
1 XCIEVENTDATA
2 XCIINDEX (I4)
1 XCIOPENPOPUP
2 FEATURES (A) DYNAMIC
2 HEIGHT (I4)
2 LEFT (I4)
2 POPUPTYPE (A) DYNAMIC
2 TITLE (A) DYNAMIC
2 TOPPOS (I4)
2 WIDTH (I4)
1 #II (I2)
END-DEFINE
EXPAND ARRAY GRIDLINES TO (1:30)
FOR #II:=1 TO 30
COMPRESS 'First ' #II INTO GRIDLINES.FIRSTNAME(#II) LEAVING NO
COMPRESS 'Last ' #II INTO GRIDLINES.LASTNAME(#II) LEAVING NO
COMPRESS 'First'#II '.Last' #II '@email.com' INTO GRIDLINES.EMAIL(#II) LEAVING NO
COMPRESS 'User' #II INTO GRIDLINES.USERID(#II) LEAVING NO
END-FOR
*
PROCESS PAGE USING "TUTMAN-A"
*
DECIDE ON FIRST *PAGE-EVENT
VALUE U'nat:page.end', U'nat:browser.end'
/* Page closed.
IGNORE
VALUE U'gridlines.onEditItem'
PROCESS PAGE UPDATE FULL
NONE VALUE
/* Unhandled events.
PROCESS PAGE UPDATE
END-DECIDE
*
END
```

Initialize the Pop-Up

We want to set a title for our pop-up. We also want to specify the pop-up size when it is opened. Add the following code to your Natural program before the PROCESS PAGE statement:

```
XCIOPENPOPUP.TITLE:='Edit User Information'
XCIOPENPOPUP.FEATURES:='sizetocontent:true'
*
PROCESS PAGE USING "TUTMAN-A"
```

In the **XCIOPENPOPUP.FEATURES** field you can set several features – like height width, whether a close button should appear etc, - separated be a semicolon. We just set one feature (sizetocontent) which means that if possible our pop-up should automatically adapt to the size of the opened page (tutorial_popup).

The last setting we need to add is just a technical one: In the **XCIOPENPOPUP.POPUPTYPE** we need to set the opened popup should be rendered using the responsive controls set (bmobile:*). Add the line in bold:

```
XCIOPENPOPUP.TITLE:='Edit User Information'
XCIOPENPOPUP.FEATURES:='sizetocontent:true'
XCIOPENPOPUP.POPUPTYPE:='BMPOPUP'
*
PROCESS PAGE USING "TUTMAN-A"
```

Call the Natural Subprogram (TUTPOP-N)

When our new icon is clicked, we will receive the event gridlines.onEditItem. The field XCIINDEX will contain the index of the item, for which the icon is clicked. Add the line in bold:

```
VALUE U'gridlines.onEditItem'
CALLNAT 'TUTPOP-N' EMAIL(XCIINDEX) FIRSTNAME(XCIINDEX) LASTNAME(XCIINDEX) ↩
USERID(XCIINDEX)
PROCESS PAGE UPDATE FULL
```

Don’t forget to press <ctrl-s>.

Run your Application

Right-click on TUTMAIN-P and select **Run As** > **Natural Application**.

SUMMARY

We have added an BMOBILE:ICON, an NJX:XCIOPENPOPUP and NJX:EVENTDATA control to the tutorial_main page. We have initialized the XCIOPENPOPUP fields for opening the pop-up. We have called our TUTPOP-N subprogram and passed the data of the selected item. We used

the XCIEVENTDATA.XCIINDEX field to know which item was clicked. We could run the completed application immediately.

15 Tutorial Reference Implementation

The application built in this tutorial is available as reference implementation. You can install it from the **Welcome** page of your NaturalONE installation.

IV First Steps Java Pages

This documentation is organized under the following headings:

About this Tutorial	How the completed "Hello World!" application will appear to the user. An overview of the basic steps that are required to create this application.
Starting the Development Workplace	How to start the development workplace.
Creating a Project	How to create the project that is to be used for this tutorial.
Getting Started with the Layout Painter	How to create a layout for your project. General information on how to use the Layout Painter.
Writing the GUI Layout	How to use the Layout Painter in order to create the GUI layout for the "Hello World!" application.
Setting Up Your Environment	How to set up Application Designer in your development environment for Java.
Writing the Adapter Code	How to define the class and how to use the Code Assistant in order to generate most of the adapter code. How to program the adapter code for the method using your development environment for Java and how to test the completed application. How to view the generated HTML page directly from the browser.
Some Background Information	Information on changing the adapter code. About name binding between controls and adapter, data exchange at runtime, and files and their locations. Some information for real development and on some tools provided in the development workspace.

It is important that you work through the exercises in the same sequence as they appear in this tutorial. Problems may occur if you skip an exercise.

16 About this Tutorial

This tutorial provides an introduction to working with Application Designer's development workplace. It explains how to create a “Hello World!” application. This covers all basic steps you have to perform when creating pages with Application Designer: you create a layout file, you create an adapter class, and you run the application. This is about eighty percent of all you need to know. The remainder of the Application Designer documentation contains just details and some more sophisticated techniques, such as navigating between pages.

When you have completed all steps of this tutorial, the page for your “Hello World!” application will look as follows:

Your application will act in the following way: When you enter a name in the **Your Name** field and choose the **Say Hello** button, the **Result** field displays "Hello World" and the name you have entered.

To reach this goal, you will proceed as follows:

1. You will first create a new Application Designer project.
2. You will then use Application Designer's Layout Painter to create the following layout:

This corresponds to the following XML layout:

```
<?xml version="1.0" encoding="UTF-8"?>
<page model="DummyAdapter">
    <titlebar name="Hello World!">
    </titlebar>
    <header withdistance="false">
        <button name="Say Hello" method="sayHello">
        </button>
    </header>
    <pagebody>
        <itr takefullwidth="true">
            <hdist width="100%">
            </hdist>
            <icon image="../cisdemos/images/hello.gif">
            </icon>
        </itr>
        <rowarea name="Input Area">
            <itr>
                <label name="Your name" width="100">
                </label>
                <field valueprop="name" width="200">
                </field>
            </itr>
        </rowarea>
        <rowarea name="Output Area">
            <itr>
```

```
                <label name="Result" width="100">
                </label>
                <field valueprop="result" width="200" displayonly="true">
                </field>
            </itr>
        </rowarea>
        <vdist pixelheight="100">
        </vdist>
        <itr>
            <label name="Input your name and press the &apos;Say Hello&apos; ↩
button." asplaintext="true">
            </label>
        </itr>
    </pagebody>
    <statusbar withdistance="false">
    </statusbar>
</page>
```

3. When you save your layout for the first time, an intelligent HTML page is generated.
4. Before you can start coding, you have to make specific definitions in Application Designer and in your development environment for Java (for example, Eclipse).
5. You will use Application Designer's Code Assistant to generate most of the adapter code.
6. You will then switch to your development environment in order to program the adapter code for the method `sayHello` which will be executed when you choose the **Say Hello** button of your application.

You can now proceed with your first exercise: *Starting the Development Workplace*.

17 Starting the Development Workplace

This tutorial assumes that you are using the Tomcat servlet engine which comes with this installation.

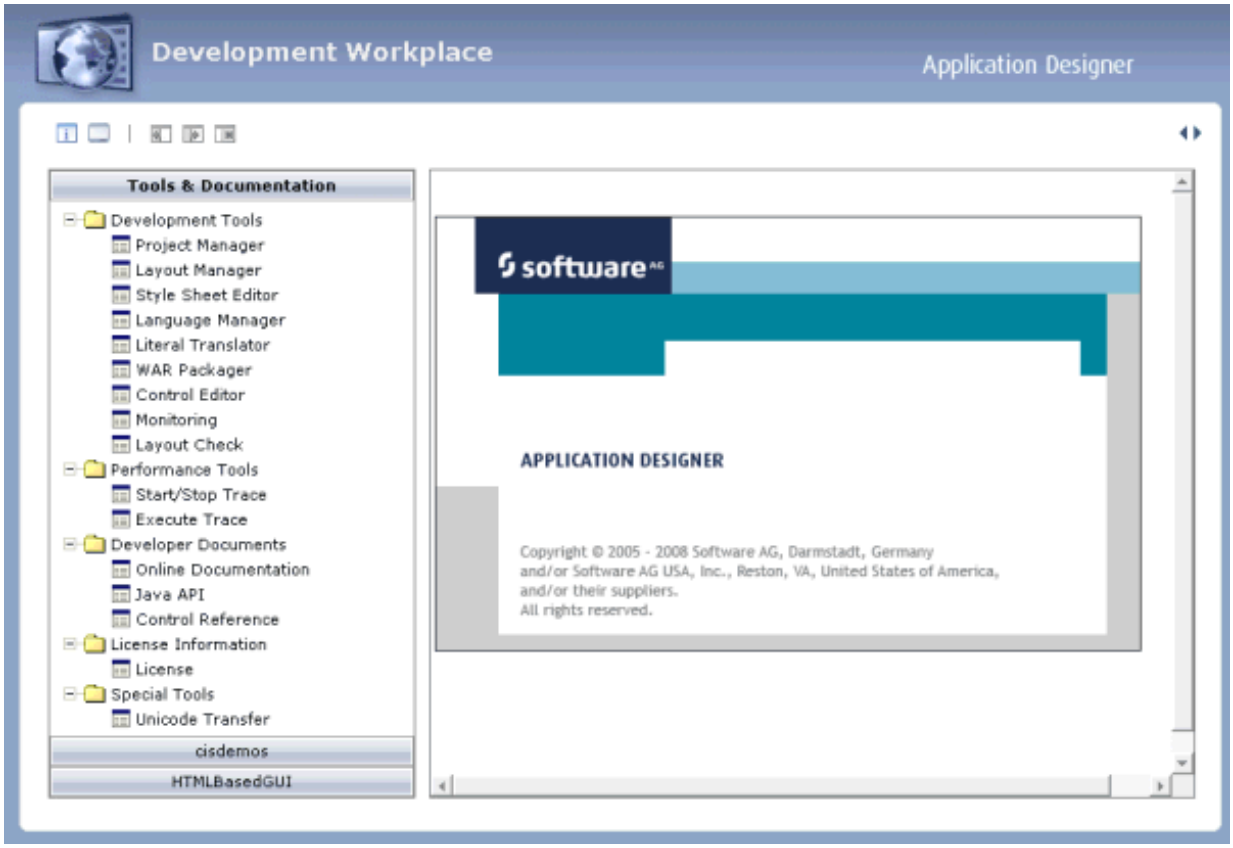
To start the development workplace

- Invoke your browser and start the development workplace with the following URL:

```
http://localhost:51000/cis/HTMLBasedGUI/workplace/ide.html
```

Note: If you have defined another port number during installation, enter this port number instead of the default port number 51000.

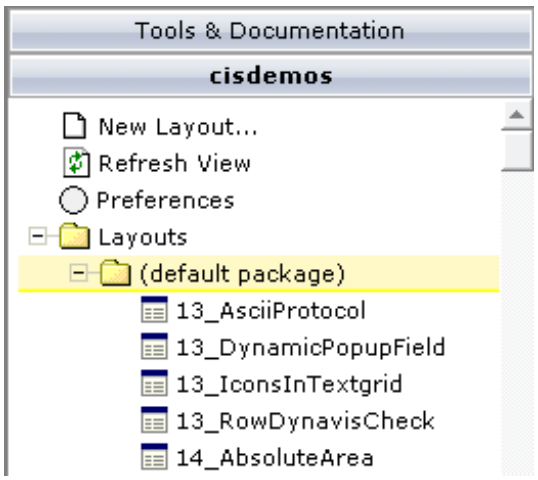
The development workplace is now shown in your browser.



You can now proceed with the next exercise: *Creating a Project*.

18 Creating a Project

In the Application Designer environment, layouts and classes are structured in so-called application projects. In the development workplace, you see the existing projects on the left. For each project, there is a tree of layout definitions that you can display when you choose the button containing the project name. For example:



For this tutorial, you will now create a project with the name "cisyourfirstproject".

To create a project

1 Choose **Tools & Documentation** to display the list of development tools.

2 Choose **Project Manager** in the tree.

A list of existing application projects is now shown on the right.

3 Choose the **New** button which is located below the list of application projects.

The following is now shown:

4 Enter "cisyourfirstproject" as the name of your project and choose the **Create** button.

Your new project is now shown in the list of existing application projects on the right.

The left side, which shows buttons for all existing projects, now also shows a button for your new project.

You can now proceed with the next exercise: *Getting Started with the Layout Painter*.

19 Getting Started with the Layout Painter

The Layout Painter, which can be accessed from the development workplace, is used to write the page layout. This is an Application Designer application itself.

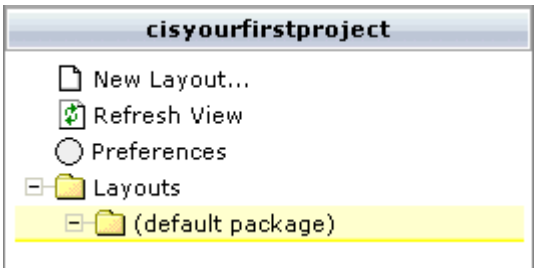
Creating a New Layout

You will now create a layout which is stored in the project you have previously created.

To choose a layout template

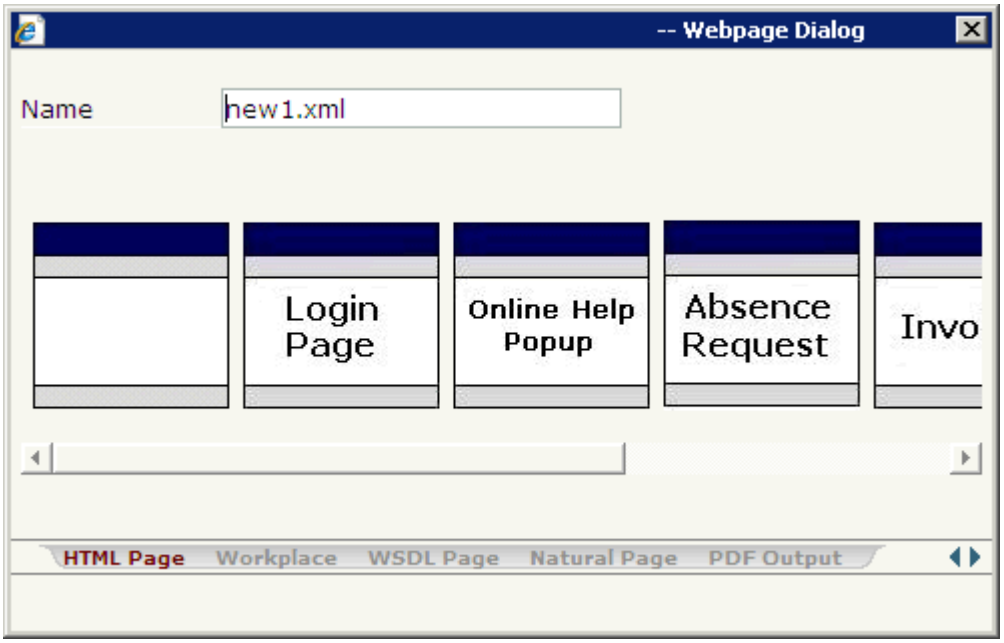
1 Choose the button for the project **cisyourfirstproject**.

The list of layout nodes inside the tree will be empty at the beginning:



2 Choose **New Layout...** in the tree.

The following dialog appears.

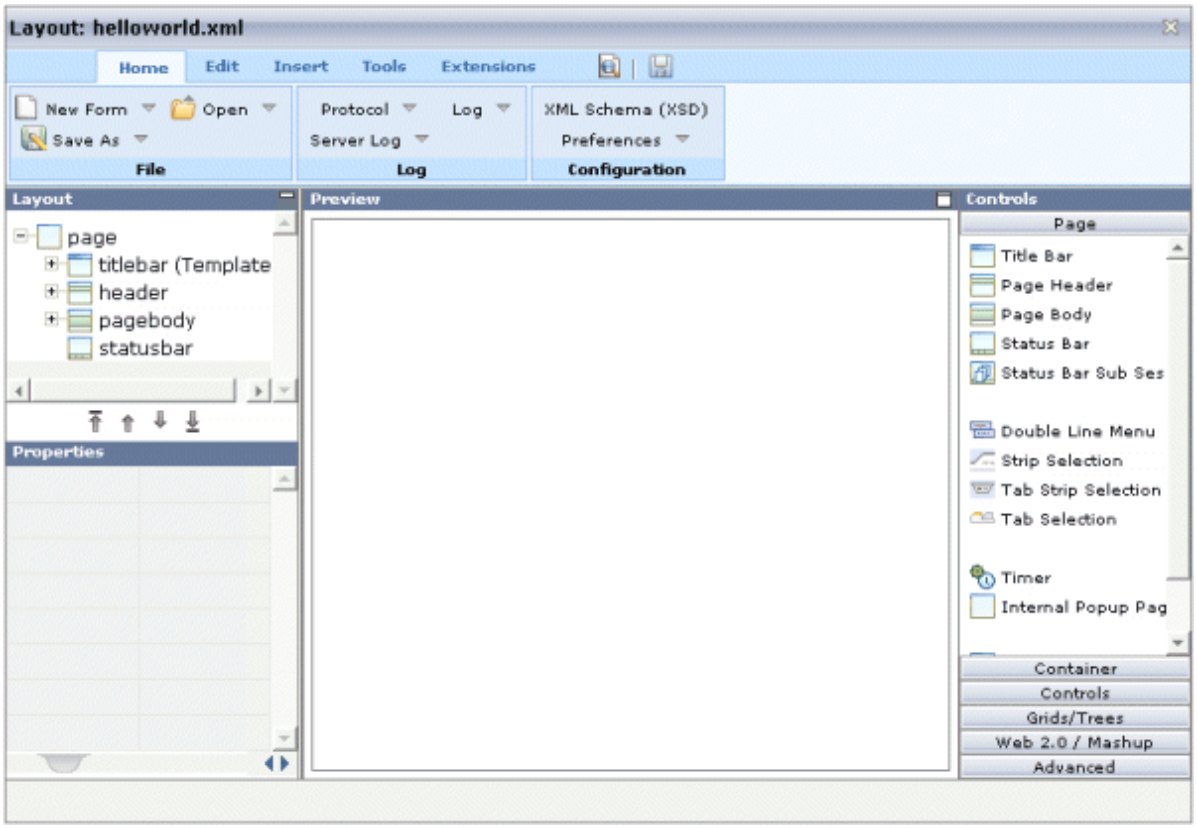


3 Enter "helloworld.xml" in the **Name** field.

This is the name of your layout definition.

4 Select the first template (when you move the mouse over this template, the tool tip "HTML Page" appears).

The main screen of the Layout Painter appears:



Note: The file *helloworld.xml* is stored in the */xml* directory of your project.

Elements of the Layout Painter Screen

The Layout Painter screen is divided into several areas:

- **Layout Area (left side)**
  This area consists of a layout tree and a properties area.

The layout tree contains the controls that represent the XML layout definition. You drag these controls from the controls palette into the layout tree. Each node in the layout tree represents an XML tag.

In the properties area below the layout tree, you specify the properties for the control which is currently selected in the layout tree.

- **Preview Area (middle)**
  The preview area shows the HTML page which is created using the controls in the layout area. This page is refreshed each time, you choose the preview button (see below).

  The Code Assistant, which will also be used in this tutorial, is also shown in the preview area.
- **Controls Palette (right side)**
  Each control is represented by an icon. A tool tip is also provided which appears when you move the mouse pointer over the control. This tool tip also displays the XML tag which will be used in the XML layout.

  The palette is structured into sections, where each section represents a certain type of controls.

Previewing the Layout

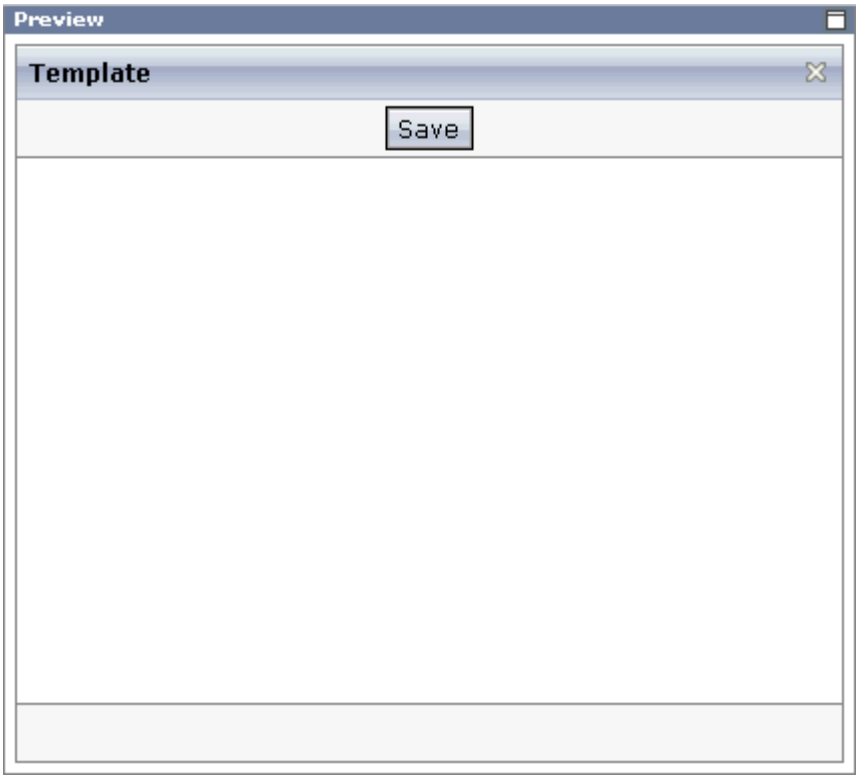
The layout tree inside the Layout Painter already contains some nodes that were copied from the template that you chose in the dialog in which you specified the name of the page. To see what the page looks like, preview the layout as described below.

To preview the layout

- Choose the following button which is shown at the top of the Layout Painter.

  The preview area is updated and you see the page. The page already contains a title bar, a header containing a **Save** button, the page body and a status bar.



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

Viewing the XML Code

When creating the layout, you can view the currently defined XML code.

To view the XML code

- From the **Edit** tab of the Layout Painter, choose **XML**.

  A dialog box appears. At this stage of the tutorial, it contains the following XML layout definition for the nodes which were copied from the template.

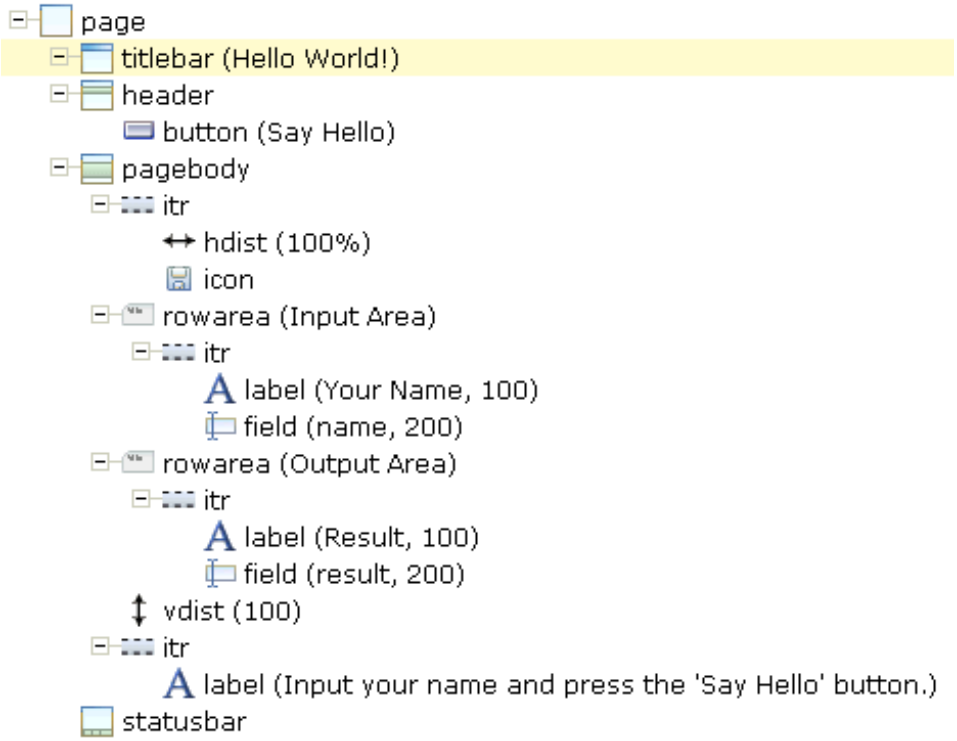
```
<page model="DummyAdapter">
    <titlebar name="Template">
    </titlebar>
    <header withdistance="false">
        <button name="Save">
        </button>
    </header>
    <pagebody>
    </pagebody>
```

```
    <statusbar withdistance="false">
    </statusbar>
</page>
```

You can now proceed with the next exercise: *Writing the GUI Layout*.

20 Writing the GUI Layout

You will now create the layout for your “Hello World!” application. When you have completed all exercises in this chapter, the layout should look as shown below and the **XML code** should be the same as shown in the section *About this Tutorial*.



Tip: **Preview the layout** and **view the XML code** each time you have completed an exercise. If the system finds some wrong or missing definitions while generating the preview page, there will be a corresponding message in the status bar. From the **Home** tab of the Layout Painter, choose **Protocol** to get more information about these problems.

Specifying a Name for the Title Bar

You will now specify the string "Hello World!" which is to appear in the title bar of your application.

To specify the name for the title bar

1 In the layout tree, select the node **titlebar (Template)**.

The properties for this control are now shown in the properties area at the bottom. You can see the default entry "Template" for the `name` property.

2 Specify the following property:

Property	Value
name	Hello World!

When you click on the layout tree, the node in the layout tree changes to **titlebar (Hello World!)**.

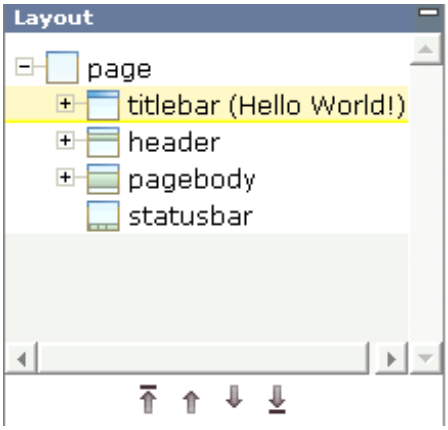
Note: Properties that are left blank are not shown in the XML code.

Using the Property Editor

You can also specify the property values using the Property Editor. In this case, you can access detailed help information on each property.

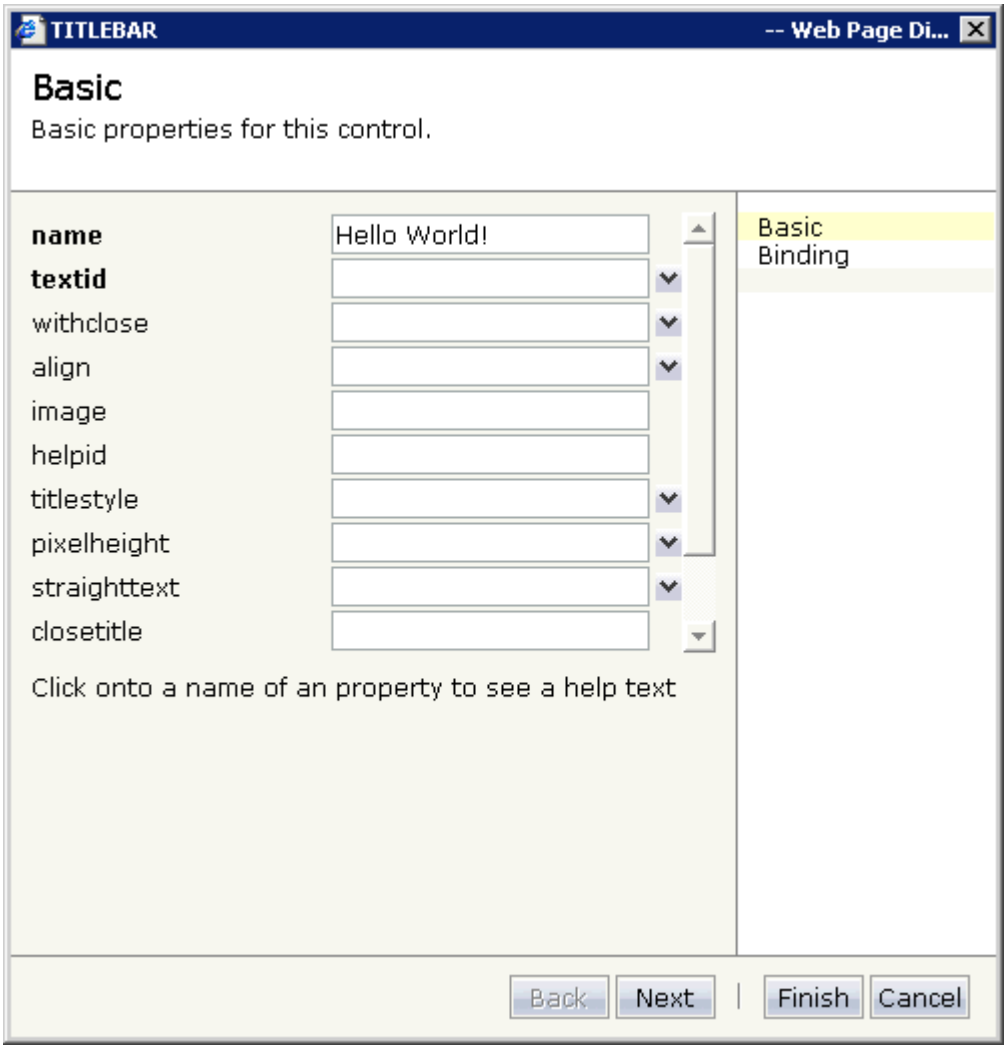
To use the Property Editor

1 Select the control in the layout tree for which you need help, for example, the **titlebar (Hello World!)** node.



2 From the **Edit** tab of the Layout Painter, choose **Property Editor**.

The following dialog appears.



The properties of the control are listed.

3 Click on the name of a property to display detailed information on this property. This information is shown below the list of properties.

4 Choose the **Finish** button to close the dialog.

Any changes you have applied in the dialog will be saved.

Specifying a Name and Method for the Button

You will now specify the string "Say Hello" which is to appear on the button. And you will specify the name of the method that is to be invoked when the user chooses this button.

To specify the name and the method for the button

1 In the layout tree, open the **header** node.

Note: By clicking the icon of a node, you hide or expand the node's subnodes.

You can now see the entry for the button with the default name "Save".

2 Select the node **button (Save)**.

3 Specify the following properties:

Property	Value
`name`	Say Hello
`method`	sayHello

The method needs to be programmed in the adapter class. This will be explained later in this tutorial.

Adding the Input and Output Areas

The input and output areas in this tutorial are created using **Row Area** controls. These controls can be found in the **Container** section of the controls palette.

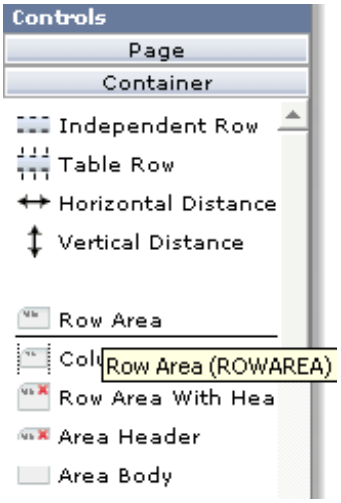
Each row area will contain an **Independent Row** control which in turn contains a **Label** and a **Field** control. These controls can be found in the **Controls** section of the controls palette.

For adding controls to your layout, you drag them from the controls palette onto the corresponding tree node in the layout tree. This is explained below.

To create the input area

1 Open the **Container** section of the controls palette.

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



2 Drag the **Row Area** control from the controls palette onto the **pagebody** node in the layout tree.

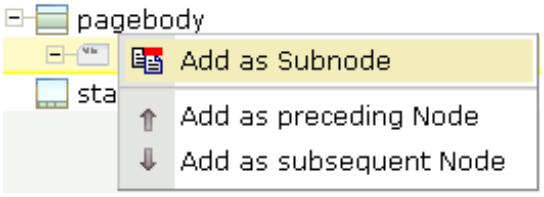
The row area is added as a subnode of the **pagebody** node. The new subnode is automatically selected so that you can maintain the properties of the row area directly in the properties area.

3 Specify the following property:

Property	Value
name	Input Area

4 Drag the **Independent Row** control from the controls palette onto the **rowarea (Input Area)** node in the layout tree.

When you drop information into the tree, the system will sometimes respond by offering a context menu with certain options about where to place the control. In this case, the following context menu appears.



Note: When you move the mouse outside the context menu, the context menu disappears. The control is not inserted in this case.

5 Choose the **Add as Subnode** command.

The control is now inserted below the **rowarea (Input Area)** node. The new node is shown as **itr**.

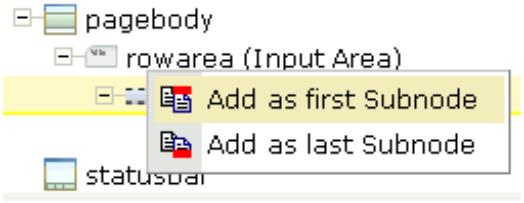
6 Open the **Controls** section of the controls palette.

7 Drag the **Label** control from the controls palette onto the **itr** node you have just inserted and specify the following properties:

Property	Value
name	Your Name
width	100

8 Drag the **Field** control from the controls palette onto the **itr** node you have just inserted.

A context menu appears and you have to specify where to place the control.



9 From the context menu, choose the **Add as last Subnode** command.

10 Specify the following properties for the field:

Property	Value
valueprop	name
width	200

Note: As long as the adapter class is not defined, the field is dimmed and it is not possible to enter information in this field. The adapter class will be defined later in this tutorial.

To create the output area

- Create the output area in the same way as the input area (add it as the last subnode of the **pagebody** node), with the following exceptions:

  Row Area

  Specify a different value for the following property:

Property	Value
name	Output Area

Label

Specify a different value for the following property:

Property	Value
name	Result

Field

Specify different values for the following properties:

Property	Value
valueprop	result
displayonly	true

Note: To display the `displayonly` property, choose the **Appearance** tab at the bottom of the properties area. You can then select the required value from a drop-down list box.

Adding the Image

You will now add the image which is to be shown above the input area. To do so, you will use the **Icon** control which can be found in the **Controls** section of the controls palette.

Note: The image is provided in Application Designer's */cisdemos/images* directory.

To add the image

1 Drag the **Icon** control from the controls palette onto the **pagebody** node in the layout tree.

The icon is added as the last subnode of the **pagebody** node. It is automatically placed into an **itr** (independent row) node.

2 Specify the following property for the icon:

Property	Value
image	../cisdemos/images/hello.gif

3 Select the **itr** node containing the icon and choose the following button below the layout tree:

The selected node is now moved up so that it appears as the first subnode of the **pagebody** node.

4 Specify the following property for the **itr** node:

Property	Value
takefullwidth	true

Adding a Horizontal Distance

When you preview the layout, you will see that the image you have just added appears centered.

You will now move the image to the right side of the page. To do so, you will use the **Horizontal Distance** control which can be found in both the **Controls** section and the **Container** section of the controls palette.

To add the horizontal distance

1 Drag the **Horizontal Distance** control from the controls palette onto the **itr** node containing the icon.

2 From the resulting context menu, choose the **Add as first Subnode** command.

The node **hdist** is inserted into the tree.

3 Specify the following property:

Property	Value
width	100%

Adding an Instructional Text

You will now enter a text which is to appear below the output area and which tells the user what to do.

To do so, you will once again use the **Independent Row** control into which you will insert a **Label** control.

Note: The **Independent Row** control can be found in both the **Controls** section and the **Container** section of the controls palette.

To add the independent row with the label

1 Drag the **Independent Row** control from the controls palette onto the **pagebody** node in the layout tree.

2 From the resulting context menu, choose the **Add as last Subnode** command.

The node **itr** is inserted into the tree.

3 Drag the **Label** control from the controls palette onto the **itr** node you have just created.

4 Specify the following properties for the label:

Property	Value
`name`	Input your name and press the 'Say Hello' button.
`asplaintext`	true

Note: Go to the **Appearance** tab to display the property `asplaintext`.

Adding a Vertical Distance

When you preview the layout, you will see that the text you have just added appears directly below the output area. You will now move the text 100 pixels to the bottom.

To do so, you will use the **Vertical Distance** control which can be found in both the **Controls** section and the **Container** section of the controls palette.

To add the vertical distance

1 Drag the **Vertical Distance** control from the controls palette onto the **itr** node containing the label.

2 From the resulting context menu, choose the **Add as preceding Node** command.

The node **vdist** is inserted into the tree.

3 Specify the following property:

Properties	Value
height	100

Saving Your Layout

If you have not already done so, you should now save your layout.

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

To save the layout

- Choose the following button which is shown at the top of the Layout Painter.

You can now proceed with the next exercise: *Setting Up Your Environment*.

21 Setting Up Your Environment

The recommendation is to use NaturalONE. Create a Java Project and enable it for AjaxDeveloper. For more information see the NaturalONE documentation.

You can now proceed with the next exercise: *Writing the Adapter Code*.

22 Writing the Adapter Code

Defining the Class Name

As a last step of your layout definition, you will now define the name of the class which is the logical counterpart of your page.

To define the class name

1 In the layout tree, select the top node **page**.

You can see the default entry "DummyAdapter" for the `model` property.

2 Specify the following property:

Properties	Value
`model`	HelloWorldAdapter

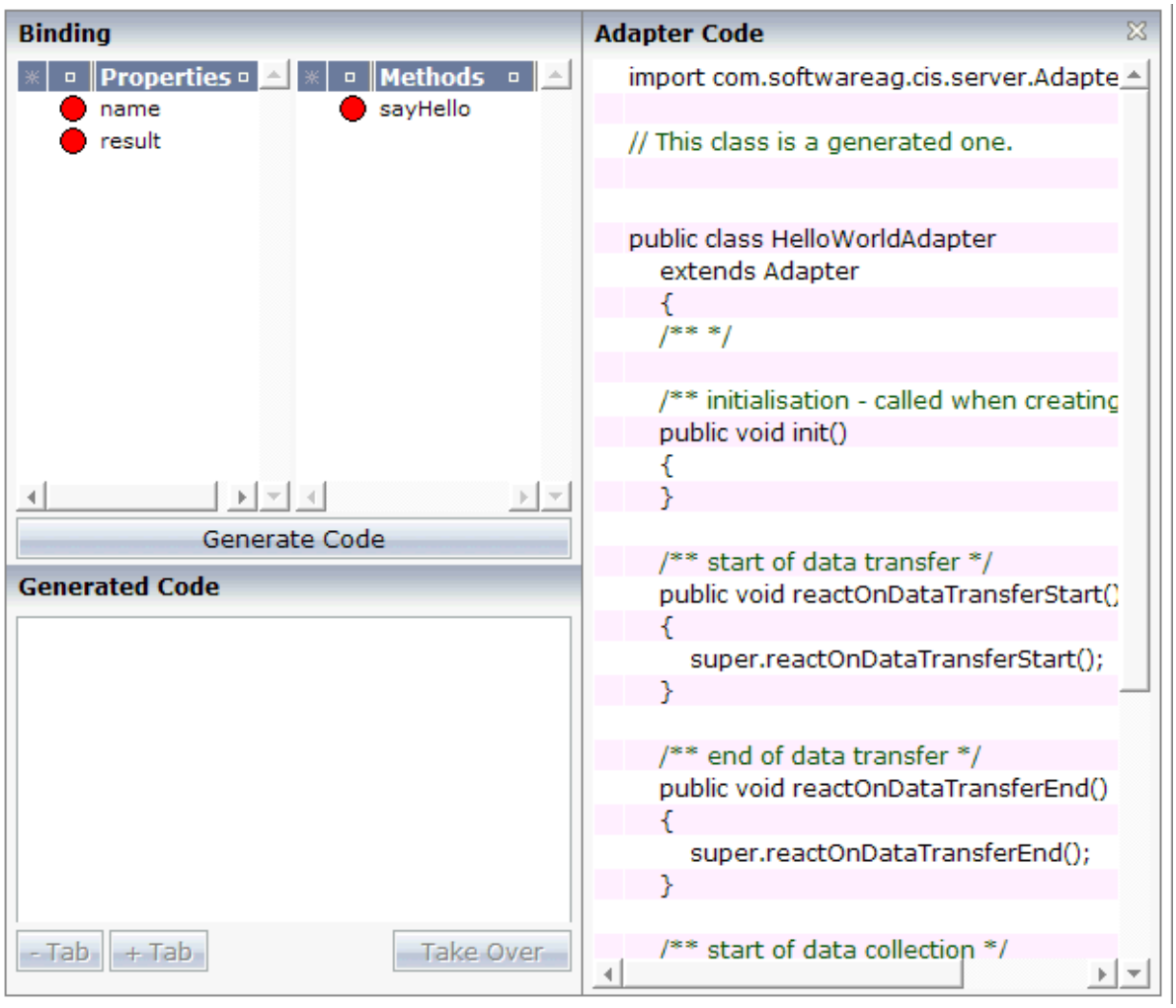
Generating the Code

You use the Code Assistant, which is part of the Layout Painter, to generate code for the referenced properties and methods.

To generate code

1 From the **Tools** tab of the Layout Painter, choose **Code Assistant**.

The following is shown in the preview area:



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

2 Select the property `name`.

3 Choose the **Generate Code** button.

The generated code appears below this button.

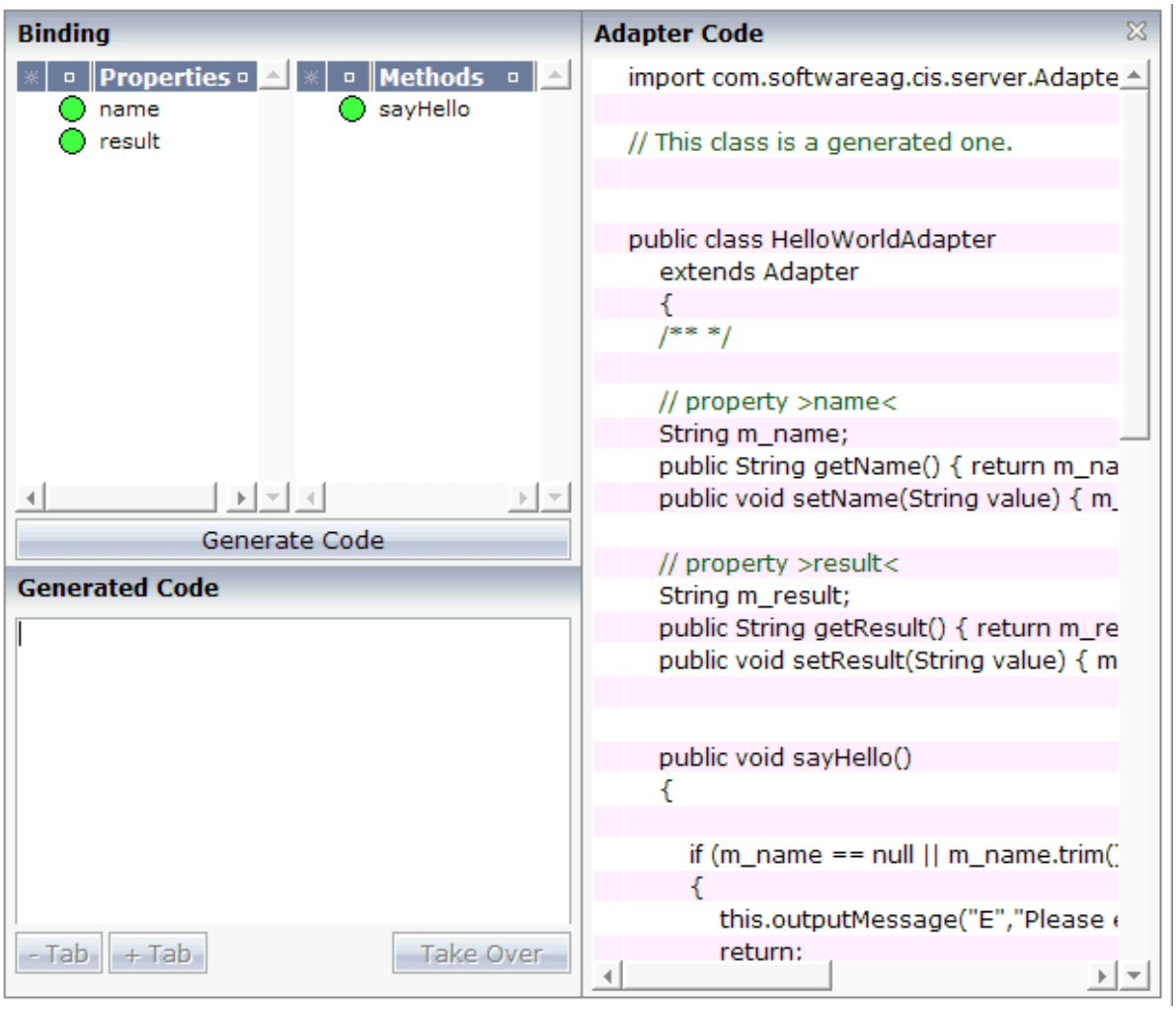
4 Select the position in the adapter code at which you want to insert the code.

5 Choose the **Take Over** button to insert the code at the selected position.

The color of the dot changes from red to green. This indicates that the property's `get` and `set` methods are now available in the adapter code.

6 Repeat the above steps for the `result` property and the `sayHello` method.

When you have finished all required steps, the page should look as follows:



7 Save your changes and close the Code Assistant.

Programming the Method

Now that the framework for the `sayHello` method has been generated in the adapter code, you have to program the method itself.

To program the method

1 Go to your development environment for Java.

2 Open the file *HelloWorldAdapter.java* which is located in the project directory */cisyourfirstproject/src*.

3 Add the line which is indicated in bold:

```
/** */
    public void sayHello()
    {
        // TODO Auto-generated method stub
     m_result = "Hello World, " + m_name + "!";
    }
```

4 Save your changes.

Testing the Completed Application

You will now check whether your application provides the desired result.

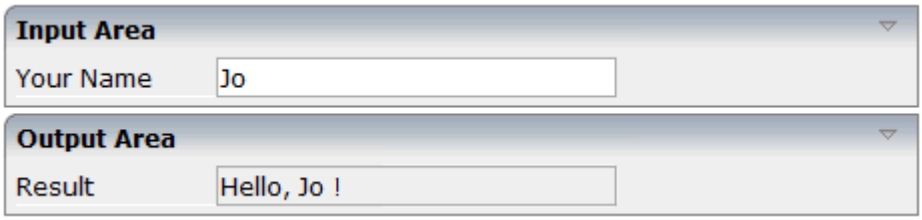
To test the application

1 Go back to your layout in the Layout Painter.

2 Choose the following button which is shown at the top of the Layout Painter.

3 Enter your name and choose the **Say Hello** button.

The page should perform correctly now, and successfully "talk" to the adapter you have just created:



Viewing the Page Outside the Layout Painter

The generated HTML file *helloworld.html* (which is updated each time you save your layout) can be found within the root of your application project, that is in *<installdir>/tomcat/webapps/cis/cisyourfirstproject*.

This HTML page has some prerequisites concerning the browser workplace in which it is running. Therefore, it is per se not usable as a directly accessible page but needs to be embedded into a frame providing a defined set of functions.

For viewing the page directly, enter the following URL inside your browser:

```
http://localhost:51000/cis/servlet/StartCISPage?PAGEURL=/cisyourfirstproject/helloworld.html
```

If the page is not displayed, check the following:

- URLs are case-sensitive. Double-check your input.
- Check whether the file *helloworld.html* is available in the directory *cisyourfirstproject*.

Inside the URL, a servlet StartCISPage is invoked. As parameter `PAGEURL`, a certain value is passed: this value is built from the application project name and the page name. Calling the servlet is the generic way of opening pages which are created with Application Designer directly inside the browser.

You have now completed this tutorial. See the remaining section of these *First Steps* for **some background information**.

23 Some Background Information

If you Change the Adapter

You can now add more functions to your “Hello World!” application by yourself. This means that you will both work with the layout description inside the Layout Painter as well as change the code of the adapter.

Changing the adapter's code requires that you replace your existing class files in the *<installdir>/tomcat/webapps/cis/cisyourfirstproject/appclasses/classes* directory by new ones. What does this mean for the runtime environment? The runtime does not see these recompiled classes because it already loaded the existing classes. So there is no difference how the application performs by default.

However, you can explicitly force the server to reload all application classes and to use the newest class version. The server is clever enough to keep old classes for existing sessions, which means that users who are already logged on will not be disturbed. But new sessions will get the newest classes.

The task to update the classes is directly triggered from the Layout Painter by choosing the save button. This tells the runtime to use new classes for new sessions. Since the preview area always opens a new session, it will be up-to-date whenever you choose the preview button.

Therefore, you can develop continuously, without restarting the GUI-server Tomcat environment.

For fast readers:

The class loader management mentioned above is designed to be used at design time. At runtime, especially if running in non-Tomcat environments, you should use the application server's web class loader to load the adapter classes. Details are provided in other sections of the Application Designer documentation.

Name Binding between Controls and Adapter

Which are the critical parts when building the “Hello World!” application?

- The PAGE control in the layout points to the class name of the adapter (property `model`).
- The FIELD control in the layout points to the property name of the adapter (property `valueprop`).
- The BUTTON control in the layout points to the method `sayHello()` of the adapter (property `method`).

There is a name binding between the layout definition and its corresponding adapter. This is the simple and effective approach of the Application Designer's development process: The adapter represents a logical abstraction of what the page displays. All layout definitions are kept in the

page - all the logic is kept in the adapter. (Or better: behind the adapter. The adapter itself should only be a facade to the “real” application logic.)

Data Exchange at Runtime

What happens at runtime?

- The browser accesses (within the preview area of the Layout Painter) the intelligent HTML page generated from the XML layout description. The HTML page is loaded. Depending on the settings of your browser, the HTML page is picked from the page buffer of the browser after the first access.
- The page is loaded and JavaScript statements are executed to send a command to the server, requesting the data content of the page.
- The server's session management finds out that a new instance of the adapter class has to be created. It calls the default constructor - without any parameters - of the adapter class and registers it within its session management.
- The server calls all `get` methods of the adapter and collects the results in a streamed string. The string is sent back as a response to the browser.
- The page inside the browser receives the response and distributes the new content to the controls. The controls are updated using JavaScript statements working with the DOM interface.
- The user provides some input, for example, enters the name. The content change is registered inside the page.
- The user does something which causes a flush of the changes (for example, the user chooses a button). Therefore, all registered data changes are packaged as a streamed string and are sent to the server including the information that a method has to be called.
- The server receives the request and finds by its session management the corresponding adapter object. First, it pushes all data changes using the `set` methods into the adapter class and calls the method.
- The method changes the data inside the adapter.
- The server calls all `get` methods of the adapter and collects the result into a streamed string. The string is sent back as a response to the browser.
- And so forth.

With a standard HTTP connection, only the changed content of the screen is passed when operating on one page. The layout is kept stable in the browser. Consequently, there is no flickering of the page due to page reloading.

All steps described in the list above are done completely transparent to your adapter; i.e. you do not have to cope with session management, stream parsing, error management, building up HTML

on the server, etc. You just have to provide an intelligent HTML page by defining it in the Layout Painter and an adapter class.

Files and their Locations

Have a look at the files created for your “Hello World!” application and take notice of the directory in which they are located.

All files are located in the directory *<installdir>/tomcat/webapps/cis/cisyourfirstproject*. The */cis* directory is the directory of the web application instance. The */cisyourfirstproject* directory is the directory that has been created for your new project

- The XML layout definition is kept in the */xml* directory.
- The generated HTML page is kept directly in the project directory. There are also some other files inside this directory that start with "ZZZZ". These files are temporary files used when previewing pages inside the Layout Painter.
- The Java class files are kept in the directory */appclasses/classes*. There is also a directory */appclasses/lib* where you can find *jar* files to be imported into your application.
- In the directory */accesspath*, “access restriction” files are generated. If you view these files inside a normal text editor (such as Notepad), you see that one file is maintained for each page; it holds the information about which properties are accessed by the page.

The directory locations described in this section are the default locations.

Application Project Management

Use the project "cisyourfirstproject" only for your first steps. For real development, create your own projects. Working inside your own projects, you can be sure that your project files will never be touched by any Application Designer updates. Therefore, patches or enhancements to Application Designer can be installed by just copying them over the existing Application Designer installation.

Keep all your files inside the project directory during the development cycle. Even if this might work correctly: for example, do not put *jar* files into the server (Tomcat) directories, but place them into the corresponding */appclasses/lib* directory of your project.

There are some rare cases in which you have to position *jar* files explicitly inside the Tomcat environment: for example, if classes of a *jar* file access native libraries (such as DLLs) by the JNI interface. They can only be loaded once by the class loader. Because both Tomcat and Application Designer have class loaders which may load a class multiple times, you have to position these classes, for

example, into the */tomcat/lib* directory. However, these are the exceptions. Do not put any classes outside of the Application Designer project directories unless there is a reason for it.