Natural for Ajax

Application Modernization

Version 9.3.2

February 2025

ADABAS & NATURAL

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

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

Document ID: ONE-NATNJX-APP-MODERNIZATION-932-20250213

Table of Contents

Application Modernization

This section describes how to convert a character-based Natural application to a Natural for Ajax application.

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

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:

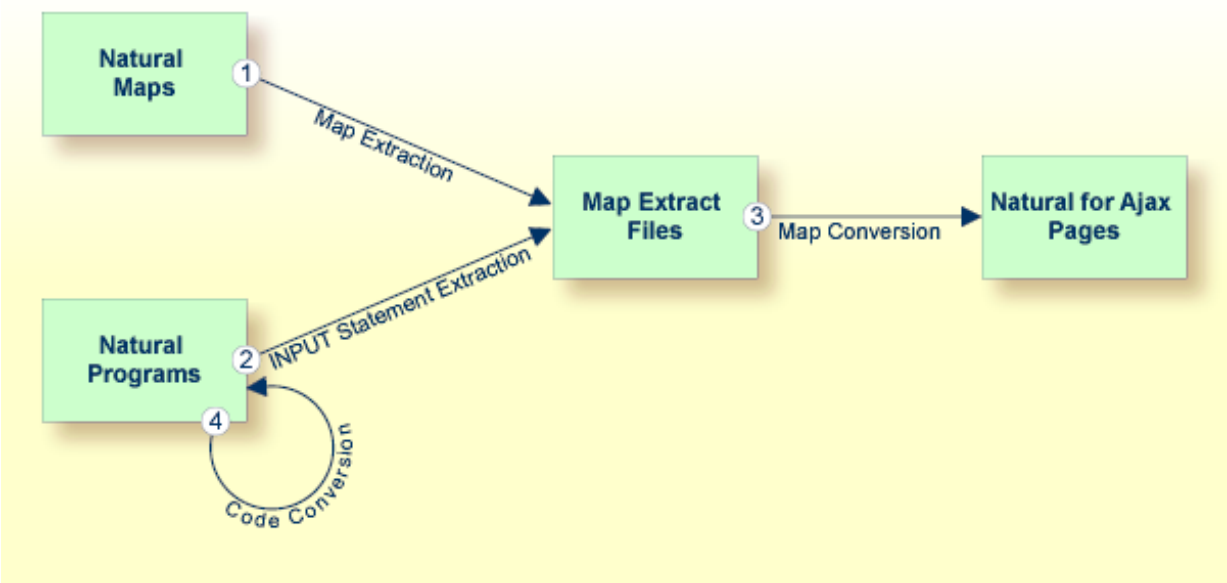
- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

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.

2 Overview of Conversion Steps

The conversion of a character-based Natural application to a Natural for Ajax application consists of several steps as illustrated in the following graphic:



- **Step 1: Map Extraction**
  Extracts from each Natural map the information that is required to create a corresponding Natural for Ajax page. For each map, a map extract file is created. This step is automatically performed by the Map Converter, see below.
- **Step 2: `INPUT` Statement Extraction**
  This step is required for Natural applications that do not use maps, but use `INPUT` statements for the dynamic specification of the screen layouts.

  Extracts from each `INPUT` statement in the source code the information that is required to create a corresponding Natural for Ajax page. For each `INPUT` statement, a map extract file is created.

This file has the same format as a map extract file created by the map extraction process, and it is also intended as input for the map conversion.

Required tool: Natural Engineer which is provided with NaturalONE.

- **Step 3: Map Conversion**
  Processes the map extract files and creates the corresponding Natural for Ajax pages.

  Required tool: Map Converter.

  See *Map Conversion* and *Customizing the Map Conversion Process* for further information.

- **Step 4: Code Conversion**
  This step requires that the Natural for Ajax pages have already been created.

  Modifies the application code in such as way that it can use the newly created Natural for Ajax pages. The application can still run in a terminal, in the Natural Web I/O Interface client or in batch as before. But it can now also run in a Natural for Ajax session with the new Natural for Ajax pages.

  Required tool: Natural Engineer which is provided with NaturalONE.

  Code conversion can also be performed manually. See *Code Conversion* for further information.

The resulting Natural for Ajax application mimics the character-based application. The user interface is not restructured in the sense that several maps are combined into a single page or that complex maps are split into several separate pages. This kind of restructuring is not part of the conversion, but of the normal development of a Natural for Ajax application.

3 Map Conversion

General Information

The Map Converter analyses the code of a Natural map and creates a so-called “map extract file” for each map. The map extract file contains information about the map. Normally, the map extract file is automatically deleted when the conversion process is completed. However, you also have the option to keep this file. The map extract files have the extension *.njx* and are not human-readable. They are only intended as input for the map conversion.

The conversion process can also be started on an existing map extract file which has been created for the `INPUT` statements in your source code.

The conversion process creates a Natural for Ajax page layout from each map extract file. Controls on the map are converted to controls on the page. Many features of the original map are converted to features of the page.

Note: It is only possible to process character maps. GUI elements contained in maps are not extracted.

By default, the Map Converter uses a predefined set of page templates and conversion rules that control the conversion process. The templates and the conversion rules can be modified or extended to adapt the converter to the requirements of a specific conversion project. With the advanced option to program own conversion handlers, the Map Converter provides additional flexibility and extensibility.

The following tools are available for the conversion of maps:

- **Map Converter**
  This tool is used for mass generation of layouts and also for generating single layouts. See ***Using the Map Converter*** for further information.
- **Conversion Rules**
  You can use this tool to copy the conversion rules from other user interface components to the current user interface component. See ***Using the Conversion Rules Tool*** for further information.
- **Conversion Logs**
  You can use this tool to view or delete the log files that have been created during the conversion. See ***Using the Conversion Logs Tool*** for further information.

Using the Map Converter

The Map Converter is used for mass generation of layouts and also for generating single layouts.

To convert maps

1 In the **Project Explorer** view or in the **Natural Navigator** view, select the Natural project which contains the maps that are to be converted.

Or:

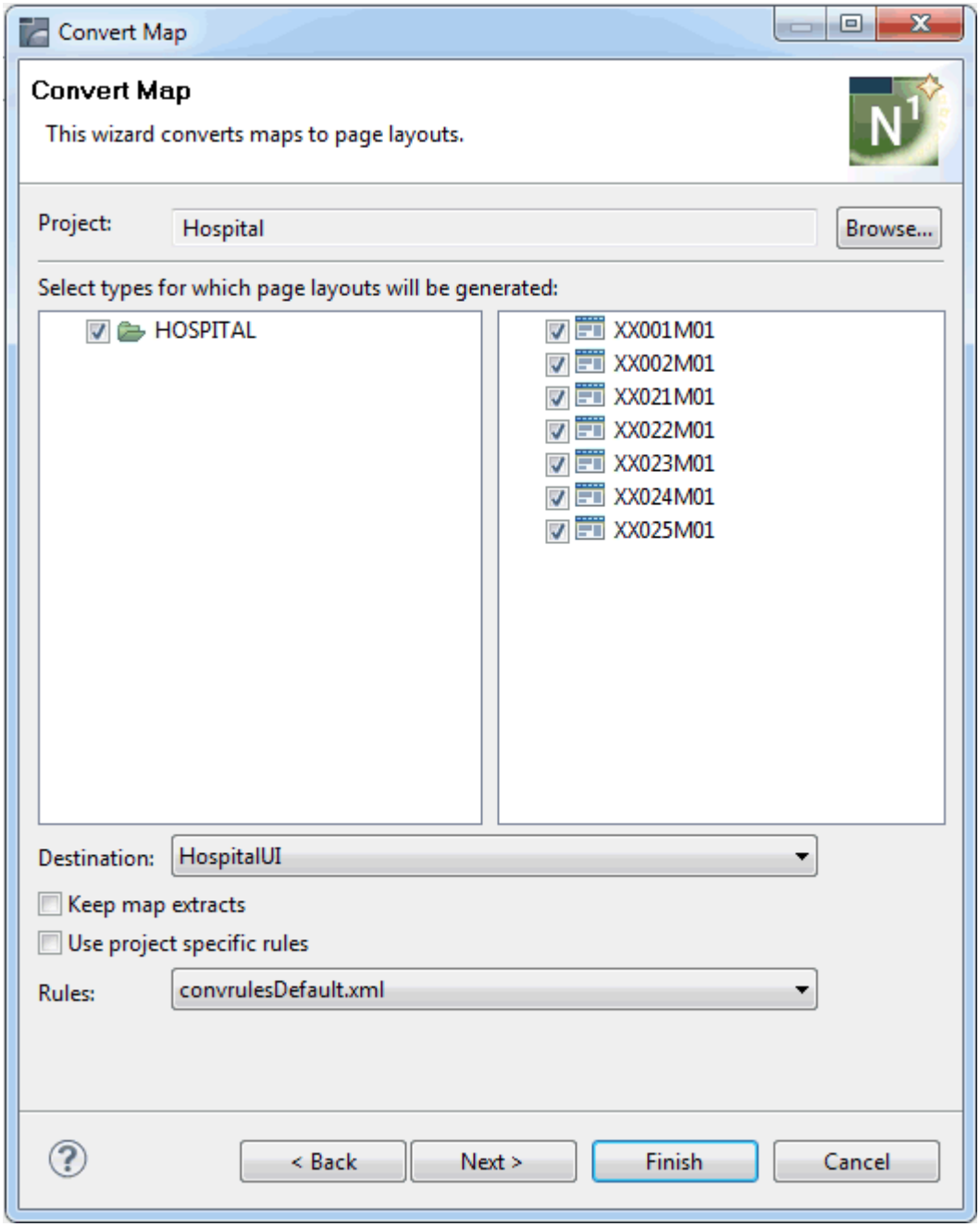
If you want to convert just a couple of maps or already existing map extract files, select them in the **Project Explorer** view or in the **Natural Navigator** view.

2 From the **File** menu, choose **New > Other**. In the resulting **New** dialog box, expand the **Natural** node, select **Map Conversion** and then choose the **Next** button.

Or:

If you have selected single maps or map extract files, invoke the context menu and choose **Convert Map**.

The following dialog box appears.



The left side of the dialog box shows the libraries in the selected project.

The right side of the dialog box shows the maps in the library which is currently selected on the left. If available, map extract files are also shown.

3 Select all maps and map extract files for which you want to generate a layout.

4 From the **Destination** drop-down list box, select the user interface component in the current project into which the layouts are to be generated.

5 If the map extract files which are generated for the selected maps are to be deleted after the generation, leave the **Keep map extracts** check box blank. If you want to keep the map extract files, select this check box.

6 If you want to use the default conversion rules and related templates, leave the **Use project-specific rules** check box empty. In this case, the **Rules** drop-down list box provides for selection the default conversion rules.

If you want to use your own project-specific conversion rules, select this check box. In this case, the **Rules** drop-down list box provides for selection the project-specific conversion rules that can be found in the *convrules* subfolder of your user interface component. See also ***Using the Conversion Rules Tool***.

7 From the **Rules** drop-down list box, select the conversion rules that you want to use.

8 Choose the **Next** button to display the optional preview pages of the wizard (see below).

Or:

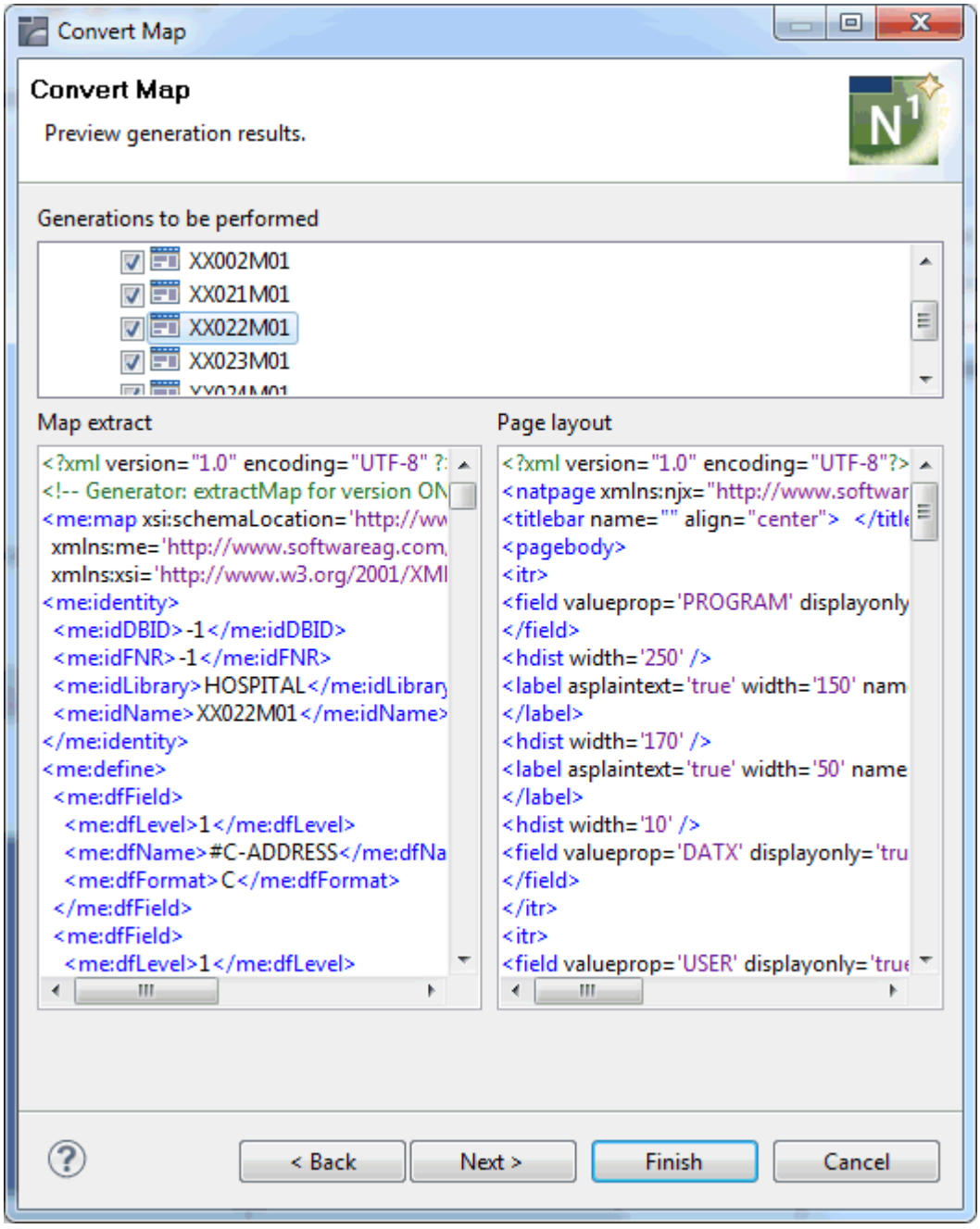
Choose the **Finish** button to perform all selected actions and to close the wizard.

The preview pages for the following actions are provided in the wizard when you choose the **Next** button repeatedly:

- Previewing the Generation Results
- Previewing the Page Layout

Previewing the Generation Results

The second page of the wizard lists all maps and map extract files that you have selected on the first page. When you select one of these entries, information such as the following is shown at the bottom of the page.

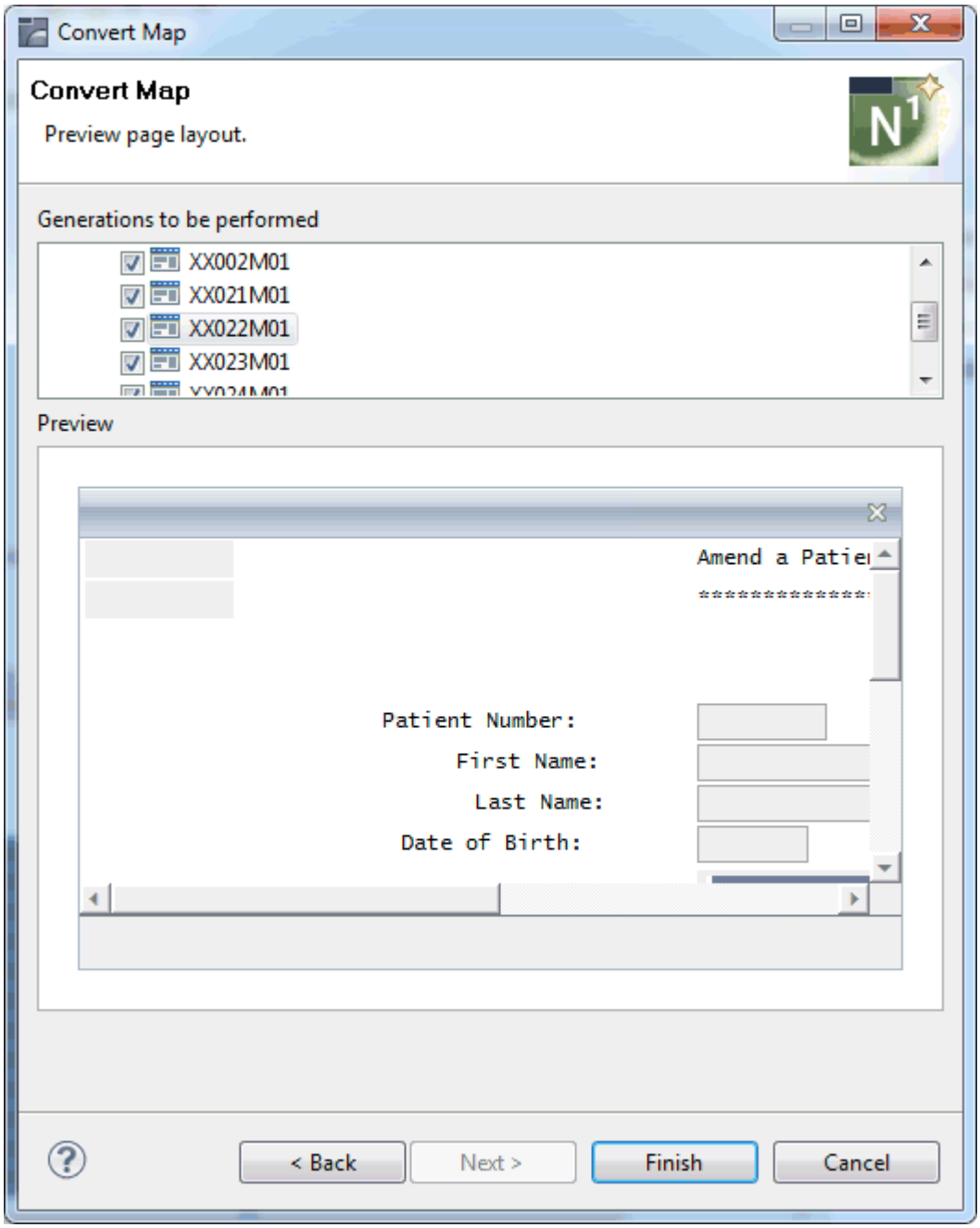


The left side of the page shows the XML code of the map extract file (either of the map extract file that will be generated for a selected map, or of an existing map extract file).

The right side shows the XML code that will be generated for the page layout.

Previewing the Page Layout

The third (and last) page of the wizard lists all maps and map extract files that you have selected on the first page. When you select one of these entries, the preview area at the bottom of this page shows the layout that will be generated for this entry. This is the page layout as it will appear in the browser.



Location of the Files

The following table explains where the different types of files can be found.

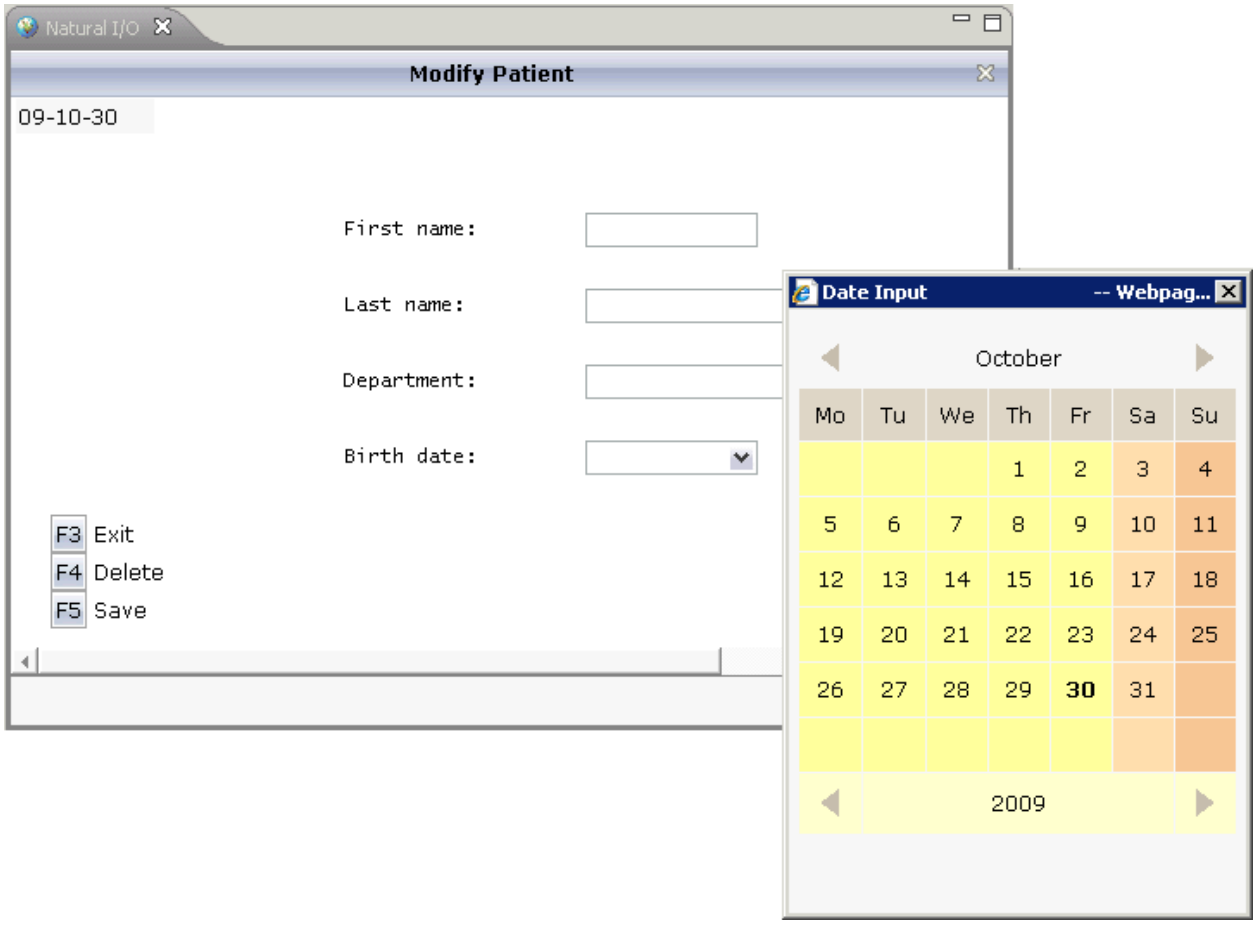
These files ...	... are stored in this folder
Adapters, maps	*SRC* subfolder of the library.
Conversion rules	*convrules* subfolder of the user interface component.
Map extract files	*RES* subfolder of the library which also contains the corresponding map (only if the **Keep map extracts** option was specified).
Page layouts	*xml* subfolder of the user interface component.

After the Conversion

The conversion process creates a page layout in the user interface component that you specified as the destination in the **Map Converter**. When you save the page layout, a Natural adapter is generated into the folder that you specified when creating a user interface component.

You will notice that the parameter data area is the same as in the original map. This is the case even though the map uses system variables and variables with special characters. The necessary translation is done inside the generated adapter code and does not influence the application code. In more complex cases, the parameter data area of the adapter will contain more fields or partly different fields than the parameter data area of the map. This depends also on the applied conversion rules.

After the conversion, you create a main program for the adapter and run it in the browser.

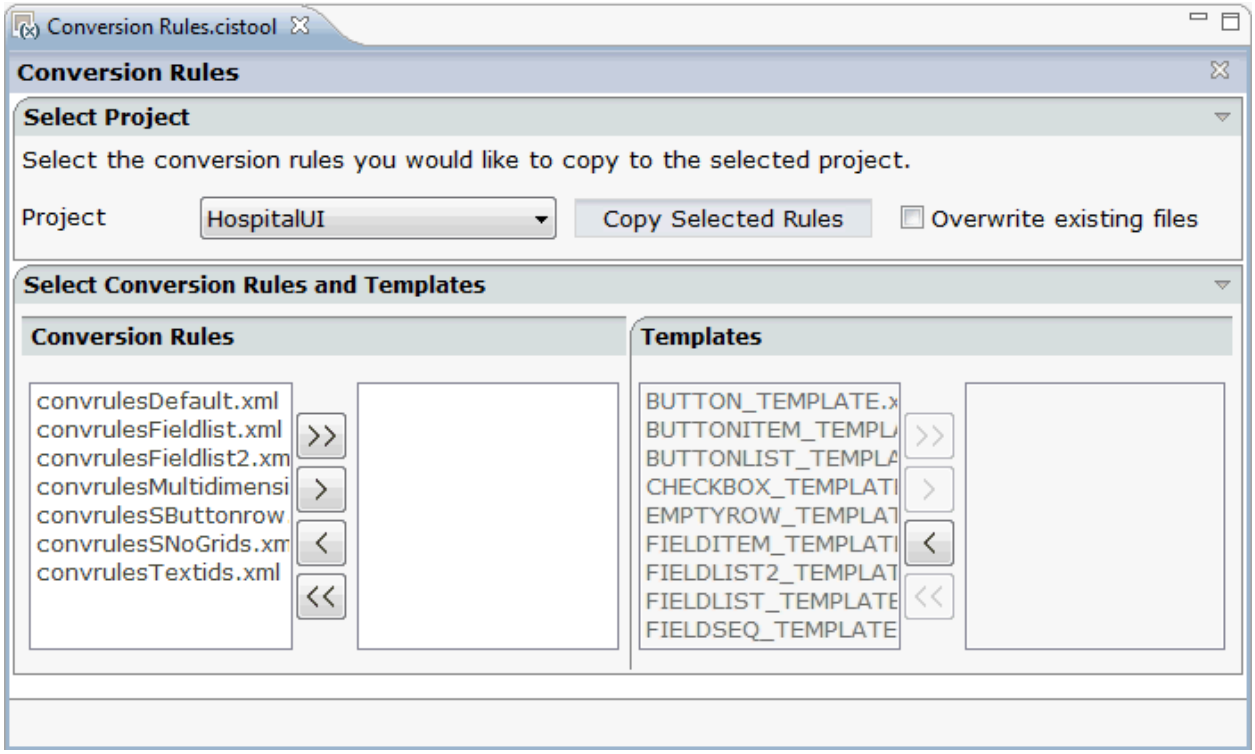


You may notice the following effects of the applied conversion rules:

- The title in the first row of the map has been placed into the caption of the page and the asterisks have been stripped off. Your application will quite surely have a different layout of the map titles. The conversion rules can therefore be adapted to accommodate the needs of your application, and the rule that identifies the title and places it into the caption is just a simple application of customizing the conversion rules.
- The literals such as "F4 Delete" on the map have each been turned into a button control and a label. This is also due to a sample conversion rule contained in the default conversion rules.
- The date field has been converted to a field control with the data type "date". This enables the user to select the date with the **Date Input** dialog box.

Using the Conversion Rules Tool

Using this tool you can copy the default conversion rules and templates to a selected user interface component for modification.



To invoke the Conversion Rules tool

1 In the **Project Explorer** view, select the Natural project for which you want to invoke the Conversion Rules tool.

2 Invoke the context menu and from the **Ajax Developer** menu, choose **Conversion Rules**.

To copy the conversion rules

1 From the **Project** drop-down list box, select the user interface component into which you want to copy the conversion rules.

2 In the **Conversion Rules** box, select the rules file(s) that you want to copy and choose the **>** button.

Or:

If you want to copy all files, choose the **>>** button.

The selected files are shown on the right side of the **Conversion Rules** box.

To deselect one or more files, you can use the **<** or **<<** button.

For each selected rules file, the templates that are used in the rules file are automatically selected in the **Templates** box, so that always a consistent set of rules and templates is selected for copying.

3 Optional. If you want to overwrite any existing rules and templates files with the same names in the selected project, activate the **Overwrite existing files** check box.

4 Choose the **Copy Selected Rules** button to copy the rules and templates files to the selected project.

Sample Conversion Rules Files

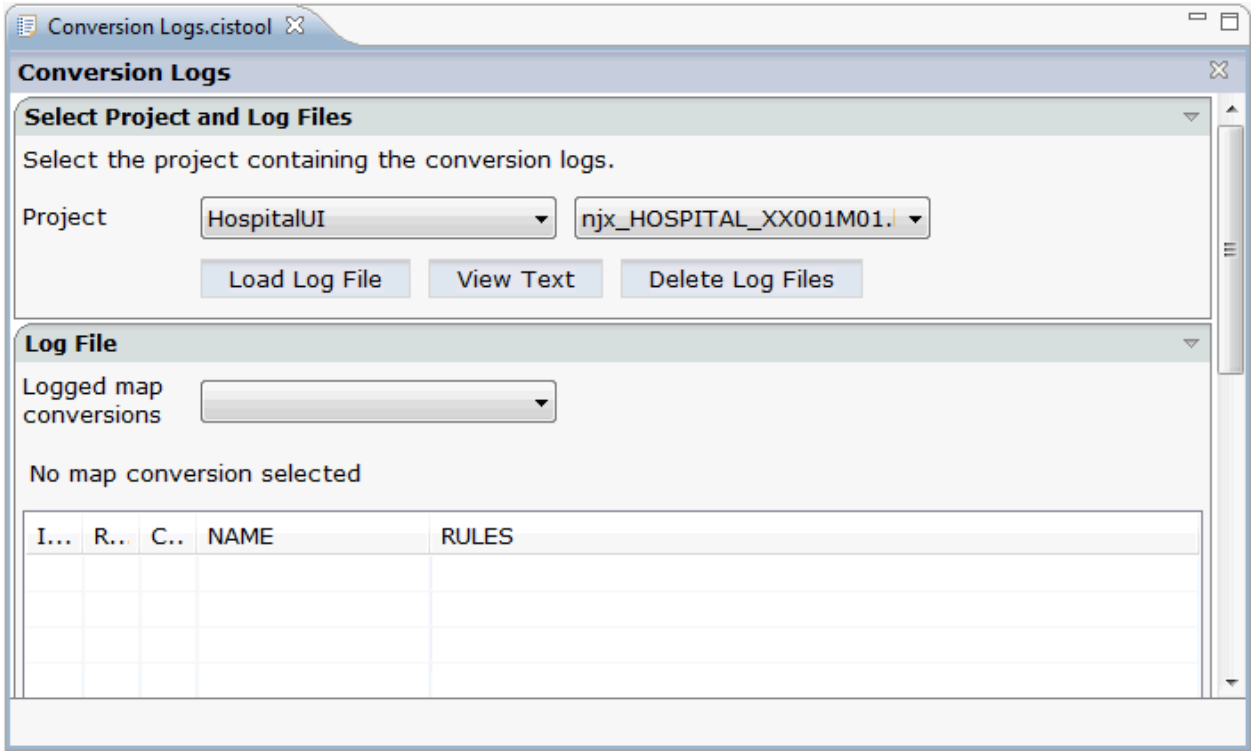
For the most common conversions the following sample conversion rules files exists:

File	Description
convrulesDefault.xml	Basic conversion rules for FIELD, TEXTGRIDSSS2 and ROWTABLEAREA2 controls.
convrulesFieldlist.xml	Example rules for NJX:FIELDLIST and NJX:FIELDVALUE controls.
convrulesFieldlist2.xml	Example rules for NJX:FIELDLIST and NJX:FIELDITEM controls.
convrulesMultidimensionalArrays.xml	Example rules for ROWTABLEAREA3.
convrulesSButtonrow.xml	Example rules for arranging all identified functions as buttons in a row at the bottom of the page layout.
convrulesSNoGrids.xml	Arrays are also mapped to simple FIELD controls. No grid controls are generated.
convrulesTextids.xml	Example rules for using a conversion listener to generate TEXTIDs (= multilanguage support).
convrulesCVVariables.xml	Example rules to generate the corresponding attributes and data controls for CV variables in the page layout.

See also the comments on top of each conversion rules file for further details.

Using the Conversion Logs Tool

Using this tool you can view the log files that have been created during the conversion of Natural maps to Natural for Ajax layouts. You can also delete these log files.



To invoke the Conversion Logs tool

1 In the **Project Explorer** view, select the Natural project for which you want to invoke the Conversion Logs tool.

2 Invoke the context menu and from the **Ajax Developer** menu, choose **Conversion Logs**.

To view a log file

1 From the **Project** drop-down list box, select the user interface component for which you want to view a log file.

The log files contained in this user interface component are shown in the drop-down list box to the right.

2 Select the log file that you want to view.

3 Choose the **Load Log File** button.

Log lines for the selected log file are now shown at the bottom of the tool. Each log file contains the conversion results of one or several maps. The log lines that are shown belong to an individual map; this is the map that is selected in the **Logged map conversions** drop-down list box.

4 Optional. Select a different map from the **Logged map conversions** drop-down list box.

The conversion result of the newly selected map is immediately shown at the bottom of the tool.

5 Optional. Choose the **View Text** button to display the content of the selected log file as a CSV file in a dialog. This shows the conversion results for all maps.

To delete log files

1 Select the project for which you want to delete the log files.

2 Choose the **Delete Log Files** button.

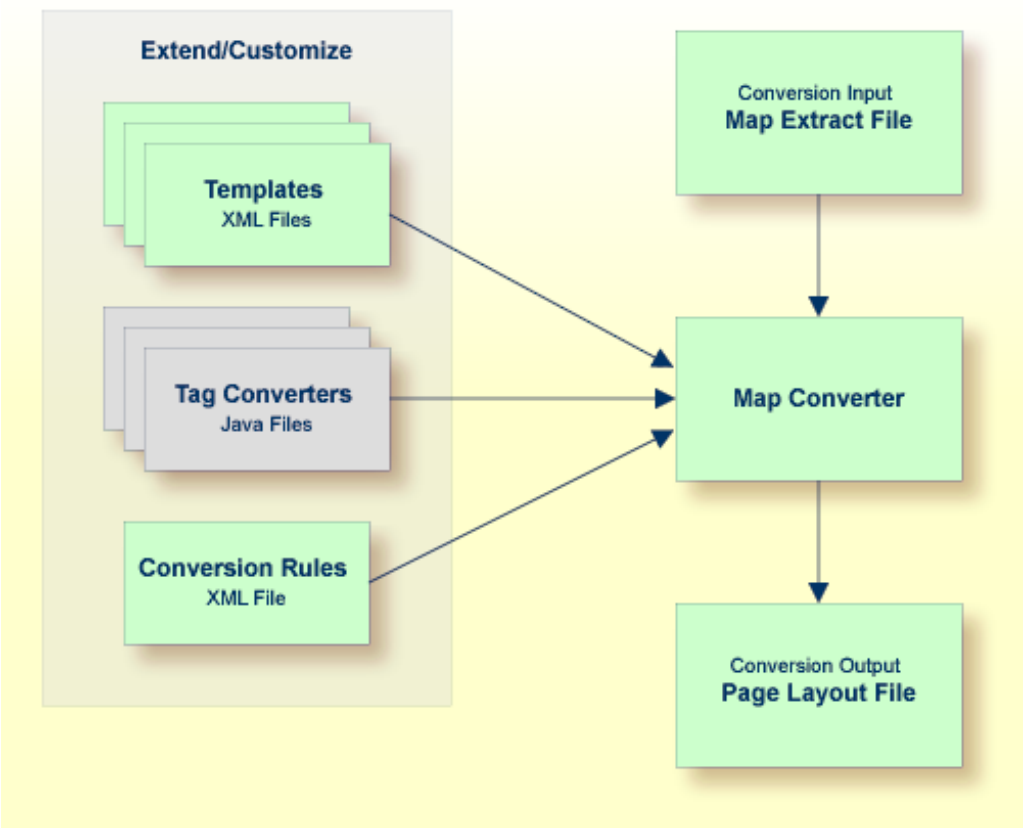
A dialog appears asking to confirm the deletion.

3 Choose the **Yes** button to delete all log files in the selected project.

4 Customizing the Map Conversion Process

Map Converter Processing

The map conversion process reads a map extract file and transforms it into a corresponding Natural for Ajax page layout file. The conversion process is controlled by rules and templates.



The Map Converter ships with a default set of conversion rules and corresponding template files. This set allows for default map conversions without changing rules or templates. In most cases, you will add or modify some conversion rules and/or templates to customize the conversion according to the requirements of your application.

For advanced customization, there is also the possibility to plug own Java-written conversion classes (the so-called “tag converters”) into the conversion processing. But you should only do this in very rare cases.

The following topics are covered below:

- Processing of Rows and Columns
- Processing of Sequence and Grid Areas

- Summary: Processing Steps of the Map Converter

Processing of Rows and Columns

By default, for each row and column in a map, a corresponding row and column is generated in the layout. By default, the Map Converter inserts the converted rows and columns at a defined position within a corresponding page template. Template and insert position can be defined by the user. Skipping or different handling of specific rows and columns can be defined via corresponding conversion rules.

The following sections describe the default processing for rows and columns in case no specific rules for different insert positions are specified:

- Rows
- Columns

Rows

For each row in a map, the Map Converter generates an ITR (independent table row) control with the default settings. For empty rows, an ITR control containing the control defined in the *EMPTYROW_TEMPLATE* is generated.

Columns

The fields and literals within a row are aligned to columns according to the following rules:

- **Column Start Position**
  If an absolute column start position is defined for a field or literal in the map, the corresponding control in the page layout is aligned so that it starts exactly with the specified column. This is done by inserting a HDIST (horizontal distance) control with a corresponding width as a filler.

  Note: A precise vertical alignment of fields is only possible if absolute column start positions are defined for the fields.

- **Conversion Rules**
  If no absolute column start position is defined for a field or literal in the map, a HDIST control is not added as a filler by default. In this case, the field or literal is simply appended as the last subnode of the current ITR control. In many cases, this would result in a layout that requires additional manual adding of fillers. This is because appending two field controls without adding any HDIST control often does not look as intended. Therefore, the Map Converter includes default conversion rules for filler settings. You can modify the default conversion rules or add your own conversion rules to fine-tune this behavior. For more information, see ***Conversion Rules***.

- **Column Width**
  A character map has a fixed number of rows and columns. For the literal "ABCD", this means that it uses exactly 4 columns. Calculating the correct width and height of field on a web page is more complex. The width of "ABCD" will most likely be greater than the width of "llll". Very short fields (with a length of one or two characters) should have a minimum width so that the content is fully visible. You can fine-tune the width by adapting the predefined conversion rule variable `$$widthfactor$$` or by adding your own conversion rules. For more information, see *Conversion Rules*.

Processing of Sequence and Grid Areas

The map extract file also contains information about arrays. With Application Designer, arrays are usually rendered as grid controls. Application Designer provides a couple of grid controls. These are:

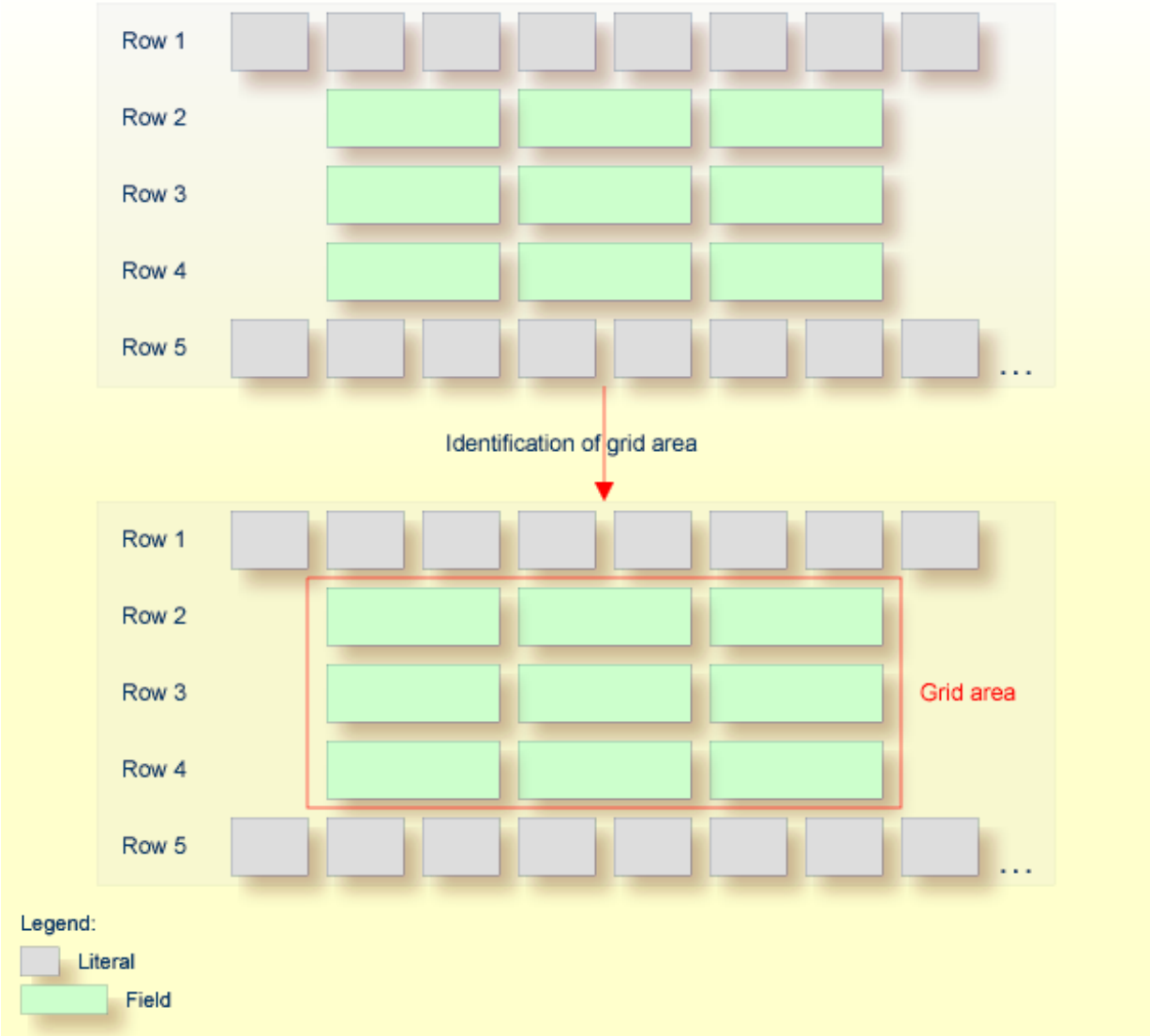
- TEXTGRID2 - a grid containing text.
- TEXTGRIDSSS2 - a text grid with server-side scrolling.
- ROWTABLEAREA2 - a grid containing other controls.
- MGDGRID - a managed grid.

You can find more details on these grid controls ...

- ... in the context of Natural: *Natural Page Layout* > *Working with Grids*
- ... in the context of Java: *Java Page Layout* > *Working with Grids*

The Map Converter tries to convert arrays into suitable grid controls. Before the real conversion of arrays to grid controls can be done, the Map Converter must first identify the sequence and grid areas on the map. During this process of area identification, the Map Converter groups literals and fields together into sequences and areas. Whether the corresponding fields or literals are actually converted into a grid depends on the conversion rules that are executed after this area identification step.

This process of area identification is simply a kind of marking. The corresponding sequence and area objects can be used as source in the conversion rules to define the actual controls.



Summary: Processing Steps of the Map Converter

The conversion is done in several steps:

1. The map extract file is loaded and the corresponding rows and columns are collected.
2. The sequence and grid areas are identified.
3. For each row, the list of items in this row is processed, according to the column order. An item can be one of the following: a simple literal, a field or an area. For each found item, the corresponding conversion rules are executed.

Conversion Rules

Different conversion projects have different requirements to the conversion process. The Map Converter is driven by conversion rules and thus allows for flexible control of the conversion process. Conversion rules define how source items (items from a given map extract file) are mapped to target items (items in the page layout to be created) and under which conditions a certain source item shall be converted to a certain target item. The Map Converter is delivered with a default set of conversion rules contained in the file *convrulesDefault.xml* in the subdirectory *convrules* in the Application Designer project *njxmapconverter*. A more application-specific conversion can be achieved by copying and modifying the default set of rules or by adding own rules.

Each set of conversion rules is defined in an XML file according to the XML schema *convrules.xsd* in the subdirectory *convrules* in the Application Designer project *njxmapconverter*. Each individual conversion rule consists of a name, a description, a source and a target. The source identifies an element in the map extract file. The target identifies controls and attributes to be generated in the page layout.

The conversion rules make often use of regular expressions and so-called capture groups. For more information about regular expressions, see for instance the web site ***http://www.regular-expressions.info***.

The following topics are covered below:

Conversion Rules Examples

The following examples are provided:

- Example 3

Example 1

The following example rule (contained in the default conversion rules file) defines that fields in the map extract file with the qualification `AD=O` shall be converted to field controls with the property `displayonly="true"`.

```
<convrule rulename="Ofield_rule">
  <description>Defines the control template to be used for input fields
  which are specified as output only.</description>
  <source>
    <sourceitem>ifField</sourceitem>
    <sourcecond>
      <condattr>//ifAD</condattr>
      <condvalue>.*O.*</condvalue>
    </sourcecond>
  </source>
  <target>
    <targetitem>$OFIELD_TEMPLATE</targetitem>
  </target>
</convrule>
```

The source element specifies that this rule applies to fields (element `ifField`) that have an `AD` parameter (element `ifAD`) that contains a letter "O" (matching the regular expression `.*O.*`). The target element specifies that these fields are to be converted to whatever is contained in the template file *OFIELD_TEMPLATE.xml*. This template file must be contained in the same directory as the conversion rules file.

The template file contains the detailed specification of the field to be generated. The file *OFIELD_TEMPLATE.xml* delivered with the map converter contains, for instance, the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<field valueprop="$$" width="$$" noborder="true" displayonly="true"/>
```

That is, the resulting field is generated without a border (`noborder="true"`) and as a display-only field (`displayonly="true"`). The `valueprop` and `width` to be assigned (`$$`) are not determined by this rule, but are left under the control of other rules.

Example 2

The following example rule (contained in the default conversion rules file) defines that for all fields that are defined with the format A*n* in the map extract file, an attribute `datatype="string n"` shall be added to the element that is generated into the page layout.

```
<convrule rulename="AfixType_rule">
  <description>All Natural "An" dfFields are converted to the
  Application Designer datatype "string n". Example: "A10" is
  converted to "string n".</description>
  <source>
    <sourceitem>dfField</sourceitem>
      <selection>
        <selectattr>dfFormat</selectattr>
        <selectval>A([0-9]+)</selectval>
      </selection>
    </source>
    <target>
      <targetitem>$$</targetitem>
      <targetattr>
        <attrname>datatype</attrname>
        <attrvalue>string $1</attrvalue>
      </targetattr>
    </target>
</convrule>
```

The source element specifies that this rule applies to fields that have in the field definition (element `dfField`) a format (element `dfFormat`) of A*n* (matching the regular expression `A([0-9]+)`). The target element specifies that for whatever element is generated into the page layout for this kind of fields, an attribute `datatype="string $1"` shall be added. In terms of regular expressions, `$1` refers to the contents of the first “capture group” of the regular expression `A([0-9]+)`. In case of a format A20, `$1` will evaluate to 20 and thus an attribute `datatype="string 20"` will be generated.

The control to be generated into the page layout (`<targetitem>$$</targetitem>`) is not determined by this rule, but is left under the control of other rules.

Summary: The combination of the two rules in example 1 and 2 makes sure that output fields, for example, of format A20 are converted to field controls with `displayonly="true"` and `datatype="string 20"`.

Example 3

The following more advanced rule was created for the use of a specific conversion project. The following task had to be achieved: A literal of the format "F10 Change" shall be converted to a button that is named "F10", is labeled "Change" and raises an event named "PF10". With the explanations from the examples above, the rule should be nearly self-explanatory.

Note that according to the rules of regular expressions, the variable `$1` refers to the string matched by the expression part in the first pair of parentheses (the first “capture group”), that is for instance "F10", and the variable `$3` refers to the string matched by the expression part in the third pair of parentheses (the third “capture group”), that is for instance "Change".

```
<convrule rulename="Function_rule" lone="true">
<description>Generates a button from specific literals.</description>
  <source>
    <sourceitem>ltLiteral</sourceitem>
    <selection>
      <selectattr>ltName</selectattr>
      <selectval>(F[0-9]+)(\p{Space})(.*)</selectval>
    </selection>
  </source>
  <target>
    <targetitem>$BUTTON_TEMPLATE</targetitem>
    <targetattr>
      <attrname>name</attrname>
      <attrvalue>$1</attrvalue>
    </targetattr>
    <targetattr>
      <attrname>method</attrname>
      <attrvalue>P$1</attrvalue>
    </targetattr>
  </target>
  <target>
    <targetitem>hdist</targetitem>
    <targetattr>
      <attrname>width</attrname>
      <attrvalue>4</attrvalue>
    </targetattr>
  </target>
  <target>
    <targetitem>label</targetitem>
    <targetattr>
      <attrname>name</attrname>
      <attrvalue>$3</attrvalue>
    </targetattr>
  </target>
</convrule>
```

Default Conversion Rules File

The Map Converter is delivered with a default set of conversion rules contained in the file *convrulesDefault.xml* in the subdirectory *convrules* in the Application Designer project *njxmapconverter*. A more application-specific conversion can be achieved by copying and modifying the default set of rules or by adding own rules.

The following topics are covered below:

- Root Rule
- Data Type Conversion Rules
- Other Default Conversion Rules

Root Rule

Like every conversion rules file, the file contains exactly one "Root_rule". The root rule specifies the template file to be used for the overall page layout. In this template file, the application-specific page layout can be defined, using company logos, colors, fonts, etc. The root rule must always have "map" as the source item and must refer to some variable defined in the page template file as the target item. The place of that variable specifies where in the page template the converted map items are placed. See for instance the root rule from the default conversion rules:

```
<convrule rulename="Root_rule">
  <description>Exactly one rule with the sourceitem "map" is required.
  This rule must define the natpage template and insert position of
  the conversion result.</description>
  <source>
    <sourceitem>map</sourceitem>
  </source>
  <target>
    <targetitem>$NATPAGE_TEMPLATE.$MAPROOT</targetitem>
  </target>
</convrule>
```

The rule refers to a page layout template *NATPAGE_TEMPLATE.xml* and refers to a variable defined in that template where the converted map elements shall be placed. Here is the corresponding content of the page layout template *NATPAGE_TEMPLATE.xml*:

```
<?xml version="1.0" encoding="UTF-8"?>
<natpage xmlns:njx=http://www.softwareag.com/njx/njxMapConverter
  natsource="$$NATSOURCE$$" natsinglebyte="true">
  <titlebar name="$$TITLEVAR$$" align="center">
  </titlebar>
  <pagebody>
    <njx:njxvariable name="MAPROOT"/>
  </pagebody>
  <statusbar withdistance="false"/>
</natpage>
```

This template specifies the following:

- The overall page layout shall consist of the elements `titlebar`, `pagebody` and `statusbar`.
- The converted map elements shall be placed into the `pagebody`.
- The name of the Natural adapter to be generated from that page layout shall be determined by a rule (`natsource="$$NATSOURCE$$"`). There must be a corresponding rule that yields a value for the variable `$$NATSOURCE$$`, for instance derived from the map name. We shall see later how to define such a rule.
- All strings in the page layout shall be mapped to Natural variables of type A in the adapter interface (`natsinglebyte="true"`).
- The text displayed in the title bar shall be determined by a rule (`name="$$TITLEVAR$$"`). There must be a corresponding rule that yields a value for the variable `$$TITLEVAR$$`, for instance derived from a literal in the first row in the map. We shall see later how to define such a rule.

Data Type Conversion Rules

The default conversion rules file contains a set of rules that control the conversion of data types: from Natural data types in the map to corresponding Application Designer data types in the page layout. An example was given above in ***Example 2***. Usually, these rules need not be adapted. They have been chosen in such a way that the process of extracting maps, converting them to layouts and generating Natural adapters for these usually yields the same data types in the adapter interface as in the map interface.

Other Default Conversion Rules

Other default conversion rules define a default mapping for literals, modifiable fields, output fields, modifiable grids, output grids, system variables and fields with special characters like "#" in their names. These rules need only be adapted in special cases.

Conversion Rules that Often Need to be Adapted

Some conversion rules need to be adapted in nearly all conversion projects. These rules are contained in the section "APPLICATION SPECIFIC RULES" in the default conversion rules file.

The following topics are covered below:

- Naming of Adapters

Naming of Adapters

Each application has a different naming convention for Natural objects. There is a rule (it is named "Natsource_rule" in the default conversion rules file) that controls how adapter names are derived from map names. The rule replaces the first letter "M" in the map name with an "A" and places the resulting string into the variable `NATSOURCE`. Remember that in the default page template, the `natsource` property of NATPAGE (which defines the adapter name to generated) is preset with the variable reference `$$NATSOURCE$$`. Thus, a map with the name `TESTM1` results in an adapter named `TESTA1`. Other naming conventions for maps will require a more sophisticated adapter naming rule.

Setting the Title of a Map

Each application has a different way of showing titles in a map. Often, the title string shall be placed into the title bar of the resulting page layout during conversion. There is a rule (in the default conversion rules file, it is named "Titlevar_rule") that controls how the title string in a map is recognized. The rule searches in the first row of a map for a literal enclosed in "***" and places the resulting string into the variable `TITLEVAR`. Remember that in the default page template, the `name` property of the `titlebar` element (which defines the string to be shown in the title bar) is preset with the variable reference `$$TITLEBAR$$`. So this rule takes care that the found literal is placed into the `titlebar` element of the page. Other conventions for map titles will require a more sophisticated rule.

Writing Your Own Conversion Rules

When writing your own conversion rules, you can use the default rules as examples. In order to write rules from scratch, you need to know the elements of the map that can be referred to as source items and the full syntax of the rule definition.

- The XML schema of the map extract files is contained in the file *naturalmap.xsd* in the subdirectory *convrules* in the Application Designer project *njxmapconverter*.
- As described in ***Processing of Sequence and Grid Areas***, one step in the map conversion is the detection of sequence and grid areas in the map. Conversion rules can also refer to the detected sequence and grid areas. The XML schema of the map extract files after the detection of sequence and grid areas is described in the extended XML schema *naturalmapxml_extended.xsd* in the same directory.
- The syntax of the conversion rules is described by the XML schema *convrules.xsd* in the same directory.

The basic structure of a conversion rule is as follows:

```
<convrule rulename="...">
  <description>...</description>
  <source>...</source>
  <target>...</target>
  <target>...</target>
  ...
</convrule>
```

This means, a conversion rule consists of one `source` element and (optionally) one or several `target` elements. The `source` element identifies an item from the map. The `target` elements specify the conversion output. If no `target` elements are specified, nothing is generated from the identified `source` element.

The basic structure of a `source` element is as follows (example):

```
<source>
  <sourceitem>ltLiteral</sourceitem>
    <selection>
      <selectattr>ltName</selectattr>
      <selectval>\*\*\*(.*)\*\*\*</selectval>
    </selection>
    <sourcecond>
      <condattr>ltRow</condattr>
      <condvalue>1</condvalue>
    </sourcecond>
</source>
```

The `sourceitem` element refers to a specific kind of item on a map, such as a literal (`ltLiteral`), a defined field (`dfField`), an input field (`ifField`) or the identifier of the map (`identity`). The elements that can be used here are specified by the XML schema that describes the map extract after the detection of sequence and grid areas (*naturalmapxml_extended.xsd*). Therefore, the elements `sequenceArea` and `gridArea`, which are only known after this processing, can also be used here.

The `selectattr` and `selectval` elements are used to match an element of a specific kind by its attribute values. The `selectval` element uses regular expressions to perform a match. Capturing groups such as `(.*)` can be used here, so that the target part of the conversion rule can later refer to parts of the matched value.

The `selectattr` element not only accepts single attributes but also XPATH expressions. You can find an example for the usage of XPATH expressions in the file *convrulesSNoGrids.xml*:

```
<source>
<sourceitem>ifField</sourceitem>
 <selection>
  <selectattr>ifIndex/ifOffset</selectattr>
  <selectval>([1-9]*)</selectval>
 </selection>
</source>
```

In the above example all `ifIndex/ifOffset` values, which are subnodes of the currently processed `ifField` are found. For each value found it is checked whether it matches the regular expression specified in the `selectval` element. Only if all values found match the regular expression, the capturing is done on the concatenated found values. If any values found do not match the regular expression, the rule is not applied to the `ifField`.

When you are using XPATH expressions, it is important to keep the two-step process in mind:

- matching for each single value and
- capturing on the concatenated values.

Finally, there can be zero, one or several `sourcecond` elements, which allow to define further to which map items the rule applies. If several `sourcecond` elements are specified, the rule is triggered only if all conditions match (logical AND).

The basic structure of a `target` element is as follows:

```
<target>
  <targetitem>...</targetitem>
  <targetattr>
    <attrname>...</attrname>
    <attrvalue>...</attrvalue>
  </targetattr>
  <targetattr>
    ...
  </targetattr>
  ...
</target>
```

In detail, there are several different options to specify a target item:

- Specify the root element name of an Application Designer control, along with its attributes and attribute values. The attribute value can be a constant, a variable or a reference to a capturing group from a regular expression in a `sourcecond` element of the same rule. In this case, the corresponding control is generated during conversion.

```
<target>
  <targetitem>label</targetitem>
  <targetattr>
    <attrname>height</attrname>
    <attrvalue>10</attrvalue>
  </targetattr>
  <targetattr>
    <attrname>width</attrname>
    <attrvalue>$$width$$</attrvalue>
  </targetattr>
  <targetattr>
    <attrname>name</attrname>
    <attrvalue>$1</attrvalue>
```

```
  </targetattr>
</target>
```

- Specify the name of a variable that is defined in the conversion rules file in a `convvariable` element.

```
<target>
  <targetitem>$$name$$</targetitem>
</target>
```

- Refer to the name of a template file, optionally along with attribute names and values. In this case, whatever is contained in the template file will be generated. Attribute definitions in the template file are replaced.

```
<target>
  <targetitem>$BUTTON_TEMPLATE</targetitem>
  <targetattr>
    <attrname>name</attrname>
    <attrvalue>$1</attrvalue>
  </targetattr>
  <targetattr>
    <attrname>method</attrname>
    <attrvalue>P$1</attrvalue>
  </targetattr>
</target>
```

- Refer to the name of a template variable and the name of a template file, separated by a dot. In this case, the template variable is replaced with whatever is contained in the template file.

```
<target>
  <targetitem>$GRIDITEM.$GRIDITEM_TEMPLATE</targetitem>
</target>
```

- Only in the root rule: Specify the name of a template file and the name of a template variable that is contained in this file, separated by a dot. In this case, the template variable is replaced with the entire result of the map conversion.

```
<target>
  <targetitem>$NATPAGE_TEMPLATE.$MAPROOT</targetitem>
</target>
```

- Specify "$$" as the target item. This is useful when writing a more general rule that is to apply after another more specific rule has already created a target item. The attributes specified along with the target item "$$" are applied to the already created target item, whatever this target item was.

```
<target>
  <targetitem>$$</targetitem>
  <targetattr>
    <attrname>datatype</attrname>
    <attrvalue>xs:double</attrvalue>
  </targetattr>
</target>
```

- Specify "$." as the target item. This refers to the template that is currently being processed. The attributes specified along with the target item "$." are applied to the current template.

```
<target>
  <targetitem>$.</targetitem>
  <targetattr>
    <attrname>$$NATSOURCE$$</attrname>
    <attrvalue>$1-A</attrvalue>
  </targetattr>
</target>
```

Templates

The Map Converter assembles page layouts from templates. Which templates are used, how they are assembled and how variables in templates are filled is controlled by the conversion rules.

A template file describes the general layout of an entire Application Designer page layout or of an individual Application Designer control. A template can contain variables and references to other templates. During conversion, the Map Converter resolves the structure of the templates and fills the variables with specific values, depending on the contents of the map.

A template file can describe a simple control such as a FIELD control or a more complex control such as a TEXTGRIDSSS2 control. For the same control, multiple templates may exist. For example, an *ofield_TEMPLATE* and an *ifield_TEMPLATE* may both be templates for the FIELD control. The *ofield_TEMPLATE* would be used for output fields, the *ifield_TEMPLATE* for modifiable fields. Which template is used for which subset of fields of the map is specified in the conversion rules.

Template files are well-formed XML files which contain control definitions. They are placed in the folder *convrules* of your Application Designer project directory. The file name must end with "_TEMPLATE.xml". The Map Converter ships with a set of default template files.

The following topics are covered below:

- Variables in Templates
- Templates in Templates

- Editing Templates

Variables in Templates

As already seen in the examples above, templates can contain variables. Variables can be freely defined by the user. Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<natpage xmlns:njx=http://www.softwareag.com/njx/njxMapConverter
  natsource="$$NATSOURCE$$" natsinglebyte="true">
  <titlebar name="$$TITLEVAR$$" align="center">
  </titlebar>
  <pagebody>
    <njx:njxvariable name="MAPROOT"/>
  </pagebody>
  <statusbar withdistance="false"/>
</natpage>
```

- **Variables as placeholders for the property values of controls**
  An example is the variable `$$TITLEVAR$$` in the template above. If a template contains a variable such as `name="$$TITLEVAR$$"`, there must be a corresponding rule that yields a value for the variable `$$TITLEVAR$$`. The Map Converter replaces the variable with this value.

  The built-in variable `$$` has a specific meaning. If it occurs as a property value, there is no specific rule needed to produce the value. Instead, the Map Converter receives the value from a so-called tag converter. Tag converters are Java classes that are delivered with the Map Converter. Exchanging or writing your own tag converters is an advanced way of extending the Map Converter and is usually not required. See *Tag Converters* for further information.
- **Variables as placeholders for controls and containers**
  An example is the variable `MAPROOT` in the template above. Such a variable is defined by inserting an NJX:NJXVARIABLE control (from the controls palette of the Layout Painter) into a template. As long as the XML of the template is well-formed, an NJX:NJXVARIABLE control can be inserted at any place in the template. Conversion rules refer to this variable as `$MAPROOT`. Notice that the value in the `name` property of an NJX:NJXVARIABLE control does not start with $. Instead, the NJX:NJXVARIABLE control itself defines that it is a variable. The NJX:NJXVARIABLE control is a special control in the **Natural Extensions** section of the Layout Painter's controls palette.

Templates in Templates

Templates can refer to other templates. This can be done via adding variables. The variable can serve as a placeholder for another template. The template name is defined via a corresponding rule.

Example (*GRID_TEMPLATE.xml*):

```
<?xml version="1.0" encoding="UTF-8"?>
<rowtablearea2 withborder="false" griddataprop="$$gridname$$" rowcount="$$" >
  <tr>
    <hdist></hdist>
    <njx:njxvariable name="GRIDHEADER" />
  </tr>
  <repeat>
    <tr>
      <hdist></hdist>
      <njx:njxvariable name="GRIDITEM" />
    </tr>
  </repeat>
</rowtablearea2>
```

This means: A conversion rule like the following maps a grid area detected in the map to a ROWTABLEAREA2 control and formats the header and rows as specified in the templates *GRIDHEADER_TEMPLATE.xml* and *GRIDITEM_TEMPLATE.xml*.

```
<convrule rulename="Griditem_rule">
    <description>Mapping rule for the items of grid.</description>
    <source>
      <sourceitem>gridArea//ifField</sourceitem>
    </source>
    <target>
      <targetitem>$GRIDITEM.$GRIDITEM_TEMPLATE</targetitem>
    </target>
    <target>
      <targetitem>$GRIDHEADER.$GRIDHEADER_TEMPLATE</targetitem>
    </target>
</convrule>
```

Editing Templates

Only NATPAGE templates (like the default NATPAGE template *NATPAGE_TEMPLATE.xml*) can be edited with the Layout Painter. Templates for individual controls must currently be edited using a text editor.

Tag Converters

A template must be a valid XML document. The root element must correspond to the root element of a valid Application Designer control. Templates can contain variables. A special variable is the variable `$$`.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<button name="$$" method="$$"></button>
```

Each template is processed by a so-called tag converter. Tag converters are in charge of resolving the variable `$$`. A tag converter is a Java class that must support a specific interface and be available in the class path of the Map Converter. Which tag converter is used depends on the root element of the template.

In the above example, the root element is the BUTTON control. The following rule applies:

- If a Java class with the name `com.softwareag.natural.mapconverter.converters.BUTTONConverter` is found in the Java class path, this Java class is used as the tag converter.
- Otherwise, the class `com.softwareag.natural.mapconverter.converters.DEFAULTConverter` is used as the tag converter.

In the above example, the Map Converter tries to find the class `BUTTONConverter` first. Since a specific tag converter for the BUTTON control is not delivered with the Map Converter, the class `DEFAULTConverter` is used as the tag converter.

In order to supply a custom tag converter for the BUTTON control, for instance, you would have to create a Java class `BUTTONConverter` that belongs to the package `com.softwareag.natural.mapconverter.converters` and make it available in the Java class path of the Map Converter.

Detailed information on how to write your own tag converters is provided in the Application Designer development workplace as Javadoc; see **Map Converter Extension API** in the **Natural Tools** node of the navigation frame (under **Tools & Documentation**).

5 Code Conversion

General Information

After the **Map Converter** has been used to create page layouts from map extract files, the last step in the conversion process is adapting the application code to the new user interface. This step can either be performed manually or, with Natural Engineer, partly automatically. In the following, the manual code conversion is described.

Generating Adapters

First of all, it is necessary to generate HTML code and Natural adapters from the page layouts that have been created by the Map Converter. This is the same procedure as with page layouts that have been created manually with the Layout Painter. Then, the adapters are imported into the Natural development environment.

Structure of a Map-Based Application

In this context, we need not consider the application code as a whole, but only the layer that handles the user interface. Often, the user interface handling part of a map-based application is structured in the following way:

- `DEFINE DATA`
- Initialization
- `REPEAT`
  - `INPUT [USING MAP map-name]`
    - Includes client-side validations (processing rules)
  - Server-side validations
    - `REINPUT` or `ESCAPE TOP`
  - `DECIDE ON *PF-KEY`
    - Function key handler 1
      - Processing
      - `REINPUT` or `ESCAPE TOP`
    - Function key handler 2
      - Processing
      - `REINPUT` or `ESCAPE TOP`

    - Function key handler *n*
      - Processing
      - ESCAPE BOTTOM
    - ...
  - END-DECIDE
- END-REPEAT
- Cleanup
- END

In practice,

- the REPEAT loop might or might not be there, and
- there might not be a clean DECIDE structure for the function key handlers. Instead, checks for the pressed function key might be spread all over the code.

However, accepting these differences, the above structure should match a large number of applications.

Structure of a Natural for Ajax Application

The corresponding part of a Natural for Ajax application looks as follows:

- DEFINE DATA
- Initialization
- REPEAT
  - PROCESS PAGE USING *adapter-name*
    - Includes client-side validations
  - Server-side validations
    - PROCESS PAGE UPDATE FULL
  - DECIDE ON *PAGE-EVENT
    - Event handler 1
      - Processing
      - PROCESS PAGE UPDATE FULL or ESCAPE TOP
    - Event handler 2
      - Processing
      - PROCESS PAGE UPDATE FULL or ESCAPE TOP

        - Event handler *n*
            - Processing
            - `ESCAPE BOTTOM`
        - ...
    - `END-DECIDE`
- `END-REPEAT`
- Cleanup
- `END`

Tasks of the Code Conversion

The code conversion should achieve the following:

- It should be minimal invasive.
- It should not duplicate business code.
- The converted application should be able to run not only with the new user interface, but also in a terminal session, in a Natural Web I/O Interface session and in batch, if it did so before the code conversion.

In detail, the code conversion needs to deal with the statements and constructs mentioned below.

DEFINE DATA Statement

The `DEFINE DATA` statement must be extended because the data structures exchanged between a program and map are not fully identical to those exchanged between a program and the corresponding adapter.

The default conversion rules delivered with the Map Converter perform a data type mapping that tries to ensure that the data elements in the map interface are mapped to data elements of the same type and name in the adapter interface.

The Application Designer controls are usually not only bound to business data elements, but also to additional control fields. Which control fields these are depends on the way in which the elements of a map are mapped to Application Designer controls by the Map Converter rules. For instance, a `statusprop` can be assigned to a field, which results in an additional parameter in the parameter data area of the adapter. An array on a map can have been converted to a grid control with server-side scrolling. In this case, the additional data structures needed to control server-side scrolling need to be added to the `DEFINE DATA` statement.

statusprop

The `statusprop` is needed to control the error status or focus of a FIELD (Java or Natural) control dynamically (see **example 3** for the `REINPUT` statement below where it is used to replace the `MARK *field-name` clause). The default conversion rules contain a rule that creates a `statusprop` property for each map field that is controlled by a control variable. The adapter generator creates from this property a corresponding status variable and a comment line that identifies the status variable as belonging to the field.

Example

The parameter data area of the map contains:

```
01 LIB-NAME (A8)
01 LIB-NAME-CV (C)
```

The parameter data area of the adapter will then contain:

```
* statusprop= STATUS_LIB-NAME-CV
01 LIB-NAME (A8)
01 STATUS_LIB-NAME-CV (A) DYNAMIC
```

The variable `STATUS_LIB-NAME-CV` is not yet known to the main program and must be defined there.

INPUT Statement

The replacement for the `INPUT` statement is the `PROCESS PAGE` statement. In its simplest form, the `INPUT` statement just references the map. In this case, it is just replaced by a `PROCESS PAGE` statement with the corresponding adapter.

Example 1

Main program before conversion:

```
INPUT USING MAP 'MMENU'
```

Main program after conversion:

```
IF *BROWSER-IO NE 'RICHGUI'
  INPUT USING MAP 'MMENU'
ELSE
  PROCESS PAGE USING 'AMENU'
END-IF
```

The INPUT statement can come with a message text that is displayed in the status bar. There is no direct replacement for this construction because the PROCESS PAGE statement (in contrast to the PROCESS PAGE UPDATE statement) does not support the SEND EVENT clause.

Example 2

Main program before conversion:

```
INPUT WITH TEXT MSG01 USING MAP 'MMENU'
```

Main program after conversion (no message will be displayed):

```
IF *BROWSER-IO NE 'RICHGUI'
  INPUT WITH TEXT MSG01 USING MAP 'MMENU'
ELSE
  PROCESS PAGE USING 'AMENU'
END-IF
```

REINPUT Statement

The replacement for the REINPUT statement is the PROCESS PAGE UPDATE statement. In its simplest form, the REINPUT statement comes with a message text that is displayed in the status bar. In the converted code, this is handled by the SEND EVENT clause of the PROCESS PAGE UPDATE statement.

Example 1

Main program before conversion:

```
REINPUT [FULL] WITH TEXT MSG01
```

Main program after conversion:

```
IF *BROWSER-IO NE 'RICHGUI'
  REINPUT [FULL] WITH TEXT MSG01
ELSE
  PROCESS PAGE UPDATE [FULL]
    AND SEND EVENT 'nat:page.message'
    WITH PARAMETERS
      NAME 'type' VALUE 'E'
      NAME 'short' VALUE MSG01
```

```
    END-PARAMETERS
END-IF
```

The REINPUT statement can come with a message number and replacements. In this case, the message must be created from number and replacements before it is sent to the status bar with the SEND EVENT clause.

Example 2

This example uses a subprogram GETMSTXT that builds the message text from number and replacements.

Main program before conversion:

```
REINPUT [FULL] WITH TEXT *MSGNR, REPL1, REPL2
```

Main program after conversion:

```
IF *BROWSER-IO NE 'RICHGUI'
  REINPUT [FULL] WITH TEXT *MSGNR, REPL1, REPL2
ELSE
  CALLNAT 'GETMSTXT' MSTEXT MSGNR REPL1 REPL2
  PROCESS PAGE UPDATE [FULL]
    AND SEND EVENT 'nat:page.message'
    WITH PARAMETERS
      NAME 'type' VALUE 'E'
      NAME 'short' VALUE MSTEXT
    END-PARAMETERS
END-IF
```

Example 3

The REINPUT statement can come with a MARK clause in order to put the focus on a field. This case requires that a statusprop property is created for the field during map conversion. The variable bound to the statusprop property is then used before the PROCESS PAGE UPDATE statement to set the FOCUS to the field.

Main program before conversion:

```
REINPUT [FULL] WITH TEXT MSG01 MARK *LIB-NAME
```

Main program after conversion:

```
01 STATUS_LIB-NAME-CV (A) DYNAMIC
...
IF *BROWSER-IO NE 'RICHGUI'
  REINPUT [FULL] WITH TEXT MSG01 MARK *LIB-NAME
ELSE
  STATUS_LIB-NAME-CV := 'FOCUS'
  PROCESS PAGE UPDATE FULL
    AND SEND EVENT 'nat:page.message'
    WITH PARAMETERS
      NAME 'type' VALUE 'W'
      NAME 'short' VALUE MSG01
    END-PARAMETERS
END-IF
```

PF-Key Event Handling

The original application might contain checks for the content of the system variable `*PF-KEY` at arbitrary places in the code. In order to handle function key events correctly in the converted application, several things need to be achieved:

- In response to the function keys, the converted application must raise events that are named like the possible contents of `*PF-KEY`. This can be achieved by using a page template such as *NATPAGEHOTKEYS_TEMPLATE.xml* which contains the required hot key definitions.
- A common local variable must be set up right after the `INPUT` or `PROCESS PAGE` statement that contains either the value `*PF-KEY` or `*PAGE-EVENT`, depending on the execution environment. The name of the variable can be freely chosen. In the example below, the name `XEVENT` is used.
- The events `nat:page.end` and `nat:browser.end` must be handled in such a way so that the program terminates. See also *Built-in Events and User-defined Events*.
- A default event handler must be set up that takes care of the values of `*PAGE-EVENT` that are not expected by the original application code. These unexpected events are simply replied with a `PROCESS PAGE UPDATE FULL` statement.

Example

```
01 XEVENT (U) DYNAMIC
...
PROCESS PAGE USING ...
...
IF *BROWSER-IO = 'RICHGUI'
  DECIDE FOR FIRST CONDITION
    WHEN *PAGE-EVENT = 'nat:page.end'
      STOP
    WHEN *PAGE-EVENT = MASK ('PF'*) OR = MASK ('PA'*)
    OR = 'ENTR' OR = 'CLR'
      XEVENT := *PAGE-EVENT
    WHEN NONE
```

```
      PROCESS PAGE UPDATE FULL
  END-DECIDE
ELSE
  XEVENT := *PF-KEY
END-IF
```

All references to `*PF-KEY` in the code must then be replaced by references to `XEVENT`.

SET KEY Statement

Natural for Ajax provides two controls (NJX:BUTTONITEMLIST and NJX:BUTTONITEMLISTFIX) that represent a row of buttons. These controls can be used to replace the visual representation of the function keys from the original application. If the page template *NATPAGEPFKEYS_TEMPLATE.xml* or a similar individually adapted template is used during map conversion, each resulting page will contain a row of function key buttons. The subject of this section is how the converted application can control the labeling and the program-sensitivity of the function keys with only little code changes.

Natural controls the labeling and program-sensitivity of the function keys in a highly dynamic way. The corresponding application code (`SET KEY` statements) can be distributed across program levels and can be lexically separated from the corresponding `INPUT` statements. Also, the `SET KEY` statement has several flavors, some affecting all keys and others affecting only individual keys. As a result, the status of the function keys at a given point in time can only be determined at application runtime.

Therefore, the following approach is chosen: Natural provides the application programming interface (API) `USR4005` that reads the current function key naming and program-sensitivity at runtime. During code conversion, a call to this API is inserted after each `SET KEY` statement or into each round trip. This call reads the function key status and passes it to the user interface.

Example

Main program before conversion:

```
SET KEY ENTR NAMED 'Enter' PF1 NAMED 'F1' PF2 NAMED 'F2'
PF3 NAMED 'Modify' PF4 NAMED 'Delete' PF5 NAMED 'F5'
PF6 NAMED 'F6' PF7 NAMED 'Create' PF8 NAMED 'Display'
PF9 NAMED 'F9' PF10 NAMED 'F10' PF11 NAMED 'F11' PF12 NAMED 'F12'
*
INPUT USING MAP "KEYS-M"
*
END
```

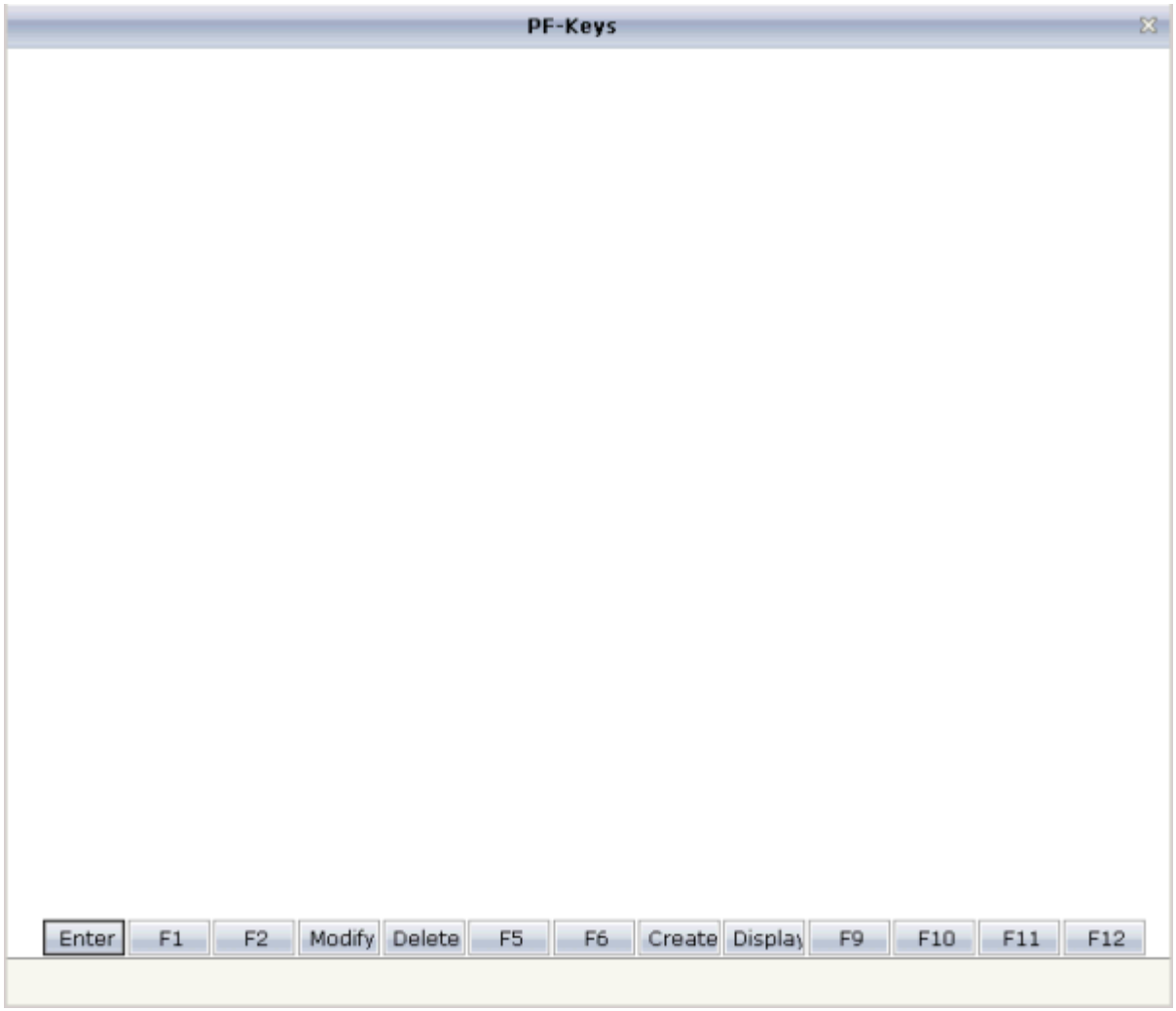
Map before conversion:

```
                                  *** PF-Keys ***









Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
Enter F1    F2    Modif Delet F5    F6    Creat Displ F9    F10   F11   F12
```

Main program after conversion:

```
DEFINE DATA LOCAL
1 PFKEY (1:*)
2 METHOD (A) DYNAMIC
2 NAME (A) DYNAMIC
2 TITLE (A) DYNAMIC
2 VISIBLE (L)
1 METHODS (A4/13) CONST <'ENTR','PF1','PF2','PF3','PF4',
'PF5','PF6','PF7','PF8','PF9','PF10','PF11','PF12'>
END-DEFINE
*
SET KEY ENTR NAMED 'Enter' PF1 NAMED 'F1' PF2 NAMED 'F2'
PF3 NAMED 'Modify' PF4 NAMED 'Delete' PF5 NAMED 'F5'
PF6 NAMED 'F6' PF7 NAMED 'Create' PF8 NAMED 'Display'
PF9 NAMED 'F9' PF10 NAMED 'F10' PF11 NAMED 'F11' PF12 NAMED 'F12'
*
IF *BROWSER-IO NE "RICHGUI"
  INPUT USING MAP "KEYS-M"
ELSE
  EXPAND ARRAY PFKEY TO (1:13)
  METHOD(1:13) := METHODS (*)
  CALLNAT "GETKEY-N" PFKEY (*)
  PROCESS PAGE USING "KEYS-A"
END-IF
*
END
```

Page after conversion:



Explanation

The structure `PFKEY` is generated into the Natural adapter of the page as the application interface to the BUTTONITEMLISTFIX control.

The subprogram `GETKEY-N` is a convenience wrapper for the API subprogram `USR4005`. It uses `USR4005` to determine the labeling and the program-sensitivity status for a given list of function keys. Each function key is identified by the `*PF-KEY` value it raises. `GETKEY-N` returns the function key information in a data structure suitable for the application interface of the BUTTONITEMLISTFIX control. The subprogram is delivered in source code with the Natural for Ajax demos and can be adapted to the needs of the application.

Array Data

To use grid controls like TEXTGRIDSSS2 and ROWTABLEAREA2, you need to bind the `griddata-prop` attribute to an array structure at level 1. For the example Natural data definitions below the `griddataprop` attribute needs to be bound to the `level1` field.

Example 1

```
1 level1 (00001:00005)
2 arrayfield1 (a10)
2 arrayfield2 (a10)
```

Example 2

```
1 level1
2 arrayfield1(a10/00001:00005)
2 arrayfield2(a10/00001:00005)
```

Natural however, also allows to have a combination of single fields and arrays as shown in the following example:

Example 3

```
1 level1
2 field1 (a10)
2 arrayfield1(a10/00001:00005)
2 arrayfield2(a10/00001:00005)
```

To bind a TEXTGRIDSSS3 or ROWTABLEAREA2 to structures as shown in example 3 you basically have two options:

Option 1

Change the original Natural data definition structure, which is usually the preferred and recommended way.

Option 2

Add an extra set of variable definitions to your Natural code like:

```
1 level1x
2 field1 (a10)
1 level1
2 array1(a10/00001:00005)
2 array2(a10/00001:00005)
```

You may need to add extra Natural code to transfer the values to/from the original fields. However, if the Natural source code only references the variables without level 1 qualifiers (for example, using `reset array1(*)` instead of `reset level1.array1(*)`) no source change is required except for the initial data definitions.

If Option 1 is not possible, the ***convrulesCVVariables.xml*** example rules file offers semi-automated support for Option 2. It automatically splits the original structure into two and adds an "x" to the name of the newly created structure for the non-array fields in the adapter and adapter interface as shown in Option 2.

Processing Rules

The Natural maps in the application to be converted may contain processing rules. In the sense of a Natural for Ajax application, the processing rules are server-side validations because they are executed on the Natural server side of the application.

In order to extract processing rules from the maps and to turn them into server-side validations in the converted application, the Natural Engineer function “Separate Processing Rules from Maps” can be used.

There is currently no function available that automatically turns processing rules into client-side validations in Application Designer.

System Variables

If a map displays a system variable (for example, `*DATX`), a specific default conversion rule takes care that the necessary code for handling the system variable is generated into the Natural adapter of the resulting page layout.

Example 1

The map displays the contents of the system variables `*DATX` and `*TIMX`. The contents of these system variables are not modifiable.

The `DEFINE DATA` statement of the adapter will then contain:

```
LOCAL
01 XDATX (A8)
01 XTIMX (A8)
```

The body of the adapter will then contain:

```
XDATX := *DATX
XTIMX := *TIMX
*
PROCESS PAGE ... WITH
PARAMETERS
...
 NAME U'XDATX'
  VALUE XDATX
 NAME U'XTIMX'
  VALUE XTIMX
END-PARAMETERS
```

The main program needs no special adaptation.

Example 2

The map displays the content of the system variable `*CODEPAGE`. The content of this system variables is modifiable.

The `DEFINE DATA` statement of the adapter will then contain:

```
LOCAL
01 XCODEPAGE (A64)
```

The body of the adapter will then contain:

```
XCODEPAGE := *CODEPAGE
*
PROCESS PAGE ... WITH
PARAMETERS
...
 NAME U'XCODEPAGE'
  VALUE XCODEPAGE
...
END-PARAMETERS
*
*CODEPAGE := XCODEPAGE
```

The main program needs no special adaptation.

Variable Names Containing Special Characters

A similar procedure applies to special characters contained in variable names. These are the following special characters:

+
#
/
@
§
&
$

Note: The hash (#) can occur only as the first character.

Variables names containing these special characters cannot be directly bound to Application Designer control attributes. A specific default conversion rule replaces the names containing these special characters with configurable replacements. The original field name is generated into the parameter data area of the Natural adapter and a corresponding mapping is generated into the `PROCESS PAGE` statement of the adapter.

Example

The map displays the variables `#FIRST` and `#LAST`.

The `DEFINE DATA` statement of the adapter will then contain:

```
DEFINE DATA PARAMETER
1 #FIRST (A16)
1 #LAST (A20)
```

The body of the adapter will then contain:

```
...
PROCESS PAGE ... WITH
PARAMETERS
...
NAME U'HFIRST'
  VALUE #FIRST
 NAME U'HLAST'
  VALUE #LAST
...
END-PARAMETERS
```

The main program needs no special adaptation.