

Axon.ivy 6.7

Designer Guide

Axon.ivy 6.7: Designer Guide

Publication date 23.08.2017

Copyright © 2008-2017 AXON IVY AG

1. Introduction	1
Axon.ivy Documentation	1
Axon.ivy Workbench	1
Perspectives	2
Most important menu entries	3
Most important toolbar items	6
Useful Commands (Shortcuts)	8
Axon.ivy Preferences (Workspace Preferences)	8
Common UI Components	16
2. Process Modeling	22
Projects	22
Process Modeling	40
Simulating process models	59
Process Elements Reference	72
Case Maps	160
3. Data Modeling	167
Data Classes	167
Business Data Store	171
Persistence	175
4. IvyScript	189
Introduction	189
IvyScript Language	189
IvyScript Editor	191
Browsers	193
Public API	196
IvyScript Reference	196
IvyScript-Java Integration	201
5. CMS	204
Content Management System	204
CMS Access	205
CMS Manipulation	206
CMS Translation	215
6. User Interface	217
User Dialogs	217
Web Page	317
7. 3rd Party Integration	336
Introduction	336
Java Interface	336
Database	336
Web Services	336
REST Services	336
8. Configuration	338
Configuration Management	338
Database Configuration	353
REST Clients Configuration	355
Web Service Configuration	358
Roles and Users	362
Configuration files	366
9. Concepts	368
Adaptive Case Management	368
Workflow	375
Offline Tasks	381
Data Storage	383
Overrides	385
Error Handling	392
Rule Engine	397
Extensions	399
Deployment	407

Continuous Integration	412
Miscellaneous	413
10. Troubleshooting	423
Introduction	423
Error Dialogs	423
Startup Problems	423
Memory Problems	424
Graphics Problems	424
OS X Problems	424
Logging	425
11. References	427
Conventions used in this book	427
Reference	428
Glossary	433

Chapter 1. Introduction

Axon.ivy Documentation

Now you are reading Axon.ivy Designer documentation. In case you are want to know more on

- Getting the latest Axon.ivy Version: Go to <http://developer.axonivy.com/download/>
- System requirement: Please read Readme.html in the installation directory
- Working with Axon.ivy Designer: Start with the Quick Start Tutorial (see next section)
- Demo projects: The Axon.ivy Designer ships with several demo projects, which can be imported.
- Engine administration: Please read the Engine Guide (in the installation folder of an Axon.ivy Engine)
- Upgrading an existing installation: Please read Migration Notes (located in installation folder).

All of above mentioned documentation is going to be brief and tend to describe only necessary functionality. We highly encourage reading these documentations to speed up your development, to get to know new features or to eliminate potential problems.



Tip

Do not forget, anytime you find some unknown features hitting F1 will show you a context sensitive compact help!

Axon.ivy Workbench

Axon.ivy is based on the Eclipse platform. So when you start Axon.ivy Designer you launch an Eclipse workbench.

The first thing you see after starting Axon.ivy Designer is a dialog that allows you to select the location of the **workspace**. The workspace is the root directory where your work will be stored.

After the workspace location is chosen, the Workbench window is displayed. Initially, at the first start the Welcome Screen is displayed. On this screen different links to tutorials and documents are displayed. New users should click on the *Quick Start Tutorial* to learn how to use Axon.ivy Designer.

You can get the Welcome Screen back at any time by selecting *Help > Welcome*.

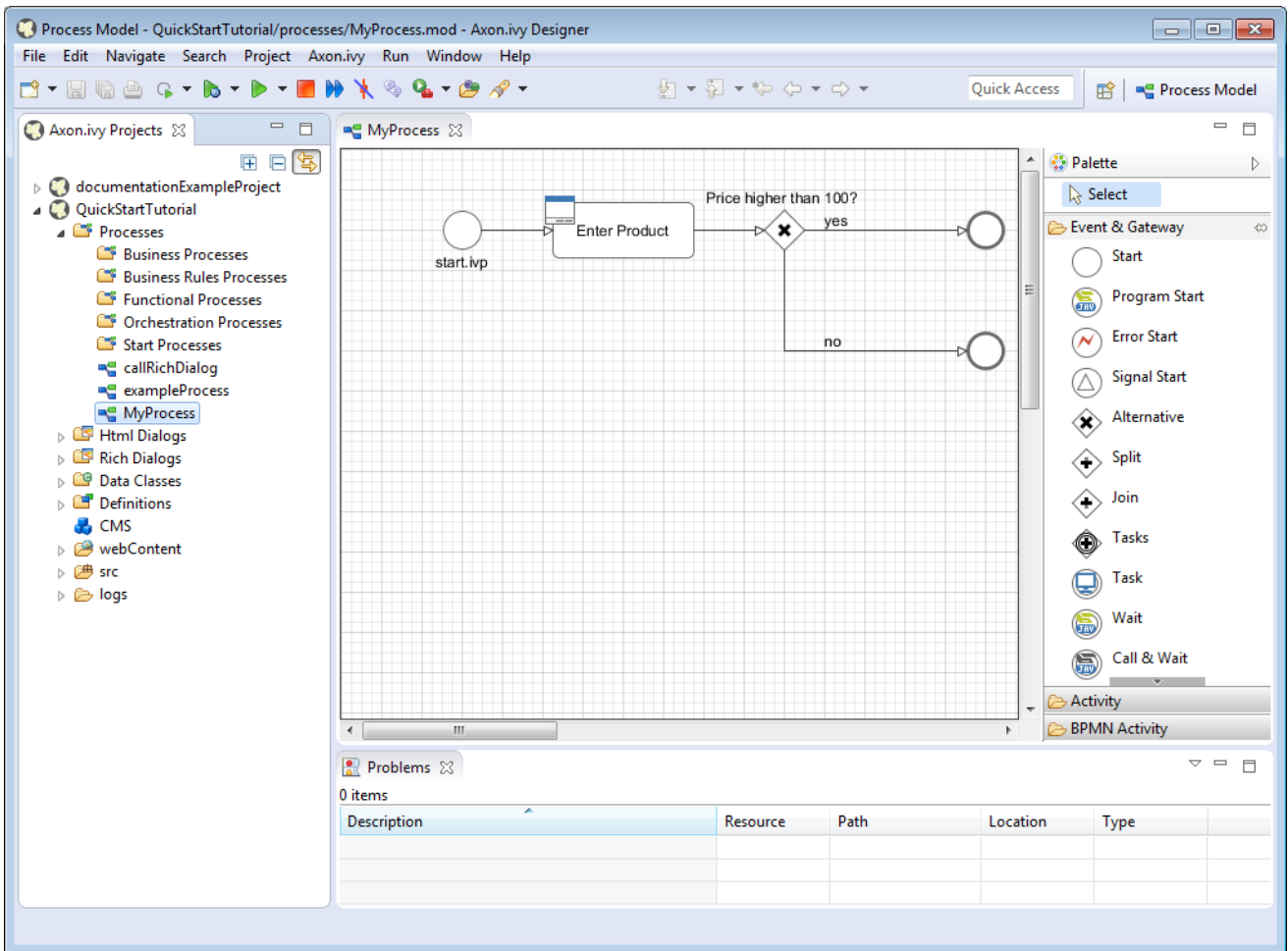


Figure 1.1. Axon.ivy Workbench Overview

Axon.ivy Editors and Views

Editors and views are visual components in the workbench. They are the tools to work with.

A **view** is typically used to display properties for an active editor or to navigate through a hierarchy of information. Modifications made in a view are saved immediately.

An **editor** is used to edit a certain type of a resource like a process model diagram or a dialog panel. Modifications made in an editor follow the open-save-close life cycle model. Multiple instances of an editor can be open within a workbench window.

The name of the resource that is shown in the editor appears in the editor's tab label. If an asterisk (*) appears at the left side of the label this indicates that the editor has unsaved changes. When an attempt is made to close the editor or exit the Workbench with unsaved changes a prompt to save the editor's changes will appear.

Double clicking the tab of an editor or view will expand the part to full size of the workbench window. Double click the tab of an expanded part again to toggle back to default size and location.

See the section Axon.ivy Views for a summary of the ivy specific views.

In the section Axon.ivy Editors you find a summary of the ivy editors.

Perspectives

A perspective defines a set and layout of views and editors in the workbench window. The current perspective is displayed on the title bar of the window and it is highlighted at the upper right corner in the shortcut bar. You can open and switch to a specific perspective using the Window menu or the buttons in the shortcut bar at the upper right corner.

You can change the perspectives as you like. Editors and views can be rearranged in a perspective by just moving them around. It's also possible to close views or display additional ones (via Window -> Show View). Furthermore there is a functionality to reset a perspective to its default definition (via Window -> Reset Perspective).



Figure 1.2. Switch between Perspectives

Perspectives provided by Axon.ivy

Axon.ivy provides following perspectives:

- *Process Model*: Create and edit business process diagrams.
- *Process Development*: Simulate and debug processes or design Html Dialogs .
- *Rich Dialog*: Design Rich Dialogs.

Other Perspectives

There are further perspectives provided by the Eclipse platform that you may use for specific development tasks:

- *Resources*: Explore and manipulate resources of the project at file system level.
- *Team Synchronization*: Browse through the changes between your local working copy and the base revision on the Subversion server.
- *Java*: Edit Java source files.



Warning

Work carefully with these perspectives, since you might get direct access to Axon.ivy resource files that lets corrupt their content.

Most important menu entries

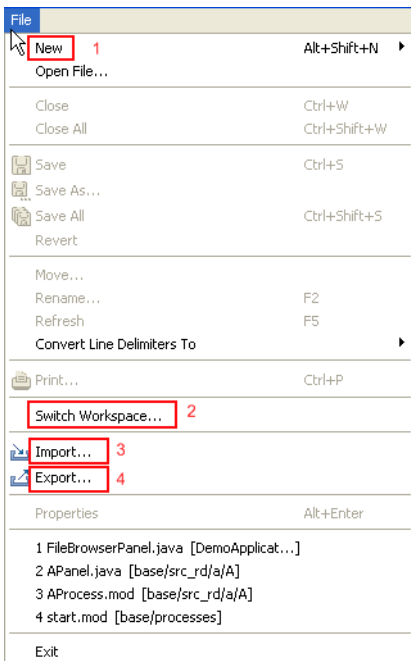
The main window of Axon.ivy contains multiple menus. This section explains the most important menus and menu items for Axon.ivy users.



Note

The availability and enablement of many menu items is dependent on the current selection and the currently active editor. They may therefore slightly vary and not exactly correspond to the screenshots below.

File menu



1 - New ...

This menu item opens a sub menu with all available *new wizards* to create new resources for editing. Some of the shown wizards are contributed by the *Eclipse* system and are not Axon.ivy specific.

To create new Axon.ivy resources it is recommended to use the menu entry *Axon.ivy > New...* or to use the context menu in the *Axon.ivy project tree...*

2 - Switch Workspace...

Allows you to switch to a different workspace. A workspace is a directory that contains a collection of projects that are related to each other in some way.

In the opening dialog simply select a different workspace directory. If you've switched workspaces before, you can also chose from a list of previously used workspaces.

3 - Import...

Use this menu to import *General > Existing Projects into Workspace* or to import *Other > Checkout Projects from SVN*.

Existing projects (that can be located anywhere on your machine, also in a different workspace) can be chosen to be imported by reference only or by copying the whole project.



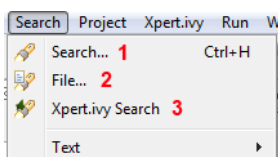
Warning

There are currently some unsolved problems importing multiple projects from SVN directly. Please import projects one by one, or check out projects from a file explorer first, and then import them into your workspace.

4 - Export...

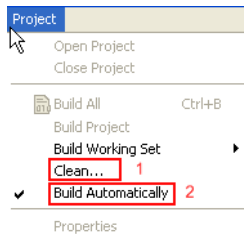
This menu can be used to export projects or parts of projects from the current workspace. Use *General > Archive File* or *General > File System*.

Search menu



- | | |
|---------------------|--|
| 1 - Search... | Opens the Search dialog. Where search tabs allows you to search for instance Axon.ivy artifacts or File contents. |
| 2 - File... | Same as <i>Search...</i> but opens directly on the <i>File</i> tab of the Search dialog, where search queries for any text within any resource of the current project or even in the whole workspace can be started. |
| 3 - Axon.ivy Search | Same as <i>Search...</i> but opens directly on the Axon.ivy Search tab of the Search dialog, where a search for Processes, CMS Objects, Data Classes or Rich Dialogs can be started. |

Project menu



- | | |
|-------------------------|---|
| 1 - Clean... | Deletes all temporary and generated files (such as compiled Java class files, compiled Data class files, etc.) from your project or workspace and rebuilds them.

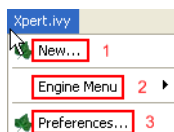
This may help to resolve building problems. |
| 2 - Build Automatically | Enables / disables automatic building (i.e. compiling) of project resources such as Java classes (also for Rich Dialogs) or Data classes. This means that all the necessary resources are automatically built and updated if changes are made in a project. |



Warning

You should never turn this option off! It may lead to seemingly erroneous behavior of Axon.ivy (chances are that most things don't work anymore as expected until the option is turned on again or until you build the workspace / projects manually again).

Axon.ivy menu

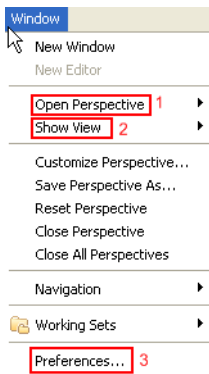


- | | |
|-----------------|---|
| 1 - New ... | This menu item opens a sub menu with all available <i>new wizards</i> to create new Axon.ivy specific resources (such as User Dialogs or Data classes or Axon.ivy projects). |
| 2 - Engine Menu | Offers the operations <i>Start Engine And Show Start Page</i> , <i>Start Engine</i> , <i>Stop Engine</i> , <i>Engine Speed</i> and <i>Enable/Disable Animation</i> .

See also Toolbar section. |
| 3 - Preferences | Opens the <i>Preferences</i> editor for the Axon.ivy specific preferences only.

See also Preferences section. |

Window menu



1 - Open Perspective

Opens a sub menu with the available perspectives. Non-standard perspectives can be selected from the sub menu *Other...*

2 - Show View

Opens a sub menu with the available views for the current perspective. Non-standard views can be selected from the sub menu *Other...*. The complete set of Axon.ivy specific views is available below the *Axon.ivy* folder.

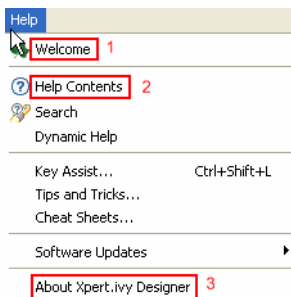
3 - Preferences...

Opens the *Preferences* editor for the all settings that are available on the *Eclipse* platform. The Axon.ivy specific preferences are available under the *Axon.ivy* branch of the preferences tree.

For convenience use the menu *Axon.ivy > Preferences...* to open the preference editor for the Axon.ivy specific settings only.

See also Preferences section.

Help menu



1 - Welcome

Opens the welcome screen as shown on the first start of Axon.ivy after installation.

2 - Help Contents

Opens the Help browser. Select *Axon.ivy Designer Guide* to access this documentation here. Many other help documents that may be relevant for Axon.ivy users are also available (e.g. *Workbench User Guide* for an introduction to the Eclipse workbench or *ULC Visual Editor Help* for a general introduction to the Visual Editor).

3 - About Axon.ivy Designer

Shows system information about the used Axon.ivy Designer application (e.g. version and build number). This might be helpful when seeking support.

Most important toolbar items

The main window of Axon.ivy shows a toolbar with various buttons which is situated right below the menu bar of the application. This section explains the most important actions for Axon.ivy users that can be found on the toolbar.



Note

The number and activity state of the buttons that are shown in the toolbar depend on the current selection and the currently active editor. They may therefore slightly vary and not exactly correspond to the screenshot below.



 - New Wizard Selector

This button opens a drop down list with all available *new wizards* to create new resources for editing. Some of the shown wizards are contributed by the *Eclipse* system and are not Axon.ivy specific.

To create new Axon.ivy resources it is recommended to use the menu entry *Axon.ivy > New...* or to use the context menu in the *Axon.ivy project tree...*

 - Save

Saves the contents of the currently active editor.

 - Pause / Restart Visual Editor


Only available if a Rich Dialog Visual Editor is active. Pauses / restarts the Visual Editor. This may help to restore a consistent state if for some reason, the visual editor has lost it's internal model (i.e. visual parts are not rendered correctly).

Pause / restart is usually faster than closing and reopening of the editor. When the Visual Editor is paused the contents of a panel can no longer be edited until it is restarted again.



Tip

In case you think your Visual Editor somehow does not work as expected, close all Visual Editor instances, and clear its cache using *Axon.ivy > Debug > Clear Visual Editor Cache*

 - Start Process Engine And Show Start Page


Starts the process engine, switches to the *Process Development Perspective* and displays the start page.

 - Start Process Engine

Starts the process engine (start page is shown only if browser view is already opened).

 - Stop Process Engine

Stops the process engine and terminates all currently running processes (including User Dialogs).

 - Set Animation Speed


Configure the animation speed for process simulation. Other animation settings you find in Engine/Simulation Settings

 - Enable / Disable Animation


Globally enables / disables animation during process simulation. Other animation settings you find in Engine/Simulation Settings

 - Insert Rich Dialog


Only available if a Rich Dialog Visual Editor is active. Opens a dialog for Rich Dialog selection and inserts the selected Rich Dialog as an embedded widget if mouse is clicked on the panel of the currently active Visual Editor.

 - Select Content and Formatting Language



See Content and Formatting Language Dialog for more details.

 - Search

Opens the Search dialog. Allows you to search for any text within any resource of the current project or even in the whole workspace (use the *File Search* tab of the dialog).

 - Goto Previous Modification

Goes back to the editor and location where the last modification was made (if available).

 - Goto Next Modification	Goes to the editor and location where the next modification was made (if available).
 - Open Perspective	Opens a drop-down list with perspectives to select for opening. Non-standard perspectives can be chosen from the <i>Other...</i> menu entry.

Useful Commands (Shortcuts)

This section explains important global available commands and their shortcuts.

Open Ivy Process Ctrl+Shift+P	Opens a dialog which allows you to browse for a Process to open in an editor.
Open Ivy User Dialog Ctrl+Shift+D	Opens a dialog which allows you to browse for a User Dialog to open in an editor.
Open Type Ctrl+Shift+T	Opens a dialog which allows you to browse for a java type to open in an editor.
Open Resource Ctrl+Shift+R	Opens a dialog which allows you to browse for a resource file to open in an editor.
Navigate Back Alt+Left	Navigates to the previous resource that was viewed in an editor. Analogous to the back button on a web browser.
Navigate Forward Alt+Right	Navigates to undo the effect of the previous back command. Analogous to the forward button on a web browser.

We recommend that you learn and use shortcuts as much as possible because you work more efficiently when you switch between keyboard and mouse as less as possible. You can find more useful shortcuts here: <https://shortcutworld.com/en/Eclipse/win/all>.

Axon.ivy Preferences (Workspace Preferences)

In the preferences, you can configure some settings of Axon.ivy Designer to adapt to your personal working style. Open the Preferences.



Note

Most of these settings (and some more settings) you can overwrite in the properties of project

Drag and Drop Settings

Drag and Drop (DnD) is one way to define data binding between instances of Ivy Data Classes and widgets members in Rich Dialogs and to define event mappings between ULC widget events and RD Logic Event Steps.

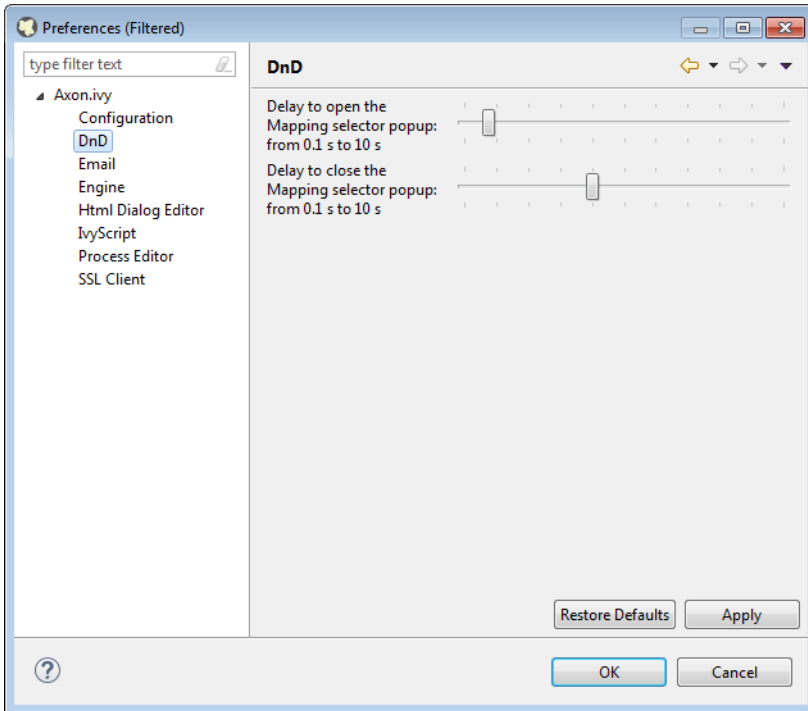


Figure 1.3. The Drag and Drop preferences

Open the Drag and Drop preference page

Open Popup Delay Specify how many seconds that the DnD drop dialog should wait until it shows.

Close Popup Delay Specify how many seconds that the DnD drop dialog should stay open until it closes.

Process Editor Settings

The process editor settings are used to configure the behavior and look of the process editor as well as some settings that are related to the use of processes.

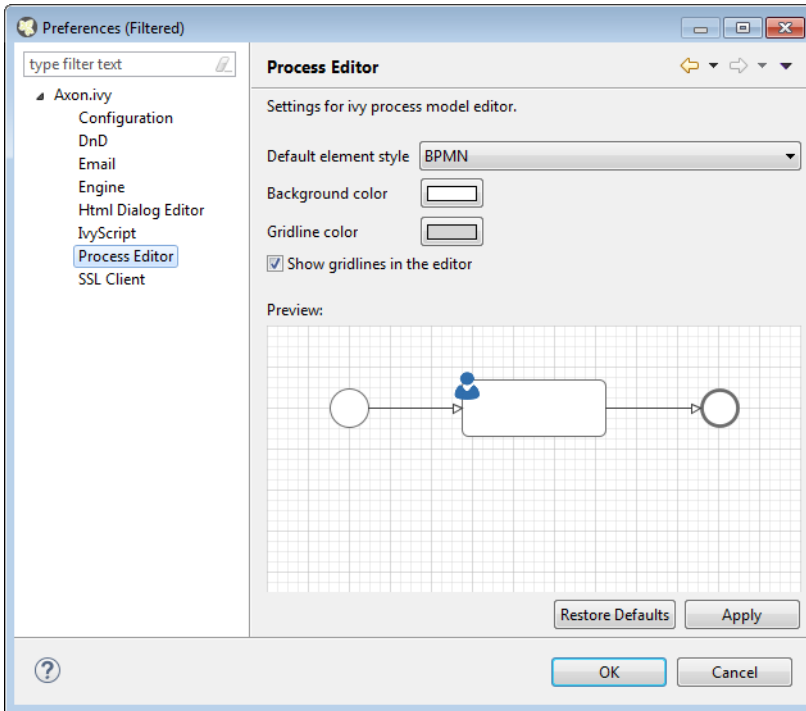


Figure 1.4. The Process Editor preferences

Default element style	The style for newly added process elements.
Background color	Specify the background color of the process editor area.
Gridline color	Specify the color of the grid lines in the process editor.
Show grid lines in the editor	Specify whether the grid lines in the process editor are shown.



Note

The process editor settings are also available as project-specific settings.

Email Settings

These settings define the configuration for the Email Process Element on the designer.



Note

On the Engine you need to specifically set the `EMail.Server.*` system properties in the Engine Administration Tool. Therefore you can choose different configurations for the designer and testing purposes than in your production environment

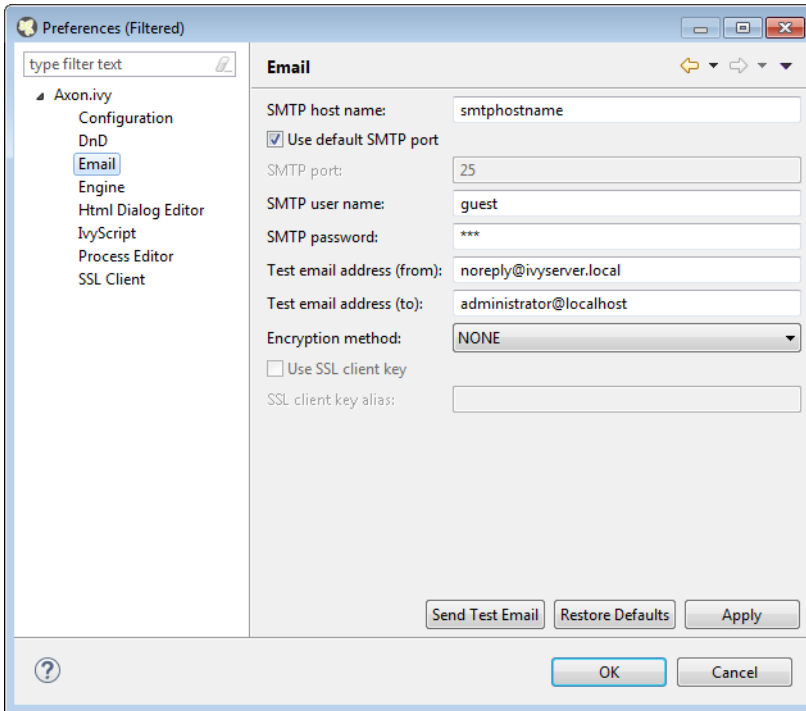


Figure 1.5. The Email preferences

SMTP host name	The name of your outgoing email server (the SMTP server). Please refer to your system administrator to get this name.
Use default SMTP port	If selected the default SMTP port depending on the encryption method is used.
SMTP port	The port your SMTP server is listening. Please refer to your system administrator to get the correct port.
SMTP user name	A valid user name for your outgoing email server to authenticate on it (if this feature is used). Please refer to your system administrator to get this name.
SMTP password	The password for the given user name above. Please refer to your system administrator to get this name.
Test email address (from)	In the designer, this email address overwrites the sender setting in the inscription mask of the Email Step. Mails are not sent to the address that is configured in the inscription mask but to this address. Therefore you can test the functioning of your processes without sending email's to the real addresses (which perhaps are only intended for production messages).
Test email address (to)	The same principle as above but for the recipient.
Encryption method	The encryption method used for the communication with your mail server.
Use SSL client key	Only select this option if the SMTP server requires a client certificate to authenticate the client. The key (certificate) will be read from a key store. See section SSL Client Settings for more information.
SSL client key alias	The name (alias) of the key to send to the SMTP server. If empty the first key found in the key store is used.
Send Test Email	Sends a test mail with the current settings. With the default preferences you should instantly see the sent mail in the 'Email Messages' view which is provided by the developer SMTP.



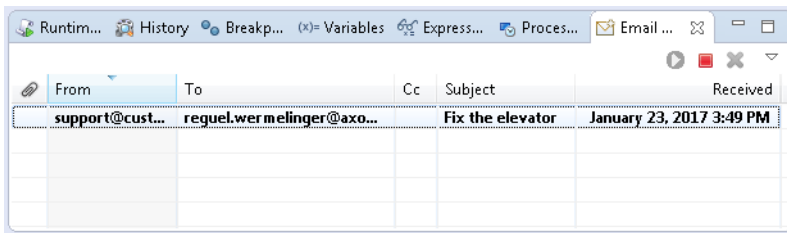
Note

All email settings are also available as project-specific settings.

Developer SMTP

During process development mails generated by the Axon.ivy BPM Suite are preferably not sent to a real SMTP server. Normally you just need to quickly inspect the contents of these generated mails. A real remote SMTP will increase the round-trip to read these message. Therefore the Designer comes pre-configured with a simple SMTP mail server.

With the default E-Mail preferences every mail will be sent to the development SMTP. You can inspect the mail inbox by opening the view 'Email Messages'.



Engine/Simulation Settings

Here you can set whether the internal Browser view of Eclipse or an external Browser is used to show the Process Start Overview and you can configure all the settings related to the animation.

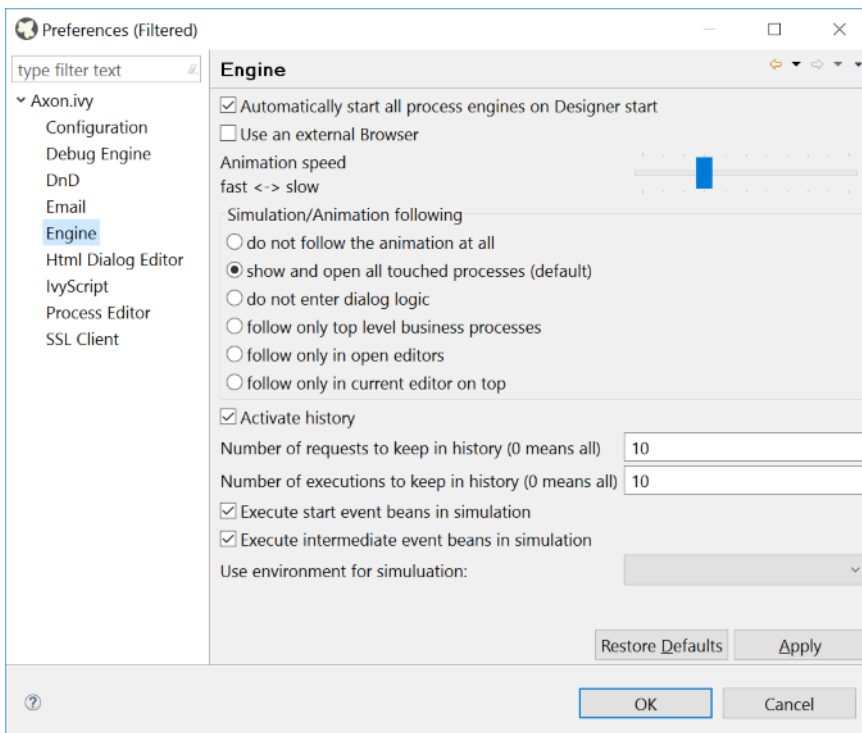


Figure 1.6. The Process Engine preferences

Automatically start all process engines on Designer start

If the check box is selected, all process engines are automatically started on Designer start. It can be disabled to prevent performance issues on large workspaces.

Use an external browser

If the check box is selected, the system Browser is used to display the Process Start Overview otherwise the internal Browser view of Eclipse will be used.

Animation speed fast < - > slow

Sets the default speed of the animation. Setting the slider to a low value lets you observe the process flow easily as the animation speed is decreased.



Tip

As the animation is very slow with low slider values adjust this setting only when you need to debug a process in its lowest details and increase the speed as soon you have finished.

Simulation/Animation follow

Here you can set in which mode the execution is animated. You can choose between the following values:

- Do not follow the animation at all - does nothing
- Show and open all touched processes (default) - Default setting, this opens a process editor window for every process (or User Dialog logic in case of inner User Dialogs) that is used within the started process
- Follow only top level business processes - Simulates and opens only top level business processes. Does not enter User Dialogs, embedded subs or callable subs.
- Do not enter dialog logic - Does not simulate User Dialogs
- Follow only in open editors - You can choose which process are animated by opening them in a process editor window. Note that the focus switches always the window displaying the currently executed process
- Follow only in current editor on top - If you are only interested to debug one specific process. Note, that this is not imperatively the top-level process

History

Here you can configure how many process data snapshots are histories in the process engine history (See also History View).

- Activate history - If ticked process data is histories, if not ticked no process data is histories.
- Number of requests to keep in history (0 means all) - Here you can configure the number of requests per process element for which snapshots of the process data are stored in the history. If you configure 0 the process data snapshots for all requests are stored in the history.
- Number of executions to keep in history (0 means all) - Here you can configure the number of executions per requests and process element for which snapshots of the process data are stored in the history. If you configure 0 the process data snapshots for all executions are stored in the history. A value of 10 means that the process data snapshots of the five oldest and youngest executions of a process element per request are stored in the history.



Note

In case of memory shortage during simulation the settings of history preferences may be ignored (resulting in less snapshots shown in the history).

Event Bean Simulation

Switch off the simulation of Process Start or Intermediate Event Beans when you do to focus on simulations of other elements (Event Bean simulation may pop up process editors with the corresponding process and may overflow the Runtime Log View. In order to apply changes, the Engine must be restarted.

- Execute Start Event Beans in Simulation - If ticked, the Process Start Event Beans are executed, otherwise not.
- Execute Intermediate Event Beans in Simulation - If ticked, the Intermediate Start Event Beans are executed, otherwise not.



Note

All engine settings are also available as project-specific settings.

ivyScript

With this preferences you can choose the default inscription on the action table and the visibility level of the completer.

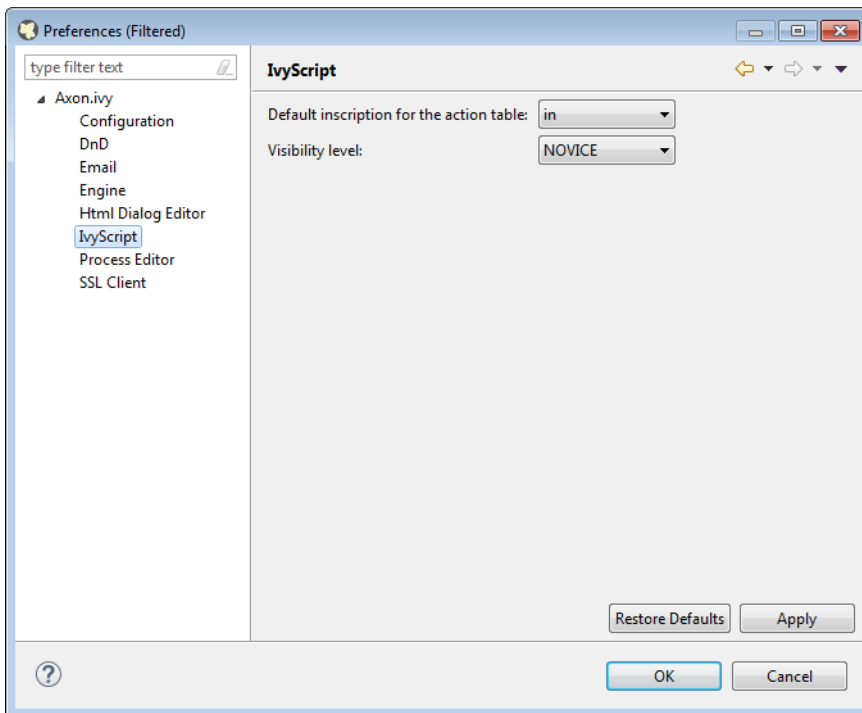


Figure 1.7. The IvyScript preferences

Default Inscription on action table	Before the action table (or any other output code) are executed on an element, the input process data is copied from the out object is assigned from the out object. Use this combo box to specify the copy behavior. The default behavior is to copy by reference (i.e. the out variable will point to the same object as the in variable).
Default IvyScript visibility level	The default visibility level of the <i>IvyScript completer</i> and the <i>function browser</i> can be configured here.

SSL Client Settings

These settings define the key and trust store to use for the client side of SSL connections.



Note

On the Axon.ivy Engine you need to specifically set the `SSL.Client.*` system properties in the Engine Administration Tool. Therefore you can choose different configurations for the designer and testing purposes than in your production environment.

A key store is used to read client keys (certificates). This is only required if a server requests a client certificate so that the server can authenticate the client.

A trust store is used to read server certificates or certificates of certification authorities. The client can validate a server by using the certificates in a trust store. If the server provides a certificate that is signed by a certification authority known by Java then the system trust store can be used. If the server uses a certificate that is self signed or signed by a unknown certification authority then a custom trust store can be used. The custom trust store must contain the server certificate or the certificate of the unknown certification authority.

Key and trust stores can be created and modified (import of certificates, keys) by the **keytool** delivered with the Java Development Kit (JDK). More information can be found in the documentation of the JDK.

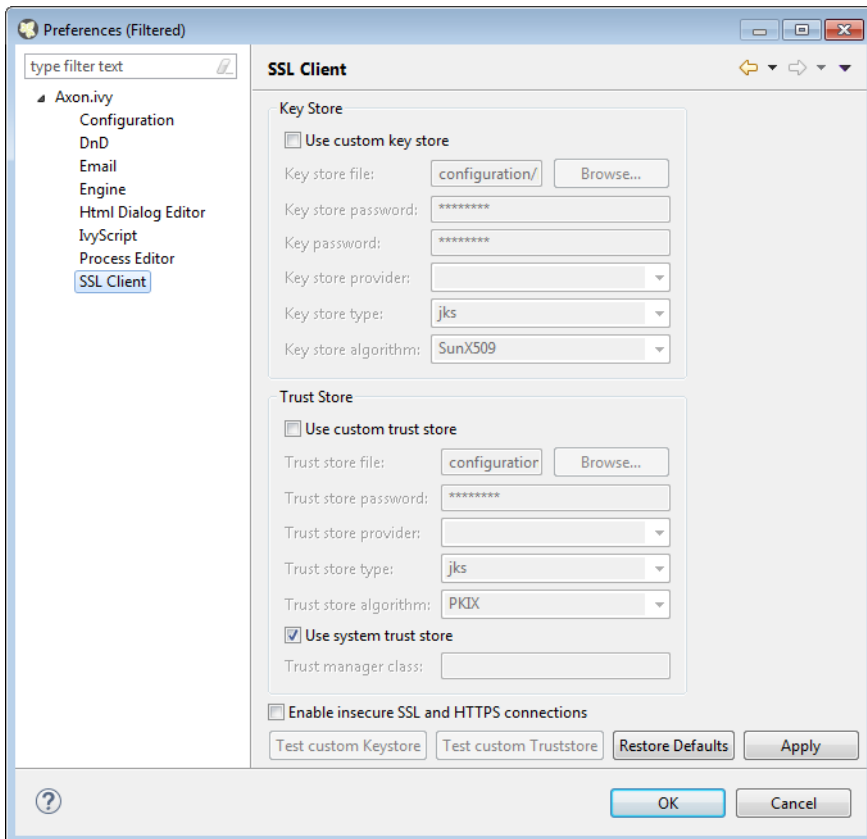


Figure 1.8. The SSL Client preferences

Use custom key store	If selected the key store configured below is used to read the client's key. A client key is only necessary if the server requests a SSL client authentication. If not selected the system keystore is used. The system keystore can be configured by setting the Java system property <code>javax.net.ssl.keyStore</code> .
Key store file	The file that contains the client keys.
Key store password	Password needed to read the key store file.
Key password	Password needed to decrypt the key. If empty the key store password is used instead.
Key store type	The type of the key store. If empty the system default type is used.
Key store provider	The provider used to read the key store. If empty the system default provider is used.
Key store algorithm	The algorithm used to read the key store. If empty the system default algorithm is used.

Use custom trust store	If selected the trust store configured below is used to read trusted server certificates and/or certificates of certification authorities. It is possible to use both the system and custom trust store.
Trust store file	The file that contains the trusted server certificates and/or certificates of certification authorities.
Trust store password	Password needed to read the trust store file.
Trust store type	The type of the trust store. If empty the system default type is used.
Trust store provider	The provider used to read the trust store. If empty the system default provider is used.
Trust store algorithm	The algorithm used to read the trust store. If empty the system default algorithm is used.
Use system trust store	If selected the system default trust store is used. It is possible to use both the system and custom trust store. The system trust store can be configured by setting the Java system property <code>javax.net.ssl.trustStore</code> . If this property is not set then the file <code>jre/lib/security/jssecacerts</code> is used as trust store. If this file is also not available the file <code>jre/lib/security/cacerts</code> is used.
Trust manager class	The full qualified class name of a trust manager class that is used to validate server certificates. This setting is only considered if no custom and system trust store is used.
Test custom Keystore/Truststore	Tests if the specified Keystore/Truststore can be opened and read with the given configuration.
Enable insecure SSL and HTTPS connections	Manipulates the JVMs default SSLSocketFactory, so that untrusted (self signed or outdated) certificates are silently accepted. This could for instance be useful to generate a Webservice stub from an insecure WSDL location.



Note

The SSL Client Trust- and Keystore settings are at the moment only considered for sending mails.

Common UI Components

Axon.ivy uses some UI components that are widely used in the UI panels of the product. This chapter introduces these components.

IvyScript Editor

IvyScript Editors are enhanced text fields or text areas where IvyScript code can be entered. For more information about IvyScript please read the chapter IvyScript. IvyScript Editors have a yellow background to help you identify them visually. On the right side some buttons are visible. These buttons are called Smart Buttons. If you click on them different Dialogs appear which provide context information that you may want to insert into the editor.

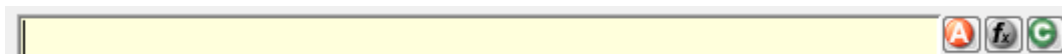


Figure 1.9. IvyScript Editor

An ivyScript Editor will validate the code you enter into it. If the entered code is not valid the editor will change it's background color to red. The invalid part of the entered text will be underlined with a red line. The tool tip of the editor will show a message with a description of the problem.

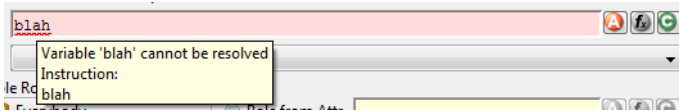


Figure 1.10. IvyScript Editor with a validation problem

An IvyScript Editor has a completer that shows context relevant information in a popup window while you are typing into the editor. You can select any of the proposals to be inserted into the editor. If you don't want to select a proposal simply continue typing. The provided information will be filtered by your input. The popup window will disappear if none proposal exists for the current input. You can enforce the appearance of the popup window by pressing CTRL+SPACE. Pressing CTRL+SPACE on a open completer popup window will cycle the visibility level. The visibility level controls how much information is displayed in the completer popup window. Pressing ALT+h will switch the help text of the selected information on and off. Pressing ESC will close the completer popup window.

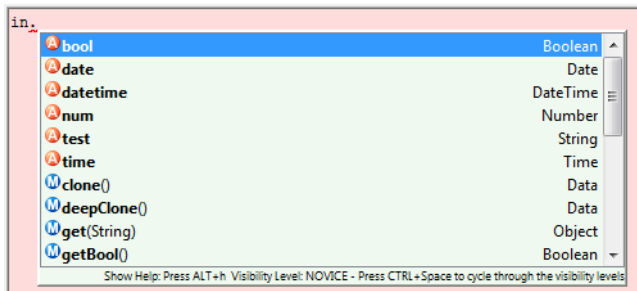


Figure 1.11. IvyScript Editor with completer Popup Window

Macro Text Editor

Like IvyScript Editors, Macro Text Editors are enhanced text fields or text areas with Smart Buttons. Macro Text Editors have a blue background to identify them visually. They are used to specify texts. Inside the texts IvyScript Macros can be used. IvyScript Macros start with `<%=` and end with `%>`. Between these two tags any IvyScript code can be written. IvyScript Macros are place holders which are replaced with the evaluated value of the ivyScript Macro before the text is used.

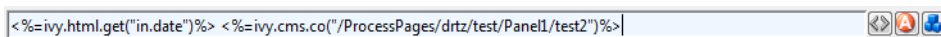




Figure 1.12. Macro Text Editor

Smart Buttons

Smart Buttons are small buttons associated with other UI components. A Smart Button opens a dialog that shows context sensitive information which can be inserted into or used to configure the associated UI component. The following table shows the various Smart Buttons and explains their function.

Smart Button	Description
	Opens an Attribute or and Attribute and Method Browser that shows the current process data structure and the methods that are available on the process data entries. Use this Smart Button if you want to insert process data into the associated UI component. More information about the Attribute and Method Browser can be found in the chapter Attribute and Method Browser
	Opens a Function Browser that shows the structure of the currently available ivy environment variable and the methods that they provide. Moreover it shows all available global functions. Use this Smart Button if you want to insert the ivy environment variable or a global function. More information













Smart Button	Description
	about the Function Browser can be found in chapter Function Browser.
	Opens a Data Type Browser that shows all available process data classes and Java classes. Use this Smart Button if you want to insert a process data or Java class into the associated UI component. Import statements for the selected class will be created on the fly. More information about the Data Type Browser can be found in the chapter Data Type Browser
	Opens the New Bean Class Wizard. Use this Smart Button if you want to create and configure a new Java bean class.
	Opens a Java editor with the class configured in the associated UI component. Use this Smart Button if you want to edit the configured Java class.
	Opens a Content Browser that shows all available content objects. Use this Smart Button if you want to insert the content or a reference to a content object into the associated UI component. More information about the Content Browser can be found in chapter Content Editor
	Opens a Database Field Browser that shows all available database fields. Use this Smart Button if you want to insert a database field into the associated UI component. More information about the databases can be found in chapter Db Step.
	Opens an Operator Browser that shows all available operators. Use this Smart Button if you want to insert an operator (e.g. a SQL operator) into the associated UI components. More information about operators can be found in chapter Db Step.
	Opens an Web Service Configuration Browser that shows all available Web Service configurations. Use this Smart Button if you want to insert a reference to a web service configuration into the associated UI components. More information about Web Service configurations can be found in chapter Web Service Call Step.
	Opens a HTML Tag/Attribute Browser that shows available html tags and attributes. Use this Smart Button if you want to configure html tags or attributes of the associated UI component.
	Opens a Link Browser that shows available link types. Use this Smart Button if you want to insert HTML references (e.g. <code></code>) or URIs to certain artifacts. More information can be found in chapter Link Browser
	Opens a Color Browser that shows available colors. Use this Smart Button if you want to insert a color definition into the associated UI component.
	Opens a Font Browser that shows available fonts. Use this Smart Button if you want to insert a font definition into the associated UI component.
	Cancels the current editing operation and resets the value in the associated UI component to the value it has before the editing was started.

Table 1.1. Smart Buttons

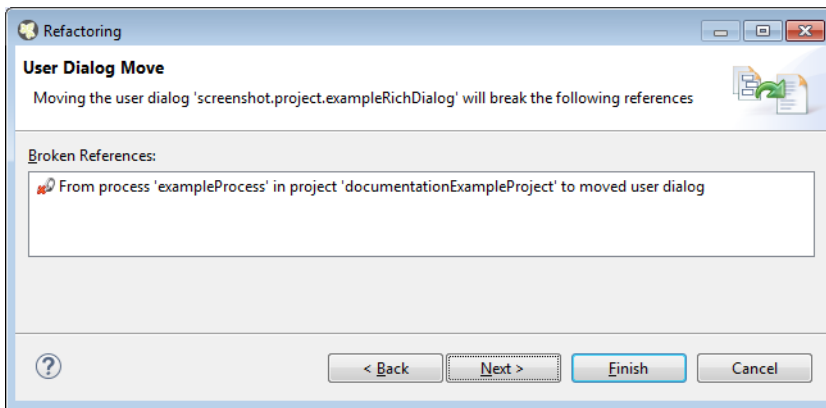
Refactoring Wizards

Refactoring wizards appear when you *rename*, *move*, *delete* or *copy/paste* Axon.ivy artifacts.

The wizards allow you to specify the new name/namespace/location of the artifacts that should be refactored and will give you an overview of the result of the operation before it is actually executed, so that you can estimate the consequences.

All refactoring wizards have the same structure:

- On the *first page* you enter the **parameters** of the operation (e.g. new name and/or namespace, target project, etc.) if any are required. Also you may choose whether any existing references to the refactored artifact should be updated automatically (e.g. if you rename a sub process then all callers to that sub process will be updated, so that they point to the renamed instance). This is the default behavior.
- On the *second page* you will be presented with a **list of references that will be broken** after the operation is executed. This page is not displayed, if no broken references are detected. This page is only displayed for *delete* or *move* refactorings, i.e. if the artifact will no longer exist in the scope of any callers that referred to it before the operation.

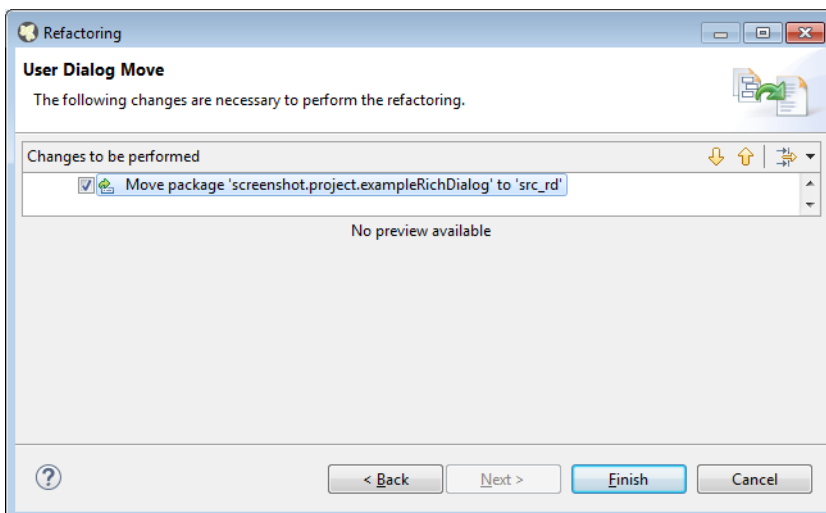


- On the *third page*, you will eventually be presented with a **detailed list of operations** that will be performed as a result of the selected refactoring and possibly a preview of any old and new resources that will be created. You may individually de-select any operations, they will then not be executed when you press *Finish*.



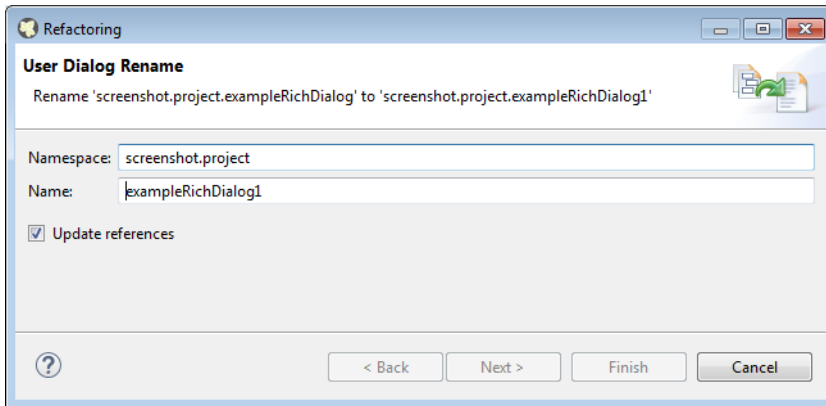
Warning

It is recommended that you don't uncheck any of the scheduled operations (unless you know exactly what you're doing), since this may leave the workspace in an inconsistent state.



You can get from one page to another by pressing the *Next* and *Back* buttons, however there is no requirement to have a look at all three pages. As soon as the *Finish* button is enabled (this may not be the case, if some required input is missing on the first page), you may press it and execute the operation immediately.

Rename Wizard



Change the name and/or namespace. If you enter invalid values then an error will be displayed.

If you leave the *Update references* box checked, then all existing references to the renamed artifact (within the current workspace) will be updated automatically. Otherwise, no callers or references will be updated, which will possibly result in broken references.

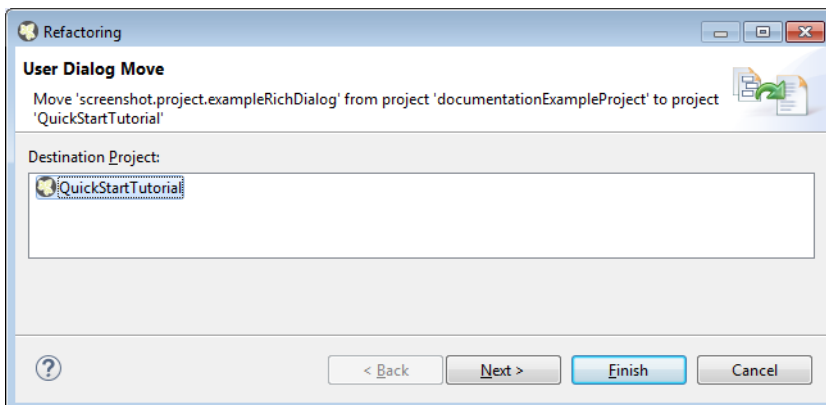


Note

Please note, that the namespace for processes is written with forward slashes '/' as separators (e.g. 'MyProcesses/ Customer/Invoice') while the namespace for Data Classes and User Dialogs is written with a dot '.' as separator (e.g. 'customerportal.users.Employee').

Click on *Finish* to actually rename the selected resource(s) or on *Cancel* to abort the operation.

Move Wizard



Select the destination project for the move operation from the proposed list.

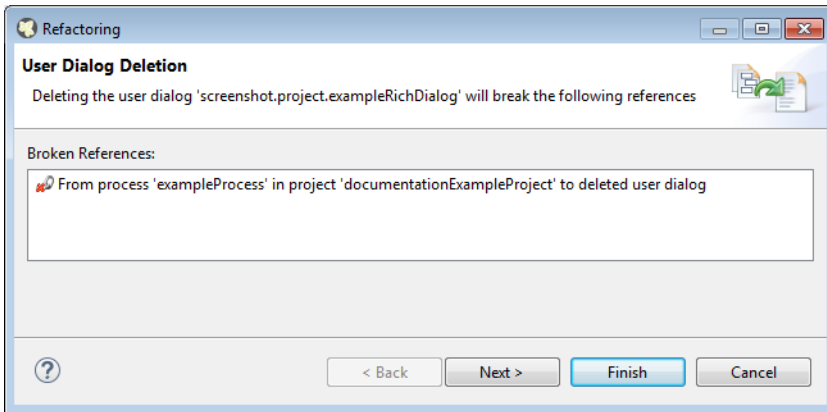
The moved artifact will keep its original name and namespace.

Click on *Finish* to actually move the selected resource(s) to the selected project or on *Cancel* to abort the operation.

Delete Wizard

If the selected resources are not Axon.ivy artifacts, then you will be presented with a confirmation dialog for the delete operation.

If you select an Axon.ivy artifact (Axon.ivy projects, User Dialog, Process, Data Class) for deletion, then you might be presented with a list of references that will break, if the operation is executed.

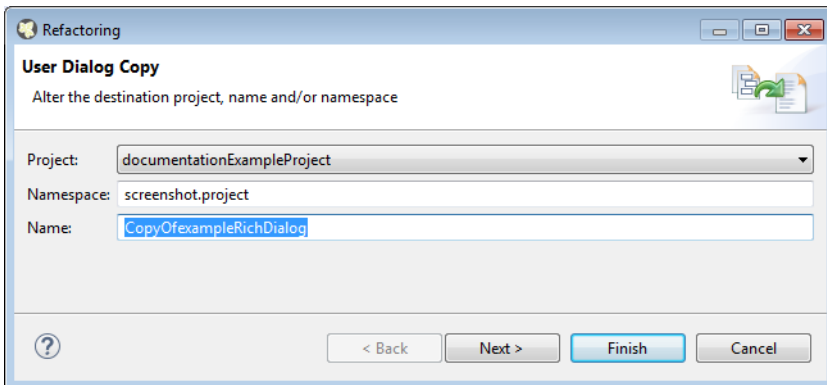


Click on *Finish* to actually delete the selected resource(s) or on *Cancel* to abort the operation.

Copy Wizard

The copy wizard appears when you execute the *Paste* operation (either through the menu action *Paste* or with *Ctrl-V*) after having copied something to the clipboard (e.g. through the menu action *Copy* or with *Ctrl-C*).

The copy wizard lets you change the project, name and namespace of the copy that will be created. All of the parameters are already filled in, the system tries to make educated guesses, if the selected target location is not valid or does not supply sufficient information (e.g. if a resource with the same name already exists at the paste location, then the name of the copy will be automatically have a "CopyOf" prefix).



Click on *Finish* to actually paste the copied resource(s) to the defined location or on *Cancel* to abort the operation.

Chapter 2. Process Modeling

Projects

Overview

Axon.ivy Projects can be seen as development modules that encapsulate the processes and other artifacts that form an application. An Axon.ivy project roughly comprises of processes, User Dialogs, Data Classes, a Content Management System and various configurations. All of those aspects are explained in separate chapters of this document.

Projects can be reused, i.e. any project can depend on functionality which is implemented by another project. Projects that implement reused functionality and/or artifacts are called *required projects* with respect to the project that makes use of that functionality. The latter is in turn called the *dependent project* with respect to its required projects.

Once you have finished your development you will usually want to install the implemented application or workflow on an Axon.ivy Engine. Projects form the single unit of deployment for this purpose, i.e. you deploy each project into a container on the engine which is called *process model version*. A project may be deployed in multiple versions on the engine; each process model version therefore contains a snapshot of a project at a specific point of time in development. See chapter Deployment for more information on this topic.

The data that specifies a project's deployment information is contained in the project's *deployment descriptor*. The deployment descriptor (formerly known as library) specifies all of the required projects and the specific versions in which they must be present on the engine in order for the deployed project to work. The descriptor also defines an unique deployment ID and the development version of a project (*not* equal to the process model version), as well as some information about the project provider and a description of the project itself.

On the engine, a project in a specific development state/version corresponds to a process model version, as explained above. On the engine, all the deployed versions of a project are children of a *process model* container (which corresponds to the project as an entity without a specific version). The process models themselves are part of an *application* (see chapter Deployment for a more thorough explanation).

In the Designer, projects may only exist in one version at a given point of time. Projects are created and organized inside an Eclipse *workspace*. Roughly, on the Designer, the *workspace* corresponds to the *application* on the engine. Since projects can only exist in one version on the Designer, there is no *process model* equivalent necessary in the Designer.

When working on a project, which depends on other projects, then the required projects need to be present as well in the Designer, which means that they must be present in the current workspace. Otherwise dependencies cannot be resolved and reused artifacts are not available, which will prevent the application from running.

Ivy Archives

There are two different types of Axon.ivy projects available. Normal Axon.ivy projects are used to develop artifacts. Artifacts in those projects are changed frequently. Once the artifacts of a project are developed and stable you can export the normal Ivy project to an Axon.ivy Archive. Archives are pre-built Ivy projects that are stored in one single *.iar file.

Ivy Archives can be imported to a workspace like normal Ivy projects. All artifacts of an Ivy Archive can be viewed but not edited. Archives already contain all built artifacts. Therefore, they do not have to be built or validated again in the workspace. As a consequence Ivy Archives will improve your workspace build, refresh and update time.

There are multiple ways to create or import Axon.ivy Archives:

- Axon.ivy Archives can be exported and imported.
- Axon.ivy Projects can be packed (archived) or unpacked (unarchived) inside the workspace.

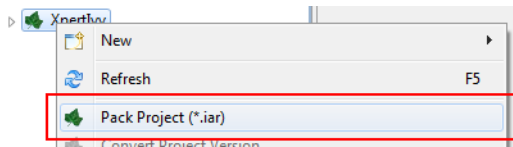


Figure 2.1. Pack Axon.ivy Archive (*.iar)

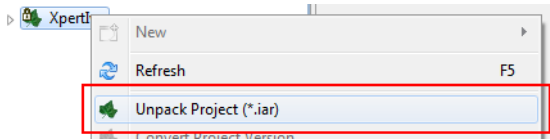


Figure 2.2. Unpack Axon.ivy Archive (*.iar)



Tip

Ivy Archives are not validated automatically. Validation can be started manually by using the context menu.

Ivy Project View

Here all the projects (including their content) in a given workspace are displayed in a tree view. This is the central component to obtain an overview of the project and to start the specific editors for all Axon.ivy entities.

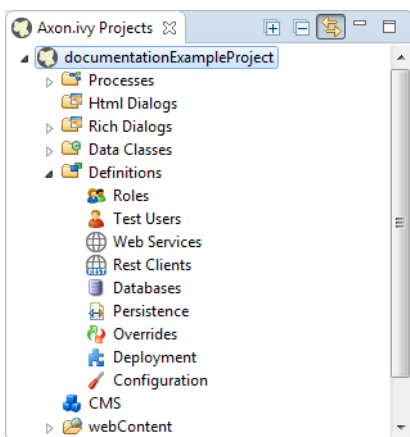


Figure 2.3. The Axon.ivy Project View with some content

Some of the entries are categorized such as User Dialogs and processes, but in general double-clicking on the leafs opens the corresponding editor. Furthermore a popup menu is provided with the most important interactions:

- *New...* - Opens a wizard for creating new Axon.ivy entities such as User Dialogs or processes
- *Refresh* - Use this to inform and refresh the project tree whenever the project resources have been changed externally. Axon.ivy with other applications.
- *Close Project* - Closes open projects. Closed project are still visible in the workspace but you cannot browse their content or execute them.
- *Open Project* - Opens closed projects.
- *Convert Project* - Converts a project so that it has the newest format.
- *Export Axon.ivy Archive (*.iar)* - Starts the Export Wizard to export normal Axon.ivy projects to Axon.ivy Archives.

- *Import* - Opens the Import Wizard. Very useful to import new projects from the file system or from a Source Repository such as Subversion or CVS
- *Export* - Opens the Export Wizard to exchange certain artifacts with other installations.
- *Rename* - Let you rename your resources (User Dialog, Data Class, Process, etc.) while keeping references to those artifacts intact. This menu item is only shown, if the selected resources are eligible for renaming. If renaming is possible, then the rename wizard will be shown, where you can enter a new namespace and/or name for the selected artifact.



Warning

Please, rename your resources only in Axon.ivy and not in *Java* or *Resource* perspectives. Trying to do renaming of Axon.ivy artifacts in other perspectives may result in an unusable project.



Tip

Commit your project in SVN before performing any rename operations.

- *Move* - Moves the selected resources to another project. The move wizard will be shown, allowing you to select the project to which the resource(s) should be moved.



Note

If Axon.ivy artifacts (such as User Dialogs, Processes or Data Classes) are moved, then the wizard will show an overview of the references (e.g. calls to sub processes) that might be broken by the operation.

- *Copy* - Copies the selected resource(s) to the clipboard
- *Paste* - Pastes the content of the clipboard into the selected node.



Note

The copy operation is intelligent: it tries to guess the correct location from the contents inside the clipboard, if the selected target node is not suitable for pasting. If there is a conflict upon paste (e.g. because the result would be two resources with the same name) then the copy wizard is presented with a new name suggestion, where you may modify the name and/or namespace of the pasted resource(s) before the operation is executed.

- *Delete* - Removes the selected node from the project. Multiple resources may be deleted at once.



Note

If Axon.ivy artifacts (such as Axon.ivy projects, User Dialogs, Processes or Data Classes) should be deleted, then the delete wizard opens and shows an overview of the references that might be broken by the operation.



Tip

Commit your project in SVN before performing any delete operations.

- *Open with* - Lets the user choose with which editor the selected entity is opened. It is possible to view a textual representation or a possible external editor for the entity.
- *Team* - Gives access to the Team functionality offered by CVS or SVN
- *Compare with* - Compares the current version of the entity with an older version from the local history or (if used) from the Source Repository.
- *Replace with* - Replaces the current version of the entity with an older version from the local history or (if used) from the Source Repository.

- *Properties* - Useful on the project level to set the properties and preferences of the project

New Project Wizard

Overview

The *New Axon.ivy Project wizard* lets you create a new Axon.ivy project. The wizard consists of three pages, of which two are optional.

On the first page you must specify the settings that are required for the new project. After filling those in, you may already press *finish* to create the new project.

The second and third page are optional and you do not have to complete them. However, they allow you to specify information with regard to deployment that you would otherwise have to specify at a later point of time, by using the *deployment descriptor editor*.

Accessibility

File -> New -> Axon.ivy Project

Features

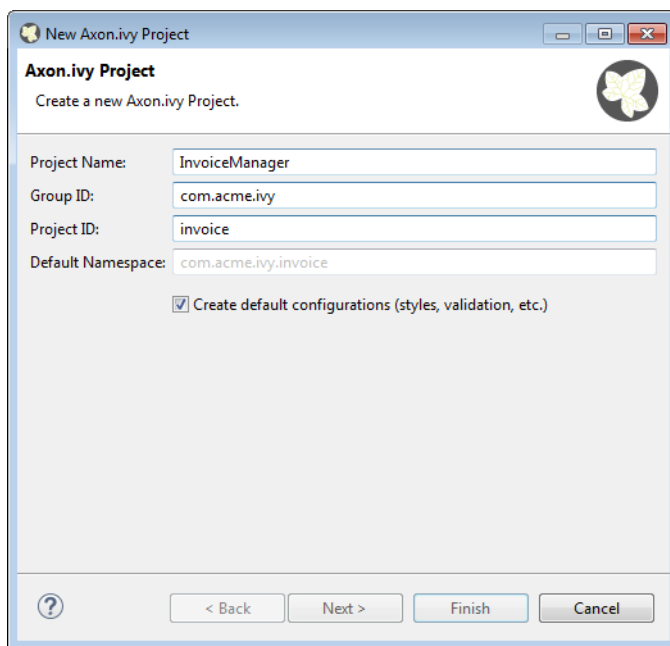


Figure 2.4. New Project Wizard: First Page

This page lets you define the minimally required settings for a new project.

Project name	Chose a name that describes the contents or the purpose of your project. You are not allowed to use any special characters or spaces.
Group ID	Identifies your project uniquely across all projects. It has to follow the package name rules, what means that has to be at least as a domain name you control, and you can create as many subgroups as you want. e.g. <code>com.acme.ria</code> .
Project ID	You can choose whatever name you want with lowercase letters and no strange symbols, e.g. <code>users</code> or <code>user-manager</code> .

During deployment to the engine the concatenated Group ID + Project ID will act as unique identifier of the project, once it is deployed.

Default namespace

Define the default namespace for your project. This namespace will be used as standard namespace for new Axon.ivy artifacts. It is also the namespace into which the project's default data class (`Data`) will be generated.

Create default configurations

If your project is a base or standalone project (e.g. if it doesn't have any dependencies on required projects) then you should leave this box checked. As a result of this, the new project will be initialized with default configurations in its configuration database.

However, if you're creating a project that is dependent on other projects (see wizard page 2) then you should uncheck this box, because configurations are inherited from required projects. If you'd leave the box checked, then the default configurations that would be created for the new project would possibly shadow (i.e. override) custom configurations with the same name from any required projects that you may have.

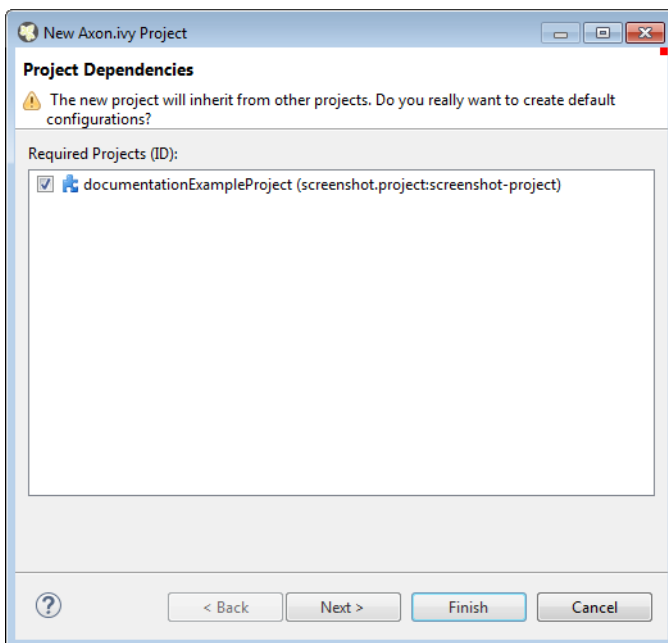


Figure 2.5. New Project Wizard: Second Page

The second page is optional. It allows you to specify any initially project from the workspace as a required project.

Required Projects

Check the projects that the new project should be depend upon. The selected projects will automatically be required with the version that they currently have in the workspace. The maximum version will be left open.

You can always reconfigure the required projects at a later point of time in the *Project Deployment* editor.



Warning

Please note that adding required projects may produce a warning (as shown in the snapshot above) due to the generated default configurations. The reason for this warning is explained in the *First Page* section above (Feature *Create default configurations*).

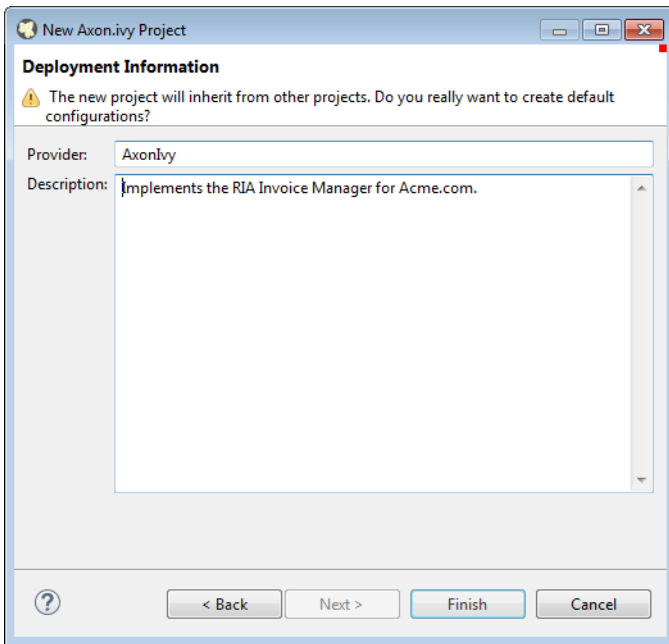


Figure 2.6. New Project Wizard: Third Page

The third page is optional. It allows you to define information about the implementor and the purpose of the new project. This information has documentation value only.

You can always specify and change this information at a later point of time in the *Project Deployment* editor.

Provider Define the company or individual that develops and maintains this project.

Description Describe the purpose of the project's contents or what the application is, that it implements.

Importing a Project

Overview

You can import existing Axon.ivy projects into your workspace using the *Import Wizard*. Projects can be exported from the workspace using the *Export Wizard* (See section Exporting a Project). This allows you to exchange or share your projects with other people.

Accessibility

You can access the Import Wizard over the menu:

File -> Import ...

Features

For Axon.ivy users the following import sources and formats are useful:

General > Existing Projects into Workspace	Imports a project from a project directory located somewhere in the file system into the workspace. The project directory may or may not be located in the workspace directory.
--	---

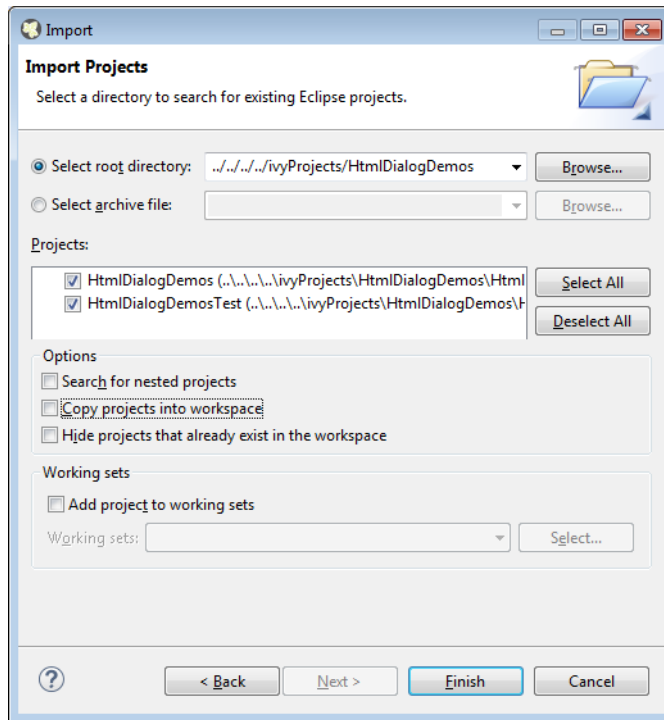


Figure 2.7. Import Wizard for Existing Projects

In the wizard page seen above you can select either the directory where your project(s) resides or a archive file (zip, jar, tar-gz) that contains the project(s). If Axon.ivy find valid projects in the given directory or archive file, they can be (de-)selected for the import and you can decide whether the projects should be copied into your workspace directory or not (which has no effect if a project already is in the workspace directory). After clicking on the button *Finish* the import is performed and you will find the imported projects in the Axon.ivy Projects View .

SVN > Checkout Projects from SVN

Checks out a project from a subversion source control repository into a new local working copy directory and imports it into the workspace.

Axon.ivy > Axon.ivy Archive (*.iar)

Imports Axon.ivy Archives (*.iar) into the workspace.

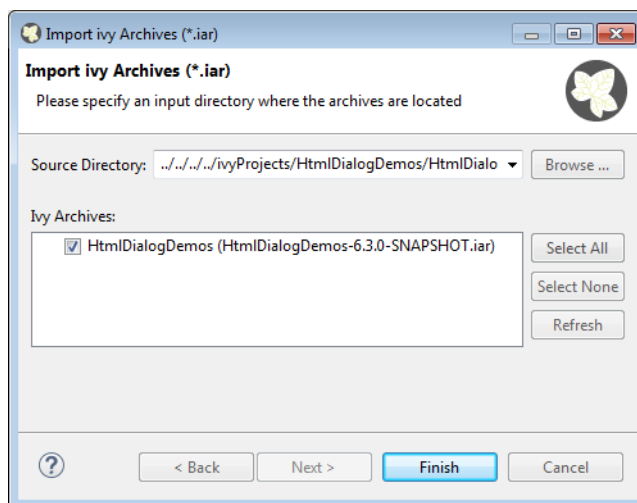


Figure 2.8. Import Wizard for Axon.ivy Archives (*.iar)

In the wizard page seen above you can select the directory where your Axon.ivy Archives resides. If Axon.ivy finds valid Axon.ivy Archives in the given directory, they can be (de-)selected for the import and you can decide whether the Axon.ivy Archives should be copied into your workspace directory or not (which has no effect if an Axon.ivy Archive already is in the workspace directory). After clicking on the button *Finish* the import is performed and you will find the imported Axon.ivy Archives in the Axon.ivy Projects View .

Xpert.ivy > Xpert.ivy 3.9 Project
(*.csp)

Imports a Xpert.ivy 3.9 project (*.csp) into the workspace. For more information how to import and convert a 3.9 project see chapter Converting old 3.x projects

Importing demo projects

The Axon.ivy Designer ships with several demo projects that are located in the `applications/samples` directory of the Designer installation. Those demo projects are delivered in the Ivy Archive (*.iar) format and can be imported with the help of the *Sample* icon on the welcome page.

Following projects are delivered with the Designer:

Project name	Demo content
ConnectivityDemos	Demonstrates the consuming and providing of REST services with ivy.
ErrorHandlingDemos	Samples that demonstrate the Error Handling.
HtmlDialogDemos	Demonstrates several JSF components that can be used in Html Dialogs.
QuickStartTutorial	The same project that is built in the QuickStart Tutorial.
RichDialogDemos	Demonstrates several ULC components that can be used in Rich Dialogs.
RuleEngineDemos	Shows how to use the Rule Engine.
WorkflowDemos	Demonstrates how to handle typical Workflow use cases, makes use of features like Signals and Business Data.

Table 2.1. Demo projects in the Designer.

Exporting a Project

Overview

Axon.ivy projects can be exported from the workspace to various output formats using the *Export Wizard*.

Accessibility

You can access the Export Wizard over the menu:

File -> Export ...

Features

For Axon.ivy users the following output formats are useful:

General > Archive File

Exports projects to a *.zip or *.tar file.

General > File System

Exports projects to the file system.

Axon.ivy > Axon.ivy Archive
(*.iar)

Exports a normal Axon.ivy project to an Axon.ivy Archive (*.iar file).

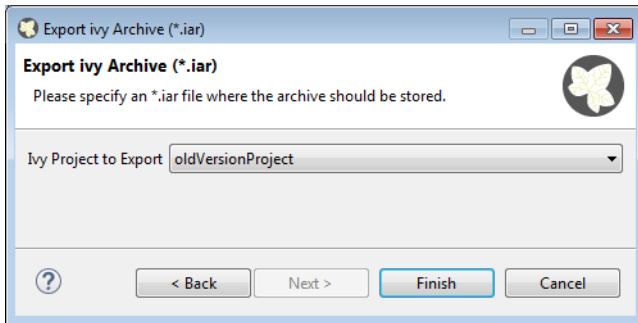


Figure 2.9. Export Wizard: Export Axon.ivy Archive (*.iar)

Converting old 3.x Projects



Warning

Not all project artefacts can be converted automatically. After importing a 3.9 project into your workspace there will be some problem and task markers on the imported project. You have to solve these problem and tasks manually before your project runs correctly in Axon.ivy 5.x

To convert an old Xpert.ivy 3.9 project you should do the following steps:

1. Open your old 3.x project with the latest Axon.ivy Designer 3.9 and save it. (This will convert the project to the latest 3.9 version).
2. Import the Xpert.ivy 3.9 project to your workspace.
3. Solve all problem markers.
4. Solve all task markers.
5. Test your imported and converted project.



Warning

Depending on the complexity of your project the steps 3 through 5 can take hours, days or weeks. Be sure that you plan enough time for the conversion and testing of your converted projects.

You may want to try an import and have a look at the problems and tasks view to get an idea how much manually work is necessary.

Import a Xpert.ivy 3.9 Project

To import a 3.9 project into your workspace use the *Import Wizard*. Use *File > Import ... > Xpert.ivy > Xpert.ivy 3.9 Project (*.csp)*.

On the Xpert.ivy 3.9 Project Import Wizard you have to specify the Xpert.ivy 3.9 Project File (*.csp):

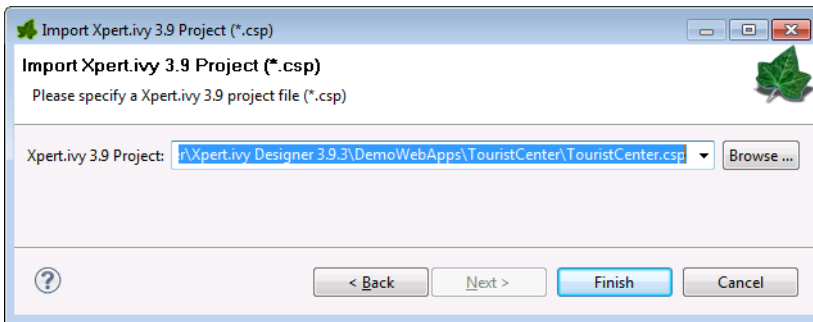


Figure 2.10. Xpert.ivy 3.9 Import Wizard

After you press *Finish* the following dialog appears:

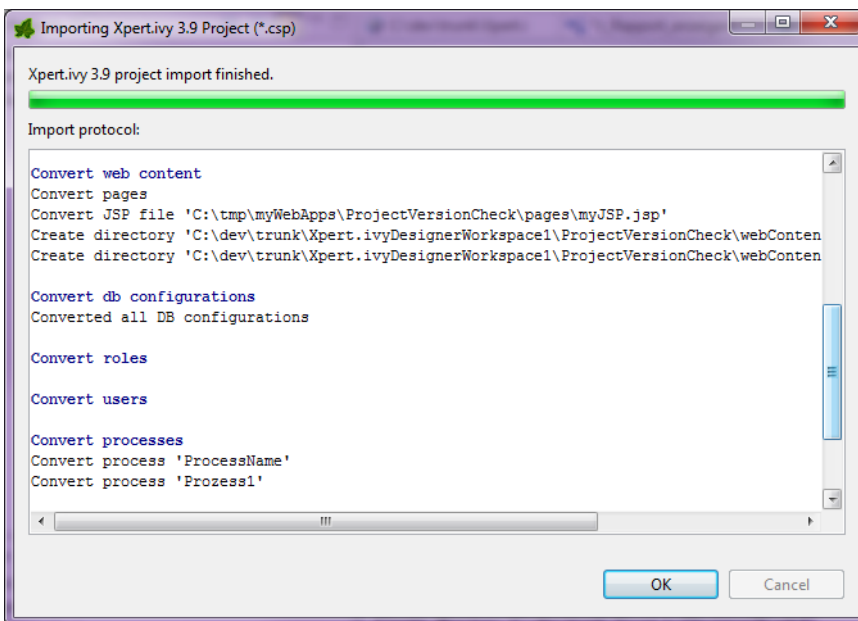


Figure 2.11. Xpert.ivy 3.9 Import Protocol and Progress Dialog

The *Import protocol* documents the conversion. After the conversion the protocol is stored in the converted project directory.

Manual Conversion Tasks

After you have imported the project into your workspace the project will have some problem and tasks markers. You have to solve all problem markers and do the tasks described by the tasks markers manually before your project will run correctly.

For more information about problem markers see chapter Problems View. For more information about task markers see chapter Tasks View.

Not supported 3.x functionality

5.x does not support all the functionality 3.x had provided. Therefore, some parts of your 3.x projects cannot be converted automatically and needs to be converted/redesigned in an other way. The following list shows the major missing functionality of 3.x.

3.x functionality	Missing in 4.x/5.x
Smart Chart	Smart Chart is not supported.

3.x functionality	Missing in 4.x/5.x	
Navigation Menu	/ CMS elements <code>Navigation</code> and <code>Menu</code> and their APIs are not supported.	
List and Recordset functions	Iteration functions like <code>inject(...)</code> or <code>reject(...)</code> are not supported.	
E-Mail Element	Adding a list of attachments to an e-mail is not yet supported	
Program Interface Element	The executed Java class needs to be refactored to implement the new interface <code>IUserProcessExtension</code> .	
Event Start Element	The executed Java class needs to be refactored to implement the new interface <code>IProcessStartEventBean</code> .	
Trigger Element	The element configuration must be converted manually.	
WS Process Start Element	The element configuration must be converted manually.	
DB Page Element	The <code>DB Page</code> element is not supported.	
SOAP Element	The <code>SOAP Element</code> element is not supported.	

Table 2.2. Missing 3.x functionality

Converting old 4.x Projects

If the project format version changes with a new Axon.ivy release, then old projects will show an error marker, describing them as *out of date* or having an invalid version. This can happen, when the technical format for Axon.ivy projects changes with a new Axon.ivy release (e.g. the way how some artifacts are stored may be changed, new artifacts may be introduced, etc.). :

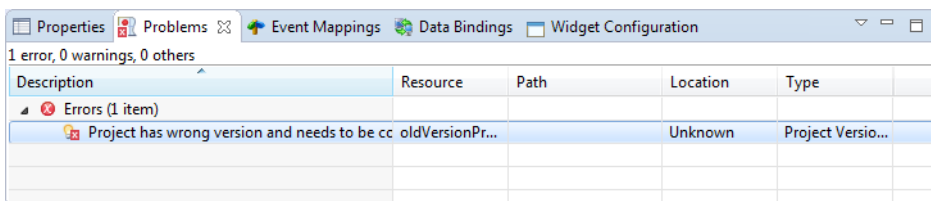


Figure 2.12. Wrong project version marker

If you inspect your project's properties, the main page will show you the actual project version and inform you whether it is up to date or not (see Project Properties below):

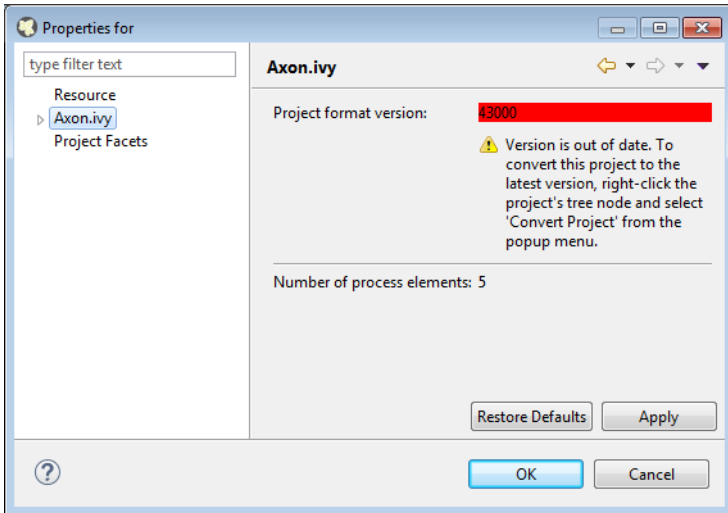


Figure 2.13. Project version before conversion

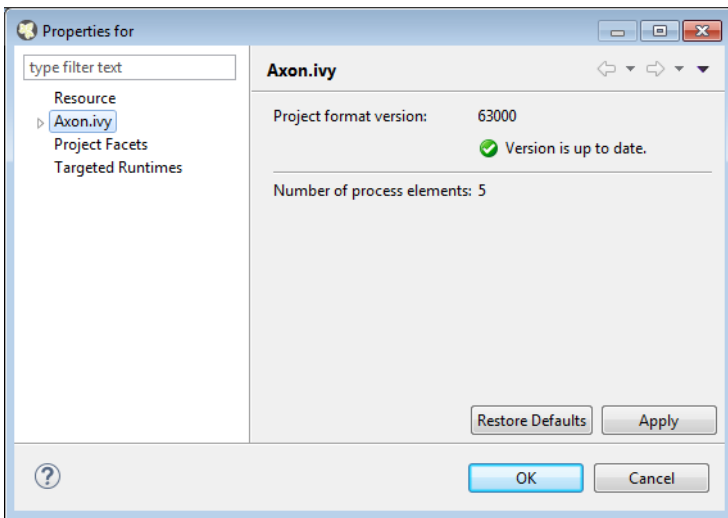


Figure 2.14. Project version after conversion

Axon.ivy can convert your old projects automatically to the newest project format for you. During this process, all existing artifacts will be converted (if necessary) so as to work with the new Axon.ivy version, and any missing but required artifacts will be added.

To run the project conversion, select the project's node in the Axon.ivy project view and right click to bring up the context menu. Select *Convert Project* to initiate the conversion. A log screen will appear that documents the conversion process (this log is also saved in the *logs/* folder inside your project), and which will inform you about whether the conversion was successful or not.

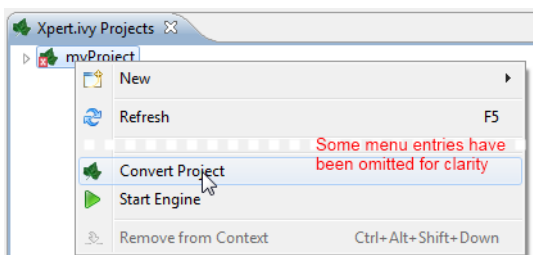


Figure 2.15. Invoking the project conversion



Note

You can not use this feature to convert 3.x projects. It only works for 4.x project versions.



Warning

It is absolutely recommended that you create a copy of your project before invoking the conversion. Alternatively you can have your project under version control. In this case, make sure that all your projects are checked in, before you invoke the conversion, so that you can easily roll back (revert) to the old version, if conversion should fail for some reason.

Project Properties (Project Preferences)

You can access the properties and preferences of a project either over the item *Properties* in the popup menu of the Axon.ivy Projects View or over the menu item *Project -> Properties*. Here you can redefine almost all of the global workspace preferences and override them with project-specific values.

Additionally, the project preferences allow you to define values for some project-only properties, that do not have a global default value. Those are described in the sections below.

Axon.ivy - Project Information

The main project properties page shows information about the project.

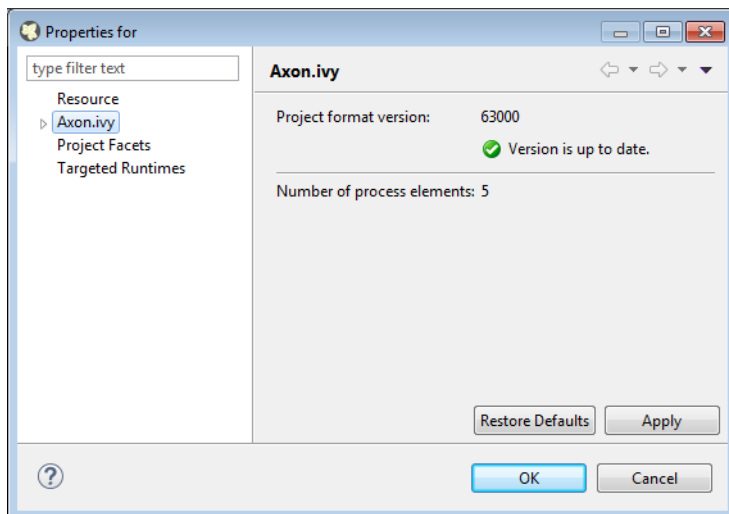


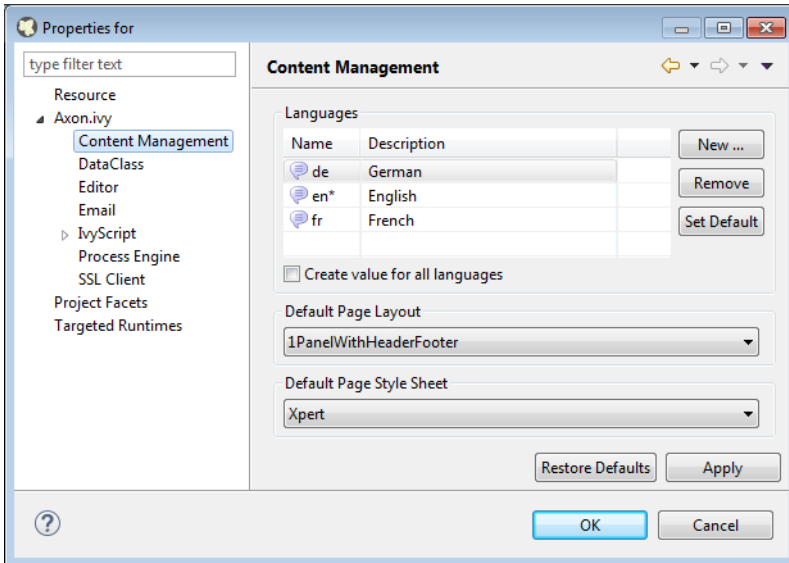
Figure 2.16. Project Properties Axon.ivy information

Project format version Shows the version of the project format. If the project was created with an old version of Axon.ivy, this is indicated with a warning message. Consult the Chapter Project Conversion to learn how to convert your project to a new version of the project format.

Number of process Elements Shows the number of process elements in this project.

Content Management System Settings

The languages in the CMS and the defaults for HTML dialog pages can be set here.



In the list at the top you can add and remove languages to/from the CMS and you can set the default language. Just below you can define whether Axon.ivy should automatically create a value for every language of the CMS if you create a new Content Object. Do not use this option if you do not need content in multiple languages or if you export the CMS content to translate it. Use the option if you know that you need to translate the vast majority of Content Objects within the Axon.ivy Designer

Furthermore, you have the choice between different HTML page layouts and CSS style sheets for use as default values for HTML dialog pages.

Data Class Settings

Allows you to specify the default namespace and the name of the project Data Class.

Java

With these preferences you can adjust the Java settings of the project.

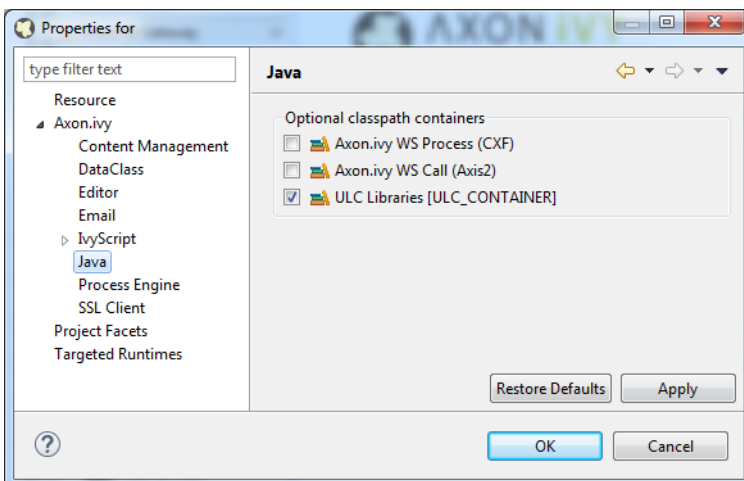


Figure 2.17. Java preferences

Optional classpath containers

Defines optional libraries which can be accessed by Java or IvyScript code of the project.

If migrated your project from 6.0 or older you may have used CXF or AXIS2 libraries by accident in your code. With the classpath container checkboxes you can put these libraries on the classpath to avoid compilation or runtime errors.

Project Deployment Descriptor

Each Axon.ivy project has a *deployment descriptor*. The deployment descriptor defines various properties of a project that are important with respect to deployment on the engine. Specifically the descriptor defines:

1. An *unique project ID* (i.e. a fully qualified symbolic name) for the project, by which it can be identified and referenced. Also, a current *development version* of the project is defined (please note that this version may, but does not necessarily have to be, identical with the project model version on the engine into which the project will eventually be deployed).
2. The *dependencies of a project to other projects* and the exact version range of those projects that must be available in order for the project to work. Once a project is referenced in this way, its artifacts may be used inside the referencing project. This applies especially to the following artifacts: User Dialogs, Data Classes, Web Service Configurations, CMS Entries, Configurations, Java classes or Java libraries (JAR files).
3. Information about the implementor of the project and its purpose.

The following figure illustrates the above:

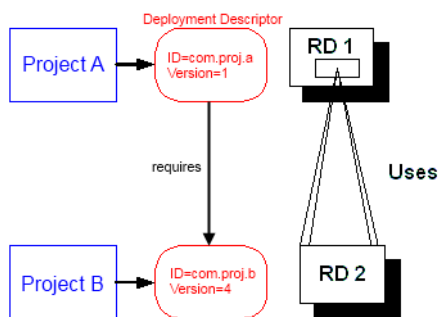


Figure 2.18. A project dependency, defined by the Project Deployment Descriptor

Since referenced projects may in turn reference other projects, a whole (acyclic) dependency graph may be constructed this way. All artifacts of projects that are reachable from some project in this way (i.e. by following the arrows) can be used.

The following figure illustrates this feature. For example, a User Dialog defined in *Project D* may be used in *Project A*. A Data Class that is defined in *Project E* may also be used in *Project A*. However, it is not possible to use a Sub Process defined in *Project B* from *Project C* (unless *Project B* is added as required project in the deployment descriptor of *Project C*).

The search order to look up reused artifacts is breadth first. The order, in which directly referenced projects are looked up, can be defined in the Deployment Descriptor editor.

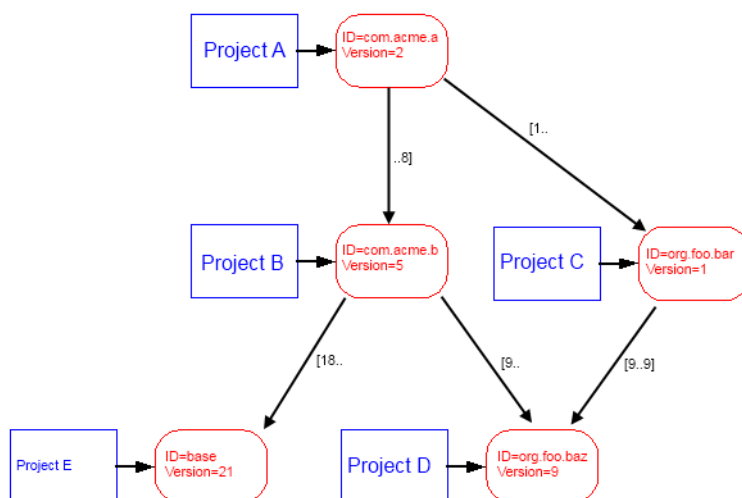


Figure 2.19. Project Dependency Graph

Projects may be required with a specific version or within a specific version range. This is also illustrated in the above figure.

When deploying projects on the engine, the availability of the required projects (and their versions) is checked. If the required project versions can not be resolved, then a project can not be deployed. Therefore projects must be deployed *bottom up*, i.e. one must start by deploying first the required projects that are lowest in the dependency hierarchy.

Deployment Descriptor Editor

The Deployment Descriptor editor allows to edit a project's deployment properties as well as the required projects and their version ranges as described above. Most of those properties can already be specified in the New Project Wizard, when a project is initially created.

The deployment descriptor editor consists of two tabs:

- The *Deployment* tab is used to configure the project's own deployment information.
- The *Required Projects* tab is used to define other projects (possibly in a specific version) that the project depends on.

The deployment description is stored as Maven pom.xml so that Ivy Projects can be built on a continuous integration server. See “Continuous Integration”

Accessibility

Axon.ivy Project Tree -> double click on the *Deployment* node inside the project tree (🔗)

Deployment Tab

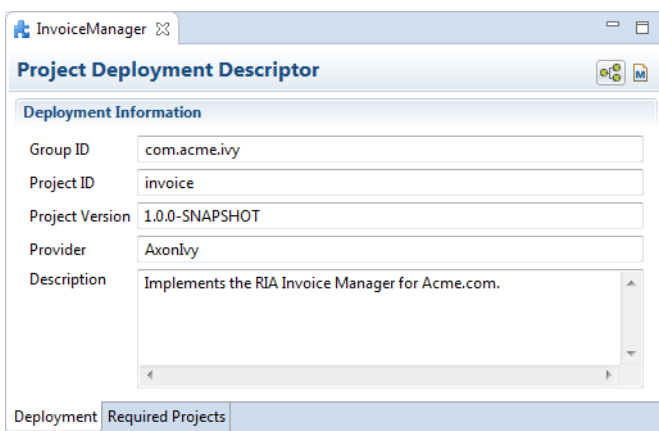


Figure 2.20. Deployment Descriptor Editor: Deployment Tab

Group ID	Identifies your project uniquely across all projects. It has to follow the package name rules, what means that has to be at least as a domain name you control, and you can create as many subgroups as you want. e.g. <code>com.acme.ria..</code>
Project ID	You can choose whatever name you want with lowercase letters and no strange symbols, e.g. <code>users</code> or <code>user-manager</code> . During deployment to the engine the concatenated Group ID + Project ID will act as unique identifier of the project, once it is deployed.
Project Version	The current development version of the project.
Provider	The name of the user or company that implements and maintains (i.e. provides) the project. The provider setting has not functional purpose. It is for documentation only.

Description A (short) description of the project's purpose and contents. The description setting has no functional purpose. It is for documentation only.

Required Projects Tab

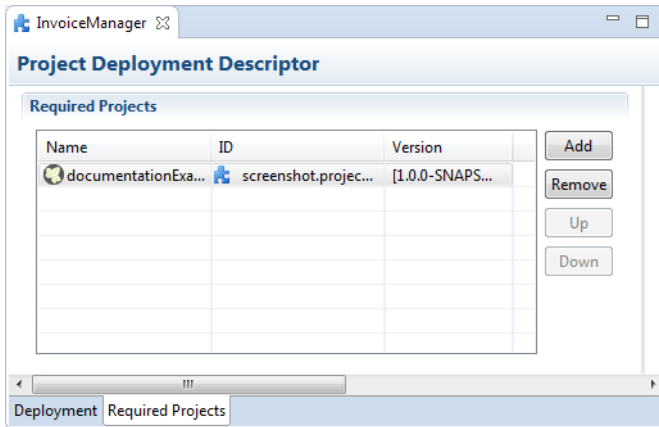


Figure 2.21. Deployment Descriptor Editor: Required Libraries Tab

Required Projects

A table shows the list of the required projects, both with their name and their ID (as defined in the project's deployment descriptor). The table also shows the version range in which the referenced project must be available.

Name The display name of the required project (how it is shown in the workspace).

ID The unique identifier of the required project.

Version The range specification of the version that the referenced project is required to have.

Note that the order in the table defines the order how referenced artifacts are searched (Use the **Up Button** and **Down Button** to change the order). The general search order in the dependency graph is *breadth first*, but the order that you define here is the search order that will be used at a specific node when searching the graph.

Clicking the *Add* button brings up a dialog with a selection box, in which any of the projects that are currently present in the workspace may be selected as required project. Closed projects or projects, that are already (directly) required, can not be selected.

Selecting an entry in the table and subsequently clicking the *Remove* button removes a project dependency.

Required Project Details

Shows the details of the currently selected project.

Group and Project ID The identifiers of the required project (not editable).

Maximum Version Optionally specify the maximum version that the required project needs to have. Choose whether you want to include or exclude this maximal version by checking the **Inclusive** box

Minimum Version Optionally specify the minimum version that the required project needs to have. Choose whether you want to include or exclude this minimal version by checking the **Inclusive** box

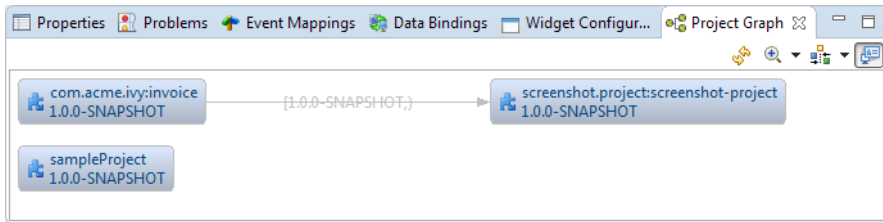


Warning

Beware of cycles in the project dependencies! You should never require a project B from a project A, if B also requires A (or if B requires any project that in turn requires A, which would form a larger cycle). Error markers will be displayed when the workspace is built, and cycles are detected, because this condition can lead to endless recursion and other unpredictable behavior when looking up artifacts.

Project Graph view

The Project Graph view shows the dependency graph of all projects in the workspace.



Toolbar actions



Refreshes the complete graph. Manually moved nodes will be re-arranged by the auto layout algorithm.



Selects the zoom level of the view.



Selects the layout algorithm that arranges the nodes and dependency edges in the view.



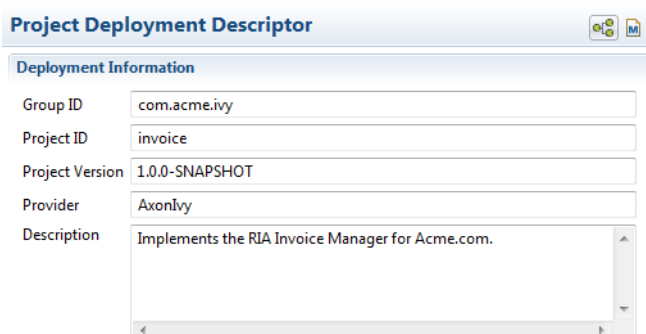
Automatically opens the Project Graph whenever a Library Descriptor Editor is opened.

Graph actions

- Double click on a node to open its Library Descriptor Editor
- Drag a node to improve the layout
- Click on a node to highlight it

Accessibility

- Windows -> Show View -> Axon.ivy -> Project Graph
- CTRL + 3 (Quick Access) -> Project Graph
- Deployment Descriptor Editor -> Open Project Graph from header toolbar



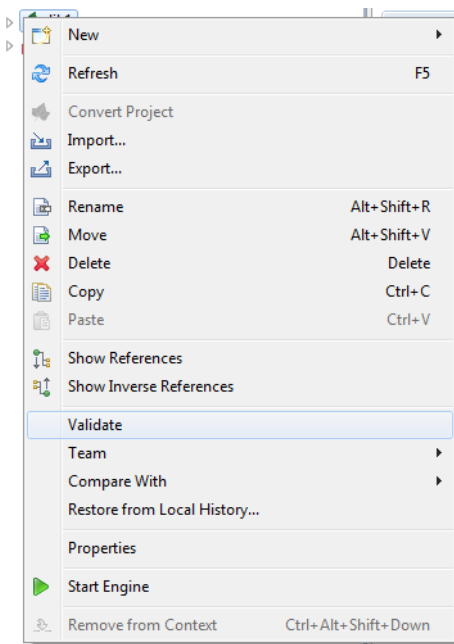
Validating Axon.ivy projects and resources

Overview

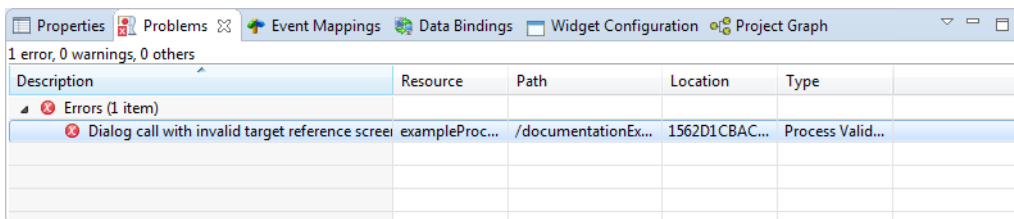
Axon.ivy comes with various validators which verify that projects and it's resources do not have any errors. After a resource has changed the responsible validator will run automatically and report errors or warnings.

Validating projects and resources

To manually validate a project or a resource you can right click on it and select **Validate**.



After the validation the errors are shown in the **Problems view**.



Validation preferences

Go to **Window -> Preferences -> Validation** to get an overview of the validations that are run.



Warning

It is recommended not to change these settings. It could lead to problems while running the projects.

Process Modeling

This chapter introduces Axon.ivy processes and how to work with them. The creation and logical organisation of processes is explained as well as the functionality of the Process editor and the different kinds of processes.

Process Kinds

There are different kinds of processes. Their use and capabilities are explained in the sections below.

Business Process

Business processes are the *regular* kind of processes that are used to implement business cases. Business processes contain starts that can be selected by any user from his/her workflow list or from the list of star table processes.

Embedded Subprocess

An embedded subprocess is essentially a syntactical collapse of elements into a single element to hide details from process design. They are available in all other process kinds. The hierarchy of embedded subprocesses is potentially indefinite, i.e. you can create further embedded subs inside an already existing subprocess.

Since *embedded subprocesses* are simply a structural means for process diagram simplification, no mapping of data is required when entering or leaving this kind of subprocess (i.e. inside an embedded subprocess the same data is available as inside the caller process).



Warning

Wrapping process elements into an embedded subprocess does not influence the functionality of most process elements. But the wrapping influences the way process elements are addressed by Axon.ivy. This may cause incompatibilities with older versions of the process and will hinder you to deploy such a process over an already deployed older version of the process. The process elements that may cause such incompatibilities are:

- Task Switch
- Task Switch Simple
- Intermediate Event
- Call And Wait

Independent Subprocess (Callable)

An independent subprocess (callable) is a process, that can be called from any other process with the *call subprocess element*. Independent subprocesses can be used to factor out frequently used functionality which can then be reused by any other process.

Because *callables* are *independent* implementations of processes (or parts of process logic) they have an own Data Class which might not match the caller's data. Therefore parameters need to be mapped in both directions when entering and leaving an independent subprocess.

To create an independent subprocess, select the *callable process* option from the New Process wizard. The created process will contain special start and end elements that must encompass the process implementation.

Web Service Process

Web Service processes are a special case of *independent subprocesses*. A Web Service process can be started (i.e. called) from any other application (or from another process) by using the *Web Service call element* or any other SOAP web service compatible client..

A web service process will provide a web service with one or more operations, which are defined by the *Web Service Process Start* elements within the process. Each of these start elements have their own input and output parameters that will be mapped to and from the process data.

Due to the nature of web services, which are intended to be called by another applications and not by a user directly, no user-interaction (HTML or User Dialogs) is allowed within such a process. If the process does contain user-interaction an exception will be thrown.

To create a web service process, select the *Web Service Process* option from the New Process wizard. The created process will contain special start and end elements that must encompass the process implementation.

User Dialog Logic

User Dialog logic processes are the implementation of the behavior of User Dialogs, the controller in the MVC pattern. A whole new set of elements is available for this kind of processes (from the *User Dialog drawer* on the process editor palette), while other elements (such as *task switch* or *HTML page*) are not available for conceptual reasons.

A User Dialog logic process is invoked with an *User Dialog* element inside a *business process*. Its execution starts with an *init start* element and ends with a *dialog exit* element. The two elements do not need to have a direct connection (in fact they never have). Once a User Dialog process is running, it is driven by *user interface events* which will trigger individual sub processes.



Note

Calling a *process based User Dialog* (and thus executing its logic) can (or rather should) be seen as equivalent to calling of a *callable process* with the sole difference that the User Dialog offers a user interface that allows a user to interact with the process logic directly.

However, from an abstract point of view, a User Dialog is nothing else than a function call. It is invoked with a list of (optional) arguments and returns a list of result values. This is exactly the same behavior as a callable process offers.

New Process Wizard

Overview

The *New Process Wizard* lets you create a new Business, Callable Sub or Web Service Process.

Accessibility

File > New > Process

Process Definition (page 1)

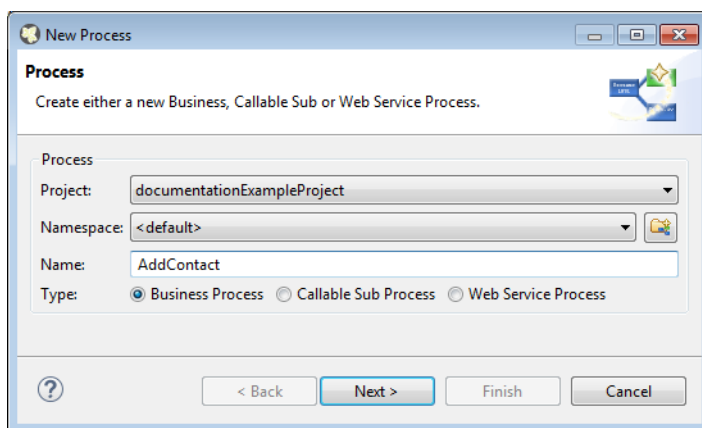


Figure 2.22. The New Process Wizard

- | | |
|-----------|---|
| Project | Choose the project where the new process should be created. |
| Namespace | Select a group where the new process will be inserted (this is roughly equivalent to a namespace). Select the <code><default></code> process group to create a process directly below the project's processes folder (i.e. equal to "no |

group"). You can click on the group folder button to open the *New Process Group Wizard*, if you want to create a new group "on the fly". The process groups are listed relative to the project's *process* folder.

Name Enter the name of the new process.

Type *Business Process*: This option is the default option and creates a normal standard business process. Use this option to implement your business logic.

Callable Sub Process: This option creates a callable sub process including a process-call-start element and a process-call-end element. You need to implement your process between those two elements. It is allowed to have multiple Process Starts and Process End elements in a callable process.

Web Service Process: This option creates a web service process which can be called from other systems. WS Start and WS End elements will be created automatically and you can implement your process between these elements. Please note that no user interaction may occur in a web service process.

Process Data (page2)

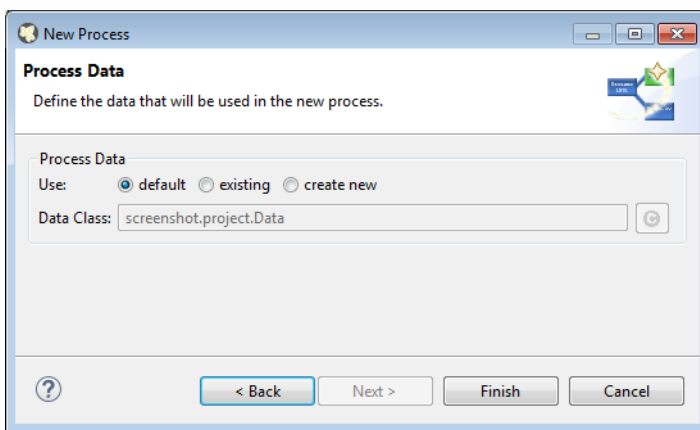


Figure 2.23. Simple Process Data selection on page 2

Process Data *default*: Select this option to use the project's default data class as data structure for the new process.

existing: Select this option to choose an already existing data class as data structure for the new process. Any existing Data Class can be chosen with the class selector button on the right side. **It is strongly recommended to select a data class from the project where the process will be created** in order to avoid dependencies on the implementation of another project.

create new: Select this option to create a new, empty data class that will be associated with the new process. Enter the name of the new data class to create (including namespace). Initially a data class name that is based on the new process' name and group will be suggested, but you're free to change it.

Process Data with simple mapping (page2)

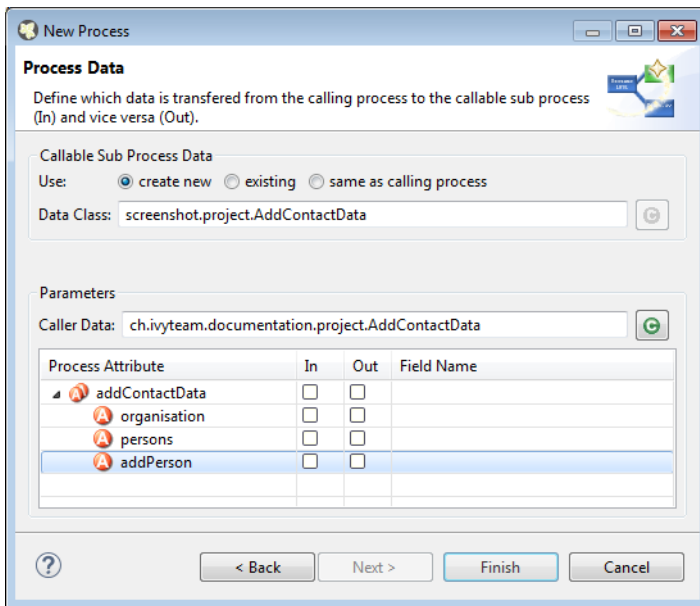


Figure 2.24. Process Data selection with auto data Mapping

Callable Sub Processes often consume or return data from a high level process. The data which is passed to and given back to the caller process can be easily mapped within this page.

Callable Sub Process Data Defines the Data Class which is used within the Process to create. The simple mapping parameters below are only available if a new Data Class is created or when the Callable Sub Process uses the same Data Class as the caller Process.

Parameters The *Caller Data* references the Data Class from the Caller Process. The fields of this Data Class can be automatically mapped to the Callable Sub Process Data.

In the mapping table below the Caller Data the In and Out arguments for the new Process can be defined. If any mappings are chosen, the Wizard will automatically configure the Call Sub Start Event, it's internal input mapping (param > in) and it's output mapping (out > result). The calling process element of the high level process will also be inscribed with input- & output mappings, if the new Process Wizard was opened from the Call Sub inscription step.

New Process Group Wizard

Overview

The New Process Group wizard lets you create a new grouping folder for business processes. Process groups can be nested.



Note

The process group is just used to categorize similar processes. A process is always treated independent from its parent process group(s)

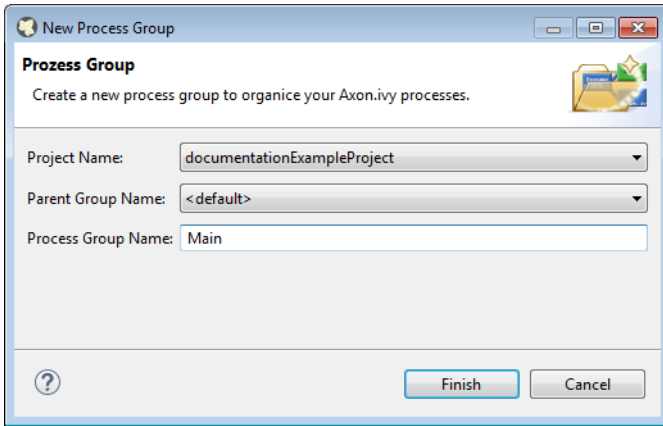


Figure 2.25. The New Process Group Wizard

Accessibility

File > New > Process Group

Features

- | | |
|--------------------|---|
| Project Name | Choose the project that your group belongs to. |
| Parent Group Name | Select a group that is the parent of your new creating group. |
| Process Group Name | Enter the name of the group that you want to create. |

Import Axon.ivy Modeler Processes

Overview

Processes exported in Axon.ivy Modeler as *BPMN XML Export* can be imported into the Designer through the *Axon.ivy Modeler Process importer*.

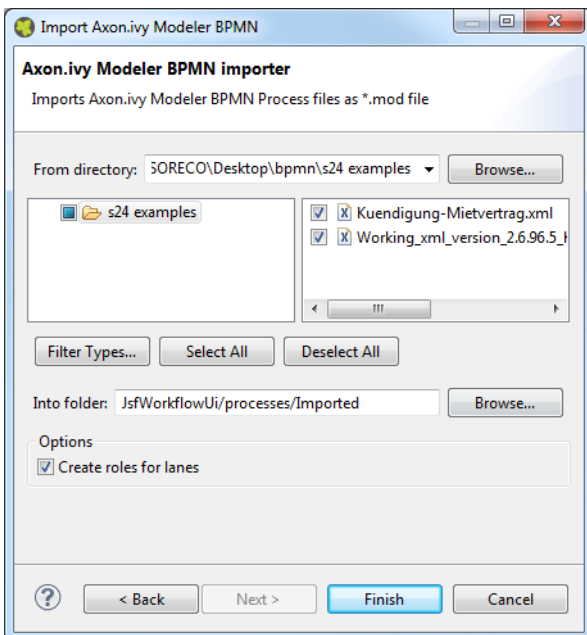


Figure 2.26. The Axon.ivy Modeler Process importer

Accessibility

File > Import > Axon.ivy Modeler Processes (*.xml)

Features

From directory	A directory containing the Modeler BPMN Files. It's allowed provided a directory structure.
File selection	Selection of any valid Modeler BPMN Files within the From directory
Into folder	A directory pointing to a process group. All imported processes will be stored in this process group. In case the specified folder does not exist, it will be created automatically.
Option: Create roles for lanes	If the Modeler BPMN Processes selected for import contain lanes, it's possible to create new roles from the lane names in Axon.ivy Designer automatically through the import by this option.

Compatibility

The Axon.ivy Designer supports the import of processes from the Axon.ivy Modeler 3.1.0. The internal version in the exported XML file is 97.1.0. Other versions or plain BPMN2 XML files might be imported anyway, but they are not supported.

```
<bpmn:definitions ... exporter="GBTEC BIC" exporterVersion="97.1.0">
```

Mapping of Elements by the importer

Axon.ivy Modeler and Axon.ivy Designer are tools to serve different needs, hence it's no possible to map any process element of the Modeler to exactly one corresponding element in Designer through the importer. As a consequence, the importer follows to main goals:

- Achieve as much recognition of the Modeler process as possible
- Provide a good basis for further implementation of process design

The mapping follows following rules:

Pools and lanes	Position, size and labels of swimlanes are adopted from the Modeler process.
Position and size of process elements	Position and size of process element nodes as well as any waypoints of process arc are adopted from the Modeler process.
Process events	In general, all process events are mapped directly to an event in the Designer.
Sub processes	Sub processes are mapped to an <i>Embedded Sub</i> element in the Designer.
Gateways	Exclusive gateways are mapped to an <i>Alternative</i> . Any other gateways are mapped to a <i>Task Switch</i> .
Tasks	In general, task process elements are mapped to a <i>BPMN Activity</i> in the Designer. If possible, the process element is mapped to a specific element e.g. <i>User</i> or <i>Manual</i> . The <i>BPMN Activity</i> elements behave basically similar to an <i>Embedded Sub</i> element, so it is possible, to implement the behavior of this process element at a lower level without changing the high level appearance of the process.
Links to other processes	Links to other processes are mapped to a <i>Subprocess Call</i> in the Designer, the call target remains empty after import.

Text annotations	Text annotations are mapped as <i>Annotation</i> in the Designer.
Data objects	Data objects are mapped as <i>Annotation</i> in the Designer, to distinguish them from text annotations, the differ in size and color.
Sequence Flows	Sequence Flows are mapped as <i>Connector</i> in the Designer.
Message Flows	Message Flows are mapped as <i>Message Flow</i> in the Designer.

Process Properties

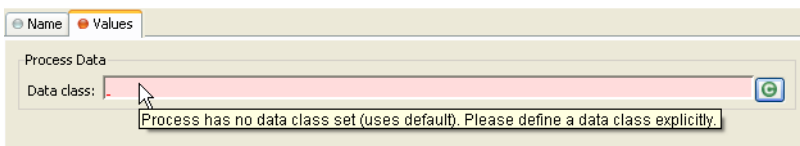
Like the process elements that are used inside a process, the process itself has an *inscription* that allows to specify and edit a processes properties. To open and show a the inscription mask of a process you simply select the process in the Ivy Projects View, right-click and select *inscription* from the pop-up menu.

Name and Description

The common name tab allows to specify name, description and associated documents for each process.

Values

The *values tab* allows to specify the data class that will be used to define the process's data structure.



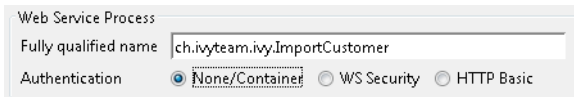
Each process must be associated with a data class, otherwise the tab will show an error. The used data class is initially specified with the New Process Wizard, but you may change this association at any later time.

You can use the (C) button next to the data class field to select any existing data class that is visible to the edited process. Please note that **it is strongly recommended that you only set data classes that are defined in the same project as the process** in order to avoid dependencies on the specific implementation of another project.

It is legal for two processes to specify the same data class. This can be desired if the processes operate on the same set of data (e.g. sub processes) and it may facilitate the mapping in some cases.

Web Service Process

The *Web Service Process* tab is only available on web service processes and allows to specify the web service configuration.



The *Fully qualified Web Service name* will be used to generate the web service class and the WSDL. The namespace part will be used as *targetNamespace* in the WSDL. Choose this name carefully since it should not be modified anymore as soon as clients start using the web service.

The *Web Service authentication* options allows you to specify how clients are authenticated when invoking the web service. You can select one of the following available authentication methods:

None/Container	Authentication is not handled by the web service element. However, if the web container (Tomcat) or a web server (Microsoft IIS/Apache) handles user authentication, the user is passed through to Axon.ivy (e.g. Single Sign On).
WS Security	UsernameToken with Password will be sent in clear-text to the ivy engine.



Warning

Only use this option in a trusted network or over a secure connection (e.g. HTTPS).

HTTP Basic

Username and Password will be sent in clear-text to the ivy engine using standard HTTP Basic authentication mechanism.



Note

HTTP Basic is the only authentication option that is supported by Web Service processes and Web Service process elements in common. It can therefore be used to call a Web Service process from a Web Service process element if authentication is required.



Warning

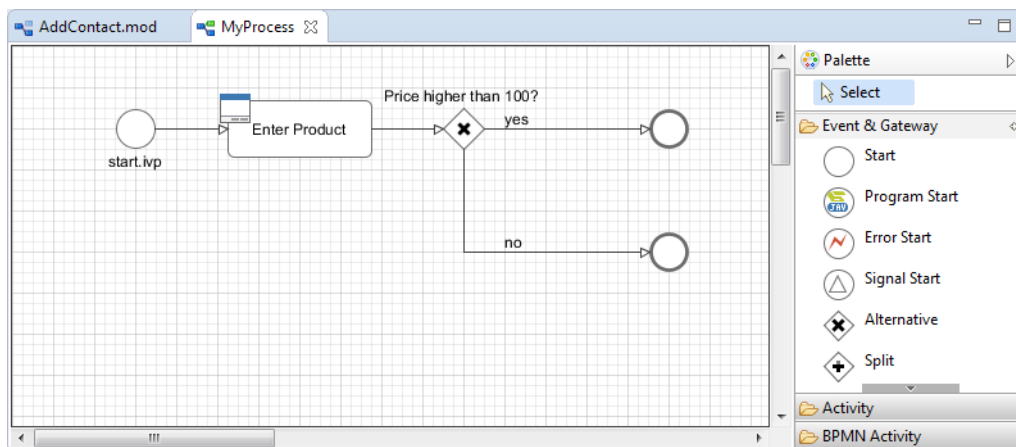
Only use this option in a trusted network or over a secure connection (e.g. HTTPS).

If the web container (Tomcat) or a web server (Microsoft IIS/Apache) already handles user authentication, the user is passed through to Axon.ivy without doing an additional HTTP Basic authentication.


Process Editor

The Process editor is used to design and edit the different process kinds (mostly *business* and *User Dialog logic* processes). The Process editor consists of two parts:

- the editor area where the process logic is constructed element for element and
- the palette where the elements that are to be placed inside the process are selected



Accessibility

Axon.ivy Project Tree > double click on a *process* node inside the project tree ()

Palette

The palette shows the process elements that are available for a specific process kind. The set of available process elements may vary for different process kinds.



Tip

The purpose and configuration of all available process elements are described in detail in the process elements reference chapter.

Editor Area

Processes are designed, drawn and modified in the process editor area. Select an element from the palette, then click in the process editor area to place it. Click and drag elements to replace them.

Arrows are drawn between two elements by clicking on the first element, then holding the left mouse button down until releasing on the second element.

You have four context menus available in the Process editor: the *editor menu*, the *element menu*, the *arrow menu* and the *selection menu*.

Editor Menu

To open the *editor menu* right click anywhere on the editor area's canvas (i.e. background). The following actions are available:

Leave Subprocess	Will jump out of an <i>embedded subprocess</i> to the process that contains the <i>Embedded Sub</i> element.
Select All	Selects all process elements.
Copy (as Image)	Copies the whole process (as image only) to clipboard.
Insert template	Inserts an existing <i>process template</i> . Opens a selection dialog to choose the template to be inserted, then inserts the selected template at the current mouse position. All of currently defined process templates are also available from the Process Template View.
Paste	Pastes a previously <i>copied</i> or <i>cut</i> element into the process at the current mouse position.
Undo	Undo the last drawing command. The process editor keeps up to 100 commands in the history buffer that can be undone.
Zoom In	Zoom in to get a close-up view of the process model. The view is enlarged by a factor of 20%. With a wheel mouse, you can also zoom in with the wheel together with the Ctrl key.
Zoom Out	Zoom out to see more of the process model at a reduced size. The view is reduced by a factor of 20%. With a wheel mouse, you can also zoom out with the wheel together with the Ctrl key.
Zoom 100%	Reset the zoom factor to the default size.
Change orientation of swimlanes	Changes the orientation of pools and lanes from <i>horizontal</i> to <i>vertical</i> or vice versa.
Add pool	Adds another pool before the swimlane at the current mouse position.
Add lane	Adds another lane before the lane or inside the pool at the current mouse position.
Edit pool/lane	Opens the configuration of the pool or lane at the current mouse position
Remove pool/lane	Removes the pool or lane at the current mouse position
Inscribe Process	Opens the configuration editor of the process.

Element Menu

To open the *element menu* right click on a process element. The following actions are available:

Copy	See selection menu.
------	---------------------

Cut	See selection menu.
Inscribe	Opens the configuration editor of the process element.
Wrap Text	Places the name of the element inside the element's icon. The icon size is stretched accordingly.
Move Text	Replaces the element's text with a box that can be moved around. You can also achieve this by simply clicking and dragging an element's associated text.
Style	See selection menu.
Open Document Reference	Opens document URLs which are configured in the elements 'Name' inscription tab.
Attach boundary event	Attaches an additional boundary event to the currently selected activity.
Breakpoint	Add a <i>regular</i> or <i>conditional</i> breakpoint to the element or remove all breakpoints from the element.
Connect	Creates an arrow that starts at this element. Click on another element to create a connection between the two elements. You can also create an arrow by clicking on the process element where the arrow should start and then move the mouse while you keep the mouse button pressed to the process element where the arrow should end.
Disconnect	Disconnects this element from another element. Click on another element to remove the connection between the two elements.
Move	See selection menu.
Extended Functions	Select from extended layout functions for the element. You can <i>reset the default size</i> of an accidentally resized element. If elements are placed on top of each other you may <i>send an element to the back</i> or <i>bring it to the front</i> of the element stack.
Delete Element	Deletes the element.
The visibility of the following menu entries are depending on the type of the process element:	
Start Process	Starts the process that begins at the process element.
Send Signal	Opens a dialog to send a signal. The dialog uses the signal code configured on the process element as default value.
Enter Subprocess	Enters the embedded subprocess and shows the encapsulated process.
Toggle Transparency	Changes the transparency state of the embedded subprocess. This either hides the process that is encapsulated by the embedded sub element or makes it visible.
Unwrap Subprocess	The elements encapsulated by the embedded subprocess are placed into the current process.
Change type	Converts the <i>Embedded Sub</i> into another subprocess type (e.g. from <i>BPMN User Activity</i> to <i>BPMN Send Activity</i>). The inner fields will be kept, but its field ids will change. This makes the Process Model incompatible as when elements are wrapped for the first time. See “Embedded Subprocess”
Search callers of this process	Displays all callers of a Start in the Search view.
Search callers of this exception element	Displays all process elements that call an Exception Start when an exception occurred.

Jump to connected element	Will jump out of an <i>embedded subprocess</i> to the process that contains the <i>Embedded Sub</i> element and selects the process element that is connected with the Embedded Start or End Event.
Jump to referenced process	Opens the process that is referenced by the process element.
Jump to User Dialog Process	Opens the process of the User Dialog that is referenced by the process element.
Edit Page	Opens the web page configured on the process element. If no page is configured then the <i>Create New Page</i> dialog is opened.
Edit Java Class	Opens the Java editor with the Java class configured on the process element. If no Java class is configured the New Bean Class Wizard is opened.
Edit User Dialog	Opens the view editor (e.g. ULC Visual Editor or JSF Editor) for the selected User Dialog.

Arrow Menu

To open the *arrow menu* right click on a an arrow. The following actions are available:

Inscribe	Opens the configuration editor of the arrow that the mouse is placed over.
Move Text	Replaces the arrow's text with a box that can be moved around. You can also achieve this by simply clicking and dragging the arrow's associated text.
Bend	Relayouts the arrow's path on the editor's grid (use only rectangular angles).
Straighten	Relayouts the arrow's path into a direct line without any angles.
Color	Changes the color of the arrow.
Bring to front	If elements and arrows are placed on top of each other then this action brings the one with the cursor over it to the front of the element stack.
Send to back	If elements and arrows are placed on top of each other then this action sends the one with the cursor over it to the back of the element stack.
Reconnect	Detaches the selected arrow's head from the element it is connected to and let's you reconnect the arrow to another element.
Delete connector	Deletes the selected arrow.

Selection Menu

To open the *selection menu* right click on a selected element or a group of selected elements (i.e. selection frame is visible). The following actions are available:

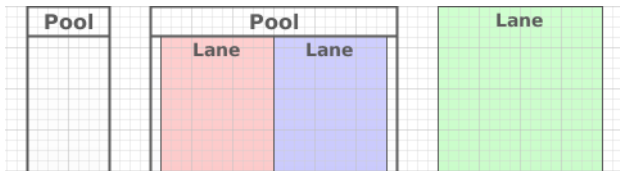
Copy	Copies the selection to the clipboard.
Cut	Copies the selection to the clipboard and deletes all contained elements from the process.
Style	Sets the style of the selected elements to a style in the predefined list of styles.
Auto Align	Aligns the selected elements <i>horizontally</i> and <i>vertically</i> .
Same Width	Assigns the same <i>width</i> to all of the selected elements. The resulting width is determined by the widest element in the selection.

Same Height	Assigns the same <i>height</i> to all of the selected elements. The resulting height is determined by the highest element in the selection.
Same Width and Height	Combination of the menus <i>Same Width</i> and <i>Same Height</i> .
Set to default size	Resets the size of the selected elements to their default sizes.
Wrap into Subprocess	Creates an <i>embedded subprocess</i> from the selected elements.
Create template	Creates a new <i>process template</i> from the selected elements. After prompting for a name for the selection, the new template will be available from the Process Template View.
Delete selection	Deletes all of the selected elements from the process.

Shortcut Keys

Some of the entries in the context menus are available with shortcut keys. To use them, place the mouse over a process element and press the according key.

Swimlanes



Processes can be visually structured by using *pools* and *lanes*. Pools and lanes are colored background swimlanes with a label that are placed *behind* the process logic. Swimlanes can have a *horizontal* or *vertical* orientation.

Swimlanes are available for all process kinds and are typically used to visualize organisations, roles, responsibility assignments or systems for process elements or sections of process logic.

A pool or lane can be widened or narrowed by dragging it's border/edge with the mouse. By default, the position of process elements lying outside the modified lane are adjusted accordingly. By pressing the **Shift-Key** during the drag, you can omit the automatic adjustment of process elements.



Note

Please note, that pools and lanes do not have any *syntactical* meaning whatsoever; their purpose is purely semantical. A pool or lane is *not* a container that elements are placed in or associated with. They are simply a structured "coloring" of the process' background; they do not grow or shrink when you change the processes logic and need to be adjusted manually.

Process Model Reporting Wizard

Overview

The Axon.ivy Process Model Reporting Wizard lets you create customized reports of your process models.

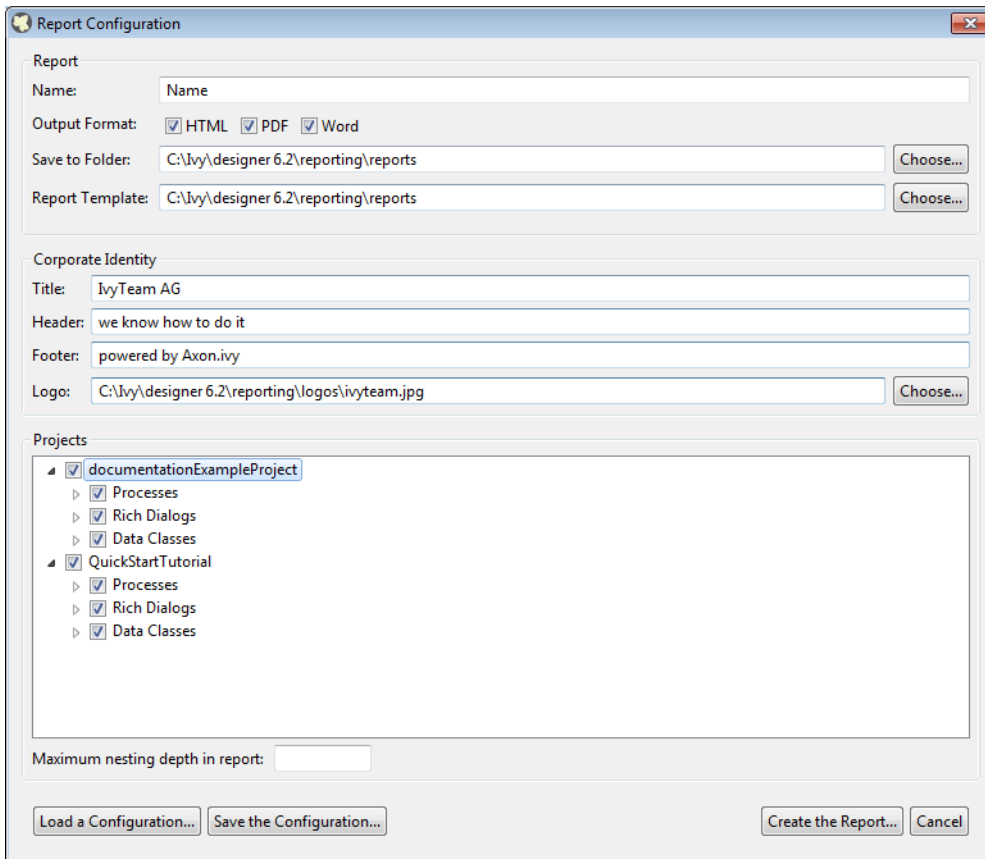


Figure 2.27. The Process Model Reporting Wizard

Accessibility

Axon.ivy->Create Report...

Features

Name	The name of the report that will be created. This name should be without file name extension. E.g. use "MyReport" instead of "MyReport.pdf".
Output Format	The report output format. Currently this can be HTML, PDF or DOC. You can also select multiple report formats that should be created simultaneously.
Save to Folder	Choose the location where the reports should be generated to. The default destination where reports are stored is <i>IvyDesigner/reporting/reports/</i> .
Report Template	Choose a report template, also known as <i>BIRT report design</i> file (*.rptdesign) which defines the structure and contents of your report. There are some BIRT report designs provided by default (e.g. <i>Default.rptdesign</i>). Please use the predefined report designs unless you want to create a custom report design.

Creating a custom BIRT Report Design

In case you want to create a custom BIRT report design you first need to install a BIRT Report Designer which can be found on the BIRT Website. With the BIRT Report Designer you can create your own reports. In order to use the Process Model Data, as e.g. Process model images, process names, User Dialog interfaces, data class attributes etc., or predefined themes in your newly created report design, you need to use the *IvyDesigner/reporting/designs/Axon.ivy.rptlibrary* BIRT Report Library within your report and link against its Data Source, Data Sets and Parameters. In this way you will also be able to use the predefined themes of the Report Library.

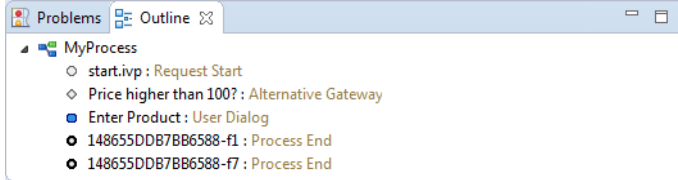
For further information on how the BIRT Designer can be used, please refer to a BIRT Book or online Resource which can be found at <http://www.eclipse.org/birt/> or <http://www.birt-exchange.com>.

Corporate Identity	<p>This group of text fields provides you some additional, optional information to customize your report.</p> <ul style="list-style-type: none"> • Title: Select a Title that will be shown on the first page of your report. • Header: Select a Header for the report, that will be shown on every page. • Footer: Select a Footer for the report, that will be shown on every page. • Logo: Select a Company Logo Image that will be displayed on the first page of your report.
Projects	<p>This Tree shows the currently active Projects that can be reported. You may check or uncheck the individual Process Models, Process Groups, Processes, Rich Dialogs or Data Classes that are to be reported.</p>
Maximum nesting depth	<p>Choose the maximum depth up to which nested embedded sub processes should be reported. By default and when the field is empty all embedded sub processes are reported.</p>
Cancel Button	<p>To cancel report creation. The current report configuration settings will be stored to <i>your_ivy_workspace/.metadata/.plugins/ch.ivyteam.ivy.designer.reporting.ui/lastReportconfiguration.xml</i>.</p>
Save the Configuration...	<p>To save the report configuration you have entered up to now into an XML report configuration file (*.rptconfig). This allows you to store multiple configurations for different types of reports and reuse them later. Note that currently the selected Projects, Processes, Rich Dialogs etc. are not remembered, as they might not be available at loading time. The default place where the report configurations are stored is <i>IvyDesigner/reporting/configurations/</i>.</p>
Load a Configuration...	<p>This allows you to load a previously stored report configuration files (*.rptconfig).</p>
Create the Report...	<p>This will start the generation of the reports. While the report generation you will be informed about its progress. After the report has been generated a confirmation window will provide you with links to the generated reports. The default destination where reports are stored is <i>IvyDesigner/reporting/reports/</i>.</p> <p>The report configuration will be stored to <i>your_ivy_workspace/.metadata/.plugins/ch.ivyteam.ivy.designer.reporting.ui/lastReportconfiguration.xml</i></p>

Process Outline View

Overview

The outline view displays all elements of the process which is currently opened in the process editor.








Accessibility

Window > Show View > Other... > General > Outline View

Features

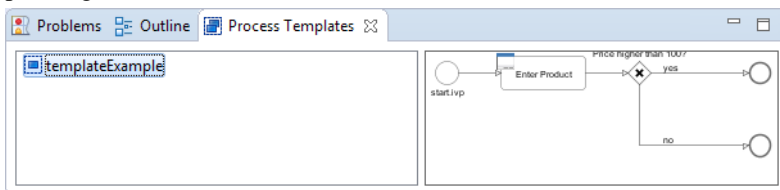
The outline view has the following features:

- | | |
|----------------|--|
| Selection | Process elements which are selected in the outline view are selected in the process editor and vice versa, which helps to search and manipulate elements, especially in large processes. |
| Classification | Elements are grouped by their BPMN type, where the element type is visualized with an icon in front of the element name. The element categories are start events  , intermediate events  , end events  , gateways  and tasks  . |

Process Template View

Overview

The process template view displays the currently defined process templates. A *process template* is essentially a selection of process elements that is stored under a specific name. Once defined, process templates can be inserted into any existing process, either by drag and drop or by selection from a dialog. New process templates can be added to the template store by pressing 't' on a selection of elements in the Process Editor.



Accessibility

Window > Show View > Axon.ivy > Process Template View

Features

The process template view has the following features:

- | | |
|---------|--|
| Preview | A preview for each selected template will be shown on the right-hand side of the process template view, showing its structure in detail. |
|---------|--|

- Drag-and-drop Templates can be dragged and dropped on the process editor. Press and hold the mouse down over a template name and drag it over to the process editor to insert the template.
- Context menu Selected templates can be renamed and deleted using the context menu or by pressing 'R' or 'DEL' keys, respectively.

Export / Import

Process templates are stored per workspace. To export a set of process templates from a workspace use *File > Export... > General > Preferences > Process Templates*. To import a set of template into a workspace use *File > Import... > General > Preferences*.

Problems View

Overview

The problems view displays errors and warnings (problem markers) that exists in yours projects. You can double click an error or warning in the view to open the associated editor.

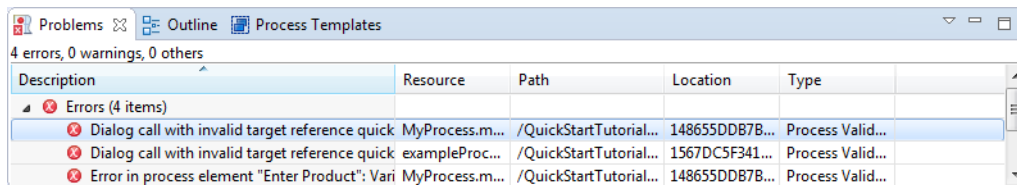


Figure 2.28. Problems View

In the process editor process elements that have errors are marked with an error overlay icon.

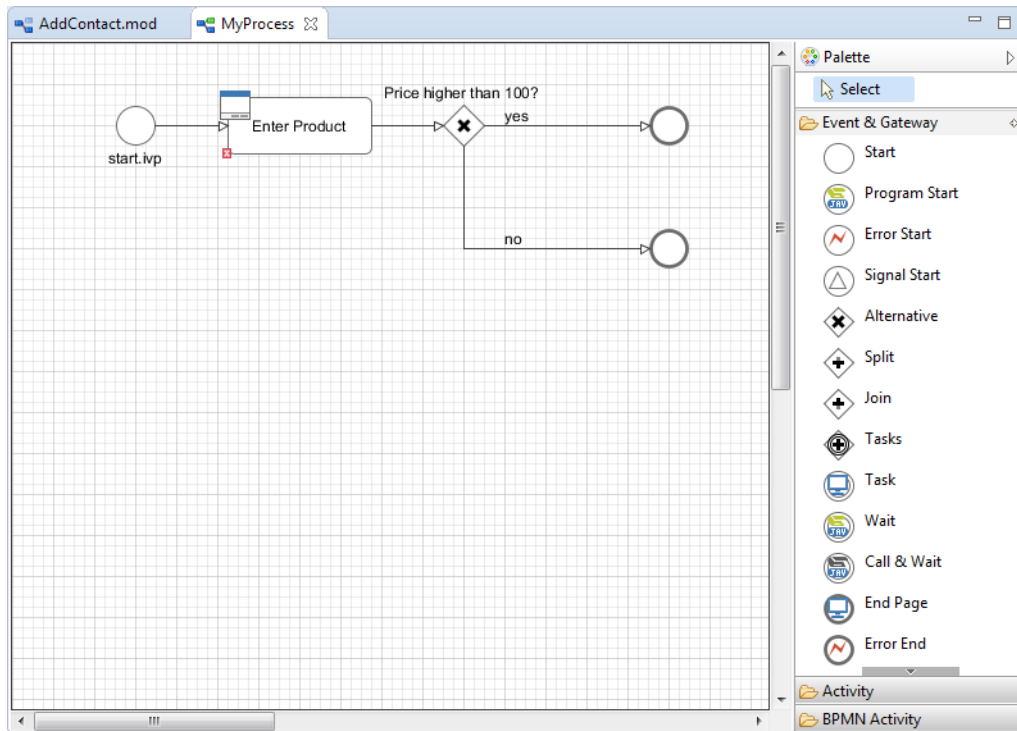


Figure 2.29. Process Element with Problem Markers

Accessibility

Window > Show View > Problems

Features

This view is a standard Eclipse IDE view. More information about the Problems View can be found in the Online Help: *Workbench User Guide > Reference > User interface information > Views and editors > Problems View*.

Tasks View

Overview

The tasks view displays tasks that exists in yours projects. You can double click a task to open the associated editor.

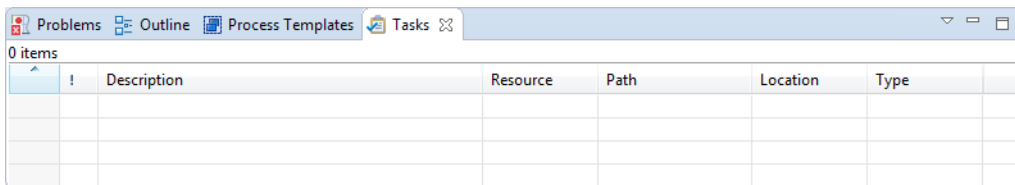


Figure 2.30. Tasks View

In the process editor process elements that have tasks are marked with a task overlay icon.

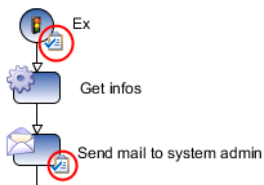


Figure 2.31. Process Elements with Task Markers

Accessibility

Window > Show View > Other ... > General > Tasks

Features

This view is a standard Eclipse IDE view. More information about the Tasks View can be found in the Online Help: *Workbench User Guide > Reference > User interface information > Views and editors > Tasks View*.

Reference View

Overview

The Reference view shows the references between the various Axon.ivy project artifacts. A reference of an artifact is everything which is used/called from the artifact (e.g. call to a callable process or User Dialog) or which is embedded in

the artifact (e.g. embedded sub element in a process or processes inside a project). Inverse references are the opposite of references. This means an inverse reference of an artifact is everything which uses/calls the artifact or which contains it.

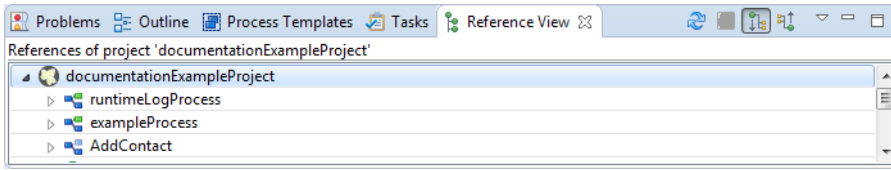


Figure 2.32. The Reference View



Tip

To work with references of process elements, there are also some useful features on the Process Editor “Element Menu”

Overview of supported References

The following table shows all supported references between Axon.ivy project artifacts.

	contains						uses / calls					
	Projects	Processes	Callables	User Dialogs	Embedded Sub	Data Classes	Projects	Processes	Callables	User Dialogs	Embedded Subs	Data Classes
Project		X	X	X		X	X					
Process					X			X	X	X ^a		X
Callable					X				X ^b	X ^a		X
User Dialog				X ^a	X				X	X ^{ab}		
Embedded Sub					X				X	X ^a		
Data Class (Entity Class)												X

^aUser Dialogs can be referenced also from other projects.

^bCallable can reference itself.

Table 2.3. Overview over the supported references

Accessibility

Window > Show View > Reference View

Right click on a project, process, User Dialog or embedded sub element in the project tree > Show References or Show Inverse References

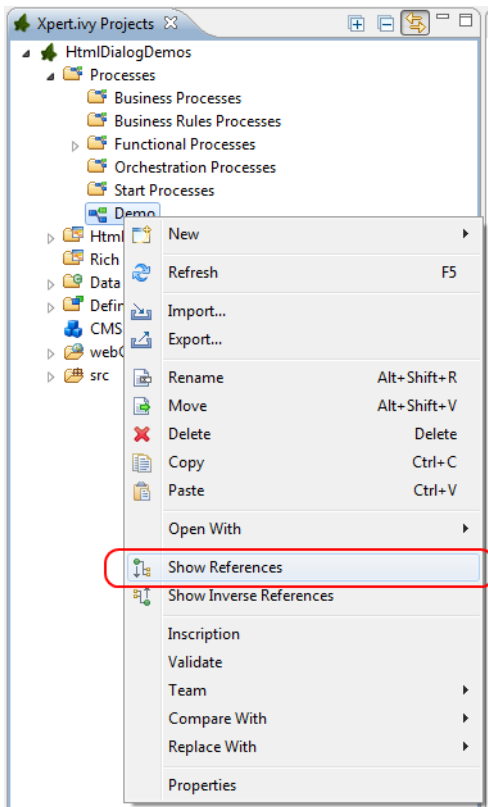


Figure 2.33. The Reference Menus

Features

The Reference view has the following functions:

- | | |
|-----------------------------|---|
| Refresh (🔄) | This function reloads the actual showed references. |
| Stop (⏹) | This function stops the calculation of references. |
| Show References (📄) | This option shows the references of the actual root object. |
| Show Inverse References (📄) | This option shows the inverse references of the actual root object. |

Simulating process models

This chapter deals with Axon.ivy debugging and simulation features. Processes, workflows, User Dialogs and changes on these should be tested before being deployed on an Axon.ivy production Engine. Therefore the Designer allows to simulate processes on your local computer, to debug it in depth and to inspect the execution history of all variable values. Hereby the process flow can be animated to visually observe the actual process execution sequence.

Simulation

A simulation can be started directly on the Request Start element or on the Designer Workflow UI Overview page displayed either in the browser view of the Process Development Perspective or in a separate browser window, depending on the setting in the corresponding preference. This Process Start Overview web page shows all processes that can be started by clicking on the link.

Also the Web Services are displayed on the Process Start Overview page. By clicking a Web Service Process link, the corresponding WSDL is displayed.

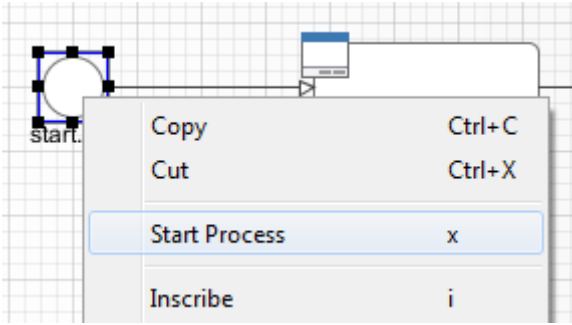


Figure 2.34. Start process on the Request Start element

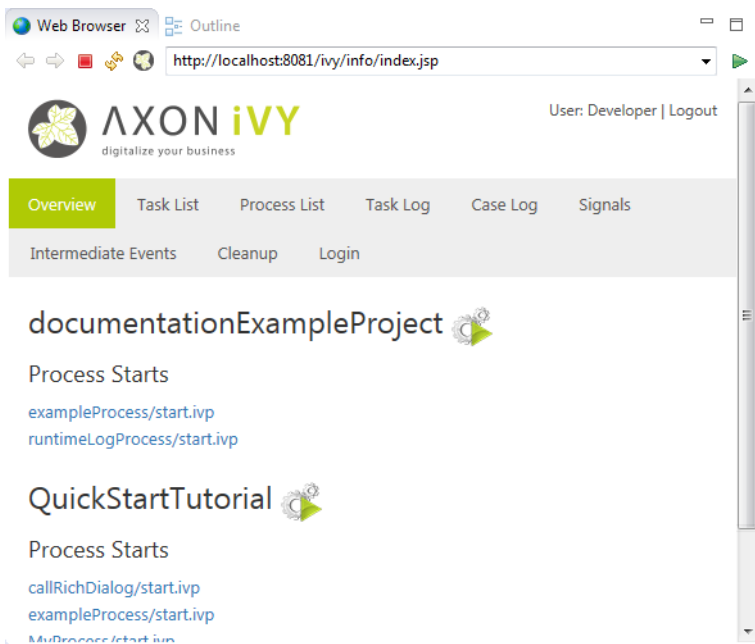


Figure 2.35. Designer Workflow UI Overview Page



Tip

You can switch off the simulation of Process Start Events and Intermediate Process Events when you want to simulate or test other parts of a projects. Just set the corresponding options in the preferences

Engine Actions

You are able to control the simulation and to influence the animation using the actions in the toolbar or the Axon.ivy menu.

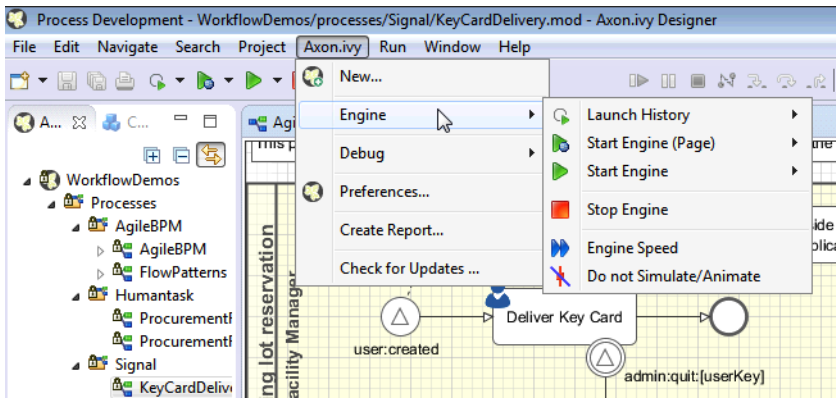












Figure 2.36. The Engine Sub-Menu

Starting the engine and show overview page	Select the entry  Start Engine (Page) in the menu or the button  in the toolbar to start the Simulation Engine, open the Process Development Perspective and refresh the Process Start Overview page.
Starting the engine	Select the entry  Start Engine in the menu or the button  in the toolbar to start the Simulation Engine and refresh the Process Start Overview page but without opening the Process Development Perspective.
Stopping the engine	Select the entry  Stop Engine in the menu or the button  in the toolbar to stop the Simulation Engine.
Adjust the engine animation speed	Select the entry  Engine Speed in the menu or the button  in the toolbar to show the slider to adjust the speed of the animation. This overwrites the corresponding setting in the preferences.
Suppressing the engine animation	Select the entry  Do not Simulate/Animate in the menu or the button  in the toolbar to switch the engine animation on and off. This overwrites the corresponding setting in the preferences.

Content and Formatting Language Settings

Overview

This dialog allows to edit the content language and the formatting language. The language settings are used at design time for displaying the Rich Dialogs in the Rich Dialog Editor. If option *Use settings while simulating* is checked the settings are also used while simulating.

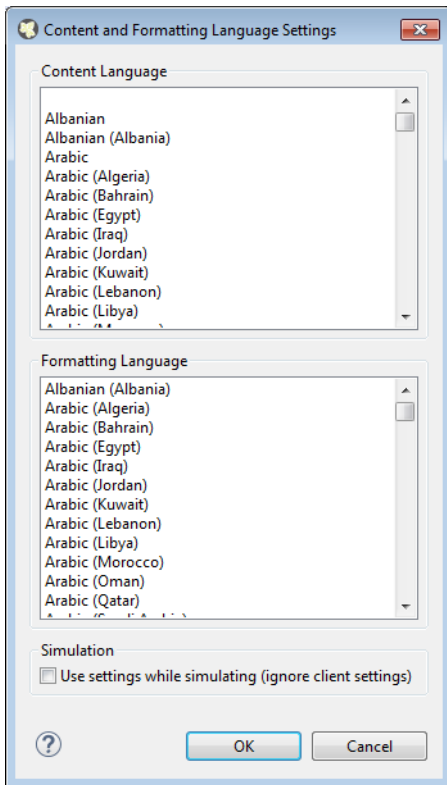



Figure 2.37. Content and Formatting Language Settings Dialog

Accessibility

Press  in the toolbar.

Settings

The following language settings can be configured:

Content Language	The content language is used to select the values of content objects.
Formatting Language	The formatting language is used when Objects are formatted using the <code>format()</code> method.
Use settings while simulating	If checked then the content and the formatting language settings will be used while simulating. If not checked then the settings of the client OS (for RIA) or the browser settings (for HTML) will be used.

How to use in IvyScript

To get or set the content or formatting language in IvyScript use `ivy.session.contentLocale` respectively `ivy.session.formattingLocale`.

Find out more about Axon.ivy's scripting language [here](#).

Breakpoints

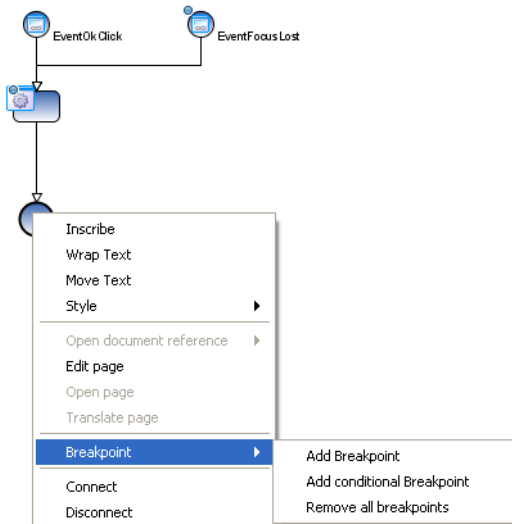
A breakpoint is a marker that tells the simulation engine to pause the execution. It is then possible to inspect the execution history, to view the values of internal variables and evaluate user defined expressions without being interfered by the running execution. The execution must be resumed by the user explicitly over the functionality of the Debug View. You can see a list of your breakpoints and edit them in the Breakpoint View.


Process Element Breakpoints

A process element breakpoint is a breakpoint that can be set on a process element. The execution of the process will be interrupted before the process element is executed.

Add / Remove a breakpoint

You can add process element breakpoints in a Process editor or User Dialog Logic editor window by using the popup menu. Right-click on the process step on which you intend to set the breakpoint and go to the *Breakpoint* sub-menu.



Adding a *conditional breakpoint* allows you to define an expression in a input box which must evaluate to true in order to suspend the execution. In the expression you have access to the `in` variable and all other variables in the context of the process step. As you can see in the figure above, process element breakpoints are visualized in the Process editor as a small filled dot at the border of the process step .

Data Class Attribute Value Change Breakpoints

A data class attribute value change breakpoint is a breakpoint that can be set on a data class attribute. The execution of the process will be interrupted before the value of the process data attribute is changed. Data class attribute value change breakpoints can be added or removed in the Data Class Editor or the Entity Class Editor. The current available variables and the current debug context is available in the Variables View. The old and new value of the debugging field is displayed in the variable debug.



Note

The breakpoint only breaks if the value of an attribute is changed by an IvyScript write attribute operation (e.g. `in.data.myField="Hello"`). If the attribute is changed by a setter method then the breakpoint will not break (e.g. `in.data.setMyField("Hello")`).

Debugger

The debugger provides a set of views and features to inspect the execution (including its history) of your processes and User Dialogs. Akin to a debugger in an Integrated Development Environment (IDE) such as Eclipse, NetBeans or VisualStudio it is possible to set breakpoints to pause an execution, to iterate through executions step-by-step and to examine the history and the current state of the execution in depth.

Debug View

The Debug view shows in a tree per open project all the currently handled requests i.e. all processes under execution in the simulation engine.

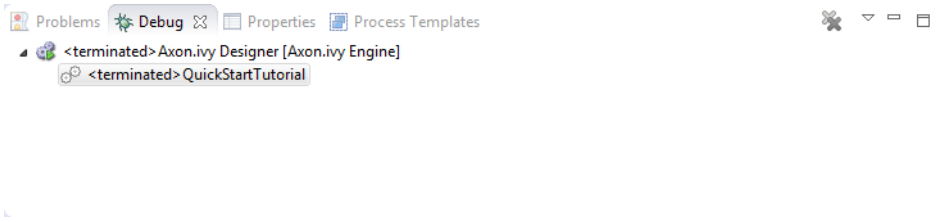


Figure 2.38. The Debug view in action

For each request to the engine the current state (i.e. the currently executed process step) are shown and can be manipulated individually with the following buttons on the toolbar:



Figure 2.39. Debugging Actions

Resume	Resumes the execution of the selected process/request until the end of the process to the next breakpoint
Terminate	Terminates the execution of the selected process/request
Step Into	This can be used to step into a (callable) process element. The current step is executed and then execution is suspended on the next step again.
Step Over	This can be used to step over a (callable) process element. The current step is executed and then execution is suspended on the next step in the current process.
Step Out	This can be used to step out of the current process, the execution is suspended again on the caller process element.

If you select a stack element then the process editor shows the process element that is executed at this stack element. Moreover, the Variable view will display the current values of the process data at the process element of the selected stack element.

History View

In this view you see the values of your process data (the `in` variable) during all runs of the currently selected process element in the process editor. The topmost tree entry shows the data of the first execution of the selected element during the first request whereas the entry at the bottom corresponds to the most current execution.

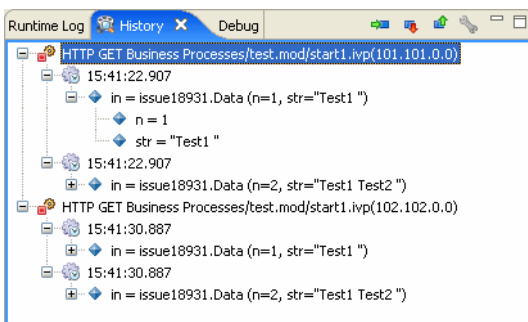


Figure 2.40. The History view in action

The following buttons on the toolbar can be used to navigate to process elements and to configure the history:

- Go to process element (📌) Marks the process element in the process editor whose history is currently displayed.
- Go to next process element (➡) Shows the history of the next process element.
- Go to previous process element (⬅) Shows the history of the previous process element.

History view preferences (🔗) Opens the preference page with the settings for the history.



Note

In case of memory shortage during simulation or due to history settings process data snapshots may be discarded. This is indicated by the message "history data no longer available".

Breakpoints View

This view lists all the breakpoint which are currently set and offers some functionality to edit and filter single breakpoints.

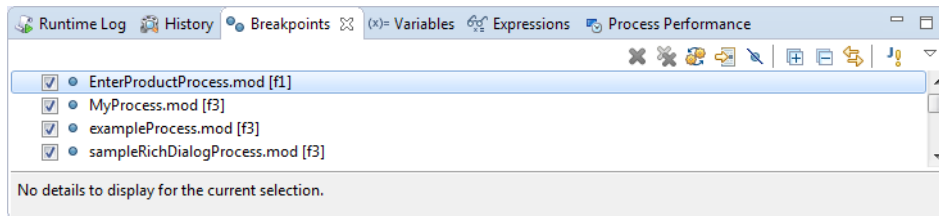


Figure 2.41. The Breakpoints view in action

You can configure and control the View with the toolbar and menu:

- Remove Breakpoints (✕ ✕) You can remove either the selected process(es) or all processes.
- Show Breakpoints Supported by Selected Target (🔗) Shows only the breakpoints in the list which are included in the process start under execution.
- Go to File for Breakpoint (📄) Opens an editor with the file containing the breakpoint or sets the focus on the corresponding editor window.
- Skip all Breakpoints (🔗) If set, all breakpoints are skipped.



Tip

This is helpful when you need to debug only some executions of a process steps. You can skip the breakpoints at the beginning and switch this button off, when the execution reaches the part you are interested in.

- Expand All / Collapse All (⊕ ⊖) If you have grouped the breakpoints together, you can quickly expand or collapse the whole tree
- Link with Debug View (🔗) Links this view together with the Debug View.
- Add Java Exception Breakpoint (⚠) Adds a breakpoint for a type of Java Exceptions, which will be used whenever this Java Exception is thrown throughout the execution.



Warning

Use this feature only if you are familiar with the Java programming language and its exception handling mechanism

- Toolbar Menu (☰) Here you can group the breakpoints according to some categories, select whether you want to restrict the view on a specific working set and set whether you want to see fully qualified names for breakpoints in Java code.

Variables View

This view shows a list of all variable in the context (or scope) of the currently executed process step. You are able to examine the structure, the types and the values of variables and it is even possible to change the values of variables which have a

simple data type (such as String, Number, Boolean, Time, Date or DateTime). The view is divided into a variable tree showing the structure, value and type of each variable (including its members) and a detail pane that displays the values for deeper examination.

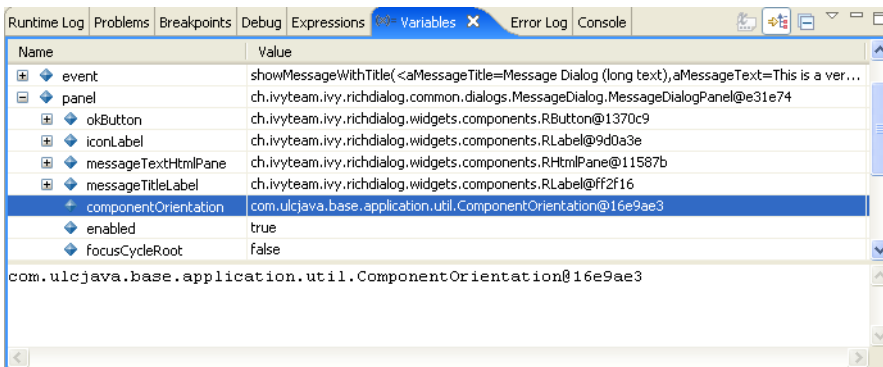

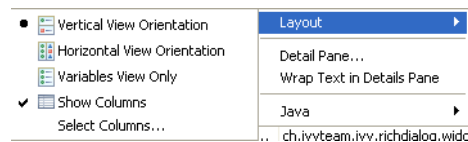


Figure 2.42. The Variables View in action

Collapse All ()

Collapse the whole variable tree to its root items.

Toolbar Menu ()



Layout

You can switch on and off the detail pane, set its orientation (vertical or horizontal) and set whether and which columns should be displayed.

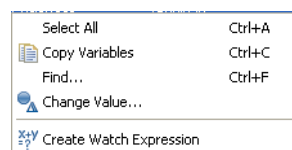
Detail pane

Setting for the size of the buffer for the detail pane, the higher the longer values you can examine (e.g. very long strings) but the more memory you use.

Wrap Text in Details Pane

Wrap text in details pane when it does not fit in the available space

Popup Menu



Select All

Selects all elements in the list.

Copy Variables

Copies all selected variables into the clipboard (e.g. for use in the Expressions view).

Find ...

Allows to find a specific variable with a filter string.

Change Value ...

The values of primitive Java data types may be changed here.

Create Watch Expression

Creates a new expression in the Expressions View.



Warning

Changing the value may cause exceptions or introduce undesired side effects with very weird behaviour in the continuation of the execution. Please use this feature with precaution!

Expressions View

In this view you can define expressions, evaluate them and examine their values (similar to the Variables view). In the expression you can use all valid IvyScript operators and language elements and at a certain point of time, only variables which are in the scope of the currently executed process step can be evaluated.

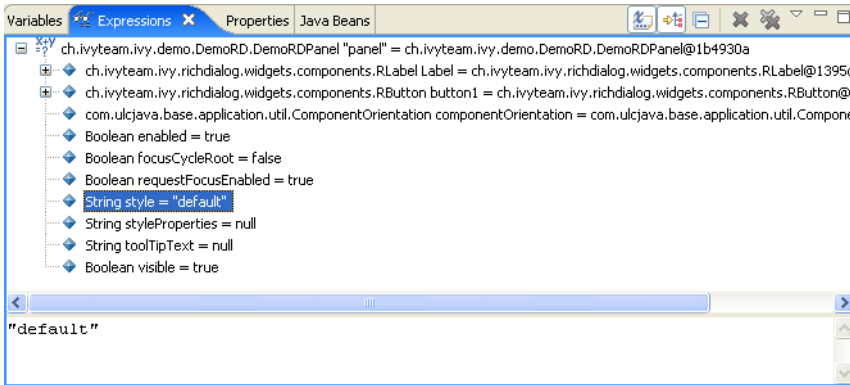





Figure 2.43. The Expressions View in action

Show Type Names ()

Shows the type names of the variables in the front of the variable.

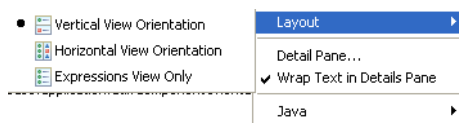
Collapse All ()

Collapse the whole expression tree to its root items.

Remove Selected Expressions /
Remove All Expressions ( )

You can remove either the selected or all expressions.

Toolbar Menu ()



Layout

You can switch on and off the detail pane, set its orientation (vertical or horizontal).

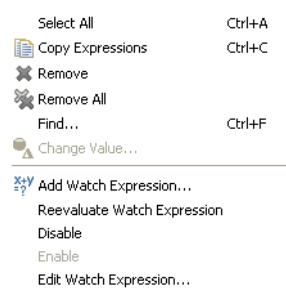
Detail pane

Setting for the size of the buffer for the detail pane, the higher the longer values you can examine (e.g. very long strings) but the more memory you use.

Wrap Text in Details Pane

Wrap text in details pane when it does not fit in the available space.

Popup Menu



Select All

Selects all elements in the list.

Copy Expressions

Copies all selected expressions and their state into the clipboard.

Find ...	Allows to find a specific variable with a filter string.
Add Watch Expression ...	Adds a watch expression into the expression view.
Reevaluate Watch Expression	Computes the current value of the expression (e.g. if expression reads data which was manipulated by concurrent threads).
Disable / Enable	Disables or enables the automatic evaluation of expressions when changes occur.
Edit Watch Expression ...	Edits the selected watch expression.

Runtime Log View

Overview

This section explains the Runtime Log view, and how it works.

The Runtime Log view displays a list of events. These events occur during the simulation. When you start the Axon.ivy process engine, this log view is opened by default and all entries are cleared.

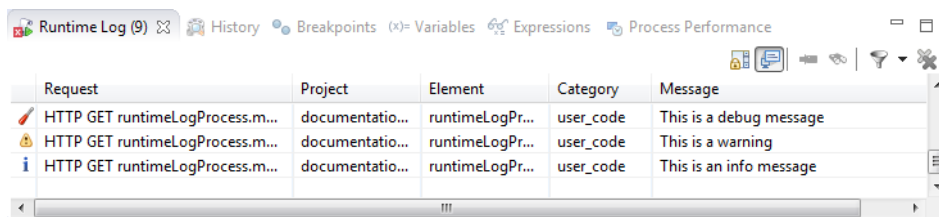


Figure 2.44. Runtime Log View: List of logged Events

Accessibility

Window > Show View > Runtime Log

Window > Show View > Other ... > Other... > Axon.ivy > Runtime Log

Columns

The following columns are displayed in the Runtime Log view:

First narrow column without name	In this column an icon is displayed that symbolizes the type of logged event (info / warning / error message)
Request	The request (HTTP, ULC, etc. with its ID) is displayed in which the log message occurred.
Project	The name of the project the log event was logged in.
Element	The identifier of the process element which logged the event.
Category	The log category refers to the Axon.ivy part which has logged the event (e.g. user_code: ivyScript by user; rich_dialog: at execution of RD; process: log from/during process model execution).

Message

The event message is displayed here.

Logged Event Details

When you double click on a log entry, a detail window will appear.

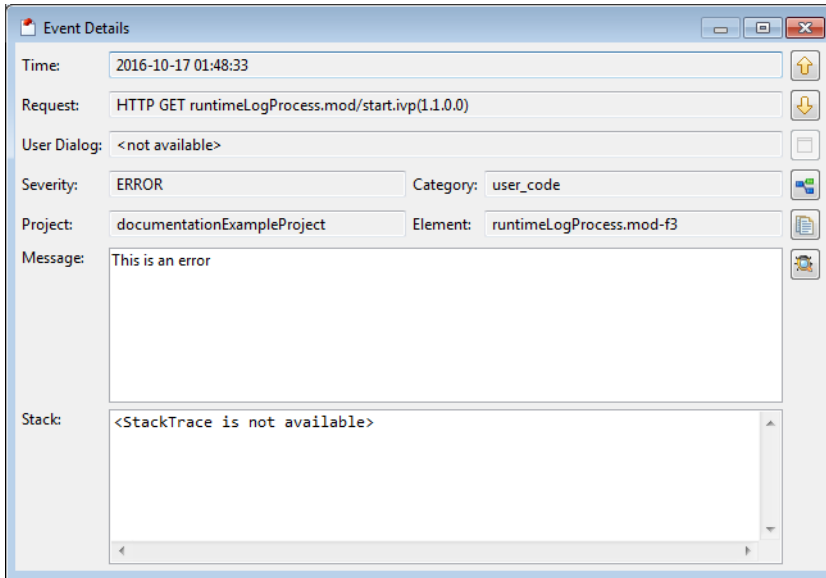


Figure 2.45. Runtime Log View Event Details

The following fields are displayed in this window:

Time Time, when the event was logged.

Request The request (HTTP, ULC, etc. with its ID) in which the log message occurred.

Rich Dialog The Rich Dialog in which context the event was logged.



Tip

You can jump to this rich dialog using the *Goto Rich Dialog* button on the right left hand side

Severity Shows how serious the logged event is (debug, info, warning, or error).

Project The name of the project the log event was logged.


Category The log category refers to the Axon.ivy part which has logged the event (e.g. user_code: ivyScript by user; rich_dialog: at execution of RD; process: log from/during process model execution).


Element The identifier of the process element which logged the event.





Message The log message is displayed here.

Stack If an exception was logged with the event, and it contains a stack trace (calling hierarchy), then it is displayed here.

On the right hand side the following buttons are located:

 Previous event Clicking on this button will open the previous event of the logged events list.

 Next event Clicking on this button will open the next event of the logged events list.

-  **Goto rich dialog** This button is available only if the log event contains Rich Dialog information. Clicking on this button opens a new editor showing the Rich Dialog which that has logged the event.
-  **Goto process element** If you click on this button a process is opened and the process element that has logged the event is selected.
-  **Copy event details to clipboard** If you click on this button all log event information are copied to clipboard.
-  **Save Error Report** If you click on this button an error report that contains information about the error, the designer machine and the current state of the Axon.ivy Designer.

How to log

This chapter describes how you can log to the runtime log.

Open any process elements that contain IvyScript (like: Step, Web Service, etc.) and type a script like the one you find in the figure below:



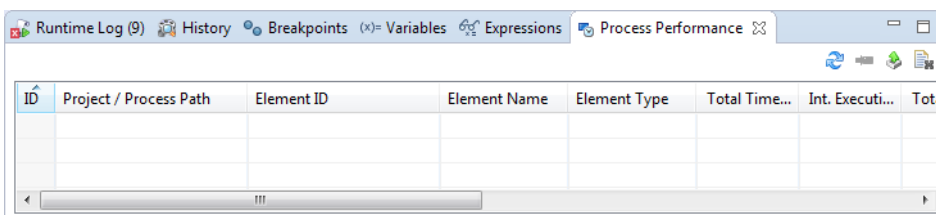
Figure 2.46. IvyScript to log into Runtime Log

Find out more about Axon.ivy scripting language here.

Process Performance View

Overview

The Process Performance View displays process performance statistics. This allows to analyse the performance and to detect long running and performance intensive process elements and processes. The view contains detailed information of each executed process element.



ID	Project / Process Path	Element ID	Element Name	Element Type	Total Time...	Int. Executi...	Tot

Figure 2.47. The Process Performance View



Note

On the Axon.ivy Engine there is the possibility to dump out performance statistics to a comma separated value file (*.csv). Check the Engine Guide for more information: *Monitoring > Process Element Performance Statistic and Analysis*

Accessibility

Window > Show View > Other... > Axon.ivy > Process Performance

Analyse the Performance Statistic

All time values are in milliseconds. The execution of some process elements are separated in two categories internal and external.

Internal Category The internal category is used for the execution time in the process engine itself without the external execution.

External Category The external category is used for execution time in external systems. In the table below the process elements are listed which use the external category.

Process Element	Internal Category	External Category
Database Step	Parameter-mapping, caching, output-mapping and ivyScript execution.	The execution of the SQL statement on the database server.
Web Service Call Step	Parameter-mapping, caching, output-mapping and ivyScript execution.	The execution of the Web Service on the web server.
E-Mail Step	Parameter-mapping	The interaction with the Mail-Server.
Program Interface		The execution of the defined Java-Class.

Table 2.4. Process elements with usage of external category

For each executed process element one entry in the view is created. See the table below which information is available.

Name	Description
Entry ID	Entry ID, useful to order the entries by its execution
Process Path	The path to the process.
Element ID	The identifier of the process element.
Element Name	The first line of the process element name (display name).
Element Type	The type of the process element.
Total Time	Total time [ms] of internal and external execution.
Int. Executions	Total internal executions of the process element.
Total Int. Time	Total internal time [ms] of process engine executions.
Min. Int. Time	Minimum internal process engine execution time [ms].
Avg. Int. Time	Average internal process engine execution time [ms].
Max. Int. Time	Maximum internal process engine execution time [ms].
Ext. Executions	Total external execution count.
Total Ext. Time	Total external execution time [ms].

Name	Description
Min. Ext. Time	Minimum external execution time [ms].
Avg. Ext. Time	Average external execution time [ms].
Max. Ext. Time	Maximum external execution time [ms].

Table 2.5. Column Description

Process Elements Reference

This chapter provides detailed explanations for the usage and configuration of all process element, for both business processes and Rich Dialog logic.

Axon.ivy provides a lot of useful process elements that can be used to define processes and the Rich Dialog Logic. These elements reside in the palette on the right edge of the Process editor and of the Rich Dialog editor. You can drag and drop the elements on the editor to use them in a specific process flow. You then can connect two elements by clicking on the source element, let the moused button pressed down, move the mouse cursor to the target element and finally release the mouse button. Reconnecting or removal is only possible by using the corresponding entries in the popup menu.

Every process element can be configured with its inscription mask. Open this mask by double-clicking on the element, use the popup-menu or press the **i**-key whenever the element is selected. The inscription mask is divided into multiple tabs and the order of the tabs indicates the sequence of processing. For example in the figure below, the Output mapping (the second tab in the middle) is performed before the code in the third tab is executed.

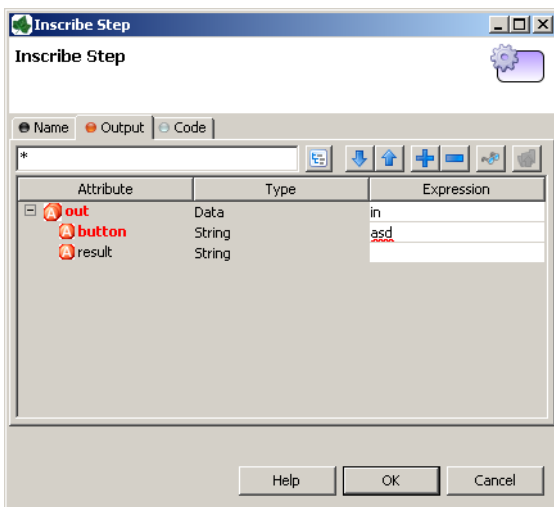





Figure 2.48. An exemplary inscription mask

The icons on the tab indicate their state:

-  the tab is empty
-  the tab has been changed by the user (default assignments are not considered as user entries)
-  the tab contains errors

Common Tabs

This section describes the most common tabs that are used on more than one element inscription mask.

Name Tab

This tab is included in the inscription mask of all process elements and contains the name and a description of the element.

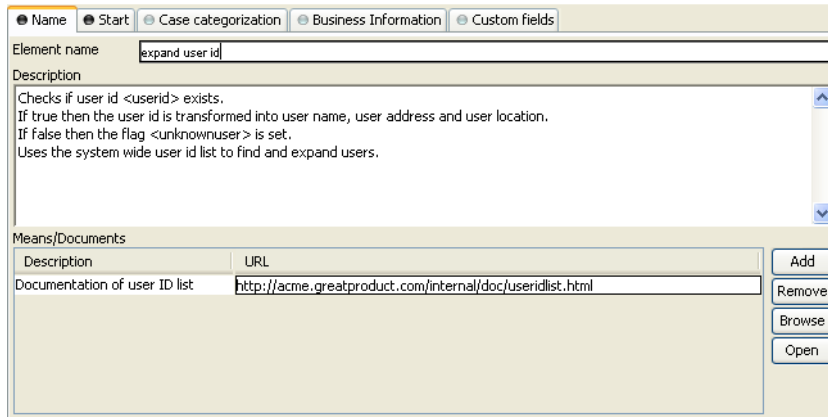


Figure 2.49. The Name tab

Element Name Name of the element in the process.

The name will be displayed in the process editor. Various font format options can be chosen on the popup menu of selected text.



Tip

Give each element a name, always. This increases the overview and simplifies the exchange of models between you and your colleagues. If you work in a team, the use of naming conventions are strongly recommended.

Description Describes the function of the element.

This text appears as tool tip in the process editor whenever the mouse stays over the element.

Means/Documents A list with references to additional stuff that is related to this process step, i.e. documentation, templates, example forms and many more.



Tip

In generated HTML reports, a link is inserted for these document references.

Output Tab

On this tab you can set all values in the output Data Class. By default the output variable is mapped directly to the input variable, but the user can overwrite either the assignment of the whole output Data Class or only of single members thereof.



Note

In Axon.ivy input and output of process or Rich Dialog logic elements are always set to the corresponding data class, i.e. in a Rich Dialog logic element it is the Rich Dialog Data Class and in a process element it is the project Data Class (or the one which was assigned to the process).

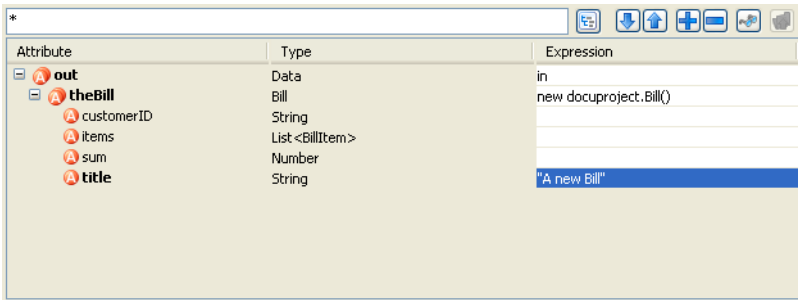


Figure 2.50. The Output tab

- Filter Box / Toolbar** In the upper part you can set a string based filter with wild cards in the text box, set a filter to only show rows with an assigned value (🔍), move between the rows with an assigned value (⬇️⬆️), expand/collapse the list (⊕⊖) and set the visibility level (🔍)
- Output Tree** Here you can see the whole structure of the output variable including each of its members with the assigned values/expressions. You may use the Attribute Browser and the Function Browser to construct the expressions.

Code Tab

The code tab is part of almost each inscription mask and allows the user to define the semantics (behaviour) of the corresponding process element with the built-in Axon.ivy scripting language.

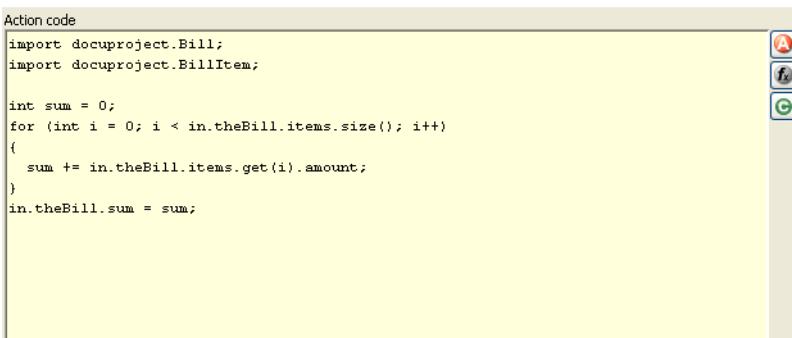


Figure 2.51. The Code Tab

- Code Editor** You can write Ivyscript code snippets in the part with yellow background. The editor supports code completion and error highlighting. If the background color changes to red, the code contains an error.



Tip

For more information about Ivyscript, see [Ivyscript](#)

- Attribute Browser** Here you have access to the local process data in the scope of the element such as the in- and out-variables, the Rich Dialog panel (if this is inside the Rich Dialog logic) and other parameters. [Click here for more information.](#)
- Function Browser** Here you have access to some of the most important mathematical functions and to the whole environment of the process such as the request and response properties, the application the process belongs to and many more. [Click here for more information.](#)

Data Class Browser

Here you have access to all data classes in the scope of the process element. This includes the built-in Ivy data types such as `String`, `Number`, `DateTime` or even `List`. Click here for more information.

Panel Tab

Here it is possible to access and manipulate the Rich Dialog panel and its members.

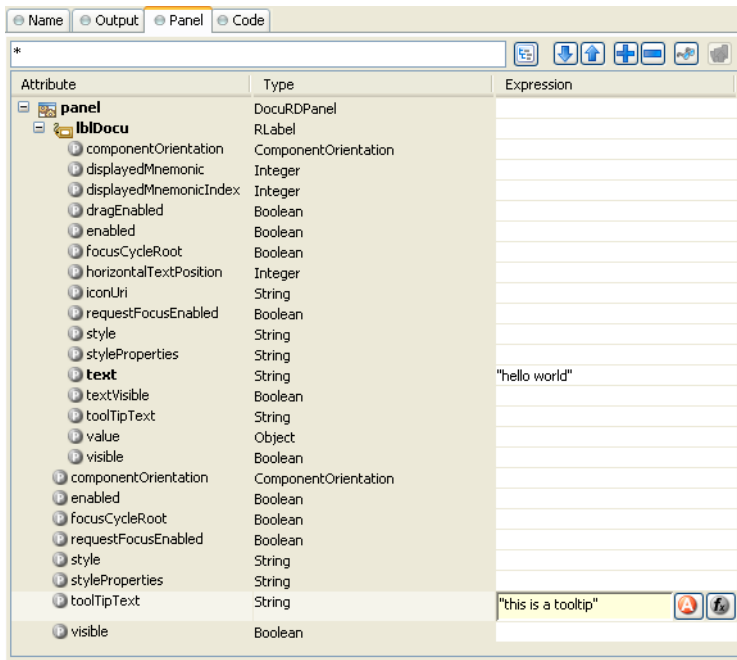


Figure 2.52. The Panel tab

Filter Box / Toolbar

In the upper part you can filter the widgets and attributes by name using wild cards in the text box, set a filter to display only rows with an assigned value (🔍), move between rows with an assigned value (⬇️⬆️), expand/collapse the list (⊕⊖) and set the visibility level (👁️).

Panel Tree

In the tree you can expand the whole panel hierarchy and manipulate the values of its members.

Data Cache Tab

Process steps that basically encapsulate a read access to master data (data that is changed very rarely) load their data each and every time they are executed even though this is not necessary. With data caches you can first store the values of such steps into a cache in the memory and later read it much faster from the cache the next time you encounter the same step.

Cache entries depend from the active environment and are always bound to their scope - either the current user session or to the running application. Furthermore, they are arranged into groups to make the manipulation for similar entries easier. Cache entries and groups are bound to the life cycle of their scope.

Both groups and entries can be invalidated after a configurable period of time, at a specific time of the day or on request.

For more information about this topic, please refer to the Data Caching section.

Caching Mode
 Do Not Cache Cache Invalidate Cache

Cache Configuration
 Scope: Session

Group Name: "Group" + in.customerID

Group Invalidation: None Fixed Time Lifetime '22:00'

Entry Name: "Entry" + in.orderID

Entry Invalidation: None Fixed Time Lifetime 100

Figure 2.53. The Data Cache tab

The tab data cache is included in the inscription mask of the Web Service and the DB element. It features the following entries:

Do Not Cache	Does not use the data caching mechanism at all, the element is executed normally. This is the default setting for all elements.
Cache	Uses the data caching mechanism to execute the element. First the whole data cache is searched for the entry described below in the <i>Info</i> part. If found, the cached value is returned and the execution of the element ends. If not found, the element is executed normally, but in the end the result is stored in the data cache.
Invalidate Cache	Invalidates the data cache entry specified in the <i>Info</i> part. Use this when your element performs a write operation that changes data which is cached. The step is executed normally, but in addition the specified data cache entry is invalidated.
Scope	Either <i>Session</i> , <i>Environment</i> or <i>Application</i> . With <i>Session</i> scope, the cache entry is linked to the currently logged in user (i.e. is specific for each user and is invalidated when the user logs out). With <i>Environment</i> scope the cache entry is linked to current environment. With <i>Application</i> scope, the cache entry is linked to the Application.



Warning

Use caches sparingly and with precaution! If you cache results from process steps with huge results (in terms of memory usage), your memory can fill up very fast. This can even get worse if you frequently use the *Session* scope and the result is cached multiple times (once for each session i.e. user)

Group Name	The name for the group to which the entry will be added. The group resides under the scope, so it is possible to have two groups with the same name, one for both scopes. Use always the same group name if you want to put multiple data cache entries in different process steps into the same group.
Group Invalidation	You can choose between no invalidation, an invalidation at a fixed time or a lifetime based invalidation. If you set a fixed time, then the cache is invalidated once a day at the chosen time. For a lifetime based invalidation, you can set for how many seconds the cache group should be valid after it's initial creation. After the expiry of the lifetime the whole group and all entries in the group are invalidated.
Entry Name	The name for the entry. This name must be unique within the group but you are allowed to have multiple entries with the same name in different groups. Use always the same entry names (as well for the group) if you want to use the same data cache entry in multiple process steps.
Entry Invalidation	You can choose between no invalidation, an invalidation at a fixed time or a lifetime based invalidation. If you set a fixed time, then the cache entry is invalidated once a day at the chosen time. For a lifetime based invalidation, you can set for how many seconds the cache entry should be valid after its initial creation. After the expiry of the lifetime the entry is invalidated.

Start



The **Start** (Request Start) element is located in the *Event & Gateway* drawer of the Process editor palette.

Element Details

The Request Start element is the start point of a workflow. A workflow contains one case and at least one task. Each start of a Request Start creates a new case and task.

There are two ways to start a new workflow:

- Request

Most workflows are started with a HTTP request. The start Links can be found on the Process Start Overview HTML page and can be placed on an external web sites or as shortcut on the desktop.

The public API provides also a ways to get a list of all request starts for custom start lists or own implemented start mechanism:

```
ivy.session.getStartableProcessStarts()
```

The HTTP request start can be configured on the Request Tab

- Triggered

The second way to start a new workflow is by a Trigger Element. On call, it creates a new case and a new task to the Request Start with the defined configurations (and parameters). This offers a simple way to create several workflows inside a other workflow.

The trigger start can be configured on the Trigger Tab

These two start types could be enabled or disabled separately.

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Start Tab

The start tab defines the name and the parameters to start the process. The signature is a definition of the name with the parameter types and its order. Elements like Call Sub or Trigger referenced to this signature.

Signature	Displays the current signature. Namespaces of the parameter types are not displayed, but they are still a part of the signature, that identifies a start uniquely.
Name	Signature name is case sensitive and can only contain letters (a-Z), numbers (0-9) and underscores (_).
Definition of input parameters	Defines the input parameter of the interface. The type of the parameters and its order is used for the signature. Changing the order or the type, changes also the signature. All referenced elements has to be updated.
Mapping of input parameters to process data	This section is to mapping the incoming parameters to the internal process data. The parameters are available as fields on a <i>param</i> variable.



Note

The reason you have to assign the incoming parameters to local data is to keep the internal implementation independent from the signature declaration. The mapping of parameters serves as a flexible adapter mechanism. The implementation can be changed (rename data, use different data types, etc.) without changing the signature. That way none of the clients of the Process have to be changed as long as only the implementation changes and the signature stays.



Note

Only the defined input parameter on the signature can be assigned to the process data. The internal process data is hidden and encapsulated from the environment. This makes it impossible to inject unintended, insecure data into the process data.



Note

To submit parameters with a HTTP-Request you can simply add them to the URL.

If you have for example defined a parameter named *myParameter* in the signature, append `?myParameter=hallo` to the URL to pass the value *hallo* to the parameter *myParameter*.

If you want to pass values for multiple parameters the following parameter need an `&` instead of an `?`. For example: `?param1=value1¶m2=value2¶m3=value3`



Tip

You may already specify the type of the parameter here by adding a colon ':' to the parameter name, followed by desired type (e.g. **myDateParameter:Date**).

myDate:Date	String
myDate	Date

Request Tab

This tab contains the configuration for the HTTP-Request start. Name and description which is displayed on the start list. The required permissions to start the process, and the workflow mode.

Enablement	If <i>Yes, this start can be started with a HTTP-request/-link</i> is checked, the HTTP request mechanism for this start element is enabled. Otherwise it is not possible to start the request start with a HTTP request.
Start Link HREF (.ivp)	Contains the name of the Process Start link. Notice that this link always has to end on <i>.ivp</i> . This is required for proper association of the request by the web server. Important: this name has to be unique within its process.
Show in Start List	Defines whether this process should appear in the start list of the Process Start Overview HTML page or not.
Name	Defines the display name of the process start in the start list.
Description	Sets a description of the process start. It is displayed in the start list of the Process Start Overview HTML page.
Responsible Role	Users which want to start the process must be assigned to this role.

**Tip**

In the Designer you can create test users and assign them the role to test this element, on the Axon.ivy Engine you must create the real users separately (roles can be imported from the designer).

Only Wf Users

Limits the process to users that are registered in the Axon.ivy Engine as users. If the box has not been checked also anonymous users (which own the *Everybody* role by default) may start the process.

Role Violation error

The selected exception element is thrown when the user lacks the required role. The error can be handled by a catching “Error Start”.

Persist task on creation

If selected, the case and task are directly persistent on request start. The Case state will be *RUNNING* instantly (skipping the state *CREATED*) and the Task state will be *RESUMED* (skipping the state *CREATED*). The task could also be reset after the *Start Request* and the next *Task Switch* or *Process End*.

This option only available when the option *Only WF User* is activated.

**Note**

If the option is selected, the task could be reseted in the process (with `task.reset()`). This will reset the process data and the current user got the task to his task list.

**Note**

When a session timeout occurs, `task.reset()` is called automatically on the task. Thereby the user has the task again in his task list.

**Tip**

Usually only processes including at least one Task Switch element has this option selected. Because per default a new task is in state *CREATED* until the task become persisted. If the option is selected, the task get directly into the state *RESUMED*. With this behaviour is now possible to distinguish tasks which have a Task Switch element in their process and others without one. This helps to separate tasks in the task overview from workflow starts (with different steps) and simple process starts (which e.g. only outputs some informations).

Trigger Tab

This tab holds definitions for starting this workflow with a Trigger Element.

Enablement

If *Yes, this start can be started with a trigger element* is checked, the trigger mechanism for this start element is enabled. Otherwise it is not possible to choose the *Request Start* element in a Trigger Element.

**Note**

When an already related Trigger Element links to a disabled start, this will not prohibit the execution at runtime. An error is logged to the log file and the process starts with its defined configuration.

Responsible Role / User

Defines the role or user required to carry out the task created with the Trigger Element.

Use *Role from Attr.* or *User from Attr.*, if the role or user must be set dynamically during process execution. For example from a parameter set by the Trigger Element.

Use *Role*, if you know the responsible role when editing the element.

The role *SYSTEM* means that no human intervention is required. The system executes the task automatically.

The role *CREATOR* means that the user who has started the current case is responsible for the task created by the Trigger Element.

Delay (blocking period)

The task can be blocked before a user can work on it. This ivyScript expression defines the Duration the task is blocked.

Case

Define whether the triggered case should be attached to the same “Business Case” as the triggering case.

Task Tab

This tab defines information relevant to the task. Only tasks created with the Trigger Element (see Trigger Tab) will appear in the task list as suspended tasks. Tasks started with a HTTP request (see Request Tab) normally do not appear in the task list.

Entry in Task List

Defines the name and description of the task that appear in the task list of the assigned role or user.

Priority

Here you select the priority of the task. In the task list you can filter the tasks according the priority.

Expiry

An ivyScript expression defines the Duration until the task will expire. If that happens the escalation procedure defined by the following parameters is executed.

- Exception: Starts an exception process if defined. >> *Ignore Exception* will not start an exception process.
- Responsible Role / User after expiry: Defines the Role / User to reassign the task to.
- Priority after expiry: Defines the new Priority of the task after it has expired.

**Note**

A task created with a HTTP request (see Request Tab) is executed immediately. Defining Expiry Timeout makes only sense in combination when starting with a Trigger Element (see Trigger Tab)

Tab Task - Business

This tab allows to set additional information to categorize the task created. The values set on this tab are only informational and have no effect on how this task is treated by Axon.ivy.

Business calendar

You can set the name of the business calendar that should be used for this task. In the context of this task `ivy.cal` will return this business calendar regardless of what you've set for the case or environment.

For more information about business calendar administration see the engine guide.

For more information about business calendar usage see the Public API of `ch.ivyteam.ivy.application.calendar.IDefaultBusinessCalendar`.

Tab Task - Custom fields

This tab allows to set additional information for the task created. The values set on this tab are only informational and have no effect on how this task is treated by Axon.ivy.

Case Tab

On this tab you can configure the Case created by this Start Request. See “Case Tab” in the Task Switch Gateway element.

Program Start

Program Start

The *Program Start* element is located in the *Event & Gateway* drawer of the process editor palette.

Element Details

The program start element allows to start a process by a trigger from embedded external Java code. This opens a possibility to integrate an Axon.ivy application into other applications and systems. The program start element will instantiate a Java class that must implement the `IProcessStartEventBean` interface. The Java class can then start the process by calling the method `fireProcessStartEventRequest` on the Axon.ivy runtime engine `IProcessStartEventBeanRuntime`. The common way to implement a Start Event Bean is to extend the abstract base class `AbstractProcessStartEventBean`. The interface also includes an inner editor class to parametrize the bean. You will find the documentation of the interfaces and abstract class in the Java Doc of the Axon.ivy Public API.

How Process Start Event Beans work on an Axon.ivy Engine Enterprise Edition

An Axon.ivy Engine Enterprise Edition consists of multiple engine instances (nodes) that are running on different machines.

Normally process start event beans are instantiated on every node but only started on the master node. This guarantees that for each *Program Start* process element only one bean is running, no matter what the total number of nodes in the Engine Enterprise Edition is.

However, if you need your process start event bean to be started on all cluster nodes, you may instruct the engine to do so. Just have your bean class implement the (empty) marker interface `IMultiNodeCapable` and the above restriction will no longer apply.

Please be aware of the fact that having multiple running instances of the same bean may lead to race conditions.

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Tab Start

On this tab you define the Java class to execute.

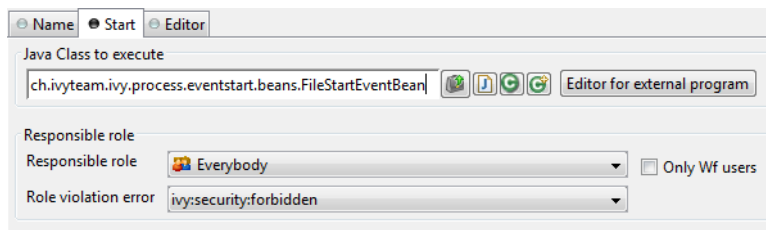


Figure 2.54. The Start tab

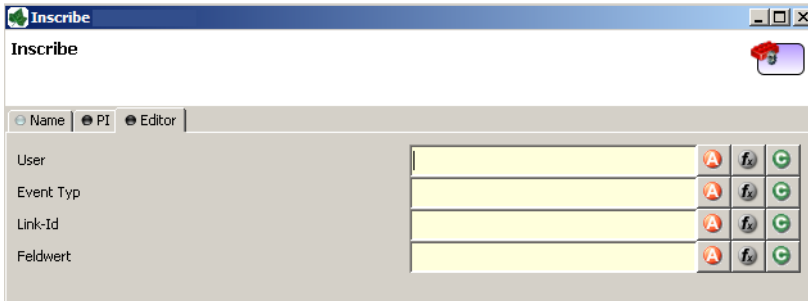
Java Class to execute	Full qualified name of the Java class that implements the <code>IProcessStartEventBean</code> interface. Use the New Bean Class Wizard (🔍) to create a new Java source file with an example implementation of the bean class.
Responsible role	Defines the role that is required to be able to start a process. The bean will set up an authorised session that calls the <code>fireProcessStartEventRequest()</code> from the <code>eventRuntime</code> to trigger a process.

Tab Editor

This tab displays the editor, that can be integrated in the external Java bean of the process element. The editor is implemented as an inner public static class of the Java bean class and must have the name `Editor`. Additionally the editor class must implement the `IProcessExtensionConfigurationEditorEx` interface. The common way to implement the editor class is to extend the abstract base class `AbstractProcessExtensionConfigurationEditor` and to override the methods `createEditorPanelContent`, `loadUiDataFromConfiguration` and

saveUiDataToConfiguration. The method createEditorPanelContent can be used to build the UI components of the editor. You can add any AWT/Swing component to the given editorPanel parameter. With the given editorEnvironment parameter, which is of the type IProcessExtensionConfigurationEditorEnvironment, you can create text fields that support ivyScript and has smart buttons which provide access to the process data, environment functions and Java classes.

Here is an example of such an editor:



As you can see, the editor provides you access to any process relevant data, which can be used by your own process elements. For instance, you can easily transfer process data to your legacy system.

The following part shows the implementation of the editor shown above. As mentioned above Axon.ivy provides the IIvyScriptEditor which represents a text field with ivyScript support and smart buttons. Inside createEditorPanelContent use the method createIvyScriptEditor from the editorEnvironment parameter to create an instance of such an editor. Use the loadUiDataFromConfiguration method to read the bean configuration and set them to the UI components. Inside this method you can use the methods getBeanConfiguration or getBeanConfigurationProperty to read the bean configuration. Use the method saveUiDataToConfiguration to save the data in the UI components to the bean configuration. Inside this method you can use the methods setBeanConfiguration or setBeanConfigurationProperty to save the bean configuration.

```
public static class Editor extends AbstractProcessExtensionConfigurationEditor
{
    private IIvyScriptEditor editorUser;
    private IIvyScriptEditor editorEventTyp;
    private IIvyScriptEditor editorLinkId;
    private IIvyScriptEditor editorFieldValue;

    @Override
    protected void createEditorPanelContent(Container editorPanel,
        IProcessExtensionConfigurationEditorEnvironment editorEnvironment)
    {
        editorPanel.setLayout(new GridLayout(4,2));
        editorUser = editorEnvironment.createIvyScriptEditor(null,null, "String");
        editorEventTyp = editorEnvironment.createIvyScriptEditor(null,null, "String");
        editorLinkId = editorEnvironment.createIvyScriptEditor(null, null, "String");
        editorFieldValue = editorEnvironment.createIvyScriptEditor(null, null);

        editorPanel.add(new JLabel("User"));
        editorPanel.add(editorUser.getComponent());
        editorPanel.add(new JLabel("Event Typ"));
        editorPanel.add(editorEventTyp.getComponent());
        editorPanel.add(new JLabel("Link-Id"));
        editorPanel.add(editorLinkId.getComponent());
        editorPanel.add(new JLabel("Feldwert"));
        editorPanel.add(editorFieldValue.getComponent());
    }

    @Override
    protected void loadUiDataFromConfiguration()
    {

```

```

editorUser.setText(getBeanConfigurationProperty("User"));
editorEventTyp.setText(getBeanConfigurationProperty("EventTyp"));
editorLinkId.setText(getBeanConfigurationProperty("LinkId"));
editorFieldValue.setText(getBeanConfigurationProperty("Feldwert"));
}

@Override
protected boolean saveUiDataToConfiguration()
{
    setBeanConfigurationProperty("User", editorUser.getText());
    setBeanConfigurationProperty("EventTyp", editorEventTyp.getText());
    setBeanConfigurationProperty("LinkId", editorLinkId.getText());
    setBeanConfigurationProperty("Feldwert", editorFieldValue.getText());
    return true;
}
}

```

At runtime you have to evaluate the IvyScript the user have entered into the ivy script editors. If you implement for example the `AbstractUserProcessExtension` class there is a `perform` method which is executed at runtime. At this point you want to access the configured data in the editor. The following code snippet show how you can evaluate the value of an `IIvyScriptEditor`. If you use the `IIvyScriptEditor` you only get the value by calling the `executeIvyScript` method of the `AbstractUserProcessExtension`.

```

public CompositeObject perform(IRequestId requestId, CompositeObject in,
    IIvyScriptContext context) throws Exception
{
    IIvyScriptContext ownContext;
    CompositeObject out;
    out = in.clone();
    ownContext = createOwnContext(context);

    String eventtyp = "";
    String linkId = "";
    String fieldValue = "";
    String user= "";

    user = (String)executeIvyScript(ownContext, getConfigurationProperty("User"));
    eventtyp = (String)executeIvyScript(ownContext, getConfigurationProperty("Event Typ"));
    linkId = (String)executeIvyScript(ownContext, getConfigurationProperty("Link-Id"));
    fieldValue = (String)executeIvyScript(ownContext, getConfigurationProperty("Feldwert"));

    // add your call here

    return out;
}

```

Error Boundary Event



The *Error Boundary Event* can be attached to any activity by using its context menu.

Element Details

The execution of an activity can be aborted with an error. The execution is then redirected to this Error Boundary Event element and continued from there. Within the follow up flow the process can handle the error by executing compensation steps or user activities.

See the Error Handling concept for sample use cases.

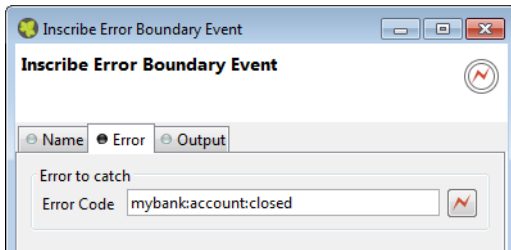
Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Error Tab

On this tab you can configure the Error Code that the Boundary Event will catch.



Output Tab

On this tab you can configure the output of the element (i.e. the data that leaves the element). See Output Tab for a more detailed description.



Note

Additionally to the regular variables of the *Output Tab* you have the following variable available:

error References the occurred `BpmError`. Gives access to the occurred Error Code, Cause and CallStack.

Error Start



The *Error Start Event* element is located in the *Event & Gateway* drawer of the process editor palette.

Element Details

The execution of a process element can be aborted with an error. The execution can continue at an Error Start Event which handles the occurred error. Within the follow up flow the process can handle the error by executing compensation steps or user activities.

See the Error Handling concept for sample use cases.

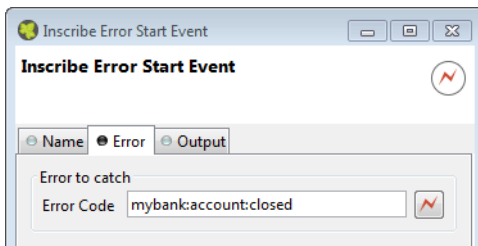
Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Error Tab

On this tab you can configure the Error Code that the Error Start Event will catch.



Output Tab

On this tab you can configure the output of the element (i.e. the data that leaves the element). See Output Tab for a more detailed description.

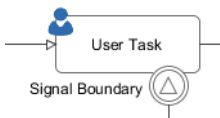


Note

Additionally to the regular variables of the *Output Tab* you have the following variable available:

error References the occurred `BpmError`. Gives access to the occurred Error Code, Cause and CallStack.

Signal Boundary Event



The *Signal Boundary Event* can be attached to a *User Task* by using its context menu.

Element Details

A Signal Boundary Event destroys an open task of a user if a signal code is received that matches the inscribed pattern.

See the Signal concept for sample use cases.

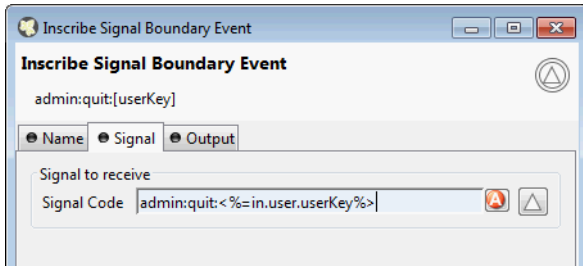
Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Signal Tab

On this tab you configure the pattern that the Boundary Event will listen to.



Output Tab

On this tab you can configure the output of the element (i.e. the data that leaves the element). See [Output Tab](#) for a more detailed description.



Note

Additionally to the regular variables of the *Output Tab* you have the following variable available:

signal Gives access to the signal event.

Signal Start



Signal Start

The *Signal Start Event* element is located in the *Event & Gateway* drawer of the process editor palette.

Element Details

The Signal Start Event listens to a signal. It starts a new process when a signal with a matching signal code has been received.

See the [Signal](#) concept for sample use cases.

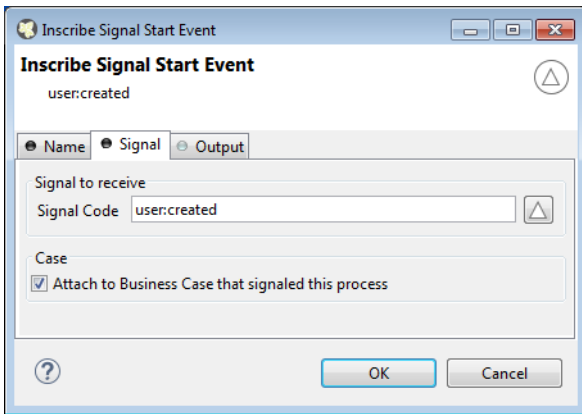
Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See [Name Tab](#) for a more detailed description.

Signal Tab

On this tab you configure the Signal Code that the Signal Start Event will listen to.



Case Define whether the triggered case should be attached to the same “Business Case” as the signaling case.

Output Tab

On this tab you can configure the output of the element (i.e. the data that leaves the element). See Output Tab for a more detailed description.



Note

Additionally to the regular variables of the *Output Tab* you have the following variable available:

signal Gives access to the signal event.

Alternative



Alternative

The *Alternative Gateway* element is located in the *Event & Gateway* drawer of the process editor palette.

Element Details

An Alternative is a switch that connects the process flow to one of the exits of the element depending on the evaluation of the exit conditions. So, you can use this element to perform business rules (in the form of *if - else if - else* decisions) or to build loops in the process flow.

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Tab Condition>

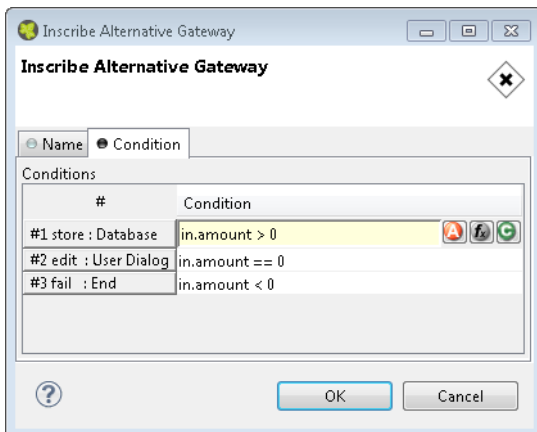


Figure 2.55. The Condition tab

Conditions Each row in this table is assigned to an exit of this element. In the column **Condition** boolean expressions must be entered. The conditions are evaluated from the top to the bottom and the exit of the first one that evaluates to *true* is chosen, i.e. the process will proceed by the path connected to this exit.

Split



Split

The *Split Gateway* element is located in the *Event & Gateway* drawer of the process editor palette.

Element Details

This allows you to model parallel process paths. The element may split one process path into a number of paths that are executed concurrently. The input process data is copied to all the parallel executions but can be manipulated in each path individually.



Note

Use this element always together with the Join element.



Tip

Use the split and the join element to execute parallel database or web service requests.



Warning

Do not put any user interaction (Rich Dialog, HTML Page) or task element within a splitted process path.

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Output Tab

On this tab you can configure the output of the element (i.e. the data that leaves the element). See Output Tab for a more detailed description.



Note

For each outgoing connection you have a separate **outX** object available which carries the data of the Xth output. Hover with the mouse over the outgoing connections of the element to see which output connection corresponds to which variable.

Code Tab

On this tab you can execute any IvyScript, e.g. define output data of this element. See code tab for a more detailed description.



Tip

The entered code will be executed *after* the execution of the *output tab*. Although this may seem a bit counter-intuitive at first, you should simply regard the code tab as an alternative way of defining output data. The general recommendation is to use the output table to define simple assignments and the code tab if more extensive scripting is needed to calculate data.

Join



Join

The *Join Gateway* element is located in the *Event & Gateway* drawer of the process editor palette.

Element Details

This element synchronizes and joins a parallel execution of process paths together. The output of each incoming path is copied into the element and can be used to define its output.



Note

This elements waits until all of the parallel incoming process paths have finished and reached this element.



Tip

Use the Split element to create parallel process paths.

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Output Tab

On this tab you can configure the output of the element (i.e. the data that leaves the element). See Output Tab for a more detailed description.



Note

For each incoming connection you have a separate **inX** object available which carries the data of the Xth input. Hover with the mouse over the incoming connections of the element to see which input connection corresponds to which variable.

Code Tab

On this tab you can execute any IvyScript, e.g. define output data of this element. See Code Tab for a more detailed description.



Tip

The entered code will be executed *after* the execution of the *output tab*. Although this may seem a bit counter-intuitive at first, you should simply regard the code tab as an alternative way of defining output data. The general recommendation is to use the output table to define simple assignments and the code tab if more extensive scripting is needed to calculate data.

Task Switch Gateway



Tasks

The *Task Switch Gateway* element is located in the *Event & Gateway* drawer of the process editor palette.

Element Details

With the task switch element a process is segmented into tasks. It interrupts the execution of a process and allows another user to proceed. The actual process state is stored in the system database. A role or user is assigned that is able to pick up and start the task.

When the role SYSTEM has been chosen the process is executed by the system, without manual intervention by a user.



Note

If any error occurs during the execution of a task that is executed by the system the task is rolled back and its state set to error. After a certain time the task is resumed and the system tries again to execute it.

The duration until a task with state error is resumed depends on the times the task had failed before (1, 2, 4, 8 minutes, ... up to 23 hours).

This is a default behaviour. To change it consult the documentation of `ch.ivyteam.ivy.workflow.SystemTaskFailureBehaviour`



Warning

The Task Switch Gateway element can have several input and output arcs and acts as an AND-Gateway. It synchronizes all incoming tasks - it waits until all incoming tasks have been completed.

For each outgoing arc the Task Switch Gateway element creates a parallel task.



Warning

The process state that is stored to the system database contains all process data values that are stored in persistent fields. Values of non persistent fields are not stored to the system database and are therefore not initialized in the process data of the created tasks.

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Output Tab

On this tab you can configure the output of the element (i.e. the data that leaves the element). See Output Tab for a more detailed description.



Note

For each incoming connection you have a separate **inX** object available which carries the data of the Xth input. Hover with the mouse over the incoming connections of the element to learn which input connection corresponds to which variable.

Task Tab

This tab defines the parameters for the tasks created by the Task Switch.

Defines the name, the description and category of the task that appear in the task list of the addressed role or user.



Tip

It is recommended practice to define and reference the text from the CMS. Look at the workflow concept for some more information about categorization.

Responsible Role / User

Defines the role or user required to carry out the task.

Use *Role from Attr.* or *User from Attr.*, if the role or user must be set dynamically during process execution. For example from a process-attribute which holds the name of a role or user.

Use *Role*, if you know the responsible role when editing the element.

The role **SYSTEM** means that no human intervention is required. The system executes the task automatically.

The role **SELF_x** (**SELF1**, **SELF2**, ...) means that the same user that has finished the task on entry **x** is responsible for the task.

The role **CREATOR** means that the user who has started the current case is responsible for the task.

Normally a user interaction ends at a Task Switch element. It will be redirected to the task list or an end page is shown. If *Skip tasklist* is activated for a task the user interaction may not end at the Task Switch element. It is automatically redirected to this new task marked with *Skip tasklist*. But only if it is allowed to work on the task and the Task Switch is not waiting for any other tasks to finish.

Only one task of a Task Switch element can activate *Skip tasklist*.

Priority

Here you select the priority of the task. In the task list you can filter the tasks according the priority.

Delay	The task can be blocked before a user can work on it. This ivyScript expression defines the Duration the task is blocked.
Expiry	<p>An ivyScript expression defines the Duration until the task will expire. If that happens the escalation procedure defined by the following parameters is executed.</p> <ul style="list-style-type: none"> • Exception: Starts an exception process if defined. >> <i>Ignore Exception</i> will not start an exception process. • Responsible Role / User after Expiry: Defines the Role / User to reassign the task to. • Priority after expiry: Defines the new Priority of the task after it has expired.



Note

If a Delay is defined, the expiry timeout begins after the Delay.



Note

A user can be informed by mail if a new task for him was created. This feature is useful for users that only occasionally participate in workflows. On the other hand a user who participate often with the workflow may find daily summary mails useful. User mail notification can be configured on the Axon.ivy Engine in the Engine Administration UI or in the Workflow UI applications (For more information read the chapter User and Roles in the Axon.ivy Engine Guide).

Custom fields Tab

This Tab allows to set additional information for the task created. The values set on this tab are only informational and have no effect on how this task is treated by Axon.ivy.

Business Information Tab

This Tab allows to set additional information to categorize the task created. The values set on this tab are only informational and have no effect on how this task is treated by Axon.ivy.

Business calendar

You can set the name of the business calendar that should be used for this task. In the context of this task `ivy.cal` will return this business calendar regardless of what you've set for the case.

For more information about business calendar administration see the engine guide.

For more information about business calendar usage see the Public API of `ch.ivyteam.ivy.application.calendar.IDefaultBusinessCalendar`.

Case Tab

Every time a process is started a case is created. This tab allows you to define additional information for the cases. The information defined on this tab has no effect how Axon.ivy treats the cases. But they can be accessed through the Public API, which allows you to use them for example to filter the task list.

You can define the name, the description and the category for the corresponding case.



Note

Look at the workflow concept for some more information about categorization.

Custom Fields Tab

Here you can set the values for at most 5 user defined fields for each of the three data types `String`, `Number` and `DateTime`. You can reuse these fields in IvyScript over its API.

Business Information Tab

On this tab it is possible to edit further information about the process such as contact and business data.

Main Contact	Here you can set the information about the company your process deals with.
Correspondent Contact	Here you can set the information about the contact person in the company your process deals with.
Business Object	Here you can set the information about the business object your process deals with.
Other	You can set here additional information about time constraint and further things.
Business calendar	You can set the name of the business calendar that should be used for this case. In the context of this case <code>ivy.cal</code> will return this business calendar instead of the default business calendar. It is also possible to set a business calendar for a task. For more information about business calendar administration see the engine guide. For more information about business calendar usage see the Public API of <code>ch.ivyteam.ivy.application.calendar.IDefaultBusinessCalendar</code> .

Tags Tab

Here you can structure processes in more depth and categorize them into a user-defined hierarchy. These categorization attributes may be accessed later over an API to filter and structure the task overview of users. The structure always consists of 4 levels, the *process category*, the *process*, the *process type* and *process sub type*, e.g. `myProcessCategory/myProcess/myProcessType/myProcessSubtype`.



Tip

Create your specific hierarchy in the CMS so you can ensure its consistent use.

Process Category	Sets the category code and name of the category this process belongs to.
Process	Sets the process code and name.
Type	Sets the process type code and name.
Sub Type	Sets the sub type code and name of this process.



Note

The code is always used for internal reference (caching, searching, sorting and the like) and never shown to the actual user. Therefore, the codes should be unique within its level. The name on the other side is human-readable and always shown to the user when the element is referenced.

End Page Tab

This Tab defines the page, which will be displayed in the web browser for each task which ends at this Task Switch Element.

If no page is defined the task list will be shown.

If no task is created because *skip task list* is enabled (see Task Tab) the case will continue without displaying a page or the task list.

Dialog Page	Pages can be referenced from the content management system or the web content directory. The wizard allows you to create, select or edit pages.
-------------	---

Please refer to [Creating and Editing HTML Pages in the HTML chapter](#) for a more thorough explanation of this tab section.

Task Switch Event



Task

The *Task Switch Event* element is located in the *Event & Gateway* drawer of the process editor palette.

Element Details

The Task Switch Event element has quite the same function than the Task Switch Gateway element. The difference is, that a Task Switch Event element could only have one input and one output, instead of multiple as the parallel Task Switch Gateway. The usage of this simplified element is to have a more BPMN conform element.

See Task Switch Gateway for a more detailed description.

Wait Program Intermediate Event



Wait

The *Wait Program Intermediate Event* element is located in the *Event & Gateway* drawer of the process editor palette.

Element Details

This element is one of Axon.ivy facilities to integrate custom-made software, legacy systems, proprietary applications or any other external system through an Axon.ivy Java interface. At an Intermediate Event element the process execution is interrupted and waits for an external event to occur. Technically spoken the current task will be ended and a new system task is created that waits for the intermediate event. If the intermediate event is fired the new task and therefore the process after the intermediate event will be executed.

You provide a listener for the external event by implementing a Java class that implements the `IProcessIntermediateEventBean` interface. The Wait Program Intermediate Event Element instantiates the Java class and can then trigger the intermediate event by calling the method `fireProcessIntermediateEventEx` on the Axon.ivy runtime engine `IProcessIntermediateEventBeanRuntime`. The common way to implement an Intermediate Event bean is to extend the abstract base class `AbstractProcessIntermediateEventBean`. The interface also includes an inner editor class to parametrize the bean. You will find the documentation of the interface and the abstract class in the Java Doc of the Axon.ivy Public API.

How Process Intermediate Event Beans work on an Axon.ivy Engine Enterprise Edition

An Axon.ivy Engine Enterprise Edition consists of multiple engine instances (nodes) that are running on different machines.

Normally process intermediate event beans are instantiated on every node but only started on the master node. This guarantees that for each *Intermediate Event* process element only one bean is running, no matter what the total number of nodes in the Engine Enterprise Edition is.

However, if you need your intermediate event bean to be started on all cluster nodes, you may instruct the server to do so. Just have your bean class implement the (empty) marker interface `IMultiNodeCapable` and the above restriction will no longer apply.

Please be aware of the fact that having multiple running instances of the same bean may lead to race conditions.


Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Event Tab

On this tab you define the Java class that the IntermediateEvent should instantiate, the identifier of the event to wait for and the timeout behaviour.

Java Class to execute Fully qualified name of the Java class that implements the `IProcessIntermediateEventBean` interface. Use the New Bean Class Wizard () to create a new Java source file with an example implementation of the bean class.

Event ID Because multiple cases (process instances) can wait on the same intermediate event you must specify which event belongs to which waiting case. Here you specify the identifier of the event the current case should wait for.



Warning

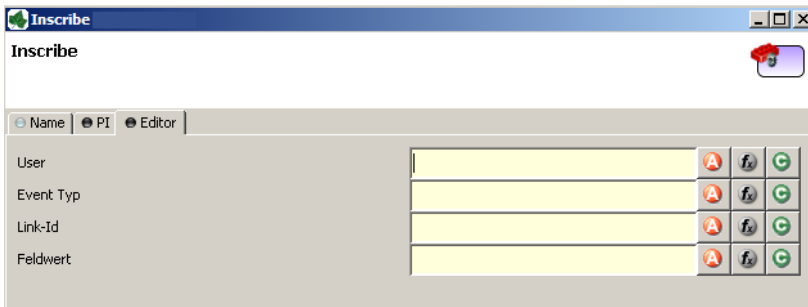
The event identifier as a String must be unique. Do not use a static string like "myID". A good practice is to integrate the case identifier (`ivy.case.getIdentifier()`) into the event id.

Timeout Here you can specify a time (Duration) how long the current case should wait for an intermediate event and what should happen if no event has been received after this time. You can optionally start an exception process, delete the waiting task or continue the waiting task without receiving an intermediate event.

Tab Editor

This tab displays the editor, that can be integrated in the external Java bean of the process element. The editor is implemented as an inner public static class of the Java bean class and must have the name `Editor`. Additionally the editor class must implement the `IProcessExtensionConfigurationEditorEx` interface. The common way to implement the editor class is to extend the abstract base class `AbstractProcessExtensionConfigurationEditor` and to override the methods `createEditorPanelContent`, `loadUiDataFromConfiguration` and `saveUiDataToConfiguration`. The method `createEditorPanelContent` can be used to build the UI components of the editor. You can add any AWT/Swing component to the given `editorPanel` parameter. With the given `editorEnvironment` parameter, which is of the type `IProcessExtensionConfigurationEditorEnvironment`, you can create text fields that support `ivyScript` and has smart buttons which provide access to the process data, environment functions and Java classes.

Here is an example of such an editor:



As you can see, the editor provides you access to any process relevant data, which can be used by your own process elements. For instance, you can easily transfer process data to your legacy system.

The following part shows the implementation of the editor shown above. As mentioned above Axon.ivy provides the `IIvyScriptEditor` which represents a text field with ivyScript support and smart buttons. Inside `createEditorPanelContent` use the method `createIvyScriptEditor` from the `editorEnvironment` parameter to create an instance of such an editor. Use the `loadUiDataFromConfiguration` method to read the bean configuration and set them to the UI components. Inside this method you can use the methods `getBeanConfiguration` or `getBeanConfigurationProperty` to read the bean configuration. Use the method `saveUiDataToConfiguration` to save the data in the UI components to the bean configuration. Inside this method you can use the methods `setBeanConfiguration` or `setBeanConfigurationProperty` to save the bean configuration.

```
public static class Editor extends AbstractProcessExtensionConfigurationEditor
{
    private IIvyScriptEditor editorUser;
    private IIvyScriptEditor editorEventTyp;
    private IIvyScriptEditor editorLinkId;
    private IIvyScriptEditor editorFieldValue;

    @Override
    protected void createEditorPanelContent(Container editorPanel,
        IProcessExtensionConfigurationEditorEnvironment editorEnvironment)
    {
        editorPanel.setLayout(new GridLayout(4,2));
        editorUser = editorEnvironment.createIvyScriptEditor(null,null, "String");
        editorEventTyp = editorEnvironment.createIvyScriptEditor(null,null, "String");
        editorLinkId = editorEnvironment.createIvyScriptEditor(null, null, "String");
        editorFieldValue = editorEnvironment.createIvyScriptEditor(null, null);

        editorPanel.add(new JLabel("User"));
        editorPanel.add(editorUser.getComponent());
        editorPanel.add(new JLabel("Event Typ"));
        editorPanel.add(editorEventTyp.getComponent());
        editorPanel.add(new JLabel("Link-Id"));
        editorPanel.add(editorLinkId.getComponent());
        editorPanel.add(new JLabel("Feldwert"));
        editorPanel.add(editorFieldValue.getComponent());
    }

    @Override
    protected void loadUiDataFromConfiguration()
    {
        editorUser.setText(getBeanConfigurationProperty("User"));
        editorEventTyp.setText(getBeanConfigurationProperty("EventTyp"));
        editorLinkId.setText(getBeanConfigurationProperty("LinkId"));
        editorFieldValue.setText(getBeanConfigurationProperty("Feldwert"));
    }
}
```

```

@Override
protected boolean saveUiDataToConfiguration()
{
    setBeanConfigurationProperty("User", editorUser.getText());
    setBeanConfigurationProperty("EventTyp", editorEventTyp.getText());
    setBeanConfigurationProperty("LinkId", editorLinkId.getText());
    setBeanConfigurationProperty("Feldwert", editorFieldValue.getText());
    return true;
}
}

```

At runtime you have to evaluate the IvyScript the user have entered into the ivy script editors. If you implement for example the `AbstractUserProcessExtension` class there is a `perform` method which is executed at runtime. At this point you want to access the configured data in the editor. The following code snippet show how you can evaluate the value of an `IIvyScriptEditor`. If you use the `IIvyScriptEditor` you only get the value by calling the `executeIvyScript` method of the `AbstractUserProcessExtension`.

```

public CompositeObject perform(IRequestId requestId, CompositeObject in,
    IIvyScriptContext context) throws Exception
{
    IIvyScriptContext ownContext;
    CompositeObject out;
    out = in.clone();
    ownContext = createOwnContext(context);

    String eventtyp = "";
    String linkId = "";
    String fieldValue = "";
    String user= "";

    user = (String)executeIvyScript(ownContext, getConfigurationProperty("User"));
    eventtyp = (String)executeIvyScript(ownContext, getConfigurationProperty("Event Typ"));
    linkId = (String)executeIvyScript(ownContext, getConfigurationProperty("Link-Id"));
    fieldValue = (String)executeIvyScript(ownContext, getConfigurationProperty("Feldwert"));

    // add your call here

    return out;
}

```

Task Tab

Name	Event	Editor	Task	Output
Name				
Description				
Category				
Task kind				
Code*				
Name				
* Maximum 20 characters				
Business milestone				
Milestone date time				
Business calendar				
Business calendar name				

On this tab you configure the parameters of the awaited event which is handled as an Intermediate Event Task. Thus it can be displayed in a task list with a filter applied to the task kind parameter. The task kind code is the internally used identifier to sort and order the processes. The task kind name is a human readable description of the code. The Business milestone usually sets the current `Date`Time for process activity analysis and reports.

You can set the name of the business calendar that should be used for the task. In the context of the task `ivy.cal` will return this business calendar regardless of what you've set for the case.

For more information about business calendar administration see the engine guide.

For more information about business calendar usage see the Public API of `ch.ivyteam.ivy.application.calendar.IDefaultBusinessCalendar`.

Output Tab

On this tab you can configure the output of the element (i.e. the data that leaves the element). You can use the variable `result` that holds additional information about the event received by the Java class.

See Output Tab for a more detailed description.



Note

For each incoming connection you have a separate **inX** object available which carries the data of the Xth input. Hover with the mouse over the incoming connections of the element to learn which input connection corresponds to which variable.

Call & Wait



Call & Wait

The *Call & Wait Intermediate Event* element is located in the *Event & Gateway* drawer of the process editor palette.

Element Details

This element is one of Axon.ivy facilities to integrate custom-made software, legacy systems, proprietary applications or any other external system through an Axon.ivy Java interface. The Call & Wait element is slitted into a Call part and a Wait part. The Call part is similar to the Program Interface process element. It can be used to call (send request) an external system. Whereas the Wait part is similar to the Intermediate Event element and can be used to wait for the response from the external system. For the process designer the use of a Call & Wait element is easier compared to the use of a Program Interface followed by an Intermediate Event because he only has to configure one Java class and does not have to care about event identifiers.

The *Call part* of the element will instantiate a Java class that must implement the interface `IUserAsynchronousProcessExtension` and will call the method `performRequest` each time a process comes to the Call & Wait element. The common way to implement a Call bean is to extend the abstract base class `AbstractUserAsynchronousProcessExtension`.

The *Wait part* of the element will interrupt the process execution and waits for an external event to occur. Technically spoken the current task will be ended and a new system task is created that waits for the event. If the event is fired the new task and therefore the process after the *Call & Wait* element will be executed.

You provide a listener for the external event by implementing a public static inner Java class of the Call part with the name `IntermediateEvent` that implements the `IProcessIntermediateEventBean` interface. The Call & Wait element instantiates the `IntermediateEvent` Java class. It can then trigger the event by calling the method `fireProcessIntermediateEventEx` on the Axon.ivy runtime engine `IProcessIntermediateEventBeanRuntime`. The common way to implement a Wait (Intermediate Event) bean is to extend the abstract base class `AbstractProcessIntermediateEventBean`.

The interface also includes an inner editor class to parametrize the beans. The editor provides one configuration which is set on both beans the Call and the Wait bean. You will find the documentation of the interface and the abstract class in the Java Doc of the Axon.ivy Public API.

How Call & Wait work on an Axon.ivy Engine Enterprise Edition

An Axon.ivy Engine Enterprise Edition consists of multiple server instances (nodes) that are running on different machines.

As described above Call & Wait consists of two parts:

- The *Call part* will be instantiated on all nodes, and it will do its job wherever it resides. After a *call part* a new task will be created in a waiting status
- Normally the *Wait part* works only on the master node. When it fires (when the external event arrives) the task state will change. Such a task can be executed on every node.

This is the standard behaviour in order to eliminate racing conditions in normal *Call & Wait* situations.

Described behaviour is regarded as correct and it should cover most of the use cases. In case you understood the behaviour and still you need the *Wait part* of your *Call & Wait* bean to run on all closer nodes, you may instruct the engine to do so. Just have your bean class implement the (empty) marker interface `IMultiNodeCapable` and the above restriction will no longer apply.

Please be aware of the fact that having multiple running instances of the same bean may lead to race conditions!

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Call Tab

On this tab you set the Java class which implements the interface `IUserAsynchronousProcessExtension` and defines a public static inner class called `IntermediateEvent` that implements the interface `IProcessIntermediateEventBean`. This class is called when the Call & Wait step is executed. Furthermore, you can define exception handlers to react on errors such as not reachable systems, insufficient privileges and many more.

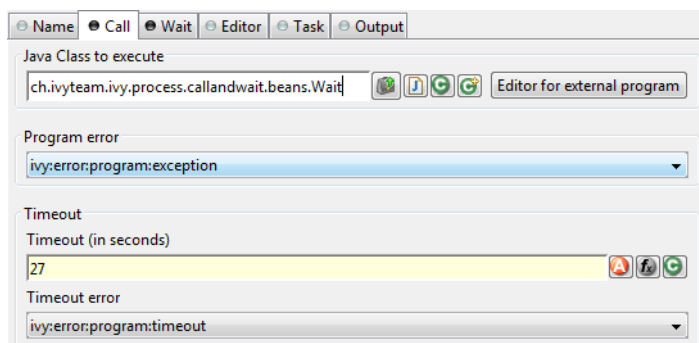


Figure 2.56. The Call tab

Java Class to Execute

The fully qualified name of the Call & Wait Java class implementing `IUserAsynchronousProcessExtension` and a public static inner class called `IntermediateEvent` that implements the interface `IProcessIntermediateEventBean`. Use the New Bean Class Wizard (🔍) to create a new Java source file with an example implementation of the bean class.



Tip

You can add a graphical configuration editor for the Java call (i.e. setting the parameter values) on the Call & Wait inscription mask. See section Tab Editor for more details.

Program Error	Occurs whenever an exception is thrown during the execution of the class. The error can be handled by a catching “Error Start”.
Timeout	Sets a timeout for the return of the call to the Call part Java class.
Timeout Error	Occurs when the timeout is reached. The error can be handled by a catching “Error Start”.

Wait Tab

On this tab you define the timeout behaviour during the Wait part of the element.

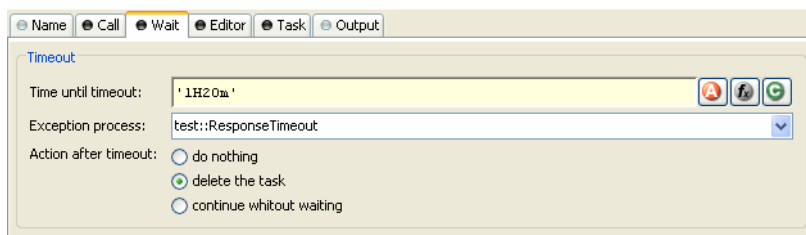


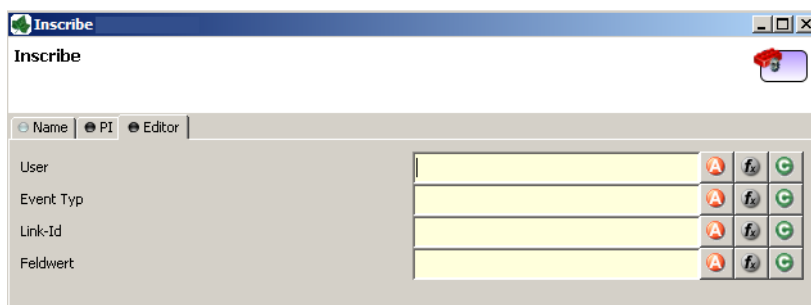
Figure 2.57. The Wait tab

Timeout Here you can specify a time (Duration) how long the current case should wait for an intermediate event and what should happen if no event was received after this time. You can optionally start an exception process, delete the waiting task or continue the waiting task without receiving an intermediate event.

Tab Editor

This tab displays the editor, that can be integrated in the external Java bean of the process element. The editor is implemented as an inner public static class of the Java bean class and must have the name `Editor`. Additionally the editor class must implement the `IProcessExtensionConfigurationEditorEx` interface. The common way to implement the editor class is to extend the abstract base class `AbstractProcessExtensionConfigurationEditor` and to override the methods `createEditorPanelContent`, `loadUiDataFromConfiguration` and `saveUiDataToConfiguration`. The method `createEditorPanelContent` can be used to build the UI components of the editor. You can add any AWT/Swing component to the given `editorPanel` parameter. With the given `editorEnvironment` parameter, which is of the type `IProcessExtensionConfigurationEditorEnvironment`, you can create text fields that support ivyScript and has smart buttons which provide access to the process data, environment functions and Java classes.

Here is an example of such an editor:



As you can see, the editor provides you access to any process relevant data, which can be used by your own process elements. For instance, you can easily transfer process data to your legacy system.

The following part shows the implementation of the editor shown above. As mentioned above Axon.ivy provides the `IIvyScriptEditor` which represents a text field with ivyScript support and smart buttons. Inside `createEditorPanelContent` use the method `createIvyScriptEditor` from the `editorEnvironment` parameter to create an instance of such an editor. Use the `loadUiDataFromConfiguration` method to read the bean configuration and set them to the UI components. Inside this method you can use the methods `getBeanConfiguration` or `getBeanConfigurationProperty` to read the bean configuration. Use the method `saveUiDataToConfiguration` to save the data in the UI components to the bean configuration. Inside this method you can use the methods `setBeanConfiguration` or `setBeanConfigurationProperty` to save the bean configuration.

```
public static class Editor extends AbstractProcessExtensionConfigurationEditor
{
    private IIvyScriptEditor editorUser;
    private IIvyScriptEditor editorEventTyp;
    private IIvyScriptEditor editorLinkId;
    private IIvyScriptEditor editorFieldValue;

    @Override
    protected void createEditorPanelContent(Container editorPanel,
        IProcessExtensionConfigurationEditorEnvironment editorEnvironment)
    {
        editorPanel.setLayout(new GridLayout(4,2));
        editorUser = editorEnvironment.createIvyScriptEditor(null,null, "String");
        editorEventTyp = editorEnvironment.createIvyScriptEditor(null,null, "String");
        editorLinkId = editorEnvironment.createIvyScriptEditor(null, null, "String");
        editorFieldValue = editorEnvironment.createIvyScriptEditor(null, null);

        editorPanel.add(new JLabel("User"));
        editorPanel.add(editorUser.getComponent());
        editorPanel.add(new JLabel("Event Typ"));
        editorPanel.add(editorEventTyp.getComponent());
        editorPanel.add(new JLabel("Link-Id"));
        editorPanel.add(editorLinkId.getComponent());
        editorPanel.add(new JLabel("Feldwert"));
        editorPanel.add(editorFieldValue.getComponent());
    }

    @Override
    protected void loadUiDataFromConfiguration()
    {
        editorUser.setText(getBeanConfigurationProperty("User"));
        editorEventTyp.setText(getBeanConfigurationProperty("EventTyp"));
        editorLinkId.setText(getBeanConfigurationProperty("LinkId"));
        editorFieldValue.setText(getBeanConfigurationProperty("Feldwert"));
    }

    @Override
    protected boolean saveUiDataToConfiguration()
    {
        setBeanConfigurationProperty("User", editorUser.getText());
        setBeanConfigurationProperty("EventTyp", editorEventTyp.getText());
        setBeanConfigurationProperty("LinkId", editorLinkId.getText());
        setBeanConfigurationProperty("Feldwert", editorFieldValue.getText());
        return true;
    }
}
```

At runtime you have to evaluate the IvyScript the user have entered into the ivy script editors. If you implement for example the `AbstractUserProcessExtension` class there is a `perform` method which is executed at runtime. At this point you want to access the configured data in the editor. The following code snippet show how you can evaluate the value of an

IIvyScriptEditor. If you use the IIvyScriptEditor you only get the value by calling the executeIvyScript method of the AbstractUserProcessExtension.

```
public CompositeObject perform(IRequestId requestId, CompositeObject in,
    IIvyScriptContext context) throws Exception
{
    IIvyScriptContext ownContext;
    CompositeObject out;
    out = in.clone();
    ownContext = createOwnContext(context);

    String eventtyp = "";
    String linkId = "";
    String fieldValue = "";
    String user= "";

    user = (String)executeIvyScript(ownContext, getConfigurationProperty("User"));
    eventtyp = (String)executeIvyScript(ownContext, getConfigurationProperty("Event Typ"));
    linkId = (String)executeIvyScript(ownContext, getConfigurationProperty("Link-Id"));
    fieldValue = (String)executeIvyScript(ownContext, getConfigurationProperty("Feldwert"));

    // add your call here

    return out;
}
```

Task Tab

On this tab you configure the parameters of the awaited event which is handled as an Intermediate Event Task. Thus it can be displayed in a task list with a filter applied to the task kind parameter. The task kind code is the internally used identifier to sort and order the processes. The task kind name is a human readable description of the code. The Business milestone usually sets the current `DateTime` for process activity analysis and reports.

You can set the name of the business calendar that should be used for the task. In the context of the task `ivy.cal` will return this business calendar regardless of what you've set for the case.

For more information about business calendar administration see the engine guide.

For more information about business calendar usage see the Public API of `ch.ivyteam.ivy.application.calendar.IDefaultBusinessCalendar`.

Output Tab

On this tab you can configure the output of the element (i.e. the data that leaves the element). You can use the variable result that holds additional information about the event received by the Wait part Java class.

See Output Tab for a more detailed description.



Note

For each incoming connection you have a separate **inX** object available which carries the data of the Xth input. Hover with the mouse over the incoming connections of the element to learn which input connection corresponds to which variable.

Process End Page



End Page

The *Process End Page* element is located in the *Event & Gateway* drawer of the process editor palette.

Element Details

This element terminates the current process and displays a dialog page.

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

End Page Tab

On this tab you could define the web page displayed when a case ends with this End Page Element.

Dialog Page Pages can be referenced from the content management system or the web content directory. The wizard allows you to create, select or edit pages.

Please refer to *Creating and Editing HTML Pages* in the *HTML* chapter for a more thorough explanation of this tab section.

Error End



Error End

The *Error End Event* element is located in the *Event & Gateway* drawer of the process editor palette.

Element Details

The Error End Event can be used to leave the happy path of a process by throwing an error (e.g. if an approval is denied). It can also be used to re-throw previously caught errors.

See the Error Handling concept for sample use cases.

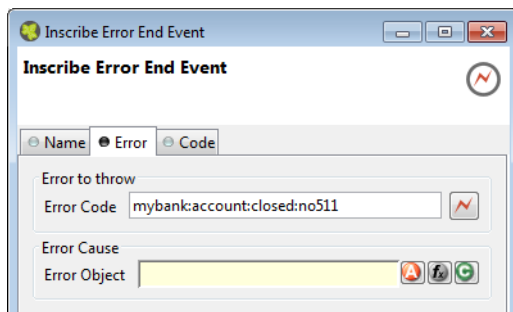
Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Error Tab

On this tab you can configure the Error Code of the error that will be thrown or the error object that should be re-thrown.



Code Tab

On this tab you can execute additional scripts after the error has been created. See Code Tab for a more detailed description.



Note

Additionally to the regular variables of the *Code Tab* you have the following variable available:

error References the `BpmError` which will be thrown.

Process End



The *Process End* element is located in the *Event & Gateway* drawer of the process editor palette.

Element Details

This elements terminates the current process.

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

User Task



The *User Task* element is located in the *Activity* drawer of the process editor palette.

Element Details

The *User Task* element calls a User Dialog in a new Task. Thus, it combines the behavior of a Task Switch Event and a User Dialog. You can either call a normal Html Dialog or an Offline Dialog - they both are based on JSF technology and can run in a Web Browser as well as on a mobile client.

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Call Tab

The *Call tab* defines what User Dialog component should be called and how it should be started. The input parameters for the selected start method can be mapped here.

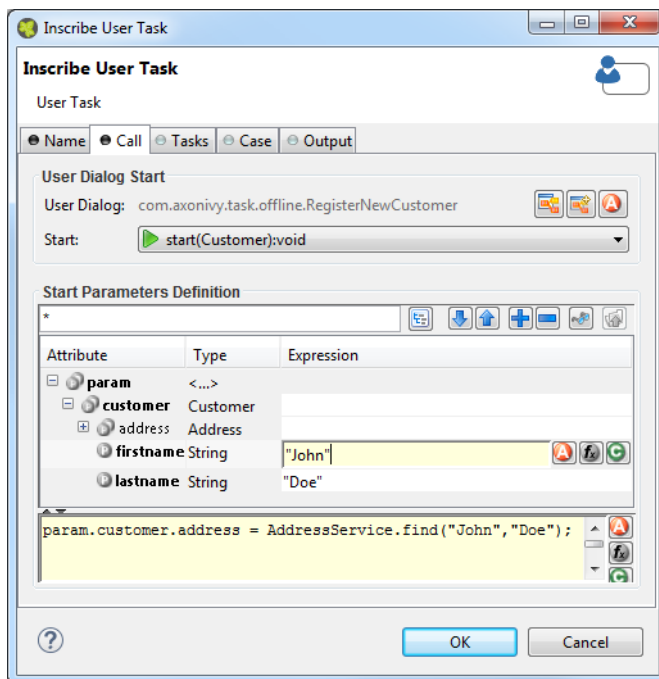




Figure 2.58. The Call tab

User Dialog

Defines the User Dialog to be started by its ID. The referenced User Dialog can either be a normal JSF based dialog component (Html Dialog) or a JSF based dialog that is designed for offline usage (Offline Dialog).

 Selects an existing User Dialog

 Creates a new User Dialog and uses it

 Uses a *dynamically* defined ID from a data class attribute



Note

The behavior of the task will be noticeable different either if you select an Html Dialog or an Offline Dialog. By selecting an Offline Dialog, an Offline Task - designed for processing without continuous connection to the workflow server - will be generated.

Start

Defines the *start method* that should be called on the selected User Dialog.

If the User Dialog to be started is defined dynamically, then the start cannot be selected. Instead the default *start()* method (no input, no output) will be called by default. If this method does not exist on the dynamically defined User Dialog, then a runtime error will occur.

If the User Dialog to be started is defined statically (i.e. with a specific ID) then the start combo box offers a list of all start methods that are declared by the selected User Dialog.

Start Parameters Definition

Define the input parameters for the called User Dialog.

If the selected *start method* requires any parameters, those may be mapped here to an **param** object, which offers a field for each declared start method parameter. You can define each parameter individually from the calling process's data.



Tip

Alternatively you can define the call parameters in the scripting field below the attribute table, e.g. if you need to perform some calculation in order to define a call parameter. You can also mix the two forms, in which case the table's definitions will be executed before the scripting block.



Note

The result values of the started User Dialog (if any) are mapped back onto the calling process's data on the *Output* tab. They are available as fields on a **result** variable.

Task Tab

On this tab you can configure the Task of this User Task. See “Task Tab” in the Task Switch Gateway element.

Case Tab

On this tab you can configure the Case of this User Task. See “Case Tab” in the Task Switch Gateway element.

Output Tab

On this tab you can configure the output of the element (i.e. the data that leaves the element). See Output Tab for a more detailed description.



Note

Additionally to the regular variables of the *Output Tab* you have the following variable available:

result

If the selected *start method* defines return parameters they will be available as fields of the **result** variable.

The variable is *not* available if the start method does not return any values (i.e. *void*).

Web Page



The *Web Page* element is located in the *Activity* drawer of the process editor palette.

Element Details

This element presents an interactive web page to the user via his browser and may be defined in the CMS or externally by a .html, or .jsp file.

For each exit of this element (the outgoing arrows from this element) a link is set that defines which way the process proceeds. By clicking on such a link (it may be a simple link or one combined with an input form) the user carries the data object with the process data to the appropriate exit, i.e. process path.

In case of a form the data that have been entered into it will be assigned to the process data attributes.

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Tab Dialog

On this tab you set the dialog page and its properties. In addition, you are able to create the Web Page from scratch and store it in the CMS.

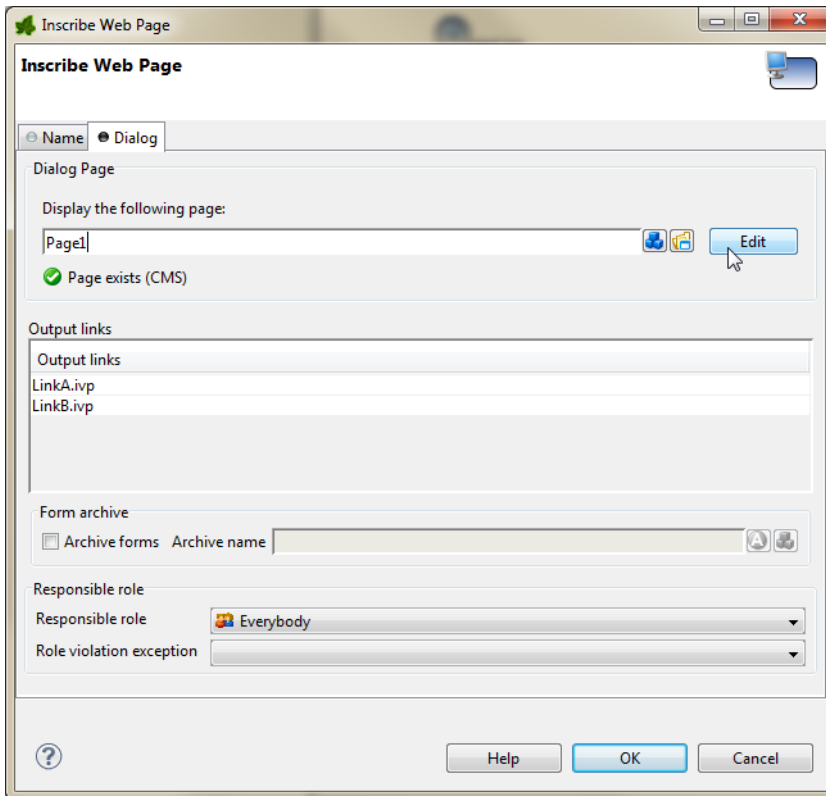
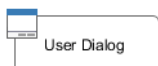


Figure 2.59. The Dialog tab

Dialog Page	The selected page will be displayed in the user's browser whenever the element is activated by the process. Pages can be referenced from the content management system or the web content directory. The wizard allows you to create, select or edit pages. Please refer to <i>Creating and Editing HTML Pages</i> in the HTML chapter for a more thorough explanation of this tab section.
Output Links	List of the links with which the process may proceed to the next step. You can edit the names of the links as you like but they always need to have an .ivp file extension. The links appear in the order they were connected with the HTML Page element. Pausing the mouse cursor shortly over one of the outgoing arrows of the element shows a tool tip that indicates the name of the corresponding link.
Form Archive	Each page (including the forms located on it and the inputs of the user) are archived on the Axon.ivy Engine. The archived pages (forms) are associated with the running case and the running task and can be inspected (viewed) afterwards in the workflow user interface.
Responsible Role	Restricts the access to this dialog to the given role.
Role Violation error	This error is thrown whenever a user tries to access the dialog page without having granted the required role. The error can be handled by a catching "Error Start" or by an "Error Boundary Event"

User Dialog



The *User Dialog* element is located in the *Activity* drawer of the process editor palette.

Element Details

The User Dialog element is used to call a User Dialog from a process. You can both start User Dialogs from a *business process* as well as from another *User Dialog*. Input and return parameters for/from the started User Dialog can be mapped using the *call* and *output* tabs of this element.

If a User Dialog is opened from within another User Dialog's logic then certain restrictions apply (see below).

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Call Tab

The *Call tab* defines what User Dialog component should be called and how it should be started. The input parameters for the selected start method can be mapped here.

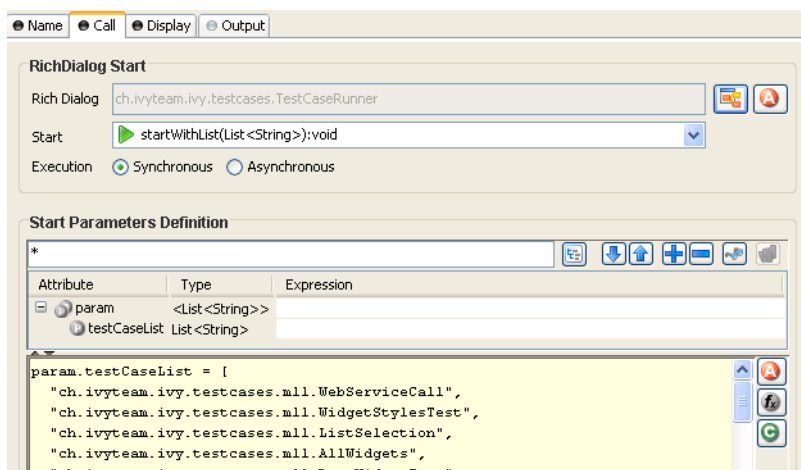




Figure 2.60. The Call tab

User Dialog	<p>Defines the User Dialog component to be started by its ID.</p> <p>You can either select a Rich Dialog component or a Html Dialog component by clicking on the <i>User Dialog Browser</i> () button or by defining the ID <i>dynamically</i> from a data class attribute by clicking on the <i>Attribute</i> () button.</p>
Start	<p>Defines the <i>start method</i> that should be called on the selected User Dialog.</p> <p>If the User Dialog to be started is defined dynamically, then the start cannot be selected. Instead the default <i>start()</i> method (no input, no output) will be called by default. If this method does not exist on the dynamically defined User Dialog, then a runtime error will occur.</p> <p>If the User Dialog to be started is defined statically (i.e. with a specific ID) then the start combo box offers a list of all start methods that are declared by the selected User Dialog.</p>
Execution (only available for Rich Dialogs)	<p>Defines whether the selected Rich Dialog should be started <i>synchronously</i> or <i>asynchronously</i>.</p>

Rich Dialogs are normally started synchronously, i.e. the calling process will stop and wait for the Rich Dialog to close before it continues. If you start a Rich Dialog *asynchronously*, then the calling process will immediately continue and will not wait for the started Rich Dialog to return any parameters. As a consequence of this, the called Rich Dialog will be running *in parallel* to the continuing process and its return parameters will not be considered for further execution.



Note

If a synchronous Rich Dialog is started from within a Rich Dialog process then it will always open as a *modal dialog*. In this case, all target parameters of the Display Tab will be ignored. Any custom window configuration, however, will be used and applied for the modal dialog.

If you call a synchronous Rich Dialog from a callable process then the specification of the Display parameters makes sense, because the callable may both be invoked from a business process (in which case they are considered) or a Rich Dialog process (in which case they are ignored).



Warning

Do not open synchronous Rich Dialogs inside a Rich Dialog start method.

This will not work for technical reasons, because the initialization of the outer Rich Dialog will not have been completed at this point of time.

You may, however, open any number of *asynchronous* Rich Dialogs inside a start method.

If you'd like to open a *synchronous* Rich Dialog immediately after some Rich Dialog becomes visible, then you should do this in a deferred event process that you trigger with the usage of a hidden button in conjunction with `ivy.rd.clickDeferred(...)`.

See the similar warning note of the Synchronize UI process element for an example of a deferred UI process execution.

Start Parameters Definition

Define the input parameters for the called User Dialog.

If the selected *start method* requires any parameters, those may be mapped here to an **param** object, which offers a field for each declared start method parameter. You can define each parameter individually from the calling process's data.



Tip

Alternatively you can define the call parameters in the scripting field below the attribute table, e.g. if you need to perform some calculation in order to define a call parameter. You can also mix the two forms, in which case the table's definitions will be executed before the scripting block.



Note

The result values of the started User Dialog (if any) are mapped back onto the calling process's data on the *Output* tab. They are available as fields on a **result** variable.

User Context (only available for Rich Dialogs)

Shows the current Rich Dialog User Context configuration.

Press the button *configure* to edit the configuration in a separate dialog.



Note

The Rich Dialog User Context is used to store the UI State.

Display Tab (only available for Rich Dialogs)

The *Display tab* defines where the Rich Dialog is opened. If you want to open the Rich Dialog in a new *Window* you can ignore this tab.

Target Location

Define here *where* the started Rich Dialog should open. The target of a call is always a *Display* container inside a *Window*, therefore you need to specify both a display and a window.



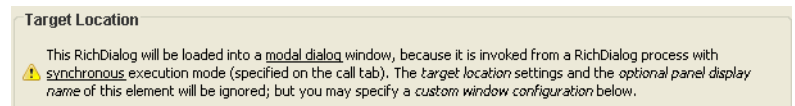
Note

Depending on whether the Rich Dialog is started from a *business process* or a *callable process* or from within another Rich Dialog you have different options available for the *target location* selection.

The option **THIS** both for display and window is only available if the Rich Dialog is started directly from within another Rich Dialog, because it uses the surrounding dialog as a reference.

If - at runtime! - a *synchronous* Rich Dialog is opened from inside a callable process that was invoked from a Rich Dialog process, then all target location parameters will be ignored and the Rich Dialog will be shown inside a modal dialog window.

If the Rich Dialog is invoked directly (e.g. not via a callable process) from a Rich Dialog process, then the *display tab* shows the following message, because it can be determined for sure that the Rich Dialog will open modally:



Target location settings can then not be specified for this element. This behavior only applies to *synchronous Rich Dialogs* that are called from a *Rich Dialog process*.

Possible *target window* specifications:

THIS The Rich Dialog will be opened in the same window as the calling Rich Dialog. (This option is not available for Rich Dialogs that are opened from a business process or a callable sub process).

NEW A new window will be opened to show the Rich Dialog. The type and layout of the new window can be specified by selecting a *window type*.



Note

The system offers some basic window types like *card* or *tabbed*, but window types can also be specified by the

user inside the configuration editor. A window type is nothing more than a window with a specific Rich Dialog (that defines at least one display component), which will serve as basic content of the window. The Rich Dialog that you actually want to load will then be loaded onto the/a display of the Rich Dialog that specifies the window type. Example: the *tabbed* window type uses a Rich Dialog that only consists of a tabbed display.

- EXISTING** A specific window id must be provided. The Rich Dialog will then be loaded into the window that is referenced by the that window id.
- DEFAULT** The Rich Dialog will be shown inside the singleton default window of the application. The default window will be created and brought to front, if it isn't open yet.
- APPLET** The Rich Dialog will be opened inside an applet. Obviously this option only makes sense inside a HTML environment/business process. For an applet, a window type may be specified, which will be used for the applets (internal) layout.

You may also specify a custom applet configuration (see *Custom Applet Configuration* below) to specify the applet tag's rendering properties on the HTML page.



Warning

Please note that for technical reasons only a single instance of an applet can exist per user and session!



Warning

If you work with applets, then you must start your Axon.ivy processes that open the applets from an external browser. The internal browser of Axon.ivy is not able to show applets for security reasons.

To start your applet processes from an external browser proceed as follows:

1. Start the engine to compile and show the list of startable process starts.
2. Open an external browser and navigate to `http://localhost:8081/ivy/`
3. Start your applet process with a click on the respective link.

Possible *target display* specifications:

- TOP** The Rich Dialog will be loaded onto the first display that is found in the component hierarchy of the specified target window.
- THIS** The Rich Dialog will be opened on the same display as the calling Rich Dialog, i.e. as a *sibling* of the surrounding dialog. (This option is not available for Rich Dialogs that are opened from a business process or a callable sub process).

SPECIFIC The id of a *display* must be provided. This display is then searched within the specified target window and the Rich Dialog will be loaded onto it, if it is found.

Custom Window/Applet Configuration

The appearance of the window or applet to open may be configured here.



Note

Whether the here defined parameters are actually used at runtime, depends on the specific execution context. If the Rich Dialog is loaded onto an existing window (e.g. with `THIS` or `EXISTING`) then the window configuration will obviously not be used.

If the Rich Dialog is opened as a modal dialog (because it is *synchronous* and invoked from a Rich Dialog process) then the window configuration will be used. Similarly it will be applied to `NEW` windows and to `APPLET` windows (for the latter in a slightly modified form).

If the `DEFAULT` window does not yet exist, then the window configuration will be used to create it, otherwise it will be ignored.

Activate the option *Specify a custom window/applet configuration*.

If you've specified a **window** target, then the *title* and *size* of the new window to be opened may be specified (if no size is specified, then the window's size is calculated from the opened Rich Dialog's preferred size). You can also specify whether the new window should be *resizeable*, *centered* and support *maximization*.

Use the option *close after last rich dialog* if the window should be closed automatically when the last Rich Dialog that it contains is closed.



Tip

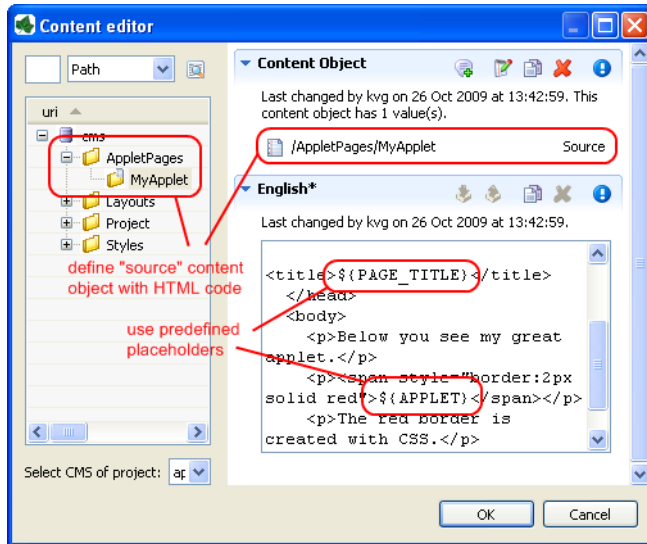
If you don't specify a *custom window/applet configuration* then the default configuration will be used: the new window's ID will be used as title, the window will be resizeable and centered and closed after the last rich dialog.

If you've specified an **applet** target, then *resizeable*, *centered* and *close after last rich dialog* are not available. Instead you may optionally define the URI of a page in the content management system that will be used to render the applet.

Inside that page's HTML code, you may use two predefined variables, `#{PAGE_TITLE}` and `#{APPLET}`. `#{PAGE_TITLE}` will be filled with the *title* that is specified inside the custom display configuration. `#{APPLET}` will be replaced with HTML code to render the applet on the page. The specified *width* and *height* and the optional *maximize* flag will determine the applets size on the page.

How to create a custom applet configuration

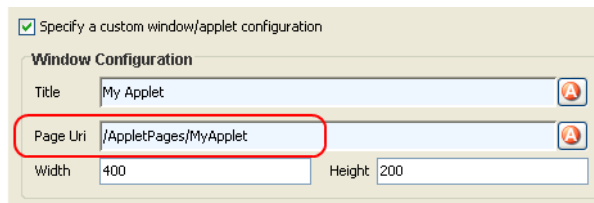
Create a content object with type `source` in the CMS. Enter the code of a valid HTML page as the content object's value. Use the predefined variables `#{PAGE_TITLE}` and `#{APPLET}` to specify where title and applet tag should be inserted into the HTML code.



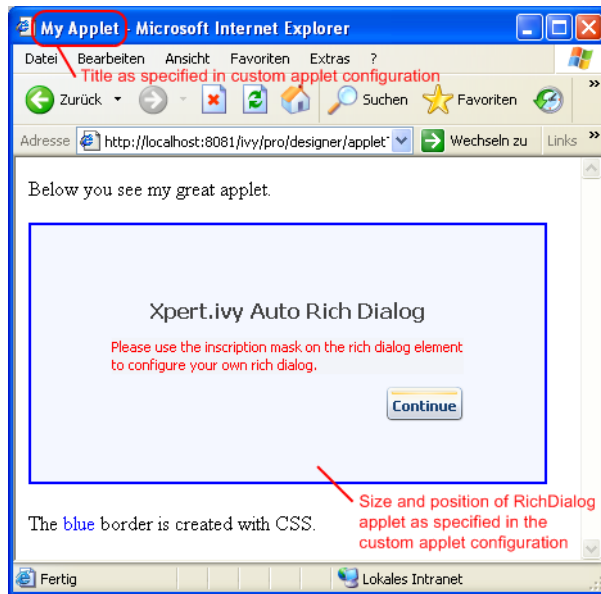
You can use the following example code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>${PAGE_TITLE}</title>
</head>
<body>
<p>Below you see my great applet.</p>
<p><span style="border:2px solid blue">${APPLET}</span></p>
<p>The <span style="color:blue">blue</span> border is created with CSS.</p>
</body>
</html>
```

After you've defined the applet page HTML code, open the Rich Dialog element from where you want to start the applet. Specify a custom applet configuration and enter the URI of the source content object that you just created in the *Page Uri* field.



Now open an external browser (as explained above) and start the applet business process from there. Something similar to the following screen shot should be the result of your first RichApplet invocation:



Note


If you don't specify a custom applet configuration, then the applet will be rendered on a page that is specified by the selected applets window type or - if that window type does not specify a custom applet configuration - on an empty page with a default size of 400 x 400 pixel.



Tip

Usually you open your Rich Applets sequentially, one at a time. However, if you choose to invoke another process that opens another asynchronous APPLET Rich Dialog and if there's already a RichApplet running, then the loading of the new Rich Dialog will be redirected to the already existing applet (similar to the DEFAULT application window).

Panel Display Name

Depending on the capabilities of the target *display* there may be the possibility to show a name for the started Rich Dialog's panel (e.g. the tab title if the target display is a *Tabbed Display*). This optional display name can be entered here, either as constant value or dynamically (use the  button to select from process data). If the field is left empty, then the Rich Dialog name will be used as display name.

Panel Display Icon

Depending on the capabilities of the target *display* there may be the possibility to show an icon for the started Rich Dialog's panel.

Custom Panel Display Parameters

Depending on the capabilities of the target *display* there may be custom parameters to define how the started Rich Dialog's panel is displayed.

Consult the documentation of the *display* to find out what custom parameters are available.

Output Tab

On this tab you can configure the output of the element (i.e. the data that leaves the element). See Output Tab for a more detailed description.



Note

Additionally to the regular variables of the *Output Tab* you have the following variables available:

result	If the selected <i>start method</i> defines return parameters they will be available as fields of the result variable. The variable is <i>not</i> available if the start method does not return any values (i.e. <i>void</i>).
panel	The panel of the just finished User Dialog is still available at the time when the output is calculated. You can e.g. use this variable to request some inner state of the just finished User Dialog (apart from the returned values available on the result variable).

Script Step



The *Script Step* element is located in the *Activity* drawer of the process editor palette.

Element Details

With this element you can perform any transformation of the process data or start some other processing in order to make preparations for later steps.



Warning

It is strongly recommended to use the dedicated process elements if you intend to use specific functionality and/or technology (such as invoking Web Services, querying Databases and so on) as these elements encapsulate their use and handle exceptions internally.

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Output Tab

On this tab you can configure the output of the element (i.e. the data that leaves the element). See Output Tab for a more detailed description.

Code Tab

On this tab you can execute any script, e.g. define output data of this element. See Code Tab for a more detailed description.



Tip

The entered code will be executed *after* the execution of the *output tab*. Although this may seem a bit counter-intuitive at first, you should simply regard the code tab as an alternative way of defining output data. The general recommendation is to use the output table to define simple assignments and the code tab if more extensive scripting is needed to calculate data.

Disable Permission Checks
(Execute this Script Step as
SYSTEM)

With this option enabled the scripts from the Output and Code Tab runs without security permission checks. The execution of the scripts will never throw any `PermissionDeniedException`.



Warning

Use this possibility with caution! In this case you as process developer are responsible that only authorized users can reach this Script Step in the process.

DB Step



The *Database Step* (DB Step) element is located in the *Activity* drawer of the process editor palette.

Element Details

With this element you can execute SQL commands on the database server. You can access all the databases that are defined in the DB Configuration.

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

DB Tab

Here you define which SQL command you want to execute on which database. Depending on the kind of SQL command you are supported during the construction of your command.

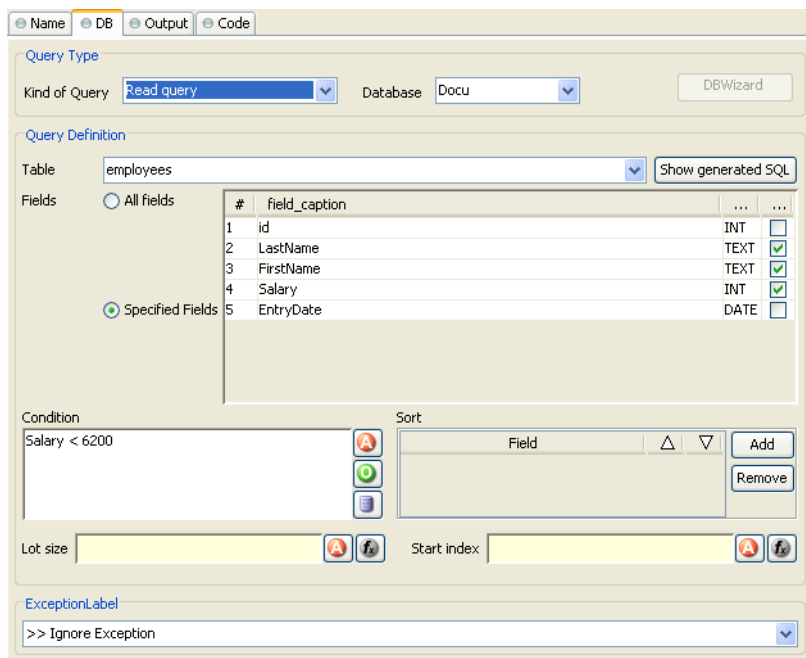


Figure 2.61. The DB Tab

Kind of Query Choose the kind of query you like to perform. Axon.ivy offers dedicated UI support for the most common query types such as **Select** (Read Query), **Insert** (Write Query), **Update** (Update Query) and **Delete** (Delete Query).

If you require some non-standard SQL or if you want to issue a complex SQL statement that is not covered by the *Query Definition* mask then you may also select **Any Query** and write pure SQL instead. Expanding of process attributes will also work in the *Any Query* mode.

Database Choose the database on which the command is executed. The database must be configured in the DB configuration. Depending on the active environment the right connection properties of the database will be used.



Warning

Please note that the **DB2** database is currently not fully supported by the DB Step. The only query kind that is suitable for DB2 connections is **Any Query**. For all other query kinds the *Query Definition* mask is currently not working correctly (e.g. query fields can not be edited / defined).

There is also an `IvySystemDatabase` datasource which points to the current System Database. Normally you would prefer your own database to split valuable customer data from the system data.



Warning

Do not manipulate system database tables prefixed with `IWA_` within the `IvySystemDatabase` this could lead to unexpected runtime behavior.

Query Definition Depending on the type of query you can compose your command with almost no knowledge about databases and SQL.

Table	The name of the database table to read from, insert into, update in or delete from.
Fields	The fields of the database table to read from, insert values into or update values in.
Condition	A condition that filters the rows of the table to read, update or delete.
Sort	Defines the fields after which the rows that are read from the database are sorted.
Lot size	Defines how many rows are read from the database. Enter 0 or leave it empty for no limitation.
Start index	Defines the number of the row that is the first row in the read recordset out of the overall rows which match the condition.
Quote IvyScript variables	IvyScript variables in the SQL query are quoted depending on the data type of the value of the variable. For example string values are quoted with single quotes (e.g. hello -> 'hello'). Sometime you do not want that the values are quoted because the variable do not contain a single value but a part of an SQL query (e.g. "id=123 AND name=ivy"). Therefore you can disable the quoting with this check box.

Error Is thrown whenever errors during the execution of the database command occur. The error can be handled by a catching "Error Start" or by an "Error Boundary Event".

**Tip**

SQL experts can review the generated SQL command by clicking on the *Show generated SQL* or by choosing *Any other query* in the query combo box.

Data Cache Tab

On this tab you can configure the settings for data cache access or invalidation. See Data Cache Tab for a more detailed description.

Output Tab

On this tab you can configure the output of the element (i.e. the data that leaves the element). See Output Tab for a more detailed description.

Code Tab

On this tab you can execute any script, e.g. define output data of this element. See Code Tab for a more detailed description.

**Tip**

The entered code will be executed *after* the execution of the *output tab*. Although this may seem a bit counter-intuitive at first, you should simply regard the code tab as an alternative way of defining output data. The general recommendation is to use the output table to define simple assignments and the code tab if more extensive scripting is needed to calculate data.

Web Service Call Step



The *Web Service Call Step* element is located in the *Activity* drawer of the process editor palette.

Element Details

Using the Web Service Call step you can invoke Web Services. Within the element you first have to select a Web Service configuration (predefined in Web service configuration editor). According to the selected configuration you then may select the port type and the Web Service operation you want to invoke. Necessary arguments must be specified for the chosen Web Service operation. You might want to map return values to process data (see Output tab), and you can even use own IvyScript code as well (see Code tab).

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

WS Tab

Figure 2.62. The Web Service tab

The tab WS is included in the mask of Web Service Call Step element. It features the following entries:

Web Service Configuration The selected global Web Service configuration. Web Service configuration of required projects can be used by this element too. Authentication information defined by such referred Web Service configuration can be overridden.



Tip

Starting typing the name of the global configuration and the configuration selection window will appear. Hitting the **DEL** key will erase selection.

Web Service Name This is an information field. It shows the Web Service name extracted from the corresponding *WSDL* file of the selected Web Service configuration.



Tip

By clicking on the info icon, a new window appears and shows additional information about the generated Web Service classes.

Web Service Description This is also an information field. If the *WSDL* file of a Web Service configuration contains a description, then it is extracted at Web Service class generation and displayed here.

Override authentication Normally the authentication information from the selected global web service configuration is used. If you check "override" then you must specify authentication information as well (authentication type, user name, password)

Following authentication types are available:

- HTTP basic: HTTP basic authentication is used.
- HTTP digest: HTTP digest authentication is used.
- NTLM: Microsoft's NTLM authentication is used.

User Name This option is only available if you selected **Override authentication**. Name of a user is expected for authentication.

Password	This option is only available if you selected Override authentication . Password for the above specified user is expected for authentication.
Host	This option is only available if you selected Override authentication and NTLM authentication.
Domain	This option is only available if you selected Override authentication and NTLM authentication. Specify the Windows domain of the user that should be used to authentication the user.
Web Service error	Is thrown whenever an error occurs during the execution of this element. The error can be handled by a catching “Error Start” or an “Error Boundary Event”.
Timeout in Seconds	You may define a timeout. When you do it, the Web Service invocation must end within a defined time period. If no response returns in time, a timeout exception is thrown.
Timeout error	Is thrown if the defined timeout is reached. The error can be handled by a catching “Error Start” or an “Error Boundary Event”.

Web Service Config Tab

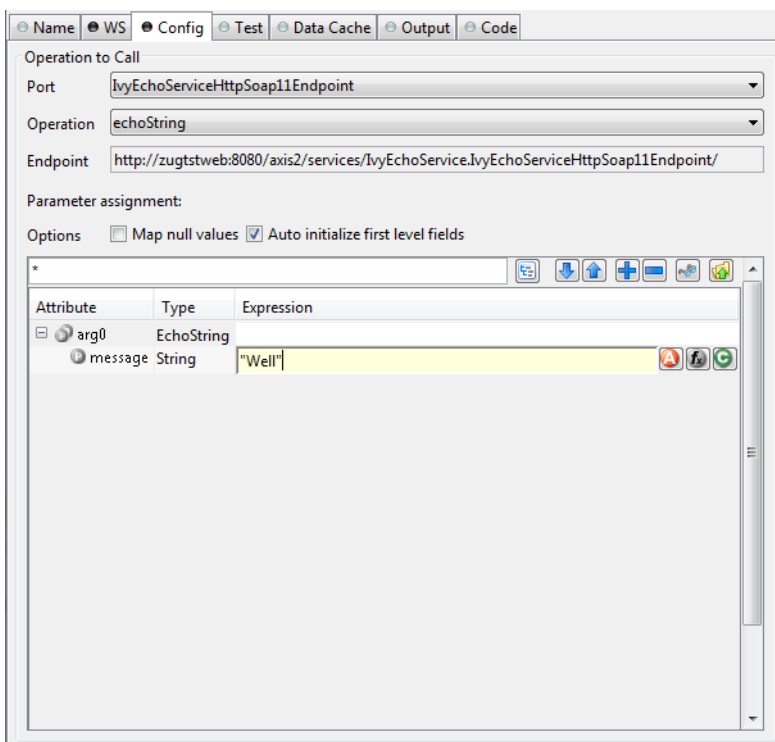


Figure 2.63. The Web Service Config tab

The tab Config is included in the mask of Web Service Call Step process element. It features the following entries:

Port	WSDL port type to be used at Web Service invocation. Service port type may group operations according to some logical or technical aspects. See further details at Web Service configuration.
Operation	Select the Web Service operation you want to invoke from server.
Endpoint	Setting displayed in this field comes from the selected Web Service configuration (default end point)

Parameter Assignment

According to the selected Web Service operation a parameter assignment is carried out here. Should the selected operation have no input arguments, this table will stay empty. As soon as the selected operation needs input arguments you should assign parameter values. Otherwise, it is likely that you get some errors due to unsatisfied input arguments.

**Warning**

If WSDL defines a parameter as a sequence of objects, the respective parameter on the generated Java client class will have an array type (on the config tab the parameter appears with type `Array<?>`). IvyScript treats such Array types as if they were Lists and allows to add members (e.g. `wsParamStringArray.add("a string")`). In fact, Array type manipulations done with a List method (e.g. `add()`, `remove()`) will have no effect on the original Array parameter value.

Members can be added to an Array parameter when they are previously stored in a temporary List. This List can then be assigned to the array parameter.

Example: Assuming our Web service "FruitOrder" expects a sequence of fruit names as strings.

```
List<String> listOfFruits = ["Apple", "Pear",  
"Strawberry"];
```

```
wsParamStringArray = listOfFruits;
```

Map null values

Depending on this setting null values evaluated on the right hand side are mapped to the Web Service parameters on the left hand side. If this option is not selected null values on the right hand side will be ignored.

Auto initialize first level fields

Depending on this setting complex fields of the Web Service parameters are auto initialized (first level only). A complex field is a field with a type that contains fields again. If this option is not selected the fields will be null if no values are explicitly mapped to them or one of its children.

Test Tab

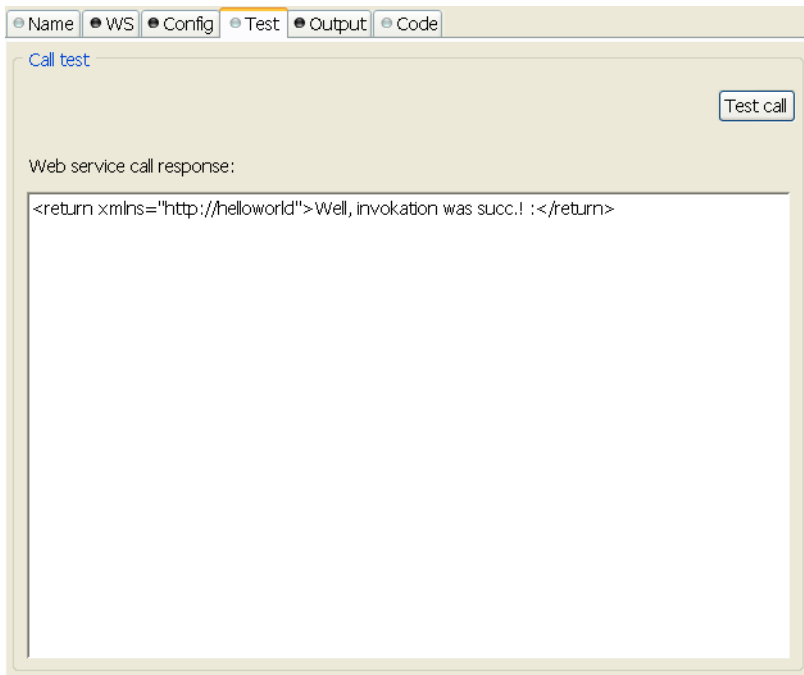


Figure 2.64. The Web Service Test tab

The tab Web Service Test is included in the mask of a Web Service call element. It features the following entries:

Test call

You can test a configured Web Service invocation. Clicking **Test Call** button invokes the web service and displays the response.



Note

Since process attributes are not available at this test time, a separate window appears (only when the Web Service needs input data) to let you enter the parameter values for the test call.

Web Service Call Response

The response of the Web Service invocation is displayed here. If an exception occurs, the error message is displayed.

Data Cache Tab

On this tab you can configure the settings for data cache access or invalidation. See Data Cache Tab for a more detailed description.

Output Tab

On this tab you can configure the output of the element (i.e. the data that leaves the element). See Output Tab for a more detailed description.

Code Tab

On this tab you can execute any script, e.g. define output data of this element. See Code Tab for a more detailed description.



Tip

The entered code will be executed *after* the execution of the *output tab*. This may feel a bit counter-intuitive at first, but you should simply consider the code tab an alternative way of defining output data. The general recommendation is to use the output table to define simple assignments and the code tab if more extensive scripting is needed to calculate data.

REST Client Activity



The *REST Client Call Activity* element is located in the *Activity* drawer of the process editor palette.

Element Details

Use the REST Client Activity to invoke REST services.

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Request Tab

On this tab you can configure the call to the REST service.

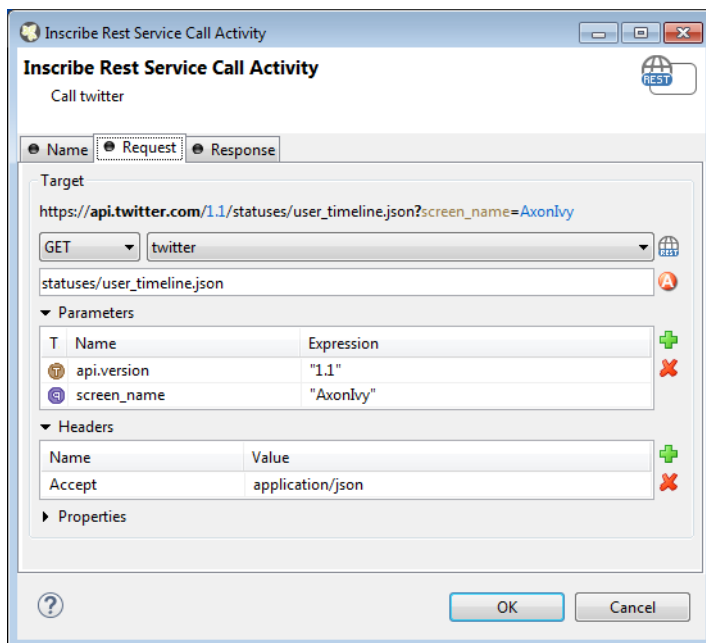


Figure 2.65. The REST Client Request tab

- Target
- **HTTP-Method:** The first combo lets you select the HTTP method to use. You can choose one of the well known methods like GET, POST, PUT or DELETE.
 - **REST-Client:** The second combo lets you pick a pre-configured “REST Clients Configuration”.

- **Path:** The text input can be used to define a resource-path. The provided path will be added to the base URI which is defined in the “REST Clients Configuration”. Use the attribute browser on the right side to insert dynamic parts to the URI.
- **Parameters:** Use this table to define query parameters that should be added to the URI. Or switch the type to 'template' in order to resolve a dynamic path template with a concrete value.

The parameter value is scriptable and can therefore contain process variables or other dynamic content.

Parameters		
T	Name	Expression
	api.version	"1.1"
	screen_name	"AxonIvy"

- **Headers:** Will be sent with the request and can be interpreted by the target service. For instance, many REST APIs can provide data in multiple serialization formats. By setting the `Accept` header, the preferred format can be propagated to the target service.

Any other HTTP-Header can also be configured. However, the `Authorization` header is easier to configure with an authorization feature on the “REST Clients Configuration”.

Headers	
Name	Value
Accept	application/json

- **Properties:** Are used to configure optional features or native properties of the REST client. They are globally configurable in the “REST Clients Configuration” properties. Here you can overwrite a property with dynamic values.

Properties	
Name	Expression
username	in.person.name
password	in.generatedPassword

Body For POST and PUT requests the body section can be used to specify that data that will be sent to the REST service.

- **Raw:** Define the Content-Type first and define any textual content in the editor part. The content can contain dynamic parts like process data fields. Use the action buttons left to the editor in order to insert a dynamic variable or function call.

Body

raw form entity Content-Type: application/json

```

1 {
2   "id" : 42,
3   "firstname" : "<%=in.name%>"
4 }

```

- **Form:** Send form values as content of type `application/x-www-form-urlencoded` The form values are scriptable.

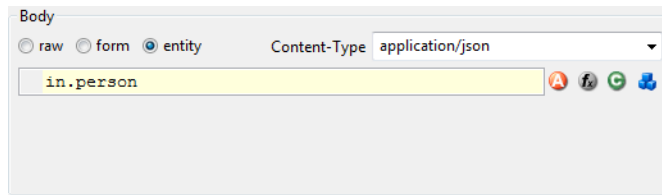
Body

raw form entity

Name	Expression
title	"I need a new car"
description	"really, I'm sick of my old Fiat Punto"

- **Entity:** Send a complex object as serialized text to the remote REST service. Most Java objects should be serializable as JSON (`application/json`) without additional configuration.

The serialization behaviour can be configured for special needs via properties on the client. See ??? [357]



Response Tab

On this tab you can consume response from the REST service.

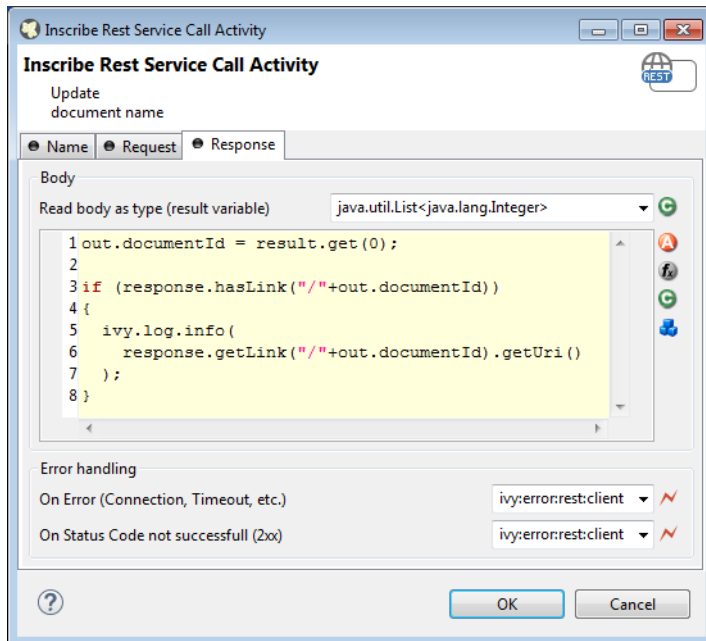


Figure 2.66. The REST Client Response tab

- | | |
|----------------|---|
| Body | <ul style="list-style-type: none"> • Result-Type: The first combo defines how the response entity will be read. Pick a Java type that can be mapped to response entity. The entity object is available in the 'result' variable.

See the chapter below (“JSON to Java”) for a quick comparison of response body mapping solutions.

The deserialization from JSON to a Java object can be customized with properties on the client. See ??? [357] • Code: Use the code editor to handle the response or its entity. In most cases, you only need to assign the 'result' variable to your process data. However, in this editor the JAX-RS 'response' variable is also available which lets you access the HTTP-status-code and other details of the HTTP response. |
| Error handling | <ul style="list-style-type: none"> • On Error: Choose the Error Code to throw if the REST client fails with an exception. This is typically the case if a connection or timeout problem exists. Pick '>> Ignore Error' to continue the process execution even though the REST service call failed with an exception. • On Status Code not successful: Fail automatically with an Error Code if the HTTP response status code is not in the 200 family. Pick '>> Ignore Error' if other status codes are valid and expected. |

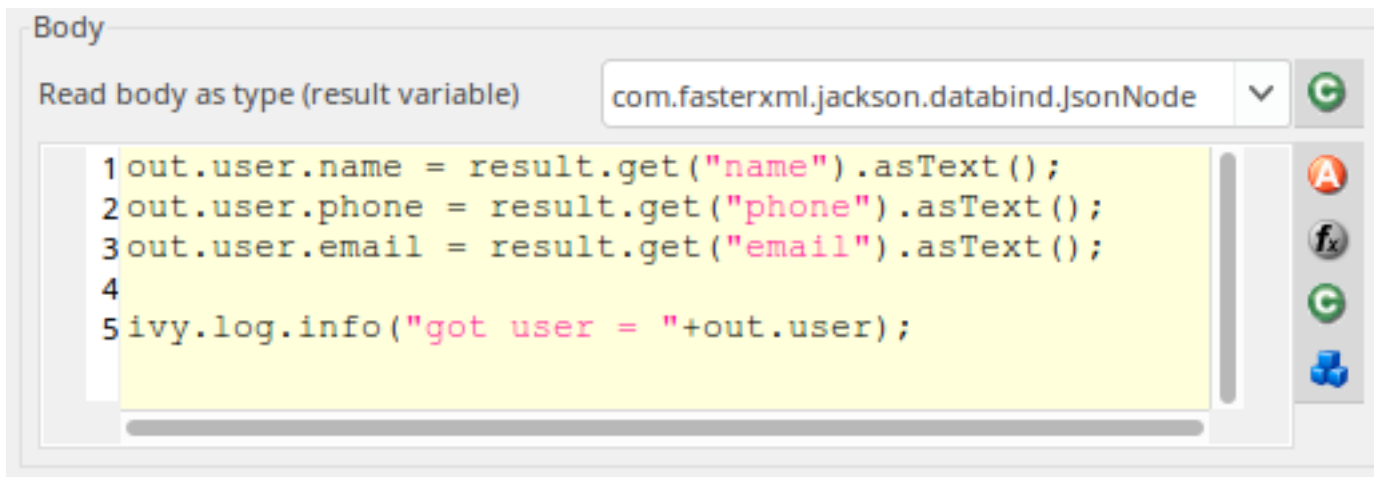
JSON to Java

The mapping of a JSON response body to a Java object is a simple task. Think of a service that returns a complex JSON. E.g:

```
{
  "id": 1,
  "name": "Leanne Graham",
  "username": "Bret",
  "email": "Sincere@april.biz",
  "address": {
    "street": "Kulas Light",
    "suite": "Apt. 556",
    "city": "Gwenborough",
    "zipcode": "92998-3874",
    "geo": {
      "lat": "-37.3159",
      "lng": "81.1496"
    }
  },
  "phone": "1-770-736-8031 x56442",
  "website": "hildegard.org",
  "company": {
    "name": "Romaguera-Crona",
    "catchPhrase": "Multi-layered client-server neural-net",
    "bs": "harness real-time e-markets"
  }
}
```

You can handle this complex JSON object with one of these solutions:

1. **Map to JSON Node:** Read the result body as `JsonNode` object. Navigate through the object tree and read its field values manually.



2. **Map to Data Class:** Create a Data Class with the attributes you need in the business process. Read the result body with this Data Class. Every attribute that matches by name (case sensitive) with an attribute in the JSON object will be mapped. Assign the `result` object to an attribute of your process data.

G User ✕

IvyScript class 'User' (com.axonivy.connectivity.rest)

▼ Class

Specify the properties for the whole class.

Comment:

Annotations: BusinessCaseData

Attributes

	Name	Type	Persistent	Comment
A	name	String	<input type="checkbox"/>	
A	email	String	<input type="checkbox"/>	
A	phone	String	<input type="checkbox"/>	

Body

Read body as type (result variable) com.axonivy.connectivity.rest.User ▼ G

```

1 out.user = result;
2
3 ivy.log.info("got user = "+out.user);
```

A
fx
G
📄

3. **Map to Generated Class:** Paste the JSON you receive from the service into a Java object source generator like <http://www.jsonschema2pojo.org/>. Generate the Java sources for the JSON structure. Download the sources and add them to a special source folder (E.g. `src_generated`). Now you can read the response body to an object of this generated class.

jsonschema2pojo - Mozilla Fire

www.jsonschema2pojo.org 80% Search

jsonschema2pojo

Star 2,817

Generate Plain Old Java Objects from JSON or JSON-Schema

```
1 {
2   "id": 1,
3   "name": "Leanne Graham",
4   "username": "Bret",
5   "email": "Sincere@april.biz",
6   "address": {
7     "street": "Kulas Light",
8     "suite": "Apt. 556",
9     "city": "Gwenborough",
10    "zipcode": "92998-3874",
11    "geo": {
12      "lat": "-37.3159",
13      "lng": "81.1496"
14    }
15  },
16  "phone": "1-770-736-8031 x56442",
17  "website": "hildegard.org",
18  "company": {
19    "name": "Romaguera-Crona",
20    "catchPhrase": "Multi-layered client-server",
21    "bs": "harness real-time e-markets"
22  }
23 }
```

Package: com

Class name: User

Source type:

JSON Schema

Jackson 2.x

Gson

Generate bo

Use primitiv

Use long int

The screenshot shows an IDE window with two main panes. The left pane displays a project tree for 'Axon.ivy Projects' with a sub-project 'CMS'. Under 'CMS', there is a folder 'src_generated' containing several Java files, with 'User.java' selected. The right pane shows the source code of 'User.java', which is a generated class with the following code:

```

26 })
27 @Generated("http://www.j
28 public class User {
29
30     @JsonProperty("id")
31     private Integer id;
32     @JsonProperty("name")
33     private String name;
34     @JsonProperty("usern
35     private String usern
36     @JsonProperty("email
37     private String email
38     @JsonProperty("addre
39     private Address addr
40     @JsonProperty("phone
41     private String phone
42     @JsonProperty("websi
43     private String websi
44     @JsonProperty("compa
45     private Company comp
46     @JsonIgnore
47     private Map<String,
48
49     @JsonProperty("id")

```

Below the IDE window, there is a 'Body' section with a dropdown menu set to 'com.typicode.jsonplaceholder.User'. Below the dropdown is a text area containing the following code:

```

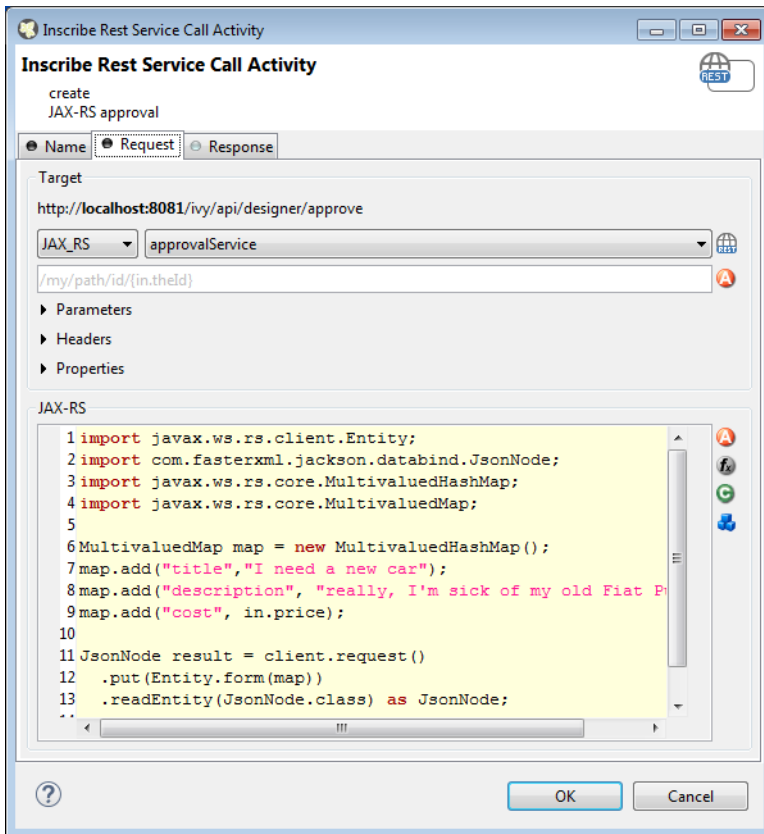
1 out.userGenerated = result;
2
3 ivy.log.info("got user = "+result);

```

Customization

The inscription mask provides a handy UI that makes most calls to a REST service very simple. However, there are always corner cases where you need to configure something, which is not configurable on the UI. In these rare cases, you can use the fluent JAX-RS API to call the service and interpret the response. To do so you can choose 'JAX_RS' as HTTP Method. In the scripting field that became visible you can configure every detail of the REST request done by this element.

In the scripting field, the variable `client` holds the REST client chosen in the *Target* section. The whole setup from the *Target* section will be applied to this client variable.



Call from Java

Rest Client calls can also be executed via Public API without using the Rest Client Activity.

The entry point to access Rest Clients is `Ivy.rest()`. The returned object is an instance of `javax.ws.rs.client.WebTarget` which is pre-configured as defined in the “REST Clients Configuration”. It provides fluent API to call the remote REST service.

Sample

```
// retrieve pre-configured rest service client
WebTarget client = Ivy.rest().client("myServiceName");

// GET request to receive a simple string
String token = Ivy.rest().client(UUID.fromString("e00c9735-7733-4da8-85c8-6413c6fb2cd3"));

// POST request to send a complex object
Ivy.rest().client("crmService").request().post(javax.ws.rs.client.Entity.json(myPerson));
```

Re-use configuration

If you notice that you configure precisely the same thing on multiple Rest Client Activities you can reduce this duplication.

Instead of applying the configuration multiple times, it can be set globally on the “REST Clients Configuration”. Almost any aspect of a Rest Client call can be configured by implementing a custom feature (`javax.ws.rs.core.Feature`). Our authorization feature can be taken as an example: `ch.ivyteam.ivy.rest.client.authentication.HttpBasicAuthenticationFeature`.

E-Mail Step



The *E-Mail Step* element is located in the *Activity* drawer of the process editor palette.

Element Details

This element allows to send e-mails out of processes (e.g. for information or alert purposes). The general configuration must be set in the E-Mail preferences for the Axon.ivy Designer and in the *Engine Administration* for the Axon.ivy Engine.

With the default E-Mail preferences mails will be sent to a development SMTP server that quickly shows you the mailboxes in the 'Mail Messages' view of the Designer.



Note

In the designer the sender (from) and the recipient (to) are always taken from the E-Mail Preferences, so you can easily test whether the mails are sent correctly by using your own e-mail address (or a dedicated test address)

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Tab Header

In this tab the e-mail header is defined. You can use the CMS and the process data (the **In** variable) to compose the header fields.

Figure 2.67. The Header Tab

Subject	The title of the e-mail to send.
From	The sender of the e-mail (always use a valid e-mail address).
Reply to	The e-mail address which is used by most e-mail clients when the reader clicks on "Reply" or "Reply all". Always use a valid e-mail address.

To	The recipient(s) of the e-mail. Multiple recipients can be separated by a comma or semi-colon.
CC	The recipient(s) in carbon copy of the e-mail. Multiple recipients can be separated by a comma or semi-colon.
BCC	The recipient(s) in blind carbon copy of the e-mail. Multiple recipients can be separated by a comma or semi-colon.
Error	Is thrown whenever an error occurs during the execution of this element. The error can be handled by a catching “Error Start” or by an “Error Boundary Event”.

Tab Content

In this tab the e-mail content is defined.

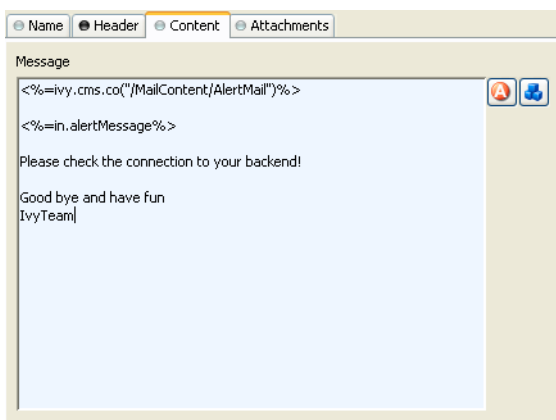


Figure 2.68. The Content Tab

Message The text of the e-mail. Use the CMS to have messages in multiple languages.



Tip

Start your message with an `<HTML>` tag to let you define your whole message in HTML format. (of course at the end of message an `</HTML>` is expected)

Tab Attachments

In this tab you can attach files to your e-mail. Each attachment line below on the screenshot represents one file. You can choose a file directly, take a process attribute with the type *File*, select a CMS entry or even build up the filename using script(s). The provided reference will be searched as CMS entry first, if no CMS entries found then the system will search the name as file in your Axon.ivy file area.



Note

CMS entry names do not have an extension (meanwhile filenames used to have one) so that the lookup order should cause no file overlapping.

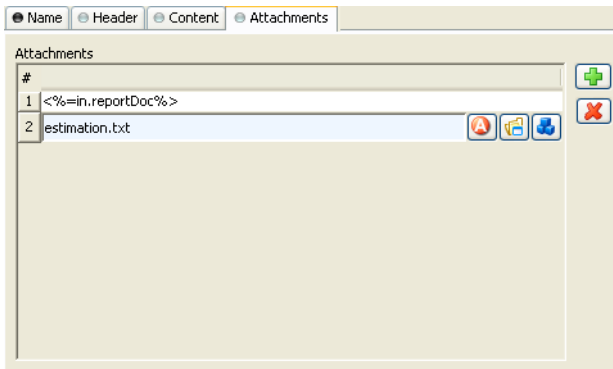


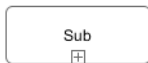
Figure 2.69. The Attachment Tab



Tip

Right click on a file input line to access further commands.

Embedded Subprocess



The *Subprocess* element is located in the *Activity* drawer of the process editor palette.

Element Details

An embedded subprocess folds a part of a process into a box. This makes hierarchical structuring of the process model possible. Sub processes are obtained top down or bottom up. Either by selecting and wrap parts of a process or by inserting an (initially empty) embedded sub element from the palette.

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Call Sub



The *Call Sub* element is located in the *Activity* drawer of the process editor palette.

Element Details

The Call Sub element allows to embed a process (independent subprocess) into an other. This is like jumping from the main process into the called sub process, execute the sub process and afterwards jump back. Process data attributes from the main process are mapped to parameters for the called sub process and the called sub process will return result parameters back to the main process.



Note

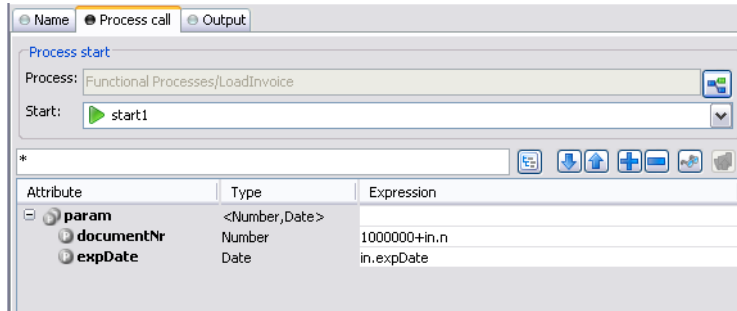
The input and result parameters of the called process are defined on the start element of the called process.

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Process Call Tab



In this tab you choose the process to be called and map process data attributes to the input parameters of the called process. You can use any IvyScript expression in the parameter mapping.

Output Tab

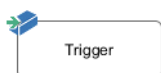
On this tab you can configure the output of the element (i.e. the data that leaves the element). See Output Tab for a more detailed description.

The variable `result` contains the output parameters that are returned by the called sub process (according to its interface definition).

E.g. if the called process returns a `String errorMessage` and an `Employee` object `employee` then the variable `result` will have two fields: `errorMessage` and `employee`, respectively. You can map those fields to the attributes of the outgoing process data:

```
out.msg = result.errorMessage is initialized
? ("An error occurred during selection: "
+ result.errorMessage)
: "";
out.selectedEmployee = result.employee;
```

Trigger Step



The *Trigger Step* element is located in the *Activity* drawer of the process editor palette.

Element Details

With the Trigger element its possible to start a new workflow. The trigger element triggers a Request Start element, which has an enabled triggered start mechanism. On call, the trigger element creates a case and a task with the defined configuration on the Request Start element. The new created task is returned to the Trigger element.

On call, after the creation of the new case and task, the workflow goes ahead through the process. When the created task starts (some time later, by user interaction or automatically by the system), the process starts at the *Triggered Start* element.

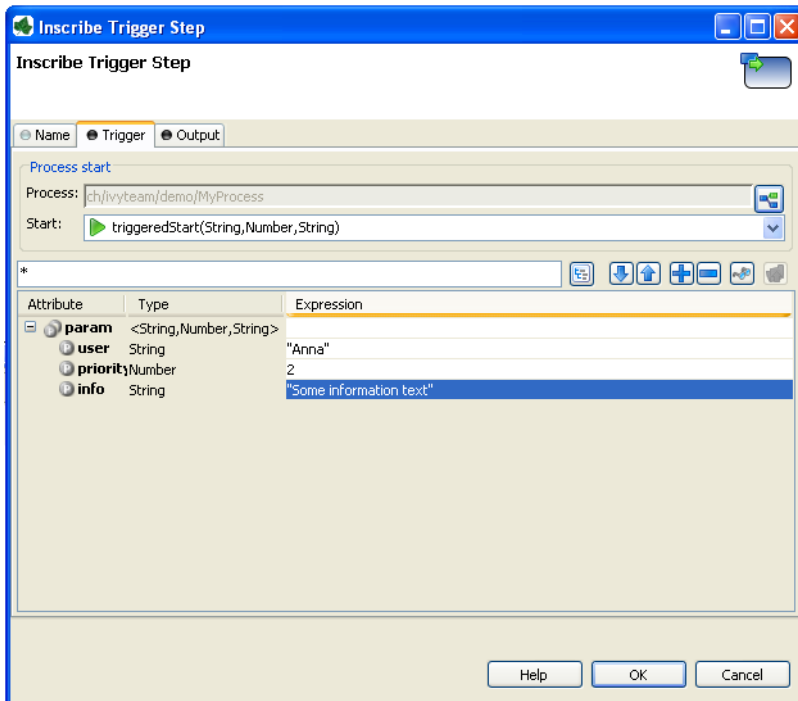
Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Trigger Tab

On this tab you can configure the Start Signature and the mapping of input parameter to the process data. The Start Signature is defined by its name and its parameter type and order.

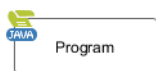


Output Tab

On this tab you can configure the output of the element (i.e. the data that leaves the element). See Output Tab for a more detailed description.

The variable `result` contains the created task that are returned by the triggered Request Start.

PI (Programming Interface) Step



The *Program Interface Step* element is located in the *Activity* drawer of the process editor palette.

Element Details

This element is one of Axon.ivy facilities to integrate custom-made software, legacy systems, proprietary applications or any other external system through a Java interface. The Program Interface element will instantiate a Java class that must implement the interface `IUserProcessExtension` and will call the method `perform` each time a process comes to the Program Interface. The common way to implement a Program Interface bean is to extend the abstract base class

`AbstractUserProcessExtension`. The interface also includes an inner editor class to parametrize the bean. You will find the documentation of the interface and the abstract class in the Java Doc of the Axon.ivy Public API.



Note

Since 3.x this element has become somewhat obsolete, since it has become very easy to create and call your own Java classes from IvyScript. However, the PI element still provides a standardized interface to a third party Java class and can provide a custom made editor for parametrization.

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Tab PI

On this tab you set the Java class which implements the interface `IUserProcessExtension` which is called when the PI step is executed. Furthermore, you can define exception handlers to react on errors such as not reachable systems, insufficient privileges and many more.

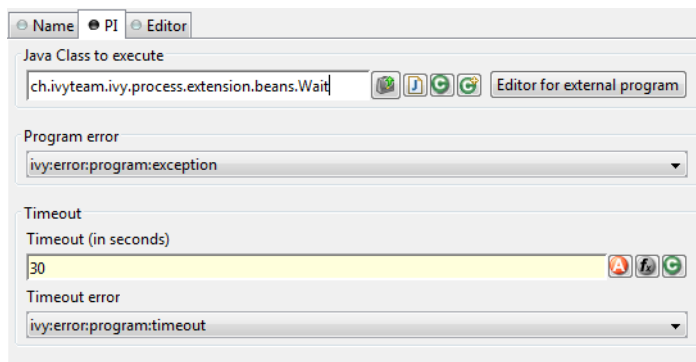



Figure 2.70. The PI tab

Java Class to Execute

The fully qualified name of the PI Java class implementing `IUserProcessExtension`. You can use default **copy & paste** commands, open a Java Type Browser to search for the class or you use the predefined `wait` class which just waits for a given period of time.

Use the New Bean Class Wizard () to create a new Java source file with an example implementation of the bean class.



Tip

You can add a graphical configuration editor for the Java call (i.e. setting the parameter values) on the PI inscription mask. See section Tab Editor for more details.

Program error

Occurs whenever an exception is thrown during the execution of the class. The error can be handled by a catching “Error Start”.

Timeout

Sets a timeout for the return call to the Java PI class.

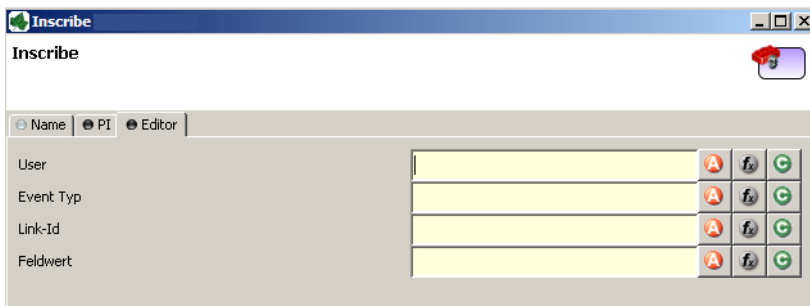
Timeout error

Occurs when the timeout is reached. The error can be handled by a catching “Error Start”.

Tab Editor

This tab displays the editor, that can be integrated in the external Java bean of the process element. The editor is implemented as an inner public static class of the Java bean class and must have the name `Editor`. Additionally the editor class must implement the `IProcessExtensionConfigurationEditorEx` interface. The common way to implement the editor class is to extend the abstract base class `AbstractProcessExtensionConfigurationEditor` and to override the methods `createEditorPanelContent`, `loadUiDataFromConfiguration` and `saveUiDataToConfiguration`. The method `createEditorPanelContent` can be used to build the UI components of the editor. You can add any AWT/Swing component to the given `editorPanel` parameter. With the given `editorEnvironment` parameter, which is of the type `IProcessExtensionConfigurationEditorEnvironment`, you can create text fields that support `ivyScript` and has smart buttons which provide access to the process data, environment functions and Java classes.

Here is an example of such an editor:



As you can see, the editor provides you access to any process relevant data, which can be used by your own process elements. For instance, you can easily transfer process data to your legacy system.

The following part shows the implementation of the editor shown above. As mentioned above `Axon.ivy` provides the `IIvyScriptEditor` which represents a text field with `ivyScript` support and smart buttons. Inside `createEditorPanelContent` use the method `createIvyScriptEditor` from the `editorEnvironment` parameter to create an instance of such an editor. Use the `loadUiDataFromConfiguration` method to read the bean configuration and set them to the UI components. Inside this method you can use the methods `getBeanConfiguration` or `getBeanConfigurationProperty` to read the bean configuration. Use the method `saveUiDataToConfiguration` to save the data in the UI components to the bean configuration. Inside this method you can use the methods `setBeanConfiguration` or `setBeanConfigurationProperty` to save the bean configuration.

```
public static class Editor extends AbstractProcessExtensionConfigurationEditor
{
    private IIvyScriptEditor editorUser;
    private IIvyScriptEditor editorEventTyp;
    private IIvyScriptEditor editorLinkId;
    private IIvyScriptEditor editorFieldValue;

    @Override
    protected void createEditorPanelContent(Container editorPanel,
        IProcessExtensionConfigurationEditorEnvironment editorEnvironment)
    {
        editorPanel.setLayout(new GridLayout(4,2));
        editorUser = editorEnvironment.createIvyScriptEditor(null,null, "String");
        editorEventTyp = editorEnvironment.createIvyScriptEditor(null,null, "String");
        editorLinkId = editorEnvironment.createIvyScriptEditor(null, null, "String");
        editorFieldValue = editorEnvironment.createIvyScriptEditor(null, null);

        editorPanel.add(new JLabel("User"));
        editorPanel.add(editorUser.getComponent());
        editorPanel.add(new JLabel("Event Typ"));
        editorPanel.add(editorEventTyp.getComponent());
        editorPanel.add(new JLabel("Link-Id"));
        editorPanel.add(editorLinkId.getComponent());
    }
}
```

```

        editorPanel.add(new JLabel("Feldwert"));
        editorPanel.add(editorFieldValue.getComponent());
    }

    @Override
    protected void loadUiDataFromConfiguration()
    {
        editorUser.setText(getBeanConfigurationProperty("User"));
        editorEventTyp.setText(getBeanConfigurationProperty("EventTyp"));
        editorLinkId.setText(getBeanConfigurationProperty("LinkId"));
        editorFieldValue.setText(getBeanConfigurationProperty("Feldwert"));
    }

    @Override
    protected boolean saveUiDataToConfiguration()
    {
        setBeanConfigurationProperty("User", editorUser.getText());
        setBeanConfigurationProperty("EventTyp", editorEventTyp.getText());
        setBeanConfigurationProperty("LinkId", editorLinkId.getText());
        setBeanConfigurationProperty("Feldwert", editorFieldValue.getText());
        return true;
    }
}

```

At runtime you have to evaluate the IvyScript the user have entered into the ivy script editors. If you implement for example the `AbstractUserProcessExtension` class there is a `perform` method which is executed at runtime. At this point you want to access the configured data in the editor. The following code snippet show how you can evaluate the value of an `IIvyScriptEditor`. If you use the `IIvyScriptEditor` you only get the value by calling the `executeIvyScript` method of the `AbstractUserProcessExtension`.

```

public CompositeObject perform(IRequestId requestId, CompositeObject in,
    IIvyScriptContext context) throws Exception
{
    IIvyScriptContext ownContext;
    CompositeObject out;
    out = in.clone();
    ownContext = createOwnContext(context);

    String eventtyp = "";
    String linkId = "";
    String fieldValue = "";
    String user= "";

    user = (String)executeIvyScript(ownContext, getConfigurationProperty("User"));
    eventtyp = (String)executeIvyScript(ownContext, getConfigurationProperty("Event Typ"));
    linkId = (String)executeIvyScript(ownContext, getConfigurationProperty("Link-Id"));
    fieldValue = (String)executeIvyScript(ownContext, getConfigurationProperty("Feldwert"));

    // add your call here

    return out;
}

```

Complete Code sample

```

public class MyOwnPiBean extends AbstractUserProcessExtension {

    /**
     * @see ch.ivyteam.ivy.process.extension.IUserProcessExtension#perform(ch.ivyteam.ivy.process.engine.IRequestId,
     * ch.ivyteam.ivy.scripting.objects.CompositeObject,
     * ch.ivyteam.ivy.scripting.language.IIvyScriptContext)
     */
    public CompositeObject perform(IRequestId requestId, CompositeObject in,
        IIvyScriptContext context) throws Exception {

        IIvyScriptContext ownContext;
        CompositeObject out;
        out = in.clone();
    }
}

```

```

ownContext = createOwnContext(context);

String eventtyp = "";
String linkId = "";
String fieldValue = "";
String user= "";

StringTokenizer st = new StringTokenizer(getConfiguration(),"|");
if(st.hasMoreElements())
    user= (String)executeIvyScript(context, st.nextElement().toString());
if(st.hasMoreElements())
    eventtyp = (String)executeIvyScript(context, st.nextElement().toString());
if(st.hasMoreElements())
    linkId = (String)executeIvyScript(context, st.nextElement().toString());
if(st.hasMoreElements())
    fieldValue = (String)executeIvyScript(context, st.nextElement().toString());

//do something with the values
return out;
}

public static class Editor extends JPanel implements IProcessExtensionConfigurationEditorEx {
    private IProcessExtensionConfigurationEditorEnvironment env;
    private IIvyScriptEditor editorUser;
    private IIvyScriptEditor editorEventTyp;
    private IIvyScriptEditor editorLinkId;
    private IIvyScriptEditor editorFieldValue;

    /**
     * Constructor for the Editor object
     */
    public Editor() {
        super(new GridLayout(4,2));
    }

    /**
     * Sets the configuration
     *
     * @param config the configuration as an String */
    public void setConfiguration(String config) {
        StringTokenizer st = new StringTokenizer(config,"|");
        if(st.hasMoreElements())
            editorUser.setText(st.nextElement().toString());
        if(st.hasMoreElements())
            editorEventTyp.setText(st.nextElement().toString());
        if(st.hasMoreElements())
            editorLinkId.setText(st.nextElement().toString());
        if(st.hasMoreElements())
            editorFieldValue.setText(st.nextElement().toString());
    }

    /**
     * Gets the component attribute of the Editor object
     *
     * @return this
     */
    public Component getComponent() {
        return this;
    }

    /**
     * Gets the configuration
     *
     * @return The configuration as an String
     */
    public String getConfiguration() {
        return editorUser.getText() + "|" + editorEventTyp.getText() + "|" +
            editorLinkId.getText() + "|" + editorFieldValue.getText() + "|";
    }

    /**
     * @return boolean
     */
    public boolean acceptInput() {
        return true;
    }

    public void setEnvironment(IProcessExtensionConfigurationEditorEnvironment env) {
        this.env = env;
        editorUser = env.createIvyScriptEditor(null,null, "String");
        editorEventTyp = env.createIvyScriptEditor(null,null, "String");
        editorLinkId = env.createIvyScriptEditor(null, null, "String");
        editorFieldValue = env.createIvyScriptEditor(null, null);

        add(new JLabel("User"));
        add(editorUser.getComponent());
        add(new JLabel("Event Typ"));
        add(editorEventTyp.getComponent());
        add(new JLabel("Link-Id"));
        add(editorLinkId.getComponent());
        add(new JLabel("Feldwert"));
        add(editorFieldValue.getComponent());
    }
}

```

}

Note



The *Note* element is located in the *Activity* drawer of the process editor palette.

Element Details

The note element

Inscription

Tab Name

An Annotation enables comments to be inserted anywhere within a model for documentation purposes

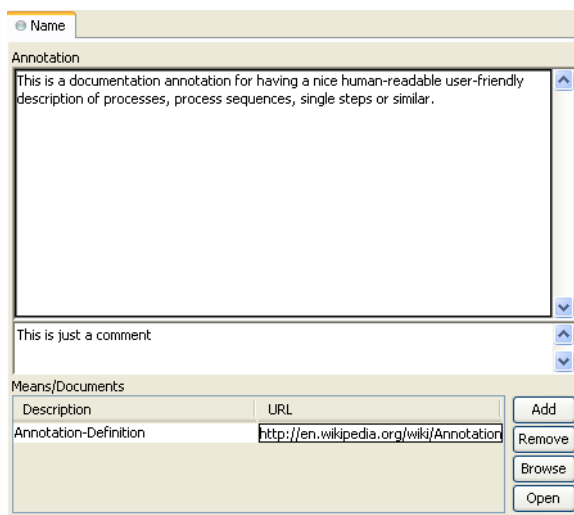


Figure 2.71. The Name Tab

Annotation	The text of the annotation. This is displayed on the process diagram for your documentation.
Comment	In this text field the function of the element is described. This text appears as Tool Tip whenever the mouse stays over the element.
Means/Documents	This table lists the means being used and which documents are available at with location (given as an URL).



Tip

In generated HTML reports, a link is inserted for these document references.

Web Service Process Start



WS Start

The *WS Start* element is located in the *WS Process* drawer of the process editor palette and only available in web service processes.

Element Details

Each Web Service Start element will create a web service operation in the web service where it is located. It has input and output parameters.

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Start Tab

This tab is used to define the signature of the web service operation.

The screenshot shows the 'Start Signature' tab in a software interface. It contains the following elements:

- Start Signature**
 - Signature: normalImport(customer)
 - Name: normalImport
- Input parameters**

Name	Type
customer	customer
- Mapping of input parameters to process data**

Attribute	Type	Expression
out	CustomerProcessData	
customer	customer	param.customer
insertedDatabaseId	Number	
ok	Boolean	

Figure 2.72. The Start Tab

Start signature

The name text field allows you to specify the name of the web service operation. This is the name that will also appear in the generated WSDL and will be used to call the web service operation.

Input parameters

This table is used to define the input parameters of the operation. The list may be left empty if the operation does not require any input parameters. To add a new parameter, click the green plus icon and specify the name and type of the parameter.



Warning

Some restrictions apply to the definition of Web Service process input parameters. Please follow the rules below:

Do not use the interface type `Number` as type for an input parameter. Instead use concrete types like `Integer` or `Double`.

Do not use complex types that contain a `List` attribute as input parameter (e.g. `Employee` with an attribute `projects`

of type `List<Project>`). Use a `java.util.List` (e.g. `java.util.List<Project>`) as type for such attributes instead.

In both cases you can still map the incoming values to process attributes of type `Number` or `List<?>` in the mapping section.

Mapping of input parameters

The input parameters defined above are available as fields on the `param` variable. You can assign the parameter values to the internal data fields in the table.



Note

The reason why you have to assign the incoming parameters to local data is to keep the implementation independent from the declaration. By doing so the implementation can be changed at a later point of time (rename data, use different data types, etc.) while at the same time the web service interface is kept stable. This has the effect that none of the clients of the web service have to be changed/adapted after an internal data change.

Result Tab

This tab is used to define the return parameters of the operation.

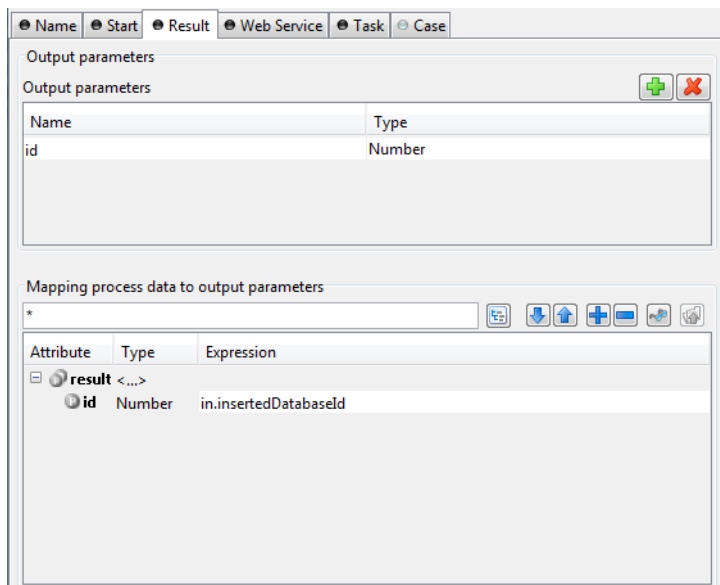


Figure 2.73. The Result Tab

Output Parameters

This table is used to define the output parameters of the operation. The list may be left empty if the operation does not return any data. To add a new parameter, click the green plus icon and specify the name and type of the parameter.

Mapping of process data

For each defined output parameter you must now specify the value that will be returned. In most cases, this is a process attribute. However you may specify any valid IvyScript expression.

Web Service Tab

This tab is used to change the web service specific settings of the operation.

Figure 2.74. The Web Service Tab

Web service The web service section shows the web service's name and authentication options. Click the *Configure...* button to open the configuration dialog. See inscription mask of the web service process for details.



Note

Since these settings are defined per web service and not per web service operation, any changes here will have an impact on all the operations within the same process, i.e. web service.



Tip

Use fully qualified class names to generate specific target namespaces in the WSDL of your web service (e.g. `ch.ivyteam.srv.CustomerService` as demonstrated on the screenshot above will result `targetNamespace="http://srv.ivyteam.ch/"` in your WSDL)

Responsible role You may specify a role that is required to call this start. If the start is invoked with a user not owning the selected role, an error will occur. The error can be handled by a catching “Error Start”.

Exception handling The exception handling allows your web service operation to throw a custom exception if your process could not complete normally. When such an exception is thrown, no output parameters are returned to the client.

After activating the exception handling, define the condition on which the exception should be thrown and the message to be returned to the caller.

Task Tab

This tab defines information relevant to the task. The task created for a web service call will normally not appear in a task list of a user. The values on this tab are therefore only relevant for analysing the finished tasks and not for the task list itself.

Entry in Task List	Defines the name and description of the task.
Priority	Here you select the priority of the task.

Tab Task - Business

This tab allows to set additional information to categorize the task created. The values set on this tab are only informational and have no effect on how this task is treated by Axon.ivy.

Business calendar

You can set the name of the business calendar that should be used for this task. In the context of this task `ivy.cal` will return this business calendar regardless of what you've set for the case.

For more information about business calendar administration see the engine guide.

For more information about business calendar usage see the Public API of `ch.ivyteam.ivy.application.calendar.IDefaultBusinessCalendar`.

Tab Task - Custom fields

This tab allows to set additional information for the task created. The values set on this tab are only informational and have no effect on how this task is treated by Axon.ivy.

Case Tab

On this tab you can configure the Case created by this Web Service Process Start. See “Case Tab” in the Task Switch Gateway element.

Customization

The Web Service endpoints are generated automatically in a Java file which contains JAX-WS annotations to define the Web Service. If the default configuration does not fit your needs, the generated Java file can be managed and extended by the developer.

The Java file is located in the folder `[project]/src_wsproc/[fully-qualified-name].java` and gets interpreted by the CXF library (<http://cxf.apache.org/>). The file has to be moved to the `src`-folder of the project and has to be in line with the configuration of the WS Start Elements of the process. The `fully-qualified-name` is defined in the inscription mask of the process.

The Java file in the `src`-folder is under control of the Developer. When a WS Start element configuration changes, the change has to be adapted manually in the Java file.

User Dialog Start



The *User Dialog Start* element is located in the *User Dialog* drawer of the process editor palette.

Element Details

The User Dialog Start element is used to map a *Start Method* (as declared in the Interface Editor) to the process that is started by this element and that implements the Start Method.

The element allows to initialize internal data of the User Dialog from the *input* parameters of the call and to define *return* values from process data (to be returned when the User Dialog finishes).



Note

With Rich Dialogs on the User Dialog Start, embedded inner Rich User Dialogs are initialized by calling individual start methods recursively.

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Start Tab

This tab is used to select the signature of the *Start Method*.

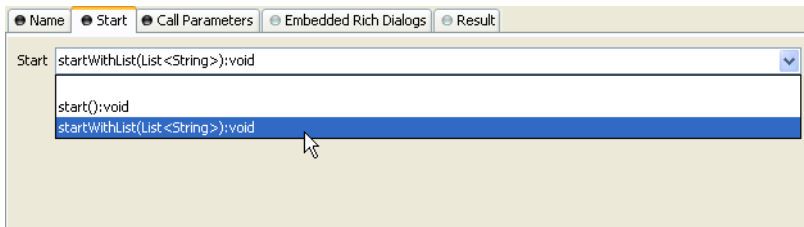


Figure 2.75. The Start Tab

Start The drop down list allows you to select one of the start methods declared in the interface of the User Dialog. Only methods that have not yet been mapped (i.e. assigned to start elements) are available for selection.



Tip

If the drop down list is empty, then all methods have already been assigned. The additional start element is therefore useless and you can delete it.

Output Tab

This tab is used to map the incoming call parameters to the User Dialog's internal process data.

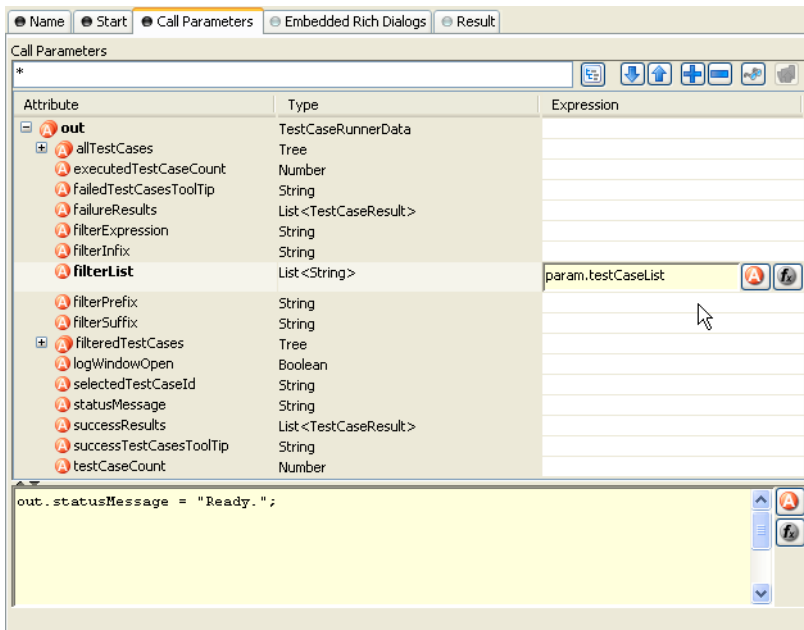


Figure 2.76. The Output Tab

Call Parameters

The call parameters are available as fields on a `param` variable. You can assign the parameter values to the internal data fields in the table or initialize them with the scripting field below the table. If both the table and the scripting area are used for assignment, then the table's assignments will be executed first.



Note

The reason why you have to assign the incoming parameters to local data is to keep the implementation independent from the declaration. The mapping of parameters serves as a flexible adapter mechanism: By doing so the implementation can be changed at a later point of time (rename data, use different data types, etc.) while at the same time the interface is kept stable. This has the effect that none of the clients of the Rich Dialog have to be changed/adapted after an internal data change.



Tip

The variable `panel` is not available on the *Start Method* element for technical reasons: At the execution time of this element the panel is not yet completely initialized. Therefore access to its components is considered potentially dangerous and prohibited on purpose.

Temporary disabled UI events (only available for Rich Dialogs)

If selected, all UI events (like `SELECTION_CHANGED`, `VALUE_CHANGED` or `LIST_SELECTION`) are disabled until the start-process reaches its end-step. All disabled events are listed in the enumeration `UiEventKind`.

- Other rich dialogs (implicitly inner Rich Dialogs) are not affected.
- The data binding will be executed before the re-enablement.



Note

You have the possibility to disable UI events permanently or temporarily by the API. See the public API,

```
ivy.rd.disableUiEvents(), ivy.rd.enableUiEvents()
and ivy.rd.disableUiEventsTemporary().
```



Tip

In most cases the panel data (like combo boxes etc.) are fully initialized on startup, for example when you get the whole data from a Web Service etc. But when widget values should be changed when another widget changes, e.g. on value change. In such cases the events could be disabled to speedup the startup behaviour of a panel, because the panel does not have to refresh its data again for each dependency event on the panel at startup.

Embedded Rich Dialogs Tab (only available with Rich Dialogs)

This tab is used to initialize the embedded Rich Dialogs within a Rich Dialog.

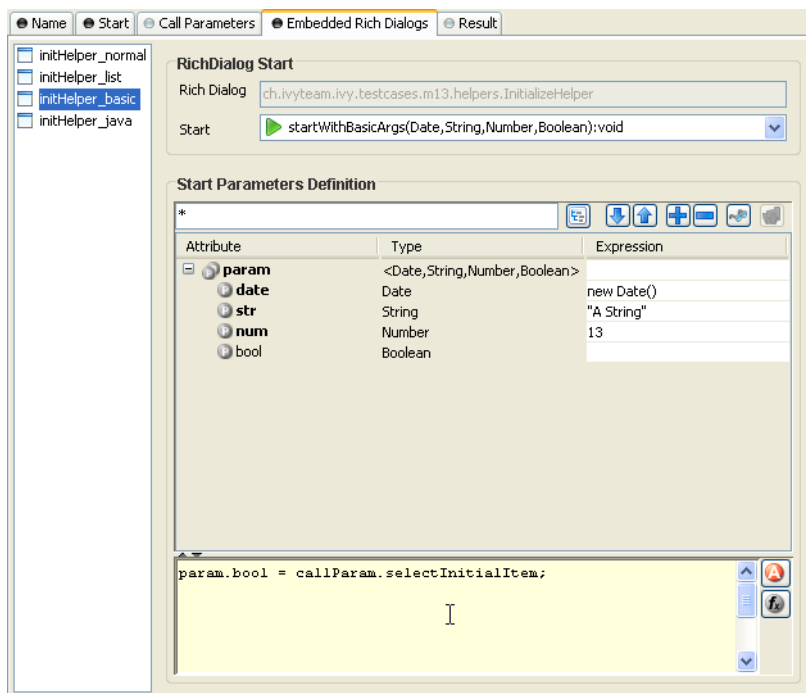


Figure 2.77. The Embedded Rich Dialogs Tab

Embedded Rich Dialogs

The embedded Rich Dialog instances can be initialized individually. The column on the left shows all embedded Rich Dialogs that are located on the configured Rich Dialog's panel. For each of them an individual Start Method may be chosen, just as described for the Rich Dialog Element.



Tip

In the context of this tab the `param` variable is reserved for the arguments of the called start method for this Rich Dialog.

If you want to use incoming call parameters values as arguments for the start methods of embedded Rich Dialogs then you can use the `callParam` variable as demonstrated above.

Result Tab

This tab is used to define the return values of the User Dialog.

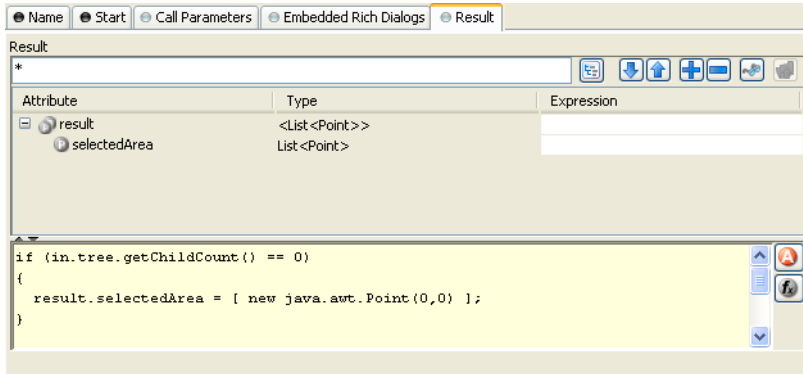


Figure 2.78. The Result Tab

Result Return values can be defined according to the declared return types that the mapped *Start Method* specifies. The table on this tab shows a `result` variable which has fields for each declared return type (none, if the return type is `void`).

You can define the returned result values either in the table or by using the IvyScript field below it. The assignments of the table will be executed before the script.



Tip

The defined result object will become available to the caller on the *Output tab* of the invoking User Dialog Element once the User Dialog has finished.

User Dialog Method Start



UD Method Start

The *User Dialog Method Start* element is located in the *Rich Dialog* drawer of the Process editor palette.

Element Details

The User Dialog method element is used to map a *User Dialog method* (as declared on the User Dialog's interface) to a process that implements the functionality of that method.

The element allows to set internal data of the User Dialog from the `input` parameters of the method call and to define *return* values either from process data or by calculation.

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Method Tab

This tab is used to select the signature of the *User Dialog Method* that is implemented by this element's process.

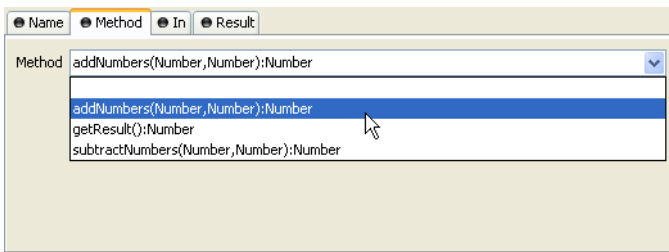


Figure 2.79. The Method Tab

Method

The drop down list allows you to select one of the regular methods declared in the interface of the User Dialog. Only methods that have not yet been mapped (i.e. assigned to method start elements) are available for selection.



Note

If the drop down list is empty, then all methods have already been assigned. The additional method element is therefore useless and you can delete it.

Temporary disabled UI events

If selected, all UI events (like `SELECTION_CHANGED`, `VALUE_CHANGED` or `LIST_SELECTION`) are disabled until the method-process reaches its end-step. All disabled events are listed in the enumeration `UiEventKind`.

- Other User Dialogs (implicitly inner RD's) are not affected.
- The data binding will be executed before the re-enablement.



Note

You have the possibility to disable UI events permanently or temporarily by the API. See the public API, `ivy.rd.disableUiEvents()`, `ivy.rd.enableUiEvents()` and `ivy.rd.disableUiEventsTemporary()`.

Output Tab

This tab is used to assign the parameter values to the User Dialog's internal data fields.

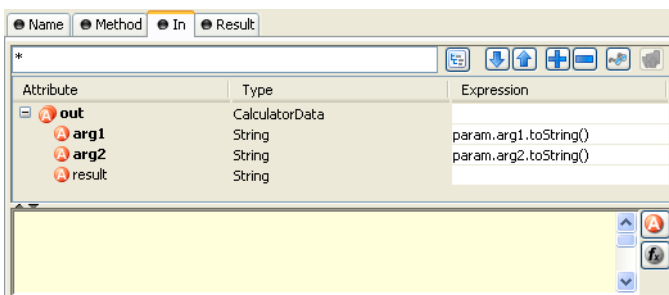


Figure 2.80. The Output Tab

Mapping Both the table's assignments and any scripting code below the table will be executed at the time of the method call. The table's statements will be executed before the scripting block.

The input parameters are available as fields on a `param` variable (none if the chosen method does not declare any input parameters). You can assign values to any internal data fields, the assignments do not have to be based on `param`.

Result Tab

This tab is used to define the values that will be returned to the caller when the method process finishes.

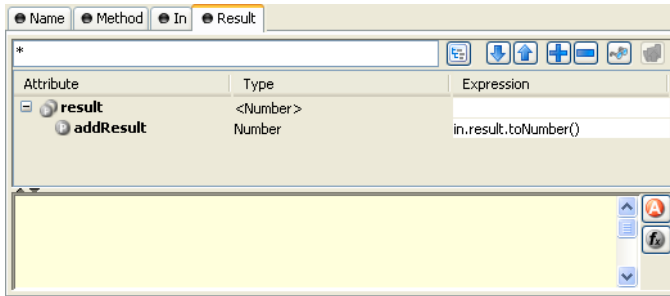


Figure 2.81. The Result Tab

Method The method's declared return parameters are shown in the table as fields of a `result` variable (none if the method's return value is `void`).

Both the table's assignments and any scripting code below the table will be executed when the method process reaches an User Dialog End Element. The table's statements will be executed before the scripting block.

User Dialog Event Start



UD Event Start

The *User Dialog Event Start* element is located in the *Rich Dialog* drawer of the process editor palette.

Element Details

The RD Event Start element represents a process start in the User Dialog logic that is triggered by means of UI event mapping. Events that are fired either by widgets or by embedded Rich Dialogs when the user interacts with the UI may result in the execution of UI processes, if a matching event mapping exists.

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Output Tab

On this tab you can configure the output of the element (i.e. the data that leaves the element). See Output Tab for a more detailed description.



Note

With Rich Dialogs, additionally to the regular variables of the *Output Tab* you have the following variables available:

event This variable contains an object of the type `java.util.EventObject` that caused the RD event process to start.

Depending on the type of event you may cast this object into different more specific event types. For example to `ch.ivyteam.ivy.richdialog.exec.RdEvent` if it is an event that was caused by an embedded User Dialog.

panel This variable contains this User Dialogs panel instance which you can use to set or query any widget properties. It is, however, recommended to use data binding instead of accessing/setting panel properties directly.

Code Tab

On this tab you can execute any IvyScript, e.g. to modify internal data or to define output data of this element. See Code Tab for a more detailed description.



Tip

The entered code will be executed *after* the execution of the *output tab*. Although this may seem a bit counter-intuitive at first, you should simply regard the code tab as an alternative way of defining output data. The general recommendation is to use the output table to define simple assignments and the code tab if more extensive scripting is needed to calculate data.

User Dialog Broadcast Start



RD Broadcast Start

The *User Dialog Broadcast Start* element is located in the *User Dialog* drawer of the process editor palette. It is only available in a Rich Dialog.

Element Details

The Broadcast Start element represents a process start in the Rich Dialog logic that is invoked when an *accepted broadcast* is received by the Rich Dialog (as declared on the Rich Dialog's interface).

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Broadcast Tab

This tab is used to select the *accepted broadcast event* that will trigger this element's process.

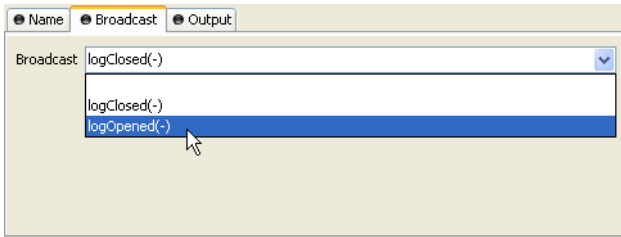


Figure 2.82. The Broadcast Tab

Broadcast The drop down list allows you to select one of the accepted broadcasts declared in the interface of the Rich Dialog. Only broadcasts that have not yet been mapped (i.e. assigned to broadcast start elements) are available for selection.



Tip

If the drop down list is empty, then all accepted broadcasts have already been assigned. The additional broadcast element is therefore useless and you can delete it.

Output Tab

On this tab you can configure the output of the element (i.e. the data that leaves the element). See [Output Tab](#) for a more detailed description.



Note

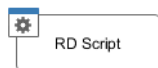
Additionally to the regular variables of the *Output Tab* you have the following variables available:

event This variable contains an object of the type `java.util.EventObject` that caused the RD event process to start.

You can cast the event object to the type `ch.ivyteam.ivy.richdialog.exec.RdEvent` in order to access the optional *event parameter* object that may have been passed along with the event.

panel This variable contains this Rich Dialogs panel instance which you can use to set or query any widget properties. It is, however, recommended to use data binding instead of accessing/setting panel properties directly. `gen`

User Dialog Script Step



The *User Dialog Script Step* element is located in the *User Dialog* drawer of the process editor palette.

Element Details

With this element you can perform any transformation of the process data within the User Dialog. In a Rich Dialog you have also access to the panel object.



Warning

It is strongly recommended to use the dedicated process elements if you intend to use specific functionality and/or technology (such as invoking Web Services, querying Databases and so on) as these elements encapsulate their use and handle exceptions internally.

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Output Tab

On this tab you can configure the output of the element (i.e. the data that leaves the element). See Output Tab for a more detailed description.



Note

In a Rich Dialog, additionally to the regular variables of the *Output Tab* you have the following variables available:

`panel` This variable contains this User Dialogs panel instance which you can use to set or query any widget properties. It is, however, recommended to use data binding instead of accessing/setting panel properties directly.

Panel Tab (only available with Rich Dialogs)

On this tab you can modify any properties of the Rich Dialog's panel. See Panel Tab for a more detailed description.



Tip

The modifications on the panel will be performed *after* the execution of the *output tab* and *before* the *code tab*.

Code Tab

On this tab you can execute any script, e.g. to modify internal data or to define output data of this element. See Code Tab for a more detailed description.



Tip

The entered code will be executed *after* the execution of the *output tab* and the *panel tab*. Although this may seem a bit counter-intuitive at first, you should simply regard the code tab as an alternative way of defining output data. The general recommendation is to use the output table to define simple assignments and the code tab if more extensive scripting is needed to calculate data.

User Dialog Fire Event Step



The *User Dialog Fire Event* element is located in the *User Dialog* drawer of the process editor palette. It is only available with Rich Dialogs.

Element Details

This element is used to fire events that are declared on the Rich Dialog's interface during process execution.

The range (i.e. the potential receivers) of the fired event is defined by the scope of the event and is part of the event's declaration (see there to learn about what types of scope that are available). A fired event may carry an optional parameter which can be retrieved by the receiving event handlers.

Events can be received in two ways:

If the fired event is a *broadcast* event then other Rich Dialogs may declare and implement an *accepted broadcast* that matches the fired event's signature.

If the event is a regular event (i.e. for *subscribers* only) then an outer Rich Dialog may map the fired event onto one of its Rich Dialog event start elements using the event mapping mechanism.



Tip

You can also fire Rich Dialog events programmatically by invoking the **fireXYZ(...)** methods on the Rich Dialog panel (*advanced visibility*). It is, however, recommended to use the *fire event* element instead due to the declarative nature of the element inside the process.

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

Event Tab

This tab is used to select the Rich Dialog event that should be fired.

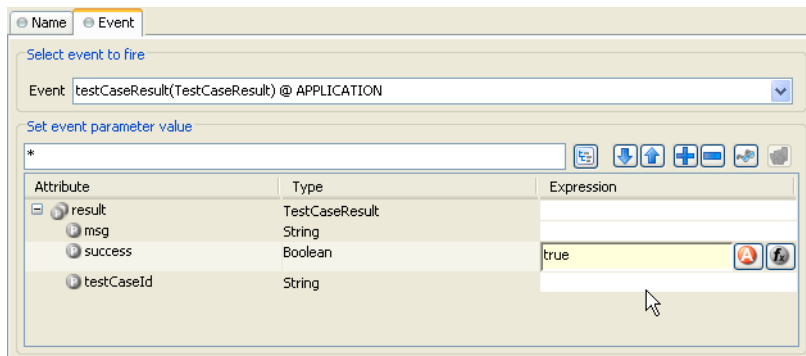


Figure 2.83. The Event Tab

Event To Fire The drop down list allows you to select one of the events that the Rich Dialog is able to fire (as declared in the interface of the Rich Dialog). The list shows all available events with their *name*, *type of event object* and *scope*.

Event Parameter Value Depending on the signature of the selected event, an optional event parameter may be attached to the event. The table in this section can be used to assign values to the fields of the specified parameter type (if available).

Leave the table empty if you don't want to set an event parameter (in which case the parameter value will stay *null*).



Warning

It is strongly recommended to use only *immutable* objects as event parameter. During the distribution of the fired event all receivers can access the parameter

object. If it is mutable and can thus be changed by one receiver then the object will contain different data when it is accessed by the next receiver, which may lead to unpredictable behavior.

User Dialog UI Synchronization



RD UI Synchronization

The *User Dialog UI Synchronization* element is located in the *User Dialog* drawer of the process editor palette. It is only available with Rich Dialogs.

Element Details

This element can be used to update the user interface on the client during a long running process on the server.

Problem: If a long running UI process is started on the server (e.g. by clicking on a button) then the user interface on the client will not be updated until the process on the server has finished, because the request will not return to the client until then. It doesn't matter whether the process performs changes on the UI (e.g. by setting a text on a label or by updating the value of a progress bar) during its execution, those changes will not be transmitted to the client until the request returns.

Solution: If you insert the *UI Synchronization* element at some points inside your long running process then the changes that were set on the UI are transmitted to the client. The currently running request will temporarily return, update the client's user interface, and then immediately come back and resume the long running process.

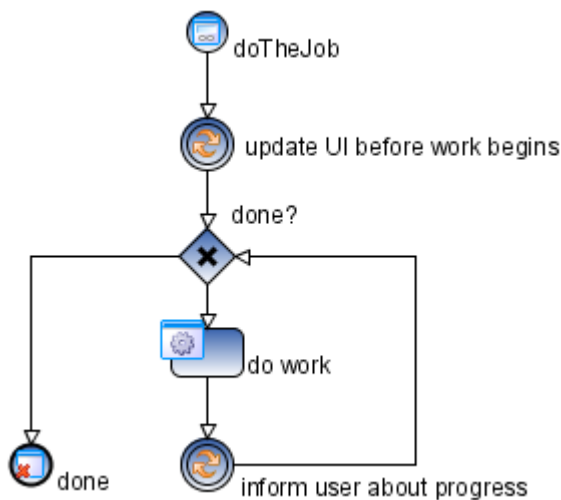


Figure 2.84. Example usage of the UI Synchronization element



Note

Use the data binding for UI update. Both an *upbinding* (i.e. *Data-to-UI*) and a *downbinding* (i.e. *UI-to-Data*) will be executed automatically every time when a Rich Dialog *UI Synchronization* element is executed.



Warning

Do not use the UI Synchronization element inside a Rich Dialog start method.

This will not work for technical reasons, because the initialization of the Rich Dialog will not have been completed at this point of time. If you want to execute a long running process immediately after a Rich Dialog is started then you should move this logic to an event process.

This process should then be triggered by invoking a hidden button (i.e. a button on the panel with the property `visible = false`) with `ivy.rd.clickDeferred(hiddenButton)`. The button will automatically be clicked on the client as soon as the current request has ended, thus immediately sending another request to the server. The following figure illustrates this:

Have a look at the *UI Refresh* demo (particularly the *Progress Dialog* part) in the *IvyDemos* project for an example on how to do this.

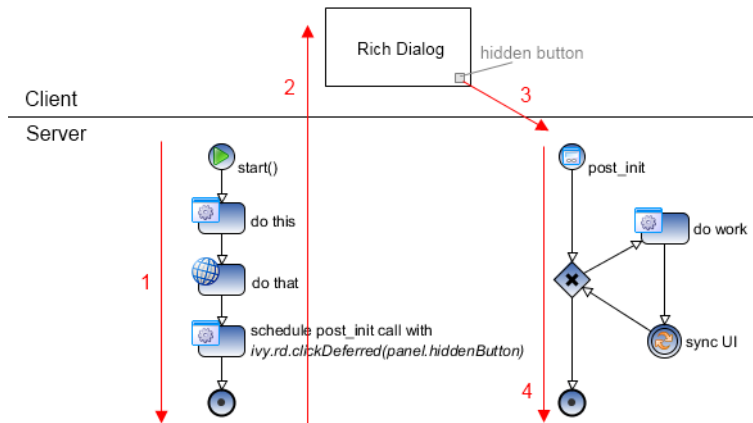


Figure 2.85. Example usage of `ivy.rd.clickDeferred(hiddenButton)`

Explanation: [1] Start method of Rich Dialog initializes the Rich Dialog. At the end of the init method a call to `ivy.rd.clickDeferred(...)` is used to schedule the clicking of a hidden button at a later point of time. [2] Rich Dialog is uploaded to the client and displayed. [3] Immediately after the Rich Dialog becomes visible, the scheduled click on the hidden button is executed and triggers another server round trip. [4] The `post_init` event handler (which is associated with the hidden button) is executed and performs the potentially long running task, where the *UI Synchronization* element may safely be used.

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

User Dialog Process End



The *User Dialog Process End* element is located in the *User Dialog* drawer of the process editor palette.

Element Details

This element is used to terminate any User Dialog processes inside the User Dialog's logic.



Note

An *up-binding* (i.e. *data-to-panel*) will be executed automatically every time when a User Dialog process end element is executed.

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

User Dialog Exit End



The *User Dialog Exit End* element is located in the *Rich Dialog* drawer of the process editor palette.

Element Details

This element is used to *terminate* the execution of a User Dialog.

If process execution reaches this element then the User Dialog's panel is immediately removed from the *display* container that it resides in. The *result tab* code of the originally invoked start method is executed and the calculated results are passed back to the User Dialog element which called the User Dialog. Afterwards the calling process continues.



Note

If the User Dialog was called *asynchronously* then the User Dialog's panel will simply be removed from the UI and the calculated result value of the earlier invoked start method will be ignored.

Inscription

Name Tab

This tab is included in the mask of all process elements and contains the name and a description of the element. See Name Tab for a more detailed description.

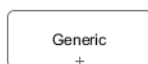
BPMN Activity Elements

The *BPMN Activity* drawer contains elements, that can be used to design a process at a high level, where details of the technical implementation are hidden inside the element itself. Thus BPMN Activities behave similar to a *Embedded Sub*, but their purpose is different.

They are intensively used by the importer of Axon.ivy Modeler Processes.

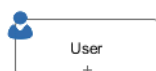
Available BPMN Activity Elements

Generic



An unspecific activity.

User



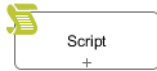
Indicates an activity that implies execution by the user.

Manual



Indicates an activity that implies execution without assistance of IT means.

Script



Indicates an activity that implies execution inside the process engine itself.

Receive



Indicates an activity that implies reception of a message.

Rule



Indicates an activity where a business rule is evaluated.

Send



Indicates an activity that implies sending of a message.

Service



Indicates an activity that implies calling an automated function.

Case Maps

Introduction

Case Maps can be used to split a long running process into multiple short running processes. The Case Map controls which processes are executed automatically in which order and which processes can be started manually by users.

New Case Map Wizard

Overview

The *New Case Maps Wizard* lets you create a new Case Map.

Accessibility

File > New > Case Map

Features

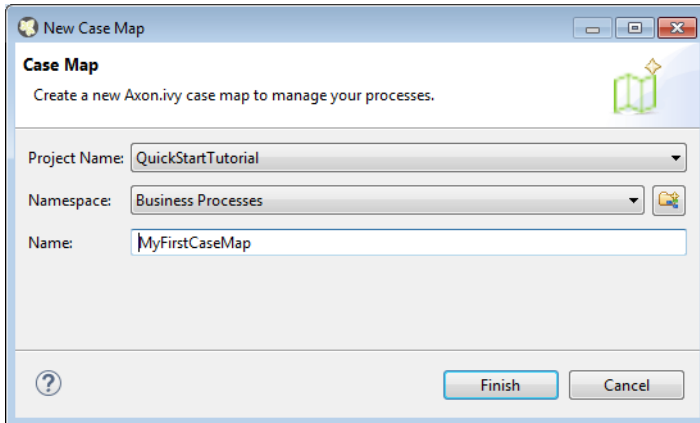


Figure 2.86. The New Case Map Wizard

- Project Name** Choose the project in which you want to create a new Case Map.
- Namespace** Select a process group where the new Case Map will be inserted. Select the <default> process group to create a Case Map directly below the project's processes folder. You can click on the group folder button to open the *New Process Group Wizard*, if you want to create a new group "on the fly". The process groups are listed relative to the project's *processes* folder.
- Name** Enter the name of the Case Map that you want to create.

Case Map Editor

Use the Case Map Editor to edit Case Maps.

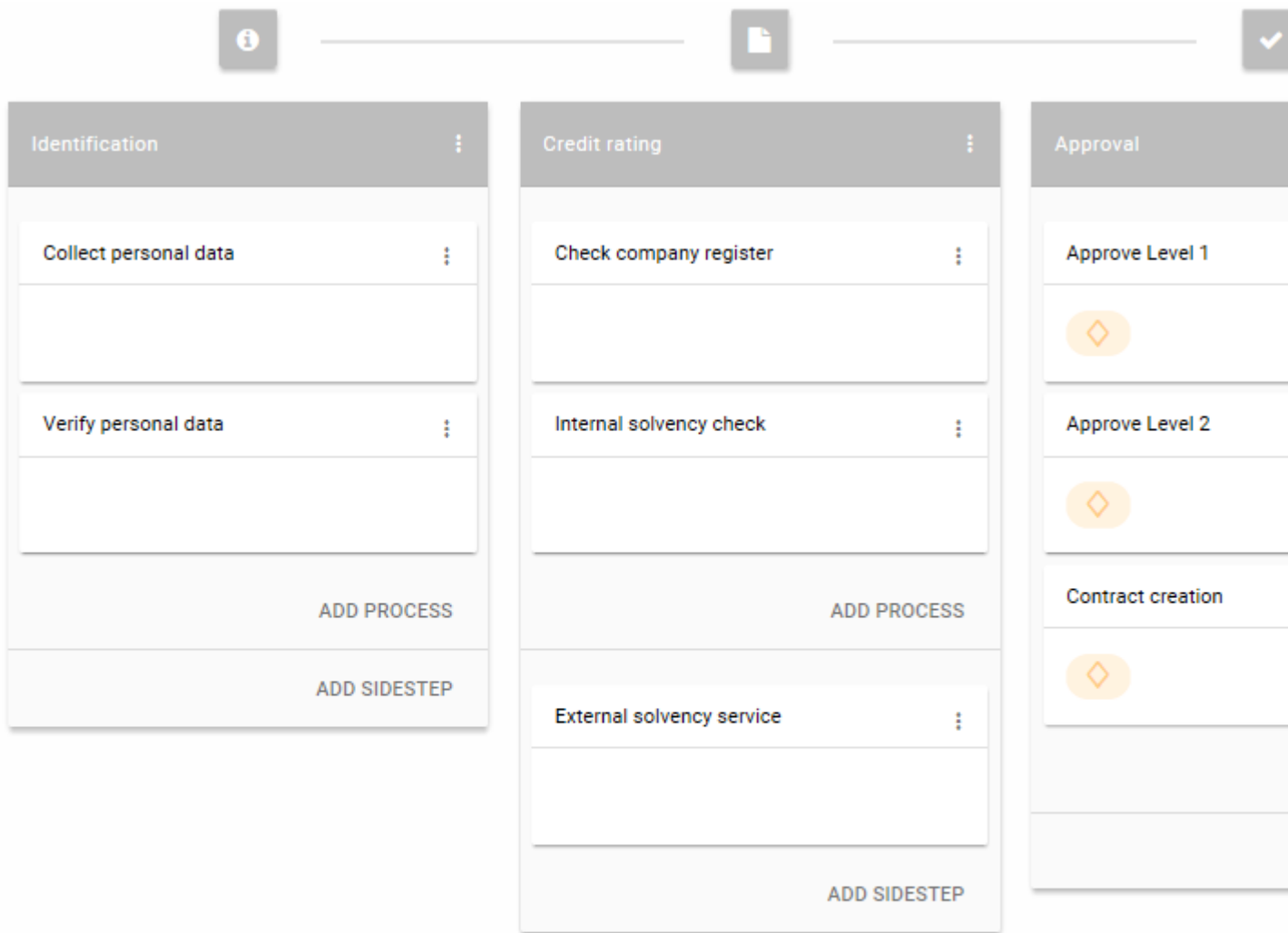


Figure 2.87. Case Map Editor

Case Map Element Reference

Stage

A case map is divided into stages. Each stage defines a certain phase in the life of a business process. A stage consists of processes. A business process can jump from stage to stage in any order. The default flow is from left to right. If the last process of a stage has finished and there is no explicit jump defined, the stage to the right will be taken.

The position of a Stage in a case map can be changed via the menu on the stage.

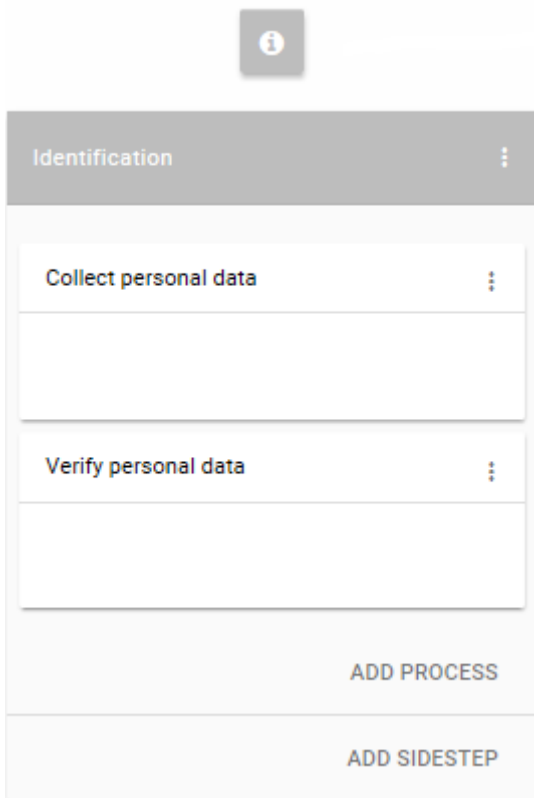


Figure 2.88. Case Map Element: Stage

Process

A process in the Case Map references to a process start of an Axon.ivy process. If the business process enters a stage, the first process, which entry condition evaluates to true, will be started.

Processes and sidesteps can be moved around by drag and drop.

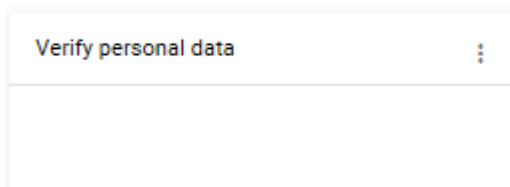


Figure 2.89. Case Map Element: Process

Process Precondition

Preconditions can be set on a process and define whether a process should be skipped. If a precondition is not met, the execution continues on the next process. For script features see “Case Map scripting”



Figure 2.90. Precondition symbol

Side Step

Side steps are nothing else than processes. They can be defined within a stage or as part of the global side step lanes. Side steps can be started manually during the ongoing business process. A typical side step could be a process which aborts the business process (e.g. abort request).

Sidestep Precondition

Decides whether that sidestep can be currently started or not. For script features see “Case Map scripting”



Figure 2.91. Sidestep Precondition symbol

Case Map scripting

Scripts within a Case Map can be written in ivyScript.

A simple Process Precondition could be implemented as follows:

```
businessCase.getCreatorUserName() == "Bruno" && creditDossier.needsApproval
```

Available variables

Within Case Map scripts the running Business Case is always accessible through the variable `businessCase`.

Any class that is annotated with `@BusinessCaseData` is accessible by its simple name. The variable value will be loaded from the Business Data Repository.

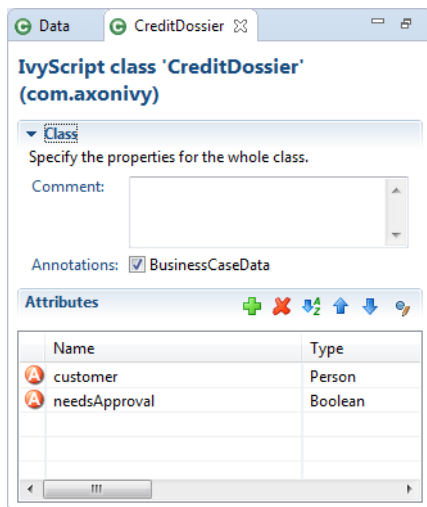


Figure 2.92. Sample DataClass with @BusinessCaseData annotation

Case Map Animation

The execution of a Case Map can be followed in the Case Map Editor. As known from the BPM processes the currently executing and already executed elements in the Case Map will be marked. It uses the simulation settings known from the BPM processes.

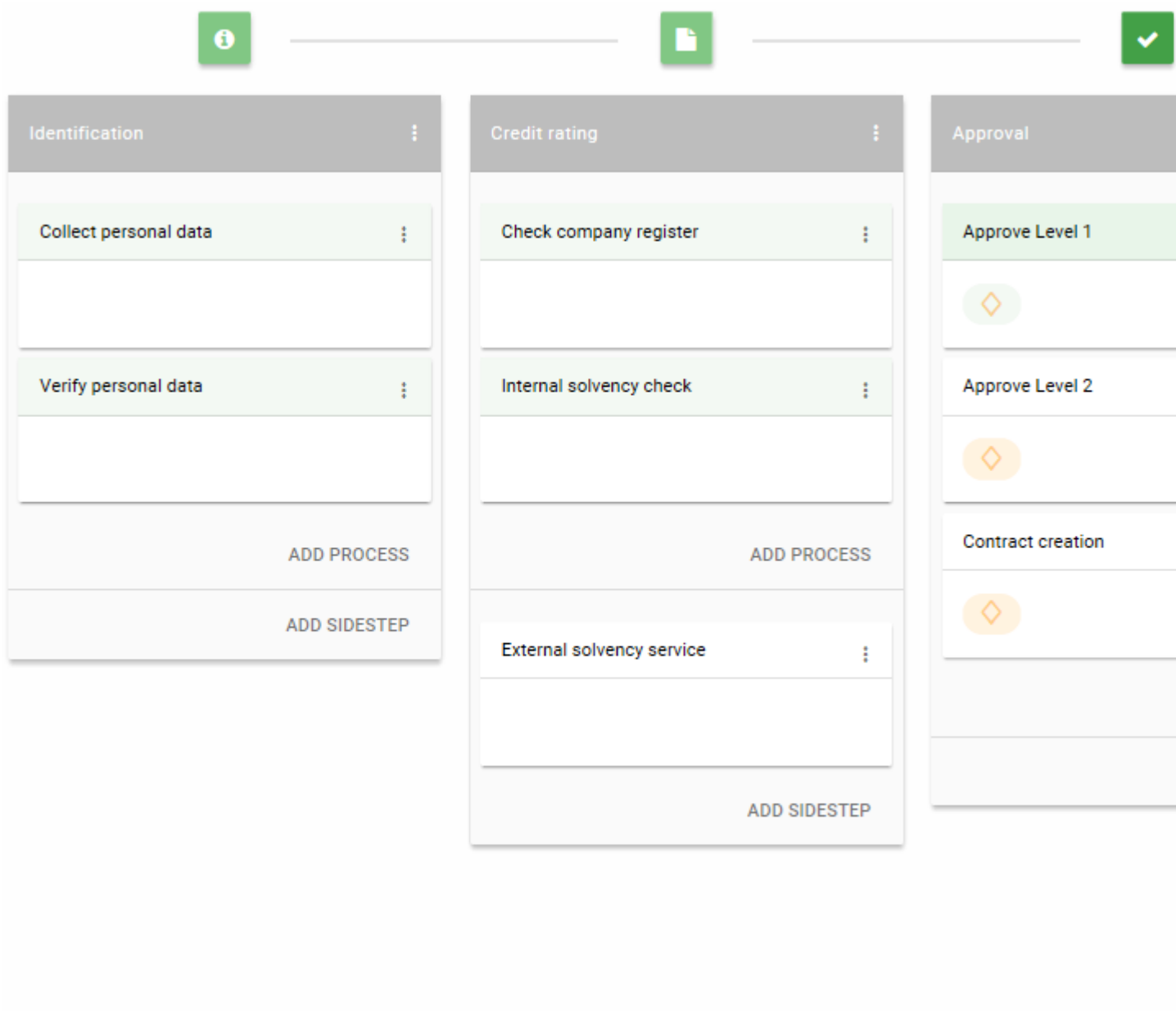


Figure 2.93. Animated Case Map Editor

Case Map Statistics

The case map provides the ability to display different process metrics of a case map. The monitoring can be enabled via the case map menu.

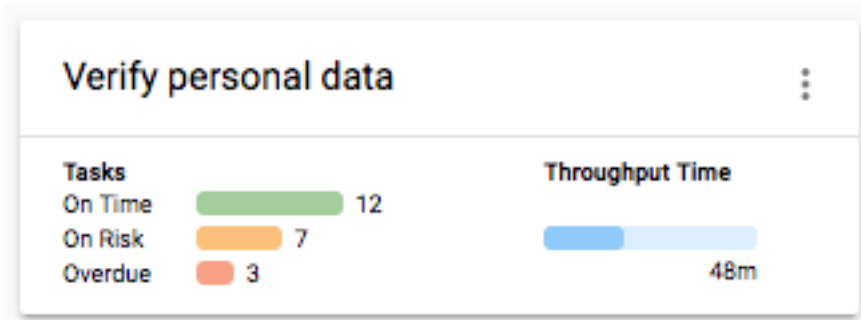


Figure 2.94. Process metrics

Tasks

The tasks statistics are based on the expiration dates of the case map tasks. Therefore, the task count is only based on tasks with an expiration date. The tasks are divided into following three categories:

- On time** Considers the average throughput time to calculate if the tasks are on time. A task is considered on time when the expiry date of the task is more than half of the average throughput time away from the current time.
- On risk** Considers the average throughput time to calculate if the tasks are on risk. A task is considered on risk when the expiry date of the task is the half of the average throughput time away from the current time.
- Overdue** The task count of expired tasks.

Throughput time

The average throughput time per task of this process is displayed.

Case and Task of Case Map Processes

For each Case Map Process in a Case Map a task and case is created when it is going to be executed. The referenced process itself is started at the referenced Request Start process element as soon as a user resumes the created task. You can configure the created case and task by using the Case and Task tab on the Request Start inscription mask. By default, the Request Start is not triggerable and the task is assigned to the responsible role configured on the Request tab. If the Request Start is triggerable then also the information on the Trigger tab is considered and the task is assigned to the responsible role or user configured on this tab. To automatically execute a process, configure the Request Start to be triggerable and set the responsible role to SYSTEM.

Chapter 3. Data Modeling

Data Classes

This chapter deals with the Axon.ivy Data Classes. In general, a Data Class holds the data that flows through your business or User Dialog process. You can build complex data structures out of your Data Classes. Use composition to split up your data if the amount of your data is getting bigger.

Types of Data Classes

There are four kinds of Data Classes in Axon.ivy.

- | | |
|-------------------------------|---|
| Global Data Classes | The global Data Classes are placed in the Data Class node in your project tree. They are accessible all over your project and the extending projects. |
| User Dialog Data Class | Each User Dialog has its own Data Class. This class holds the data that flows through your User Dialog processes. In your User Dialog Data Class it is possible to define fields with a type of a global Data Classes. The User Dialog Data Class is not visible at any other place except the User Dialog Processes. |
| Web Service (WS) Data Classes | The Web Service Data Classes are automatically generated if you define a Web Service configuration. The Web Service Data Classes are as well as the global Data Classes accessible from all over the project and the extending projects. Use this Data Classes to communicate with your Web Services. |
| Entity Classes | Entity Classes are like Global Data Classes but with additional information where and how to store the data of a class and its attributes to a relational database. See chapter Entity Classes for more details. |



Tip

The properties of a Rich Dialog's data class can be dragged from the Ivy Outline View and dropped onto a widget on the Rich Dialog's panel to add a data binding.

New Data Class Wizard

Overview

The New Data Class wizard lets you create a new global Data Class.

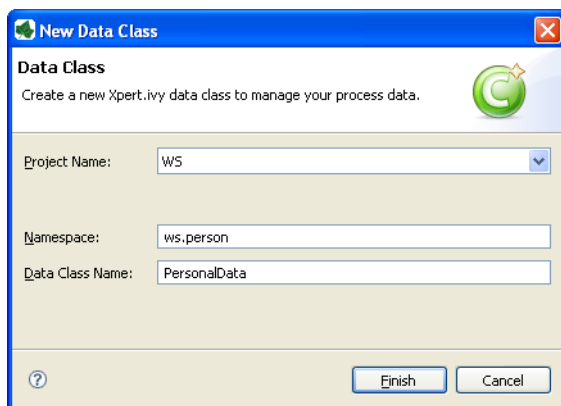


Figure 3.1. The New Data Class Wizard

Accessibility

File > New > Data Class

Features

- Project Name** Chose the name of the project the new Data Class should belong to.
- Namespace** Chose a namespace for your Data Class. The name space lets you create a structure to organise your data. Use the dot character '.' to separate the folders from each other. The namespace will be visible in the Axon.ivy project tree.
- Data Class Name** Enter the name of your Data Class. Do not use the names twice in your project, its getting confusing if you do so.

Data Class Editor

Overview

The Axon.ivy Data Class editor lets you configure the process data objects of Axon.ivy. The process data is the data that "flows" through your processes. It represents the state of the respective process.

Use this editor to add new data fields to your process data class, to change the type of a field or to document your Data Class or Data Class Field.

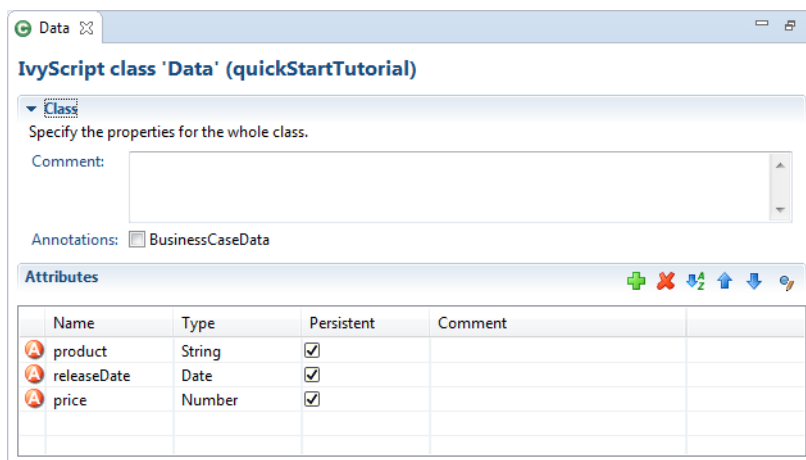


Figure 3.2. The Data Class Editor

Accessibility

1. Axon.ivy Project Tree > double click on a Data Class entry in the tree.
2. New > Data Class > then the editor opens if the class was created successfully


Attributes


The attributes table specifies the Data Class contents.


- Comment** Use this field to document your data class
- Annotations** Annotations can be set to control certain behaviours:



BusinessCaseData Objects of the data class are stored in the Business Data Store (`ivy.repo`) in the context of the current business case. See Business Case Data for more information.

Table actions

 Adds a new attribute to the table. Alternatively the new attributes can be added by clicking on an empty row.

 Deletes the selected attribute.

 Re-orders the selected attributes. The order influences just the presentation and has no logic implication.

 Toggles the value change breakpoint for selected attribute. The attribute icon  shows that a breakpoint is installed on an attribute. More information about value change breakpoints can be found in chapter Breakpoints.

Name column

Enter the name of your attribute. The name should not contain any special characters or spaces.

**Tip**

You may already specify the type of the attribute here by adding a colon ':' to the attribute name, followed by the desired type (e.g. **myDateAttribute:Date**).



The entered type is used as search filter. The following examples using a data or java class with the name `ch.ivyteam.demo.Person`:

- `person:Person` results in `person, ch.ivyteam.demo.Person`.
- `personList>List<Person>` results in `personList, List<ch.ivyteam.demo.Person>`.
- `javaPersonList:java.util.List<Person>` results in `person, java.util.List<ch.ivyteam.demo.Person>` (Here a prefix of the package name `java.util` is used as filter instead of the full qualified name `java.util`).
- `timestamp:Timestamp` results in displaying the type selection dialog because there are multiple types matching the type name `Timestamp`.

Type column

Enter the type of the attribute or press the  to bring up the data type selection dialog.

Persistent column

Decide if the data should be persistent between a task switch. If the data is not set to be persistent, then you lose all information if the execution of the process passes a task switch process element.

**Note**

This flag can be removed on attributes if the value is stored in the business data repository or using persistence or the data is only used temporarily within a task.

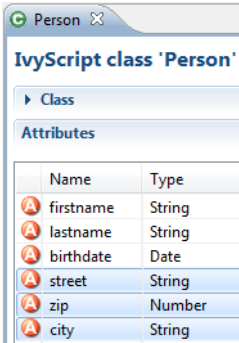
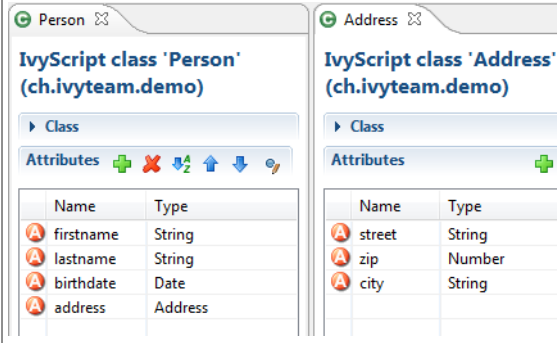
Attribute refactoring

The Data Class editor supports Data Class attributes refactoring.

Combine into new Data Class refactoring

Over time the amount of attributes in a Data Class may become excessive. This decreases the maintainability and reusability of your process logic. Therefore the editor allows you to extract multiple attributes from an existing Data Class into a new Data Class. The extracted attributes will be replaced with a delegate field for the new Data Class.

E.g. if you have a Data Class that describes a person you could extract the attributes that belong to the address part into an extra address Data Class.

Before refactoring	After refactoring																																
 <p>IvyScript class 'Person'</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Type</th> </tr> </thead> <tbody> <tr><td>firstname</td><td>String</td></tr> <tr><td>lastname</td><td>String</td></tr> <tr><td>birthdate</td><td>Date</td></tr> <tr><td>street</td><td>String</td></tr> <tr><td>zip</td><td>Number</td></tr> <tr><td>city</td><td>String</td></tr> </tbody> </table>	Name	Type	firstname	String	lastname	String	birthdate	Date	street	String	zip	Number	city	String	 <p>IvyScript class 'Person' (ch.ivyteam.demo)</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Type</th> </tr> </thead> <tbody> <tr><td>firstname</td><td>String</td></tr> <tr><td>lastname</td><td>String</td></tr> <tr><td>birthdate</td><td>Date</td></tr> <tr><td>address</td><td>Address</td></tr> </tbody> </table> <p>IvyScript class 'Address' (ch.ivyteam.demo)</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Type</th> </tr> </thead> <tbody> <tr><td>street</td><td>String</td></tr> <tr><td>zip</td><td>Number</td></tr> <tr><td>city</td><td>String</td></tr> </tbody> </table>	Name	Type	firstname	String	lastname	String	birthdate	Date	address	Address	Name	Type	street	String	zip	Number	city	String
Name	Type																																
firstname	String																																
lastname	String																																
birthdate	Date																																
street	String																																
zip	Number																																
city	String																																
Name	Type																																
firstname	String																																
lastname	String																																
birthdate	Date																																
address	Address																																
Name	Type																																
street	String																																
zip	Number																																
city	String																																

Start Refactoring

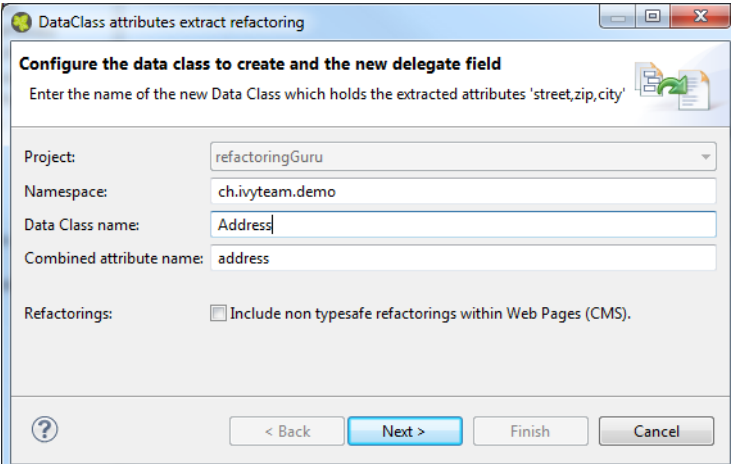
Select the attributes to extract in the Data Class attribute table. Open the context menu of the selected attributes. Choose *Combine to Data Class* to open the refactoring wizard.

Name	Type	Per...	Comment
firstname	String	yes	
lastname	String	yes	
birthdate	Date	yes	
street	String	no	
zip	String		
city	String	yes	

Context menu: Combine to DataClass

Wizard Page 1 - Define the new Data Class

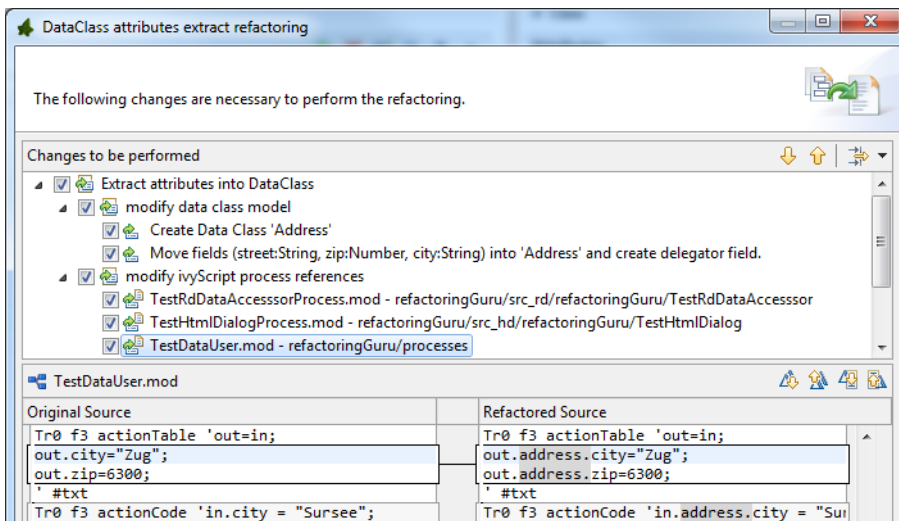
The first refactoring wizard page lets you define the location and name of the Data Class.



The checkbox 'Include non type safe refactorings within Web Pages (CMS)' enables the refactoring of JSP (macro) expressions within Web Pages. But Web Page modifications are not type safe. This means that attributes that are collected as change candidates may be wrong. So these changes should be inspected in detail in the refactoring preview (page 2) and tested manually.

Wizard Page 2 - Preview modifications

The second page previews all changes that will be applied to your artifacts. You can inspect most of the changes within the textual compare view on the lower part of the page.



The refactoring will modify all programmatic references to the attributes. This means that statements written in Java, ivyScript, JSP or El-Expressions could be changed by this refactoring.

Business Data Store

The Business Data feature allows to store and load business data in a built-in repository without defining a schema. Compared to other features like database steps and JPA no additional database, database connection, schema or tables are needed.

The business data structure can be defined by declaring normal data classes. When storing a business data object all objects that are referenced from the root object are stored as well. Loading a business data object will recreate the whole object tree again. The data is stored in a schema less JSON based data format in the ivy standard System database. This allows to add fields to data classes over time and still be able to load old business data without migration.

Moreover, the business data feature allows to search the stored data by defining field based filters.

Business Data Concept

Basically the Business Data Store implements a document store. Beside the stored value, the repository stores additional information about the Business Data, like an identifier, a version and the creation / update date.

A value data class can have fields of complex types, which allows to create an object hierarchy or tree. The storage mechanism can handle recursions and will respect objects of same instances. So if the same instance of an object is referenced in a field and in a list - after storing and loading the value - the loaded value will have the identical structure, the field and the list entry will reference the same instance.

Identity

A Business Data value is identified by an identifier given by the repository and its data class.

A unique id is generated if a Business Data value is stored the first time. If there is a field of type `String` with the name `id` in the Business Data class, the generated id will be stored into this field too.

It is also possible to use your own id if you set the id to the Business Data value before saving it for the first time.

Business Case Data

The handling of identifiers of Business Data values is complex because you have to manage the identifiers in the process data manually. Therefore the Business Data Store can store data in the context of a business case. You can activate this by annotating

a data class with the `@BusinessCaseData` annotation. On the Data Class Editor simply check the *BusinessCaseData* checkbox in the *Annotations* section. Now, all values of the annotated data class are automatically associated with the current business case. You can use the `get` method to load the value associated with the current business case. If no value is associated it simply returns a new object.

Migrate data classes

It is allowed to add and remove fields in the value data class hierarchy. New fields will be initialized with null, when old values get loaded. Deleted fields will no longer be available. The information will still be persisted until the value gets stored with the new information - which will override the old information.

Optimistic locking

Business Data supports optimistic locking. It is possible to check if the current version is up to date and save only if this is the case. It is possible to update a value partially so that multiple participant can work on different parts of the same Business Data value. See the *Concurrent Modification* demo in the *WorkflowDemos* project for a practical example.

Business Data Usage

The Business Data feature methods like `get`, `save`, `find` and `delete` are accessible under `ivy.repo` in IvyScript.

Associate value with the business case (BusinessCaseData context)

Annotate the main data class of the business case with the `@BusinessCaseData` annotation:

```
@BusinessCaseData
public class BusinessCaseDossier
{
...
}
```

Get (load or create), modify and save a dossier value in the context of the current business case:

```
BusinessCaseDossier dossier = ivy.repo.get(BusinessCaseDossier.class);
dossier.getPerson().setLastName("Polo");
ivy.repo.save(dossier);
```

Note, that the method `get` either loads the dossier if there is already a dossier associated with the current business case or creates a new dossier.

Store (without BusinessCaseData context)

Create and save:

```
Dossier dossier = ...
out.businessDataId = ivy.repo.save(dossier).getId();
```



Tip

It is recommended to only store the Id of the business value in the process data. After a Task Switch you must load the business data value from the repo with the stored Id. This is required, because the business data repo does not keep the reference to the instance on the Task Switch.

Load (without BusinessCaseData context)

Load, modify and save:

```
Dossier storedDossier = ivy.repo.find(in.businessDataId, Dossier.class);
```

```
storedDossier.getPerson().setLastName("Polo");
ivy.repo.save(storedDossier);
```

Search

The search capabilities of the Business Data Store are based on Elasticsearch and therefore fast and powerful.

There is a fluent API to search stored business data. The API supports filtering, ordering and limiting of the results:

```
List<Dossier> result = ivy.repo.search(Dossier.class)
    .allFields().containsAnyWords("Polo Columbus")
    .execute()
    .getAll();
```

Also fuzzy search and search engine like query strings are supported:

```
List<Dossier> result = repo.search(Dossier.class)
    .score()
    .allTextFields()
    .query("Baldwin~1 -Alec")
    .execute()
    .getAll();
```

Store with own Id

Create and save with own Id:

```
Dossier dossier = ...
String yourId = ... // generate your own id, be sure it is unique!
dossier.id = yourId; // set your id to the Business Data value
ivy.repo.save(dossier);

ivy.repo.find(yourId, Dossier.class) // get your Business Data value
```



Warning

Be aware that the id can not be changed later and the maximum length of the identifier is 100 characters.

Samples

The WorkflowDemos sample project of the Axon.ivy Designer contains examples on how to use the Business Data Store.

See Public API of `BusinessDataRepository` for more code samples.

Business Data Limitations

Size The Business Data store is not designed for storing huge binary objects like PDF's.

Types The ivy scripting types `XML`, and `Tree` are not serializable.

Collection types like an `ArrayList` can be stored in a field, but not as root object. Always use a simple `DataClass` or plain old Java objects as root object to store and load in the repository.

Public API objects like `IUser`, `ITask` or similar should not be stored into the Business Data repository. As workaround its better to store the Id of a Task or User and re-load it via this identifier.

The type of a stored field should never be changed (E.g. from `Number` to `String`). The already stored data de-serialization could fail and more likely Business Data with the new type can no longer be found via the search API as the search index is strongly typed.

Project Dependencies

When using the same Business Data value type in multiple projects 'a' and 'b' it is best to define the data classes for the business data in a own project 'base'. Then define a dependency from projects 'a' and 'b' to project 'base'.



Warning

If you use an object of a type that is defined in project 'a' inside the business data value (e.g. add it to a list) then the business data value cannot be loaded in project 'b'. This is because project 'b' is not dependent to project 'a' and therefore cannot load objects of classes that are defined in project 'a'.

Customization

The BusinessData store serializes Java objects to schema-less JSON by using the Jackson. Ivy DataClasses are predestinated to be serialized with Jackson. However, Jackson is able to store and load any Java object hierarchy. The following customizations could help to store your special plain old Java objects, which might can not be serialized by default.



Warning

Jackson is not only used for BusinessData serialization, but also to provide and consume “REST Services”. If you customize the serialization of Jackson, it will very likely also affect the serialization of Java objects which are used as input or return parameter of any REST service. If a serialization behaviour must only be applied for the BusinessData serialization, declare it as “Own module”.

Custom constructor

The de-serializer expects an empty default constructor to re-create a Java object. If you have a non default constructor (with parameters) or a factory method to create instances of your object, Jackson annotations are required so that the de-serializer knows how to re-create the object.

For a sample see: <https://github.com/FasterXML/jackson-databind/#annotations-using-custom-constructor>

Field without get/setter

The ObjectMapper will only store fields as JSON which are public accessible either by getter method or its field visibility. The re-creation of such field will fail if no setter is public accessible. Via annotations either the serialization of this field can be avoided or the re-creation can be enabled.

Avoid the serialization of a field:

```
public class MyCar{
    private List<Wheel> wheels;

    @JsonIgnore
    public List<Wheel> getWheels(){
        return wheels;
    }
}
```

Enable re-creation of a setter-less field:

```
public class MyCar{
```

```
@JsonProperty
private List<Wheel> wheels;

public List<Wheel> getWheels(){
    return wheels;
}
}
```

Own module

If simple annotations do not solve a serialization task, it's possible to write a completely custom serializer and de-serializer for Jackson. To do so implement a class that extends `com.fasterxml.jackson.databind.module.SimpleModule` and add your customization code into it. Register the class via SPI: create a file `META-INF/services/com.fasterxml.jackson.databind.Module` and store the qualified name of your module implementation in this file.

However, if you need to serialize instances of a popular library there could already be a Jackson module available that handles its serialization. See <https://github.com/FasterXML/jackson#third-party-datatype-modules>

If a module is already public available, simply add its JAR to the classpath of your project.

Persistence

This chapter introduces the Persistence Configuration and the Persistence API of Axon.ivy. The persistence framework is based on the Java Persistence API, aka JPA) and provides support for storing and loading business objects from and to a database in an easy way.

In order to use automated persistence in your business or User Dialog processes you need to define some Entity Classes first. An entity class is similar to a data class (i.e. a business object) but holds additional information that is necessary to map the class to a database table and it's attributes to database columns.

Once you have created entity classes, you need to define at least one persistence unit configuration. A persistence unit is responsible for managing all or a subset of your entity classes and defines the database where those entities are stored. Once you have configured one or more persistence units you can use them in your process steps with the Persistence API to load/update entity objects directly from the database or save/update them to the database.

Entity Classes

Entity Classes are like global Data Classes but with additional information where and how to store the data of a class and it's attributes to a relational database. An Entity Class is mapped directly to a database table and the attributes of an Entity Class are mapped directly to the fields of a database table. Therefore the database schema can be generated directly out of an Entity Class. It is possible to load, save, and update entity objects with the Persistence API.

Entity Classes are created with the New Entity Class Wizard and can be edited afterwards in the Entity Class Editor. Both of those are similar to the wizard and editor for regular Data Classes, but allow to specify additional settings, that are necessary for automated persistence.

New Entity Class Wizard

Overview

The New Entity Class wizard lets you create a new global Entity Class.

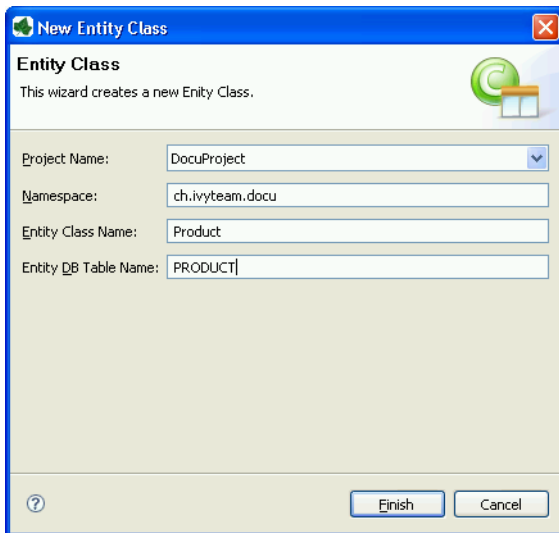


Figure 3.3. The New Entity Class Wizard

Accessibility

File > New > Entity Class

Features

Project Name	Chose the name of the project the new Entity Class should belong to.
Namespace	Chose a namespace for your Entity Class. The name space lets you create a structure to organise your data. Use the dot character '.' to separate the folders from each other. The namespace will be visible in the Axon.ivy project tree.
Entity Class Name	Enter the name of your Entity Class. Do not use the names twice in your project, its getting confusing if you do so.
Entity DB Table Name	Enter the name of the database table name of your Entity Class. If empty the name of your Entity Class is used. This name is used if the database table of this Entity Class is generated.

Entity Class Editor

Overview

The Axon.ivy Entity Class editor lets you configure the process data objects of Axon.ivy similar to the Data Class Editor. The process data is the data that "flows" through your processes. Additionally an Entity Class has information where and how to store the data of a class and it's attributes to a relational database.

Use this editor to add new data fields to your Entity Class, to change the type of a field or to document your Entity Class or Entity Class Fields.

Accessibility

Axon.ivy Project Tree > double click on a Entity Class entry in the tree.


New > Entity Class > then the editor opens if the class was created successfully


Features



Section Class Comment

Enter your text here to describe in prose what kind of data your Entity Class represents.

Section Attributes

Enter a list of attributes into the table. Use the  icon to add a new attribute or just click on the next empty cell in the "Name" column of the table.

If you want to reorder your entries in the table, then you can use the  icons to do so. The order influences just the presentation and has no logic implication.

Use the  icon to toggle the value change breakpoint for the currently selected attribute. The attribute icon  shows that a breakpoint is installed on a attribute. More information about value change breakpoints can be found in chapter Breakpoints.


Name Enter the name of your attribute. The name should not contain any special characters or spaces.



Tip

You may already specify the type of the attribute here by adding a colon ':' to the attribute name, followed by desired type (e.g. **myDateAttribute:Date**). When only adding a colon to the name without a type, the data type selection dialog will appear.



Type Enter the type of the attribute (fully qualified) or press the  to bring up the data type selection dialog.

DB Field The name of the field in the database table of this attribute. If you generate the database from this Entity Class for this attribute the DB field name is used as database field.

Persistent Decide if the data should be saved in the database if you use the persistence API and if the data should be persistent between a task switch. If the data is not set to be persistent, then you loose all information if the execution of the process passes a task switch process element.

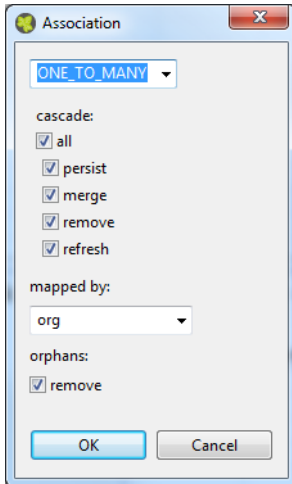
Length You can specify the length of the field in the database. This can only specified if the type is a String, BigDecimal or BigInteger. The default length for string fields is 255 and for decimal fields 19,2 on the database. Changes of the length has only an effect if the database schema is created new.

Properties	id	Specifies the primary key field of an entity. Every Entity Class must have exactly one primary key.
	generated	Specifies if the primary key should be generated automatically.
	not nullable	Whether the database column is not nullable.
	unique	Whether the field should be a unique key in the database.
	not updateable	Whether the column is not included in SQL UPDATE statements generated by the persistence provider.
	not insertable	Whether the column is not included in SQL INSERT statements generated by the persistence provider.
	version	Specifies the version field of an entity that serves as its optimistic lock value. The version is used to ensure integrity when performing the merge operation.

Association	Defines the association to another Entity Class. The actual configuration is done in the embedded Association Editor.
Comment	Describe the means of your attribute here.

Association Editor

Defines the association to another Entity Class and are only allowed to them and not other types of classes.



Association

ONE_TO_ONE Defines a one to one (1:1 on the database) association to another Entity Class. Can only be used if the type of the attribute is an Entity Class.

MANY_TO_ONE Defines a many to one (n:1 on the database) association to another Entity Class. Can only be used if the type of the attribute is a List or Set of an Entity Class. The inverse association of a **MANY_TO_ONE** is a **ONE_TO_MANY** association.

ONE_TO_MANY Defines a one to many (1:n on the database) association to another Entity Class. Can only be used if the type of the attribute is a List or Set of an Entity Class. This type of association needs always a mapped by specification, because this is always the inverse side of an **MANY_TO_ONE** association.

Cascade

Defines the cascadable operations which are propagated to the associated Entity. E.g. if persist is enabled then the associated object will be persisted automatically if an instance of this class is persisted.

persist	If enabled the associated object is persisted automatically if an instance of the class is persisted. See persist operation.
merge	If enabled the associated object is merged automatically if an instance of the class is merged. See merge operation.
remove	If enabled the associated object is removed automatically if an instance of the class is removed. See remove operation.
refresh	If enabled the associated object is refreshed automatically if an instance of the class is refreshed. See refresh operation.

Mapped by

The field that owns the relationship on the specified type which must be an Entity Class. This element is only specified on the inverse (non-owning) side of the association. Mapped by can only be used for **ONE_TO_ONE** and **ONE_TO_MANY** associations. The inverse side of the association must be also the inverse association (**ONE_TO_ONE** inverse **ONE_TO_ONE**, **ONE_TO_MANY** inverse **MANY_TO_ONE**)

Orphans

If orphans remove is enabled and an entity that is the target of the relationship is removed from the relationship (either by removal from the collection or by setting the relationship to null), the remove operation will be applied to the entity being orphaned. If the entity being orphaned is a detached, new, or removed entity, the semantics do not apply.

If orphan remove is enabled and the remove operation is applied to the source entity, the remove operation will be propagated as defined in the cascade section.

The remove operation is applied at the time of the flush operation. The orphans removal functionality is intended for entities that are privately "owned" by their parent entity.

Example:

A 'Basket' entity holds a list of 'Product' entities. What happens if `basket.getProducts().remove(..)` is called?

- orphan remove enabled: the product is removed from the list of referenced products even if the entity is reloaded or refreshed.
- orphan remove disabled: the product stays in the list of referenced products if the entity is reloaded or refreshed.

Persistence Configuration Editor

Overview

The Persistence Configuration Editor lets you configure the persistence units you use in your project and the extending projects.

Usually you only need a single persistence unit that manages all of your project's entity data classes. In this case you can simply create a new persistence unit, associate it with a data source (i.e. data base) and you're done. All of the project's entity classes will then automatically be managed by this unit.

However, if you wish to do so, you can divide your entity data classes into subsets and manage each subset through an individual persistence unit. In this case you need to specify multiple persistence units and then define an explicit list of managed entity classes for each unit.



Warning

Although it is theoretically possible to have the same entity class managed by two or more persistence units, you should be aware of the consequences. Once you have generated/attached an object of a specific class through a specific persistence unit, you must ensure that it is managed uniquely by that unit afterwards.

Example: Assume that you have two different objects of the same entity class, e.g. `Person`, generated through different persistence units, e.g. `employee` through `employee_unit` and `customer` through `customer_unit`. In this case you must absolutely avoid to mix persistence units and objects. For the above example, handing over `employee` to `customer_unit` (or `customer` to `employee_unit`) will consequently result in errors.

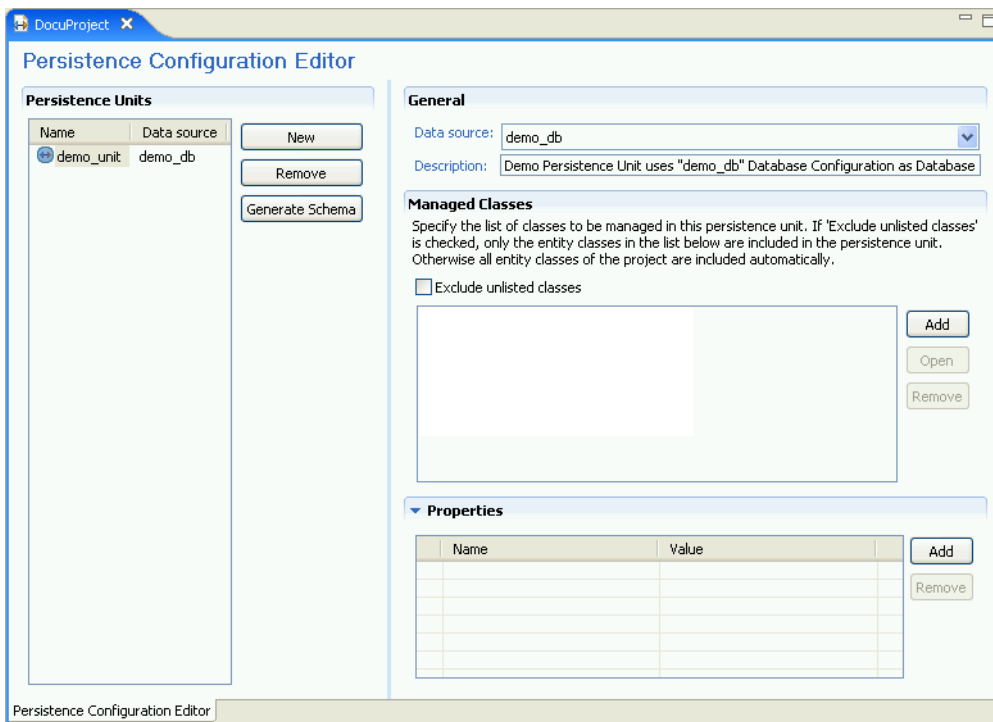


Figure 3.4. The Persistence Configuration Editor (single persistence unit)

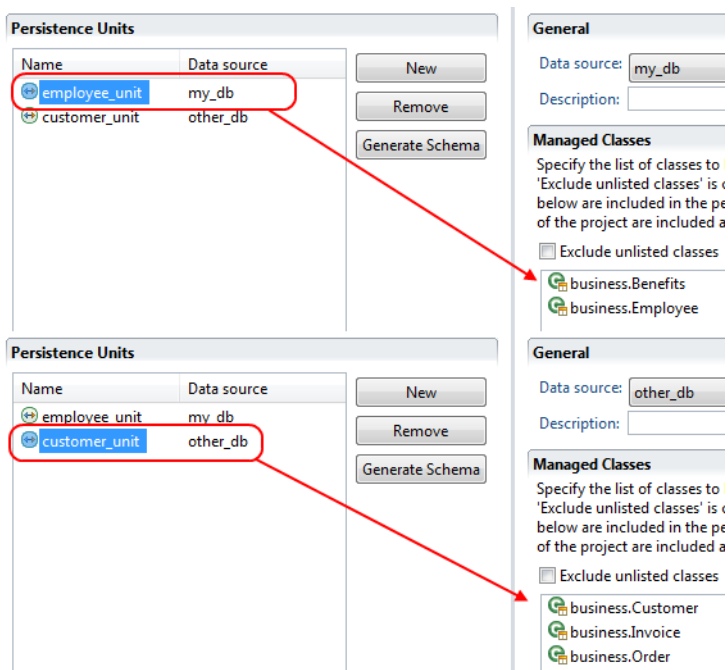


Figure 3.5. The Persistence Configuration Editor (multiple persistence units)

New Add a new persistence unit configuration

Remove Remove the selected persistence unit(s)

Generate Schema Generates the database schema out of the entity classes who belong to the selected persistence unit. See Generate database schema from persistence unit for details.

Accessibility

Axon.ivy Project Tree > double click on the Persistence label.

Features

Data source	Here you have to choose a database configuration which will be the data source of this persistence unit. Means all the data are loaded and stored within this database. Go to the Database Configuration Editor to configure available datasources. There is also an <code>IvySystemDatabase</code> datasource which points to the current System Database. Normally you would prefer your own database to split valuable customer data from the system data.
Description	You can give your persistence unit any description here.
Managed Classes	Specify the list of classes to be managed in this persistence unit. If 'Exclude unlisted classes' is checked, only the entity classes which are defined in the list are included in the persistence unit. Otherwise all entity classes of the project are included automatically plus the entity classes defined in the list.
Properties	Specify some properties for the persistence unit. You do not have to specify something here except you now what you are doing.

Generate database schema from persistence unit

Generation options (Step 1)

On the first wizard page you can specify the environment and the type of the schema generation.

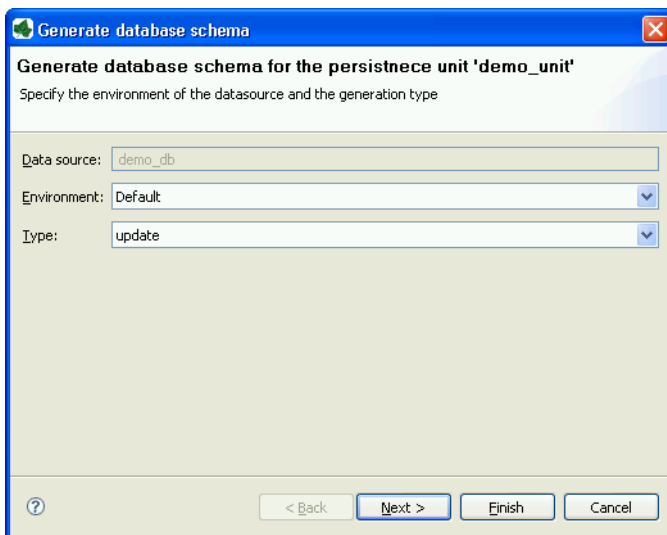


Figure 3.6. Database schema generation option

Data source	For the schema generation the data source of the persistence unit is used.
Environment	Specify the environment on which you like to generate the database schema.
Type	Specify the type of the schema generation. You can choose between update and create. <ul style="list-style-type: none"> • update: Does update the current available schema on the database.



Warning

The update does not refactor any changed table names, field names or field types. If a table or field does not exist in the database a new one is created even if the same table or field with another name exists.

- create: Does drop the current schema on the database and create a new one.



Warning

This option does delete all data which is stored in the database.

Generation preview (Step 2)

The second wizard page shows a preview what will be executed on the database.

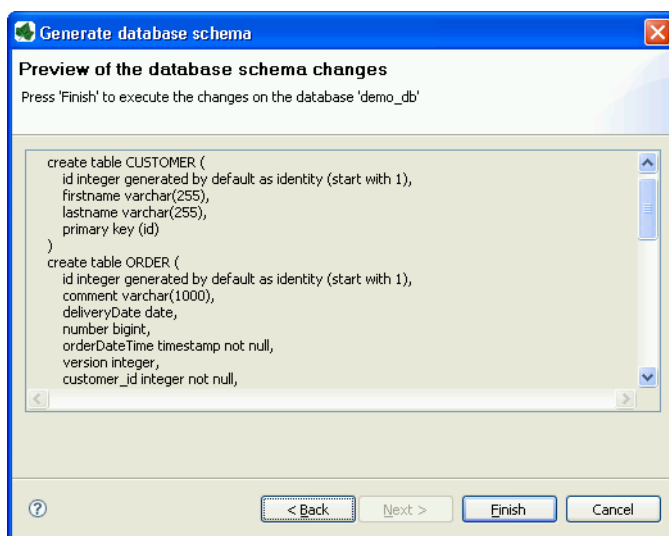


Figure 3.7. Database schema generation preview

Accessibility

Axon.ivy Project Tree > double click on the Persistence label > Select a persistence unit > Generate Schema.

Persistence API

Overview

The Axon.ivy Persistence API is used to load entity objects directly from the database or save/update them on the database. The Persistence API can be accessed by IvyScript anywhere scripting is supported. The Persistence API can only deal with entity objects, means objects of type Entity Classes. The Persistence API can be found under `ivy.persistence.<persistence unit>`. Here you find all the methods for finding, persisting, updating and querying entity objects. See `IIvyEntityManager` for more information. Replace `<persistence unit>` with the name of a persistence unit. The persistence units can be configured with the Persistence Configuration Editor.

Persist an entity object

To persist (save/create object on the database) you can use the `persist()` method of the Persistence API.



Warning

This method only works properly if the entity object and all the associated objects are not jet persistent. Otherwise you have to use the merge method.

Example (Product is an Entity Class):

```
// persist new created product
Product product;
product.name = "Product name";
product.nr = 12;
ivy.persistence.<persistence unit>.persist(product);

// get id of new created product
Number newProductId = product.id;
```

Find an entity object by id

To find an entity object by id (select object on the database) you can use the `find()` method of the Persistence API.

Example (Product is an Entity Class):

```
// load product with id 1 from the database
Product product = ivy.persistence.<persistence unit>
.find(Product.class, 1) as Product;
```

Merge an entity object

To merge (update or save/create object on the database) you can use the `merge()` method of the Persistence API.



Warning

Only the returned entity object of this method is the really updated or saved/created object. The object given to this method is not changed.

Example update (Product is an Entity Class):

```
...
// change before loaded product
product.name = "New product name"
Product updatedProduct = ivy.persistence.<persistence unit>
.merge(product) as Product;
```

Example save/create (Product is an Entity Class):

```
// save new created product
Product product;
product.name = "Product name";
product.nr = 12;
Product savedProduct = ivy.persistence.<persistence unit>
.merge(product) as Product;

// get id of new created product
Number newProductId = savedProduct.id;
```

Remove an entity object

To remove (delete object on the database) you can use the `remove()` method of the Persistence API.

Example (Product is an Entity Class):

```
...
// delete the product from the database
ivy.persistence.<persistence unit>.remove(product);
```

Refresh an entity object

To refresh (reload object from the database) you can use the `refresh()` method of the Persistence API.

Example (Product is an Entity Class):

```
...
// change before loaded product
product.name = "New product name"
// reload object from the database and revert local changes
ivy.persistence.<persistence unit>.refresh(product);
```

Persistence Queries (JPA QL)

With the Persistence API it is possible to execute Java Persistence API Query Language (JPA QL) statements. See `IIvyQuery` for more information about the Query API. The query language based around the objects that are persisted but with syntax very similar to SQL. You have always to use the names of the Entity Class and the attributes and not the names from the database.

Case Sensitivity

Queries are case-insensitive, except for names of Java classes and properties. So `SeLeCT` is the same as `sELeCt` is the same as `select` but `PRODUCT` is not `product` and `foo.barSet` is not `foo.BARSET`. This manual uses lowercase JPA QL keywords.

Single Result

To execute a JPA query where you are expecting a single value to be returned you would call `getSingleResult()`. This will return the single Object. If the query returns more than one result then you will get an Exception. This should not be called with "UPDATE"/"DELETE" queries.

Example (Product is an Entity Class):

```
Product product = ivy.persistence.<persistence unit>
.createQuery("select p from Product p where p.id = :id")
.setParameter("id", 1)
.getSingleResult() as Product;
```



Warning

Calling this method in automatic transaction mode (by default) will close the recordset automatically. Consequently you can not invoke this method multiple times or in combination with `getResultList()` on the same query.

Result List

To execute a JPA query you would typically call `getResultList()`. This will return a List of results. This should not be called with "UPDATE"/"DELETE" queries.

Example (Product is an Entity Class):

```
List<Product> products = ivy.persistence.<persistence unit>
.createQuery("select p from Product p where p.price > :price")
.setParameter("price", 10)
.getResultList();
```




Warning

Calling this method in automatic transaction mode (by default) will close the recordset automatically. Consequently you can not invoke this method multiple times or in combination with `getSingleResult()` on the same query.

Execute Update

To execute a JPA UPDATE/DELETE query you would call `executeUpdate()`. This will return the number of objects changed by the call. This should not be called with "select" queries.

Example delete (Product is an Entity Class):

```
// delete all products
Number deletedRows = ivy.persistence.<persistence unit>
.createQuery("delete from Product p")
.executeUpdate()
```

Example update (Product is an Entity Class):

```
// update product name
Number updatedRows = ivy.persistence.<persistence unit>
.createQuery("update Product set name = :newName where name = :oldName")
.setParameter( "newName", "New Product Name" )
.setParameter( "oldName", "Old Product Name" )
.executeUpdate();
```

Parameter binding

The JPA Queries supports named and numbered parameters and provides methods for setting the value of a particular parameter.



Tip

You should always use parameter binding and do not build the query with string concatenation, because of performance reasons.

Example with named parameter:

```
ivy.persistence.<persistence unit>
.createQuery("select p from Product p where p.price > :price")
.setParameter("price", 10)
```

Example with positional parameter:

```
ivy.persistence.<persistence unit>
.createQuery("select p from Product p where p.price > ?1 and p.amount <= ?2")
.setParameter(1, 10).setParameter(2, 80)
```

Paging the result

To specify the range of a query you have the two methods `setFirstResult()` and `setMaxResults()` available. The start position of the first result, numbered from 0.

Example (Product is an Entity Class):

```
List<Product> products = ivy.persistence.<persistence unit>
.createQuery("select p from Product p where p.price > :price")
.setParameter("price", 10)
.setFirstResult(40)
.setMaxResults(20).getResultList();
```

The call to `setFirstResult(40)` means starting from the fortieth object. The call to `setMaxResults(20)` limits the query result set to 20 objects (rows) returned by the database.

Ordering

JPA QL provide an ORDER BY clause for ordering query results, similar to SQL.

Returns all Products ordered by name:

```
from Product p order by p.name
```

You specify ascending and descending order using `asc` or `desc`:

```
from Product p order by p.name desc
```

You may order by multiple properties:

```
from Product p order by p.name asc, p.description desc
```

Distinct results

When you use a select clause, the elements of the result are no longer guaranteed to be unique.

DISTINCT eliminates duplicates from the returned list of product descriptions.

```
select distinct p.description from Product p
```

Comparison expressions

JPA QL support the same basic comparison operators as SQL. Here are a few examples that should look familiar if you know SQL:

Binary comparison (`=`, `<>`, `<`, `>`, `>=`, `<=`, `[NOT] BETWEEN`, `[NOT] IN`):

```
from Product p where p.amount = 100
from Product p where p.amount <> 100
from Product p where p.amount > 100
from Product p where p.amount <= 100
from Product p where p.amount between 1 and 10
from Product p where p.name in ('Product A', 'Product B')
```

Null check (`IS [NOT] NULL`):

```
from Product p where p.name is null
from Product p where p.name is not null
```

Arithmetic expressions (`+`, `-`, `*`, `/`):

```
from Product p where ( p.amount / 0.71 ) - 100.0 > 0.0
```

The LIKE operator accepts a string value as input parameter in which an underscore (`_`) stands for any single character, a percent (`%`) character stands for any sequence of characters (including the empty sequence), and all other characters stand for themselves:

```
from Product p where p.name like 'A%'
from Product p where p.name not like '_a_'
```

Logical operators (`NOT`, `AND`, `OR`):

```
from Product p
  where p.name like 'A%' and p.price > 10
```

Expressions with collections (`IS [NOT] EMPTY`, `[NOT] MEMBER [OF]`):

```
from Product p where p.customers is not empty
from Product p, Category c where p member of c.products
```

Operators	Description
.	Navigation path expression operator
+, -	Unary positive or negative signing (all unsigned numeric values are considered positive)
*, /	Regular multiplication and division of numeric values
+, -	Regular addition and subtraction of numeric values
=, <>, <, >, >=, <=, [NOT] BETWEEN, [NOT] IN, IS [NOT] NULL, [NOT] LIKE	Binary comparison operators with SQL semantics
IS [NOT] EMPTY, [NOT] MEMBER [OF]	Binary operators for collections in JPA QL
NOT, AND, OR	Logical operators for ordering of expression evaluation

Table 3.1. JPA QL operator precedence

Calling functions

An extremely powerful feature of JPA QL is the ability to call SQL functions in the where and HAVING clauses of a query.

Lower cases or upper cases a string (LOWER(string), UPPER(string)):

```
from Product p where lower(p.name) = 'product name'
from Product p where upper(p.name) = 'PRODUCT NAME'
```

Another common expression is concatenation, although SQL dialects are different here, JPA QL support a portable concat(string1, string2) function:

```
from Product p where concat(p.name, p.description) like 'A% B%'
```

Size of a collection (SIZE(collection)):

```
from Product p where size(p.customers) > 10
```

Function	Return	Description
UPPER(string), LOWER(string)	string	Lower cases or upper cases a <i>string</i> value
CONCAT(string1, string2)	string	Concatenates <i>string</i> values to one string
SUBSTRING(string, offset, length)	string	Substring <i>string</i> values (<i>offset</i> starts at 1)
TRIM([[BOTH LEADING TRAILING] char [from]] string)	string	Trims spaces on BOTH sides of <i>string</i> if no <i>char</i> or other specification is given
LENGTH(string)	number	Gets the length of a <i>string</i> value
LOCATE(search, string, offset)	number	Searches for position of <i>search</i> in <i>string</i> starting at <i>offset</i>
ABS(number), SQRT(number), MOD(dividend, divisor)	number	Returns an absolute of same type as input, square root as double, and the remainder of a division as an integer
SIZE(collection)	integer	Size of a <i>collection</i> ; returns an integer, or 0 if empty

Table 3.2. JPA QL functions

Aggregate functions

The aggregate functions that are recognized in JPA QL are count(), min(), max(), sum() and avg().

This query counts all the Products:

```
Number productCount = ivy.persistence.<persistence unit>
.createQuery("select count(p) from Product p").getSingleResult() as Number;
```

This query calculates the average the sum, the maximum and the minimum from the amount of all products:

```
select avg(p.amount), sum(p.amount), max(p.amount) min(p.amount) from Product p
```

Accessibility

You can use the Persistence API everywhere you have the ivy variable in the IvyScript. Use `ivy.persistence.<persistence unit>`. Here you find all the methods for finding, persisting, updating and querying entity objects. Replace `<persistence unit>` with the name of a persistence unit.

Chapter 4. IvyScript

Introduction

The IvyScript language is used to write business rules, for manipulating process data, to define data mappings and to set properties and parameters on Rich Dialog components.

IvyScript Language

The Ivy scripting language IvyScript provides elements to write simple computational expressions but also more complex elements to program conditional-, loop- and exception handling blocks.

The IvyScript data types are defined for easy use. Especially, IvyScript beware the programmer from null pointer exceptions because ivy data objects are automatically initialized to a default value. Read the section Null Handling for more details.

IvyScript can also directly manipulate Java objects in a easy way. Thus Java objects can be used without mapping and auto-casting simplifies the usage.

Language Elements

Conditions

Conditional expressions

Function style

```
IF (cond, ifExpr, elseExpr)
```

Java style

```
cond ? ifExpr : elseExpr
```

Conditional statements

```
if (cond) { ... } else { ... }
```

Loops

for

```
for (init; cond; increment)
{
  // do something here
  ...
}
```

```
for (element: list)
  // do something here
  ...
}
```

while

```
while (cond)
{
  // do something here
  ...
}
```

Exception Handling

IvyScript supports the try/catch/finally construct to handle exceptions that happen while executing external Java code.

```

try
{
// some code here
...
}
catch (Exception ex)
{
// compensate code
...
}
finally
{
// some code that is executed regardless of whether exceptions occurred
...
}

```

Null handling / Automatic object creation

IvyScript supports auto-initialization of the ivy basic types, i.e. you don't have to create/initialize fields or variables explicitly with `new` after declaration. `Strings` are initialised to an empty `String`, `Numbers` to zero, `Lists` to an empty `List`.

Ivy composite types (ivy Data Classes) are automatically created. Due to that automatic object creation, a null check expression like `if(in.customer == null)` is always false.

You can to use the `.#` operator to suppress the automatic object creation.

```

if( in.#customer == null)
{
// object is null
}

if (in.#customer is initialized)
{
// object is not null or has been set to a non-default value
}

```



Note

Any fields or variables of Java classes are also created automatically if they're referenced for the first time and if the type has a default constructor. Interface types and abstract class types are not auto-created because no instances can be created of such types in Java.



Note

Inside IvyScript it is generally recommended to use `is initialized` rather than comparing against `null` with the `==` operator. Because Java types may be `null` and IvyScript base types never, this operator will always ensure the correct checking depending on the type of the tested object.

Axon.ivy also supports auto-initialization of AXIS types:

```
org.apache.axis.types.Time to '00:00:00'
```

```
org.apache.axis2.databinding.types.Time to '00:00:00'
```

```
org.apache.axis.types.Duration to 'PT0S'
```

```
org.apache.axis2.databinding.types.Duration to 'PT0S'
```

A `java.util.Date` is auto-initialized to a default value of `'0001-01-01 00:00:00'`.

However, you do not have to compare the values of those types against hard coded default values in your code, simply use **is initialized** to find out if a value has been changed by the user or still bears the default value.

```
if (webserviceData.caseDuration is initialized)
{
    // do something
    ...
}
```

IvyScript Editor

Overview

There exist two flavours of an IvyScript editor, a yellow editor for coding IvyScript and a blue editor for writing plain text that contains macros (this means, that you can mix IvyScript with normal text).

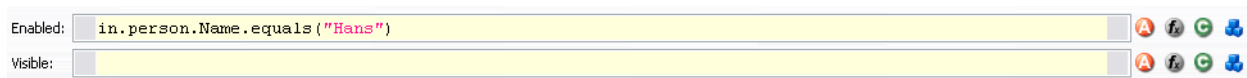
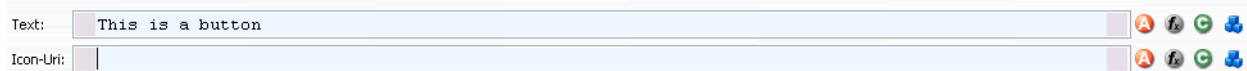


Figure 4.1. Standard IvyScript Editor



The yellow editors expect you to enter either a script with multiple statements (e.g. a script, that contains semicolons) which performs a certain task, or just an expression that evaluates to a certain value. Which is expected, should be clear by the context.

Figure 4.2. IvyScript Editor for macros

Features

Content Assist

Content Assist is invoked by pressing *CTRL+SPACE* at any point of editing a script. Content assist will open a popup, displaying proposals that are available in the current context, from which you may then select a suitable option. The selected proposal is inserted into the editor. You can get proposals for functions, types, packages, variables and methods and after the keyword "new" you also get a list of constructor proposals.

Example 1: When you would like to have displayed a list with all proposals that match with an already entered "c", you just enter "c" and press *CTRL+SPACE*. You will then get a list with proposals of functions, types and packages, each displayed with a help text if available.

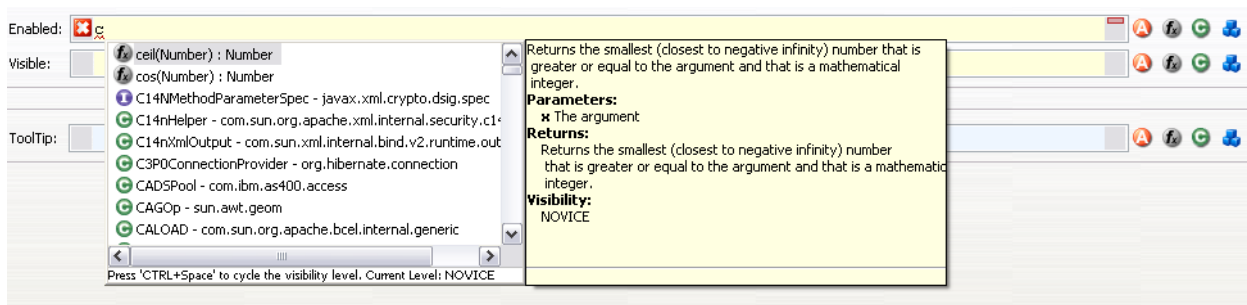


Figure 4.3. Content assist in action

Example 2: The constructor proposal list just appears after the keyword `new`. So you could create a new `date:Date d = new` and press *CTRL+SPACE* after typing "new" and you get a list of possible constructors to create a new date.

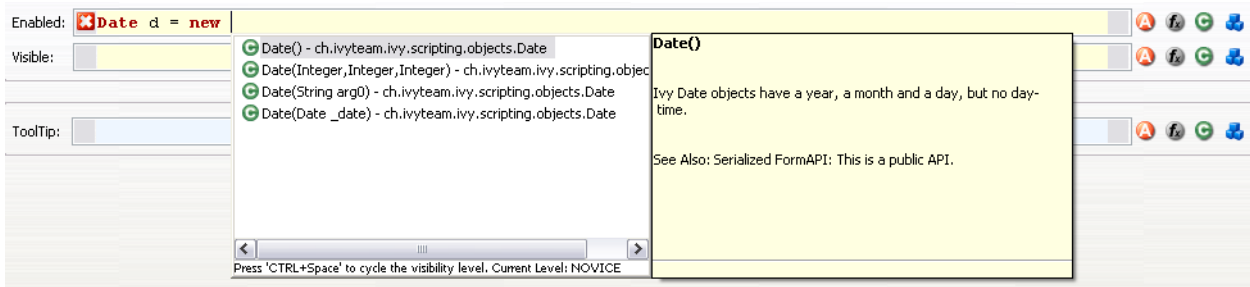


Figure 4.4. Constructor proposals

Example 3: Similar to types, you can also get proposals for packages.

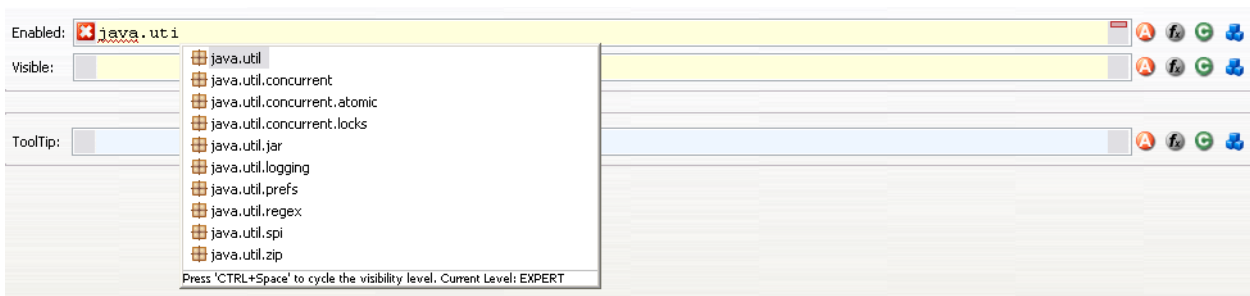


Figure 4.5. Package proposals



Tip

When the content assistant is opened and you press *CTRL+SPACE* again, the visibility level of the proposals is cycled. There are three different levels: *Novice*, *Advanced* and *Expert*. Depending on the visibility level, you get to see more or less proposals.

Parameter hopping

Another special feature is parameter hopping. When you insert a constructor or a method that has parameters, the first parameter is selected. When you now press the *Tab* key, then the next parameter gets selected. This way you may edit one parameter after another and simply jump to the next one when you're finished. After the last parameter was selected, the first is selected again. When you have finished, you can press *Enter* and the cursor jumps to the end of the inserted method or constructor.



Figure 4.6. Parameter hopping: After insertion of proposal first parameter is selected

Shortcuts

Shortcut	Action
CTRL+SPACE	Opens content assistant, when pressing again, the visibility of the content assistant is cycled.
F2	When pressing F2 in an editor, a bigger editor is opened in an own Dialog.
ESC	Inside an editor that was opened with F2, this closes the dialog and stores the entered text in the editor from where the F2 editor was opened.

Shortcut	Action
CTRL+Z	Undo
CTRL+Y	Redo

Table 4.1. Available Shortcuts inside the IvyScript (and Macro) Editor

Smart Buttons



Figure 4.7. Smart Buttons

Next to the editors you usually find buttons (which ones, depends on the context), that hold certain actions. The exact actions that those buttons realize are described in the section Smart Buttons. There are e.g. actions to select an attribute from the current process data, to select content or to insert a link.

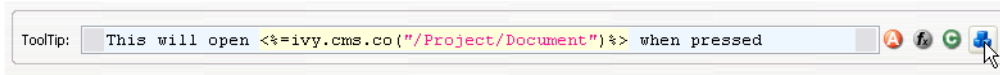


Figure 4.8. Macro Editor after insertion of a CMS object with help of the Content Smart Button

Browsers

Attribute and Method Browser

This browser is used to construct and insert IvyScript expressions for IvyScript text fields or areas in inscription masks. Those expressions are based on the process data in the context of the current step.

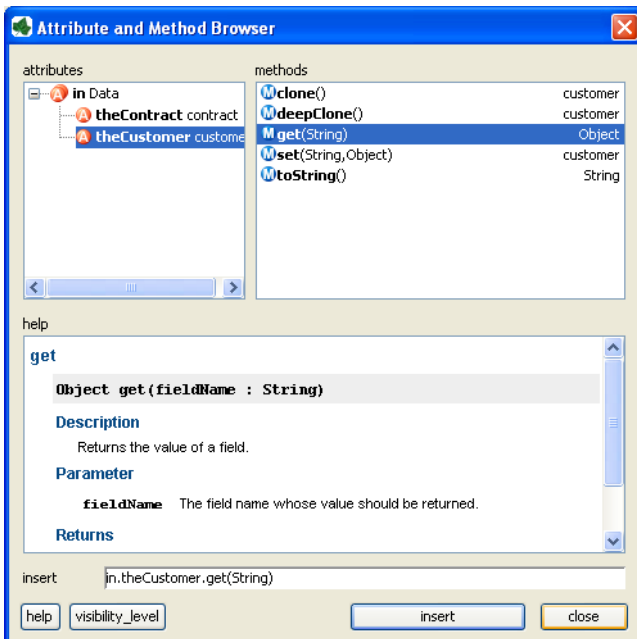


Figure 4.9. The Attribute and Method Browser

In the upper left area, you can choose between the different process attributes in the current context (such as `in`, `out`, `param`, `result` or `panel`). Depending on the selection, you can add a corresponding method to the expression in the upper right

area and the *help* area displays information to the selected attribute/method. The constructed expression can be previewed in the *insert* text box at the bottom and be inserted into the inscription mask by clicking on the button *insert*.



Tip

By default only the most common attributes/methods are displayed. With the *visibility_level* button you can relax this filter in two steps. The same may be configured permanently in the Axon.ivy Designer IvyScript preferences.

Function Browser

This browser is used to construct and insert IvyScript expressions for IvyScript text fields or areas in inscription masks. Those expressions are based on the environment in the context of the current process or on general-purpose functions.

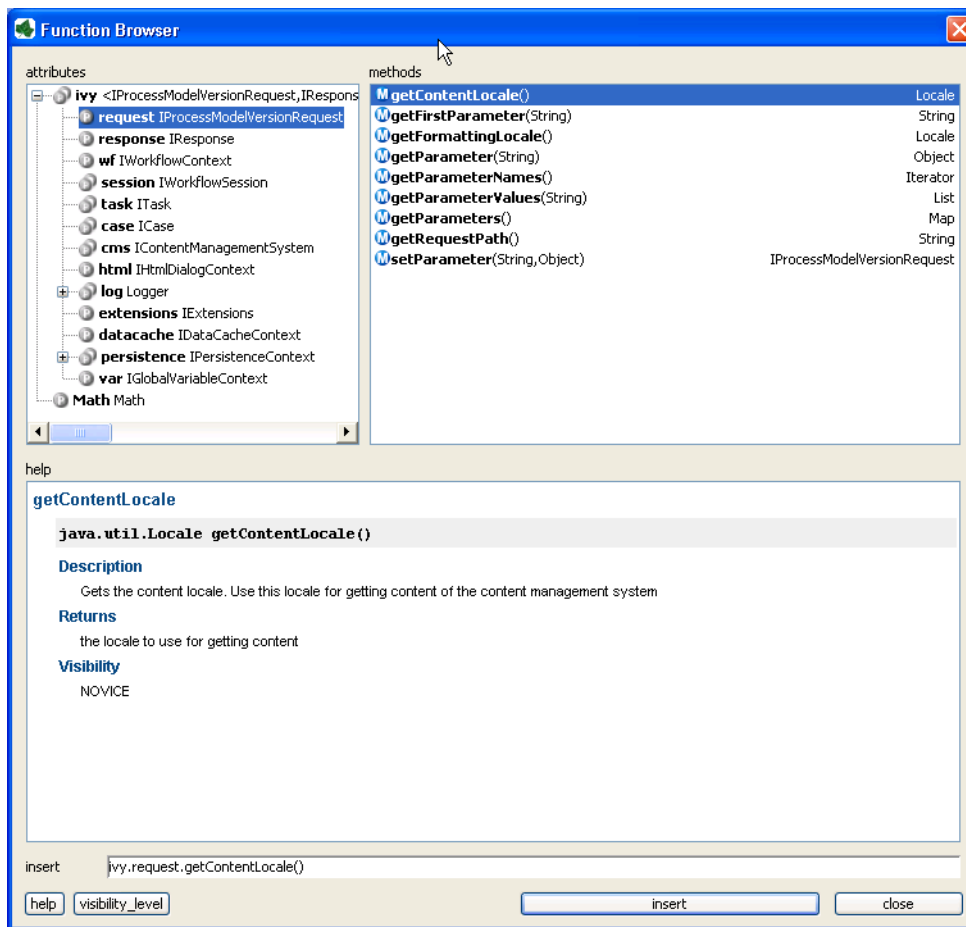


Figure 4.10. The Function Browser

In the upper left area, you can choose between the different attributes of the different environment variables in the current context. You can add a corresponding method to the expression in the upper right area and the *help* area displays information to the selected attribute/method.

A description of the accessible objects can be found in the section *ivy environment variables*

The constructed expression can be previewed in the *insert* text box at the bottom and be inserted into the inscription mask by clicking on the button *insert*.



Tip

By default only the most common attributes/methods are displayed. With the *visibility_level* button you can relax this filter in two steps. The same may be configured permanently in the Axon.ivy Designer IvyScript preferences.

Data Type Browser

The data type browser is used to choose a data type in the:

- Process Data Class editor
- User Dialog Data Class editor
- User Dialog Interface editor
- Code tabs of inscription masks

Data types are divided into two categories:

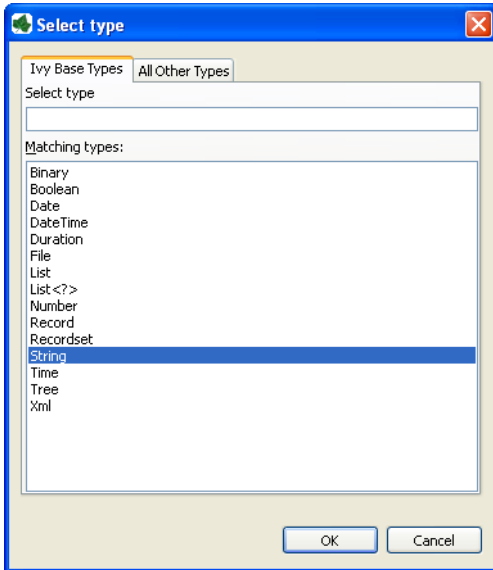


Figure 4.11. Ivy Base Types

This category contains the Axon.ivy base types. These types may be used within IvyScript without any restriction. Note that for convenience reasons database `Record` and `Recordset`, `XML`, `Tree` and `List` types are supported out of the box.

Selecting `List<?>` will bring up another data type browser where you specify the type of the list members.

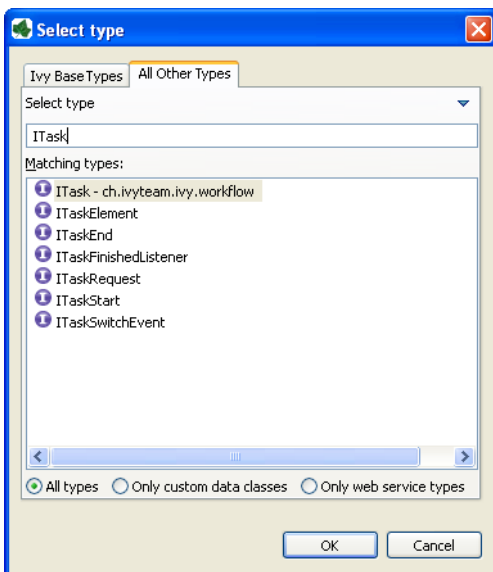


Figure 4.12. All Other Types

Here you can find all other types that are available in your project. This includes all Data Classes that you have created, all classes that were generated from Web Services and all other Java classes that are on the build path.

Start typing the name of the desired type to get suggestions in the list. On the bottom, you may limit the suggestions to only custom Data Classes or Web Service classes.



Tip

In the filter, you can use * (any string) and ? (any character) as wild cards.



Tip

To switch between the tabs, use the shortcuts *Alt+Left*, *Alt+Right* or *Ctrl+Shift+T*.

To change the focus from the filter to the list, press the *Tab* or *Arrow-Down* key

Public API

For access within IvyScript a substantial portion of Axon.ivy functionality has been released as a Public API. You may use all classes and their objects in IvyScript fields.

IvyScript Reference

Operators

Operator	Explanation	Usage
.	Field and method access of ivy objects	<code>in.customer.name</code> addresses the name attribute in the data structure <code>in.message.length()</code> calls the method <code>length()</code>
#	Field access with suppressed auto initialisation	<code>in.#customer == null</code> null check of customer which is not initialised
as	Type cast operator	<code>in.anObject as Date</code> casts the object to a Date

Table 4.2. IvyScript Field Access and Type Cast Operators

Operator	Explanation	Usage
>	greater than	<code>5 > 3</code> is true
<	less than	<code>5 < 3</code> is false
==	equals (Java equals)	<code>5 == 5</code> is true <code>"Hello" == "HELLO"</code> is false
!=	unequal	<code>7 != 2</code> is true
>=	greater than or equal	<code>7 >= 6</code> is true
<=	less than or equal	<code>2 <= 5</code> is true
&&	Boolean AND	<code>true && true</code> is true
	Boolean OR	<code>true false</code> is true
!	Boolean NOT	<code>! true</code> is false

Table 4.3. IvyScript Logic Operators

Operator	Explanation	Usage
+	Addition	12.5+17.0 is 29.5
	String Concatenation	"Hello "+"World" is "Hello World"
-	Subtraction	3020-12 is 3008
*	Multiplication	2*4 is 8
/	Division	7/2 is 3.5
%	Modulo Division	7%2 is 1
**	Power	2%5 is 32
++	Increment	in.n++
--	Decrement	in.n--
-	Negative Number value	-9

Table 4.4. IvyScript Arithmetic Operators

Ivy Script Data Types

Boolean

A boolean has the values `true` and `false`.

The IvyScript `Boolean` is based on the `java.lang.Boolean` but has a simplified class reference definition. Type conversion and format methods has been added while most other methods are hidden.

You can refer to the Java language documentation for a description of the methods of the data type `Boolean`.

Date

This class represents a date (without time of day).

Date constant objects are entered in the ISO 8601 format as `'yyyy-mm-dd'` Where `yyyy` is for the year, `mm` for month and `dd` for day.

Accepted is also the format: `'dd.mm.yyyy'`

`new Date()` returns the current date.

Read the Axon.ivy API Java Doc for the constructors and method summary of the data type `Date`.

DateTime

An object of this class represents a Date with Time.

Constant `DateTime` objects are entered in the ISO 8601 format as `'yyyy-mm-dd hh:nn'` or `'yyyy-mm-dd hh:nn:ss'`. `yyyy` is for the year, `mm` for month, `dd` for day `hh` for hours, `nn` for minutes and `ss` for seconds.

Accepted is also the format: `'dd.mm.yyyy hh:nn'` or `'dd.mm.yyyy hh:nn:ss'`

`new DateTime()` returns the current date and time.

Read the Axon.ivy API Java Doc for the constructors and method summary of the data type `DateTime`.

Time

An Object of this class represents a time of day.

Time constants are entered as 'hh:mm' or 'hh:mm:ss' Where hh is for hour, mm for minutes and ss for seconds
`new Time()` returns the current time.

Read the Axon.ivy API Java Doc for the constructors and method summary of the data type `Time`

Duration

This data type is used for **time periods**.

You enter a duration in the ISO 8601 time period format such as: '12h20m' or '12h20m30s'

An example for the full format is: 'P3Y6M4DT12h30m10s'

Read the Axon.ivy API Java Doc for the constructors and method summary of the data type `Duration`

Number

IvyScript Numbers are Java Numbers. Number objects are integer or fixed-point numbers or floating point numbers.

Integer are entered as: 23 or -10

Fixed Point number are entered as: 0.1 or -123.57458

Floating point numbers are given with exponent: 1.2345E3 or 42.3234E-4

IvyScript Numbers `java.lang.Number` objects but has a simplified and extended class reference definition. Format methods has been added for convenience.

You can refer to the Java language documentation for a description of the methods of the data type `Number`

String

String objects represent character strings.

You enter a String literal in double quotes: "Hello John"

Strings can be concatenated with the + operator: "Hello " + "John"

The IvyScript Strings are `java.lang.String` objects but has a simplified and extended class reference definition. Conversion and format methods has been added for convenience.

You can refer to the Java language documentation for a description of the methods of the data type `String`

Record

Usually Records are obtained in the context of data base queries, where they represent a row in a table. `Record` objects are similar to a `List` where each element has an assigned field name.

Read the Axon.ivy API Java Doc for the constructors and method summary of the data type `Record`

Recordset

A Recordset may be the result of a database query representing part of a table.

Read the Axon.ivy API Java Doc for the constructors and method summary of the data type `Recordset`

XML

This class is used for the processing of XML documents. You can create XML Documents or apply XPath expression to filter and extract values.

Read the Axon.ivy API Java Doc for the constructors and method summary of the data type `Xml`

Tree

This data type holds the data for a tree. A tree is a hierarchy of nodes and sub nodes. A node in the tree contains a value object and an info string and might have any number of attached children sub nodes.

Read the Axon.ivy API Java Doc for the constructors and method summary of the data type `Tree`

Binary

A Binary object is a wrapper object for a byte array.

Read the Axon.ivy API Java Doc for the constructors and method summary of the data type `Binary`

List

List objects can contain any number of other objects of any type. Each object in a list has an index which starts at zero.

Examples are: `[1 , 2 , 3]` a list with three numbers

`[1 , "Red" , 2 , "Green" , 3 , "Blue"]` a list with different objects.

Beside this general list type, so called typed list exists. A typed list can only contain objects of certain type.

Those list types are written as follows: `List<aClass>`

Read the Axon.ivy API Java Doc for the constructors and method summary of the data type `List`

File

A File object can be used to read/write temporary or persistent data. IvyScript Files are created in a confined area that belongs to the running application. Temporary files are created in a session-specific file area and are automatically deleted if a session ends. Temporary Files can be made persistent.

IvyScript Files are always addressed relatively, i.e. absolute addressing will lead to errors. You can create folders and files, i.e. a hierarchical structure, but you can not navigate outside the confined area (which is also the reason why absolute File paths are disallowed).

Read the Axon.ivy API Java Doc for the constructors and method summary of the data type `File`



Tip

You can always use `java.io.File` as an alternative to the IvyScript File object. However, in this case you must always use the Java File fully qualified, i.e. you can not import the class. Any IvyScript File can be transformed into a Java File (e.g. if needed to pass as parameter to a Java method).

The Environment Variable `ivy`

The `ivy` environment variable is provided to access the context of the current process, especially the workflow environment objects, the content management system and the Rich Dialog or HTML dialog contexts. The environment information is available as fields on the global `ivy` variable, e.g. to access the CMS of the current project you use:

```
String okMessage = ivy.cms.co("/text/messages/ok");
```



Note

Access from IvyScript:

The variable `ivy` is available everywhere, where *IvyScript* can be used, e.g. on *Step* elements or in *output tables* of other elements.

Access from Java:

You can also access `ivy` from a Java context, e.g. from helper classes or on a Rich Dialog panel implementation. To do so, simply import the `ch.ivyteam.ivy.environment.Ivy` class and use its static API.

Please note that it is necessary that the Java code which makes use of the `ivy` context variable must run within an Ivy request. Otherwise context information will not be available, most likely resulting in an `EnvironmentNotAvailableException`.

Access from JSP:

The `ivy` variable is also accessible from JSP. You can import the class `ch.ivyteam.ivy.page.engine.jsp.IvyJSP` and declare the variable `ivy` in your JSP as follows:

```
<%@ page import="ch.ivyteam.ivy.page.engine.jsp.IvyJSP"%>
<jsp:useBean id="ivy" class="ch.ivyteam.ivy.page.engine.jsp.IvyJSP" scope="session"/>
```

The following environment objects are available on `ivy` (details of the objects are described in the Public API):

- `cal` - an `IDefaultBusinessCalendar` object that gives access to business calendar informations and calculations.
- `request` - an `IProcessModelVersionRequest` object, the representation of the request against the server to execute the current step
- `response` - an `IResponse` object, the response of the Axon.ivy Engine on the request to execute the most current step
- `wf` - an `IWorkflowContext` object giving access to all workflow objects (all tasks, all cases) of all users for the application under execution. Can be used to build a whole workflow administration UI application, find tasks, cases, do statistics, etc. There is a workflow context for each application and vice versa.
- `session` - an `IWorkflowSession` object gives access to all workflow objects (task and cases) that belongs to the user of the current session. A workflow object belongs to a user if:
 - A task is assigned to him or a role he owns.
 - A task he is currently working on.
 - A task he worked on in the past (needs permission).
 - A task that a member of a role he owns has worked on in the past (needs permission).
 - A case he has started (needs permission).
 - A case that have been started by a member of a role he owns (needs permission),
 - A case that has a task which he worked on (needs permission).
 - A case that has a task which a member of a role he owns has worked on in the past (needs permission).
- `task` - an `ITask` object, the representation of the user's current work unit in the process under execution.
- `case` - an `ICase` instance that represents the current process under execution
- `cms` - a `IContentManagementSystem` object representing the CMS used in this project.
- `html` - a `IHtmlDialogContext` object specifies the Axon.ivy HTML environment
- `rd` - a `IRichDialogContext` instance allowing access to the context of the displayed Rich Dialog

- `log` - a `Logger` object. You can define log outputs here that will be collected for each run. You can see these log entries in the Runtime Log view.
- `extensions` - a `IExtensions` instance allowing access to Axon.ivy extensions
- `datacache` - the reference to the `IDataCacheContext` instances for the application and session (see `Data Cache` for more information)
- `persistence` - references to the existing persistence units in this application (see `Persistence Configuration Editor` and `Persistence API` for more information about the API of the Persistence)
- `var` - references to the global variables that are defined for this application (see `Data Global Variable` for more information)
- `rules` - references to the rule engine integration within Axon.ivy.



Note

The `rd` object is only available within a Rich Dialog logic step and the `html` object is only available within a business process.

IvyScript-Java Integration

Call Java methods and fields

You can easily write own Java classes and use it directly in IvyScript. You can call static methods and fields from Java classes (e.g. `java.lang.Math`). You have to address the class with the qualified name or use `import` statements. If a Java method has no return parameter (void) then the called object of the method is returned (e.g. a call to `user.setName(...)` returns object `user`).

```
import java.lang.Math;

Number r = Math.random();
Number pi = Math.PI;

out.n= r*pi;
```

Working with different Date, Time and DateTime implementations

When working with Databases and Web Services in Axon.ivy, then different implementations of Date, Time and combined Date-Time information are encountered (e.g. `java.util.Date`, `java.sql.Date`, Axis Time, etc). To complicate matters further, some of those implementations are - for historical reasons - mutable (e.g. `java.util.Date`) which is from today's perspective an unwelcome behavior. This has been remedied by some other implementations.

To facilitate working with values of those different types, IvyScript will *always* convert them to the corresponding, *immutable* IvyScript base types whenever such values are encountered, according to the following table:

Java type	Ivy type
Axis1 Time	Ivy Time
Axis2 Time	Ivy Time
Axis1 Duration	Ivy Duration
Axis2 Duration	Ivy Duration
JDBC (SQL) Timestamp	Ivy DateTime
JDBC (SQL) Date	Ivy Date

Java type	Ivy type
JDBC (SQL) Time	Ivy Time
Java Date	Ivy DateTime

Table 4.5. Automatic conversion of foreign Date / Time values

The automatic conversion into IvyScript types takes place transparently. The developer should therefore only think in terms of the IvyScript Date / Time types. No explicit conversion has to be made, neither when reading nor when writing those types.



Warning

As a general rule, *do not create any variables or objects of foreign (i.e. Java) Date / Time types inside IvyScript*. Although a statement such as

```
java.util.Date myDate = new java.util.Date();
```

is valid and permitted in IvyScript, the actual type of the *myDate* object will always be IvyScript DateTime, due to the auto-conversion. This can lead to confusion.

When trying to find out if date or time values are *null* or not initialized, developers should always use the `is initialized` operator rather than testing against *null*:

```
// recommended style
if (person.birthday is initialized) ...
if (in.lunchTime is initialized) ...
if (schedule.appointment is initialized) ...

// unsafe style, not recommended
if (person.birthday != null) ...
if (in.lunchTime != null) ...
if (schedule.appointment != null) ...
```

Auto casting rules

IvyScript supports auto casting between the most important Java types and IvyScript types. This means, that you no longer have to use the `toXYZ()` methods on your IvyScript values. Instead you can directly assign IvyScript types to Java types and vice versa. This also holds for lists (IvyScript) and arrays (Java).

The following auto-casting rules are supported by IvyScript (bidirectional):

Java type		Ivy type
Axis1 Time	<->	Ivy Time
Axis2 Time	<->	Ivy Time
Axis1 Duration	<->	Ivy Duration
Axis2 Duration	<->	Ivy Duration
JDBC (SQL) Timestamp	<->	Ivy DateTime
JDBC (SQL) Date	<->	Ivy Date
JDBC (SQL) Time	<->	Ivy Time
Java Date	<->	Ivy DateTime
Java Date	<->	Ivy Date
Java Date	<->	Ivy Time

Java type		Ivy type
byte[]	<->	Ivy Binary
aType[]	<->	Ivy List<aType>>

Table 4.6. Auto casting rules

Chapter 5. CMS

Content Management System

The content management system (from now on CMS) in Axon.ivy is a hierarchically organized container for content like labels, short texts, images, source snippets or documents. You can store elements in the CMS and refer them later in processes or User Dialogs. And you can store content in multiple languages thus enabling you to internationalize your processes or applications.



Note

CMS content can be overridden in Axon.ivy. You can use this feature to customize the products you develop with Axon.ivy. See the chapter Overrides for more details.

CMS Structure

A Content object is identified by its *path* which is expressed as an URI of the form **/Labels/Common/CustomerName**. The first */* represents the root of the CMS whereas the rest forms a recursive tree of so called *Content Objects*. Each Content Object can contain other Content Objects thus forming the recursive structure of the CMS. Each Content Object has one or multiple *Content Object value(s)*. A Content Object Value is always bound to a specific locale. A locale is a combination of a language identifier and a region identifier. For example the locale en_US represents the language English for the US region. So, you can define values for different languages but as well for different regions which use the same language (see how this is used for the resolution of CMS content at run-time).

In Axon.ivy, each project has its own CMS. Content Objects are looked up by means of the Content Object URI mentioned above. If the lookup for a Content Object fails in the current project, then Axon.ivy will recursively lookup the URI in the CMS's of the required projects (breadth-first).



Tip

Put common content that you use in multiple projects into a base project and make your other projects dependent on the base project. Then you can share and re-use all Content Objects from the base project.

Content Object Types

There are various types of content that can be stored in a CMS. Every Content Object does have a specific content *type*. Content Object Values inherit that type from their Content Object. The types are used to access the content in the correct way (e.g. to set the MIME type in HTTP requests) but as well to provide specific editors for the manipulation of the values.

Symbol	Type Name	Purpose	Edited with ...
	Folder	For structural purposes only, folders are container for other Content Objects.	
	String	Short texts (single line), e.g. labels, names, descriptions, tool tips.	String Editor
	Text	Longer and/or formatted texts with multiple lines or even multiple paragraphs.	Text Editor
	Image	An image of arbitrary size. <i>GIF Image, PNG Image, JPG Image</i> types are supported.	Image Editor
	Source	Scripts of any form, e.g. <i>javascript</i> or <i>jsp snippets</i> .	Source Editor











Symbol	Type Name	Purpose	Edited with ...
	Page	Container object for <i>HTML Page</i> content. This is used in Web Page process elements.	HTML Page Editor
	CSS	Cascaded Style Sheet definitions	CSS Editor
	Layout	JSP HTML layout with included Content Objects. Typically created and used as part of a <i>Page</i> object.	Layout Editor
	Panel	Panels are the content parts for <i>Page</i> objects and are defined with <i>Layout</i> objects.	HTML Panel Editor
	JSP	Alternative to the <i>Page</i> object. Uses pure JSP for layouting.	JSP Editor
	Table	Allows to place content and Content Objects into a HTML table.	HTML Table Editor.
	Link	Generates a HTML link or form.	HTML Link Editor
	Result Table	Generates dynamic tabular HTML content from process data.	Result Table Editor
	Smart Table	Generates dynamic tabular HTML content from process data gathered from a data base, Supports paging.	Smart Table Editor
	Document	Any document (the most common document formats are supported such as PDF, DOC, XLS, MP3 ...)	Document Editor

Table 5.1. Content Object Types

CMS Access

In Axon.ivy

CMS content can be used in the most locations where Axon.ivy displays text for example in User Dialogs, Web Pages or in processes. Use Content Objects to set the text of your labels, the images for your icons or the content of your HTML pages. There are two ways how to use content from the CMS:

- In most Axon.ivy Editors you have a Smart Button (see here too) for the CMS. The smart button will create the correct code to access the CMS in the current editor.
- In IvyScript you can use the `ivy.cms` environment variable and hereby the Public API class `IContentManagmentSystem`. The class offers the method `co` that returns content itself and the method `cr` which returns a link to the content. In Java the same environment variable is available with `ch.ivyteam.ivy.environment.Ivy.cms()`.

Depending on the context Axon.ivy will return the content (link) in the correct form. For example, if you use an image Content Object in the Icon-Uri field of the widget configuration of a RLabel then Axon.ivy will display the image in the Rich Dialog. On the other hand if you use a document Content Object in the HTML panel editor, then it will be rendered as a link to the document in the HTML page.

Access with a Browser

Some content objects can be accessed directly from the browser with the URL pattern `http://<servername>:<port>/ivy/cm/<application name>/<process model>/<path in CMS>`. Assumed you have created a page in a CMS with the path `/StaticContent/MyPage` in a project named `Test`. Type the URL `http://localhost:8081/ivy/cm/designer/Test/StaticContent/MyPage` in your browser and the page will be rendered in there.



Note

The engine of the Axon.ivy Designer must be started to render the Content Objects.



Note

Technically it is possible to display any page with this mechanism. But most pages display information from a process and therefore access the data of that process. With this mechanism you access the content outside of the process scope. Therefore you do not have a data class in access so that it might lead to an error.

Content resolution

If content from the CMS is requested, it is addressed using the URI of the Content Object. But the real content (the text, the string, ...) is stored in a Content Object Value. How does Axon.ivy resolve the Content Object Value whose content is returned?

First, Axon.ivy tries to find the requested Content Object. It looks up in the current project first. If not found Axon.ivy will recursively look up in the CMS's of the required projects in a breadth-first manner (i.e. first it searches in all of the directly required projects, then in all of the required projects of the directly required projects and so on).

Second, as soon as Axon.ivy has found the Content Object, it evaluates which is the correct value to return. First, the lookup locale is defined. The algorithm to resolve the lookup locale is like this:

1. If the content locale was set on the session, then take this locale. See the Public API method `ISession.setContentLocale(java.util.Locale)`.
2. If the request comes from the Designer and the user defined a specific content locale, then use this locale.
3. If the (root) request was initiated from a browser, then use the browser locale. Please look up the help of your browser to see how you can edit this setting.
4. If the (root) request was initiated from a Rich Client, the locale of the client operating system is used.
5. Otherwise use the default locale of the operating system.

When Axon.ivy knows the lookup locale, then it tries to resolve the correct value. The algorithm for that is like this:

1. If there is a value with the same locale like the lookup locale, then return this value
2. If there is a value with the same language in the locale like the language of the lookup locale, then return this value.
3. If there is a default value, then return this value.
4. Otherwise return the first value.

CMS Manipulation

CMS View

This view is the central UI element for the interaction with the CMS. It shows the CMS of all open projects in the workspace including all Content Objects and their values and offers multiple ways to perform actions on the CMS and the Content Objects.

URI	English*	German
Expenses		
Dialogs		
expenses		
ExpenseEdit		
FirstName	First name	Vorname
Layouts		
Project		
Banner	...	
FooterText	<P>Powered by <%=ch.ivyteam.ivy.A...>	
HeaderText	<H1>My Project</H1>	<H1>My Project in German</H1>
Styles		
system		
Invoice		


Accessibility


Window -> Show View -> CMS

Features

Display Content Object values

The central element in the view is a table tree that shows the structure of the CMS in the first column (the tree column). Furthermore the table can display one column for every language that is available for at least one project in the workspace. In those columns the value of the corresponding Content Object in the corresponding language is shown.

If you want to focus on the CMS structure then you can hide all the language columns so that only the first column is displayed. Just click on  in the toolbar of the view to toggle between hiding and displaying the columns for the languages.

If you want to see the column for the values then you can configure for which languages the table shows a column. By default the view shows a column for the default language(s) of the project(s) in the workspace. Click on the menu () in the view to configure which columns are visible.




Note

The * in the title of a language column indicates that the language is the default CMS language for a project. Because you can have multiple projects in your workspace, it may be that you have multiple default languages and therefore multiple columns with a *. You can define the default languages for every project in its CMS preferences.

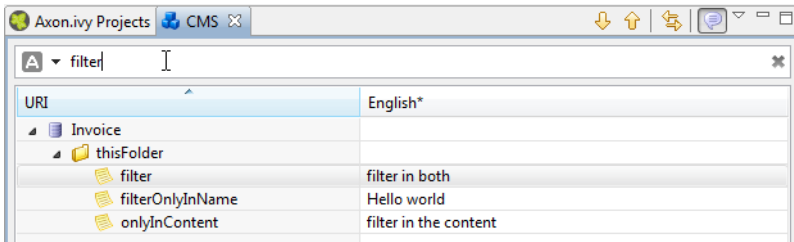
Inline Editing

For some types you can edit the Content Object values directly. For (some) text based types you can edit the text directly in the view, just click in the cell of the value and type.

For file based types you can import a file for each value directly in the cell. Just move your mouse over the cell of the value for which you want to import a file and click on the  icon on the right. Values for which already content is available show a ... in their cell.

Filter the view

By using the CMS search you can filter the contents in the view according to your search string. Enter a filter expression and the CMS table tree will be reduced to only show the Content Object matching that expression. If there is no match, all Content Objects are shown.



Use an asterisk (*) as wildcard to search for *any sequence of characters*: E.g. the filter expression **ivy*data** would match **ivyMyFancyData**, **ivydata**, **ivy something else data**, etc.

Use a question mark (?) as wildcard to search for *any single character*: E.g. the filter expression **image?data** would match **imageZdata**, **image0data**, etc.



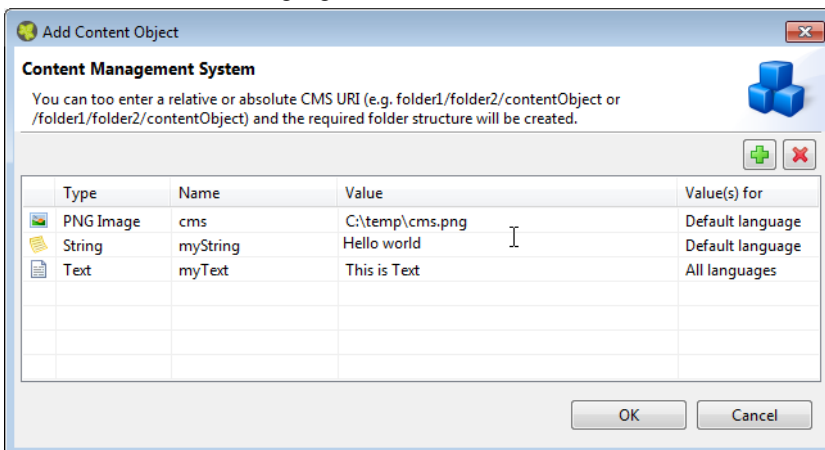
Tip

An asterisk (*) is always added implicitly at the end of your filter expression. So if you enter the string **ivy**, the filter expression that is really used is **ivy***.

Add new Content Objects

You have basically two options to create new Content Objects, either you do it kind of generic or you can create them from file(s). As a convenience method you can create folders in a more direct and simpler way than other Content Objects as a folder does not have value(s). You can execute all three actions only in the popup menu of the view.

Use **Add...** in the popup menu to create new Content Objects and enter the type, the name, the value and whether you want to create values for all languages in the CMS.



The default for the type is *String*. The *Document* types are not available for manual choosing, use **Add from file(s)...** if you want to create such Content Objects.

In the name column you can use either a simple name or as well an absolute or relative path. If you do so, then Axon.ivy will check the corresponding path and create folder Content Objects where necessary.

For text based types you can edit the text directly in the value column. For file based types you can import a file, just move your mouse over the cell and click on the icon.


In the last column you can decide whether you want to create automatically one value for every CMS language or not. The default comes from the corresponding project property and overwrites that if changed.

If you choose to create the Content Objects from file (the **Add from file(s)...** command in the popup menu, then first a file chooser dialog is opened. In there select the files that you want to have in the CMS and click **OK**. Then Axon.ivy will create a Content Object for each file. The type is detected automatically (if it cannot be detected then that file will be omitted), the name is set to the file name and the default value is the file content. After the file chooser, Axon.ivy opens the normal Add Content Object dialog so that you can revise the decisions before the Content Objects gets actually created.

You can add Content Objects too from the web. Just copy the URL and click on the **Add from URL...** command in the popup menu. In the next dialog, just enter the URI and continue to import the Content Object from the web.

Other actions


In the view you can invoke several actions from the popup menu:


 **Rename** Opens a dialog where you can enter a new name for the currently selected Content Object.



Warning

When you rename a Content Object, the URI of all it's children will change (e.g. from **/Labels/Common/Ok** to **/Labels/Buttons/Ok**). Any references to those objects (including the renamed object) will not be updated automatically and might be broken!


 **Copy** Copies the currently selected Content Object (including all of it's children) to the clipboard. The copied Content Objects can be inserted somewhere else in the content tree with *Paste*.

 **Paste** Inserts any Content Object(s) that was copied before to the clipboard with *Copy*. The copied Content Objects are inserted as children of the selected Content Object .



Note


Not all Content Object types are allowed as children of other Content Object types. In such a case the *Paste* menu entry might be disabled.

 **Delete** Deletes the currently selected Content Object (*including all of its children*) from the CMS after requesting a confirmation from the user.




Warning

Deleting a Content Object will break all references to the object or its children!

 **Copy URI** Copies the URI of the currently selected Content Object to the system clipboard. Use **CTRL + v** to insert the URI into any text fields or editors.

 **Copy URI as IvyScript macro** Copies the URI of the currently selected Content Object as an *IvyScript macro tag* to the system clipboard. Use **CTRL + v** to insert the macro into a IvyScript Macro text editor.

 **Refresh Content** Refreshes (i.e. reloads) the content below the currently selected Content Object.

Drag and Drop

Content Objects (e.g. *strings* and *images*) from the CMS view can be dragged and dropped into

- the Rich Dialog editor (aka Visual editor)
- Widget Configuration fields
- the Html Dialog editor

to be used for label texts or for images.

Content Object Editor

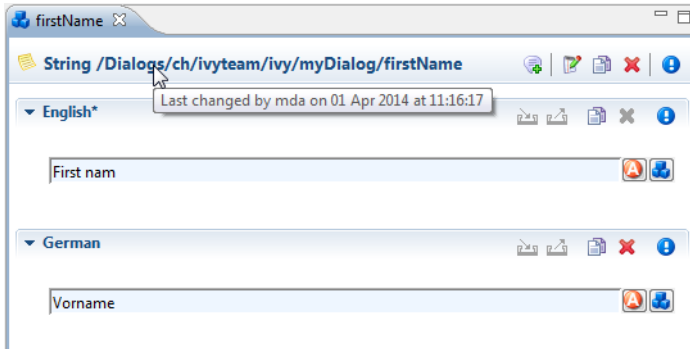
The Content Editor is used to manipulate Content Objects in the Content Management System (CMS) of a project.

Accessibility

Double click on a Content Object in the CMS view or select one and press the **ENTER** key.

Content Object header

The editor has a header with a title and buttons for the most important actions.



In addition to the Content Object type and the path in the title you can find more information about the Content Object in the tool tip of the title. There you see the date, time and the author of the last change. And the following actions are available on the right end of the header:



Open page preview

Opens a preview of the Content Object in a web browser. This action is only available within a Page Content Object. If the *Web Browser View* is active it is used otherwise an external browser is opened to show the preview.



Add new Content Object value

Adds a new value to this Content Object. A dialog is opened for the user to select the language of the new value.



Rename Content Object

Opens a dialog where a new name for the this Content Object can be entered.



Warning

When you rename a Content Object, the URI of all it's children will change (e.g. from **/Labels/Common/Ok** to **/Labels/Buttons/Ok**). Any references to those objects (including the renamed object) will not be updated automatically and might be broken!



Copy Content Object

Copies this Content Object (including all of it's children) to the clipboard. The copied objects can be inserted somewhere else in the content tree with *Paste*.



Delete Content Object

Deletes this Content Object from the CMS. A Content Object is deleted *with all of its values and child Content Objects*.



Warning

Deleting a Content Object will break all existing references to it or to any of its children!

Content Object Values area

Each of the values of a Content Object is shown with its corresponding value editor inside a collapsible section that is labelled with the *language* of the value. The *default* value is marked with a * (star) after the language name. The date, time and author of the last change is also shown for each value in the tool tip of the title of the value. Like in the header for the Content Object you find some actions on the right side:



Import value content

Opens a file dialog that allows to select a file with content to be imported.



Note

Not all content types allow to import content (e.g. *strings* do not). If the import is not supported, then the toolbar action will be disabled.

The file selection dialog will only show files that are suitable for import, depending on the standard extension for the required content type. i.e. you cannot select a *.css* file for import into a *png Image* Content Object.

Export value content

Some content types allow to export the content of the value into a file. If the export is not supported, then the toolbar action will be disabled.

Copy Content Object value

Copies this Content Object value to the clipboard. The copied objects can be inserted somewhere else in the content tree with *Paste*.

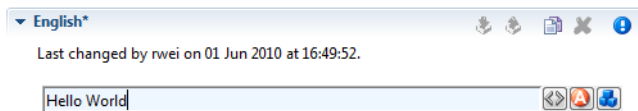
Delete Content Object value

Deletes this Content Object value from the CMS.

Content Object Value Editors

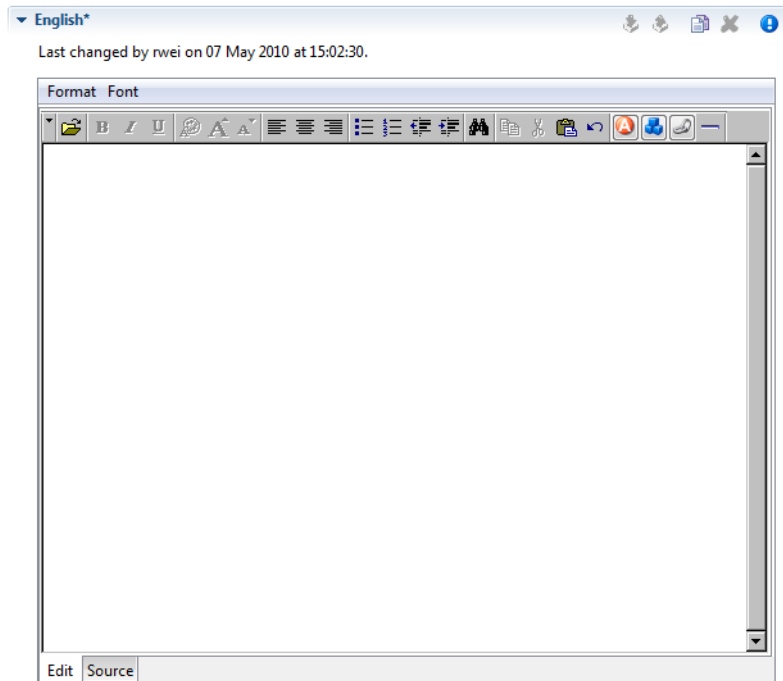
The Content Object Values area of the CMS editor contains specific editors for values of the different Content Object Types. This section briefly introduces them.

String Editor



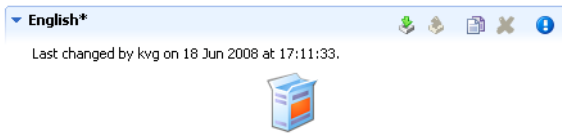
The *string* editor is simply a single-line text field; it does not accept line breaks. Content cannot be imported.

Text Editor



The text editor has two views: an *Edit* and a *Source* view. The *Edit* view is a WYSIWYG HTML text editor in which you can edit and format your text and the text appears like it will be at run-time. The *Source* view is a text only editor where you can edit the text directly in HTML. Both views are synchronized, if you edit text in the *Edit* view then the text in the *Source* View gets updated and vice versa. Content cannot be imported.

Image Editor

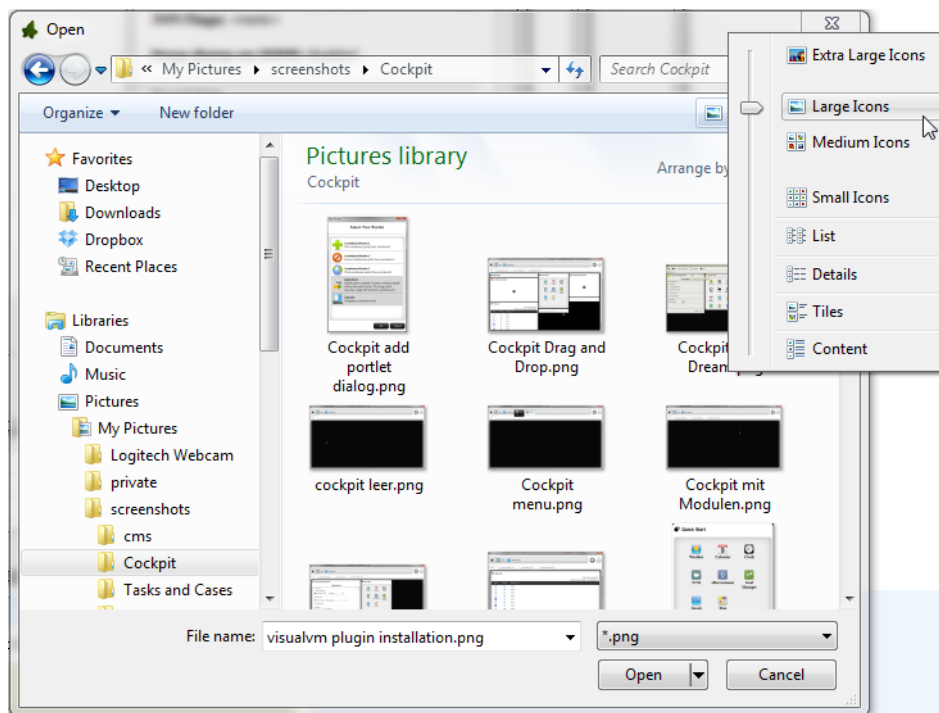


The *image* editor shows images of the types *GIF*, *PNG* and *JPG*. Content import is supported. For images that are larger than the available space just the top left corner is displayed.

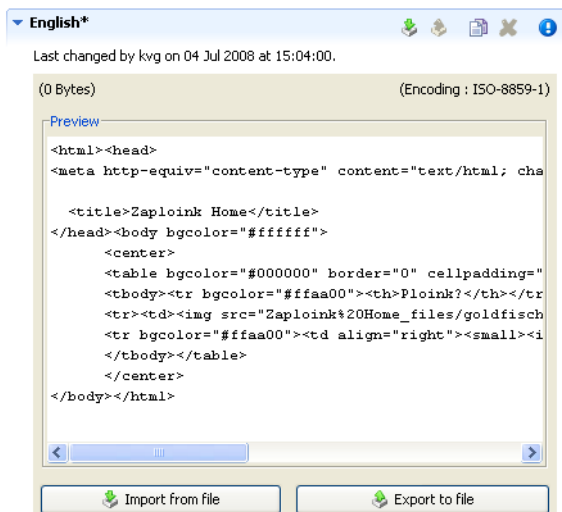


Tip

Change the file browser's view to show thumbnails of your images. This helps you to select the correct image. Depending on your operating system (version), the way to turn this on varies.



Document Editor



Preview	The document editor is used for almost all <i>document</i> content objects (basically for binary data). The editor can show a preview of textual content and will also show information about the <i>size</i> and <i>encoding</i> of the displayed content. For binary document types (e.g. PDF, audio or video) a preview is not available.
Import from File	Importing of content is supported. The import will try to infer the encoding of the imported document. If this is not possible, the user is asked to set the encoding.

Source Editor


The screenshot shows a source editor window titled "English*" with a toolbar and a status bar. The status bar indicates "Last changed by nwei on 07 Nov 2006 at 15:41:14." The main text area contains the following code:

```
<%@ page import="ch.ivyteam.ivy.page.engine.jsp.IvyJSP"%>
<jsp:useBean id="ivy" class="ch.ivyteam.ivy.page.engine.jsp.IvyJSP" scope="session"/>
<% ivy.setRequest(request); %>

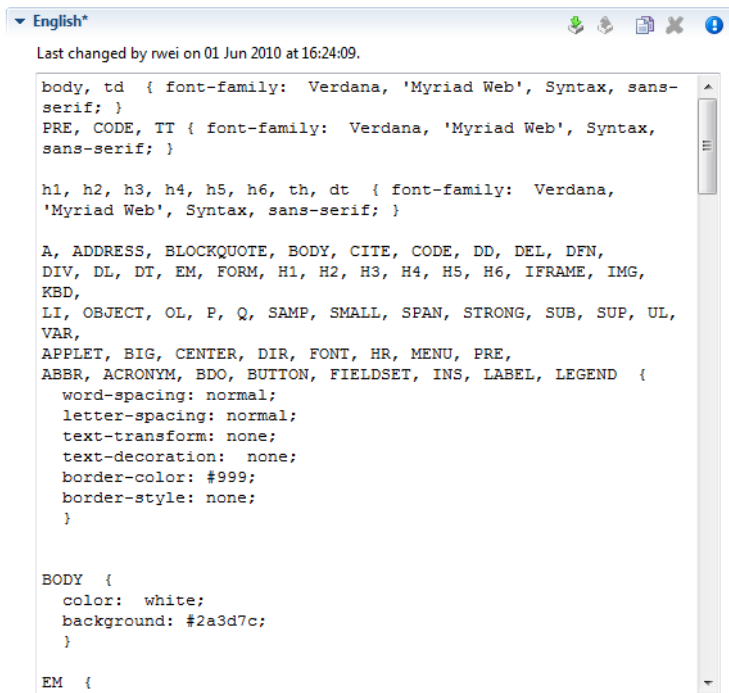
<html>
<head>
<meta http-equiv="expires" content="0">
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<link rel="stylesheet" type="text/css" href="<%=ivy.style()%" />
<title>ivyTeam at Work</title>
</head>

<body>
<table width="100%" border="0">
<tr>
<td width="100%" align="middle">
<%=ivy.cms.co("/Project/Banner")%>
</td>
</tr>
</table>
<table width="100%" border="0">
<tr>
<td>
<h1>
<%=ivy.content("Caption", "String")%>
</h1>
<hr size="4">
</td>
</tr>
</table>
```

The source editor is used to edit any kind of source text like JSP, HTML or JavaScript.

By clicking on the  button, the attribute browser opens where the user can insert *process data*. An optional *condition* may be specified as well as a suitable *format* for the type of the selected attribute (if available).

CSS Editor

A screenshot of a web browser window titled "English*" showing a CSS editor. The editor displays the following CSS code:

```
Last changed by rwei on 01 Jun 2010 at 16:24:09.

body, td { font-family: Verdana, 'Myriad Web', Syntax, sans-serif; }
PRE, CODE, TT { font-family: Verdana, 'Myriad Web', Syntax, sans-serif; }

h1, h2, h3, h4, h5, h6, th, dt { font-family: Verdana, 'Myriad Web', Syntax, sans-serif; }

A, ADDRESS, BLOCKQUOTE, BODY, CITE, CODE, DD, DEL, DFN, DIV, DL, DT, EM, FORM, H1, H2, H3, H4, H5, H6, IFRAME, IMG, KBD, LI, OBJECT, OL, P, Q, SAMP, SMALL, SPAN, STRONG, SUB, SUP, UL, VAR, APPLET, BIG, CENTER, DIR, FONT, HR, MENU, PRE, ABBR, ACRONYM, BDO, BUTTON, FIELDSET, INS, LABEL, LEGEND {
  word-spacing: normal;
  letter-spacing: normal;
  text-transform: none;
  text-decoration: none;
  border-color: #999;
  border-style: none;
}

BODY {
  color: white;
  background: #2a3d7c;
}

EM {
```

The CSS editor is a simple text editor. You can import the content from a file.

HTML Table Editor

The HTML Table Editor is explained in the HTML chapter.

HTML Link Editor

The HTML Link Editor is explained in the HTML chapter.

Result Table Editor

The Result Table Editor is explained in the HTML chapter.

HTML Page Editor

The HTML Page Editor is explained in the HTML chapter

HTML Panel Editor

The HTML Panel Editor is explained in the HTML chapter.

Smart Table Content Editor

The Smart Table Content Editor is explained in the HTML chapter.

JSP Editor

The JSP Editor is explained in the HTML chapter.

Layout Editor

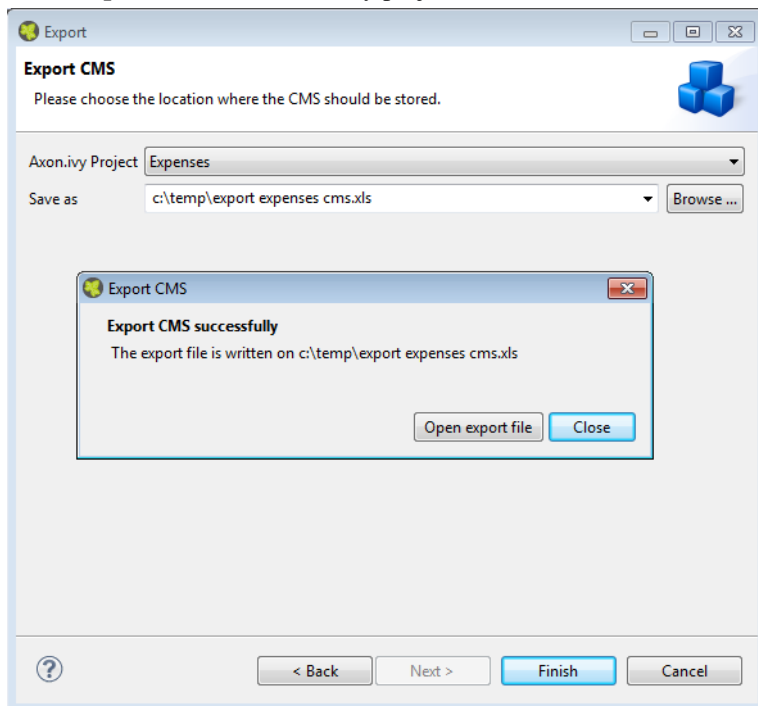
The Layout Editor is explained in the HTML chapter.

CMS Translation

The CMS is usually used for internationalization or regionalization of content. Often the necessary translations are not done by the Ivy developers but by dedicated persons within the organization or even by external persons e.g. professional translators. To simplify the exchange of the CSM content you can export the CMS into a Excel file and import it again after the translation. As long as you can import Excel files you can use your favorite translation tool for the actual translations.

Export from CMS

Click **Export...** from the Axon.ivy project tree view or from the *File* menu. Then choose *CMS* from the category *Axon.ivy*.

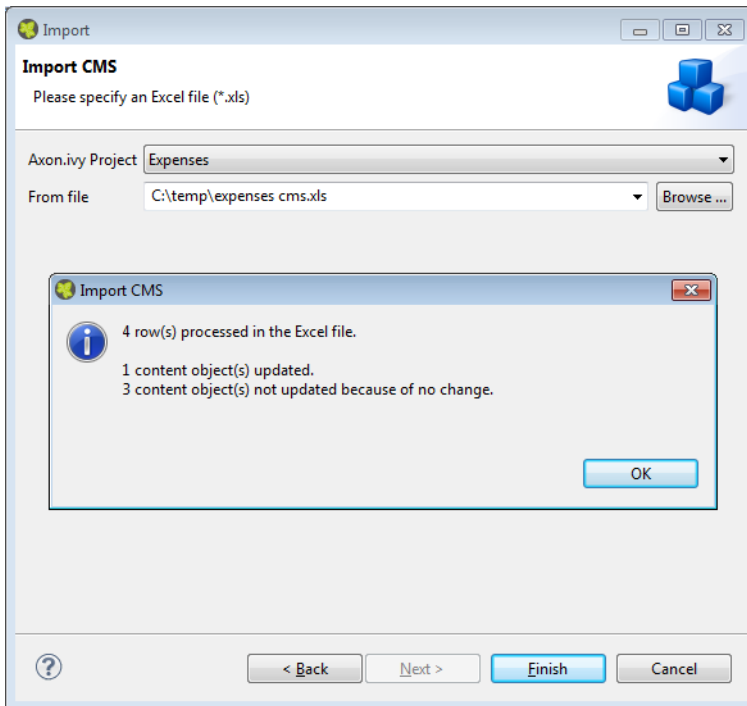


Choose which CMS you want to export and where in the file system it should be stored. After you started the export, you can open the exported file directly from the confirmation dialog.

The exported file contains one column for the name, one for the URI and one for each language of the CMS. Only *String* and *Text* types are exported.

Import into CMS

Click **Import...** from the Axon.ivy project tree view or from the *File* menu. Then choose *CMS* from the category *Axon.ivy*.



Choose in which project you want to import and where in the file system the import file comes from. After the import you will see a dialog that shows you the stats of the import like how many Content Objects were updated.

In the import file, the *URI* column is used as ID. If a Content Object with the same URI is found, then the content in the language columns in the Excel file is put in the corresponding value of the Content Object.



Note

The import can only update already existing objects or values but not create anything new. So, if you add a column for a new language or you add a new row in the Excel file with a new URI, then the CMS import will omit this data.

Chapter 6. User Interface

User Dialogs

A User Dialog is one of the two possibilities to interact with the user in a process. The other possibility are simple Web Pages as already used in pre Xpert.ivy 4.x releases. User Dialogs are provided in Axon.ivy 5.x using either the Java Server Faces (JSF) technology from Oracle or a Rich Internet Applications (RIA) technology from Canoo with the name ULC.

In Axon.ivy we use *Html Dialog* - or HD for short - as the name for a User Dialog Component built with JSF. On the other side a User Dialog Component based on the Canoo ULC Rich Internet Application Technology is called *Rich Dialog* or RD respectively.

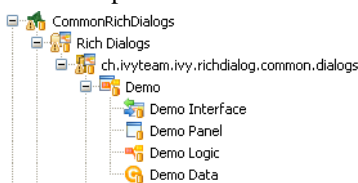
User Dialog Concept

The concept of a User Dialog follows the famous Model-View-Controller pattern. It consist of the following parts:

- Data - The internal data of the User Dialog (the model)
- View / Panel - The visual representation of the User Dialog (the view)
- Logic - The implementation of the functionality (the behavior) of the User Dialog (the controller)
- Interface - A description of the capabilities of the User Dialog
- (Data Binding) - A mapping of widget properties with data members
- (Event Mapping) - A mapping of an UI event with a process in the User Dialog Logic

The logic (i.e. controller) of User Dialogs is implemented in a process based manner. This means that all the GUI events (which are generated by the user who interacts with the dialog) are handled by means of a corresponding UI processes in the logic of the User Dialog component. So the behaviour of the User Dialog is not implemented by writing source code in a programming language (such as Java, Visual Basic or C#) but rather by graphically modelling a process logic in Axon.ivy.

The multi-part structure of a User Dialog becomes also evident when looking at it's representation in the Axon.ivy project tree:



Interface

The interface of a User Dialog defines its behavior in an abstract way and independent of it's implementation. In other words it defines *what* a User Dialog is capable to do. In more detail, the interface defines *Start Methods* and *Methods* of a User Dialog.

The interface is edited and defined using the Interface Editor.

Logic

The logic of a User Dialog defines *how* a User Dialog performs its work by means of a process model. For each *UI Event* (triggered by the actual user, e.g. by clicking on a button) and for each *Start Method* and *Method* defined on the interface you may implement a process to handle these events.

The logic of a User Dialog is edited and defined using the Process Editor.

To build the logic of a User Dialog the Process Editor offers a set of process elements that is somewhat different from the standard set. The extra *User Dialog* drawer of the Process Editor palette contains elements, which can only be used within User Dialog logic:

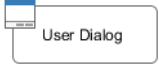










Icon	Title	Short Description
	User Dialog	Opens another User Dialog
 UD Init Start	Init Start	Invoked when the User Dialog is started. This element is executed at most once and initializes the User Dialog and its data.
 UD Method Start	Method start	Invoked when one of the methods declared in the User Dialog Interface is called.
 UD Event Start	Event Start	Invoked when a mapped widget event is received from the view of the User Dialog.
	Script	Encapsulates <i>IvyScript</i> code or changes in the User Dialog data.
 UD Process End	Process End	Ends a User Dialog UI process.
 UD Exit End	Exit End	Exits and closes this User Dialog and continues with the calling process (if opened synchronously).

Table 6.1. Process elements only available in User Dialog Logic

On the other hand, some elements of the *Dialog & Control* drawer are missing, because they are forbidden:

Icon	Title	Short Explanation
 Start	Request Start	The normal Request element is replaced with the User Dialog start element.
	Web Page	The Page element is HTML-specific and thus not available in the User Dialog logic.
 Tasks	Tasks	The Tasks element is not available because role change and task data persistence can only happen between User Dialogs.
 Task	Task	The Task element is not available because role change and task data persistence can only happen between User Dialogs.






Icon	Title	Short Explanation
 Program Start	Event Start	Start Event Beans are currently not supported inside User Dialogs.
 Wait	Intermediate	Intermediate Events are currently not supported inside User Dialogs.
 Call & Wait	Call & Wait	Call & Wait are currently not supported inside User Dialogs.
 End	Process End	The regular Process End element is replaced with the <i>User Dialog End</i> element.
 End Page	End Page	The End Page element is HTML-specific and thus not available in the User Dialog logic.

Table 6.2. Process elements that are *forbidden* in User Dialog Logic



Warning

When invoking Callables from inside a User Dialog you have to bear in mind, that the Callable process will be executed within the scope of the User Dialog that executes it, i.e. the same restrictions apply as if the Callable was defined right inside the User Dialog's logic. This ultimately means that you have to ensure that the called (business) process does not contain any of the forbidden elements mentioned above. Otherwise you will experience failures or unpredictable results during execution of the Callable process.

Data

The data of a User Dialog define its internal state (if you are familiar with the *MVC* pattern, you should consider the data as the *Model* of a User Dialog). The data of a User Dialog has private scope (i.e. is not visible from outside). Access can be granted by defining and implementing methods that return or manipulate internal data.

The data of a User Dialog is edited with the Data Class Editor.

User Dialog Interface Editor

Overview

The User Dialog Interface editor is used to define the *API (Application Programming Interface)* of a User Dialog. Since User Dialogs are components which are intended for reuse, they must define a stable interface on which other clients (i.e. processes or User Dialogs) can rely upon. An interface is defined independently from the implementation of the User Dialog and therefore separates the way of *how* a User Dialog performs its work from the declaration of *what* it is capable to do.

Accessibility

Axon.ivy Project Tree -> double click on the *Interface* node below a User Dialog in the tree:  Demo Interface

Interface tab

The User Dialog Interface editor consists of the sections for the declaration of Start methods and Methods. Each section can have multiple entries, which can be added, edited and removed with the respective buttons. You can also edit an entry by simply double clicking on it (with the exception of the initially present default *start()* method).



Tip

It is strongly recommended to write a short description for each declared interface part in the *Description* area of the details pane. This will help clients of the User Dialog to understand the characteristics of the respective part.

Start Methods

Start methods define different entry points into a User Dialog. A User Dialog can be started with different parameters and return different values, depending on which entry point is chosen at call time.

Start methods

```

start():void
startWithData(Data):Data,User
startWithFilter(String,String,String)void
startWithList(List<String>):void
  
```

Annotations in the image:

- Output / Return Parameter Types: points to the return type `void` of `start()`.
- Call / Input Parameter Types: points to the parameters `(String,String,String)` of `startWithFilter`.
- Start Method Name: points to the method name `startWithList`.

Buttons: Add, Remove, Edit

When a new *Start method* is added (or edited) you must provide a *name* as well as *input* and *output* parameters. The name of the method is entered on the first page of the opened wizard.

Add a start method

Method name

A method already exists with this signature
(The case doesn't matter, neither the return type of the method)

Name: start

< Back Next > Finish Cancel

The second page of the wizard is used to define the *input* and *output* parameters of the method. Both lists may be left empty. By clicking on the *add* button a new entry can be generated. Each method parameter consists of a *name*, a *type*, an optional *description* and the definition of whether *null* should be accepted at this position or not.

Edit a start method

Method parameters

Specify the input and output parameters of the method (optional)

Input Arguments

Name	Type	Description	null?
data	testcasesm11.Data	A data object from the ...	no

Buttons: Add, Remove, Up, Down

Output Arguments

Name	Type	Description	null?
resData	testcasesm11.Data	The modified data object	no

Buttons: Add, Remove, Up, Down

< Back Next > Finish Cancel



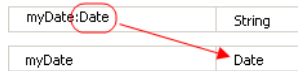
Tip

If the *name/parameter* combination (the so called *signature*) of the Start method as defined so far is identical to the signature of another Start method, then a warning will be displayed. The warning disappears when either arguments are added or argument types are changed or if the name of the method is altered accordingly, i.e. the signatures are no more identical.



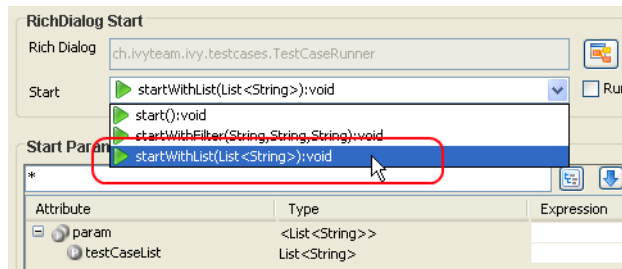
Tip

You may already specify the type of the parameter here by adding a colon ':' to the parameter name, followed by desired type (e.g. **myDateParameter:Date**). When only adding a colon to the name without a type, the data type selection dialog will appear.



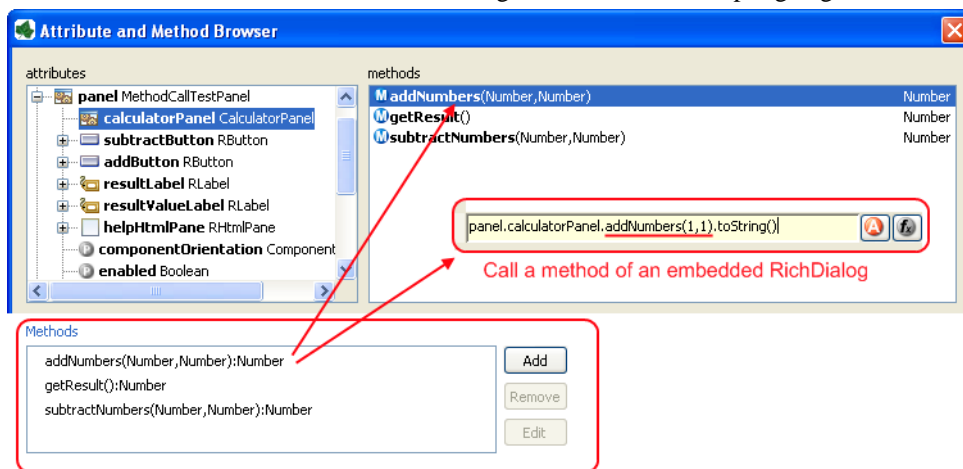
Note

Start methods defined in the User Dialog Interface can be selected inside a User Dialog Process element when the respective User Dialog is chosen to be started.



Methods

In the *Methods* section of the User Dialog Interface Editor the regular interaction methods of a User Dialog are defined. Those methods become available when a User Dialog is accessed with scripting, e.g. when used as an *embedded* User Dialog.



The declaration of *Methods* is absolutely similar to the declaration of *Start methods*, with the sole difference that a *Method* can only have a single return parameter (or none).

Metadata tab

You can define a textual description, a set of tags (keywords) for each User Dialog. These can be searched by the Ivy search page.

Description A description of the User Dialog

Tags The tags are a space separated list of keywords used to categorize User Dialogs. We suggest to define a vocabulary of tags within your team/company to always use the same terms.

New User Dialog Wizard

Overview

The New User Dialog wizard lets you create a new User Dialog. This can be a Rich Dialog, an Html Dialog or an Offline Dialog.

The wizard creates several resources for the new User Dialog:

View	The visual component of the User Dialog (different technologies are possible).
Process	The Process that contains the logic of the User Dialog.
Data Class	The Data Class that holds the data of the User Dialog.
Interface	The Interface defines the ways of interaction with other User Dialogs or business processes.

Accessibility

File -> New -> Rich Dialog

or

File -> New -> Html Dialog

Page 1: Dialog Definitions

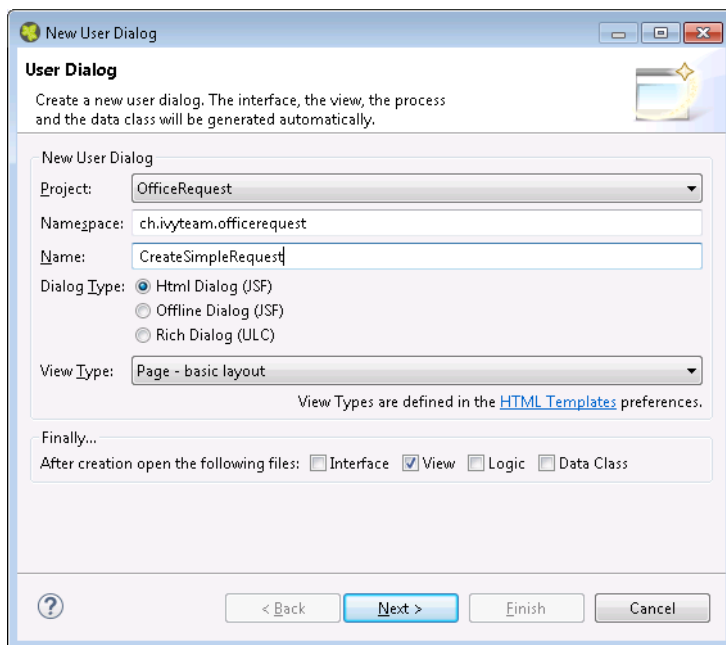


Figure 6.1. The New User Dialog Wizard Page 1

Project name	Choose the project that your User Dialog belongs to.
Namespace	Enter a namespace for your User Dialog. Use namespaces to group your User Dialogs. The grouping hierarchy is separated by the dot character. This is a similar concept as <i>packages</i> in the Java programming language.

- Name Enter the name of the User Dialog that you want to create.
- Dialog Type Select the type of the User Dialog that you want to create.



Tip

Since *Html Dialog* as well as *Offline Dialog* both base on JSF technology, it's possible to switch between those two dialog types after creation. Whereas it's not possible to switch from or to a *Rich Dialog* once it's created.

- View Type A view type defines the base layout of a User Dialog. Depending on the dialog type the view layouts vary and have the following attributes:

For the dialog types *Html Dialog (JSF)* and *Offline Dialog (JSF)* the view type could be selected from a predefined list of layouts. The list contains page and component layouts. Use a page layout for a standalone Html page, select a component layout to create a reusable Html component. See the corresponding chapter layouts and templates for more information.

For the dialog type *Rich Dialog (ULC)* one of the following view types could be selected: *GridBagLayoutPane*, *BoxLayoutPane* and *BorderLayoutPane*.



Tip

Changing the view type of a Rich Dialog after it has been created may be tricky, therefore you should be aware of layout requirements that your new Rich Dialog might have. Select a *GridBagLayout* if you're not sure, it is the most flexible and used layout.

Page 2: Dialog Data

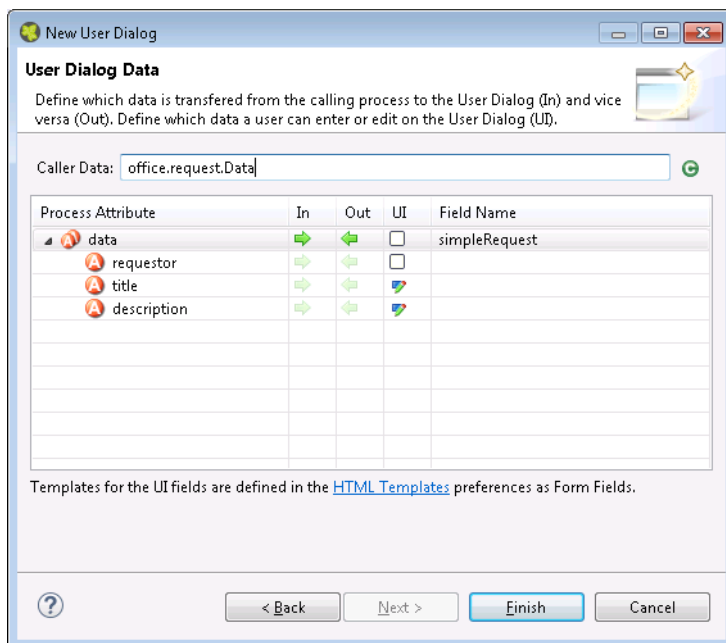


Figure 6.2. The New User Dialog Wizard Page 2

This page provides the functionality to create necessary configuration of a User Dialog simple and quickly. The starting point is a data class, e.g. the data class of the process that calls the User dialog. Based on this data class the following configuration could be created:

- Start-Method A start method is created based on the selected In and Out attributes. The necessary parameter mapping will be automatically generated.

Data Class fields	For each selected In/Out attribute a field will be created in the Data Class of the User Dialog. The name of the created field could be defined in the column 'Field Name'.
Form (for Html/Offline Dialogs)	For each selected UI attribute a form field is created in the View. E.g. for a field birthday, of type Date, a Label and a Datepicker will be generated.



Tip

Create a User Dialog in the context of a User Dialog Process Element: If the New User Dialog Wizard is opened on a User Dialog Process Element, the initial Caller Class will be the class of the calling process and the in/out parameter mapping from the process to the User Dialog and back will be generated automatically.



Tip

Create a User Dialog in the context of a Data Class: If the New User Dialog Wizard is started via the context menu on a Data Class, the initial Caller Class will be the selected Data Class.

Html Dialogs

An Html Dialog (in the following abbreviated as HD) is one of two possibilities to implement a User Dialog Component. HDs are implemented using the Java Server Faces technology from Oracle

This means, that the view of an HD is defined with the means of an XHTML document and that it is displayed in a web browser.

PrimeFaces JSF Component Library

Axon.ivy is bundled with the JSF component library PrimeFaces, an open source JSF component library developed by Prime Teknoloji. It provides a collection of mostly visual components (widgets). These can be used by JSF programmers in addition to the small set of basic components that are shipped with the core JSF platform. A very good starting point to learn more about PrimeFaces can be found in the PrimeFaces Showcase. Detailed PrimeFaces widget API documentation can be found in the PrimeFaces VDL doc.



Tip

The **Html Dialog Editor** supports PrimeFaces during design time. This means that you can profit from code completion support, tag validation, structured properties in the property view and a graphical representation in the preview part for all PrimeFaces widgets.

The elements of the PrimeFaces library are introduced with the `<p:>` namespace on your XHTML page. In addition also the **primefaces-extension** `<pe:>` and **primefaces-mobile** `<pm:>` widget libraries are included in the Axon.ivy installation.



Note

It is also possible to install and use additional JSF libraries. To do so you copy the concerning .jar file into the folder `/webapps/ivy/WEB-INF/lib` of Axon.ivy Designer and Axon.ivy Engine respectively. Then you have to add a namespace attribute `xmlns:xx` on your html pages to use the widgets.

Themes

With themes the visual appearance of the application such as the color scheme and the decoration of components can be changed. PrimeFaces comes with a number of predefined themes where you can choose from. Or you can create your own theme using the theme generator tool jQuery ThemeRoller. To learn more about PrimeFaces themes, the web site PrimeFaces Themes is the right starting point:

The theme called **modena-ivy** is configured as default. However, you can easily configure another default theme with the following steps:

- The PrimeFaces Community Themes are already included in the product. To use a own theme copy your themeXY.jar file into the folder `/webapps/ivy/WEB-INF/lib` of Designer and Engine respectively
- Edit the theme setting parameter `primefaces.THEME` in the file `/webapps/ivy/WEB-INF/web.xml`
- Restart Axon.ivy



Tip

By using the `ch.ivyteam.ivy.jsf.primefaces.IvyPrimefacesThemeResolver` the theme can be configured by application and session. See *Engine Guide > Miscellaneous > Html Dialogs > Primefaces Theme*

Html Dialog Data Binding and Event Mapping

An Html Dialog follows the model-view-controller pattern of the Axon.ivy User Dialog concept. So part of an implemented HD is a data class (the model) whose data fields can be bound to widget properties of the view. To define such a binding, Axon.ivy provides the special object **data**.

On the other hand, the controller part of an Html Dialog is implemented by a series of UI processes that can be mapped to events on the view such as mouse clicks. To define such an event mapping, Axon.ivy provides the keyword **logic** to call an event process or a method process in the logic.

Look at the following small code sample of a form with a mapped data attribute on an input text field and a button with a bound event process:

```
<h:body>
  <h3>My JSF Form</h3>
  <h:form id="myForm">
    <p:outputLabel value="#{ivy.cms.co('/labels/street')}}" for="street" />
    <p:inputText value="#{data.address.street}" id="street" />
    <p:commandButton value="#{ivy.cms.co('/labels/submit')}}" actionListener="#{logic.submit}"/>
  </h:form>
</h:body>
```

Data Class Auto Initialization

Data Classes are automatically initialized if an Html Dialog sets a property on it.

E.g. If `data.address` is null and a form is submitted with a value for `data.address.street` then a `data.address` object is automatically created.

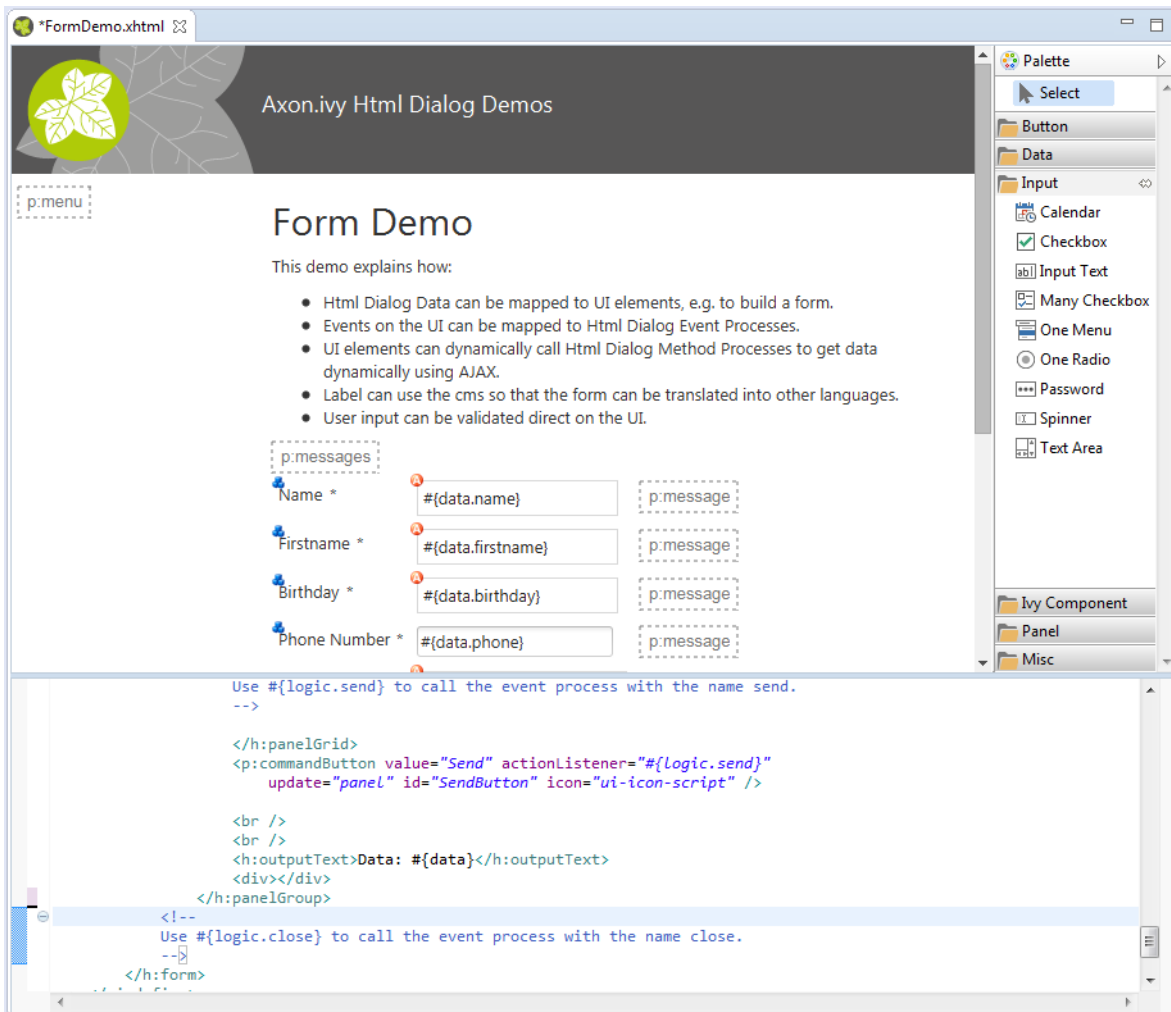
See also Public API `ch.ivyteam.ivy.scripting.objects.jsf.el.AutoInitializable`.

Html Dialog Editor

Overview

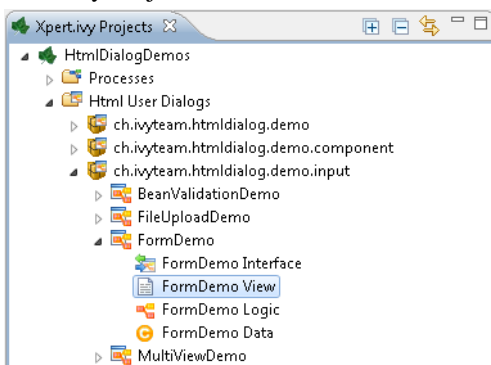
The Html Dialog editor allows to implement the view of an Html Dialog (i.e. the *JSF view*). The Html Dialog editor consists of two views, the source view and the graphical view. In the source view you can read and edit the JSF (or xhtml) source in a text-based editor. In the graphical view you can preview and edit the visual representation of the JSF page. Both views are linked to each other and every change is synchronized to the other view. So a change in one view is automatically reflected in the other one, e.g. if I change the text of a h1 HTML element in the source view then the design view is immediately updated and shows the new text. There are options to arrange the two views in horizontal or vertical panes or to show only one of them.

The third element is a palette with drawers for the most important Primefaces and JSF components and widgets that can be used in views. Such components/widgets can be dragged from the various palette drawers and then be dropped onto both the design view or onto the source view As well the rearrangement of already positioned widgets is possible in both views.



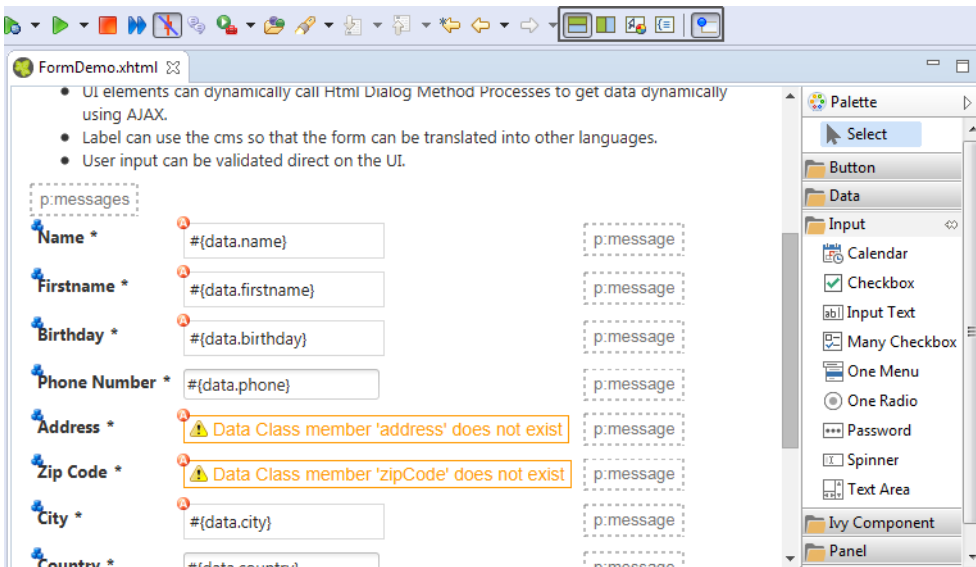
Accessibility

Axon.ivy Project Tree -> double click on the *View* node of a Html Dialog:



Graphical View

The graphical view of the Html Dialog editor allows to compose an Html Dialog view in a *graphical* mode by selecting a UI element from the palette and positioning it on the view. Similarly, already positioned elements can be dragged to another position on the view, simply by selecting and dragging them with the mouse. In the same way, just select an element and press the *delete* key to remove an element from the JSF page. As the graphical and the source views are linked together, all these actions are synchronized to the source view.

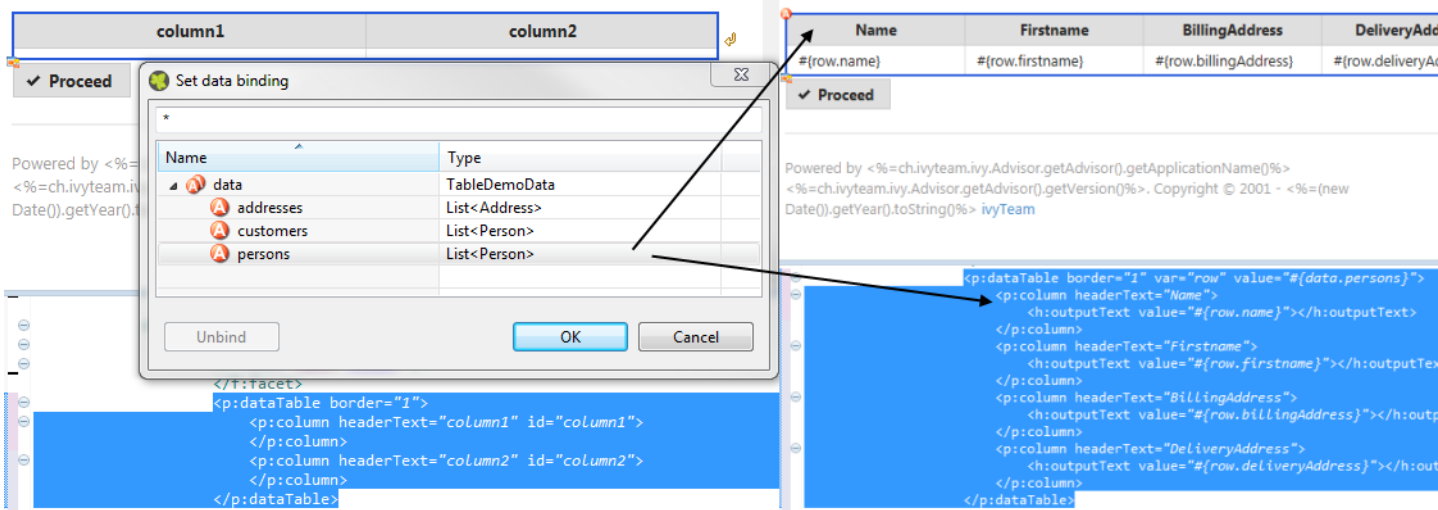


Tip

The graphical view displays the JSF page as realistic as possible. It also shows all the content from the template or from includes. But you cannot select or edit these elements.

Default Actions

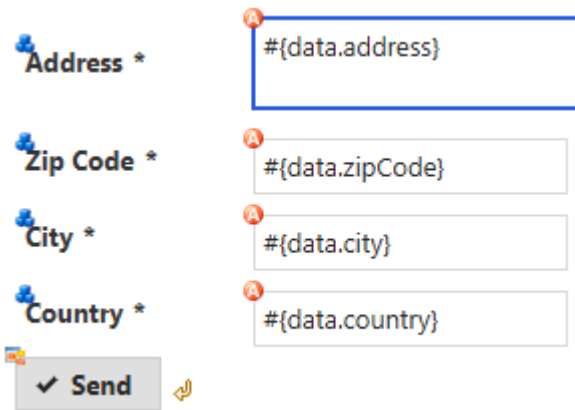
If you double click on one of the most important elements (the ones that you find in the palette) then the default action of this element is triggered. It depends on the element what happens. For example for a Primefaces OutputLabel you can edit the text of the label or choose a CMS content object for it. On the other hand, for the Primefaces DataTable you can select which list from the data class will be used as data source for the table.



Visual Markers

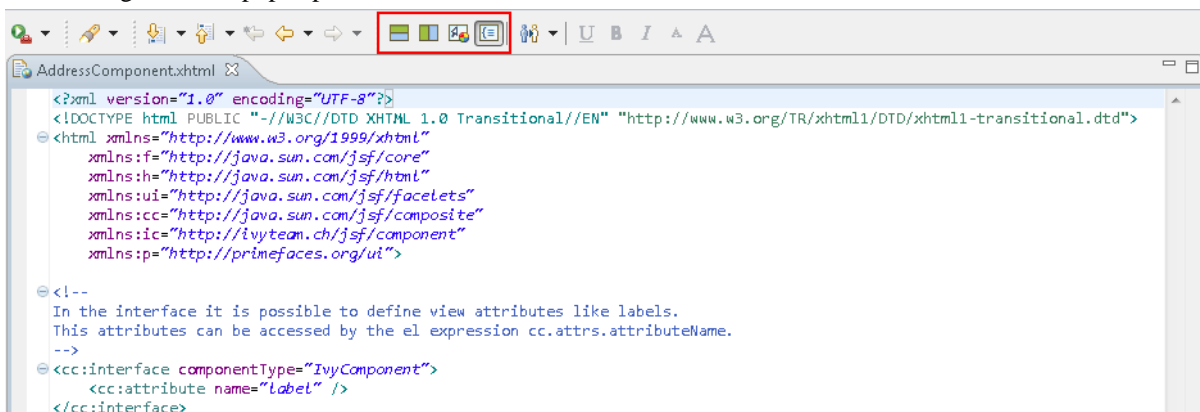
The graphical view displays overlay markers for some elements:

- CMS markers are displayed if you use the CMS for displaying text or an image. This helps you to verify very fast whether your JSF page is properly translated/internationalized.
- Data binding markers are visible when the value of an input element is bound to a data element with an EL expression. You can use these markers to verify whether all your inputs are bound to data or a backing bean.
- You see Logic mapping markers if you call a Html Dialog logic element in a button or link. Use these to verify if all your actions are properly mapped to logic elements.



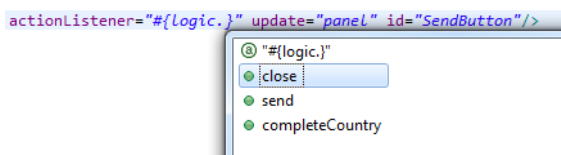
Source View (Code)

For each element that has been dropped on the view the corresponding code is generated in the source. Alternatively you can use the auto completion support in the source editor. Just enter the first letter(s) of a valid code fragment and a selection list of matching elements pops up.



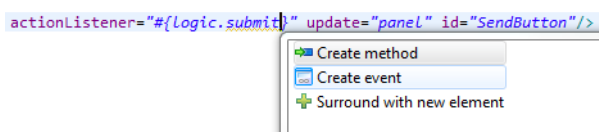
Content Assist (Ctrl+Space)

In addition to the auto completion support you get further assistance for writing expressions if you press **Ctrl+Space** on an expression to get a pop-up with context aware list of proposed code fragments to select from.



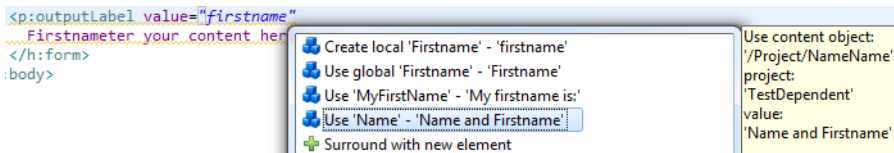
Quick Fix (Ctrl+I)

There are Quick Fixes available to create missing events, methods and data attributes on the current Html Dialog. Simply press **Ctrl+I** on a **logic** or **data** expression respectively.



CMS Quick Assist (Ctrl+I)

There are Quick Assists available to create or use content objects in the current Html Dialog. Simply press **Ctrl+I** on a text attribute or on text between xml tags.

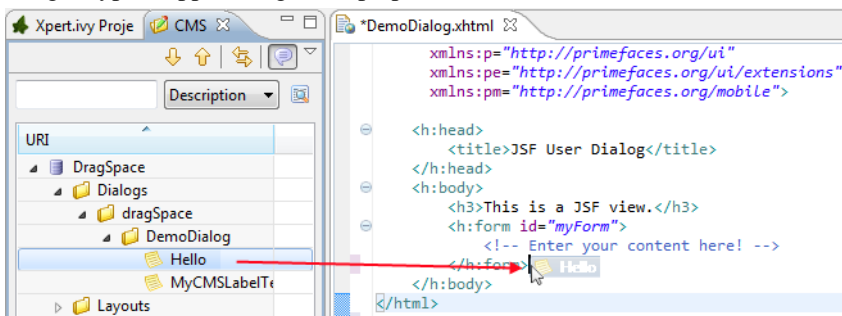


Tip

If a new content object is created with the Quick Assist you can directly rename the created content object in the Html Dialog Editor. Just type a new name and confirm with **Enter**, to abort the renaming press **Esc**.

CMS Drag & Drop support

CMS contents can directly be dragged from the CMS View into the Html Dialog Editor. The dropped content will be automatically converted into a valid JSF tag or EL-expression. Currently the content types String and Text as well as all Images-types support drag & drop operations.



Linking to CMS content (F3 or Ctrl)

There is a shortcut to navigate to CMS content. Simply press **F3** on a **ivy.cms.co(...)** expression to jump to the corresponding object in the CMS. Alternatively press **Ctrl** and click on the link.

Linking to Data Class (F3 or Ctrl)

There is a shortcut to navigate to a data class definition. Simply press **F3** on a **data.xyz** expression to jump to the corresponding Data Class. Alternatively press **Ctrl** and click on the link.

Linking to Logic (F3 or Ctrl)

There is a shortcut to navigate to a logic element definition. Simply press **F3** on a **logic.xyz** expression to jump to the corresponding logic element. Alternatively press **Ctrl** and click on the link.

Properties View

Together with the Html Dialog Editor you will want to use the Eclipse properties view to define attributes for the UI elements of your dialog. Simply switch to the **Process Development Perspective** that will display the properties view at the bottom left area of the workbench window.

Html Dialog View Types

An Html Dialog is either a page or a component. Both are complete Html Dialogs and have therefore their own view, model (data class) and controller (logic). This concept allows to build up component oriented UI design.



Note

The templates for page and component are configured in the Html Preferences.

Html Dialog Page

An Html Dialog page represents a full page that is finally displayed in the web browser. Therefore a page can be opened by a User Dialog Process Step.

Html Dialog Layouts

An Html Dialog Page uses an Html Dialog Layout. An Html Dialog Layout is the concept of a master page that defines a common layout for a number of similar dialogs. An Html Dialog Page references a layout and implements defined parts of it. For example the layout provides the header- and footer-area whereas the content-area is individually implemented on each dialog.

Axon.ivy brings a set of predefined layouts. The layout (together with the View Type) is chosen in the New User Dialog wizard.

For more information about templating have a look at the official JSF documentation.

Custom Html Dialog Layouts

Axon.ivy is not limited to the usage of the built-in Html Dialog Layouts. Custom layouts can be added with small effort.

In order to add a custom layout - which is a normal .xhtml file - it needs to be stored into the folder `webContent` of the project. In doing so, the custom layout can now be referenced as a layout inside an Html Dialog.

To make the custom layout show up in the New User Dialog wizard (for selection of the View Type), it must be stored in the folder `webContent/layouts` of the project.

The folder structure of `webContent/layouts` should follow the following contract:

- `webContent/layouts/[MyTemplateName].xhtml`
- `webContent/layouts/[A sub folder]/[for additional template content]`

Html Dialog Component

A component can be embedded into other Html dialog or again in another component.

View Definition

The view consists of two parts, an interface and the implementation. The interface is constituted by a `<cc:interface componentType="IvyComponent">` tag and is followed by an optional list of component attributes. The implementation part starts with a `<cc:implementation>` tag and the component attributes can be accessed with the expression `cc.attrs.attributeName`

The following code fragment defines an example Html Dialog component:

```
<cc:interface componentType="IvyComponent">
  <cc:attribute name="caption" />
</cc:interface>
<cc:implementation>
  <p:fieldset legend="Address Component">
    <h:outputLabel value="#{cc.attrs.caption}" />
    <h:panelGrid columns="2">
      <p:outputLabel value="Street" for="street" />
      <p:outputLabel value="Country" for="country" />
      <p:inputText value="#{data.address.street}" id="street" />
      <p:inputText value="#{data.address.country}" id="country" />
    </h:panelGrid>
  </p:fieldset>
</cc:implementation>
```

Usage

A component could be inserted with the `<ic:-tag`. E.g. `<ic:my.namespace.ComponentName ... />`.



Tip

In the **Html Dialog Editor** you have pretty nice tool support for inserting components. You can drag and drop an available component from the palette. You can select one from the auto completion popup list and you can define required attributes in the properties view.

Start Method

Optionally you can define the start method that should be used to start the embedded component with the attribute `startMethod`. If you do not define the start method, then a default start method will be used. Parameters of the start method can be defined by adding them as named attributes. Parameters are mapped by name, i.e. an attribute of the tag will be mapped to the start method parameter with the same name. Furthermore you can set the component attributes that you defined in the interface of the component by simply adding them as attributes of the tag too.



Note

You can not override start methods. So do not use multiple start methods with the same name but different parameter lists.

See the following code fragment that inserts a **Html Dialog** component. The start method `start(data.person.BillingAddress:address)` will be used, the current value of the data class property `billingAddress` will be used as parameter for the start method and the component attribute `caption` will be set to the value "Billing Address"

```
<h:panelGrid columns="2">
  <ic:ch.ivyteam.htmldialog.demo.component.AddressComponent
    startMethod="start" address="#{data.person.billingAddress}"
    caption="Billing Address">
  </ic:ch.ivyteam.htmldialog.demo.component.AddressComponent>
</h:panelGrid>
```

Html Dialog Preferences

In the Axon.ivy Designer preferences you can configure the templates used for the creation of **Html Dialogs** and **Offline Dialogs**.



Note

There are different templates for **Html Dialogs** and **Offline Dialogs**. Whereas **Html Dialog** templates are targeted to make use of the full JSF stack, **Offline Dialog** templates are designed to work without enduring connection to the Engine and are optimized for use on rather small mobile devices with touch input.

Accessibility

Axon.ivy Designer *Menu* -> *Windows* -> *Preferences* -> *Web* -> *HTML Files* -> *Editor* -> *Templates*

Html Dialog View Type Templates

View Type Page and *View Type Component* are the predefined view types for **Html Dialogs**. Furthermore every template with a name that starts with 'View Type' is considered as an **Html Dialog View Type** and therefore listed in the **New User Dialog Wizard**.

Offline Dialogs have only one predefined view type called *Offline View Type Page*. Custom templates for **Offline Dialogs** have to start with 'Offline View Type'



Tip

When the template for a View Type contains `<ui:composition template="\${layout}">`, it will be a template for an Html Page. Otherwise it will be a template for an Html Component.

Form Field Templates

Every template with a name pattern 'form field [Type]' (for Html Dialogs) respectively 'offline form field [Type]' (for Offline Dialogs) is considered as a form field template of the specified type. The form field templates are used during the creation of a Html Dialog by the New User Dialog Wizard.



Tip

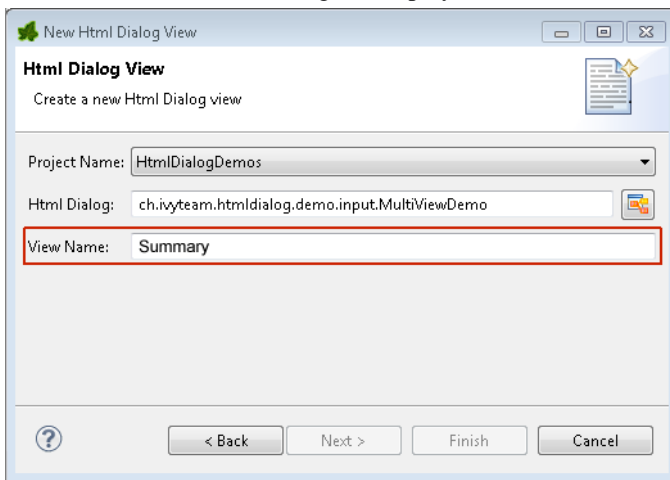
Each template can be inserted into an Html Dialog view via the auto complete function (CTRL+Space).

Html Dialog with Multiple Views

To implement a user interaction that consists of several pages (for example a wizard), one Html Dialog can have multiple views (.xhtml files). This allows to enclose a complex user interaction in one Html Dialog and to abstract it from the business process.

How to add a view

In the **New** menu in the Axon.ivy project tree you find the **New Html Dialog View** wizard to add a view to a Html Dialog. Just enter the name of the view and the xhtml file for the view is created and added to the Html Dialog. In the ivy project tree all view files of a Html Dialog are displayed below the main node of that Html Dialog.



How to switch views during runtime

If you have created a Html Dialog with several views you have to implement the navigation between the views for the user on your own. The basic solution is a `commandButton` with an `update` attribute to load the next view.

You find an example of a multi view Html Dialog in the `HtmlDialogDemos` project that is provided with the Axon.ivy Designer.

```
<h:form id="myForm">
  <p:panel header="Multi View Demo" id="panel">
    <h3>Payment - Credit Card</h3>
    <p:messages id="msgs"/>
    <h:panelGrid columns="2">
      <p:outputLabel value="Credit Card Number"/>
      <p:inputMask required="true" value="\#{data.creditCardNumber}"
        id="CreditCardNumber" mask="9999-9999-9999-9999"></p:inputMask>
```



```

</h:panelGrid>
<p:commandButton value="Next" update="myForm" action="#{logic.nextView('CreditCard')}" />
</p:panel>
</h:form>

```

Converters

Converters are used to convert number or date/time values for string representation and vice versa. If you want to display a Number or Date/DateTime process data attribute well formatted in an input widget then use the basic converters provided by the JSF core framework: **convertNumber** and **convertDateTime**.

See this code fragment from an input form:

```

<p:calendar id="Birthday" value="#{data.birthday}" navigator="true"
    required="true" pattern="dd.MM.yyyy">
    <f:convertDateTime pattern="dd.MM.yyyy" />
</p:calendar>

```

Custom Faces Converters

Custom Faces Converters can be implemented as a Java class with a specific `FacesConverter` annotation and then be used in your Axon.ivy project.

Example:

```

@FacesConverter("MyCustomFacesConverter")
public class MyCustomFacesConverter implements Converter

```

Validators

The JSF core framework provides a number of basic validators that can be used to validate the entered values in an input form.

- `validateDoubleRange`
- `validateLength`
- `validateLongRange`
- `validateRegex`
- `validateRequired`

Example code fragment from an input form:

```

<p:inputText value="#{data.zipCode}" id="ZipCode" required="true">
    <f:convertNumber integerOnly="true" groupingUsed="false" />
    <f:validateLength minimum="4" maximum="5" />
</p:inputText>

```

Client Side Validation

In some cases it makes sense to perform the validation of the entered values before they are sent to server (e.g. in an Offline Dialog). For this reason, Primefaces provides a client side validation framework. Client side validation is added as addition to the JSF validators. Thus, it can give instant feedback - even while typing - to the user. Since the JSF validators (see above) remain still active, the data is also validated on server side after the form has passed client side validation and is submitted.

Example code fragment from an input form:

```

<p:inputText value="#{data.zipCode}" id="ZipCode" required="true">
  <f:convertNumber integerOnly="true" groupingUsed="false"/>
  <f:validateLength minimum="4" maximum="5"/>
  <p:clientValidator event="keyup" />
<p:clientValidator event="blur" />
</p:inputText>
<p:message for="ZipCode" display="text" showDetail="true" />

<p:commandButton actionListener="#{logic.close}" value="Proceed" validateClient="true" />

```



Tip

It's useful, to add the client validators to the desired input field and also to trigger client side validation on the submit button. This way you make sure, that client side validation is performed during field modification, but also if the user tries to submit the form without any modification.

In order to provide a good instant feedback, a message element dedicated to the input field might be quite helpful.

Managed Beans

In Html Dialogs it is possible to communicate with normal Java objects by using *ManagedBeans*. Use the following annotations to define the lifecycle scope of the managed bean:

- `@ApplicationScoped` - the bean instance is created at creation of the application or at the engine startup and destroyed when the application is either deleted or the engine shuts down.
- `@SessionScoped` - the bean lives for the whole duration of the session
- `@RequestScoped` - an instance of the bean is created for each new request and thrown away after the response has been sent. This is the default scope that will be used when no scope is set specifically.



Note

JSF 2.0 introduced an additional scope `@ViewScoped` and offers the possibility to define custom scopes. This is basically also supported in Axon.ivy, but it is recommended to use it with care since it might not behave as expected.

In the `HtmlDialogDemo` Project that is included in the Axon.ivy Designer you find an example.

Bean Validation (JSR 303)

The *JSR 303* is a specification that defines a metadata model for Bean Validation. The fields of the JavaBean classes that are used for storing the data are annotated to describe the constraints and their validation. Experienced programmers can use JSR 303 annotations in Axon.ivy projects. The validation information will then be considered by Html Dialogs when the field of the class is bound to a widget. There is no validation information given in the *.xhtml file of the Html Dialog itself. However, the Html Dialog uses the annotations of the fields to validate the user input.

All annotations defined in the package `javax.validation.constraints` are supported. For the validation messages you can use Ivy macros to get the message content from the CMS. For example:

- `@NotNull` "means that a value is required"
- `@Size` "restricts the length of a string or array or the size of a collection or map"
- `@Max` "restricts the maximum allowed value"
- `@Min` "restricts the minimum allowed value"
- `@Pattern` "restricts a string to a given regular expression"
- `@Digits` "restricts the maximum number of digits of the integer and fraction part"

- @Future "restricts a date to the dates in the future"
- @Past "restricts a date to the dates in the past"

```
@SessionScoped
public class Person
{
    @Size(min=3, max=10, message="<%=ivy.cms.co(\"/ch.ivyteam.htmldialog.demo/BeanValidationDemo/size\");
    @NotNull(message="<%=ivy.cms.co(\"/ch.ivyteam.htmldialog.demo/BeanValidationDemo/notnull\");
    private String name;

    @Pattern(regexp="[1-9][0-9]{2}\\.[0-9]{2}\\.[1-8]([0-8][0-9]|9[012])\\.[0-9]{3}", message="<%=ivy.cms.co(\"/ch.ivyteam.htmldialog.demo/BeanValidationDemo/pattern\");
    @NotNull(message="<%=ivy.cms.co(\"/ch.ivyteam.htmldialog.demo/BeanValidationDemo/notnull\");
    private String socialSecurityNumber;
```

There will always be validation requirements for which these standard annotation will not suffice. For these cases it is possible to create your own annotation. You find an example in the HtmlDialogDemo project that is included in the Axon.ivy Designer.

```
public class Person
{
    @LicensePlate(message="<%=ivy.cms.co(\"/ch.ivyteam.htmldialog.demo/BeanValidationDemo/license\");
    private String vehicleLicense;
```

```
// re-use other existing constraints:
@NotNull
@Size(min=4, max=20)
@UpperCase // custom constraint in same package
@StartsWith(prefix="ZG") // custom constraint in same package

//only show the validation message from this annotation and not from it's re-used types:
@ReportAsSingleViolation

@Target( { METHOD, FIELD, ANNOTATION_TYPE })
@Retention(RUNTIME)
@Constraint(validatedBy = {})
@Documented
public @interface LicensePlate {
    String message() default "Field does not contain a valid license plate";
    Class<?>[] groups() default {};
    public abstract Class<? extends Payload>[] payload() default {};
}
```

Ajax Method Call API

Methods of a Html Dialog can be called with JavaScript through a REST like Interface. This Ajax method call API of Axon.ivy can be used to integrate JavaScript libraries like D3, jQuery or your own JavaScript scripts. See the Ajax Method Call Demo in the Html Dialog Demo Project:

```
<script type="text/javascript">
    // jQuery is used to intercept the click on the Button with id #hello
    $("#hello").click(function(){

        // The ivyajaxapi.js script provides the logic object,
        // which contains a function for each method available on the dialogs interface.
        // If you would like to use the REST API in a more advanced way or without jQuery,
        // have a look at the generated ivyajaxapi.js script to see how the REST API is used.
        logic.helloWorld(

            // The first parameter is a data structure, which represents the list of parameters
            {"name": "World"},
```

```

// The second parameter is a function, which is called on a successful response.
function(returnData)
{
  // returnData is a JavaScript Object containing one field for each Method return value.
  // returnData.result accesses the return value named result.
  $("#result").html(returnData.result);
};
});
</script>

```

Rich Dialogs

A Rich Dialog (in the following abbreviated as RD) is one of two possibilities to implement a User Dialog Component. RDs are implemented in Axon.ivy using a Rich Internet Applications technology from Canoo with the name ULC

Process based Rich Dialogs

Process based Rich Dialogs are components that follow the MVC based User Dialogs of Axon.ivy. But Rich Dialogs differ in some portions to normal User Dialogs.

The 4-part structure of a process based Rich Dialog becomes also evident when looking at it's representation in the project tree:

Logic

Like for all other User Dialogs, the logic of a Rich Dialog defines *how* a Rich Dialog performs its work by means of a process model. But in addition to *UI Events*, *Start Methods* and *Methods* also the *Fired Event* and *Accepted Broadcast* that are defined on the interface are implemented as a process.

For Rich Dialogs, the Process Editor offers a few additional elements. The *User Dialog* drawer of the Process Editor palette contains elements, which can only be used within the Rich Dialog logic:


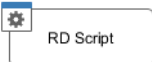
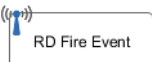
Icon	Title	Short Description
	Broadcast Start	Invoked when the Rich Dialog receives an accepted broadcast event as declared on it's interface
	Script	Encapsulates <i>IvyScript</i> code or changes in the Rich Dialog data (replaces the normal Script element).
	Fire Event	Fires one of the events declared on this Rich Dialog's Interface.

Table 6.3. Process elements only available in Rich Dialog Logic

Data

For Rich Dialogs, you have extended Data Binding support for automatic update or synchronization.

Panel

The panel represents the UI of a Rich Dialog, it defines the screen mask with which the user interacts.

The panel of a Rich Dialog is edited with the Visual Editor.

The UI of a Rich Dialog consists of multiple widgets, such as buttons, labels and text fields, which can be nested and grouped using containers with a specific layout.

Different widgets offer different events that are triggered when the user interacts with the application (i.e. clicking a button). Those events can be mapped to event process starts, where they invoke a process in the Rich Dialog Logic and thus may change internal data or perform other work.

All widgets of a Rich Dialog offer certain properties which can be manipulated using the Widget Configuration View. Properties of the panel's widgets can also be bound to the Rich Dialog data for automatic update if either the UI (e.g. user enters input on a text field widget) or the mapped data (e.g. data of a list is read from a Web Service) changes.

Data Binding

Data binding is a mechanism that allows to map (i.e. *bind*) widget properties and Rich Dialog data either two-way or one way. If a widget property is bound to a data field it will be updated whenever one of the two changes. This process happens automatically; once a binding is defined, the programmer no longer needs to update the fields explicitly.

We differentiate two kinds of data binding: *UI-to-Data* and *Data-to-UI*. UI-to-Data bindings are responsible for updating the Rich Dialog data fields, based on the current value of the UI. They are also called *downbinding*. On the other hand, Data-to-UI bindings are responsible for updating parts of the Rich Dialog panel if internal data changes. This kind of binding is also called *upbinding*.

The *downbinding* is performed whenever a process in the Rich Dialog logic is triggered (with the exception of the Rich Dialog Start method). An *upbinding* is performed whenever a process in the Rich Dialog logic ends.



Note

It is very important to understand that data binding is only performed if a process inside the Rich Dialog logic is executed! Normally an upbinding takes place the first time at the end of the Rich Dialog Start method.

Data bindings are defined either indirectly with the Widget Configuration View or directly inside the Data Binding View.

Event Mapping

Event mappings "attach" the UI of a Rich Dialog to its implementation, by defining links between widget events and processes that will be executed when those events are triggered. Every widget on a Rich Dialog Panel offers a different set of events that it can generate (e.g. clicking on a button or hitting a key inside a text field) and each of those events can be mapped individually to an *Event Start* element in the Rich Dialog Logic.

Event mappings are a potential *n:m* (many-to-many) association between the Rich Dialog Panel and the Rich Dialog Logic. In other words: different events from different widgets can be mapped onto the same process start, and also a single event can be mapped to multiple process starts.

Events are generally executed *synchronous* (see the default *Delivery Mode* below). If a *synchronous* event is executed, then the UI will be blocked until the mapped process has finished. If a *asynchronous* event is executed, then the UI will stay responsive during the execution of the mapped process. Because the UI stays responsive more events can be triggered during the execution of the first event. It is ensured that the execution order of the events is always chronological and no events are executed in parallel.

Event	Delivery Mode
Key-Event	Asynchronous
Focus-Events	Asynchronous
Others	Synchronous

Table 6.4. Default Event Delivery Mode

To change the default behaviour the *Delivery Mode* can be set for each component by API, see: `ClientContext.setEventDeliveryMode(...)`.



Note

Per default the *Event Delivery Mode* is synchronous. The *Default Event Delivery Mode* in the table above is only valid, when at least one corresponding *Event Mapping* exists for this event. Therefore, if an event listener is

added manually (e.g. in the Rich Dialog Java Code) but no corresponding event-mapping exists, then the *Event Delivery Mode* for the event is synchronous.



Warning

Please note, changing the *Event Delivery Mode* to *asynchronous* can result to inconsistent data and it will be hard to debug such problems.

Existing event mappings of a Rich Dialog can be inspected with the Event Mapping View. New event mappings are created with the New Event Mapping Wizard.

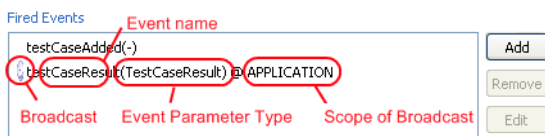
Rich Dialog Interface Editor

Interface tab

In addition to the normal User Dialog Interface Editor, the Rich Dialog Interface Editor consists of two more sections, the declaration of Fired Events and Accepted Broadcasts.

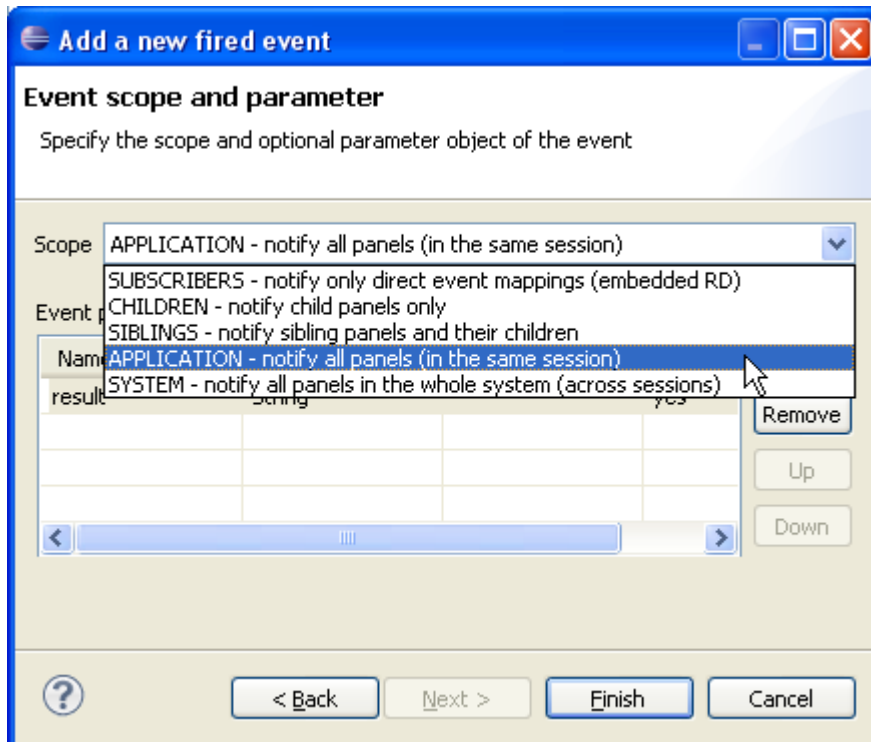
Fired Events

The *Fired Events* section defines the events that the given Rich Dialog is capable of firing during its lifetime. An event has a *name* and may carry one optional *parameter*. When fired, it is distributed to the correct recipients according to the specified *scope*.



When a new Fired Event is added (or edited) you must provide a *name* on the first page of the wizard dialog.

The second page of the wizard is used to define a *scope* for the distribution of the event and one optional *parameter* that can be passed along with the event (i.e. an attachment).



The scope of a Fired Event can be one of the following:

- SUBSCRIBERS** (no broadcast) The Fired Event will *not* be distributed as *Broadcast* event. Only selected components that had themselves *registered* for the specific event at the Rich Dialog will be notified.
- CHILDREN** (broadcast) All embedded Rich Dialogs (and their children) will be informed about the Rich Dialog Fired event. This includes the dynamically loaded Rich Dialogs that are located on any *Display* below the current Rich Dialog.
- SIBLINGS** (broadcast) All Rich Dialogs that are located on the same component hierarchy as the current Rich Dialog will be notified about the fired event. This includes the dynamically loaded Rich Dialogs that may be located on the same *Display* as the current Rich Dialog as well as any other embedded Rich Dialogs with the same parent Rich Dialog.
- APPLICATION** (broadcast) All Rich Dialogs of the current application that are instantiated in the current user session will be informed, no matter whether located on the same window or in a different window.
- SYSTEM** (broadcast) All Rich Dialogs that are instantiated for the current application instance will be informed, across all user sessions. This scope is identical to **APPLICATION**, but with the difference that *all* sessions (not just the current one) will be included in the broadcast. Check out the section about system events to learn more about this concept.



Tip

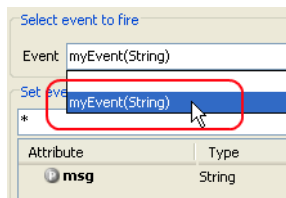
The selected scope is shown with an @ sign in the Fired Events list of the interface. If the selected scope is a broadcast scope, then a satellite dish icon is displayed in front of the entry.

Please note that the event can only be processed if it is declared with a matching *signature* (i.e. name/parameter combination) in the *Accepted Broadcast* section in the interface of any potentially receiving Rich Dialog!



Note

The Fired Events defined for a Rich Dialog are selectable inside any Rich Dialog Fire Event Element that is located in the logic of the defining Rich Dialog. The element also allows to specify the (optional) parameter object that should be sent along with the event:

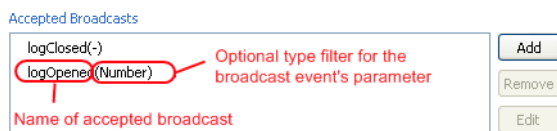


Alternatively all defined Fired Events can also be specified by means of scripting. For each declared event there is a `fireXYZ` method available on the `panel` variable of the Rich Dialog that defines it, e.g.:

```
panel.fireMyEvent(in.myMessage);
// if no parameter should be passed
panel.fireMyEvent();
```

Accepted Broadcasts

The *Accepted Broadcasts* section defines the broadcast events that the given Rich Dialog is interested in receiving. For each accepted broadcast at least a *name* must be given.



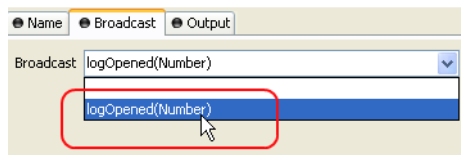
By defining the type of the optional event parameter an additional *filter* can be defined. In other words: if `logOpened(Number)` is defined as an Accepted Broadcast, then any received events with the name `logOpened` but with a parameter of type `String` (instead of the required `Number`) will be ignored.

The declaration of an Accepted Broadcast works similar to the declaration of *Fired Events* and *Rich Dialog Methods* explained above.



Note

The Accepted Broadcasts declared for a Rich Dialog are selectable inside any Rich Dialog Broadcast Start Element that are located in the logic of the defining Rich Dialog.



Please note, that each Accepted Broadcast can only be mapped once. A declared accepted broadcast event will no longer be selectable (i.e. will not be shown for selection) if it is already mapped in another Broadcast Start element.

See description of the Rich Dialog Broadcast Start Element to learn about how to access the optional *event parameter* object.

Metadata tab

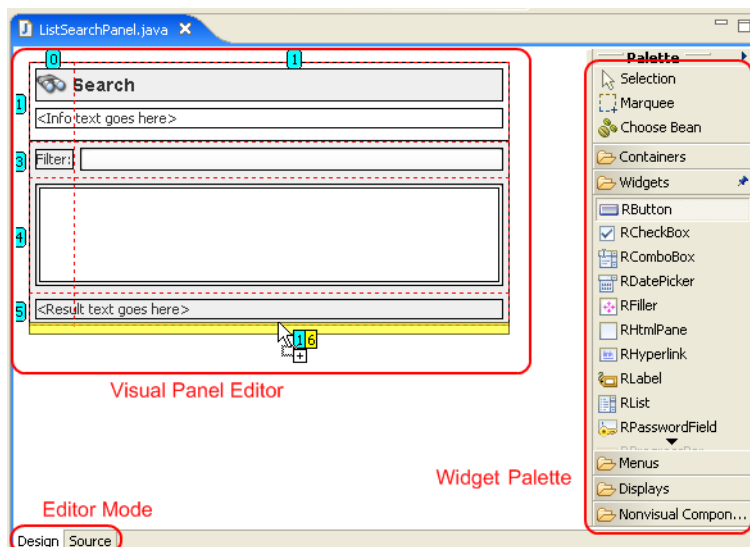
For Rich Dialogs it is possible to store a screenshot in the metadata. If *create screenshot automatically* is checked, every time the panel of the rich dialog is saved, the screenshot is replaced with the actual one.

Using the *Change* button you can replace the automatically created screenshot with your own. Don't forget to uncheck *create screenshot automatically* if you want to keep your screenshot after the next change of the panel.

Rich Dialog Editor


Overview

The Rich Dialog editor (also called *Visual Editor*) allows to compose and edit the visual representation of a Rich Dialog (i.e. it's *Panel*). The panel editor consists of an editor area and a palette with drawers. Widgets (i.e. UI elements) can be dragged from the various palette drawers and be dropped onto the panel. Rearrangement of already positioned widgets is also possible. Container widgets can be used to arrange related widgets and support different layout policies for its members.



The Rich Dialog editor consists of two areas: the actual *Editor Area* and the *Palette* where the widgets to build the UI can be selected. Both are described in more detail in the sections below.

Accessibility

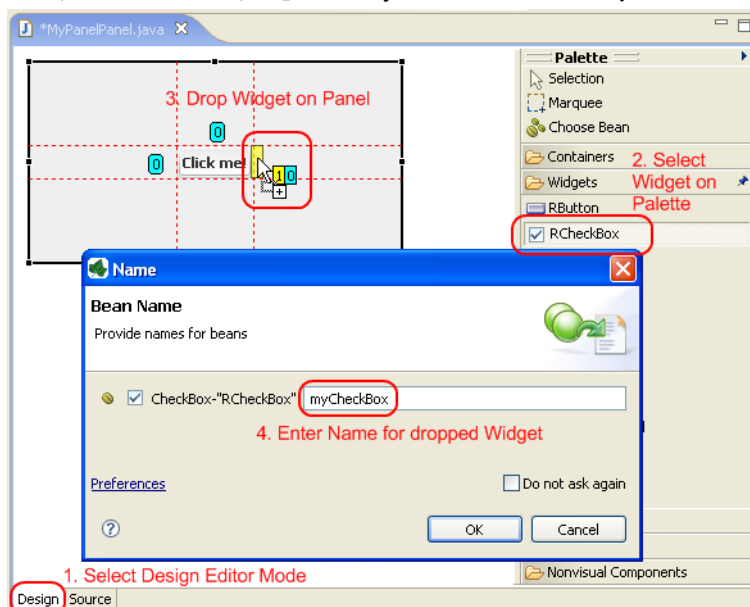
Axon.ivy Project Tree -> double click on the *Panel* node below a Rich Dialog in the tree:  Demo Panel

Editor Area

The Rich Dialog editor offers two modes for editing a Rich Dialog panel: *Design* and *Source*.

Design Mode

The design mode of the Rich Dialog editor allows to compose Rich Dialog panels in a *WYSIWYG* (what you see is what you get) mode by selecting a widget from the palette and positioning it on the panel area. Similarly, already positioned elements can be dragged to another position on the panel, simply by selecting and dragging them with the mouse. The layout of the used containers defines the way how widgets are positioned in the final end. By default, the top level container (i.e. the panel itself) has a *GridBagLayout*. If you are not used to *Layouts* and *Layout Managers*, please read the related section.



The different container layouts are explained in the following *Layout Managers* section.

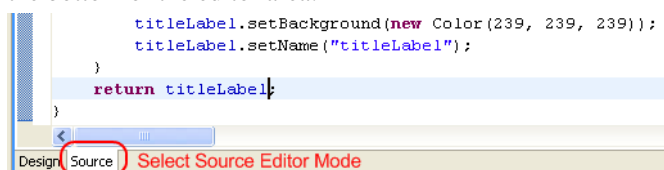


Note

The widgets that are positioned on the panel area and sub containers thereof form a hierarchical structure that can be inspected with the Rich Dialog editor Beans view. Positioned elements can also be selected in this view, which may sometimes be necessary, if e.g. a container widget is completely covered by other widgets and can thus no longer be selected on the visual representation.

Source Mode

All visual content which is created by using the Rich Dialog editor is transformed into Java code in the background. The Java code that describes and creates the panel UI can also be edited directly. To do so, you simply select the *Source* tab at the bottom of the editor area:





Warning

Be very careful if you edit the source code of a panel directly! The Rich Dialog editor requires a very strict coding convention and may fail to re-parse the source if that convention is not followed. In such a case the panel can no longer be visually edited. *In general it is strongly discouraged to edit the code of a Rich Dialog panel manually.*

Widget Palette

The palette offers some special operations.


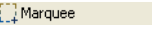
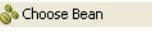

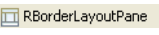

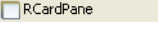

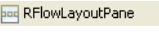
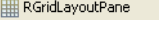
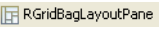
Operation	Description
 Selection	This is the default mode. If <i>selection</i> is enabled then the mouse pointer will select single elements in the editor area only. Nonetheless, you can select multiple widgets by keeping the <i>CTRL</i> key pressed down.
 Marquee	This is the multi-selection mode, it is not selected by default. If enabled, the mouse spans up a selection area which allows to select multiple widgets at once. <i>Please note that the Axon.ivy Widget Configuration currently does not permit to alter properties of multiple selected widgets.</i>
 Choose Bean	The <i>Choose Bean</i> operation can be used to insert (ULC) widgets that are not available from the palette. In fact this operation allows to insert arbitrary Java objects from classes that adhere to the Java Bean standard (see Sun Java Beans home page for more information). <i>The usage of this operation is only recommended for advanced users.</i>
	By default, the Rich Dialog editor palette drawers do close if another drawer is opened. Clicking the pin of a drawer keeps the drawer always open, even if other drawers are opened. The available vertical space is then divided evenly among all opened and pinned drawers.

Table 6.5. Palette Operations

Containers are widgets that allow to group widgets (including other containers) together using a specific layout.

Widget	Name/Link	Short Description
 RBorderLayoutPane	RBorderLayoutPane	Groups widgets with a <i>border layout</i> (north, east, south, west, center).
 RBoxPane	RBoxPane	Groups widgets with a <i>box layout</i> .
 RCardPane	RCardPane	Holds multiple sub-components in a "card stack" (only the top most component is visible).
 RCollapsiblePane	RCollapsiblePane	A panel that can expand and collapse a single sub component.
 RFlowLayoutPane	RFlowLayoutPane	Groups widgets with a <i>flow layout</i> .
 RGridLayoutPane	RGridLayoutPane	Groups widgets with a <i>grid layout</i> (cells of equal size arranged in rows and columns).
 RGridBagLayoutPane	RGridBagLayoutPane	Groups widgets with a <i>grid bag layout</i> . This is the most flexible layout pane of all.



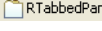
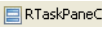
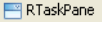

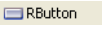
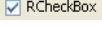
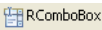
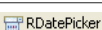


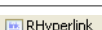

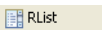
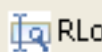
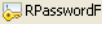
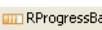
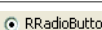
Widget	Name/Link	Short Description
 RScrollPane	RScrollPane	Holds a single sub-component. Allows to scroll that component if it is larger than the available space.
 RSplitPane	RSplitPane	Holds two components in a vertical or horizontally split area which is separated by a slider.
 RTabbedPane	RTabbedPane	Holds multiple sub-components in register tabs (only top component is fully visible, other components can be selected by tab).
 RTaskPaneContainer	RTaskPaneContainer	A container for <i>Task Panes</i> only.
 RTaskPane	RTaskPane	Similar to <i>Collapsible Pane</i> but with a title bar and collapse/expand button. Intended to be used with <i>Task Pane Container</i> .
 RToolBar	RToolBar	Groups widgets (typically buttons with an icon) in a tool bar fashion.

Table 6.6. Containers

Regular widgets are the atomic building blocks of a Rich Dialog user interface. They provide functionality to enter, edit and display data in various ways.

Widget	Name/Link	Short Description
 RButton	RButton	A button with text and/or icon.
 RCheckBox	RCheckBox	A check box to select or deselect an option.
 RComboBox	RComboBox	A combo box offers to select one item from a list of options.
 RDatePicker	RDatePicker	Enter or select a date from a pop-up calendar.
 RFiller	RFiller	A component that takes up space, either flexibly or fixed-size.
 RHtmlPane	RHtmlPane	A multi line label which can render HTML.
 RHyperlink	RHyperLink	A label with a hyperlink that can be clicked.
 RLabel	RLabel	A (single line) label with text.
 RList	RList	List of arbitrary size which allows to select elements with text and optional image.
 RLookupTextField	RLookupTextField	A text field that shows a list of hints while typing.
 RPasswordField	RPasswordField	An encrypted text field to enter sensitive data.
 RProgressBar	RProgressBar	Shows progress (relative or indefinite).
 RRadioButton	RRadioButton	A radio button (used in combination with a <i>Button Group</i> to select exactly one item from a list of options).


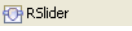
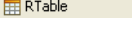
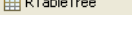
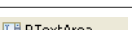
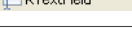
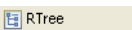


Widget	Name/Link	Short Description
 RScrollBar	RScrollBar	A scrollbar (can be used to select a position).
 RSlider	RSlider	Allows to select from a range of numerical values.
 RTable	RTable	Displays data in a spread sheet fashion, i.e. cells arranged in columns and rows.
 RTableTree	RTableTree	A combination of <i>RTree</i> and <i>RTableTree</i> where the first column of the displayed table contains a tree.
 RTextArea	RTextArea	A multi-line input text field.
 RTextField	RTextField	A single-line text input field.
 RToggleButton	RToggleButton	A button that stays on or off.
 RTree	RTree	Displays data in a hierarchical way.
 RBrowser	RBrowser	An embedded browser that can be used to display HTML pages from an URL.

Table 6.7. Widgets

Menus are widgets that offer a selection of options for the user to select. Menus can be offered either as pop up menus or as part of a menu bar on the panel.



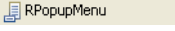

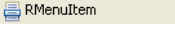
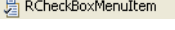
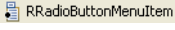

Widget	Name/Link	Short Description
 RMenuBar	RMenuBar	A common menu bar that may contain multiple menus.
 RMenu	RMenu	Groups different menu items together, intended to be used in an <i>RMenuBar</i> .
 RPopupMenu	RPopupMenu	Defines a context menu that can pop up anywhere on a component (e.g. on right mouse click).
 RMenuSeparator	RMenuSeparator	A menu item used to separate groups of other menu entries.
 RMenuItem	RMenuItem	A regular menu item for use inside a <i>RMenu</i> or a <i>RPopupMenu</i> .
 RCheckBoxMenuItem	RCheckBoxMenuItem	A menu item with a check box, use for options that can be selected or deselected.
 RRadioButtonMenuItem	RRadioButtonMenuItem	A menu item with a radio button, use to select exactly one from multiple options.

Table 6.8. Menus

Displays are place holders on a panel that serve as target locations for the dynamic loading of Rich Dialogs at runtime.

Widget	Name/Link	Short Description
 RTabbedDisplay	RTabbedDisplay	Shows dynamically loaded Rich Dialogs as tabs on a tabbed pane.


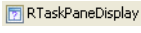
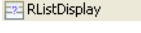
Widget	Name/Link	Short Description
 RCardDisplay	RCardDisplay	Shows dynamically loaded Rich Dialogs on a card stack, i.e. only top Rich Dialog is visible.
 RTaskPaneDisplay	RTaskPaneDisplay	Shows dynamically loaded Rich Dialogs as expandable/collapsible panels on a task pane container.
 RListDisplay	RListDisplay	Shows dynamically loaded Rich Dialogs in a scrollable list.

Table 6.9. Displays

Non visual Components are UI elements that do not have a visual representation. They may serve as supporting objects for visual components and are placed next to the panel rather than onto it. The user can not see those objects on his screen.


Widget	Name/Link	Short Description
 ButtonGroup	ButtonGroup	Logically groups exclusively selectable widgets such as <i>RRadioButton</i> . Ensures that not more than one element from the group is selected at the same time.

Table 6.10. Non visual Components

Cell Widgets are elements for table editing, used for displaying and manipulate data of a RTable.


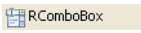
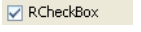
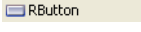
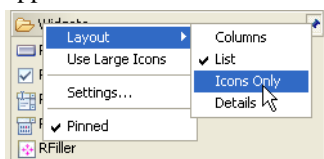
Widget	Name/Link	Short Description
 RTextField	RTextFieldCellWidget	A single-line text input field.
 RComboBox	RComboBoxCellWidget	A combo box offers to select one item from a list of options.
 RCheckBox	RBooleanCellWidget	A check box to display or change the boolean cell value.
 RButton	RButtonCellWidget	A Button to start an action with a text. The cell value are displayed as text.

Table 6.11. Cell Widgets**Tip**

If you right-click on a palette drawer or palette entry, a popup-menu opens which allows you to configure the appearance and the behavior of the palette.



For example, all entries can be displayed by their icons only. This saves a lot of space and allows you to keep all drawers open at the same time to access all widgets at once. It is recommended that you try some of the palette settings until you find a configuration that suits your personal needs. The configuration is saved and reused for all projects in this workspace.

Layout Managers

Components in a container are arranged according to the layout manager that the specific container is using. Layout managers are mainly responsible for arranging the widgets within a container, and especially for *re-arranging if the size of the container changes* (e.g. if the size of a window is changed, then the layout of all the contained components will be done again).

The Rich Dialog editor offers containers with different layouts (all links are pointing to pages of the Sun Java Swing Tutorial):

Layout Name	Short Description
Border Layout	<code>BorderLayout</code> object has five areas: NORTH, EAST, SOUTH, WEST and CENTER. If the window is enlarged, the center area gets as much of the available space as possible. The other areas expand only as much as necessary to fill all available space. Often a container uses only one or two of the areas of the <code>BorderLayout</code> object — just the center, or the center and the bottom. See Border Layout for more information.
Box Layout	<code>BoxLayout</code> either stacks its components on top of each other or places them in a row — your choice. You might think of it as a version of <code>FlowLayout</code> , but with greater functionality. Working with nested box layout containers is very similar to working with HTML tables for layouting. See Box Layout for more information.
Card Layout	The <code>CardLayout</code> class manages two or more components (usually other containers) that share the same display space. Only the component on top is visible. The stacked components that are not visible can be shown by setting properties on the <code>CardLayoutPane</code> . See Card Layout for more information
Flow Layout	The <code>FlowLayout</code> class puts components in a row and sizes them according to their preferred size. If the horizontal space in the container is too small to put all the components in one row, the <code>FlowLayout</code> class uses multiple rows. By default, if the container is wider than necessary, the row is centered horizontally within the container. See Flow Layout for more information.
Grid Layout	A <code>GridLayout</code> object places components in a grid of cells. Each component takes all the available space within its cell, and each cell has exactly the same size. If the window is resized, the <code>GridLayout</code> object changes the cell size so that the cells are as large as possible in relation to the given space. See Grid Layout for more information.
GridBag Layout	<code>GridBagLayout</code> is one of the most flexible — and complex — layout managers of Swing. A <code>GridBagLayout</code> layouts components in a grid of rows and columns, allowing specified components to span multiple rows or columns. Not all rows necessarily have the same height. Similarly, not all columns necessarily have the same width. Essentially, <code>GridBagLayout</code> places components in rectangles (cells) in a grid, and then uses the components' preferred sizes to determine how big the cells should be. See GridBag Layout for more information.

Table 6.12. Summary of different Layout Managers



Tip

Because *GridBagLayout* is the most powerful and flexible layout available, it is generally recommended to use a *GridBagLayoutPane* as container, unless specific demands are favouring another layout manager.



Warning

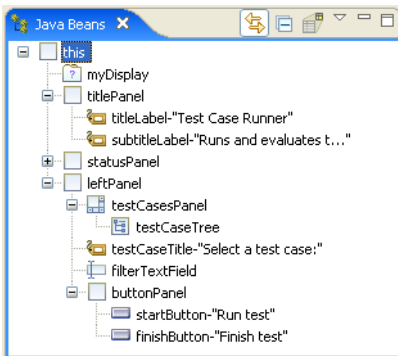
Once you've selected a layout it is not so easy to change it again afterwards. So you should quickly think about your layout requirements before selecting a specific layout.

Java Beans View

Overview

The Java Beans View (or just *Beans View* for short) is used to visualize the structural hierarchy of all widgets on the Rich Dialog. It can be used as an alternative to the Rich Dialog editor area when dropping or arranging widgets from the palette.

Inside the Rich Dialog editor it can sometimes be impossible to select a container with the mouse if it's children are maximized; i.e. completely cover the space of their parent container. In such a case the "covered" component can still be selected inside the Java Beans view. The same holds true for components that are located inside a container that uses a layout hides it's child components (examples are *RCardPane* or *RMenu*).



Accessibility

Window -> Show View -> Other... -> Java -> Java Beans

Features

Synchronize



Will synchronize the selection in the Java Beans view and in Rich Dialog editor area.

Collapse All



Will collapse the widget hierarchy tree completely.

Overview

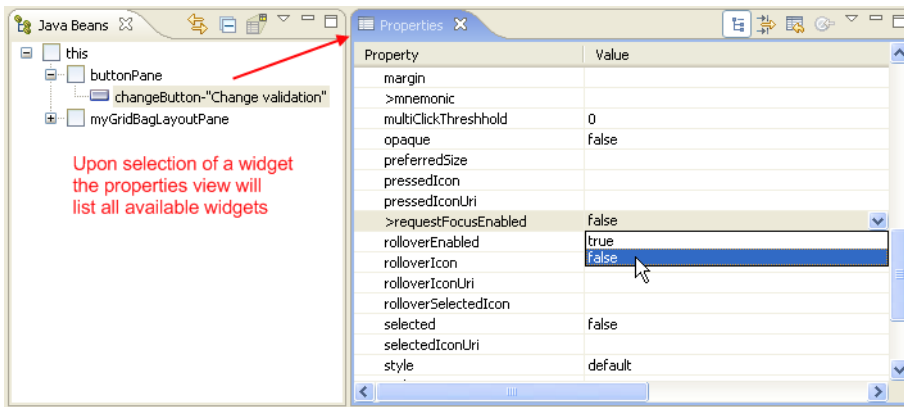


Shows an overview (thumbnail) of the edited panel. May be helpful if the Rich Dialog editor area is too small to show the whole edited panel. The view point of the editor (shown as blue area over the panel) can be dragged around (watch the Rich Dialog editor scroll while doing this). Positioning of the view point only works if editor area is smaller than the panel.

(Rich Dialog Editor) Properties View

Overview

The Properties view of the Rich Dialog editor offers an alternative way to the Widget Configuration View for setting the properties of a widget. The view will always show the available properties for the currently selected widget (Java Beans view and/or Rich Dialog editor).



Warning

The *Properties view* shows and allows to alter the complete set of properties and attributes of widgets. Most of them are of no relevance for the creation of applications with Axon.ivy. The explanation of all available properties for each widget type goes far beyond the scope of this document and is thus generally left undone.

The reason why the *Properties view* is available to the user, is that currently not (yet) all relevant properties can be edited through the Widget Configuration View. Some of the properties that are documented in the Widget Reference chapter, you *have* to set or alter inside the *Properties view* because there is currently no other way to do it.

Generally, however, we strongly discourage the usage of the *Properties view*. One reason being that it offers far too much information and at the same time is less powerful than the widget configuration view. Another reason being that some harm may be done if undocumented attributes are altered without a complete understanding of the consequences.

Accessibility

Window -> Show View -> Properties

Features

Show Advanced Properties



Will show even more properties for the currently selected widget.

Collapse All



Restores the default value for the currently selected property, if defined.

Overview



Sets the currently edited property value to *null*.

Widget Configuration View

Overview

In the Axon.ivy Widget Configuration view you can manipulate the most important configuration issues of the selected widget in the Rich Dialog editor or Java Beans view.

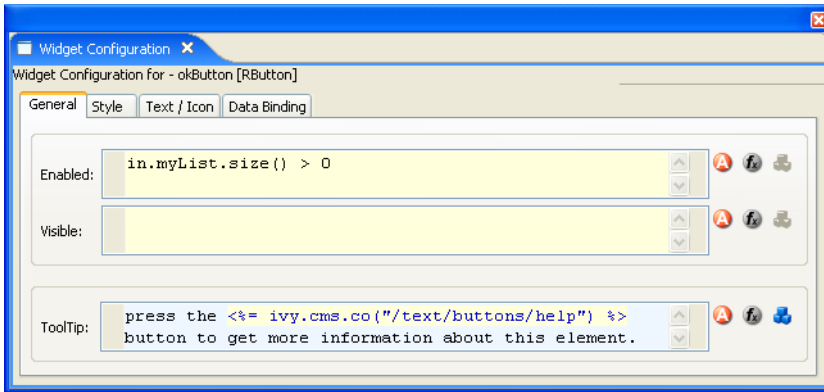


Figure 6.3. The Widget Configuration View



Note

Open the Properties View to configure advanced properties that are not visible in the Widget Configuration.

Accessibility

Window -> show View -> Widget Configuration

Features

The configuration is split up in several tabs, which depend on the selected Widget. In this section all the tabs are described.

Tab General

Overview

The General tab groups together general properties that all Widgets have in common.

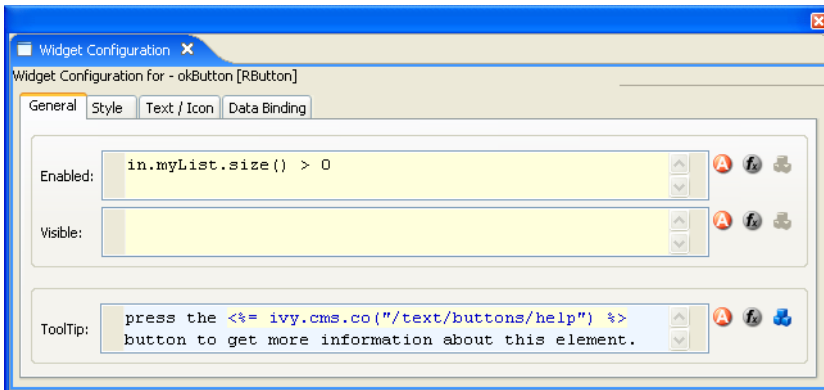


Figure 6.4. The General tab

Accessibility

Window -> show View -> Widget Configuration -> General

Features



Note

Enter IvyScript into a IvyScript text field if the background color of the text field is yellow, enter IvyScript macros if the background is blue and enter normal text if the background is white.

- Properties**
- Enabled:** Enter a IvyScript that evaluates to a boolean to set the enabling of the widget.
 - Visible:** Enter a IvyScript that evaluates to a boolean to set the visibility of the widget.
 - Tooltip:** Enter the tool tip for the widget, IvyScript macros syntax may be used.

Tab Style

Overview

The Style tab shows the graphical user interface style information for the selected widget. This includes formatting options such as colors or fonts, positioning options for alignment, filling of empty space or inset size and as well graphical options such as border style and color.

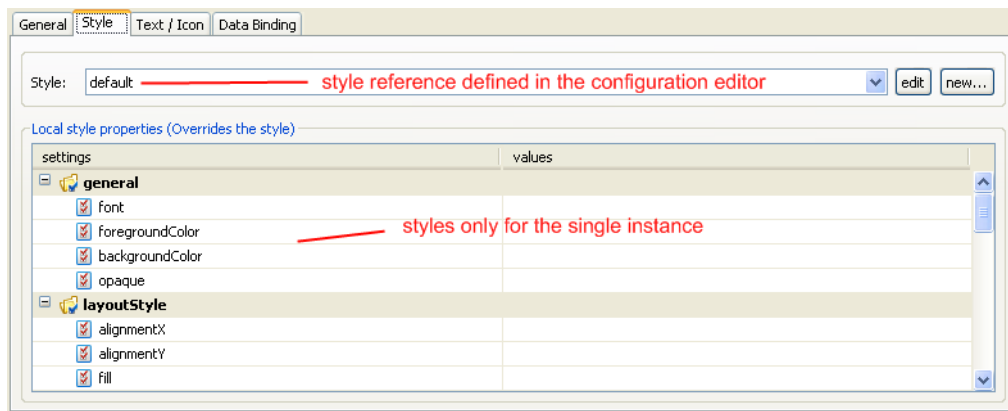


Figure 6.5. The Style tab

Accessibility

Window -> show View -> Widget Configuration -> Style

Features

In the first section you can select a reference to a predefined style configuration in your project. You are also able to select a style configuration from a dependent project. The edit button opens the configuration editor on the currently selected style. You can then edit that style or define a new one before returning to the widget's style configuration. See here for information about how to create your own style configurations.

In the second section you define style parameters that are only applied to the current selected widget. The table shows you, how the widget is configured at the moment. If the widget inherits a style from the global style configuration, then a light gray value will be displayed in the table cell.



Note

Use the configuration editor to configure styles that all widgets of the same type have in common. E.g. set the fill characteristic for all scroll panes to FILL_BOTH in the configuration editor.



Warning

Do not use the properties view to set style properties, unless the property you want to adjust is not on the widget configuration.

Data Binding Tab

The Data Binding tab shows a table displaying all data bindings that depends on the selected widget. See the Data Binding View section for more information.

Tab Text/Icon

Overview

The Text/Icon tab shows the configuration for text/icon based widgets like buttons and labels.



Figure 6.6. The Text/Icon tab

Accessibility

Window -> show View -> Widget Configuration -> Text/Icons

Features

properties **Text:** The text displayed on the widget (e.g. button)

Icon-Uri: Default icon displayed next to the text on the widget. Access the icons from the CMS via the `ivy.cms.cr` function.

Pressed Icon-Uri: The icon that should be displayed if the button was pressed. If this field is empty, then the default icon from the text field Icon-Uri is displayed.

Rollover Icon-Uri: The icon that should be displayed if the mouse is over the widget. If this field is empty, then the default icon from the text field Icon-Uri is displayed.

Selected Icon-Uri: The icon that should be displayed if the widget is selected. If this field is empty, then the default icon from the text field Icon-Uri is displayed.

Tab ComboBox Data

Overview

On the ComboBox Data tab you configure what to show in the combo box. You can configure the label, icon and the tool tip text for each entry in the combo box.

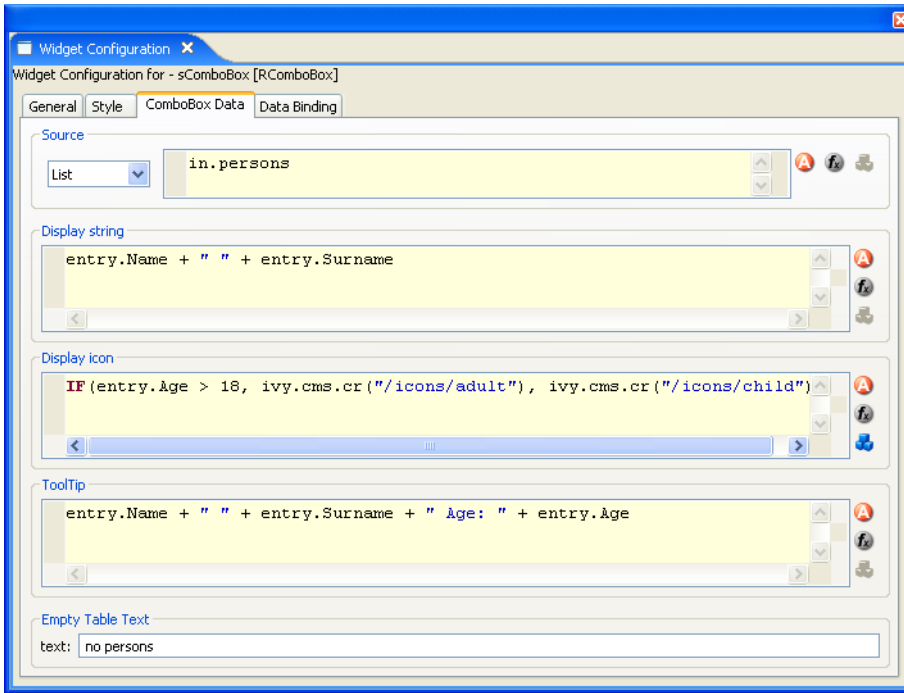


Figure 6.7. The ComboBox Data tab

Accessibility

Window -> show View -> Widget Configuration -> ComboBox Data

Features

sections **Source:** Select the type of the source for your combo box. Either `List` or `Recordset`. Define in the IvyScript text field the path to your source.

Display String: Use the IvyScript text field to enter what text to show for each entry in the combo box.

Display Icon: Use the IvyScript text field to enter which icon to show for each entry in the combo box, if any.

Tool tip: Use the IvyScript text field to enter what text to show as tool tip for each entry in the combo box.

Empty table text: Enter a plain text that will be displayed if the data source is empty.

Tab Lookup Data

Overview

On the Lookup Data tab you configure what to use in the lookup list of the lookup text field. A lookup text field shows only entries of the look list that match the search criteria specified in the `RLookupTextField`. You can set up the filter that interprets what the user types and searches accordingly. Per default a filter is set that filters the lookup list according to the user input with following characteristics. It searches case insensitive, and assumes free text between two words user types (e.g. typing "sun time" would show up lookup entries like "Sunday times", or "monsoon weather sometimes"). Right now you can configure the way the displayed text entries in the lookup list should be calculated using the `listData` (or `recordData`) just like on the combo box or on the list.

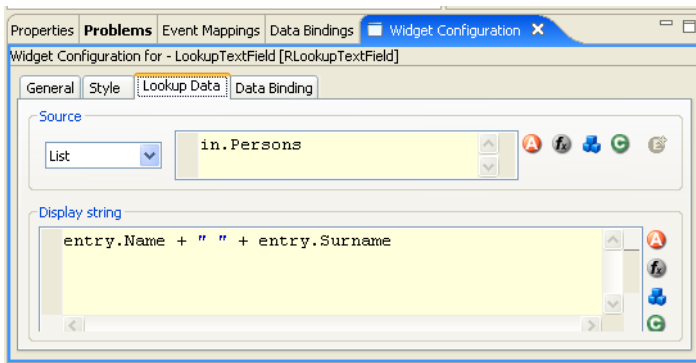


Figure 6.8. The Lookup Data tab

Accessibility

Window -> show View -> Widget Configuration -> Lookup Data

Features

sections **Source:** Select the type of the source for your combo box. Either `List` or `Recordset`. Define in the IvyScript text field the path to your source.

Display String: Use the IvyScript text field to enter what text to show for each entry in the lookup text field's lookup list (that appears when the user types a search expression).

Tab List Data

Overview

On the List Data tab you configure what content to show in the list widget. You can configure the label, icon and the tool tip text for each entry in the List.

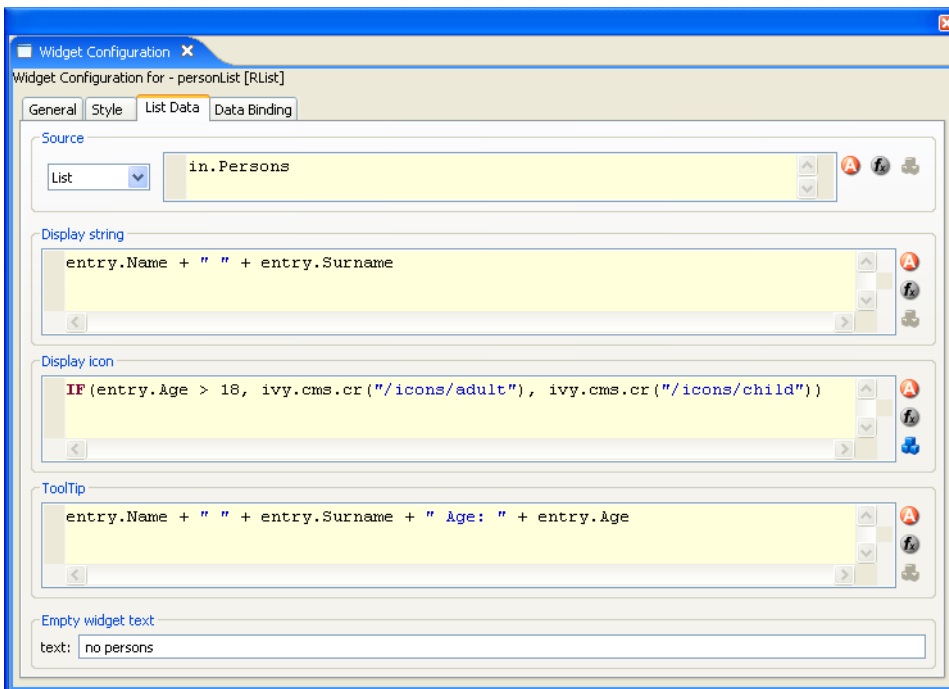


Figure 6.9. The List Data tab

Accessibility

Window -> show View -> Widget Configuration -> List Data

Features

- sections
- Source:** Select the type of the source for your list. Either `List` or `Recordset`. Define in the IvyScript text field the path to your source.
 - Display String:** Use the IvyScript text field to enter what text to show for each entry in the list.
 - Display Icon:** Use the IvyScript text field to enter which icon to show for each entry in the list, if any.
 - Tool Tip:** Use the IvyScript text field to enter what text to show as tool tip for each entry in the list.
 - Empty Widget Text:** Enter a plain text that will be displayed if the data source is empty.

Tab Table Data

Overview

On the Table Data tab you configure what content to show in the table. You can configure the data source for the table, but you can too filter the source to show only a certain part of the data source. Furthermore, you can influence the visual look and feel and set options such as label texts, icons and tool tip texts for each entry in the table.

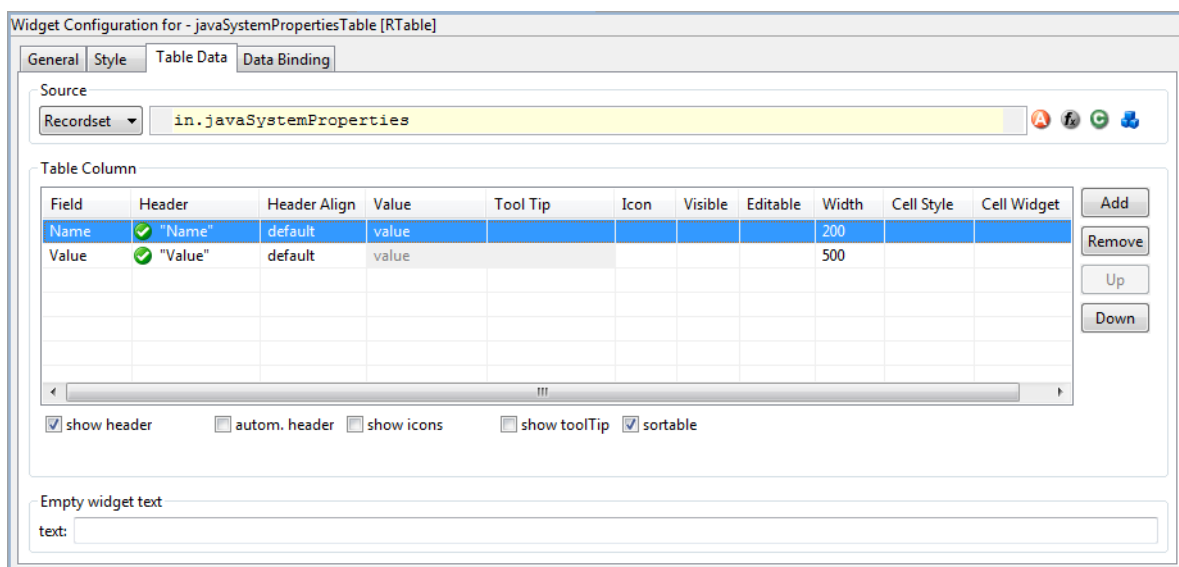


Figure 6.10. The Table Data tab

Accessibility

Window -> show View -> Widget Configuration -> Table Data

Features

- Source
- Select the type of the source for your Table. Either `List` or `Recordset`. Define in the IvyScript text field the path to your source.
- Table column:
- Columns
 - Field:** Define the name of the field of your data source to map to the current column. If your data source is a list, then the list elements (i.e. the runtime data class(es) of all list members) must contain a field with the configured name. If your data source is a recordset, then the records must have a field with the configured name. The value of the field is mapped to the variable `value` which you can use in the IvyScript text field.
 - Header:** Enter a text to display as header of the column. Use IvyScript macros if needed. example: `<%=ivy.cms.co("/labels/header")%>`. Note hat this has only an effect if the **Show Header** check box is selected.

Header Align: Select the alignment of the header text (default, left, center, right). Note that this has only an effect if the **Show Header** check box is selected.

Value: Enter a IvyScript expression to configure what label that should be displayed in the table cell.

Tool tip: Enter a IvyScript expression to configure what tool tip text that should be displayed in the table cell.

Icon: Enter a IvyScript expression to configure which icon should be displayed in the table cell.

Visible: Enter a IvyScript expression which results in a Boolean value, to configure the visibility of the column.

Editable: Enter a IvyScript expression which evaluates to a Boolean value, to configure the editable-state of the column. Note: also a simple `true` or `false` is allowed.

Width: Enter the number of pixels the column should have in width, only works if the configured `autoResizeMode` is set to `AUTO_RESIZE_OFF`

Cell Style: Select a table cell style if needed. To create new table cell styles use the configuration editor.

Cell Widget: Select a cell widget to customize the table cell rendering and editing. The cell widgets must be created in the Visual Editor, placed beside the panel. Each created cell widget must **only be used once**.

Actions **Add:** Adds a new column configuration to your table definition. The column configuration will be shown as row in the table columns configuration.

Remove: Removes the selected column configuration from your table columns configuration.

Up: Moves a table columns configuration up in the table. This means that the column moves one position to the left in the resulting table.

Down: Moves a table columns configuration down in the table. This means that the column moves one position to the right in the resulting table.

Show header: Activate this check box if you want your table to have a header row.

Autom. header: Activate this check box if you want the system to label your headers.

Show icons: Activate this check box if you want to decorate your table with icons.

Show tool tip: Activate this check box if you want your table to support tool tips per table cell.

Sortable: Activate this check box if you want your table to be sortable. Clicking on the header of a column triggers then a sorting of the whole table according to the selected column.

Empty widget Text

Enter a plain text that will be displayed if the data source is empty. Note that the column titles will be hidden when the text is shown.



Note

When an entry of a data source was changed, a call to `elementChangedAt()` on the source is needed. Otherwise the UI will not be updated. This is only required when the effected entry was not added to the list and the change of the entry has an effect to the UI.

Short example: When a list `myList` is bound to a `RTable`, after the instruction `myList.get(x).myAttribute = 123` a call to `myList.elementChangedAt(x)` is needed, otherwise the new value 123 isn't displayed in the UI.



Note

When an expression of a column definition includes the `in`-object AND a row-based variable (`entry`, `recordset` or `value`) and only the `in`-attribute has changed, then you have to call `RTable.refresh()` in order the changes getting effect on the UI. Otherwise there is no guarantee, that the ui side is up to date after the data binding.

Short example: An editable expression looks like: `"in.editable && entry.editable"`, when now only the attribute `in.editable` has changed, a call to `RTable.refresh()` is required. Otherwise the change on `in.editable` has no effect on the ui.

Tab Cell Widget

Overview

The Cell Widget tab shows the configuration data mapping between the combo box and the cell.

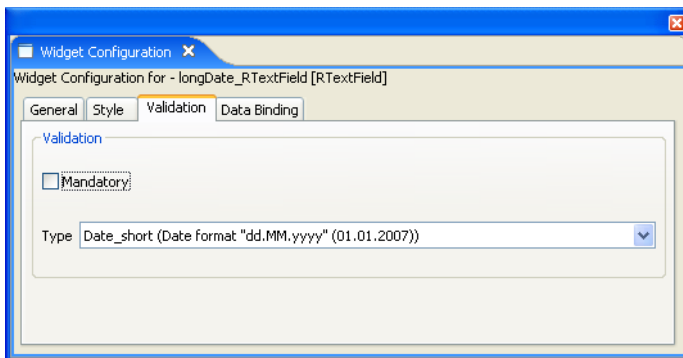


Figure 6.11. The Cell Widget tab

Accessibility

The tab is only visible if a `RComboBoxCellWidget` is selected.

If it is not visible, go to: Window -> show View -> Widget Configuration and select the tab Cell Widget in the opened view "Widget Configuration".

Features

Cell Binding Specify the data binding between the combo box and the table cell value, if needed. Per default (empty) the whole selected entry is mapped to the table field.

Field: Specify the field (or a sub field/value) of the combo box entry, which is used for the data binding between the combo box and the table cell. The returned value from the defined field will be set to the cell value on user selection. Also backwards, the cell values from the table are compared with each defined field value. The found entry will be displayed in the combo box. Note: The mapping values are cached after the first usage.

Tab Tree Data

Overview

On the Tree Data tab you configure what content to show in the Tree widget. You can configure the tree, based on the type or information of the tree node by adding some filters.

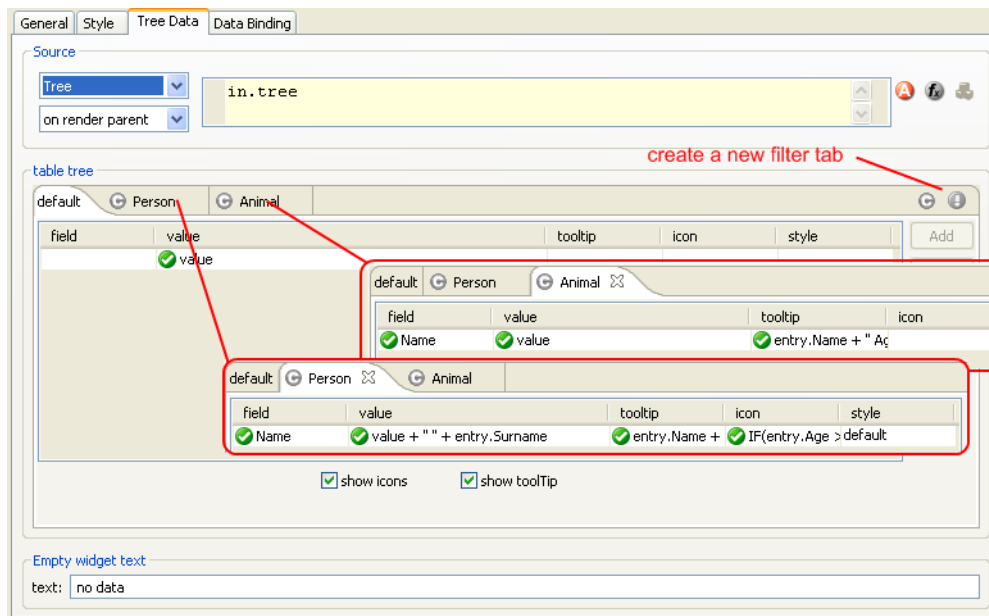


Figure 6.12. The Tree Data tab

Accessibility

Window -> show View -> Widget Configuration -> Tree Data

Features

Source The type of the source must be `Tree`. Define in the IvyScript text field the path to your source. The second combo box lets you configure the rendering behaviour of the tree. If you select **on render parent** or **on expand parent** then the tree is lazy loaded and the tree widget fires `TreeLoad` events to let the user populate the tree. The difference between the two modes is the moment when the tree widget fires the `TreeLoad` event. If you set the mode to **on render parent** then the tree widget fires the event as soon as the parent gets visible on the user interface. If you select **on expand parent** then the event fires only if the user really expand the parent node. If you configure the last mode (**on expand parent**) then the '+' sign is always rendered for the parent node unless you set the `mayHaveChildren` flag of the tree node to false.

Table Tree **Columns** **Field:** Define the name of the field of your data source to map to the current column. This is only allowed on a object-type based filter tab. The name entered in the field cell must map to a property of the value of the Tree node. The value of the field is mapped to the variable `value` which you can use in the Ivy Script text field.

Value: Enter a IvyScript expression to configure what label should be displayed in the tree row.

Tool tip: Enter a IvyScript expression to configure what tool tip text should be displayed in the tree row.

Icon: Enter a IvyScript expression to configure which icon should be displayed in the tree row.

Style: Select a table cell style to use as style for the tree row if needed. To create new table cell styles use the configuration editor.

Actions **Show Icons:** Activate this check box if you want to decorate your tree with icons. The check box is only visible on the **default** filter tab.

Show Tool Tip: activate this check box if you want your tree to support tool tips per row. The check box is only visible on the **default** filter tab.

Empty Widget Text: Enter a plain text that will be displayed if the data source is empty.

Tab TableTree Data

Overview

On the TableTree Data tab you configure what content to show in the TableTree widget. You can configure the TableTree, based on the type or information of the tree node by adding some filters.

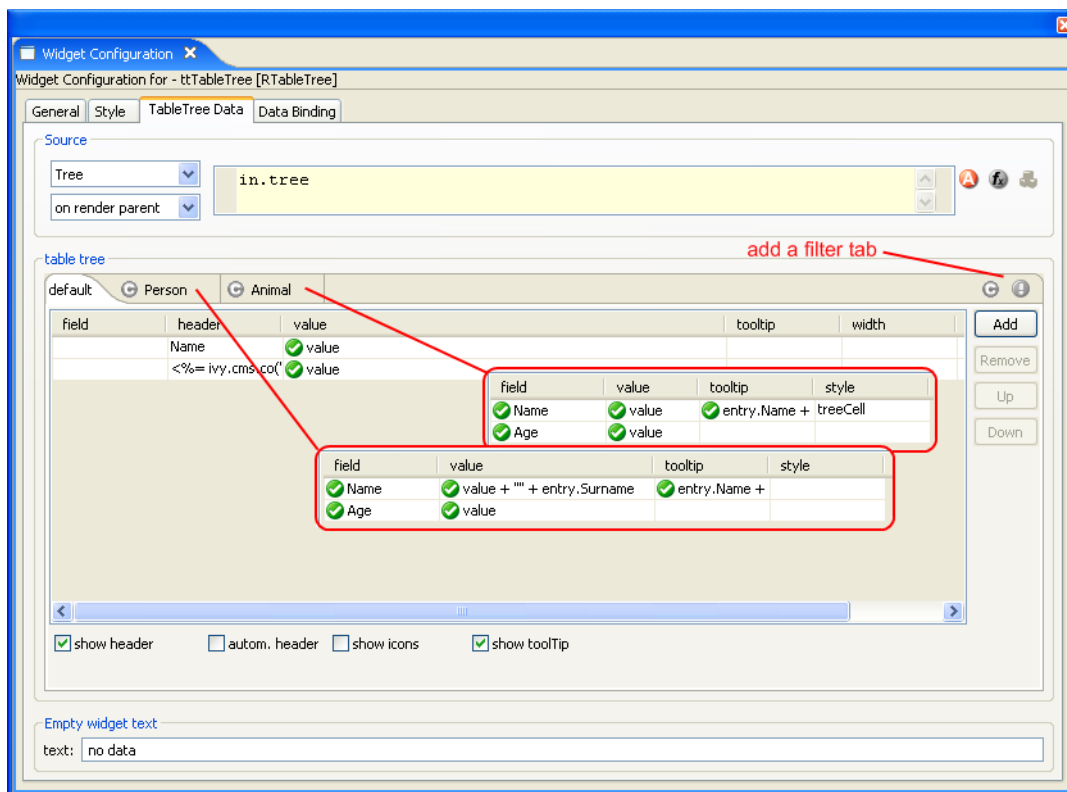


Figure 6.13. The Table Data tab

Accessibility

Window -> show View -> Widget Configuration -> TableTree Data

Features

Source The type of the source must be **Tree**. Define in the IvyScript text field the path to your source. The second combo box lets you configure the rendering behaviour of the tree. If you select **on render parent** or **on expand parent** then the tree is lazy loaded and the tree widget fires **TreeLoad** events to let the user populate the tree. The difference between the two modes is the moment when the TableTree widget fires the **TreeLoad** event. If you set the mode to **on render parent** then

the TableTree widget fires the event as soon as the parent gets visible on the user interface. If you select **on expand parent** then the event fires only if the user really expand the parent node. If you configure the last mode (**on expand parent**) then the '+' sign is always rendered for the parent node unless you set the `mayHaveChildren` flag of the tree node to false.

Table column:	<p>Columns</p> <p>Field: Define the name of the field of your data source to map to the current column. This is only allowed on a object-type based filter tab. The name must map to a property of the data source. The value of the field is mapped to the variable <code>value</code> which you can use in the IvyScript text field.</p> <p>Header: Enter a text here to be displayed as header of the column. Use IvyScript macros if needed. example: <code><%=ivy.cms.co("/labels/header")%></code></p> <p>Value: Enter an IvyScript expression to configure what label should be displayed in the tree row.</p> <p>Tool Tip: Enter an IvyScript expression to configure what tool tip text should be displayed in the tree row.</p> <p>Icon: Enter an IvyScript expression to configure which icon should be displayed in the tree row.</p> <p>Width: Enter the number of pixels the column should have in width. Only works if the configured <code>autoResizeMode</code> is set to <code>AUTO_RESIZE_OFF</code></p> <p>Style: Select a table cell style to use as style for the TableTree cell if required. To create new table cell styles use the configuration editor.</p> <p>Actions</p> <p>Add: Adds a new column configuration to your TableTree definition. The column configuration will be shown as row in the TableTree columns configuration.</p> <p>Remove: Removes the selected column configuration from your TableTree columns configuration.</p> <p>Up: Moves a TableTree columns configuration up in the table. This means that the column moves one position to the left in the resulting TableTree.</p> <p>Down: Moves a TableTree columns configuration down in the table. This means that the column moves one position to the right in the resulting TableTree.</p> <p>Show Header: Tick this check box if you want your TableTree to show a header row. The check box is only visible on the default filter tab.</p> <p>Autom. Header: Tick this check box if you want the system to label your headers. The check box is only visible on the default filter tab.</p> <p>Show Icons: Tick this check box if you want to decorate your TableTree with icons. The check box is only visible on the default filter tab.</p> <p>Show Tool Tip: Tick this check box if you want your TableTree to support tool tips per row. The check box is only visible on the default filter tab.</p>
Empty widget Text	Enter a plain text that will be displayed if the data source is empty.

Tab Validation

Overview

The Validation tab shows the configuration for the client side validation for widgets the user can input data into.

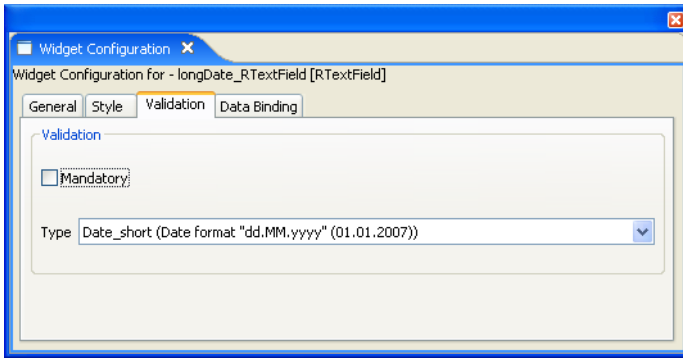


Figure 6.14. The Validation tab

Accessibility

Window -> show View -> Widget Configuration -> Validation

Features

Mandatory Activate this check box if the user must enter a value in the text field.

Type Select the type of validation that should be applied to the value entered by the user. See the Configuration Editor for information about how to create your own input validation configurations.

IvyScript Editor Widget

Overview

The IvyScript editor widget allows you to enter IvyScript expressions, IvyScript snippets or IvyScript macros combined with plain text. The background color of the IvyScript Widget signals what syntax is allowed. Is the background color yellow, then you have to enter pure IvyScript. Depending on the context this IvyScript Widget is embedded, you have to enter a expression or a whole IvyScript snippet.

If the background color of the IvyScript Widget is blue, then only the IvyScript macro syntax is accepted. Enter plain text combined with IvyScript macros.

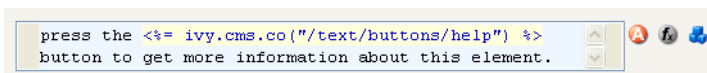



Figure 6.15. The IvyScript Editor Widget



Figure 6.16. The IvyScript Editor Widget as cell editor in a table

Features

- *Code completion (Ctrl+Space)*: By pressing the '!' character or by pressing **Ctrl+Space** the code completer shows you some code completion proposals depending on the already entered expression.
- *Quick Assist (Ctrl+I)*: There are Quick Assists available to create or use content objects in the current Rich Dialog. Simply press **Ctrl+I** on a text. A newly created content object can directly be renamed in the IvyScript Editor. Just type a new name and confirm with **Enter**, to abort the renaming press **Esc**.

- **Zoom:** By pressing the **F2** key, a new dialog opens with the zoomed IvyScript editor.
- **Link to CMS (F3 or Ctrl):** Press the **F3** key on a `ivy.cms.co(...)` expression to jump to the corresponding CMS content object. Alternatively press **Ctrl** and click on the expression.
- **Smart Buttons:**  Use the smart buttons to insert predefined IvyScript expressions or IvyScript macros into your editor.

Ivy Outline View

Overview

The Axon.ivy Outline view shows outline information of the current Rich Dialog and offers some actions to configure it. Furthermore, the Outline view helps you to compose Rich Dialogs by supporting Drag and Drop of elements from the outline tree into the Rich Dialog editor to create data bindings and event mappings.

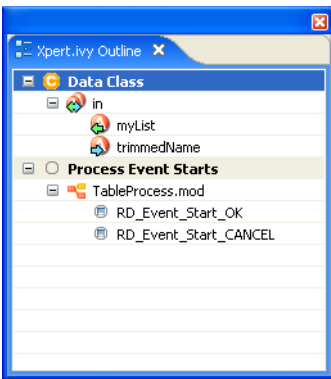


Figure 6.17. The Ivy Outline View

Accessibility

Window -> Show View -> Axon.ivy Outline

Features

Shows a tree containing two root elements:	Data Class Fields	Data Class fields are decorated with data binding arrows. A green left arrow indicates a up binding (data to UI), a blue right arrow indicates a down binding (UI to data) and a left-right arrow indicates a bidirectional binding. The decoration should also work if the binding is complex (includes #, line breaks, brackets,...).
	Event Process Starts	The Event Process Starts are decorated with blue right arrows if they are mapped to a UI element.
Refresh	The outline tree should refresh itself whenever a resource changes. (Data Class, Rich Dialog, Event Process Starts)	
DnD	Drag tree items onto the visual editor to add mappings and bindings to your rich dialog. See DnD for more information.	
Context menu - Create Rich Dialog Widgets	Creates a label and a data widget in the currently opened Rich Dialog for the selected data class field. A Dialog pops up asking you to specify the details of the widgets you want to create.	

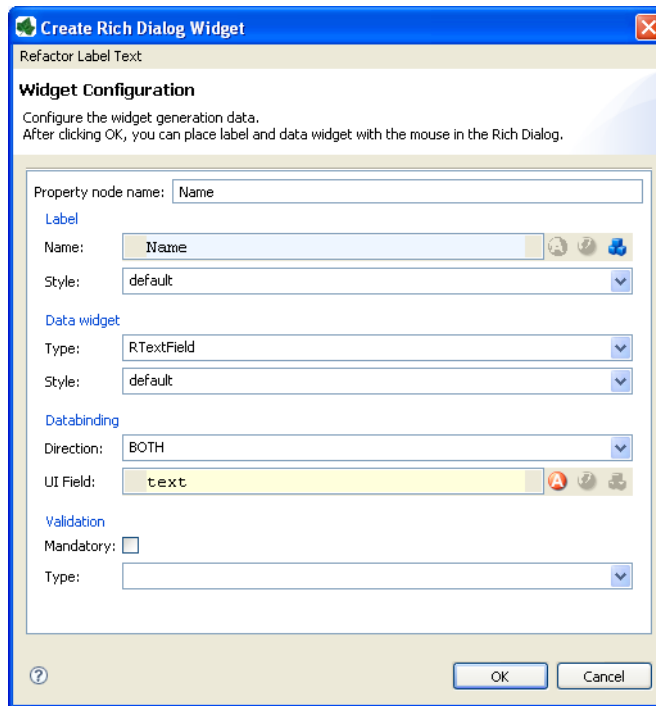


Figure 6.18. Create Rich Dialog Widgets dialog

Menu "Refactor Label Text"

Create CMS node: Creates a CMS node for the text of the label. The name of the CMS reference is configured by the string that is currently defined as "Label Name".

Guess from CMS (CONTENT): Looks up in the CMS if it can find a node with the content string that is equal to the one set as "Label Name".

Guess from CMS (URI): Looks up in the CMS if it can find a node whose URI contains the string that is equal to the one set as "Label Name".

Dialog

The details of the pair are shown in the dialog. Not all details are configurable for all type of "Label Widget - Data Widget" pairs. If you select more than one element in the tree, then only all the common configuration details are enabled, the varying options cannot be manipulated.

Property Node Name: Prefix for the name of this "Label Widget - Data Widget" pair in the panel data class, e.g. the label for the pair eRMMessagenumber will be referenced by eRMMessagenumberRLabel.

Label Name: The string that will be shown in the GUI as label text.

Label Style: The style of the label of the "Label Widget - Data Widget" pair

Data Widget Type: The type of the data widget e.g. `RTextField`, `RComboBox`, ...

Data Widget Style: The style of the data widget used for the data widget chosen above.

Data Binding Direction: The direction of the data binding

Data Binding UI Field: The field on the Rich Dialog side of the data binding. e.g. `text` or `toolTipText` for a "Label Widget - Data Widget" pair that has a `RTextField` as its type of the data widget.

Validation (only for `RTextField`) Mandatory: If the user must enter a value in this field

Validation (only for `RTextField`) Type: The name of the validation configuration reference.

At last, you have to place the widgets into your Rich Dialog using the Rich Dialog editor. First you have to place the label and secondly you have to place the data widget, e.g. the `RTextField`. Press the **ESC** key to cancel the action.

Data Binding View

Overview

The Data Binding view shows which process data fields are bound to which user interface widget properties. See Data Binding section for a more detailed explanation of the concepts of data binding.

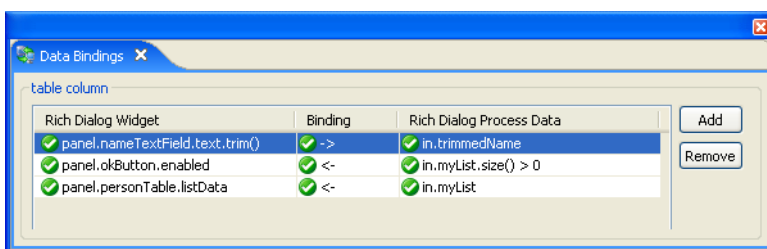


Figure 6.19. The Data Binding View



Note

The -> (down) data bindings will be executed at every Rich Dialog Start element (e.g. *Rich Dialog Event* start or *Rich Dialog Method* start) except the *Rich Dialog Start method* element. The <- (up) data bindings will be executed at every Rich Dialog Process End element.

Accessibility

Window -> show View -> Data Bindings

Features

Columns: **Rich Dialog Widget:** Enter the path to the rich dialog widget's property that you want to use for your binding.

Direction:

-> A down data binding. Maps the data from the user interface widget property to your process data property.

<- An up data binding. Maps your process data property to the user interface widget property.

<-> A bi-directional data binding. Maps in both directions, from the widget property to your process data at Rich Dialog process starts and back to the user interface at the end of them.

Rich Dialog Process Data: Enter the path to your Data Class attribute that you want to bind to the widget's property. Use IvyScript functions to manipulate your data before the binding. If you do so, then you are only allowed to bind the data in one direction (from process data to ui).

```
panel.myWidget.text      "from/to "      in.name
panel.myWidget.text.trim() "to (ui to data)" in.name
panel.myWidget.text      "from (data to ui)" in.name.trim()
```

Refresh The table should refresh itself whenever a resource changes.

Cell Editing Use the **F2** Key to enlarge the cell editor.

Use the Smart buttons, to get a faster access to your properties.

Use the auto completer to find your properties easier.

New Event Mapping Wizard

Overview

The New Event Mapping wizard lets you map a user interface events (e.g. a `RButton`) to Rich Dialog events.

With the wizard you can create several types of mappings:

GUI Events	GUI (graphical user interface) events are triggered by the user, e.g. a mouse click leads to an <code>Action</code> event on a <code>RButton</code> .
Display Events	On the panel you have access to the <code>LOAD</code> and <code>UNLOAD</code> event of the Rich Dialog panel.
Embedded (inner) Rich Dialog Events	If you embedded a inner Rich Dialog onto your current Rich Dialog then the <code>Fired Events</code> of the embedded Rich Dialog are displayed to map to a process start of your current (outer) Rich Dialog.

Make your choice from left to right. Start with the selection of the source widget from the tree widget on the left. The configuration propositions on the right change when the selection changes on the left.

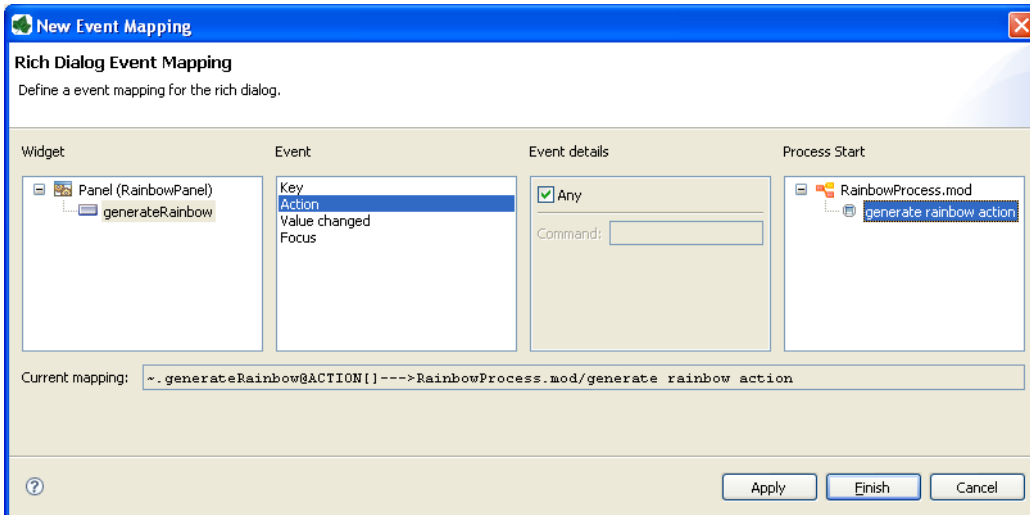
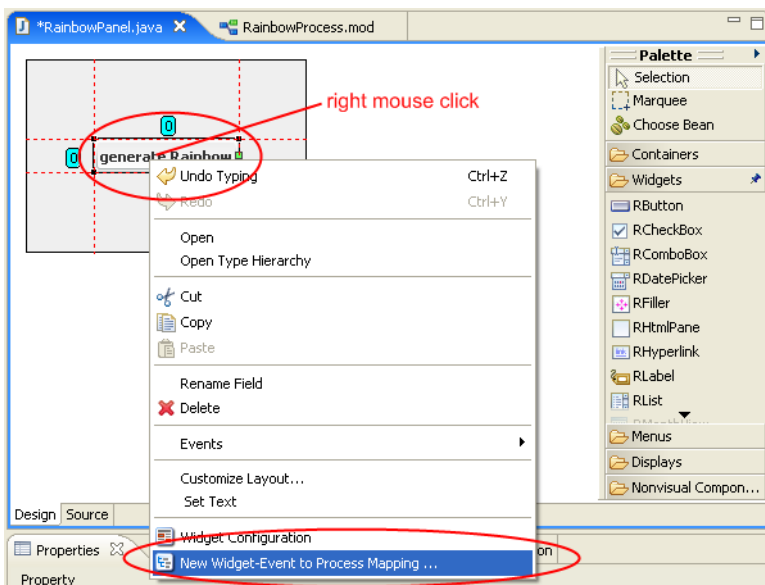


Figure 6.20. The New Event Mapping Wizard

Accessibility

Rich Dialog editor -> context menu of a widget (e.g. a RButton) -> New Widget-Event to Process Mapping



Features

- Widget Choose the source widget from whose events you want to map one.
- Event Choose the event to map.
- Event details Depending on the chosen widget and event, you have here the opportunity to configure the details of the event.
- Process Start Finally select the Rich Dialog process start that should be triggered after the chosen widget event was fired.



Note

If the tree is empty, then you have to define some Rich Dialog Event starts in your Rich Dialog logic first.



Tip

It is also possible to define the event mappings by Drag and Drop. Open the Ivy Outline View to use it.

Event Mapping View

Overview

The Axon.ivy Event Mapping view shows which widget events are mapped to which Rich Dialog events.

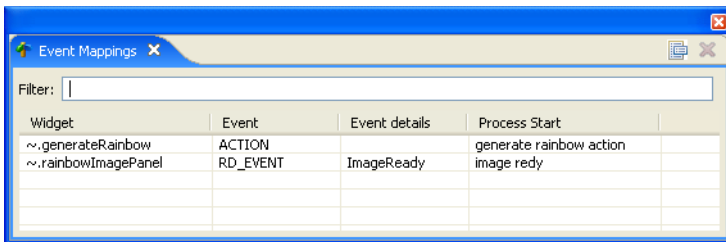


Figure 6.21. The Event Mapping View

Accessibility

Window -> show View -> Event Mapping

Features

Focus	Lists only the event mappings of the selected widget in the Rich Dialog editor.
Delete	Removes the selected event mapping.
Filter	Enter a string to filter the listed event mappings.
Widget	Shows the source widget of the event that is mapped.
Event	Shows the mapped event of the chosen widget.
Event details	Shows the details of the mapped event if any.
Process Start	Shows the Rich Dialog event that should be triggered after the widget event has been fired.
Edit an Event Mapping	To edit a event mapping, just double click on the list entry. The Event Mapping Wizard will open. Make sure that the Rich Dialog editor is the active editor and a widget is selected.

Drag and Drop in Axon.ivy Designer

Overview

The Axon.ivy Designer offers you to use Drag and Drop while creating your Rich Dialog panel. Drag and Drop can be used to:

- map Rich Dialog Event starts to your Rich Dialog widget events
- create data bindings from Data Class attributes to your Rich Dialog widgets
- associate CMS objects to widget properties

As Drag source you need the Ivy Outline view or the CMS view. As Drop target you need to point to a widget in an open Rich Dialog editor.

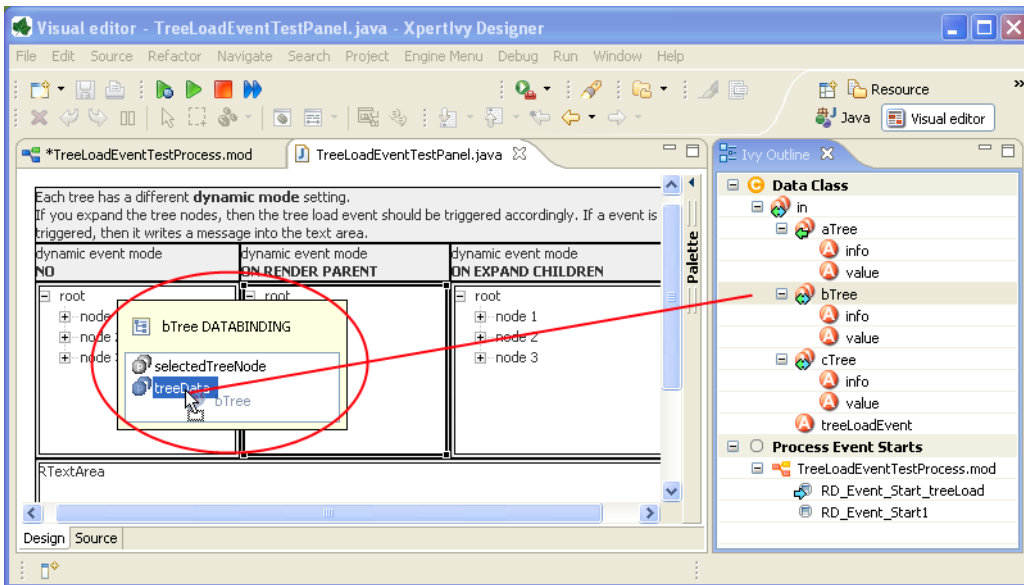


Figure 6.22. The DnD Axon.ivy Designer support

If you drag an element (e.g. a Data Class field `in.myName`) onto the Rich Dialog editor over a label widget, then after some milliseconds a popup window appears and shows the possible mapping destinations. If you wait longer, more mapping proposals will appear. Drop your `in.myName` over the proposal text in the popup and the data binding is created.

If you do not wait until the popup appears, and you drop the `in.myName` over the label widget, then a predefined proposal will be used to create the data binding. The generation of the proposals in the popup is configurable as well as the auto-bind if you do not wait until the popup appears. But this configuration is out of scope of this document.

Accessibility

Ivy Outline view -(drag to)-> Rich Dialog editor

CMS view / CMS editor -(drag to)-> Rich Dialog editor or a property in the Widget Configuration View

Rich Dialog User Context

The *Rich Dialog User Context* is a concept to store data for a Rich Dialog for the user currently logged in. E.g. it is possible to store the last search string of a search dialog inside the context to show this string next time the user opens the panel again. Therefore the Rich Dialog User Context only works when a user is logged-in. When no user is logged-in, the context is empty and each interaction with the context has no effect.

Following Public API allows you to set and get data from the Rich Dialog User Context:

```
ivy.rd.context.setAttribute(String key, String value):  
Associates the specified value with the specified key in the current Rich Dialog User  
Context.
```

```
ivy.rd.context.getAttribute(String key) : String: Returns the  
value of the specified key from the current Rich Dialog User Context.
```

Axon.ivy itself uses the Rich Dialog User Context to store the UI-State of certain widgets.



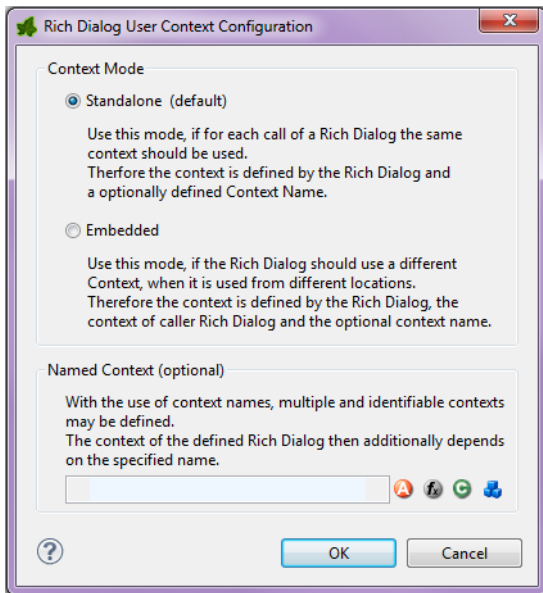
Restrictions

Do NOT use the Rich Dialog User Context for business relevant data. It is not guaranteed, that the data can be restored in each situation. When the structure of the a Rich Dialog changes (e.g. the field name of a component inside the Rich Dialog) to lookup of the context can fail, without any message.

Basically the context is designed for non business-critical information like UI-State.

Configuration

The configuration of the Rich Dialog User Context will be done in the Rich Dialog Call element inside a Process and in the Rich Dialog Start element inside a Rich Dialog logic.



Context Mode

The context mode allows to configure two modes, Standalone and Embedded:

Standalone

Use this mode, if for each call of a Rich Dialog the same context should be used. Therefore the context is defined by the Rich Dialog (e.g. ch.foo.bar.MyRichDialog) and a optionally defined Context Name (e.g. MyContext).

If this configuration is made for a Rich Dialog Call element the configured Rich Dialog is used. If this configuration is made for a Rich Dialog Start element, then the embedded Rich Dialog is used.

The context identification is: `User, Rich Dialog` and optional `Context-Name`

Embedded

Use this mode, if the Rich Dialog should use a different Context, when it is used from different locations. Therefore the context is defined by the Rich Dialog, the context of parent Rich Dialog and the optional context name.

If this definition is made for a Rich Dialog Start element, the parent Rich Dialog of the embedded Rich Dialog is used as parent Rich Dialog. If this configuration is made for a Rich Dialog Call element, the Rich Dialog which triggers the call of this element is used as parent Rich Dialog.

The context identification is: `User, Rich Dialog, parent Rich Dialog` and optional `Context Name`

Context Name

With the use of context names, multiple and identifiable contexts may be defined. The context of the Rich Dialog then additionally depends on the specified name. This configuration can be done for both modes, the context name is simply added to the context identification.

Default Configuration

The Rich Dialog Call element is by default configured as `Standalone`. Therefore by default such a Rich Dialog will always have the same Context.

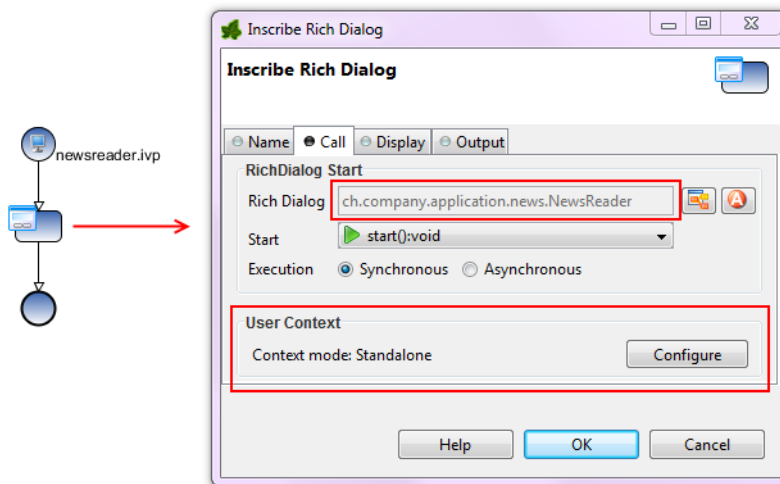
For embedded Rich Dialogs inside the Rich Dialog Start element the default context-mode is configured as `Embedded`. Therefore all embedded Rich Dialogs are linked to the parent Rich Dialog context.

Rich Dialog User Context Examples

This section describes how to configure the context for a small application, like a News Reader.

Standalone Configuration

Each time the Rich Dialog `NewsReader` is called, the same context should be used. So the user will get the same context each time when he opens the Rich Dialog `NewsReader`.

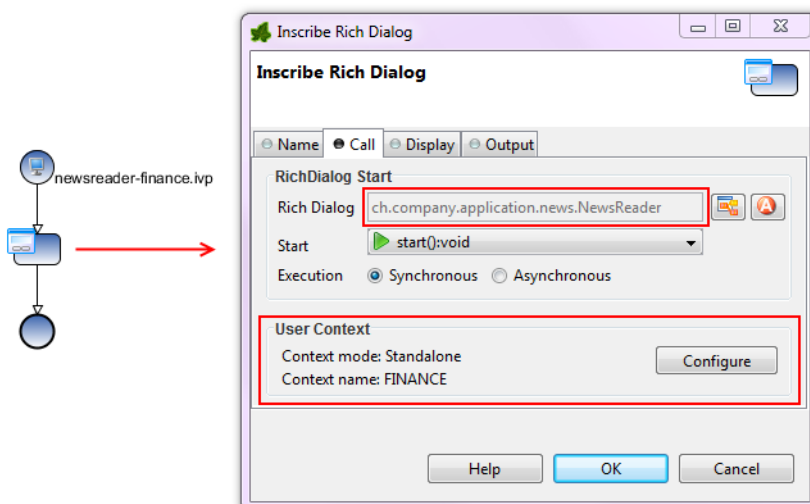


Technically one Rich Dialog, but functional multiple different usages

The Rich Dialog `NewsReader` is used e.g. for financial and local news. The context should be different. The user likes to have one context for financial news and one for local news.

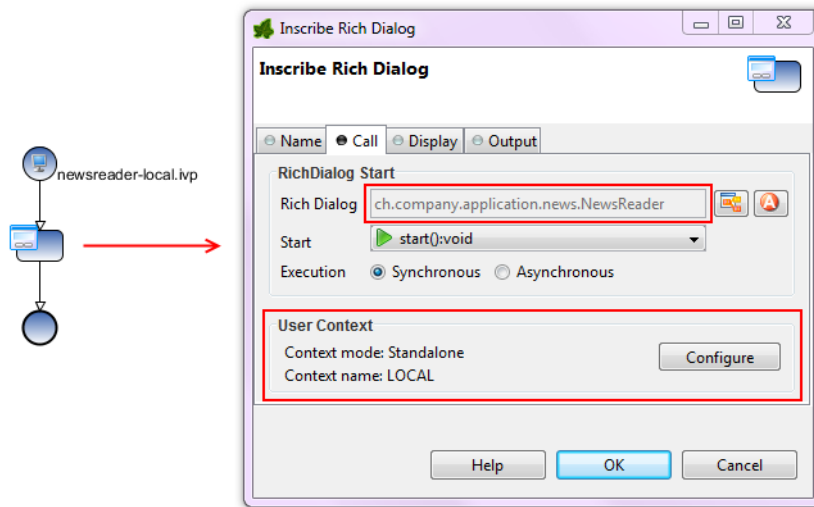
Configuration for Financial News:

Each time the Rich Dialog `NewsReader` is opened with the context-name `FINANCE`, this specific Rich Dialog User Context will be used. There is no difference from which location (Rich Dialog Call element) the Rich Dialog is opened.



Configuration for local News:

Each time the Rich Dialog `NewsReader` is opened with the context-name `LOCAL`, this specific Rich Dialog User Context will be used. There is no difference from which location (Rich Dialog Call element) the Rich Dialog is opened.

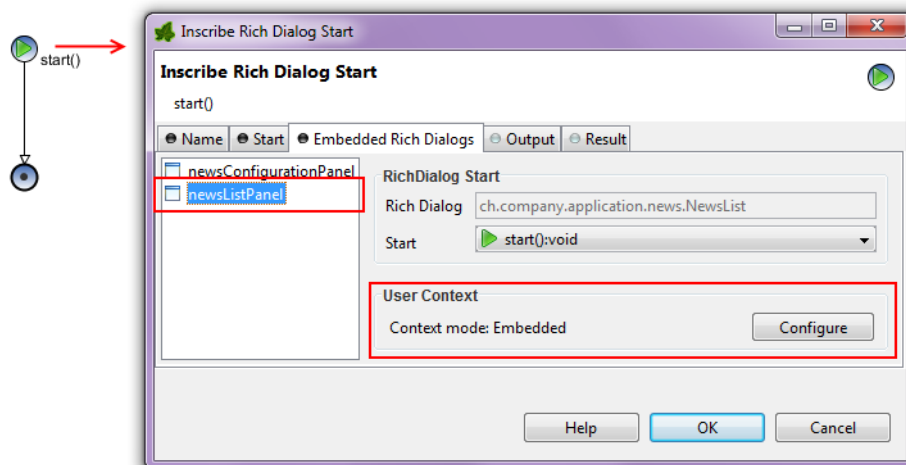


Embedded Rich Dialog

The Rich Dialog `NewsReader` contains two embedded Rich Dialogs, the `NewsList` and the `NewsConfiguration`. The embedded Rich Dialog `NewsList` has configured the context mode `Embedded`, because the context is depending on its parent Rich Dialog Context, the `NewsReader` context. Therefore separate contexts will be used for financial and local news, as defined in the `NewsReader` Rich Dialog. The embedded Rich Dialog `NewsConfiguration` contains the global configuration of all news readers in the application. Therefore the context mode is `Standalone`. It should always use the same context without dependency to the context of the `NewsReader`.

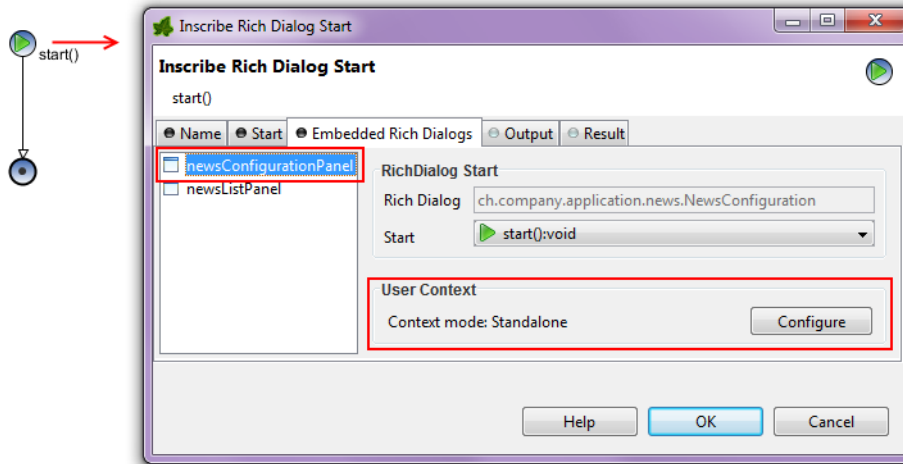
Configuration of NewsList

The context of the Rich Dialog `NewsList` is linked to the parent Rich Dialog `NewsReader`. Therefore, related to the configuration of the parent Rich Dialog two contexts will be created. One for finance news and one for local news.



Configuration of NewsConfiguration

The context configuration of the Rich Dialog `NewsConfiguration` does not depend on the parent Rich Dialog. Therefore only one context will be created. It will always use the same context regardless which context configuration `NewsReader` does use.



Rich Dialog UI State

The *Rich Dialog UI State* stores and restores a part of the UI state. The state is stored automatically when a window closes. At startup of a window, the last known state is restored automatically too.

The data is stored on the Rich Dialog User Context. Therefore, when no user is logged-in, this functionality is disabled.

Following public API allows to force a store or restore of the UI State:

`ivy.rd.context.storeUiState()`: Stores the UI state in the context of the called Rich Dialog. Also the UI State of all embedded Rich Dialogs will be stored too.

`ivy.rd.context.restoreUiState()`: Restores the UI state in the context of the called Rich Dialog. Also the UI State of all embedded Rich Dialogs will be restored too.

Currently the following widgets are supported:

Supported widgets: `Dialog / Window`: The size and the location of a window / dialog on a screen.

`RSplitPane`: The relative position of the divider in the `RSplitPane`.

`RTable`: The column position, width and order.

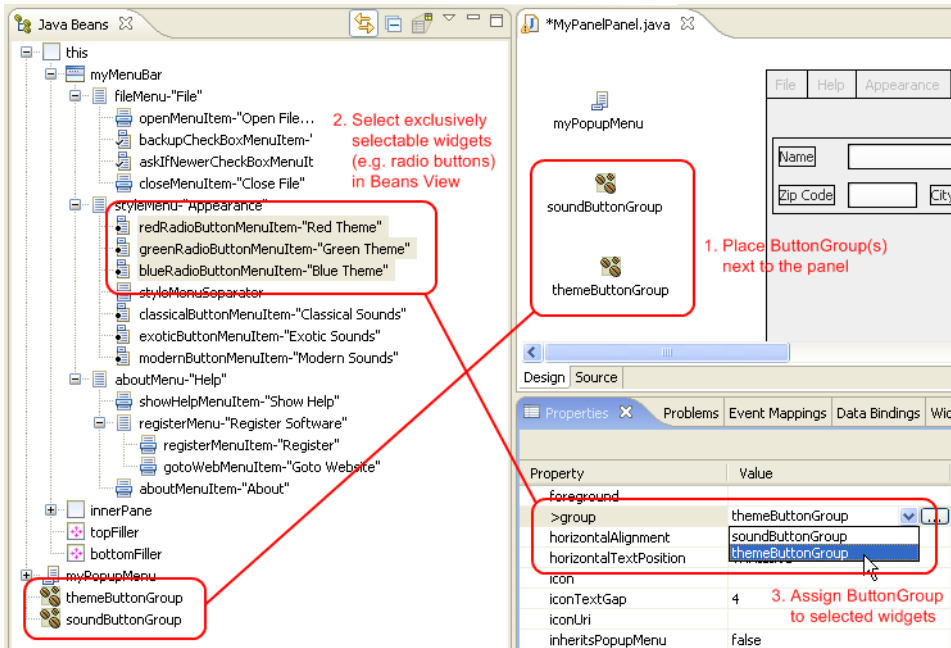
Rich Dialog Widget Reference

This chapter explains the usage and configuration of all available Rich Dialog widgets. All widgets are sorted alphabetically, regardless of their category.

ButtonGroup

Button groups are used to *logically* group exclusively selectable widgets (e.g. radio buttons or toggle buttons). In order for a button group to effectively control the select state of its guarded components it must be associated specifically with each widget that it contains.

Button groups do not have a visual representation and must therefore be placed *next to the panel* rather than on the panel itself:



To associate a button group with a widget, you first select the widget(s) inside the Visual Editor or inside the Beans View and then set the button group object to be associated on the **group** attribute in the Properties View, as demonstrated above.


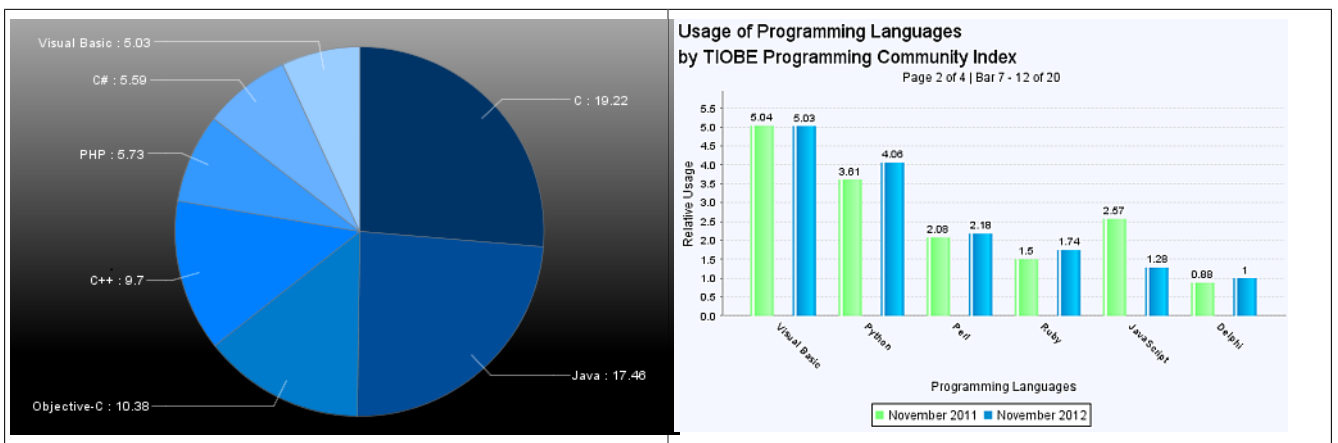

Symbol on Palette	
Category	Non Visual Widgets
Purpose	Logically group a number of selectable items and ensure that at most one is selected at a time.
Events	-
Important Properties	-

Table 6.13. ButtonGroup Summary

Chart widgets



The chart widgets can be used to display numeric data in a graphical way. Two types of charts are supported, pie charts for small one-dimensional data sets and bar charts for two-dimensional data. The charts widgets must be primarily used with a Java API, there is only limited tool support in Axon.ivy, e.g. there is no Drag and Drop of events for the event mapping or there is no widget configuration view.

To add a chart widget to a Rich Dialog use the  Choose Bean. Choose Bean entry from the palette in the Visual Editor. Then choose either the ULCVPieChart or the ULCVBarChart and place it on the Rich dialog. Alternatively you can use the

widgets by adding the corresponding code in the Java source of the Rich Dialog yourself. You can set most of the properties in the properties view of the Rich Dialog perspective. But full support for the widgets is only given in code, either in Java by enhancing the Rich Dialog source file manually or by accessing the widget with IvyScript.

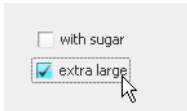
Both charts can be used in 2D or 3D, for 3D charts use the appropriate constructors `ULCVPieChart(boolean is3DChart)` and `ULCVBarChart(boolean is3DChart)` respectively. Both charts can optionally display a legend, can use titles and sub titles and both offer adding a listener to be notified when the user clicks on the pie or bar. Use `addMouseListener(IChartMouseListener<EV extends ChartMouseEvent>)` then you get either a `BarMouseEvent` or a `PieMouseEvent` that you can query for more information about the click.

Pie charts are ideal to display small one-dimensional data sets with only some values. Use `addPieSlice(PieSlice slice)` or `setPieSlices(List<PieSlice> slices)` to add data to the pie chart. In addition to the value it is possible to set a solid color or a gradient and a title (only by constructor) for each slice.

For two-dimensional data, bar charts are better suited. You can add data by using `addDataSeries(DataSeries series)` or `setDataSeries(List<DataSeries> series)`. A data series is a list of values that belong together and each value is represented as a bar in the same color/gradient. The other axis is defined by categories that group together the values with the same index from different data series. In other words, the values of the data series represent the y-values where as the categories are the x-values. For data series with lots of values, the bar chart widget supports paging. Switch it on with `setPagingEnabled(true)`. It's even possible to configure the number of values per page and whether the coordinate system should be the same for all pages or whether it should be scaled.

Both widgets support the most important visual settings for charts such as background colors/gradients and transparency, shadows, tool tips and setting the fonts and font sizes of the texts used in the charts (values, axis labels, titles).

RBooleanCellWidget



The `RBooleanCellWidget` is like a check box for table editing. Check boxes are used to let the user select an option.

The mapped table cell value should be a boolean value, like `true` or `false`.

The check box will generate both an *Action Event* and a *Value Changed* event if it is selected or deselected.

You can provide your own *selected* and *unselected* icons (if you want to replace the standard check mark).

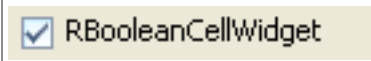
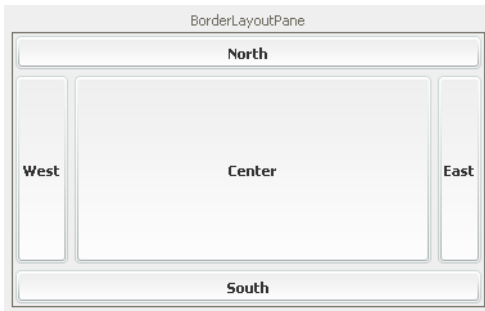
Symbol on Palette	
Category	Action / Selection widget
Purpose	A check box to display and change a boolean cell value.
Events	Action, Focus, Key
Important Properties	<code>text</code> , <code>icon</code> , <code>pressedIcon</code> , <code>rolloverIcon</code> , <code>selectedIcon</code> , <code>toolTip</code>

Table 6.14. RButtonCellWidget Summary

RBorderLayoutPane



A border layout pane is a container which arranges its components using a Border Layout. Essentially, a border layout consists of 5 areas: *North*, *East*, *South* and *West* which are arranged around the *Center*. The component in the center always consumes as much of the available space as possible. The surrounding components are stretched accordingly.

You can specify the amount of vertical and horizontal spacing in between the 5 areas by setting the `vGap` and `hGap` properties, respectively.



Tip

Not all areas of a border layout have to have a component assigned. If an area stays empty, it will not consume any space (except for the center, which will also consume all available space if it does not have a component assigned). It is common practice to use only the center in combination with one or two of the remaining sections.

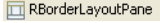
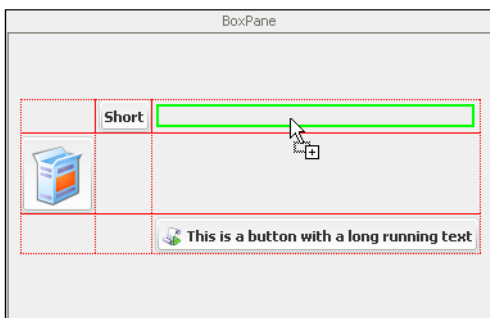
Symbol on Palette	
Category	Container widget
Purpose	A container which uses a <i>Border Layout</i> to arrange its components.
Events	Focus, Key
Important Properties	<code>hGap</code> , <code>vGap</code>

Table 6.15. RBorderLayoutPane Summary

RBoxPane



A box pane is a container which arranges its components using a Box Layout. Essentially, working with a Box Layout is similar to layouting with *HTML tables*. A box layout consists of n rows and m columns, where the height of each row and the width of each column are determined by the tallest or widest component, respectively. If a widget does not fill the available cell space, then it may be stretched and/or positioned inside the cell by defining its *widget style properties* accordingly.

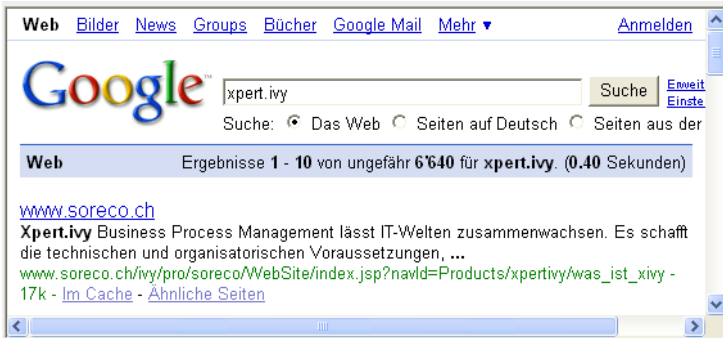
You can specify the amount of vertical and horizontal spacing that is used between the rows and columns by setting the `verticalGap` and `horizontalGap` properties, respectively.

Symbol on Palette	
-------------------	---

Category	Container widget
Purpose	A container which uses a <i>Box Layout</i> to arrange its components.
Events	Focus, Key
Important Properties	horizontalGap, verticalGap

Table 6.16. RBoxPane Summary

RBrowser



The browser widget can be used to show an embedded web browser.



Note

The web browser widget has no preview at design time in the Visual Editor. Instead the widget is rendered as a label of the configured size, which shows a *Embedded Web Browser : <no preview>* message.

The browser widget will generate *Web Browser - COMPLETED* event, every time a new web page is loaded.



Tip

You can also use the web browser to display HTML pages or other content that you prepared yourself. The content to display by the native browser can be set as a String on the **content** attribute.

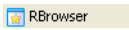

Symbol on Palette	
Category	Information / Input widget
Purpose	An embedded native browser which can be used to display web pages.
Events	Web Browser, Focus, Key
Important Properties	Url, Content

Table 6.17. RBrowser Summary

RButton



Buttons are typically used to start an action or to generate an event. The button generates an *Action Event* if it is clicked.

Symbol on Palette	
-------------------	---

Category	Action widget
Purpose	A button with text and/or icon.
Events	Action, Focus, Key
Important Properties	text, icon, pressedIcon, rolloverIcon, selectedIcon, tooltip

Table 6.18. RButton Summary

RButtons in Frames

The `action` event of a `RButton` will be fired as soon as the user clicks the mouse on it. It will also be fired, if the `RButton` owns the focus and the user hits `ENTER` or `SPACE`.

If the focus owner does not support the `action` event, then the **default button** of the frame will be fired.



Note

To set a `RButton` as the default button of a Rich Dialog you have to map the `DISPLAY load` event of the panel and implement the following code snippet on the code tab of the *event start process* element:

```
panel.getRootPane().setDefaultButton(panel.okButton);
```

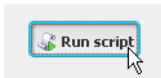
Badge mode for RButton

The `RButton` can be rendered like a smart phone like badge, a clickable icon paired with an overlay that displays a number. Such badges are typically used to start dialogs, tools or processes that work on a list of entities, e.g. a badge could be used to start a mail inbox dialog using the overlay to display the number of new unread messages.



Set the `setShowBadge()` method to change between normal button and badge mode and use the `setBadgeValue()` method to set the overlay content.


RButtonCellWidget



Buttons are typically used to start an action or to generate an event. The button generates an *Action Event* if it is clicked. The Button text, icon and tooltip are taken from the cell value.

To process the action event of the button, map the event to a Rich Dialog Process start in the Rich Dialogs panel logic. In the code tab of the Event Start Process step enter the following code:

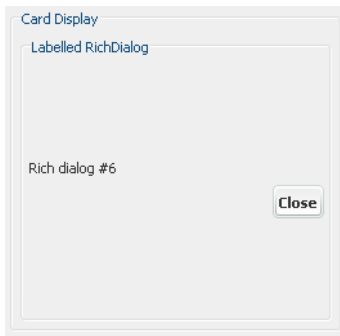
```
import ch.ivyteam.ivy.richdialog.widgets.components.customrenderers.RButtonCellWidget;
RButtonCellWidget button = event.getSource() as RButtonCellWidget;
int row = button.getRow();
int column = button.getColumn();
```

Symbol on Palette	 RButtonCellWidget
Category	Action widget

Purpose	A Button with a text to start an action. The cell value are displayed as text.
Events	Action, Focus, Key
Important Properties	text, icon, pressedIcon, rolloverIcon, selectedIcon, tooltip

Table 6.19. RButtonCellWidget Summary

RCardDisplay



Displays are placeholder widgets onto which Rich Dialogs can be loaded dynamically at runtime. A *Card Display* works like a "stack of cards": only the last loaded Rich Dialog on top is visible. The previously loaded Rich Dialogs will become visible again if the top component is unloaded (i.e. removed).

A card display generates a *Selection Changed Event* if a new Rich Dialog is loaded onto it.



Warning

In order to being able to address a display widget (when loading a Rich Dialog with a RD Process Element) you must not forget to assign the display an *unique* id on the **displayId** property. If you fail to do so then the display might not be found at runtime and Rich Dialogs may not be loaded correctly.


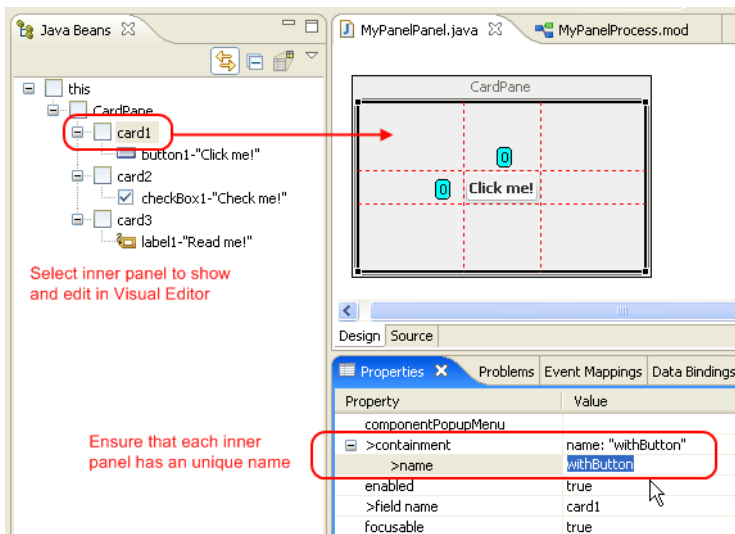
Symbol on Palette	
Category	Display widget
Purpose	Placeholder for Rich Dialogs that are loaded dynamically at runtime. Shows only last loaded component.
Events	Selection Changed, Focus, Key
Important Properties	displayId, tooltip

Table 6.20. RCardDisplay Summary

RCardPane



A card pane is a container which arranges its components using a Card Layout. The components or containers that are managed with a Card Layout are handled like a "stack of cards" where only the top component is visible. In order to identify the contained elements (i.e. cards) must be provided with a *unique* name. By setting a name to the `selectedName` property of a card pane the currently shown (i.e. top) component can be changed.

Any kind of widget or container can be a "card" on a card pane. Typically cards are containers, i.e. sub-panels, with an individual layout.



Tip

You can use the following example code to cycle through the cards of a card pane at runtime:

```
List<String> names = panel.CardPane.getNames();
panel.myCardPane.selectedName = names.get(in.cardIndex);
in.cardIndex = (in.cardIndex + 1) % names.size();
```

Place a button *next* to your card pane container on the main panel and associate it's action with a process that executes the above code. Each time when you click the button, the currently shown card will be changed.




Tip

When you work with a card pane in the Visual Editor then the editor area will only show the currently selected card. To change this (i.e. make another card visible) simply select one of the sub components of the card pane the Java Beans view.



Warning

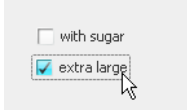
It is absolutely crucial that each added card has a unique name assigned. Otherwise selection of the cards will not work correctly.

Symbol on Palette	
Category	Container widget
Purpose	A container which uses a <i>Card Layout</i> to arrange its components (only component is visible).
Events	Focus, Key

Important Properties	selectedName
----------------------	--------------

Table 6.21. RCardPane Summary

RCheckBox



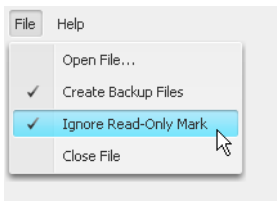
Check boxes are used to let the user select an option. The check box will generate both an *Action Event* and a *Value Changed* event if it is selected or deselected.

You can provide your own *selected* and *unselected* icons (if you want to replace the standard check mark).

Symbol on Palette	
Category	Action / Selection widget
Purpose	A check box with text that allows to select/deselect an option.
Events	Action, Focus, Key, Value Changed
Important Properties	text, tooltip, selected

Table 6.22. RCheckBox Summary

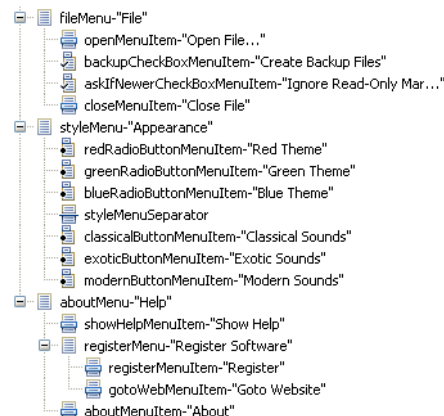
RCheckBoxMenuItem



Check box menu items are menu entries that represent on/off options. They can be used both inside RMenu and RPopupMenu widgets. The standard check-box icon of a check box menu item can be replaced by setting a different *Icon Uri* and *Selected Icon Uri* in the Widget Configuration Text/Icon Tab. Generally, check box menu items behave exactly the same as a RCheckBox widgets.

Check box menu items generate both an *Action* event and a *Value Changed* event if they are selected or deselected.

Since check box menu items do not have a direct visual representation they should be placed directly on or below a menu widget inside the Java Beans View. This view allows you to comfortably set up menu hierarchies using sub menus and menu items:



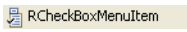
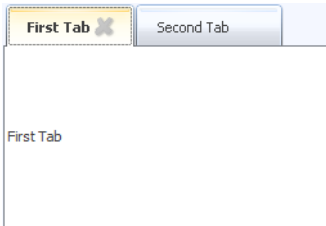
Symbol on Palette	
Category	Action widget
Purpose	A selectable (on/off) option item in a menu.
Events	Action, Value Changed, Key, Focus
Important Properties	text, selected, toolTip

Table 6.23. RCheckBoxMenuItem Summary

RCloseableTabbedDisplay

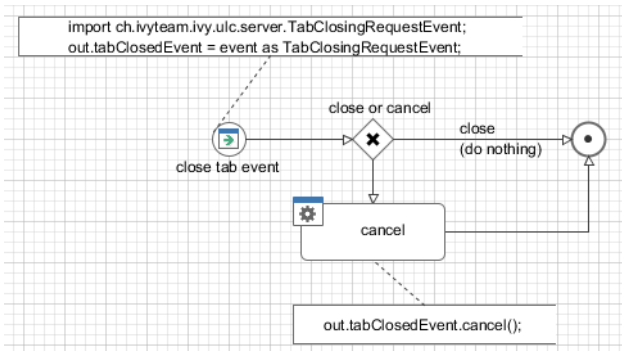


The `RCloseableTabbedDisplay` is an extension of the `RTabbedDisplay` but with closable tabs. Please consult the documentation of `RTabbedDisplay` for general information about tabbed displays.

In addition to the `RTabbedDisplay` the `RCloseableTabbedDisplay` displays a close icon for each tab.

A *Tab Closing Request* event is generated, when a user clicks the close icon of a tab. Processes that are mapped to this event can cancel the closing of the tab by calling the `cancel` method on the event. If none of the mapped event processes call the `cancel` method on the event, then the tab will be closed automatically at the end of the event process.

The following process and ivyScript snippets show how to cancel the closing somewhere in the event process of the *Tab Closing Request* event:



The close icon of the tab can be changed. For that the properties `tabCloseIconUri` and `tabInactiveCloseIconUri` are provided. Set a path to a CMS icon (for example `/myIcons/closeIcon`) to change the icons. If only the `tabCloseIconUri` is specified, a gray version of it is used as the inactive close icon.

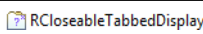
Symbol on Palette	
Category	Display widget
Purpose	Extension of <code>RTabbedDisplay</code> to handle closing of tabs.
Events	TabClosingRequest, SelectionChanged, Focus, Key
Important Properties	displayId, tabPlacement, toolTip, tabCloseIconUri, tabInactiveCloseIconUri

Table 6.24. RCloseableTabbedDisplay Summary

RCollapsiblePane

A collapsible pane is a container for *exactly one* sub component (e.g. another container) which can be collapsed and expanded programmatically by setting the property `collapsed` to `true` or `false` respectively. By default a collapsible pane collapses *vertically*.

You can switch off the animated collapse by setting the `animated` property to `false`. In this case the embedded component will simply disappear or appear at once.

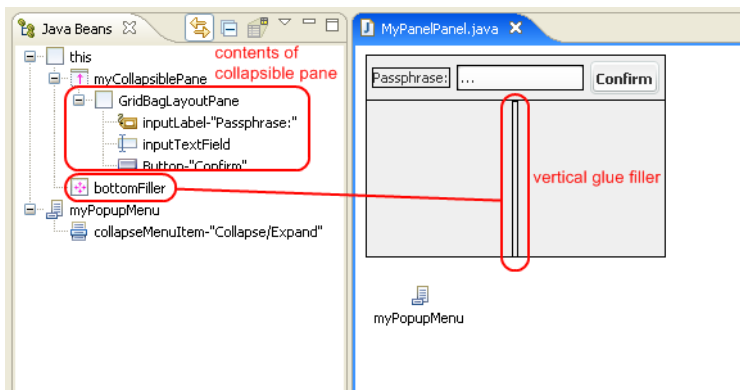
When working with a collapsible pane, a few things have to be taken into consideration. If a collapsible pane collapses with all it's content (i.e. effectively reduces it's size to 0), then extra space becomes available for all the other components on the panel.

This means that a collapsible pane can only be used within a container with a layout that in principle can handle the extra space, e.g. `RGridBagLayoutPane` or `RBorderLayoutPane`.

Furthermore the contents of the collapsible pane (including the collapsible pane itself) should never have any layout property set to request any extra space in the direction of the collapse (it's okay though to request any extra horizontal space, if the collapsible pane is oriented vertically).

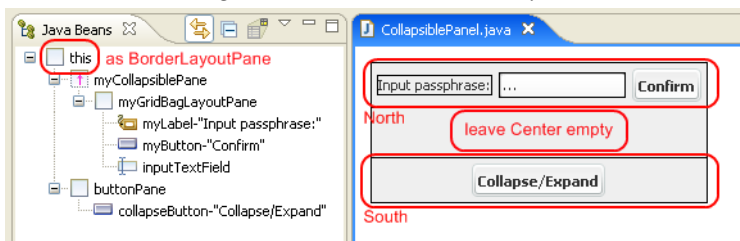
To achieve those requirements, the following two layout structures are recommended:

1. Use a surrounding container with a GridBag Layout.



Assuming a vertical collapsing pane place a component that is configured to consume any extra space above or below the collapsible pane. This component will take up the extra space that becomes available when the panel collapses and will grow into that direction. Therefore the collapsible panel will shrink either from bottom to top or from top to bottom, depending on whether the "space consumer" is placed below or above, respectively. The example above simply employs a "vertical glue" `RFiller` component to consume the extra space.

2. Use a surrounding container with a Border Layout.



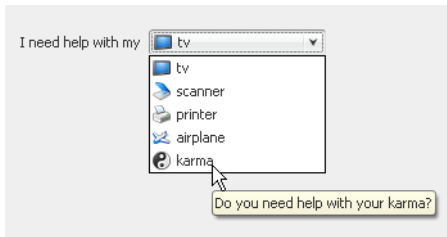
Assuming a vertical collapsing pane place the collapsible pane either into the north or south area of the surrounding container. If the panel collapses, the center area will receive and consume all extra space automatically and give it to the component which is placed there (if any). Therefore the collapsible panel will shrink either from bottom to top or from top to bottom, depending on whether it is placed in the north or south area, respectively. Typically all other areas of the surrounding border layout pane stay empty.

Symbol on Palette	
-------------------	--

Category	Container widget
Purpose	A container which has the ability to collapse (i.e. hide) and expand (i.e. show) a single embedded component.
Events	Focus, Key
Important Properties	collapsed, animated

Table 6.25. RCollapsiblePane Summary

RComboBox



A combo box allows the user to select one from a pull-down list of multiple options.

A combo box generates an *Action Event* if an option is selected.

You can configure both the appearance and the source of the data that a combo box should show on the Combo Box data tab of the Widget Configuration View. You can specify text and/or icon as well as a tool tip for each entry depending on the entry's value itself.

The data source can either be a *List*, a *Record Set* or an *Enumeration*. Depending on what is used, the properties `selectedListEntry`, `selectedRecord` or `selectedEnumValue` will return the currently selected value.



Note

If you have large lists to display in a Combo Box, you should set the `maximumRowCount` property. This limits the number of rows that are shown in the combo boxes drop-down list and instead offers a scroll bar to browse the whole range of options.



Tip

The width of the combo box will be decided by the widest list entry of the provided data. But what, if you load the data into the combo box lazily? Then the combo box will change its appearance after loading the data, which is often not desirable. You can prevent this by setting a `prototypeDisplayValue`, which is a string that will only be used to calculate the width of the initially shown combo box - but without displaying the string itself.


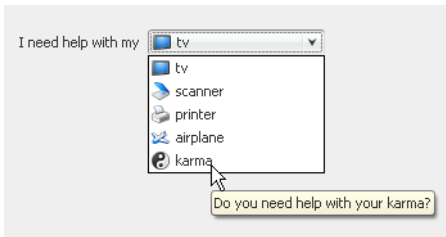
Symbol on Palette	
Category	Data widget
Purpose	A combo box allows to select exactly one from a list of options.
Events	Action, Focus, Key
Important Properties	<code>listData</code> , <code>recordsetData</code> , <code>enumClassName</code> , <code>selectedIndex</code> , <code>selectedListEntry</code> , <code>selectedRecord</code> , <code>selectedEnumValue</code> , <code>maximumRowCount</code> , <code>prototypeDisplayValue</code> , <code>font</code> , <code>toolTip</code>

Table 6.26. RComboBox Summary

RComboBoxCellWidget



A combo box cell widget is similar to a normal combo box like the RComboBox.

A combo box allows the user to select one item from a pull-down list of multiple items.

A combo box generates an *Action Event* if a item is selected.

You can configure both the appearance and the source of the data that a combo box should show on the Combo Box data tab of the Widget Configuration View. You can specify text and/or icon as well as a tool tip for each entry depending on the entry's value itself.

The data source can either be a *List*, a *Record Set* or a *Enumeration*. Depending on what is used, the properties `selectedListEntry`, `selectedRecord` or `selectedEnumValue` will return the currently selected value.



Note

You could define the mapped field between combo box value and the table cell value. See the Cell Widget Tab of the Widget Configuration View. for more details.



Note

If you have large lists to display in a Combo Box, you should set the `maximumRowCount` property. This limits the number of rows that are shown in the combo box drop-down list and additionally offers a scroll bar to browse the whole range of options.


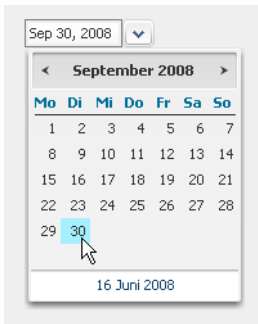
Symbol on Palette	 RComboBoxCellWidget
Category	Data widget
Purpose	A combo box allows to select exactly one from a list of options.
Events	Action, Focus, Key
Important Properties	<code>listData</code> , <code>recordsetData</code> , <code>enumClassName</code> , <code>selectedIndex</code> , <code>selectedListEntry</code> , <code>selectedRecord</code> , <code>selectedEnumValue</code> , <code>maximumRowCount</code> , <code>prototypeDisplayValue</code> , <code>font</code> , <code>toolTip</code>

Table 6.27. RComboBoxCellWidget Summary

RDatePicker



Date Pickers are used to query a date from an user. The date can either be entered directly into the Date Picker's date field or be comfortably selected with the integrated pop-up calendar (will start at the currently set date). The date picker widget is by default not initialized (empty) , use the `valueAsDate` property to set another date. Use the `setDate()` method with the parameter `null` to reset the `RDatePicker`

The date picker widget generates an *Action* event if a new date has been selected.



Note

If you want to force a certain date format for a `RDatePicker` to show and accept for input, then you should configure enable *Validation* in the Widget Configuration view of the respective Date Picker widget. See also section below.

Validation

The Widget Configuration View shows an additional Validation Tab if a `RDatePicker` is selected.

On this tab you can make this widget mandatory. You do not need any additional format validation for user entered dates, since the widget accepts only valid input. The widget uses the client language settings for date formatting.

Given that a valid format is ensured, you can access the value of a `RDatePicker` from attributes `date` and `valueAsDate`.

Attributes

The `date` field returns the selected/entered date value or `null` if the input was invalid (be careful with Axon.ivy's Null Handling feature).

The attribute `valueAsDate` delivers the selected date or a `Date.UNINITIALIZED_DATE` value if the user input does not represent a valid value.

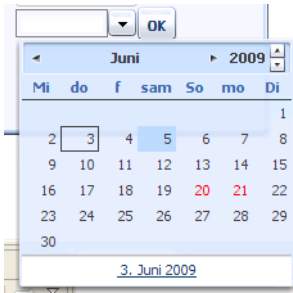
The attribute `firstDayOfWeek` sets the day that starts the week. So that setting e.g. the value "3" (which means Wednesdays, the default would be 1=Monday) then weeks under each other will be displayed starting with Wednesdays. (see screenshot below)

The attribute `flaggedDates` enables you to highlight some dates. You should provide an array of dates that should be highlighted (just like on the screenshot the 20 and 21.June).

The attribute `daysOfTheWeek` let you provide own titles for the days (see screenshot: "Mi"=Wednesday, "do"=Thursday, etc.)

The attribute `linkDate` enables you to customise the label and the destination date of the link at bottom of the picker (on the screenshot it is "[3.Juni 2009](#)", the default is today in the current language).

The attribute `timeZone` let you change the time zone the date picker should work with.



Events

The date picker generates *Action*, *Value Changed* and as usual *Focus Gained* and *Focus Lost* events. The *Action* event occurs whenever a user picks a date, or when he enters it into the text field manually (and hits an *Enter* key or leaves the field). Meanwhile *Value Changed* occurs when a user leaves the widget and the value has changed.

Summary


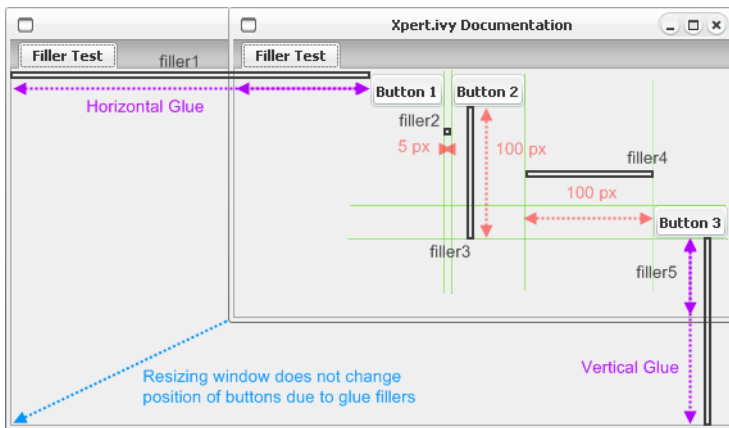
Symbol on Palette	
Category	Input widget
Purpose	Allows the user to specify a date with the help of a calendar.
Events	Key, Focus, Action, ValueChanged
Important Properties	valueAsDate, tooltip

Table 6.28. RDatePicker Summary

RFiller



Fillers are invisible glue components that consume space in a specific layout. They can either have a rigid fixed size (in which case they simply keep a fixed distance between two components) or they can have a *glue* behaviour, in which case they consume additional space, e.g. if a window gets resized. The latter is typically the intended behaviour if fillers are used inside a GridBagLayout.



Tip

The real advantage of fillers only comes into play when they are used with style configurations. By using differently styled fillers you can later simply change the styles to adjust the distances kept instead of having to reconfigure each filler instance individually. However, when defining styles for fillers, keep in mind to use *logical* names for your styles: if you name a style "50x50px" it may lead to unclear behavior if you later redefine the filler style to occupy 40x30 pixels.

Style Configuration

The style configuration of a filler define whether it is used as *glue* or as a *rigid size* block.

If the `preferredIsExactSize` style property of a filler is set to `true`, then the filler will never change it's preferred size, no matter whether the layout changes due to resizing. In other words: the minimum and maximum size will be set to the preferred size and all other properties will be ignored.

settings	values
size	
<input checked="" type="checkbox"/> <code>preferredIsExactSize</code>	<code>true</code>
preferredSize	
<input checked="" type="checkbox"/> <code>width</code>	<code>30</code>
<input checked="" type="checkbox"/> <code>height</code>	<code>30</code>
<input type="checkbox"/> <code>minimumSize</code>	
<input type="checkbox"/> <code>maximumSize</code>	
<input type="checkbox"/> <code>layoutStyle</code>	

If the property is set to `false`, then the filler will behave as configured by the rest of the style properties, i.e. it will request to be displayed with it's preferred size, but may change size to the specified minimum and maximum size, if the managing layout manager (typically `GridBagLayout`) requests so, due to size changes of the surrounding container. In this way, the described *glue* behaviour can be achieved. By default, the styles `verticalGlue` and `horizontalGlue` are already available for all fillers.

settings	values
size	
<input checked="" type="checkbox"/> <code>preferredIsExactSize</code>	<code>false</code>
<input type="checkbox"/> <code>preferredSize</code>	
<input type="checkbox"/> <code>minimumSize</code>	
maximumSize	
<input checked="" type="checkbox"/> <code>width</code>	<code>1000</code>
<input checked="" type="checkbox"/> <code>height</code>	<code>1000</code>
layoutStyle	
<input checked="" type="checkbox"/> <code>alignmentX</code>	
<input checked="" type="checkbox"/> <code>alignmentY</code>	
<input checked="" type="checkbox"/> <code>fill</code>	<code>BOTH</code>
<input checked="" type="checkbox"/> <code>weightX</code>	<code>1</code>
<input checked="" type="checkbox"/> <code>weightY</code>	<code>1</code>
<input checked="" type="checkbox"/> <code>anchor</code>	


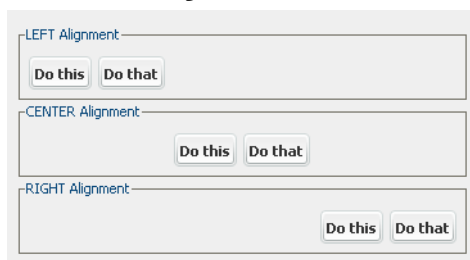
Symbol on Palette	
Category	Layout Helper widget
Purpose	Fillers consume space between widgets in a configured way.
Events	No events
Important Properties	Do not set properties of a filler directly, use pre-configured <i>Styles</i> instead.

Table 6.29. RFiller Summary

RFlowLayoutPane



A flow layout pane is the simplest container layout available. It uses a Flow Layout where all added components are arranged in a single row. The row's contents are aligned either *left*, *right* or *centered* if extra space is available.

The desired alignment of components can be set with the `alignment` property. The amount of horizontal spacing that is used between the components can be specified by setting the `hGap` attribute. The amount of space that is added above and below the components can be set with the `vGap` attribute.



Note

Usage of `RFlowLayoutPane` is generally not recommended. Use a flow layout only as a simple solution if you need to arrange few (one or two) widgets of about equal height in a single row and if you know that you have enough space available.

A common use is to put a flow layout into the *north* or *south* area of a `RBorderLayoutPane` to add, for example, a single button or a label. The surrounding flow layout will then prevent the automatic horizontal stretching of the component(s). But even here you should consider using a `RGridBagLayoutPane` with some horizontal glue filler components instead, because you'll stay more flexible.


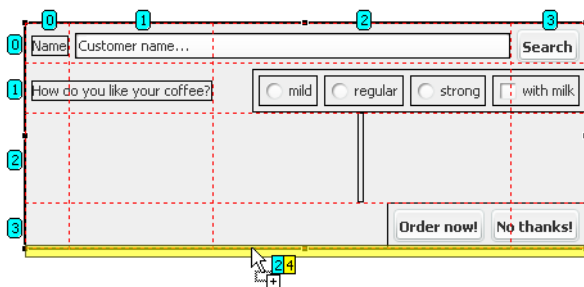
Symbol on Palette	 RFlowLayoutPane
Category	Container Widget
Purpose	A container which aligns it's added component <i>left</i> , <i>centered</i> or <i>right</i> in a single row.
Events	Focus, Key
Important Properties	<code>alignment</code> , <code>hGap</code> , <code>vGap</code>

Table 6.30. RFlowLayout Summary

RGridBagLayout



A grid bag layout pane is the most powerful container in terms of layouting, it uses a `GridBag Layout` to arrange it's components. In a grid bag layout components are arranged and aligned in a grid, but each component may span several grid cells both horizontally and vertically. The sizes of grid cells are generally determined by the tallest and widest components in their row or column, respectively. If a component has more space available than it's preferred size measures, then it may be instructed to consume the additional horizontal or vertical space or to align itself inside the cell, either at *north*, *northwest*, *west*, *southwest*, *south*, *southeast*, *east*, *northeast* or *northwest* position. By using nested containers that provide an own layout but for the area they occupy the design of forms is very flexible.

Because components can be configured to consume extra available space, grid bag layout containers are largely insensitive with respect to size changes of a window, for example. Components maintain their relative positions and may grow or shrink according to the configuration. By using `RFiller` components, placeholders can be used that have the sole purpose of consuming as much space as possible either horizontally or vertically or both.



Tip

When designing complex layouts with grid bag it is generally advisable to first sketch the screen that you want to create with a pencil on a piece of paper before starting to design on screen. In a next step you should then identify (perhaps with another color) the groups of components that might be layouted by a sub-container as well as the major horizontal and vertical layout guide lines, which will usually indicate the row and column

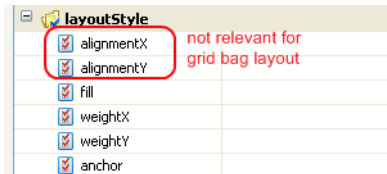
borders. Finally you should then identify the components that may grow and shrink if the container is resized and possibly place filler components in strategic places, which may take up extra space.



Warning

The flexibility and power of grid bag layout may lead to some undefined results if configured improperly, especially when using cell spanning. Make sure that you understand the possibilities of the layout if you intend to create complexly layouted screens. You may for example consult GridBag Layout for a more detailed explanation of the features of this layout.

The following layout style properties are available for all components that are contained inside a grid bag layout pane:



Property Name	Description
fill	Defines whether the component should consume extra space that becomes available. Options are: <i>horizontal</i> , <i>vertical</i> , <i>both</i> or <i>none</i> .
weightX	Specifies the percentage of any extra available <i>horizontal</i> space that the component will consume. <i>0</i> means <i>none</i> and <i>1</i> means <i>all</i> . This value should add up to 1 over all components in a column. If the cumulated value is greater or smaller than 1 then the available space will be distributed to the components relatively to the specified values.
weightY	Same as <code>weight X</code> , but for the extra <i>vertical</i> space.
Anchor	This setting has only an effect, if <code>fill</code> is not equal to <i>both</i> . In the event of any extra available space the component will then "stick" to the position defined here, e.g. to the top of the cell if <i>north</i> is defined or in the lower left corner if <i>southwest</i> is defined.

Table 6.31. Style Properties for Grid Bag Layout

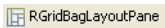
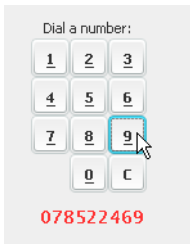
Symbol on Palette	
Category	Container widget
Purpose	A container with a very flexible and powerful row/column based layout manager.
Events	Focus, Key
Important Properties	-
Component Layout Properties	fill, weightX, weightY, anchor (see above)

Table 6.32. GridBagLayoutPane Summary

RGridLayoutPane



A grid layout pane arranges its components in n rows and m columns in cells of equal size according by using a Grid Layout. The size of the grid cells is determined by the tallest and the widest contained component. All components are stretched to this size. Components *can not* span across multiple columns or rows.

The horizontal and vertical space between rows and columns can be configured by setting the `hGap` and `vGap` properties, respectively. The number of desired *columns* and *rows* can be configured by setting the `columns` and `rows` properties, respectively. The number of columns will increase automatically by one, if more than $n \times m$ components are added, but the number of rows stays constant at the configured value.

Because all contained components are forced to have the same size, this pane is usually only employed in special situations, e.g. for button panels, where it is desired for aesthetic reasons that all buttons have the same size, independent of the length's of their texts.



Note

Arranging the components of a grid layout pane in the editor area of the Visual Editor can be tricky or even impossible. *You should always use the Java Beans View to line up the components in the correct sequence.* The grid layout pane will fill then the grid in a row-by-row manner, starting with the first component in the top row.

If you want to keep grid cells "empty" you should place `RFiller` widgets as placeholders into the sequence of components. Inserting filler with a *rigid* size that is larger than those of all other components can also be used to define a prototype cell size which will be used for all other cells.


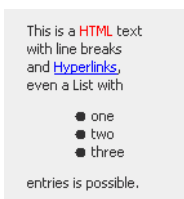
Symbol on Palette	 RGridLayoutPane
Category	Container widget
Purpose	A container which arranges its components in $n \times m$ grid cells of equal size.
Events	Focus, Key
Important Properties	<code>rows</code> , <code>columns</code> , <code>hGap</code> , <code>vGap</code>

Table 6.33. RGridLayout Summary

RHtmlPane



The html pane widget can be used to render a (small) HTML document on a label. Actually it is between `RBrowser` and `RLabel`. `RBrowser` is capable to render more complicated HTML pages, and it allows navigation for a user just like `RHtmlPane` per default. `RLabel` can also render HTML content if you surround content with the `<HTML>` tag. Meanwhile working with `RHtmlPane` we recommend starting the text that the widget should display with an `<HTML>` tag.



Warning

RHtmlPane enables navigation just like a browser through HTML links per default.



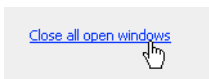
Tip

To disable hyperlink navigation use the property *veto* to true. This case you may add VetoListeners that can decide if a Hyperlink navigation should be enabled. Per default there is no such VetoListener added, so that Hyperlinks will have no effect.

Symbol on Palette	
Category	Label widget
Purpose	A multi-line label which can display HTML texts with links.
Events	Focus, Key, Hyperlink
Important Properties	text, tooltip, veto

Table 6.34. RHtmlPane Summary

RHyperlink



A hyperlink can be used in place of a button. It takes less room and may look better in some places. An optional icon can also be provided, the configuration possibilities are equal to those of a button.

Hyperlinks are rendered in a different color, once they have been activated. The hyperlink generates both an *Action* event and a *Clicked* event if it is clicked.

Symbol on Palette	
Category	Action widget
Purpose	A hyperlink that can be activated with a mouse click.
Events	Action, Clicked, Focus, Key
Important Properties	text, icon, focus, key, tooltip

Table 6.35. RHyperlink Summary

RHyperlinkCellWidget



Hyperlinks are typically used to start an action or to generate an event. The hyperlink generates an *Action Event* if it is clicked. The hyperlink text, icon and tooltip are taken from the cell value.



Table Column

Field	Header	Value	ToolTip	Icon
		value		ivy.cms.cr("/icons/twitter")

You will usually use `RHyperlinkCellWidget` if you need a clickable icon and / or text cell.

To process the action event of the hyperlink, map the event to a Rich Dialog Process start in the Rich Dialogs panel logic. In the code tab of the Event Start Process step enter the following code:

```
import ch.ivyteam.ivy.richdialog.widgets.components.customrenderers.RHyperlinkCellWidget;
Object source = event.getSource();
if(source instanceof RHyperlinkCellWidget) {
    RHyperlinkCellWidget hyperlink = source as RHyperlinkCellWidget;
    ivy.log.debug(" " + hyperlink.getRow() + " : " + hyperlink.getColumn());
}
```


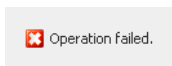
Symbol on Palette	 RHyperlinkCellWidget
Category	Action widget
Purpose	A hyperlink with a text and / or icon to start an action. The cell value is displayed as text.
Events	Action, Focus, Key
Important Properties	text, icon

Table 6.36. RHyperlinkCellWidget Summary

RLabel



Labels are used to write short texts on the UI, e.g. for labelling text fields or for writing status or title messages. A label can have an optional icon to display, the relative position of text and icon can be configured, as well as the gap between them.

Labels are also capable of displaying simple HTML. Start the text that the label should display with an `<html>` tag. This way you can for example use the `` tag to show parts of a text in a different color, or use `` to render text or in bold type. Use `
` to render multiple lines.

Labels can be assigned to another widget by using the `labelFor` property. If this property is set, the focus will be forwarded to the assigned widget, if the label receives it.


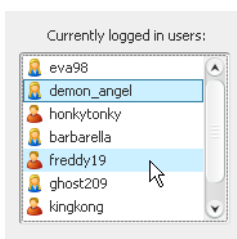
Symbol on Palette	 RLabel
Category	Label widget
Purpose	A label which can display plain or simple HTML texts.
Events	Focus, Key
Important Properties	text, icon, tooltip, labelFor, font

Table 6.37. RLabel Summary

RList



The list widget shows an arbitrary number of entries which are also available for (multi-) selection.

A list generates a *List Selected Event* if an entry is selected and an *Action* event if a list entry is double clicked.

You can configure both the appearance and the source of the data that a list should show on the List Data tab of the Widget Configuration view. You can specify text and/or icon as well as a tool tip for each entry depending on the entry's value itself.

The data source can either be a *List* or a *Record Set*. Depending on what is used, the properties `selectedListEntry`; `selectedListEntries` or `selectedRecord`; `selectedRecords` will return the currently selected value(s). The selection mode (single selection, single interval selection or multiple selection) can be set with the `selectionMode` property.

With the property `autoSelectFirstEntry` you can have the widget select the first data entry automatically (if data is present). This has also the effect, that a call to `clearSelection()` will reset the selection to the first data entry (instead of clearing). The property is set to *false* by default.



Note

Because lists tend to contain many items, it is recommended that you place RList widgets inside an RScrollPane container, so that they become scrollable if the number of items becomes too large for the allocated view port size.

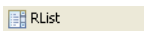
Symbol on Palette	
Category	Data widget
Purpose	A list shows an arbitrary number of items and allows the user to select one or multiple entries from that list.
Events	List Selected, Action, Focus, Key
Important Properties	<code>listData</code> , <code>recordsetData</code> , <code>selectedIndices</code> , <code>selectedListEntry</code> , <code>selectedListEntries</code> , <code>selectedRecord</code> , <code>selectedRecords</code> , <code>autoSelectFirstEntry</code>

Table 6.38. RList Summary

RListDisplay



Displays are placeholder widgets onto which Rich Dialogs can be loaded dynamically at runtime. A *List Display* shows the loaded Rich Dialog panels as entries of a list.

A list display automatically shows a scroll bar if the number of loaded Rich Dialogs exceeds the available space. You can configure and override this behavior by setting the properties `horizontalScrollBarPolicy` and

`verticalScrollBarPolicy` to the respective values `NEVER`, `AS_NEEDED` or `ALWAYS` in the properties view (the default is `AS_NEEDED`).

The `RListDisplay` does not support selection or offer selection events, but a Rich Dialog can be scrolled into the visible area by either selecting it on the display with its name (i.e. the name that is provided by a call to `IRichDialogPanel.getName()`).



Warning

In order to being able to address a display widget (when loading a Rich Dialog with a Rich Dialog Process element) you must not forget to assign the display an *unique* id on the **displayId** property. If you fail to do so then the display might not be found at runtime and Rich Dialogs may not be loaded correctly.


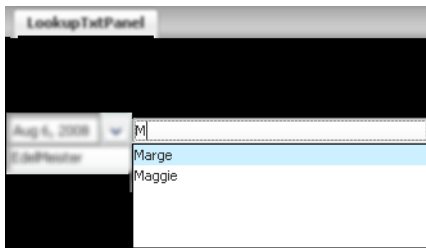
Symbol on Palette	
Category	Display widget
Purpose	Placeholder for Rich Dialogs that are loaded dynamically at runtime. Shows loaded components in a scrollable list.
Events	Focus, Key
Important Properties	<code>displayId</code> , <code>horizontalScrollBarPolicy</code> , <code>verticalScrollBarPolicy</code> , <code>toolTip</code>

Table 6.39. RListDisplay Summary

RLookupTextField



The `RLookupTextField` is a mixture between `RTextField` and `RComboBox`. The text field shows a list of suggestions while typing, the suggestions are filtered according to the entered text, as the user continues typing. At any point of typing, the user may choose to select one of the suggestions or stay with his input.

The selection of a suggested entry is performed if the user hits either *Enter* or *Tab* during typing. The user may use the *up* and *down Arrow Keys* or the mouse to select a specific entry from the suggestion list.

Events

As well as `RTextField`, `RLookupTextField` also offers events like *Value Changed Event*, *Key Event* and *Selection Changed* events.

Key Event is sent when ever a key is typed, the *Value Changed Event* is only sent when the user leaves the text field and the value has changed.

The *Value Changed Event* is transmitted *synchronously* and the *Key Event* is transmitted *asynchronously*.

Data

Data for an `RLookupTextField` is configured very similar to a `RComboBox`. You can configure both the appearance and the source of the suggestions that the text field shows on the Data tab of the Widget Configuration view. You can specify the display strings of the suggestions for each entry depending on the entry's value itself.

The data source can either be a *List* or a *Record Set*. Depending on what is used, the properties `getSelectedListEntry()` or `getSelectedRecord()` will return the currently selected value. If no value from the suggestion list has been chosen, then `null` will be returned by those methods.

Use `getText()` to get the text that is actually shown in the text field, which may be different from the associated selection's model entry (e.g. if the user has entered a text that does not match an entry, or if the entry's display string is different from the entries value).

Special Properties

With the property `lazyLookupIdleTime` you can set an idle time in milliseconds. The text field will wait for a pause in typing that is longer than this idle time, before the entered text is sent to the server (thus filtering the displayed suggestions). Setting an higher idle time is recommended, if your users type very fast and feel annoyed by the fact, that the outgoing requests to filter large suggestion lists block their input.

If the property `forceSelection` is set to `true` then only selections from the suggestion list are accepted. The text field is then used merely to filter the proposals. If the entered text does not match any of the suggestions it will be marked as invalid. When leaving the text field, then invalid text will be discarded and the text will be reset to the last selected valid value (which is possibly empty).

If the property `forceSelection` is set to `false`, then user input which is not among the proposed entries will also be accepted. In this case, it may occur that querying the selected value with `getSelectedRecord()` or `getSelectedEntry()` results to `null`, because the user input does not match any of the proposals. In this case you should use `getText()` to access the text that was entered by the user.

If `autoSelectFirstEntry` is set to `true`, then the text of the `RLookupTextField` will be initialized with the first entry of the data list.

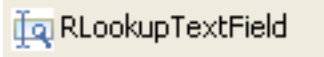
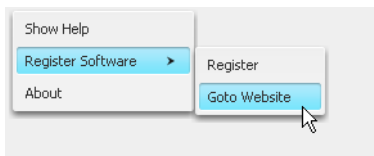
Symbol on Palette	
Category	Input widget
Purpose	An <code>RTextField</code> that offers a lookup function to help text selection from a list. As you begin to type your text a selection list will appear. This list shows prefix matches with the text you typed in.
Events	Key, Focus, Value Changed
Important Properties	<code>listData</code> , <code>recordsetData</code> , <code>selectedListEntry</code> , <code>selectedRecord</code> , <code>valueAsString</code> , <code>lazyLookupIdleTime</code> <code>autoSelectFirstEntry</code> , <code>forceSelection</code> , <code>autoSelectFirstEntry</code>

Table 6.40. RLookupTextField Summary

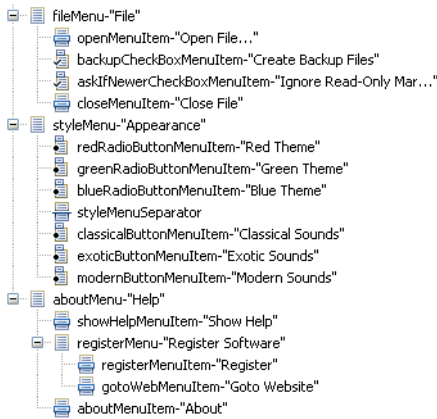
RMenu



A menu is a container for `RMenuItem` (and similar) widgets. It can be *nested* within other menus, including `RPopupMenu`. Menus can also be added to a `RMenuBar`.

Menu widgets do not provide any events and they don't have any special properties worth mentioning, use the `Widget Configuration Text/Icon Tab` to set text and optional icon of a menu.

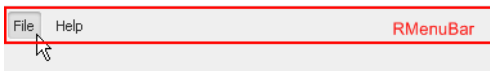
Since menu widgets do not have a visual representation on the panel they should be placed directly inside the widget tree of the Java Beans view. This view allows you to comfortably set up menu hierarchies using sub menus and menu items:



Symbol on Palette	
Category	Container widget
Purpose	Provide a container for menu items and thereby define contents of a menu.
Events	-
Important Properties	-

Table 6.41. RMenu Summary

RMenuBar



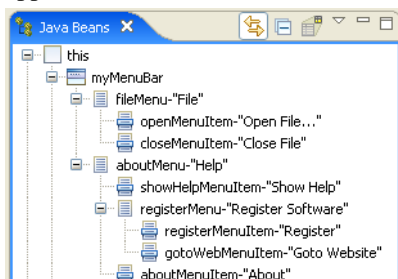
Menu bars are containers for RMenu widgets. A menu bar is constantly visible (as opposed to a RPopupMenu which only appears on demand) and shows the available top level menus with their name. If a menu is clicked, then it unfolds and displays it's entries which are then available for selection. Menu bars are typically placed at the top of a screen, but they don't have to.

Menu bars do not support any events and they don't have any special properties worth mentioning.



Note

Although menu bars do have a visual representation on the edited panel, the added menus should not be dropped onto the menu bar component inside the visual editor area. Instead they should be inserted below the menu's representation in the Java Beans view, where the hierarchical structure of the menu components becomes apparent.

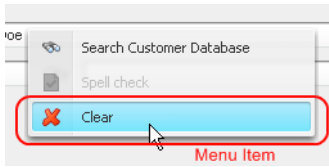


Symbol on Palette	
Category	Container widget
Purpose	A container for menu objects.
Events	-

Important Properties	-
----------------------	---

Table 6.42. RMenuBar Summary

RMenuItem



Menu items are the standard entries for RMenu and RPopupMenu widgets. They can have a *text* and/or *icon*. In fact, a menu item behaves exactly the same as a RButton in all aspects of its use.

Menu items generate an *Action* event if they are selected.

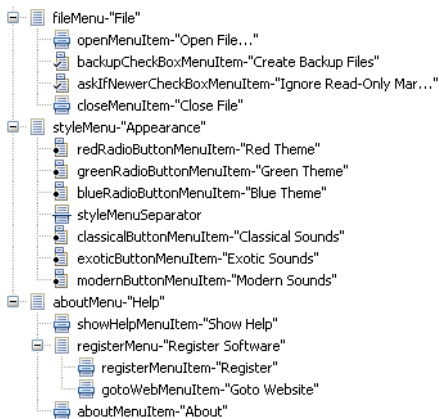
Since menu items do not have a direct visual representation they should be placed directly on or below a menu widget inside the Java Beans view. This view allows you to comfortably set up menu hierarchies using sub menus and menu items:



Note

In the current version of Axon.ivy you are not yet able to map the action for menu entries to process starts with drag and drop, as with button actions for example.

To map menu entries you have to open the *New Event Mapping dialog* from the context menu of any RMenuItem widget (or a subclass thereof) inside the Rich Dialog editor.




Symbol on Palette	
Category	Action widget
Purpose	A clickable item of a menu.
Events	Action, Key, Focus
Important Properties	text, icon

Table 6.43. RMenuItem Summary

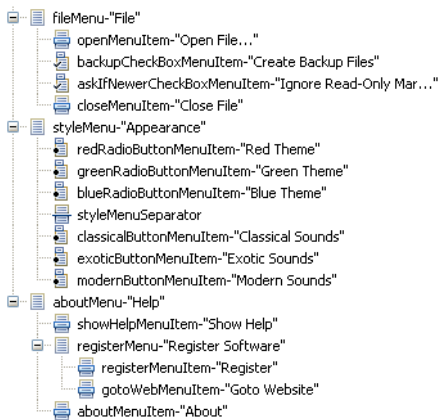
RMenuSeparator



Menu separators are used to visually separate menu items into groups. Simply place them between any menu items of a menu container in the Java Beans view.

Menu separators do not generate any events.

Since menu separators do not have a direct visual representation they should be placed directly on or below a menu widget inside the Java Beans view. This view allows you to comfortably set up menu hierarchies using sub menus and menu items:




Symbol on Palette	 RMenuSeparator
Category	Layout Helper widget
Purpose	Visually separate menu items into groups.
Events	-
Important Properties	-


Table 6.44. RMenuSeparator Summary

RPasswordField



Password fields are used to query passwords from the user. The entered input is not rendered in clear text but as * characters. The text field generates a *Value Changed* event and a *Key* event every time a key is typed and an *Action* event if the return/enter key is pressed.

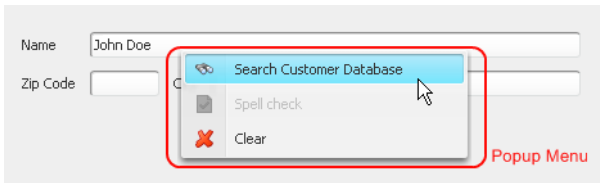
The *Value Changed* event is transmitted *synchronously* and the *Key* event is transmitted *asynchronously*. Since the synchronous event handling may lead to a blocking of the UI if the user types too fast, it is generally recommended to map the asynchronous *Key* event instead (will keep the UI responsive).

Symbol on Palette	 RPasswordField
Category	Input widget
Purpose	A field where the user can enter a password.

Events	Key, Focus, Action, Value Changed
Important Properties	text, echoChar, toolTip, font

Table 6.45. RPasswordField Summary

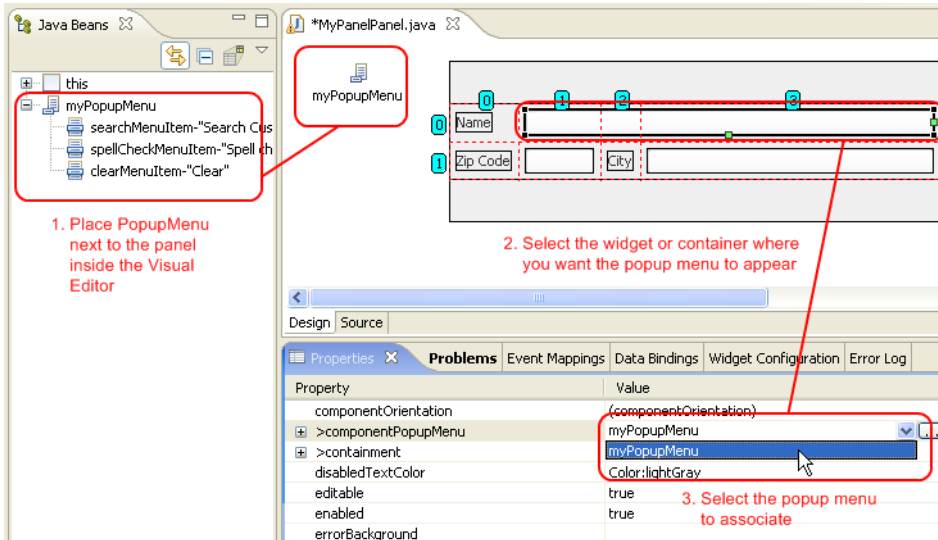
RPopupMenu



A popup menu is a container for RMenuItem (and similar) widgets. It can be associated with a component or a container and will normally appear when the user clicks the right mouse button on that component.

RPopupMenu widgets do not provide any events and they don't have any special properties worth mentioning.

Since popup menus do not have a visual representation on the panel they must be placed *next to the panel* rather than on the panel itself:



To associate a popup menu with a component, you first select the component inside the Visual Editor or inside the Java Beans view and then set the menu object to be associated on the `componentPopupMenu` attribute in the Properties view, as demonstrated above.



Note

The popup menu can also be inherited from a parent component, i.e. you can associate a popup menu with a top level container (e.g. the panel itself) and can then reuse it on the inner components. If you consult the Java Beans view, you can easily identify the components that are able to inherit the popup-menu of a container.

foreground	51,51,51
horizontalAlignment	LEFT
inheritsPopupMenu	true
mandatory	true
mandatoryBackground	false
mandatoryBorder	

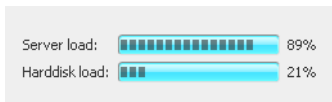
By inheriting the popup menu of a parent component you don't have to associate a popup menu with each sub component again, if you should decide to use another menu on the parent. Inheritance is supported over multiple levels, as long as the direct parent component also inherits the menu.

Symbol on Palette	
-------------------	--

Category	Container widget
Purpose	Define the contents of a context / popup menu on a component.
Events	-
Important Properties	-

Table 6.46. RPopupMenu Summary

RProgressBar



A progress bar can be used to inform the user about the amount of completed work or to show a percentage graphically. Progress bars can either have a *horizontal* or *vertical* orientation, which can be specified with the `orientation` property. Progress bars are pure information widgets, i.e. the displayed information can not be altered directly by the user in any way. .

The appearance and behavior of a progress bar can be configured: The minimum and maximum value can be set and an optional label string can be defined which can be rendered over the progress bar.



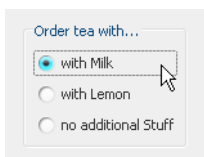
Tip

When you update progress (value property) the GUI on the client side will only be updated when you leave the process you are currently in. So that you have to use a timer with callback method to show a changing progress bar.

Symbol on Palette	RProgressBar
Category	Information widget
Purpose	Show a percentage value graphically.
Events	Focus, Key
Important Properties	value, maximum, minimum, orientation, string, stringPainted, toolTip

Table 6.47. RProgressBar Summary

RRadioButton



Radio buttons are used give the user the possibility to select exactly one from a number of options. The radio button will generate both an *Action* event and a *Value Changed* event if it is selected or deselected.

You can provide your own *selected* and *unselected* icons (if you want to replace the standard radio mark).



Note

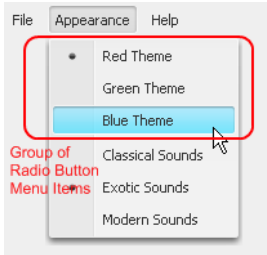
Radio buttons should always be used in conjunction with a `ButtonGroup` to ensure that only one can be selected at a given time.

Symbol on Palette	RRadioButton
Category	Action / Selection widget

Purpose	A radio button that allows to select exactly one from multiple options.
Events	Action, Focus, Key, Value Changed
Important Properties	text, tooltip, selected, group

Table 6.48. RRadioButton Summary

RRadioButtonMenuItem



Radio button menu items are typically used in groups of at least two and represent a number of *exclusively* selectable options. They can be used both inside RMenu and RPopupMenu widgets. The standard radio button icon of a RRadioButtonMenuItem can be replaced by setting a different *Icon Uri* and *Selected Icon Uri* in the Widget Configuration Text/Icon Tab. Generally, radio button menu items behave exactly the same as a RRadioButton widgets.

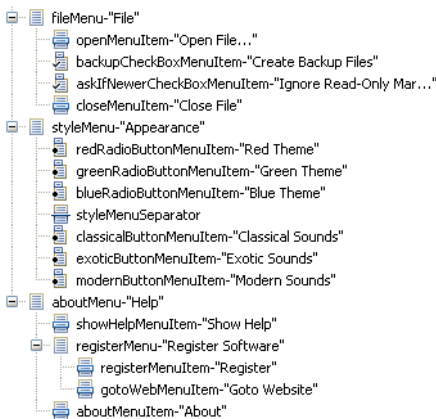
Radio button menu items generate both an *Action* event and a *Value Changed* event if they are selected or deselected.

Since radio button menu items do not have a direct visual representation they should be placed directly on or below a menu widget inside the Java Beans view. This view allows you to comfortably set up menu hierarchies using sub menus and menu items:



Note

Radio button menu items should always be used in conjunction with a ButtonGroup to ensure that only one can be selected at a given time.




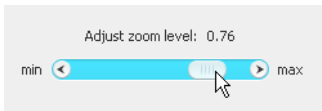
Symbol on Palette	
Category	Action widget
Purpose	One of a number of exclusively selectable items in a menu.
Events	Action, Value Changed, Key, Focus
Important Properties	text, selected, tooltip, group

Table 6.49. RRadioButtonMenuItem Summary

RScrollBar



A standalone scroll bar can be used to let the user adjust a property in a continuous way; e.g. as an alternative to a RSlider widget which only offers to select a range value in discrete steps. Scroll bars can either have a *horizontal* or *vertical* orientation, which can be specified with the `orientation` property.

A scrollbar produces continuously *Adjustment* events if the knob is dragged with the mouse.

The scrollbar's position value is a value between 0 and 1, indicating the percentage at which the bar's knob is positioned.


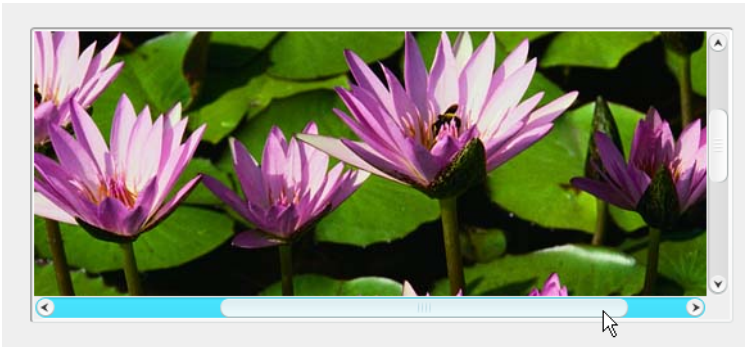
Symbol on Palette	 RScrollBar
Category	Input widget
Purpose	Select a value between 0 and 1 in a continuous way, by positioning the scrollbar knob with the mouse.
Events	Adjustment, Focus, Key
Important Properties	<code>position</code> , <code>orientation</code> , <code>unitIncrement</code> , <code>blockIncrement</code> , <code>toolTip</code>

Table 6.50. RScrollBar Summary

RScrollPane



A scroll pane provides a "viewport" (i.e. a window) that can be moved about a single sub component which is typically larger than the area taken up by the scroll pane container. Different scroll bar policies both for vertical and horizontal scrolling can be configured by setting the properties `verticalScrollBarPolicy` and/or `horizontalScrollBarPolicy` to *AS_NEEDED*, *ALWAYS* or *NEVER* respectively. Scrolling with the mouse wheel can be enabled or disabled by setting the property `wheelScrollingEnabled`.

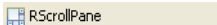
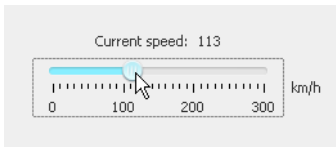
Symbol on Palette	 RScrollPane
Category	Container widget
Purpose	A for a single sub-component providing horizontal and vertical scrollbars to move the viewport about the contents of the subcomponent.
Events	Focus, Key
Important Properties	<code>verticalScrollBarPolicy</code> , <code>horizontalScrollBarPolicy</code> , <code>wheelScrollingEnabled</code>

Table 6.51. RScrollPane Summary

RSlider



A slider widget can be used to let the user adjust a property in a discrete or continuous way. Sliders can either have a *horizontal* or *vertical* orientation, which can be specified with the `orientation` property. With the `inverted` property the orientation of the range of values can be selected.

The appearance and behavior of a slider can be further configured: The maximum and minimum range value can be set, as well as the minor and major tick spacing. You can define whether the ticks should be painted, whether the track should be painted and whether the knob should snap to the tick marks or not.

A slider produces *Value Changed* events if the knob is dragged with the mouse.

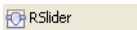
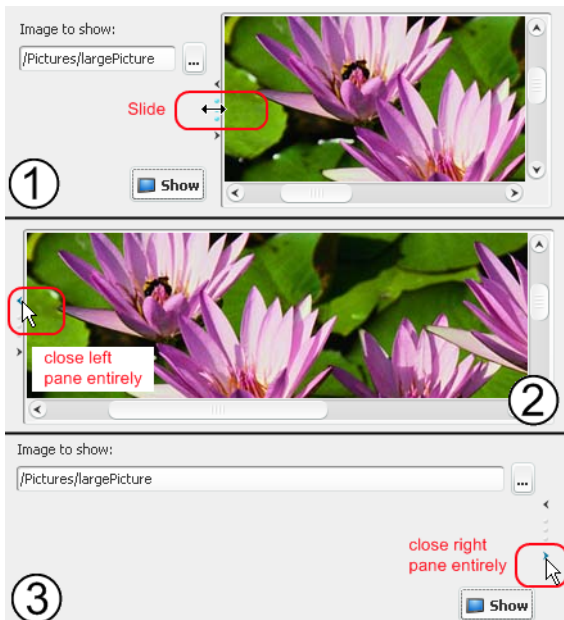
Symbol on Palette	
Category	Input widget
Purpose	Select an numerical value between a defined minimum and maximum, by positioning the slider knob with the mouse.
Events	Value Changed, Focus, Key
Important Properties	<code>value</code> , <code>minimum</code> , <code>maximum</code> , <code>orientation</code> , <code>inverted</code> , <code>minorTickSpacing</code> , <code>majorTickSpacing</code> , <code>paintTicks</code> , <code>snapToTicks</code> , <code>toolTip</code>

Table 6.52. RSlider Summary

RSplitPane



A split pane divides the available space into two areas, either *left* and *right* or *top* and *bottom*, into each of which exactly one component or sub-container may be placed.

In between the two areas, a *divider* component allows to distribute the available space with the mouse. Use the `orientation` property to set the split pane's orientation to either *horizontal split* or *vertical split*. The `dividerLocation` attribute may be used to initially place the divider.

The divider can either be continuously dragged with the mouse or any of its two arrow icons can be clicked, in which case it will move completely to the indicated side, hence closing the component on this side entirely (see examples above). If you set the `oneTouchExpandable` property to `true` then the divider may be moved to the opposite side with a single mouse click (this behaviour is *off* by default).

Normally the two components are only re-laid out when the user lets the divider go. If you want a continuous re-laying out during the drag then you must set the `continuousLayout` property to `true`.



Note

Initially the split pane container shows with two "dummy" buttons in its left and right (or upper and lower) half. Those buttons are not really contained on the panel and only shown as visual help. Simply drop a component on top of the respective button to replace it.



Warning

Unfortunately the initial location of the split pane divider can only be set in absolute pixels (and not relative in %). Since you may not always know the size that your split pane will receive initially it is recommended to set a preferred or minimum size of the split pane and then position the slider with respect to the width (or height) of the configured size. This is not a perfect solution but for many cases an acceptable workaround.



Tip

Resizing split pane does not work as you expect?

The `dividerLocation` seems to change in its own way. Take a look at the `setResizeWeight(double)` method to control distribution of extra space.


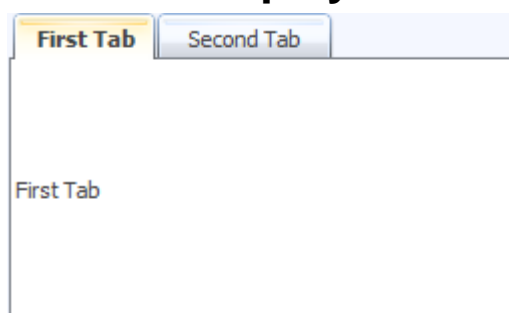
Symbol on Palette	
Category	Container widget
Purpose	A two-area container with a movable divider to redistribute space.
Events	Focus, Key
Important Properties	<code>orientation</code> , <code>dividerLocation</code> , <code>resizeWeight</code> , <code>oneTouchExpandable</code> , <code>continuousLayout</code> , <code>dividerSize</code>

Table 6.53. RSplitPane Summary

RTabbedDisplay



Displays are placeholder widgets onto which Rich Dialogs can be loaded dynamically at runtime. A `RTabbedDisplay` shows the loaded Rich Dialog panels as tabs.

A tabbed display generates a *Selection Changed* event if one of its tabs (i.e. loaded Rich Dialogs panels) is selected. This also happens if a new Rich Dialog is loaded onto the display, because it will become the selected tab.



Warning

In order to being able to address a display widget (when loading a Rich Dialog with a Rich Dialog Process element) you must not forget to assign the display an *unique* id on the `displayId` property. If you fail to do so then the display might not be found at runtime and Rich Dialogs may not be loaded correctly.


Symbol on Palette	
Category	Display widget
Purpose	Placeholder for Rich Dialogs that are loaded dynamically at runtime. Shows loaded components as tabs.
Events	Selection Changed, Focus, Key
Important Properties	<code>displayId</code> , <code>tabPlacement</code> , <code>toolTip</code>

Table 6.54. RTabbedDisplay Summary

RTabbedPane



A tabbed pane is a container which arranges its components behind each other (similar to `RCardPane`) but making each one directly accessible by providing tabs. By clicking on a tab the associated component is brought to front. The location of the tabs can be configured as *top*, *bottom*, *left* and *right* with the property `tabPlacement`. The currently selected tab can be set with the `selectedIndex` attribute. The added components are associated with an index starting at `0`.

Any kind of widget or container can be a "tab" on a tab pane. Typically tabs are containers, i.e. sub-panels, with an individual layout.

Each tabbed component can have its tab labelled by given a tool tip by setting the properties `title` and `toolTipText` inside the `containment` attribute, respectively.

>containment	title: "Customer", icon: , tip:
icon	
>title	Customer
toolTipText	
enabled	true



Note

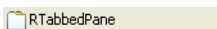
As you can see in it is also possible to set an icon for each tab. However this is currently not well supported by *Axon.ivy*. To do so, you should copy the image that you want to use (*png*, *gif* or *jpg* format) to a source folder of the project and then link it with the inline editor of the Properties View by selecting it. Do *never* link from the file system as this will *not* work in a productive environment.



Tip

When you work with a tabbed pane in the Visual Editor then the editor area will only show the currently selected tab. To change this (i.e. make another tab visible) simply select one of the sub components of the tabbed pane in the Java Beans view.

The indices of the tabbed components correspond to the order in which they are displayed in the Java Beans view, i.e. the first component has index `0` and the last one has index *(number of tabs) - 1*.

Symbol on Palette	
-------------------	--

Category	Container widget
Purpose	A container which arranges it's components behind each other, making them accessible with a tab.
Events	Focus, Key
Important Properties	selectedIndex, tabPlacement

Table 6.55. RTabbedPane Summary

RTable

To do:

What	Until when	Priority	Responsible
Buy food	21.6.2008		Peter
Mow lawn	12.6.2008		Eva
Pay phone bill	31.5.2008		Aline
Download Firefox 3	30.6.2008		Tamas

Table widgets are used to show tabular data.

A table generates a *List Selected Event* if a row is selected, an *Action Event* if a row is double clicked and a *Value Changed Event* if one of the RTable cell values has changed.

You can configure both the appearance and source of the data that a table should show on the Table Data tab of the Widget Configuration view. There you can specify a style configuration for each column cell as well as icon, text and tool tip for each column, based on the value of the cell or the whole row data. The widget config also allows you to specify the table columns as editable and/or sortable. The column entries are sorted according to the value they represent if you double click on the header of a column.



Note

The column value are sorted according to the value they represent. If the column is configured to show a Recordset field or a List entry, then the column is sorted against the original value from the Recordset or the List. E.g. if you transform the original data to a string in the Widget Configuration then the column sorting ignores the transformation for the sorting. If you do not specify any source for a column, then the column will be sorted according to the string representation of the cell value.

The Widget Configuration lets you define each column to be editable or not, by ivyScript. The edited values are stored in the data model directly (List or Recordset). There is no event at the moment, that signals a change in the RTable data.

The appearance of the table widget itself can be further configured by setting the properties `showGrid`, `showHorizontalLines` and `showVerticalLines`.

The data source for a table can either be a *List of Lists with equal length* or a *Recordset*. Depending on what is used, the properties `selectedListEntry`; `selectedListEntries` or `selectedRecord`; `selectedRecords` will return the currently selected list or record. The selection mode (*single selection*, *single interval selection* or *multiple selection*) can be set with the `selectionMode` property.

With the property `autoSelectFirstEntry` you can have the widget select the first data entry automatically (if data is present). This has also the effect, that a call to `clearSelection()` will reset the selection to the first data entry (instead of clearing). The property is set to *false* by default.



Note

Because tables tend to contain many items, it is recommended that you place table widgets inside an RScrollPane container, so that they become scrollable if the number of items becomes too large for the allocated view port size.

The headers, if any, are only displayed if the RTable resides inside a RScrollPane.



Tip

To process the *Value Changed* event, map the event to a Event Process Start element in your Rich Dialog logic. Enter the following script into the code tab of the Event Process start element:

```
import ch.ivyteam.ivy.richdialog.widgets.components.RTableValueChangedEvent;
RTableValueChangedEvent changedEvent = event as RTableValueChangedEvent;
int row = changedEvent.getFirstRow();
int column = changedEvent.getColumn();
```

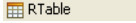
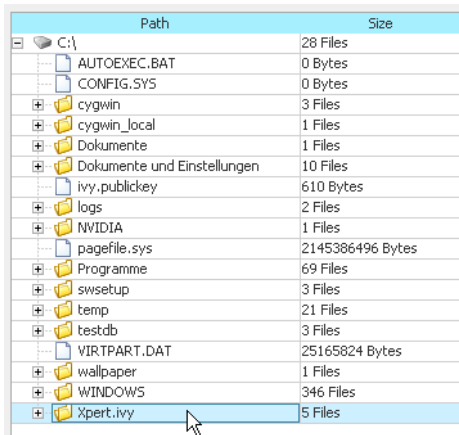
Symbol on Palette	 RTable
Category	Data widget
Purpose	A table can show a (large) amount of tabular data and allows the user to select one or multiple rows from the spread sheet.
Events	List Selected, Action, Focus, Key, Value Changed
Important Properties	listData, recordsetData, selectedListEntry, selectedListEntries, selectedRecord, selectedRecords, selectionMode, autoSelectFirstEntry

Table 6.56. RTable Summary

RTableTree



The RTableTree widget is a combination of the RTable and RTree widgets. It therefore offers the combined amount of events and configuration possibilities. Please consult the documentation of the mentioned widgets for a detailed explanation of the respective aspects.

Use the Widget Configuration TableTree data tab to configure the table tree widget style and data settings.

With the property `autoSelectFirstEntry` you can have the widget select the first data entry automatically (if data is present). This has also the effect, that a call to `clearSelection()` will reset the selection to the first data entry (instead of clearing). The property is set to *false* by default.



Note

Because table trees tend to contain many items, it is recommended that you place tree widgets inside an RScrollPane container, so that they become scrollable if the number of items becomes too large for the allocated view port size.

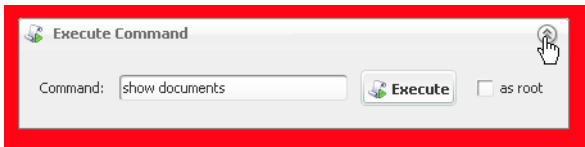
The headers, if any, are only displayed if the RTableTree resides inside a RScrollPane.

Symbol on Palette	 RTableTree
-------------------	--

Category	Data widget
Purpose	A table displays hierarchical tree data in a tabular fashion. Nodes/rows can be selected, expanded and collapsed.
Events	Tree Selection, Tree Load (configurable), Tree Expansion, List Selected, Action, Focus, Key
Important Properties	treeData, listData, recordsetData, selectedTreeNode, selectedTreeNodes, selectedListEntry, selectedListEntries, selectedRecord, selectedRecords, selectionMode, autoSelectFirstEntry, rootVisible

Table 6.57. RTableTree Summary

RTaskPane



A `RTaskPane` is essentially the same as a `RCollapsiblePane`, with the difference that it provides a title bar which consists of a title text, an optional icon and a collapse/expand button. Any kind of widget or container can be the content of a task pane on a task pane container.

Task panes are best used inside a `RTaskPaneContainer`, which is a parent container designed specifically to show task panes in a side bar below each other. The task pane container takes care of taking up the extra space if a pane is collapsed and shifts the other panes accordingly. Task pane container can also be configured to ensure that *at most a single task pane is open* at one time.

If you want to use a task pane outside a task pane container then you have to take care of correct re-laying yourself (i.e. provide other components that take up extra available space in case of collapsing). Consult the documentation of `RCollapsiblePane` for more information on this matter.

Use the Widget Configuration text/icon tab to set title and icon of a task pane. The state of a task pane (i.e. collapsed or expanded) can be defined by setting the `expanded` attribute.



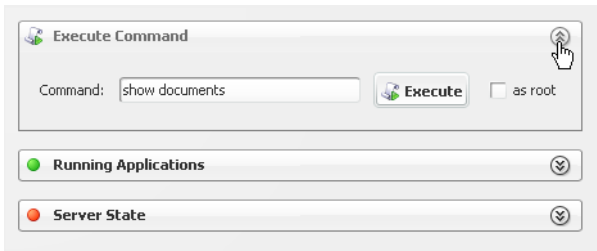
Tip

To initially add contents (e.g. an empty panel) to a task pane you simply drop the component either on the task pane's title bar (in the Visual Editor area) or onto the task pane node inside the Java Beans view. The task pane will then expand and show the added contents so that you can continue to edit it.

Symbol on Palette	
Category	Container widget
Purpose	A collapsible container with a title bar intended to be used inside a <code>RTaskPaneContainer</code> .
Events	Focus, Key
Important Properties	text, icon, expanded

Table 6.58. RTaskPane Summary

RTaskPaneContainer



A task pane container is a container for `RTaskPane` components only. The added task panes are arranged vertically below each other. The task pane container itself has only limited configuration possibilities: if you set `singleTaskPaneOpen` to `true`, then only one task pane can be opened at a time (i.e. the previously opened task pane will be closed before opening a new one).

Any kind of widget or container can be the content of a task pane on a task pane container. See `RTaskPane` to learn how to configure a single task pane.

The task pane container will show the contained task panes in exactly the state as they were configured in the Rich Dialog editor. I.e. all closed panes will be initially shown in collapsed state and all open panes will be initially shown in expanded state.



Tip

To add contents (e.g. an empty panel) to a previously added task pane you simply drop the component either on the task pane's title bar (in the Visual Editor area) or onto the task pane node inside the Java Beans view. The task pane will then expand and show the added contents so that you can continue to edit it. If you want to close an `RTaskPane` then you have to set its `expanded` property to `false`.


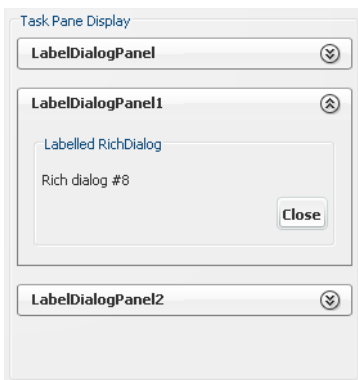
Symbol on Palette	
Category	Container widget
Purpose	A container for <code>RTaskPane</code> widgets.
Events	Focus, Key
Important Properties	<code>singleTaskPaneOpen</code>

Table 6.59. RTaskPaneContainer Summary

RTaskPaneDisplay



Displays are placeholder widgets onto which Rich Dialogs can be loaded dynamically at runtime. A *Task Pane Display* shows the loaded Rich Dialogs as individual *Task Panes* on a *Task Pane Container*. By setting the property `singleTaskPaneOpen` to true you can configure that at most one task pane should be open (i.e. the previously open task pane will be closed if another task pane is selected or opened).

A task pane display does not generate any non-standard events.



Warning

In order to being able to address a display widget (when loading a Rich Dialog with a Rich Dialog Process element) you must not forget to assign the display an *unique* id on the `displayId` property. If you fail to do so then the display might not be found at runtime and Rich Dialogs may not be loaded correctly.

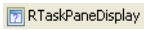
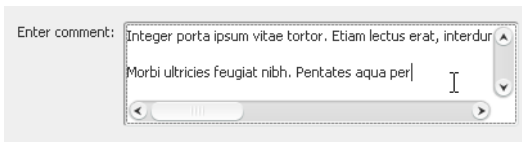
Symbol on Palette	
Category	Display widget
Purpose	Placeholder for Rich Dialogs that are loaded dynamically at runtime. Shows loaded components as task panes.
Events	Focus, Key
Important Properties	<code>displayId</code> , <code>singleTaskPaneOpen</code> , <code>toolTip</code>

Table 6.60. RTaskPaneDisplay Summary

RTextArea



Text areas are used to allow the user to enter multi-line texts. The text area generates a *Value Changed* event and a *Key* event every time a key is typed.

The *Value Changed* event is transmitted *synchronously* and the *Key* event is transmitted *asynchronously*. Since the synchronous event handling may lead to a blocking of the UI if the user types too fast, it is generally recommended to map the asynchronous *Key* event instead (will keep the UI responsive).

Use the attributes `lineWrap` and `wrapStyleWord` to enable/disable line wrapping and to enable/disable whole word wrapping, respectively. Obviously the setting second attribute has only an effect, if line wrapping is enabled.



Note

Although the text area allows for multi-line input, it will not automatically show scroll bars if the edited text grows out of the available space, either horizontally or vertically. Placing the text area widget inside a `RScrollPane` container resolves this issue. The scroll pane container allows you to configure individual policies for the horizontal and vertical scroll bars (show always, show if needed, never show).



Note

To append a complete line (including line break) to the end of a text area's text, you can use the method `appendLine(String)`.

Validation

The Widget Configuration view shows an additional Validation Tab if a text area is selected.

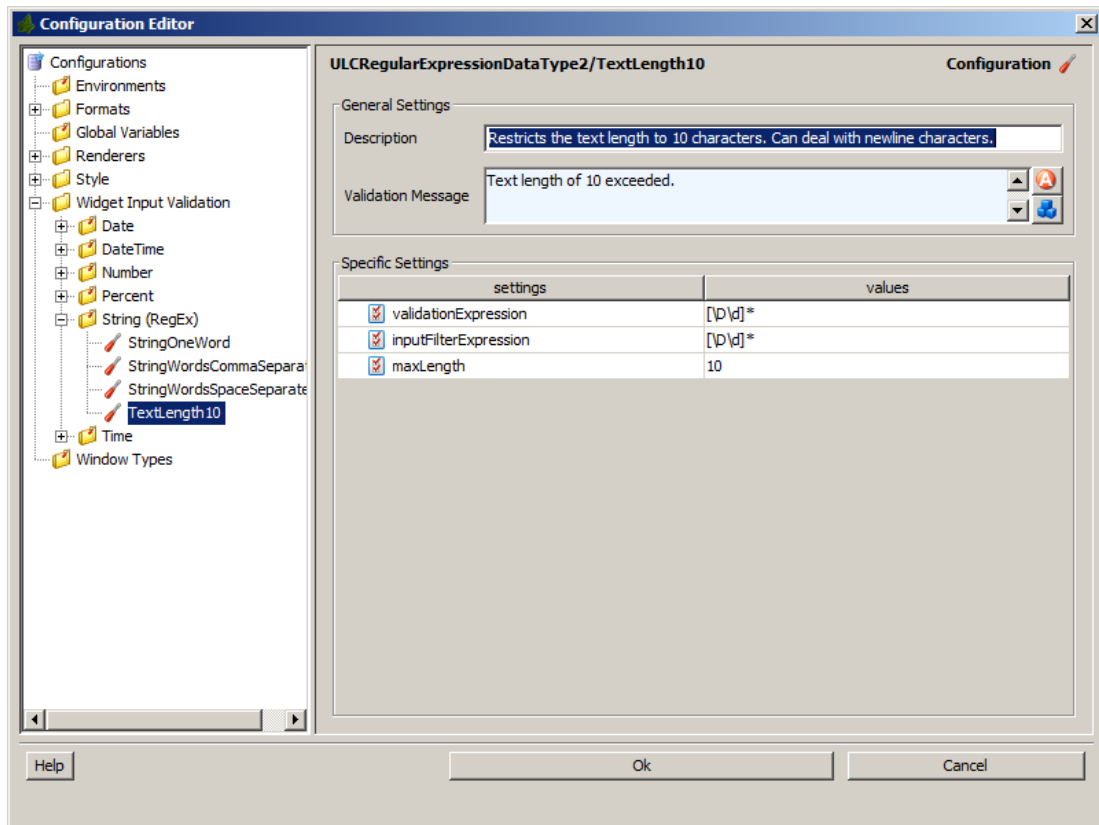
By selecting a specific validation configuration you can define the format of the input.

You can edit existing configurations or add new ones in the Widget Input Validation subtree of the Configuration Editor. You can specify *Number*, *Date*, *Percent*, *String/Regular Expression* based validations.



Tip

Most of the existing validation configurations do not support multiple lines input. Thus they will not suite ideally for a text area. Therefore, it is recommended to create regular expression based validation configurations specific to the text area. In order to support multiple lines, the existence of newline characters (`\r\n`) must be considered.



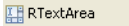
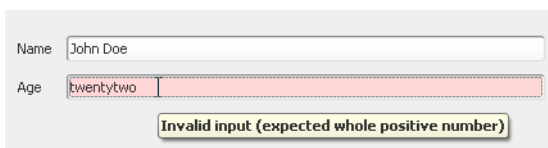
Symbol on Palette	
Category	Input widget
Purpose	An area where the user can enter a multi-line text.
Events	Key, Focus, Value Changed
Important Properties	text, lineWrap, wrapStyleWord, caretPosition, tooltip, font

Table 6.61. RTextArea Summary

RTextField



Text fields are used to get (single line) input from the user. The text field generates a *Value Changed* event and a *Key* event every time a key is typed.

The *Value Changed* event is transmitted *synchronously* and the *Key* event is transmitted *asynchronously*. Since the synchronous event handling may lead to a blocking of the UI if the user types too fast, it is generally recommended to map the asynchronous *Key* event instead (will keep the UI responsive).



Tip

The text that the user enters in a text field can be restricted and/or validated. If it does not match the required format, then an (automatic) visual feedback can be given to the user. Read the validation section below to learn how this can be done.

Validation

The Widget Configuration view shows an additional Validation Tab if a text field is selected.

By selecting a specific validation configuration you can define the format of the input.

You can edit existing configurations or add new ones in the Widget Input Validation subtree of the Configuration Editor. You can specify *Number*, *Date*, *Percent*, *String/Regular Expression* based validations.

Given that a valid format is ensured, you can access the value of a text field from the specialized attributes `valueAsNumber`, `valueAsDate` and `valueAsString`.

Note that the return value is a e.g. `Date.INVALID` if the user input does not represent a valid value.



Warning

Be aware that the accessing of a text field's value as number or as date will fail with an exception at runtime if the input entered by the user does not match the requested format. Therefore you should always enforce valid input with a proper *Validation Configuration* if you intend to access an entered date and number directly with the respective attribute.

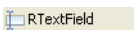
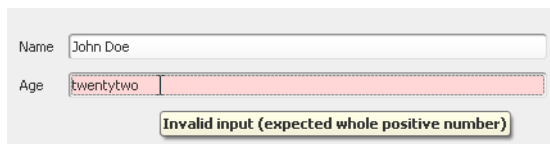
Symbol on Palette	
Category	Input widget
Purpose	A text field where the user can enter a string.
Events	Key, Focus, Value Changed
Important Properties	<code>text</code> , <code>valueAsDate</code> , <code>valueAsString</code> , <code>caretPosition</code> , <code>toolTip</code> , <code>font</code>

Table 6.62. RTextField Summary

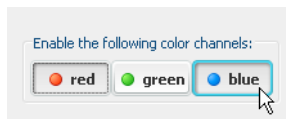
RTextFieldCellWidget




Text fields are used to get (single line) input from the user in a RTable.

For more informations about this widget see the documentation of RTextField. The possibilities and functionalities are quite the same.

RToggleButton



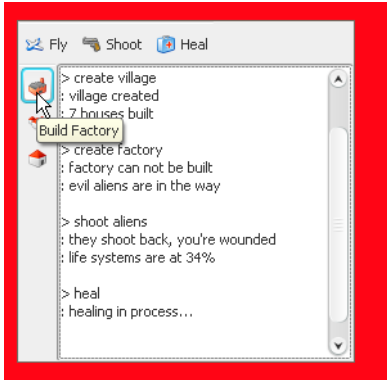
Toggle buttons are used to let the user visually enable an option in an on/off style. If a toggle button is clicked, it will stay in pushed state if it was previously unselected or vice versa. A toggle button will generate both an *Action* event and a *Value Changed* event if it is selected or deselected.

Symbol on Palette	
Category	Action / Selection widget
Purpose	A toggle button with text and/or icon that allows to select/deselect an option.

Events	Action, Focus, Key, Value Changed
Important Properties	text, toolTip, selected

Table 6.63. RToggleButton Summary

RToolBar



A tool bar lines up widgets in a dock. Although any kind of widget or container can be a component of a tool bar, typically buttons are used. The look and feel of a tool-bar is determined by the kind of components it contains, but generally it will remove any borders of the added contents to make them look smoothly integrated. When rolling the mouse over the contents of a tool bar then the currently pointed at component appears enhanced.

A tool bar can have a *horizontal* or *vertical* orientation which can be set with the `orientation` attribute.



Tip

By using buttons as components with or without text you can create tool bars with or without labels.


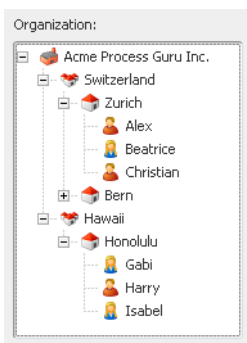
Symbol on Palette	
Category	Container widget
Purpose	A tool bar contains other widgets (typically buttons) and aligns them in a dock style either horizontally or vertically.
Events	Focus, Key
Important Properties	<code>orientation</code>

Table 6.64. RToolBar Summary

RTree



Tree widgets are used to display hierarchically organized data.

A tree generates a *Tree Selection* event if a node is selected and an *Action* event if a node is double clicked. To support the lazy loading of data, the generation of a *Tree Load* event can be configured in the Tree Data tab of the Widget Configuration view. Additionally the tree widget supports *Tree Expansion Events* that occur whenever a node is expanded or collapsed.

The data source of a tree is always a Tree object. The Widget Configuration Tree Data tab allows you to define the appearance and rendering of each tree node, which can be based on the tree's *value* object and/or on a separate *info* string. It possible to define style, text, icon and tool tip for each rendered tree label.

The appearance of the rendered tree can be further configured by setting the properties `rootVisible` (i.e. whether the root node should be displayed or not) and `showsRootHandles` (i.e. whether the root node should be displayed with an expand/collapse handle in front).

The tree widget supports multi-selection of nodes by default. The properties `selectedTreeNode;` `selectedTreeNodes` will return the currently selected node(s).

With the property `autoSelectFirstEntry` you can have the widget select the first data entry automatically (if data is present). This has also the effect, that a call to `clearSelection()` will reset the selection to the first data entry (instead of clearing). The property is set to *false* by default.



Note

Because trees tend to contain many items, it is recommended that you place tree widgets inside an `RScrollPane` container, so that they become scrollable if the number of items becomes too large for the allocated view port size.

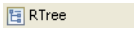
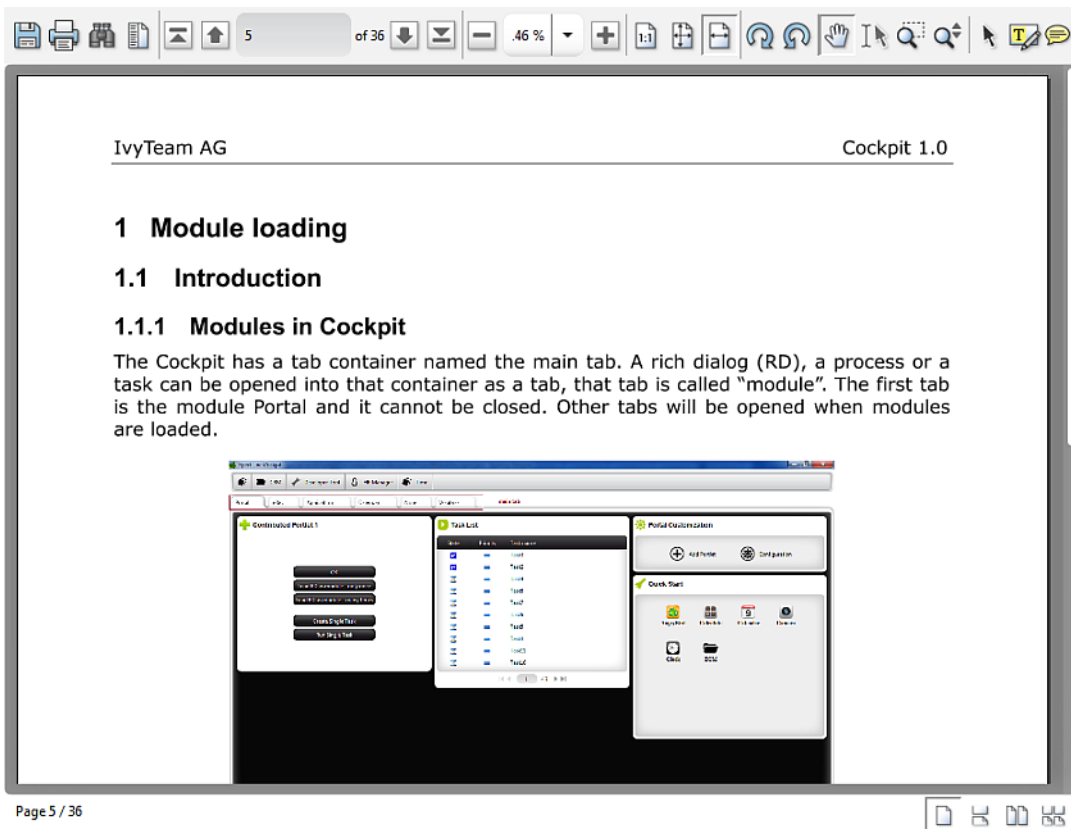
Symbol on Palette	
Category	Data widget
Purpose	A tree displays hierarchical data. Nodes can be selected, expanded and collapsed.
Events	Tree Selection, Tree Load (configurable), Tree Expansion, Action, Focus, Key
Important Properties	<code>treeData,</code> <code>selectedTreeNode,</code> <code>selectedTreeNodes,</code> <code>autoSelectFirstEntry,</code> <code>rootVisible</code>

Table 6.65. RTree Summary

PDF Viewer Widget



There are two ways to display PDF documents in a Rich Dialog based process. First you can open the PDF in the RBrowser widget. Internally the RBrowser widget then opens a real browser in a separate operating system process and in there the embedded PDF plugin (usually Acrobat Reader) will be used to display the PDF. This has the disadvantage that you have no real control about the PDF viewing, e.g. if the browser decides to open the PDF document in its own window there is nothing you can do about it. To avoid such problems, you can use the *ULCVdfViewer* widget which is purely Java based and does not use any external operating system processes. So, you can fully control where the PDF is displayed.

To add a PDF viewer widget to a Rich Dialog use the **Choose Bean** Choose Bean entry from the palette in the Visual Editor. Then choose the *ULCVdfViewer* and place it on the Rich Dialog. The widget must be used primarily with a Java API as there is only limited tool support in Axon.ivy, e.g. there is no Drag and Drop of events for the event mapping or there is no widget configuration view.

To open a PDF document, you simply use the method `openDocument (java.net.URL)`. The PDF viewer does support most of the actions that users expect from a PDF viewer like printing, zooming, searching or navigating the document through the built-in tool bar and status bar. If not required then you can switch off the tool bar and status bar with `setToolBarVisible(false)` and `setStatusbarVisible(false)`.

Developers can use the `setXyzToolBarVisible()` or `setXyzButtonVisible()` methods to define whether they want that their users can use the corresponding command groups or commands respectively. And end users can customize the tool bar groups and buttons themselves with a popup menu. All these settings are stored when a Rich Dialog is closed and restored at the next start. If you don't want to allow this to your users, then use the method `setToolBarPopupMenuVisible()`.

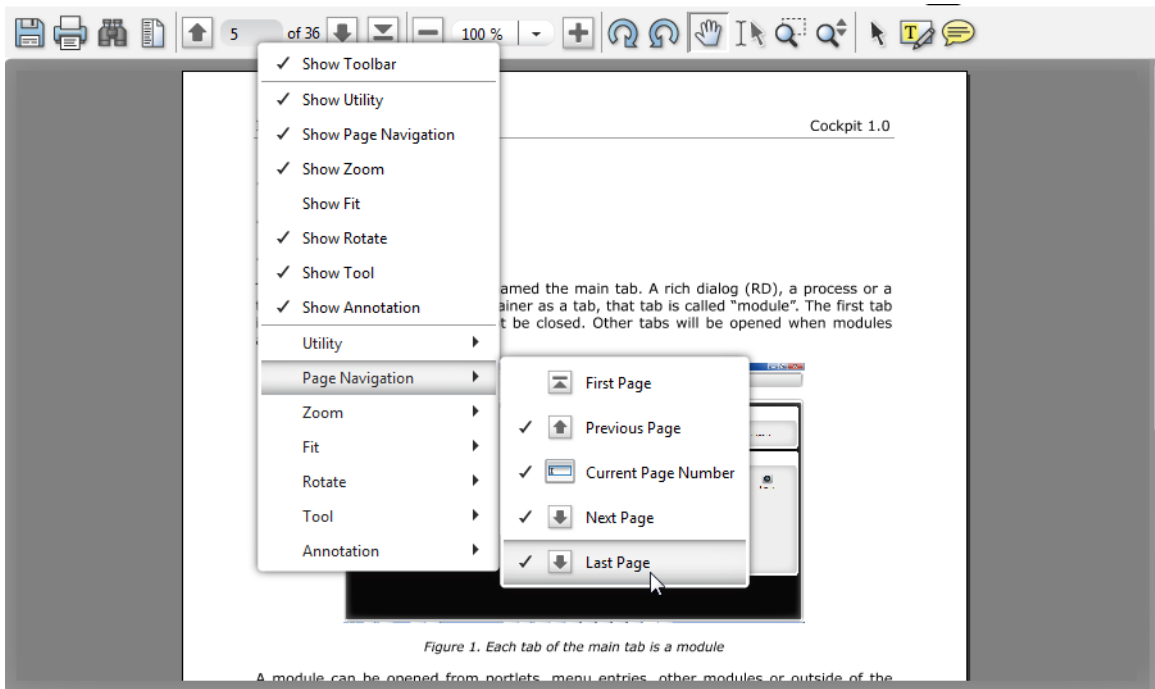
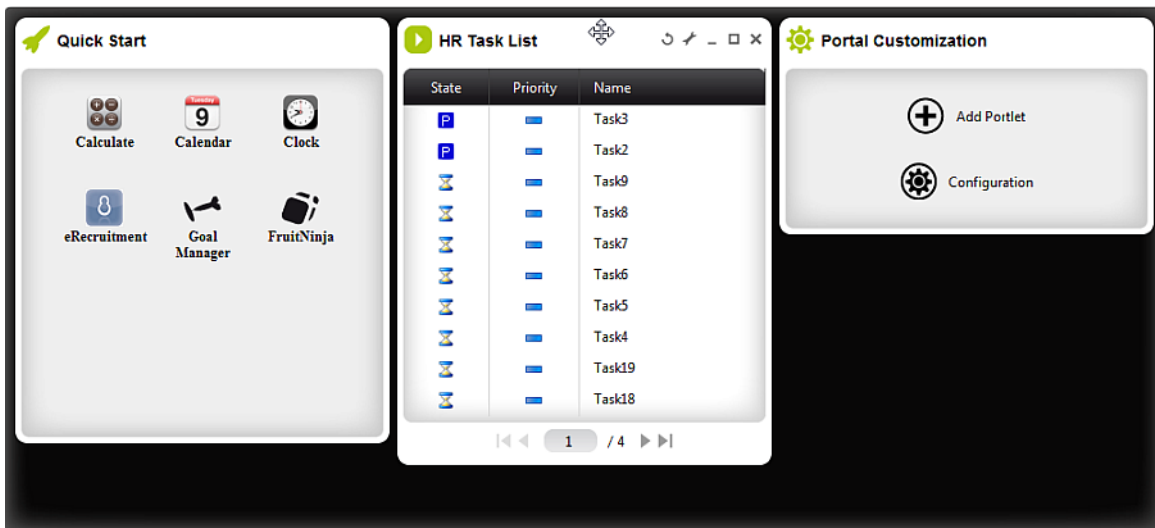


Figure 1. Each tab of the main tab is a module

Portal Display



With the portal display you can create your own portal with Rich Dialogs (from now on called *Portlets*). Use the User Dialog Call element to load your Portlets asynchronously into the display exactly like you do with other displays. The display then arranges them in a column layout where the number of columns and the column width ratio can be configured.



Note

There is only limited further tool support in Axon.ivy, e.g. there is no Drag and Drop of events for the event mapping or there is no widget configuration view. To add a portal display widget to a Rich Dialog use the **Choose Bean** Choose Bean entry from the palette in the Visual Editor. Then choose the LiquidPortalDisplay and place it on the Rich Dialog. Alternatively you can use the widgets by adding the corresponding code in the Java source of the Rich Dialog yourself. Most of the widget's properties can be set in the properties view of the Rich Dialog perspective. But to access the full API please write your code in the Rich Dialog source view file or in IvyScript on the Rich Dialog logic.

The display adds to every Portlet a title bar that offers some generic features. Each Portlet supports off-the-shelf closing, minimizing, maximizing and title renaming. All without adding a single line of code. Furthermore you can Drag and Drop

each Portlet in the portal and move it to a new position. The portal supports even two more operations, refresh and handling a configuration dialog for a Portlet. Let's see how these features can be used:

Refresh

If you want to use the refresh feature, two things must be done:

1. The corresponding button must be visible in the title bar. To do that add the Custom Panel Display Parameter *refreshable* with the value *true* in the Display tab of the User Dialog Call element.
2. The actual refresh must be implemented. To do that the portlet must have a Rich Dialog Method `doRefresh()` with no parameters and no return value. Do in this method whatever you need to do.

Configuration

If you want your Portlet to be configurable, then the portal offers some nice facilities to make life easier for you. Four steps have to be fulfilled:

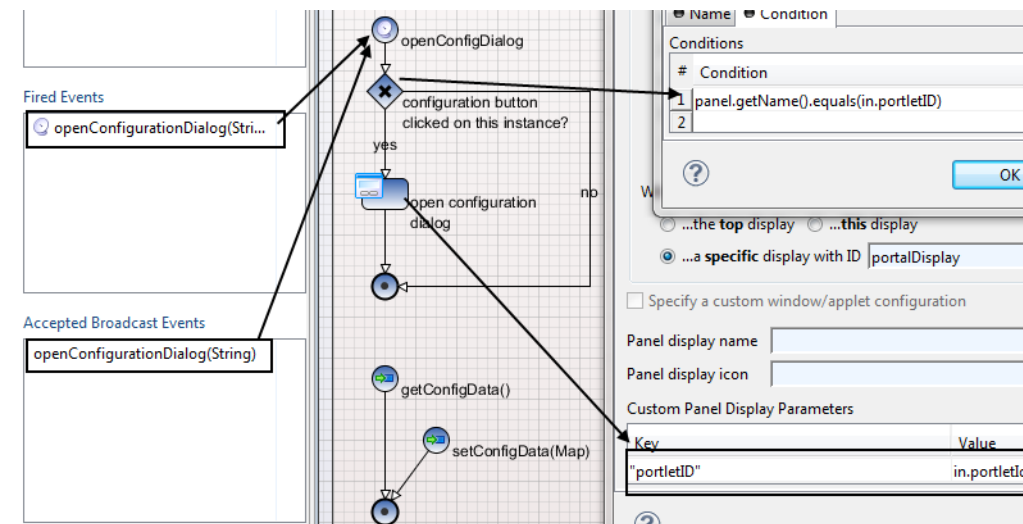
1. Configuration button in the title bar:

To see that button, just add the Custom Panel Display Parameter *configurable* with the value *true* in the Display tab of the User Dialog Call element that loads the Portlet into the display.

2. Opening the configuration dialog::

As soon as the user clicks on the configuration, the portal fires a Rich Dialog Event with the signature `openConfigurationDialog(String)` from the Portlet. So, you need to add the event to the Rich Dialog Interface of the Portlet Rich Dialog. The `String` parameter carries a unique ID for the Portlet. The internal name of the Portlet is used for that. See below for details.

To catch the event, add a Broadcast Event to the Portlet Rich Dialog. In the implementation of the Broadcast you should first find out whether the configuration dialog should be opened for the correct instance of the Portlet because the same Portlet can exist multiple times in the display. Here the Portlet ID comes in very handy. Use it to compare to `panel.getName()`. The second thing to do in the Broadcast implementation is to open the configuration dialog in a Rich Dialog call element. Load your configuration dialog asynchronously into the saem window and display as your Portlet. The only special thing you have to do, is to provide the Custom Panel Display Parameter *portletID* with the Portlet ID.



3. Exchanging the configuration data:

The Portlet and the configuration dialog must have some shared data. That's the configuration data that is used to retrieve or render the content of the Portlet. That's also the same data that is manipulated in the configuration dialog. The portal uses simple getter and setter Rich Dialog Methods to achieve that.

- a. **From Portlet to the configuration dialog:** First the portal calls the Rich Dialog Method `getConfigData():java.util.Map` on the Portlet. Hereby you are completely free what to put in that map. In a second step the portal then takes this map and calls the Rich Dialog Method `setConfigData(java.util.Map)` on the configuration dialog so that it can work with the map.
- b. **From the configuration dialog to the Portlet:** When the user hits *Ok* on the configuration dialog, then the reverse thing happens. First the portal gets the interesting data by calling the Rich Dialog Method `getConfigData():java.util.Map` on the configuration dialog. Then this is given to the Portlet by calling the Rich Dialog Method `setConfigData(java.util.Map)`.

4. Refreshing:

To reflect the changes from the configuration, the portal automatically calls the refresh feature (see above) if the Portlet supports it. If not it is your turn to make sure that a configuration change takes effect.

Web Page

This chapter shows how Web Pages and -content are used within Axon.ivy.

Using Web Pages (web content) in a Business Process

As an alternative to User Dialogs, you can display Web Pages or other web resources to the user of an Axon.ivy process application in a Browser to let her interact with the executed process.

Such content is embedded within a business process by using the Web Page element or by using end pages for various elements in a process (End Page element, a Simple Task Switch element or a Task Switch element). Whenever the process reaches such an element, then the defined content is presented to the user.

You are free to use HTML-based content (plain HTML, JSP) or other resources (such as images, text files, RTF and many more) as long as they can be displayed in a web browser. In addition, you are free to use resources from the CMS or from the web content folder of the project.



Warning

When you re-use resources like JSP pages in different process elements, you need to carefully consider how to integrate the content with the process. When you access members of the data class (e.g. in the JSP) you must ensure that this member is accessible in all processes that use the resource. Furthermore, in Web Page elements you need to ensure that a) there is a way to continue with the process, e.g. an out-link and that b) that way can be used in all usages of the resource.



Warning

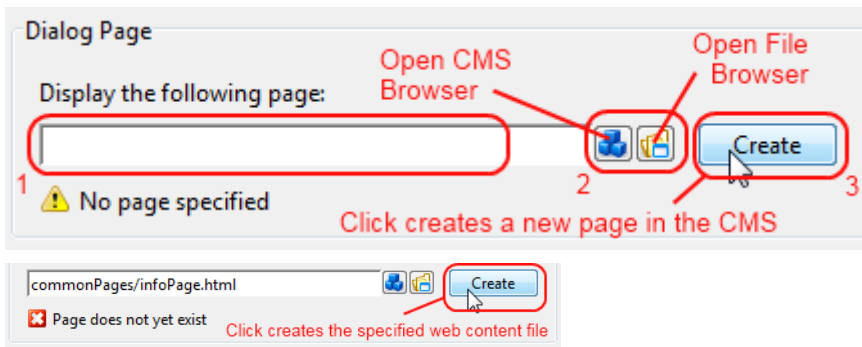
Please bear in mind that Web Pages can only be used if a single HTTP request is available from the client. This means, that you *cannot* use Web Pages if at least one of the following conditions apply:

- The current request was not triggered by HTTP (e.g. started by an Event Bean).
- The current task is executed by SYSTEM.
- The process is running in a parallel execution section, e.g. after a *Split* element.

Creating and Editing Web Pages from within the Process

A Web Page can be created or accessed directly from the process. Open the inscription mask of any page-capable process element. Depending on whether the page is already defined or not, you will be presented with either of the two scenarios described in the sections below.

No page has been specified yet



You can define the Web Page that should be displayed by using one of the following three methods:

1. Enter the path to an existing page by hand. Specify either a CMS path (e.g. `/HtmlPages/myPage`) that points to a content object or give a path to a web content file (e.g. `commonPages/infoPage.html`) instead.

Note that content object paths do not have a file extension, but web content paths do. Web content paths are always specified relative to the `webContent` folder of the current project.

If you enter a path to a non-existing web content resource, then pressing the *Create* button will create an empty file at the specified location in the `webContent` folder and open Eclipse's default editor on it.

If you enter the name of a non-existing content page, then pressing the *Create* button will have the same effect as described under (3).

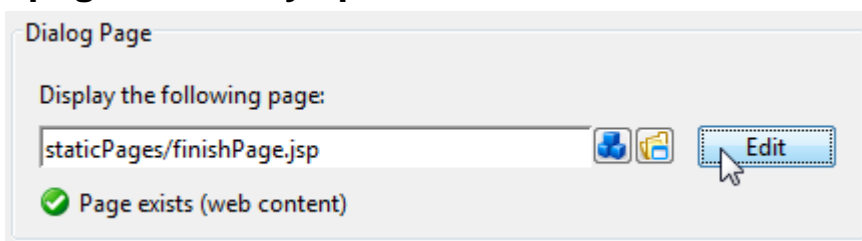
2. Select an existing content object by using the *content smart button* or an existing web content file by using the *file smart button*.

You can select any content object and any file, but a "wrong type" error will be displayed if the selected content object is not suitable as a page. Likewise a "invalid web content path" error will be shown, if you select a file outside the project's web content folder.

3. Click *Create* to generate an entirely new page in the content management system.

A dialog will appear that allows you to enter the name and type (normal or JSP) of the new page. The created page will be associated with the current element and it will be placed appropriately inside the CMS `ProcessPages/<ProcessName>` folder (see below).

A page is already specified



Click on *Edit* to open the specified page either in the content editor (if it is a content object) or the system's default editor (if it is a web content file) so that you can edit its contents. You can change the default editor for any file type by opening *Window/Preferences* and navigating there to `/General/Editors/File Associations`.

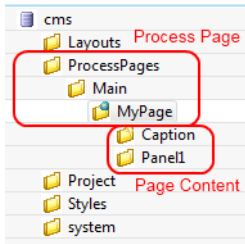
Alternatively you can use the *content smart button* or the *file smart button* to select an entirely different page to be displayed.

You can also edit the specified path by hand if you like.

Where Web Pages are stored

The page that is displayed is either stored in the CMS or in the web content folder of the project.

By default, CMS pages are stored in the `ProcessPages` hierarchy in a further sub folder named after the containing process (e.g. `ProcessPages/Main/MyPage`). Although it is not required to store the pages within this scheme, it is recommended to keep your pages separated. All content objects that can be downloaded can be used, especially the Page and JSP types, but also some Document (e.g. RTF, HTML, PDF or Plain Text) and other types are working.



Pages in the web content folder can be stored in any hierarchy below the project's web content folder (it is not allowed to use or reference content that is stored outside the project's web content folder). You can use any type as long as it is possible to render it in the browser of the user.



Note

Some browsers delegate the displaying of certain file types to third party plugins and/or applications depending on the configuration of the client. Thus this behaviour cannot be controlled by Ivy.

HTML content in the CMS

There are a number of specific content object types that are uniquely used inside HTML Dialog pages. All of them have their own editors, the usage of which is described in the following sections.



Note

Web Pages can be accessed without starting a process. This allows you to create for example a translated start page with some process start links. See also chapter Access CMS Content with a Browser.

Web Page Editor

English*

Last changed by mda on 15 Jun 2010 at 16:47:23.

Page Style Sheet

Inherit style from parent /Styles/Xpert

Volatile: /Styles/Blue Search ...

Page Layout

Inherit layout from parent /Layouts/1PanelWithHeaderFooter

Volatile: /Layouts/2horizontalPanels Search ...

Page Layout Actions

Layout /Layouts/2horizontalPanels parsed successful.

Warning: Layout needs a 'Panel2' child.

Delete children with conflicts
Delete unused children
Create required children

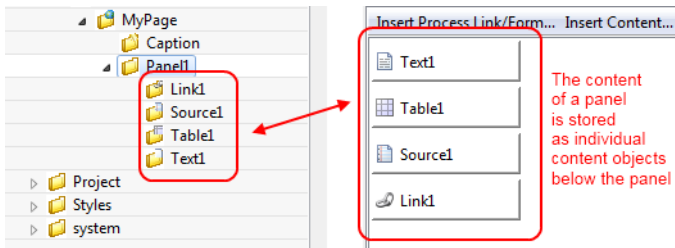
The Page Editor is used to specify the *Style Sheet* and *Layout* of a page. They can either be inherited from their parent objects in the CMS hierarchy or can be set explicitly. Use the combo boxes to set one of the style sheets, i.e. page layouts which are contained in the corresponding CMS top level folders. The *Search...* buttons on the other hand opens a dialog which does show all *CSS* and *Layout* content objects in the whole CMS to select one. So the children content objects of the page represent the selected *Layout* (see Layout Editor). The missing or obsolete parts (i.e. children in the CMS hierarchy) of a page (e.g. after changing layout) may be created or deleted with the corresponding buttons at the bottom of the editor.

HTML Panel Editor

Overview

This is the most important editor to edit your HTML content. Each Web Page (see Page Editor) has sections that are called *Panels* and which are defined by the page's layout. Generally, a page has as many panels as are specified by it's associated layout (see Layout Editor).

In the panels of a page, the actual content of the page is defined. Each content part of a panel is stored as an own content object below that panel's content object inside the content management:



The contents of a panel are arranged in a table from left to right and from top-to bottom in a structured manner. Each table cell contains a single content object or some custom HTML or JSP code.

A panel can be rendered as plain HTML or as a HTML table. If it is rendered with plain HTML cells are simply rendered one after another. Rows are separated with a `
` tag. On the other hand, if it is rendered as a HTML table, each cell will be rendered as a cell of the table. You can switch the rendering kind using the checkbox *As HTML Table*. Switching the rendering influences the available context menu items of a cell and the buttons *Table*, *Row* and *Cell*.

You can switch between the *Edit* and the *Source* View of a panel. The *Edit* View as explained above shows the table of the panel with its cells. The *Source* View on the other hand shows the JSP Code that is generated from the *Edit* View.

Menus

Insert Process Link/Form ...	Use this menu to insert a link or form that triggers the continuation of the process after the page element.
Insert Content ...	Use this menu to create sub content objects (Texts, Strings, Sources, Process Links, Tables, Result Tables, Smart Tables, Images) and insert them or already existing content objects into the panel.
Insert Attribute	Use this menu to insert process attribute values into the panel. The menu will open an Attribute Browser where you can choose the process attribute to insert. You can specify a condition that gets evaluated during the rendering of the panel to control whether the content of the cell is rendered into the panel or not. Moreover you can specify how the process data value should be formatted.
Insert JSP	Use this menu to insert JSP code into the panel.

Context Menus

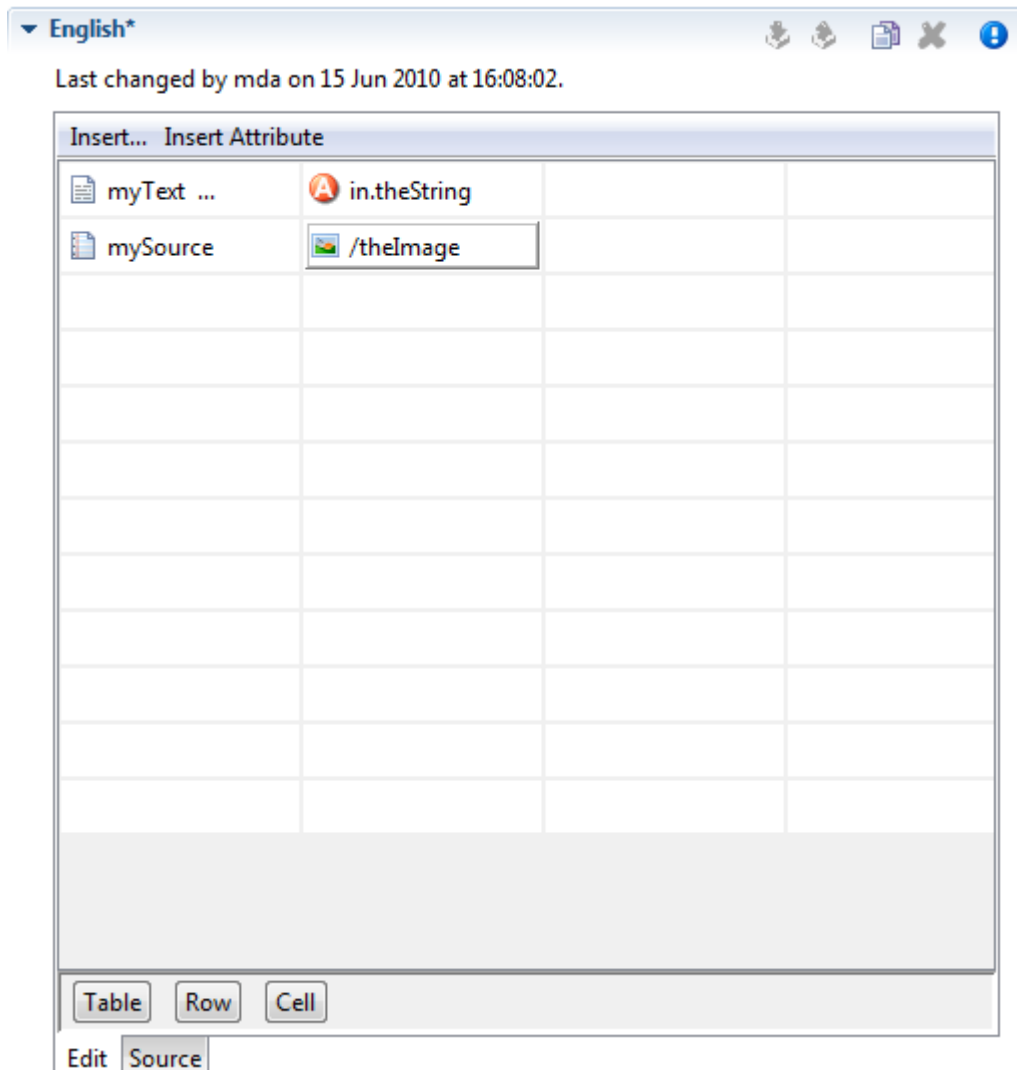
Edit	Opens the type specific editor (Attribute, Content, JSP, ...) of the selected cell.
Edit Condition	Opens an IvyScript editor to edit the condition that controls whether the content of the cell is rendered or not.
Move	Use this menu to move the current cell to another location.
Remove	Removes the selected cell.
HTML Attributes <...> Table	Opens an HTML attribute editor to edit the HTML attributes of the table.
HTML Attributes <...> Row	Opens an HTML attribute editor to edit the HTML attribute of the currently selected table row.
HTML Attributes <...> Cell	Opens an HTML attribute editor to edit the HTML attributes of the currently selected table cell.
Insert Row	Inserts a row above the selected cell.
Delete Row	Deletes the row of the selected cell.
Merge Cell Left	Merges the selected cell with the cell at the left side.

Merge Cell Right	Merges the selected cell with the cell at the right side.
Merge Cell Up	Merges the selected cell with the cell at the top.
Merge Cell Down	Merges the selected cell with the cell at the bottom.
Split Cell into Rows	Splits the selected cell into multiple cells one cell for each row.
Split Cell into Columns	Splits the selected cell into multiple cells one cell for each column.

Buttons and Check boxes

As HTML Table	Use this checkbox to switch the rendering kind of the panel from raw HTML to HTML table and vice versa.
Table	Use this button to open an HTML attribute editor to edit the HTML attributes of the table.
Row	Use this button to open an HTML attribute editor to edit the HTML attribute of the currently selected row.
Cell	Use this button to open an HTML attribute editor to edit the HTML attribute of the currently selected cell.

HTML Table Editor



The *HTML Table* editor can be used to configure a HTML table. In contrary to the *Result Table* the numbers of rows and columns is defined here statically. And for each cell the content must be chosen specifically although the content itself may be generated dynamically at run-time. Using the menu *Insert...* it is possible to use *Text*, *Source* or *CMS* elements and using the menu *Insert Attribute* it is possible to use attributes from the process data in the cell content.



Tip

A *Text* element can be inserted in a cell by just selecting the cell and starting to type the text. The Text Editor opens automatically then.

With the buttons *Table*, *Row* and *Cell* or the corresponding entries in the popup menu the HTML attributes for the corresponding table part can be manipulated. Furthermore you can add/delete table rows and merge or split table cells to influence the layout of the table. The source tab allows to view and edit the generated HTML code directly.

Typically a table object is not created as an independent content object in the CMS tree, but rather inserted below a *Page* content object. The cell contents are no first-class CMS elements but are stored within the HTML table content object itself.

Result Table Editor

The *Result Table* editor is used to configure the dynamic generation of (HTML) data tables on either *DB page* or *Page* process elements. Typically a result table object is not created as an independent content object in the CMS tree, but rather inserted below a *Page* content object when a table gets inserted onto one of the pages panels (using the Panel Editor).

Data Source	Specify here the source of the result table's data. You can select a <i>recordset</i> or a <i>list</i> as data source from the process data attributes.
Visible Columns	Select the columns that should be rendered by the result table. You can either use the column numbers (1,5,8,2,3) or column names (name, first_name, customer_id) in a comma-separated list to specify subset and order.
Selection Links	Hers it is possible to select which column entries that should be turned into a link for the one of the page's output links. The <i>select</i> entry specifies what value that should be assigned to which <i>result value</i> (a process attribute). This is used to identify the selected record on the next process element.
Table Caption	A caption text for the rendered result table. Macros may be used.
Column Headers	Specify the names of the columns as they should appear in the header row as a comma-separated list. Macros may be used, the order must be the same as specified in the <i>Visible Columns</i> field.
Automatic Headers	If selected, then the column headers will be selected from the specification in the database.
Empty Table Text	The text to display if the source data is empty. May contain macros.
HTML Attributes	Specify the HTML attribute values for <i>table</i> , <i>even rows</i> , <i>odd rows</i> , <i>header cells</i> and <i>column cells</i> .

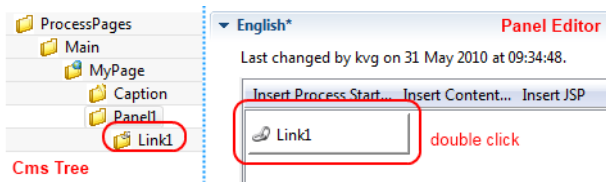
For almost all of the above described features, the attributes of the corresponding HTML tags can be edited explicitly by clicking on the *HTML tag button*.

HTML Link Editor

Link editor is used to render process and static links. Whereas the execution of a process link will lead to the continuation of the process (i.e. activate a specific arc in the process model) and the activation of a static link leads to some other HTML page independent of the process.

Links can be rendered in many different forms, e.g. both the "classical" hyperlink as well as the display of a form with a submit button are considered to be special forms of links.

All of those different link types are configured with the HTML Link Editor. The link editor can be opened in two different ways: either by selecting a link content object below a page in the CMS editor tree or by double clicking on a link field in the HTML Panel Editor.



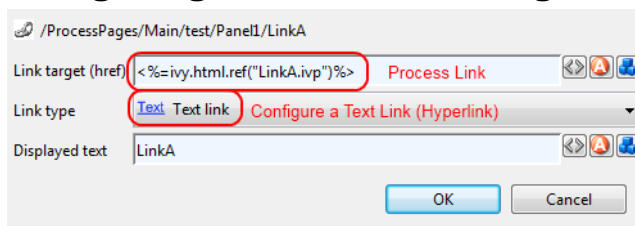
The following options can be configured for all links:

Link Target (Href) Specify the target of the link that will be jumped when the link is activated. If this is the configuration of a process link, then a macro that specifies the link that should be executed will already be specified (see example image above). However, any valid HTTP-link is accepted as value for this field, e.g. `http://www.acme.com`.

Link Type Specifies the visual appearance (type) of link that should be configured. Depending on the type that is selected, the configuration fields below this attribute will change.

Depending on the selected *Link Type*, additional options are available for configuration.

Configuring Text, Button, Image and Timed Auto-Redirect Links



For *Text*, *Button*, *Image* and *Timed Auto-Redirect*, the Link Editor allows for configuration of rather simple additional options. The smart buttons at the right side of the macro fields can be used to either configure the HTML attributes of the rendered link source or to insert and expand process attributes and content objects.

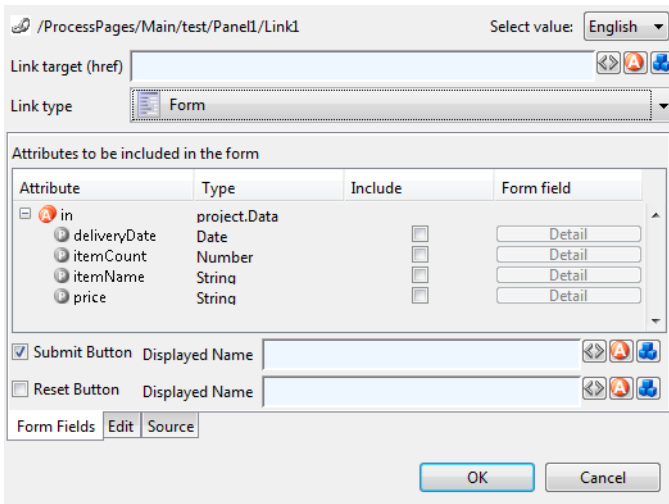
The following options can be configured for those link types (configuration of a Form Link is described in the next section):

Displayed Text Specifies the text that will be shown to the user and presented as link. This option is available for the link types *Text*, *Button* and *Image + Text*.

Alternate Text Specifies the alternate text that will be set on the displayed image (is shown by most browsers as tool-tip when the mouse hovers over the image). This option is only available for the link type *Image*.

Time to go in Seconds Specifies number of seconds to wait before the link is activated automatically. This option is only available for the link type *Auto-Redirect (timed)*.

Configuring a Form Link

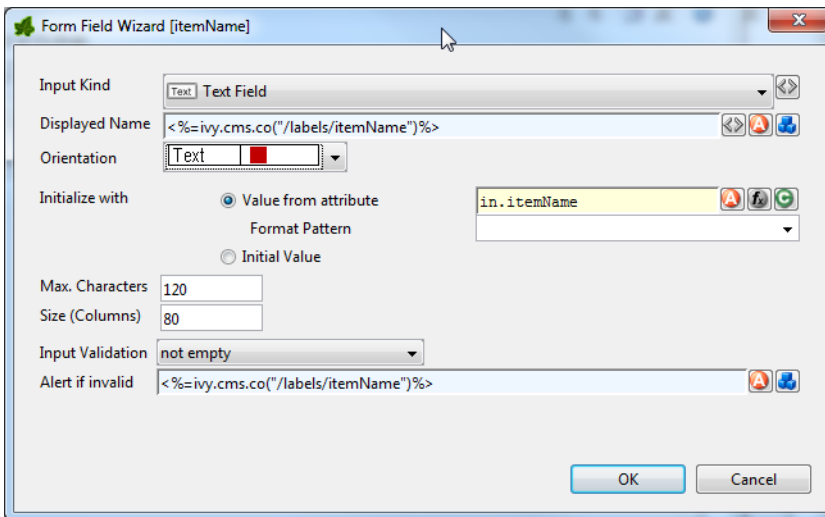


When selecting *Form* as *Link Type* the link editor will show a sub-editor that allows you to specify a form and all form fields, that are initialized and/or set from/to process data. Furthermore, you can configure whether a *Submit* and/or a *Reset* button should be displayed (including the text of the buttons). At the bottom you even have the choice to change the editor's view for either designing the tabular layout of the form in a graphical editor or directly editing the HTML code.

At run-time the form defined in the *Form Field Editor* will be rendered in a HTML page. Clicking the submit button results in the continuation of the process of this element. Therefore, if you disable the submit button you have to ensure that a process link for process continuation is part of the HTML page (e.g. by adding such a link in the HTML view of the editor).

Configuring Form Field Details










If the Details button of a form field is clicked (when configuring a Form Link) a specialized sub-editor is presented to the user. This editor allows to configure in detail how a specific form field is to be rendered in HTML.



Several types of input kinds for the form fields are available such as text fields, lists or date pickers. For each type, the configuration and therefore the layout of this editor may differ slightly and some parts (irrelevant to the chosen) may be invisible. For most form fields you can define the layout, the default value and the validation.


Input Kind

Input Kind	Image	Description	HTML Tag
Text Field		A single line input field for text of all kind.	<input type="text">

Input Kind	Image	Description	HTML Tag
Password Field		The same as the text field but each inserted character is displayed as black bullet.	<input type="password">
Text Area		This is a multi line input field for text of all kind.	<textarea>
Check Box		A simple box to selecting zero, one or more of two or more options (ideally for yes/no or true/false information). The data that is associated with the chosen option is saved.	<input type="checkbox">
Radio Button		It is sort of a button for selecting one and only one of two or more options which are mutually exclusive. The data that is associated with the chosen option is saved.	<input type="radio">
Combo Box		A text field with an attached drop-down list to select from several predefined values.	<select size="0">
List		A list to choose from several predefined values.	<select size="nbOfOptions">
File Upload		With this input kind, the user can choose a file to upload to the Axon.ivy Engine file area.	<input type="file">
Hidden Field		Hidden fields can be used to transfer additional data invisible for the user (for use in e.g. JavaScript or e-mail transfer of forms)	<input type="hidden">
Date Picker		To choose a date. Note that the date is displayed according to the language	Implemented in Java Script

Input Kind	Image	Description	HTML Tag
		set in the browser of the user (while simulating in the Designer you can set the language with a toolbar button). Note too, that if dates are entered in a 2-digit format, then Axon.ivy will interpret numbers within the next 20 years as years in the future. All other numbers are interpreted as years from the past (e.g. in 2010 entering 30 leads to 2030 whereas entering 35 leads to 1935). The Javascripts used for the Date Picker are copied into the <i>webContent</i> folder of the project to <i>scripts/datepicker</i> and <i>scripts/jquery</i> .	

Table 6.66. All the different input kinds

Displayed Name	Axon.ivy adds a text label to the form field for reference. The text of this label can be set here using process data or CMS entries.
Orientation	It is possible to choose how the label and the form field are positioned to each other, either side by side or super-imposed.
Initialize with...	You can use process data, function return values (both <i>value from attribute</i>) or plain text (<i>initial value</i>) to set a default value for the form field. For Check Boxes it is additionally possible to select whether it is checked or not.
	<div style="display: flex; align-items: center;">  <div> <p>Warning</p> <p>Make sure that the data types of the default value matches to the chosen form field otherwise an error will be thrown at runtime.</p> </div> </div>
Maximum Characters	The maximum number of characters which are allowed to enter. The user will not be able to enter more characters.
Size (Columns)	You can set the width of the form field to layout your forms nicely.
Options	Here the content of the list based form fields (Combo Box, List, Check Box, Radio Button) can be defined. You can either choose an Ivy Script recordset or list (<i>select options form attribute</i>) or you can define the content in a table with values and display texts. For recordset based content, the first column is used as values and the second column as display texts (all other columns are omitted). For list based content all the list entries are used as value and as display names.

Values per Row	For check boxes and radio buttons it is possible to define how many options are rendered in the same line (or row)
Rows	Only used for the list form field. It represents the number of visible rows in the list. If more items than rows are defined, the list will have scroll bars.
Mime Type	Is only used for the file upload form field. The mime type corresponding to the type of file which can be uploaded.
Input Validation	<p>By selecting an input validation script you may validate the values that an user enters in a specific field on the client, e.g. before the form is submitted. If the user tries to submit a form but input validation fails on some of the form fields, then a message will be displayed to the user and the form will not be sent.</p> <p>Axon.ivy offers you a number of built in validation functions that can be selected from the <i>Input Validation</i> combo box. The scripts that are available depend on the input kind of the currently edited form field (e.g. different scripts are available for checkbox and text input fields).</p> <p>If the built-in scripts do not satisfy your needs, you may provide your own validation scripts written in JavaScript. Read the section <i>How to provide own Validation Scripts</i> to learn more about this topic.</p>



Warning

Please note, that *JavaScript* must be enabled in your client's browsers in order for the validation scripts to work!

Alert message if invalid	<p>For any selected Input Validation a predefined warning message exists. If you don't like the default message, you may specify a custom message that will be displayed if the entered value is invalid.</p> <p>You may use the content smart button to select and insert a textual content object that should be displayed as message or as part of your message. This allows you to specify messages in multiple languages.</p> <p>If you leave the field empty, the default message of the selected validation script will be displayed.</p>
--------------------------	--

Smart Table Content Editor

Overview

On the Smart Table Content editor you configure what content to show in the smart table. You can configure the data source for the table. Furthermore, you can influence the visual look and feel and set options such as header, body or footer label texts, for each entry in the table.

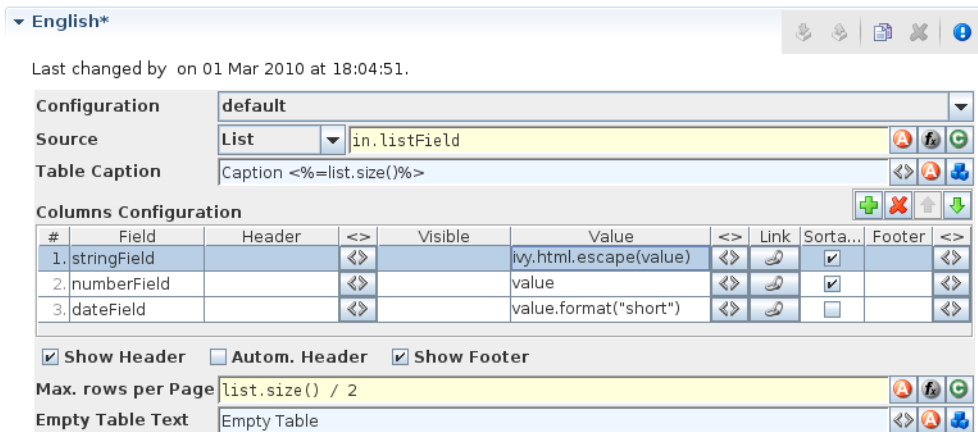


Figure 6.23. The Smart Table Content Editor

Accessibility

Content Management -> Smart Table Content Object

Features

- | | |
|------------------------|---|
| Configuration | Select the Renderer Configuration which is used when the Smart Table is rendered and then displayed on the client browser. See the Smart Table Configuration for more details. |
| Source | Select the type of the source for your Table. Either List, Recordset or Record. Define in the IvyScript text field the path to your source. |
| Table Caption | Enter a text to display as caption of the table. Use IvyScript macros if needed. example: <code><%=ivy.cms.co("/labels/header")%></code> |
| Columns Configuration: | <p>Columns (a column configuration is shown as row in the editor)</p> <p>Field: Define the name of the field of your data source to map to the current column. If your data source is a list, then the list elements must contain a field or method with the configured name (e.g. <code>userName</code> or <code>getUserName()</code>). If your data source is a recordset, then the records must have a field with the configured name. The value of the field is mapped to the variable <code>value</code> which you can use in the IvyScript text field of the Value field.</p> <p>Header: Enter a text to display as header of the column. Use IvyScript macros if needed, e.g. <code><%=ivy.cms.co("/labels/header")%></code>. Note that this has only an effect if the Show Header check box is selected too.</p> <p>Visible: Enter a IvyScript expression which results in a Boolean value, to configure the visibility of the column. If no condition is set, then the column is visible.</p> <p>Value: Enter a IvyScript expression to configure what label that should be displayed in the table cell. This IvyScript is executed for every cell in this column (once per row in the table).</p> |

Link: Choose the link where the process goes if we click on a cell on this column. It is also possible to define the process attributes where to store the row index, the column index or the column name.

Sortable: Activate this check box if you want your table to be sortable. Clicking on the header of a column triggers then a sorting of the whole table according to the selected column. The variable `value` is used for sorting which is defined in the column **Field**.

Footer: Enter a IvyScript expression to configure what label that should be displayed in the table footer.

<>: Allowed to edit the HTML attributes for the table header, body or footer cells. E.g. choose colors or borders. Note that this specific settings of HTML attributes will override any settings from the configuration.

Actions **Add:** Adds a new column configuration to your table definition. The column configuration will be shown as row in the Columns Configuration table.

Remove: Removes the selected Column Configuration(s) from your Columns Configuration table.

Up: Moves the selected Columns Configuration(s) up in the table. This means that the column(s) moves one position to the left in the resulting table.

Down: Moves the selected Columns Configuration(s) down in the table. This means that the column(s) moves one position to the right in the resulting table.

Show Header: Activate this check box if you want your table to have a header row.

Autom. Header: Activate this check box if you want the system to label your headers.

Show Footer: Activate this check box if you want your table to have a footer row.

Variables **in:** The process data at the point of time when this CMS element is executed.

index: Table index of the currently selected table entry.

originalIndex: Index of the currently selected entry in the data source (without respect to the visual order, e.g. after a sorting).

value: The value of the currently selected field (as defined in the *field* column) in the data source element (i.e. record).

it: The currently rendered element of this widget's list data source (only available, if the source is of type List).

list: The list defined as data source (only available, if the source is of type List).

record: The currently selected record (only available, if the source is of type Recordset).

recordset: The recordset defined as data source (only available, if the source is of type Recordset).

Max. rows per Page Enter a IvyScript expression to configure the maximum number of rows are shown per page.

Empty Table Text Enter a plain text that will be displayed if the table source is empty.

JSP Editor

The JSP editor is used to edit a JSP content object which can be used in the Web Page, End Page, Task Switch or Simple Task Switch process elements.

Inside JSP content objects you can make use of the Environment Variable *ivy*. It is imported and declared as follows:

```
<%@ page import="ch.ivyteam.ivy.page.engine.jsp.IvyJSP"%>
<jsp:useBean id="ivy" class="ch.ivyteam.ivy.page.engine.jsp.IvyJSP" scope="session"/>
```

You can also use the *in* object (i.e. process data) of the process where the associated process element is located. You can access the process data by using the `ivy.html.get()` method, e.g.:

```
<%=ivy.cms.co("myUri")"%>
<%=ivy.html.get("in.myString")%>
```

Furthermore you can insert references to content from the web content directory into your JSP content objects, e.g.:

```
<jsp:include page="/jspToInclude/include.jsp" />

```

Layout Editor

The Layout editor is used to edit a JSP Layout content object which defines the *Layout* of an HTML dialog page (see Page Editor).

Inside the Layout JSP content objects you can make use of the Environment Variable *ivy* or you can insert references to content from the web content directory (see JSP Editor).

For the layout creator there are some useful functions on the *ivy* variable:

Layout functions	Description
<code>ivy.style()</code>	Returns the URL to the CSS style, which is set on the current HTML dialog page (see Page Editor).
<code>ivy.content("coName", "coType")</code>	The layout creator can define a placeholder for content, that the page designer should fill in. When a page with this layout is created the specified placeholder is created as a content object with the given name and type under the HTML dialog page whose value is set by the page designer. At execution time the value of the content object is set as a String into the layout.
<code>ivy.panel("panelName")</code>	This creates a Panel content object with the specified name under the HTML dialog page when a page with this layout is created.

A very simple JSP Layout example which includes the style of the page, creates a content object named *Caption* with the type *String* and a *Panel* content object with the name *Panel1* looks as follows:

```
<%@ page import="ch.ivyteam.ivy.page.engine.jsp.IvyJSP"%>
<jsp:useBean id="ivy" class="ch.ivyteam.ivy.page.engine.jsp.IvyJSP" scope="session"/>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <link rel="stylesheet" type="text/css" href="<%=ivy.style()%>"/>
  <title><%=ivy.content("Caption","String")%></title>
</head>
<body>
<jsp:include page='<%=ivy.panel("Panel1")%>' flush="true"/>
</body>
</html>
```

Link Browser

The link browser can be used to insert a HTML `` link tag. The appearing dialog shows the available link types to choose from. If you select a link type and then press Ok (or double-click the link type), then another dialog appears to configure the link.

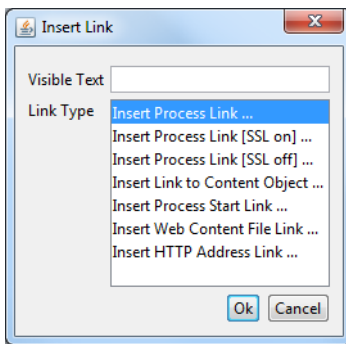


Figure 6.24. Link Browser

Visible Text The visible text of the HTML link that will be inserted. Or, in HTML language, the text between the `<a>` and the `` tags.

Link Type The type of link to insert.

Link Type	Description	HTML href link target
Process	A link to continue the process with the corresponding sequence flow out of the <i>Web Page</i> element that shows this page. The link target is addressed relatively, i.e. uses the same protocol like the request to the page. See the section for the inscription mask of the web page element for further information about process links.	<code><%=ivy.html.ref("MyLink.ivp")%></code>
Process (SSL on)	Same as the process link, but addresses the process link absolutely using HTTPS.	<code><%=ivy.html.ref("MyLink.ivp", true)%></code>
Process (SSL off)	Same as the process link, but addresses the process link absolutely using HTTP.	<code><%=ivy.html.ref("MyLink.ivp", false)%></code>

Link Type	Description	HTML href link target
Content Object	This is a link to a content object in the CMS. Note that not all content object types may be used but only the ones that represent file-based resources. This holds true especially for file-based content objects such as images or documents.	<code><%=ivy.html.coref("/uri/to/my/jpeg")%>"></code>
Process Start	A link that starts a new process. Note that the process is started in a new case.	<code><%=ivy.html.startref("myPID/start.ivp")%></code>
Web Content	For linking to a resource in the web content folder	<code><%=ivy.html.wr("myResource")%></code>
HTTP Address	Inserts a link to an arbitrary web resource such as a web page or a video on the web.	<code>http://www.example.com</code>

Table 6.67. Link Types

Other content editors

Content editors that are not described in this chapter are described in the Content Object Value Editor chapter.

HTML content in the Web Content Folder

Although it is recommended that you store all (or most of) your content in the project's CMS, as an alternative you can place web resources into a folder within the root folder of your project with the name *webContent*.

These web content files can be referred from CMS pages either by addressing them relatively (to the web content directory) or by using the method `ivy.html.wr(fileNameRelative)`. Of course, you can also reference directly from one web content file to another (e.g. a web content HTML file that displays a web content image with a `img` tag.). It is even possible to mix references between CMS and web content files (e.g. a JSP in the web content includes a JSP in the CMS which imports a JavaScript file in the web content and so on).



Warning

Web content files can always be referenced relative to the web content folder. But in contrast to content objects in the CMS, resources in the web content folder are only visible in the same project. If the resource is not found then there is no lookup in the required projects.



Tip

To gain access to Axon.ivy functionality please insert the following code to your hand-made JSP page in the web content folder:

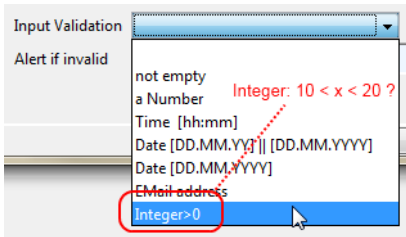
```
<%@ page import="ch.ivyteam.ivy.page.engine.jsp.IvyJSP"%>
<jsp:useBean id="ivy" class="ch.ivyteam.ivy.page.engine.jsp.IvyJSP" scope="session"/>
```

HTML Best Practice

How to provide own HTML Validation Scripts

Inside the Form Field Details Editor you may specify client side validation scripts to validate user input before submitting a form. If the validation fails (i.e. some user input is invalid) then the form will not be submitted.

Out of the box, Axon.ivy allows you to select from a number of predefined scripts, e.g. *Integer > 0* as shown below:



Testing for *Integer > 0* may be fine for some cases, but what if you need to test whether an integer value is greater than 10 and smaller than 20? In this case you need to provide your own validation script to test your specific input requirements.

HTML validation scripts are written in *JavaScript* and are stored in a *.js file. The script file contains a header line and a check function. For example:

```
<!--ivy.input_validate name_de="Ganzzahl: 10 < x < 20" name_en="Integer: 10 < x < 20"-->
function integerBetween10And20(field,msg,loc)
{
  msg_en=field.name+" must be a Number between 10 and 20";      // default message en
  msg_de=field.name+" muss eine Zahl zwischen 10 und 20 sein";  // default message de

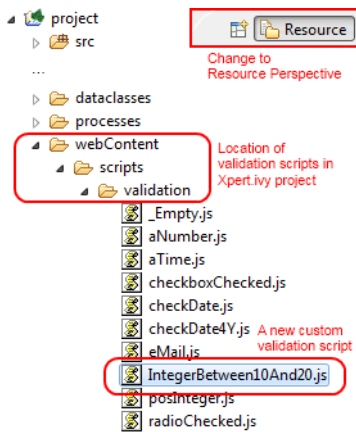
  if(field.value.length==0
    || isNaN(new Number(field.value))
    || !(field.value > 10 && field.value < 20)) // check function
  {
    if(msg!=null && msg.length>0)
      alert(msg);      // alert with custom message, if defined
    else if(loc=="de")
      alert(msg_de);  // alert with default german message
    else
      alert(msg_en);  // alert with default english message
    return false;
  }
  else
  {
    return true;
  }
}
```

The header of the script file defines the name of the script as it is displayed in the validation combo box of the *Form Field Details Editor*.

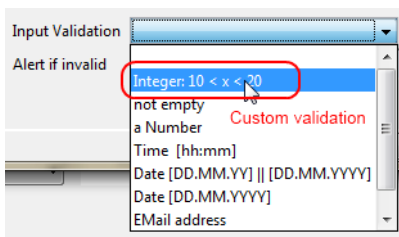
The JavaScript validation function must always define the same 3 parameters: `field` is the field that is being validated, `msg` is the custom validation alert message that may optionally be defined when selecting a validation script and `loc` is the language code that will be provided by the Axon.ivy runtime system.

The above defined checks if the given input field contains a number between 10 and 20 and if so, `true` is returned. If the value is not a number or not between 0 and 20, then an alert message is shown to the user and `false` will be returned (which will prevent the sending of the form, as long as the field value is not correct).

In order for the script to be able to selected from the Form Field Details Editor, it must be located in the `webContent/scripts/validation` folder of the Axon.ivy project where it will be used:



Once you have copied your script to this location, it will become available in the validation selection combo box of the *Form Field Detail Editor*:



Tip

If you don't like the behavior of any built-in scripts then you may change them as well. Simply edit the associated validation script file.

You may generally edit the available validation scripts at any time. The changes will become effective immediately on save (unless browsers or other internet entities use a previously cached copy of an old script).



Warning

Please note, that *JavaScript* must be enabled in your client's browsers in order for any validation scripts to work!

Also make sure that the name of your script function and the name of your script file are exactly identical (including capitalization). The script will be referred by name of the file. If it does *not* match the actual function name, then the script will not be executed, and validation will not take place without any obvious error.

Chapter 7. 3rd Party Integration

Introduction

The basic idea of integrate Axon.ivy with 3rd party systems is either to invoke an operation on a foreign system out of an ivy process (call) or to have a remote system that invokes an operation in Axon.ivy (being called). There are several options to implement such an integration. This chapter will give you an overview.

Java Interface

The generic approach to integrate 3rd party system with an ivy process is the usage of the Axon.ivy Java interface. The Java interface is provided in following process elements:

“Program Start”	Triggers the start of a new process upon an (external) event.
“PI (Programming Interface) Step”	Executes generic Java code - which may interact with a foreign system.
“Wait Program Intermediate Event ”	Interrupts process execution until an (external) event occurs.
”	
“Call & Wait ”	Basically a combination of the <i>PI</i> and <i>Wait</i> process elements.

Database

A simple way to integrate Axon.ivy is the usage of an external database. From an ivy process, database contents can be read/written by “DB Step ” or by using JPA.

Web Services

SOAP based web services are often used to integrate varying systems together. In Axon.ivy, the “Web Service Call Step ” is used to call a web service on a foreign system.

Moreover, Axon.ivy is also capable to provide web services to other systems through Web Service Processes. They are started by an external web service call to Axon.ivy.

REST Services

REST (representational state transfer) is an architectural style based on resources to provide inter-system communication.

The Java API specification for RESTful Web Services is called JAX-RS. It provides portable APIs for developing, exposing and accessing Web applications designed and implemented in compliance with principles of REST architectural style.

Axon.ivy uses the reference implementation libraries of JAX-RS called Jersey.

Call remote REST Service

To call a remote REST service it has to be defined in the “REST Clients Configuration”. After that a “REST Client Activity” can be used to call the REST service.

Examples can be found in the *ConnectivityDemos* project which is packed with the Axon.ivy Designer. It is located under *[DesignerRoot]/applications/samples/ConnectivityDemos.iar*.

Provide own REST Services

To provide a custom REST services from an ivy project, JAX-RS annotations can be used. A REST resource is created by adding a Java class to the *src* directory. The Java class has to use the correct annotations (as shown below), then it will be

detected as REST resource and published automatically. After publishing, the resource will be available on the base path `/ivy/api/`.

```
/**
 * Provides the person REST resource
 * on the path /ivy/api/myApplicationName/person
 */
@Path("/{applicationName}/person")
public class CustomProjectResource {
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Person get() {
        Person p = new Person();
        p.setFirstname("Renato");
        p.setLastname("Stalder");
        return p;
    }
}
```



Tip

It's necessary to include the `{applicationName}` in the resource path. `{applicationName}` refers to the name of the ivy application. The application is required to perform authentication of the REST resource. A call to the REST resource must provide valid credentials of a workflow user.

Further information is available in the JAX-RS API Specification. If you are looking for a sample about how to use JAX-RS in an ivy project, you can study the *ConnectivityDemos* sample project in the Designer.

Workflow API

Axon.ivy provides a basic Workflow API REST Service. It can be used to enable remote systems to request information about tasks of a user etc.

Chapter 8. Configuration

Configuration Management

This chapter explains the *configuration* concept of Axon.ivy and describes the *configuration editor* and the various *configuration types*.

Configuration Management

Axon.ivy allows for individual configuration of many of its features, specifically for the following:

Environments	Define environments for your project
Global Variables	Define global variables for your project and assign environment values for global variables.
Widget Styles	Define logical GUI styles for all available Rich Dialog widgets and containers.
Output Format	Define output formats for various data types.
Input Validation	Define validation policies for user input.

The specifics of the above described configuration types are explained in more detail in a separate section.

A configuration is always associated with a *type* (the type of object that it configures) and with an *unique name*. When applying a configuration to an object, only the name of the configuration must be given, because the type is defined by the object being configured.

By consistently defining and using the *logically named* configurations instead of setting properties individually on each configurable object you can later easily change or redefine the looks and behavior of the application in a single place.

All configurations can be created, edited and deleted using the Configuration Editor.



Tip

In Axon.ivy, each project has its own *configuration database*.

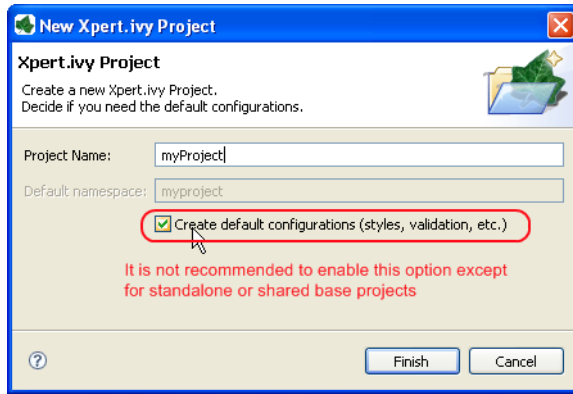
Lookup of any configurations is performed by an *(type, configuration name)* request. If the look-up of such a configuration fails for the current project then the configuration databases of the required projects are searched with the same query recursively. With this mechanism, configurations can effectively be shared among projects.

Since the current project is always asked first for a specific configuration, you can easily *overwrite* configurations that come from a required project, simply by re-defining them locally (on the same type with the same name).



Warning

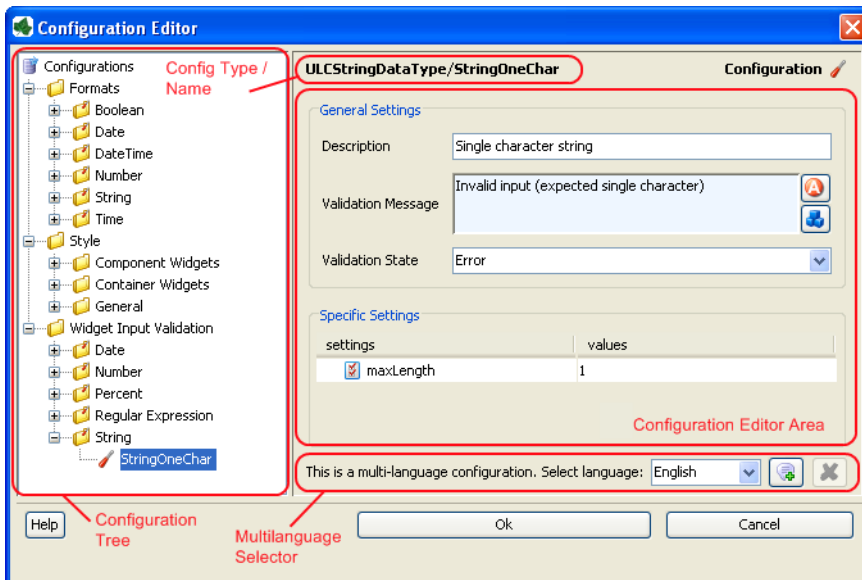
When creating a new project with the New Project Wizard you will be asked if default configuration content should be created for the new project. You should always *disable* this option if you're creating a project that will be dependent on other projects. If you don't disable it then all of the default configurations that are defined in the base project will be redefined automatically in the new project (and thereby *shadow* the inherited configurations), which may lead to unexpected behavior of the application.



If you unintentionally forgot to disable the option in the wizard you can still manually delete all configurations that have been created in the *content database* using the Configuration Editor at a later point of time.

Configuration Editor

The configuration editor is used to add/edit/delete configurations in the hierarchically organized *configuration database* of a project.




The configuration editor consists of different parts that are described separately below:

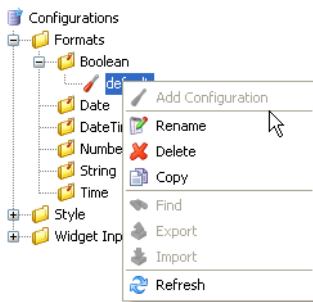
- Configuration Tree
- Configuration Editor Area (with header showing Type/Name of currently selected configuration)
- Multi-Language Selector (not for all configuration types) This part is only visible if you have enabled the multi-language feature in the Axon.ivy preferences. *Axon.ivy preferences > Configuration > Show Multi language Configurations*

Generally, configurations are selected from the tree on the left hand side of the editor and edited in the details area on the right hand side. Modification of the configuration tree is performed with various actions that are available from the tree's context menu.

Accessibility

Axon.ivy Project Tree > double click on the  Configuration entry in the tree.

Configuration Tree



The configuration tree shows the currently defined configurations inside a hierarchically organized tree. *Configurations* (🔧) are associated with a *configuration class* or *configuration type* (📁) which in turn are logically grouped in a hierarchical folder structure (📁).

Only configuration nodes can be edited in the tree, the structure of the configuration database (i.e. the folders and types) is given by the system and cannot be changed.

The following operations are available from the context menu of the various configuration tree nodes:

🔧 Add configuration Adds a new configuration below the selected *configuration class* node.

🔄 Rename Renames the currently selected configuration.



Warning

Renaming of a configuration does not update the objects that currently use the configuration with the old name. I.e. objects or scripts that used the old configuration name may not produce the expected outcome anymore, because the specified configuration does no longer exist. Typically this leads to the fallback of using the *default* configuration of the respective *configuration class*.

✖ Delete Deletes the currently selected configuration node.



Warning

Deleting configurations may lead to the same effects as *renaming* them. See warning above for a description of the possible consequences.

📄 Copy Copies the selected configuration into a new configuration with a name prompted from the user.



Note

Please note that you don't have to *paste* the new configuration, it will be pasted automatically next to the selected *source* configuration, because a configuration is always associated with *exactly* one configuration class. It is not possible to copy a configuration of type *A* to a type *B*.

🔄 Refresh Refreshes the configurations below the selected node.

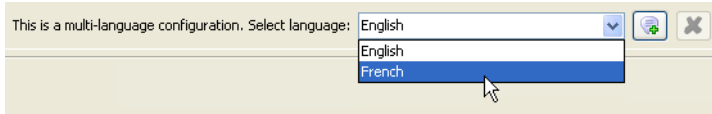
Configuration Editor Area

The editor area shows a different editor depending on the *configuration type* (also called *configuration class*) of the currently selected configuration. The individual editors are described in the configuration types section.

The title bar of the editor area shows the type (i.e. the *configuration class*) and the name of the currently selected configuration as `<ConfigClassName>/<ConfigurationName>`.

Multi language Configuration Selector

Enable the multi selection configuration by setting the flag in the Axon.ivy preferences.



Some configurations can be defined in multiple languages because they contain language specific settings. The *validation* configurations may have to validate input differently if it is entered in another language.



Tip

Use CMS references for the error message property in order to support multiple languages.

The *language selector* is shown only for configurations that support multiple language values per configuration. Use it to select the language of the configuration that you want to edit.

Features:



Add new language value

Adds a value in a new language to the currently selected configuration. The contents of the last shown value are copied as initial content to the new configuration value.



Delete language value

Clicking on this button removes the currently shown value from the set of values for this configuration. Only works if more than one language value are defined.

Renderers Configuration

Renderer Configurations are used for specific Content Object when they are rendered, the result then is displayed on the client browser.

Smart Table Configuration Editor

The Smart Table Configuration editor is part of the Configuration Management. See Configuration Editor for more information. A Smart Table Configuration can be added by using the add configuration function. The name is used in the Smart Table Content Object Editor for choosing a specific Configuration.

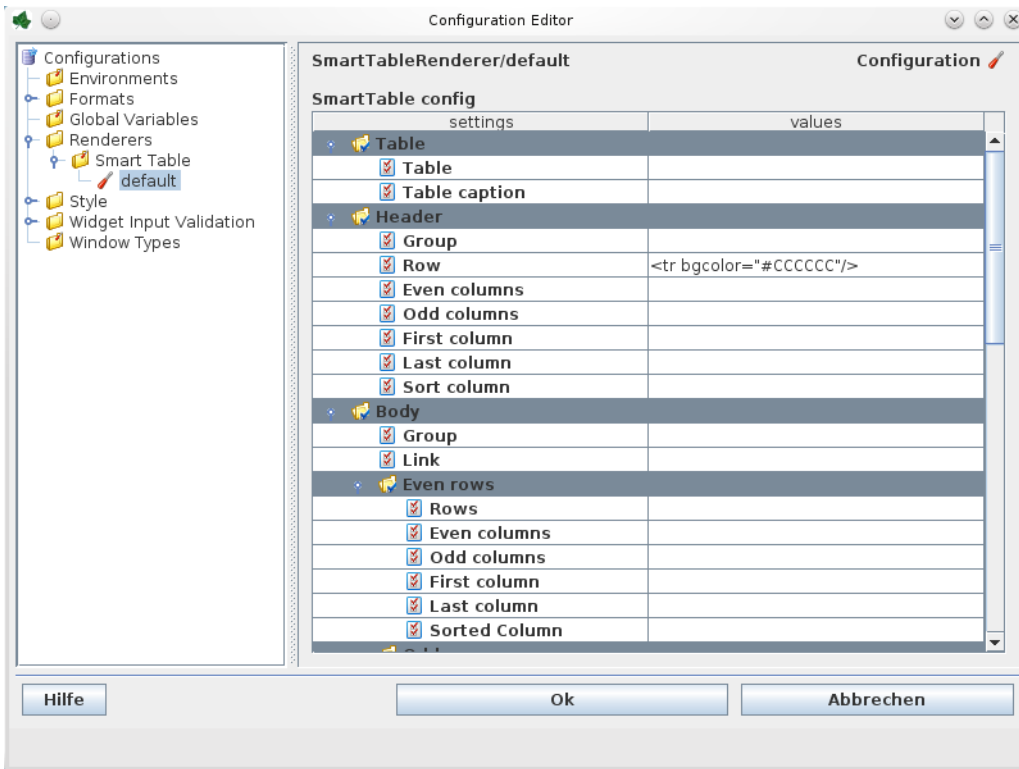


Figure 8.1. The Smart Table Content Editor

Table Settings	The HTML attributes settings for the table and the caption HTML elements.
Header Settings	The HTML attributes settings for the table header group (thead element), the header row (tr element) and the header cells (th elements).
Body Settings	The HTML attributes settings for the table header group (thead element), the header row (tr element) and the header cells (th elements).
Footer Settings	The HTML attributes settings for the table body group (tbody element), the body rows (tr elements) and the body cells (td elements). Here it is possible to define different configuration for the rows and cells depending if they are displayed on an even or odd row position. The cells can also be different if they are in a even or odd column, in the first or last column or if they are in the column which is actually sorted. The first and last column settings does override the even or odd columns settings and the sorted column setting does override all other settings.
Sort Settings	Enter the text which is displayed in the header if a column is sortable. Then Ascending and Descending values are shown. When a column actually is sorted then the Sorted ascending or Sorted descending values are shown.
Page Selector Settings	Enter the format of the page selector, where it is shown (default: bottom) and the values displayed for the first, previous, next and last page to show. For the format there are the following possibilities:

Symbol	Description
{<}	Link to jump to the first page is shown with the value configured with the setting "First page".
{<}	Link to jump to the previous page is shown with the value configured with the setting "Previous page".

Symbol	Description
{123}	Links to jump to every page is shown with the page number.
{>}	Link to jump to the next page is shown with the value configured with the setting "Next page".
{> }	Link to jump to the last page is shown with the value configured with the setting "Last page".
{n}	The number of the actual page is displayed.
{m}	The number of the total amount of pages is displayed.
{x}	The number of the first showed row is displayed.
{y}	The number of the last showed row is displayed.
{z}	The number of the total rows in the table source is displayed.

Table 8.1. Format Symbols

Configuration Types

The *configuration database* of a project contains configurations for various configurable targets (e.g. widgets, date or time values, input text fields, etc).

The user can define configurations for the following three aspects of an application: *formats*, *styles* and *validation*. Each of those aspects encompasses a number of configurable types (e.g. *date* or *time* or *number*) for each of which an arbitrary number of *named* configurations can be defined, which can then be referred from different locations in the configuration.

This section briefly discusses the different configuration types and explains the usage of their respective *configuration editors*.

Format Configurations

Purpose

Format configurations define how values of a specific type will be rendered when output to the user. Formats can be defined for all the *base types* of *IvyScript*, i.e. **Boolean**, **Date**, **DateTime**, **Time**, **Number**, and **String**. All non-null values of those types may be output as *formatted strings* by calling the `format("<format name>")` within any script.

Examples:

```
panel.label.text = in.selectedDate.format("long");
out.invoice.customerNumber = wsData.customer.format("customer_id");
```

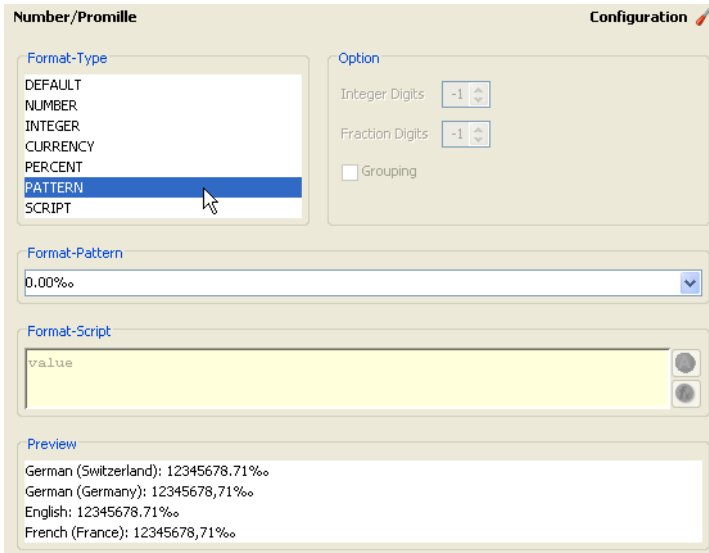
Editor

The format configuration editor allows to specify various ways of formatting for the different *IvyScript* base type values. The formatting options (described below) vary from type to type; not all options are available on all types.



Tip

Format configurations are always *language dependent*. Just watch the *Preview* editor area after each selection or option that you enable for an understanding what the effects are for each language.



- Format Type** Select a predefined kind of formatting. The selected format kind may be further specified in the editor areas *Option*, *Format Pattern* and *Format Script* (depending on which kind that was selected).
- Option** This editor area is only available for some of the *Number* format kinds. The number of digits for the integer and/or fraction part may be selected (-1 stands for *as many as needed*). If you select *grouping* then a group character will be inserted to group the integer digits of the formatted number.
- Format Pattern** This group is only available if you have selected the *PATTERN* format from the *Format Type* list.

 If *Pattern* is selected as format then this combo box allows for a selection from a number of predefined patterns. Use *Script* if you'd like to specify an own pattern format.
- Format Script** This group is only available if you have selected the *SCRIPT* format from the *Format Type* list.

 Specify your own format with a script. The *value* variable will contain the value to be formatted. If you want to specify different patterns for different languages you should use the language selector to create multiple configurations for various languages.
- Preview** Shows an instant preview of the configured formatting applied to a default value.

Style Configurations

Purpose

Style configurations define the *appearance* of widgets when rendered as part of an user interface. Styles can be defined for each widget and container type separately, e.g. for **Label**, **TextField** or **BorderLayoutPanel**. You can also specify *general* Styles that can be used for all container and widget types.

The Look up for the right settings to apply to a specific widget takes place in a provided order. The first match wins. Each of the following six pints will be evaluated respective to the project dependency:

1. Local Style set via the Widget Configuration.
2. Widget style (e.g. for a RLabel) set by a reference in the Widget Configuration.
3. The style with the same reference as for point (2) of the RWidget type.
4. Default style of the specific widget (e.g. RLabel).
5. Default style of the RWidget type.

6. Styles set in the Properties view or in the Java source code.



Note

The dependent projects are browsed by a **breadth first** algorithm.

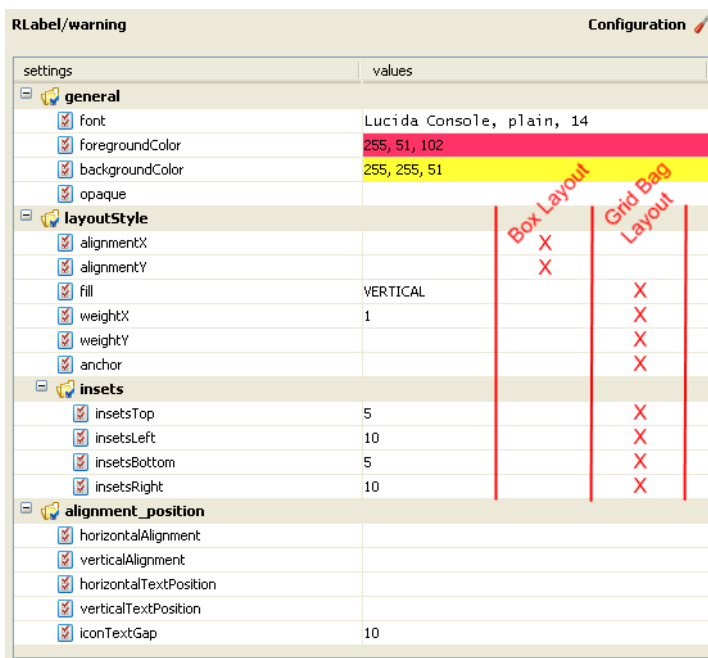
Editor

The style configuration editor allows to specify style properties for each widget or container type. Not all widget types have the same properties that may be specified.



Note

The *layout style* properties are *layout dependent*. They may not have an effect, if the widget is added to a different layout than the property is intended for (mostly `GridLayout`). For example, the property **weightX** will not have any effect if the widget is added to a `BorderLayout`.



General

Define general properties such as the *font*, the *foreground color* and the *background color*. Setting of the background color may not have an effect, if the widget or container do not have *opaque* to *true* (this is look-and-feel dependent).



Warning

The Look&Feel may override your settings. Specially the Background is usually overridden by the Look&Feel.

Layout Style

Settings to be used when the configured component type is placed within a specific layout.

Insets

Insets (i.e. *inner padding*) that should be applied to the component. Padding will be applied within the specified size (i.e. the actual size of a component is given by it's specified size minus the inset values).

Margin

Margin (i.e. *outer padding*) that should be applied to the component. The specified margin will effectively be added to the specified size of a component.

Alignment Position

Arrangement definition for components that have a text label and an optional icon.

Size Define the *maximum*, *minimum* and *preferred* size of a component. The specified values will be taken into account by the layout manager of the container where the component is added.

Input Validation Configurations

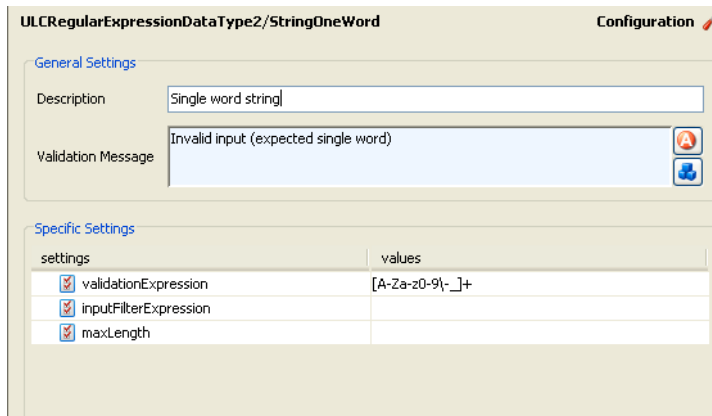
Purpose

Validation configurations can be specified for the following types of input data: **Date**, **DateTime**, **Time**, **Number**, **Percent** and **Regular Expression**. The validation type **Regular Expression**(formal *String*) can be used to validate any kind of data. The result of a validation is always either `true` (valid) or `false` (invalid).

Each validation configuration contains of a set of *general settings* and a set of *specific settings* that are dependent on the selected type of validation.

Editor

The validation configuration editor allows you to specify various kinds of input validation that can be applied to user text input.



General Settings

Description A short verbose description of the validation. This text will be shown in the widget configuration to help selecting the correct validation.

Validation Message Enter the message that will be displayed to the user on a tool tip, if the entered input is not valid. Use CMS (Content Management System) references here, in order to support multi language messages.

Specific Settings

Specific setting vary between the different validation types The list below describes all available settings.

Format	Defines the format on how the value will be presented to the user on the UI. Select a format out of your format configurations.
--------	---

Table 8.2. Date Validation Type

Format	Defines the format on how the value will be presented to the user on the UI. Select a format out of your format configurations.
--------	---

Table 8.3. DateTime Validation Type

Format	Defines the format on how the value will be presented to the user on the UI. Select a format out of your format configurations.
--------	---

Table 8.4. Time Validation Type

Format	Defines the format on how the value will be presented to the user on the UI. Select a format out of your format configurations.
Max	Maximally permitted number value.
Min	Minimally required number value.

Table 8.5. Number Validation Type

Format	Defines the format on how the value will be presented to the user on the UI. Select a format out of your format configurations.
--------	---

Table 8.6. Percent Validation Type

Validation Expression	A regular expression that the entered input must match. This expression is not evaluated until the field loses focus.
Input Filter Expression	A regular expression that the input must conform to in order to be set on the widget. Default value is '.*'
Max Length	Maximum length of the input text.

Table 8.7. Regular Expression (String) Validation Type

Validation Framework

The validation framework enables a Rich Dialog developer to implement the needs for GUI widgets that give feedback to the user about its entries in the GUI widgets. E.g. it draws a red background into a RTextField if the entered value does not conform to a date if a date is expected.

Supplied Features by the Validation Framework

Background Coloring	The background of the supported widgets can be colored do to its validation or mandatory state.
Tool tip	The tool tip of the supported widgets is decorated do to its validation and mandatory state.
Server Side Validation	Its possible to set a error on a supported widget from the server side.
Supplied Widgets	The framework supplies two widgets that are not on the palette: RErrorHandler and RErrorView.

Widgets of the Validation Framework

Widget	Mandatory State	Validation Types
RTextField	yes	all
RComboBox	yes	no

Widget	Mandatory State	Validation Types
RTextArea	yes	no
RLookupTextField	yes	no
RButtonGroup	yes	no
RCheckBox	yes	no
all RContainers	inherited	inherited

Table 8.8. Supported Widgets of the Validation Framework

Widget	Description
RErrorHandler	A <code>CollapsiblePaneContainer</code> that shows the current errors of the <code>observedContainer</code> widget in a list. The list will be updated as soon as some content or state of the observed container or its children changes. Important properties: <code>ShowMandatoryErrors</code> , <code>VisibleMessages</code>
RErrorView	A widget that shows the errors of the observed container in a list. The list will be updated on each client-server round trip.

Table 8.9. Supplied Widgets by Validation Framework (not on the palette)

Configurations

General framework configurations Edit the file `%AXON_IVY_INSTALL%/configuration/validationconfig.any` in order to configure the framework for your needs. The configuration file lets you configure the following properties:

<code>mandatory-error-message</code>	The message that will be displayed on the tool tip to the user if the mandatory field has no user changes.
<code>mandatory-background-color</code>	Color of the background if the field has no user changes.
<code>error-background-color</code>	Color of the background if the field has erroneous user changes.
<code>background-feedback</code>	Set this flag to true if the background feedback should be enabled.
<code>show-highlight-color-when-disabled</code>	Set this flag to true if the background feedback should be enabled even if the widget is disabled.



Tip

The configuration file `validationconfig.any` changes the validation behaviour for the whole designer/engine not only for selected project(s). This setting will not be uploaded automatically but has to be set up manually on the Engine just like on the Designer.

Programming Interface (API)

The API of the Validation Framework is very slim. It consist mainly of a `IValidator` object per supported widget. To access this `IValidator` object you can call the method `ivy.rd.getValidator(widget)`. Some widgets offer special

properties to get validated values (e.g. `valueAsNumber` on `RTextField` if validated for number). In case the entered value is not accepted by the actual validation, then a so called `INVALID` value of the specific IvyScript type is returned by the widget (e.g. `Date.INVALID` from `RDatePicker`'s `valueAsDate` property).

All `RContainers` implements the `IEnabler` interface, that enables those widgets to act as enabler for other widgets (E.g. a submit button).

The `IValidator` object returns `IvyErrorObjects` if you ask for pending errors on a `IValidator`.

IValidator
<code>hasErrors()</code>
<code>getErrors() : List<IvyErrorObject></code>
<code>getError() : IvyErrorObject</code>
<code>showErrorMessage(String errorMessage, Serializable illegalValue)</code>
<code>addPropertyChangeListener(PropertyChangeListener listener)</code>
<code>removePropertyChangeListener(PropertyChangeListener listener)</code>

Table 8.10. `ch.ivyteam.ivy.richdialog.server.validation.IValidator`

IValidator
<code>getMessage() : String</code>
<code>getParameter() : Object[]</code>
<code>getError() : IvyErrorObject</code>
<code>getIllegalValue() : Object</code>

Table 8.11. `ch.ivyteam.ivy.richdialog.shared.IvyErrorObject`



Tip

Server side validation cannot be used alone, it must be combined with a client side validation!

Environments

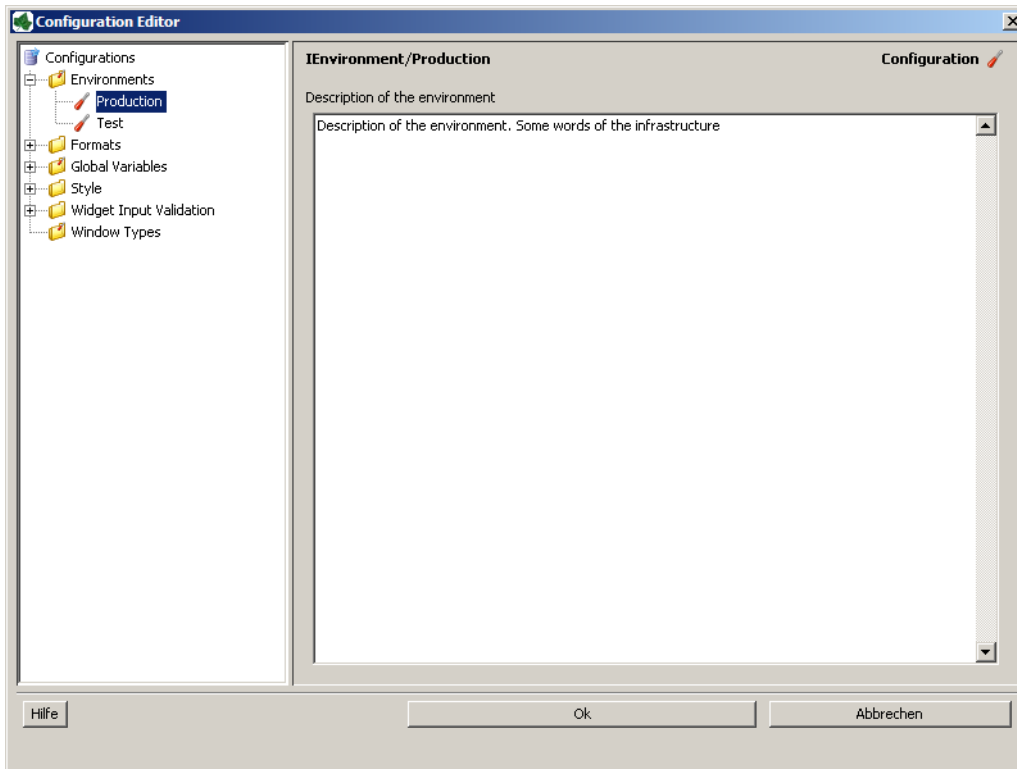
This section briefly discusses the usage of environments in your projects .Developers should have the possibility to configure multiple environments (pointing to an infrastructure) and decide at deployment, runtime and/or design time which environment should be used for the application. Here are some examples where environments can be used

- Companies provide different environments for their software products, like Development, Test and Productive. Each environment has its own infrastructure including databases, web services and other connections used by the project.
- Multi Client Capability. When the user logged into the system he can choose the client (German: Mandant) (e.g. Company 1, Company 2, etc.) and works with the data of the selected company. In the background the right databases connections, web services and other services for the selected environment will be used.

Actual projects use the CMS functionality to manage global Constants and set these constants every time on the engine when a new version of the project will be deployed. Another solution for this problem is the usage of property files which will be loaded by the application. If you use environments, the user only change the environment and all configuration settings of the environment will be used (e.g. database, web service, etc.) once. The system switch the settings in the background.

Editor

The environment editor is part of the Configuration Management. The basic handling of the configuration editor will be described here. By using the add configuration function a new environment will be added. The name of the environment is the name of the configuration.



Description of the environment

Provide a description of the environment. The description can also be used to describe some server IPs, specifics of the infrastructure and other interesting stuff.



Note

Remove Environments If environment configurations are removed all environment values for global variables, databases and web services will also be removed. If you environments removed from projects included in the Library the values will also be removed in your project.



Warning

Rename Environments If environment configurations are renamed all configurations for global variables, databases and web services will NOT be replaced by the new name. This will be supported in a future version.

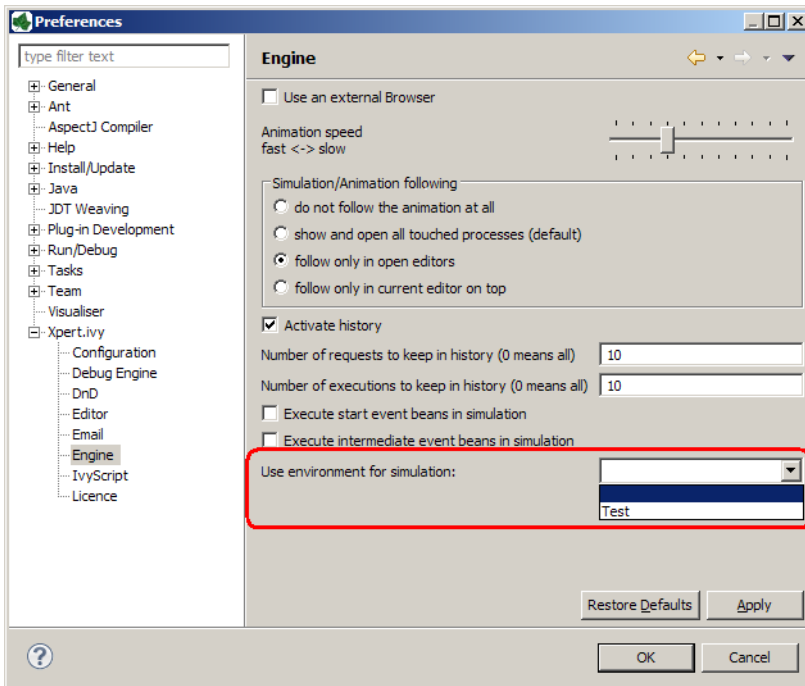
Environment settings

After the environment is created you can now assign environment values for different project objects. The environment settings for the project objects are done in their associated editors. Environment setting can be done for:

Global Variables	Assign environment values to global variables
Databases	Add environment connection properties to databases
Web Services	Add environment connection properties to web services

Change environment for Simulation

It is possible to change the environment for the Simulation by setting the Environment in the Preferences.



Change environment at runtime

It is possible to change the environment at runtime on Case, Session and Application Level. This is an essential feature in order to fulfill the requirements of an multi-client capability application.

Scope	How to set in IvyScript
Case Scope	<code>ivy.case.setActiveEnvironment(String name)</code>
Session Scope	<code>ivy.session.setActiveEnvironment(String name)</code>
Application Scope	<code>ivy.wf.getApplication().setActiveEnvironment(String name)</code>



Note

The `setActiveEnvironment(String name)` methods throw a runtime exception if no environment with the given name exist.



Important

How does the system check which environment is active?

The environment determination is as follows:

1. Environment that is set on the current **case**.
2. Environment that is set on the current **session**.
3. Environment that is set on the current **application**.
4. **Default** environment

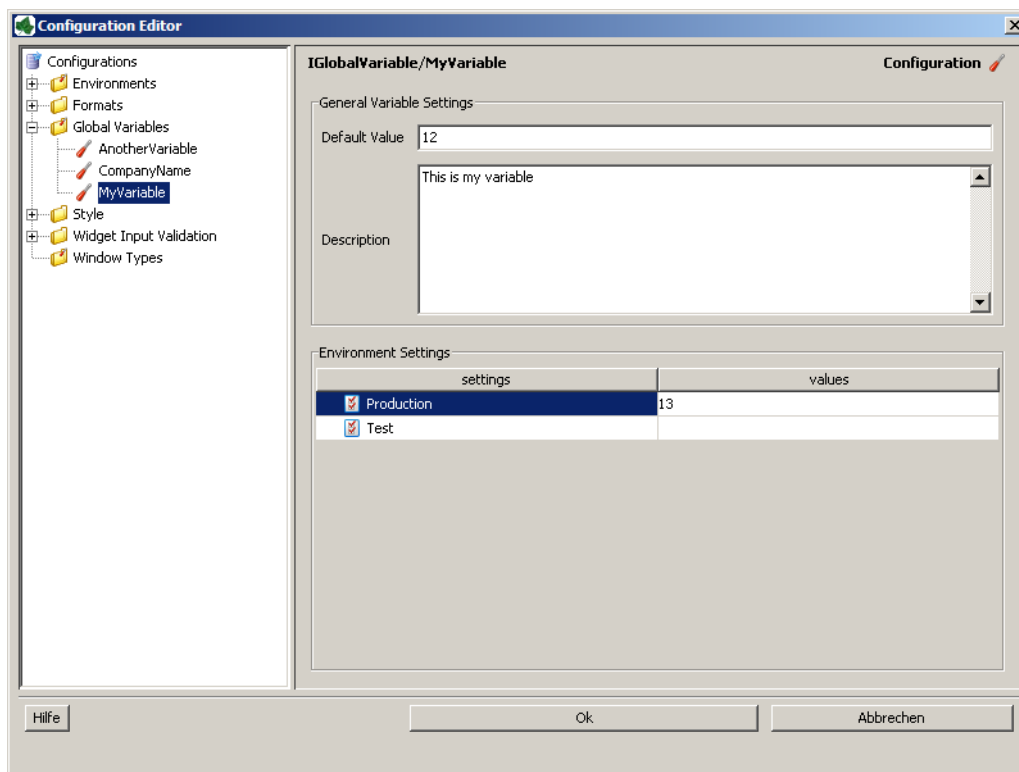
Global Variables

Global Variables acts as global constants which can be used in your application. Global Variables are simple Key/Value pairs which can be specified by the developer. Some examples for global variables are:

- Company data (name, address, contacts)
- Simple Rule Values (e.g. credit account)
- Path values for saving files
- Path values for 3rd party systems and some other variables

Editor

The global variable editor is part of the Configuration Management. See Configuration Editor for more information. A global variable can be added by using the add configuration function. The name of the global variable is the name of the configuration.



Default value

Provide a default value of the global variable. This can be a Number or a String. This value will be used if you access the global variable in the application.

Description

Provide a description of the global variable.

Environment settings

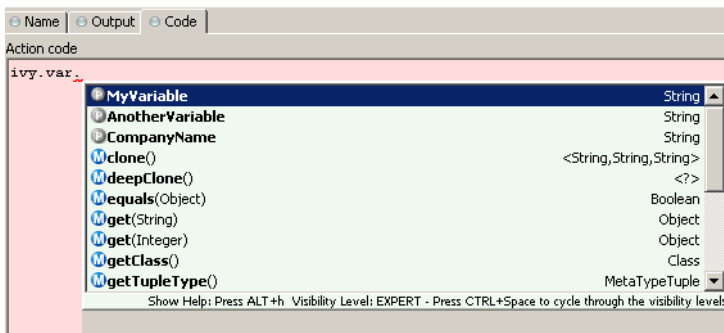
In this section the default value of the global variable can be overridden/new assigned for the specific Environment.

How does it works at Runtime

If you work with environments at runtime, the system looks for an environment value of the global variable. If no environment value was found for the global variable, the default value will be used. Note that it is not necessary to assign global variable values for every environment. If a global variable has always the same value for all environments, only the default value should be set.

Access global variables in IvyScript

In order the access the Global Variables in your code a new environment variable **var** was introduced in IvyScript which provides a comfortable access to your defined variables. This approach has the advantage that, if global variables are removed developers will immediately informed in which process element the variable was used.



Database Configuration

This chapter deals with the database configuration. To use databases in your business or User Dialog processes you need to define some database configurations first. After you have configured the databases (data sources) you can use them in your process steps. The process steps references only the database configuration ids. So you can use different database configuration settings on your productive server.

Database Configuration Editor

Overview

The Axon.ivy database configuration editor lets you configure the databases you use in your project and the extending projects.

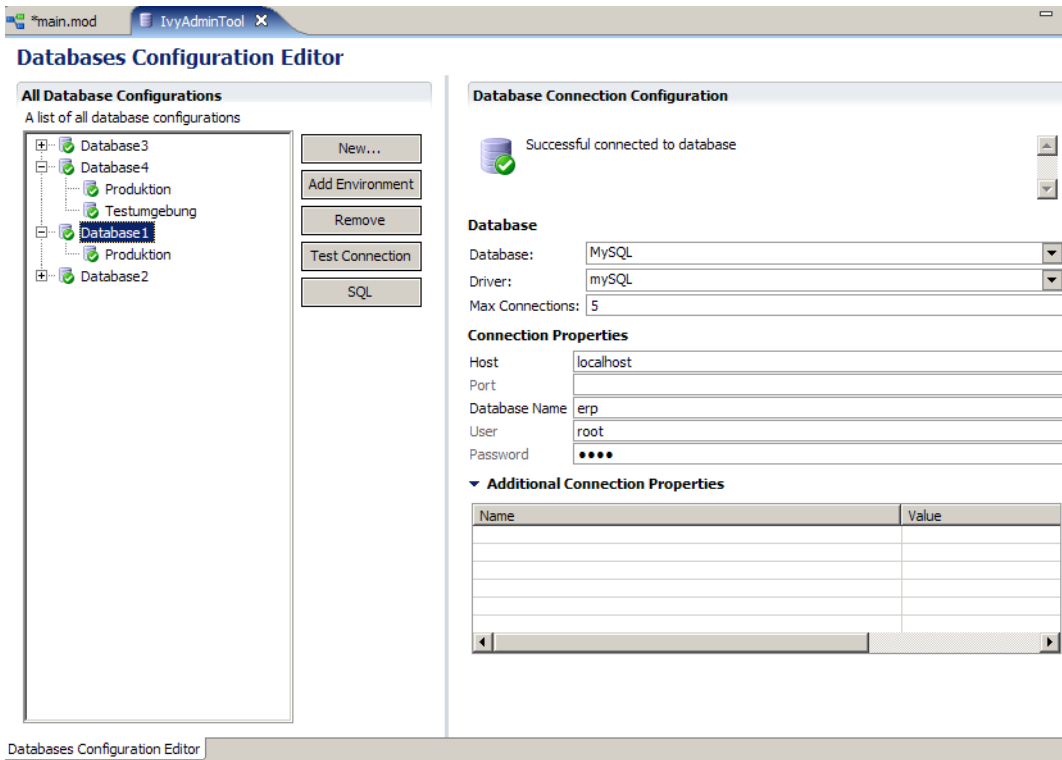
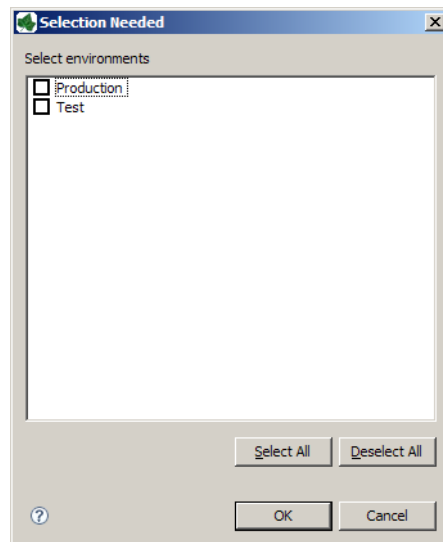


Figure 8.2. The Database Configuration Editor

- New** Add a new database configuration
- Add environment** Add an environment configuration for the selected database. Select one or more environments for the environment list. For new environment configurations the standard configuration is taken. Please note, existing environment configurations will be overridden, if you add an environment twice.



- Remove** Remove the selected database configuration
- Test Connection** Test database connection. A dialog shows the result status if the database can be connected or not. In the case of a failure the reason will be displayed
- SQL** Opens a SQL editor in order to set up SQL Statements. The SQL Editor displays the result in a result table

Accessibility

Axon.ivy Project Tree > double click on the Persistence label.

Environments

In the configuration editor you can define different Environments. The environments can be added to existing database configurations. For example it is possible to define a Database for your development, test and production environment. Depending on the Active Environment the associated environment configuration will be used. If no environment configuration for a database is available, the default configuration will be used.

If a new database configuration is added to the container, the system automatically adds empty environment database configurations for each environment found in the project.



Note

The root database acts as the default configuration and must always be set



Tip

You don't must define environment configurations for every database. If the database configuration is the same for all environments, only the default configuration is necessary.

Features

All Database Configurations A list of all database configurations defined in this project. A red or green icon indicates the result of the automatically executed connection test.

Database Connection Configuration Shows the state of the automatically executed connection test.



Database

Select the type and driver of the database you use. Some often used and tested drivers are shipped with the Designer. Additional drivers can be added by copying its JDBC driver Jar into the *jre/lib/ext* folder of your Axon.ivy installation.

The field `max. connections` lets you specify the number of concurrent connections to your database.

Connection Properties

Specify the properties for the connection to your database.

Additional Connection Properties

If your database needs more information you can use this section to define the additional properties.

REST Clients Configuration

The REST Clients configuration contains the definition of all REST services, which can be consumed from a BPM process.

REST Client

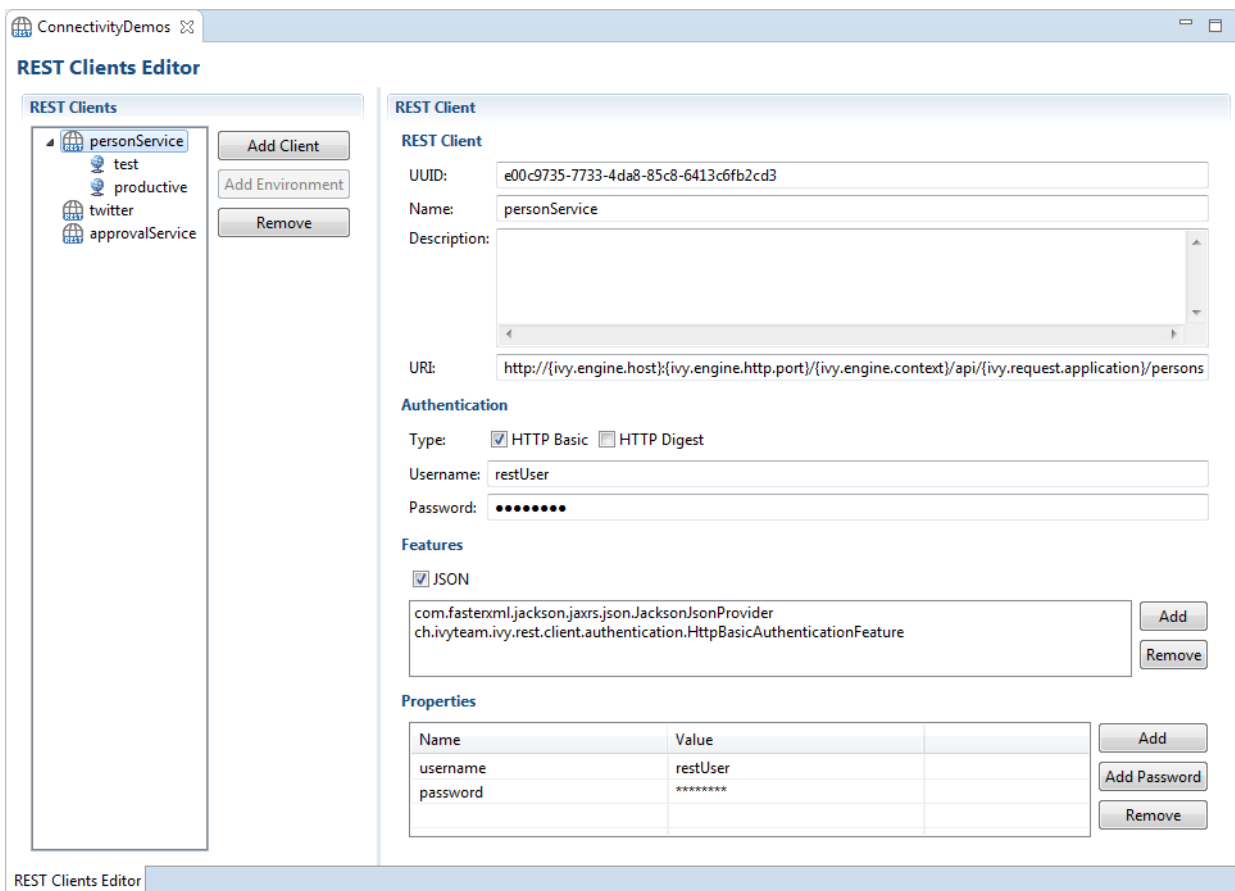
A REST Client can be referenced by its name or by its universal unique identifier (uuid). The uuid is generated when a new REST Client is created and will never change. The name of a REST Client is given when a new REST Client is created. It can be changed later. By referencing a REST Client by its uuid ensures that renaming of the REST Client will not break the reference.

Further information about how to use the REST Clients can be found in the chapter “Call remote REST Service”.

Like other configurations a REST Client can be configured differently per environment.

REST Client Editor

The REST Client Editor allows to configure REST Client default and environments configurations.



REST Clients Tree Editor

Shows the REST Clients and its environment configurations.

Add Client Adds a new REST Client.

Add Environment Adds a new environment configuration for the selected REST Client.

Remove Removes the selected environment or REST Client configuration.

REST Client Details Editor

Shows the currently selected REST Client or environment configuration.

REST Client Section

UUID	Universal unique identifier of the REST Client. The REST Client can be referenced by this uuid. Cannot be modified.
Name	The name of the REST Client. The REST Client can be referenced by this name. Can be modified. Note that references using the name will break if you change it.
Description	Description of the REST Client.
Uri	The base URI under which the remote service publishes its resources (e.g. <code>https://api.twitter.com/1.1</code>). The URI can contain template placeholders which are resolved later by the client user (e.g. <code>https://api.twitter.com/{version}</code>).

```
ivy.rest.client("twitter").resolveTemplate("version", "1.1").get()
```



Tip

To consume a REST service running in the same Axon.ivy Engine / Application as the client a set of Axon.ivy placeholders can be used. These placeholders are automatically resolved: `{ivy.engine.host}`, `{ivy.engine.http.port}`, `{ivy.engine.context}`, `{ivy.request.application}`.

E.g. `http://{ivy.engine.host}:{ivy.engine.http.port}/{ivy.engine.context}/api/{ivy.request.application}/my/service`

Authentication Section

HTTP Basic	Adds support for HTTP Basic authentication.
HTTP Digest	Adds support for HTTP Digest authentication.
Username	The name of the user used to authenticate the client.
Password	The password of the user used to authenticate the client.

Features Section

JSON	Adds a feature so that responses in JSON are mapped to Java Objects and Java Objects in requests are mapped to JSON.
Features List	Shows the configured "features" classes. The classes configured here are registered in the <code>WebTarget</code> using the method <code>register(Class)</code> . The classes needs to implement a JAX-RS contract interface and must have a default constructor.
Add	Adds a new feature class.
Remove	Removes the selected feature.

Properties Section

Properties Table	Properties can customize the settings of the REST Client or one of its features.
------------------	--

Client properties

Well known properties of the client are documented here: `org.glassfish.jersey.client.ClientProperties`.

JSON properties

The JSON feature knows many properties that customize the serialization from JSON to Java objects and vice versa.

It is for instance possible to read a very complex JSON object with many fields back to a Java object that contains only a subset of these fields. To allow this incomplete but efficient mapping the property `Deserialization.FAIL_ON_UNKNOWN_PROPERTIES` must be set to `false`.

Consult the Jackson documentation for a list of all configurable items:

- Jackson Deserialization features can be set using `Deserialization.` as prefix. E.g. `Deserialization.FAIL_ON_UNKNOWN_PROPERTIES`
- Jackson Serialization features can be set using `Serialization.` as prefix. E.g. `Serialization.WRITE_ENUMS_USING_INDEX`

Add	Adds a new property.
Add Password	Adds a new password property. The value of a password property is not visible in the table and is stored decrypted in the configuration file.
Remove	Removes the selected property.

Web Service Configuration

This chapter shows how web service configurations are organized. You get to know web service configuration views, tree entries, configuration types, and how to use, group, set up, change, and refer them.

Structure

Web service configurations are stored together and are represented as a tree. This configuration tree contains as many elements as you need in a structure you like. This is a kind of web service repository. The right place to store web service settings. There are three types of web service configuration nodes:

- **Web service group**
- **Web service configuration**
- **Web service environment configuration**

Web service configuration nodes define a referable unit. Such an entry typically holds a WSDL, service access URL(s), accessing protocol versions, and optionally security configurations. A web service configuration can be referred in multiple Web Service process elements, which actually do the web service invocation. Some of the configurations can be overridden by the Web Service process element (e.g. authentication). The name of the web service configuration can be changed any time. The name is only an attribute and not used as reference key. Each web service configuration is in one of following two states, either:

- **Not referable** or
- **Referable**

A configuration starts its life as not referable. It means that it can not (yet) be used by Web Service process elements. Axon.ivy - just like Java - needs some client side service stub classes to be able to access a web service. These classes are generated using a WSDL file. WSDL files describe how the targeted web service should be accessed like (data types to use, available operations, etc.). Once these necessary client classes are generated, the web service configuration becomes referable. Only referable configurations can be selected in Web Service process elements. Web service groups are just like normal folders in the filesystem. They can contain web service configurations or other groups. Therefore, user can build a practical hierarchy according to his own needs.

Web Service Configuration Editor

With the help of this master-details editor you can build up your own web service configuration tree: Add, remove, and change both groups and web service configurations.

Overview

This chapter describes how to use the web service configuration editor. At the end you will be able to add, delete, and change your web service configurations.

Web service configuration tree editor

Web service configurations are displayed in a tree hierarchy. The tree editor helps to add and delete existing tree elements (web service groups, web service configurations and web service environment configurations).

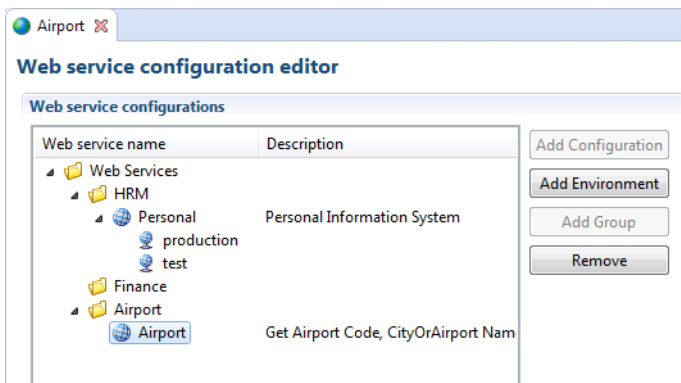
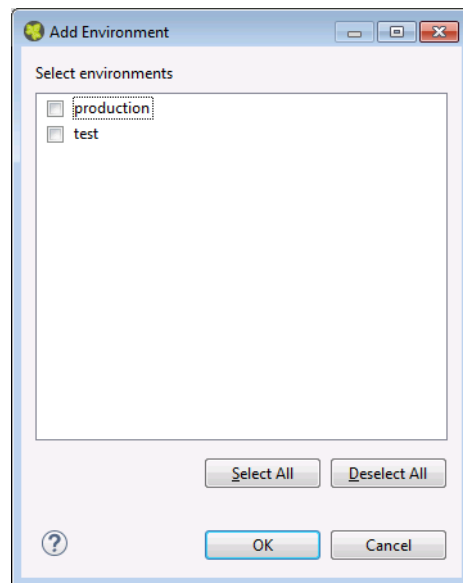


Figure 8.3. Web service tree editor part

In the editor the groups are symbolized with a folder icon. The web service configuration elements are symbolized with a globe icon. On the right side there are buttons offer following functions:

Add Configuration Adds a new web service Configuration

Add Environment Adds an environment configuration to the selected web service configuration. Select one or more environments from the environment list



Add Group Adds a new web service group. These can be used to structuring your services.

Remove Remove the current selection.



Warning

Entries you want to erase, might still be in use. It is encouraged to remove generated client classes as well with your service entry, but take care: some of these classes might be referred by other web service configurations as well.

Web service detail editor

Details of currently selected web service configuration node will be displayed on the right hand side. This editor helps to change details of an existing element (web service group web service configuration or web service environment configuration).

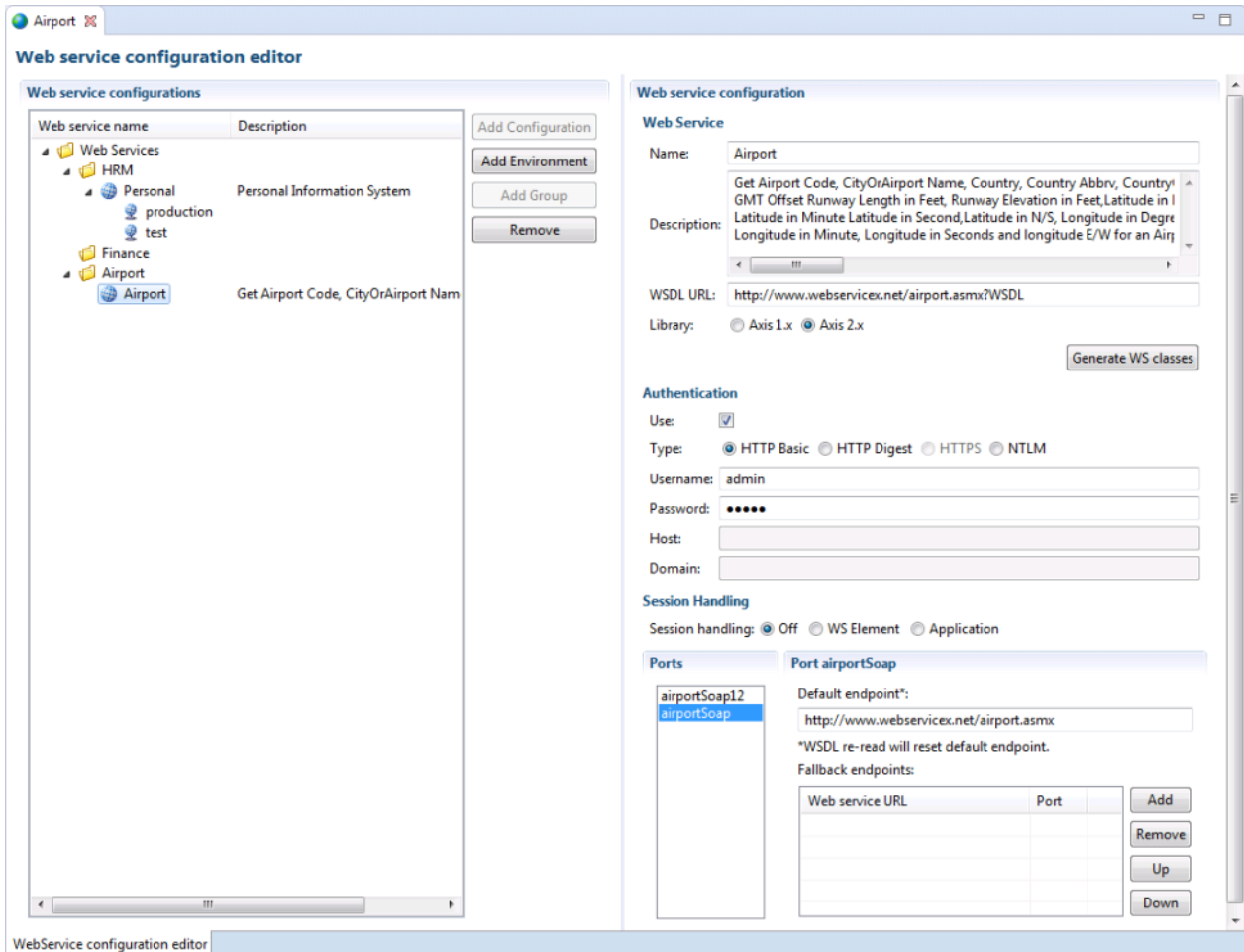


Figure 8.4. Web service configuration details part

The web service configuration detail editor helps to change attributes of an existing web service configuration.

The following attributes are available in the *Web Service* Section:

- Name** The attribute specifies the displayed name of a web service configuration. The name is not used as identifier, so feel free to rename existing entries.
- Description** Description attribute is for own textual descriptions. Not used anywhere in ivy, but you are **encouraged to fill it out**. It helps years later to understand the exact purpose of web service configuration and the way it should be accessed.
- WSDL URL** Service details and classes will be generated using the WSDL specified here. Please **use protocol prefix** like:

http://myserver.ch/hello.wsdl

file://c:\temp\myWis.wsdl

Library The library that is used to generate the web service client classes. Unless you have some special reasons to use the older Axis implementation, please select Axis 2.x (faster, less memory usage, etc.). Note that you have to regenerate the web service client classes if you change this setting. Otherwise the setting will be set back once you save your changes.

Generate WS classes When you specified the necessary fields: WSDL URL, and Library (Axis 1.x or 2.0) then you have to read the WSDL and generate client side classes to be able to access the web service (e.g. use its data types, read its operations, etc.). These generated files will be pre-compiled to be tested, and packed in jar file together. These jar files are located in the *lib/ws_server* folder of ivy project.

Note: **when you change the WSDL URL**, the WSDL itself or the *Library* you **always** have to **regenerate** service classes.

The following attributes are available in the *Authentication* section:

- Use** If this check box is set, then authentication is used to access the web service.
- Type** If the *Use* is checked, then here you can choose authentication type to use. Currently HTTP Basic, HTTP Digest and NTLM is supported.
- Username** Name of the user that will be applied to get access to this web service. You can use IvyScript in this field.
- Password** Password that will be applied to get access to web service. IvyScript is also interpreted in this field.
- Host** The host is used for NTLM authentication only. IvyScript is also interpreted in this field.
- Domain** The domain is used for NTLM authentication only. Specify the Windows domain of the user that should be used to authentication the user. IvyScript is also interpreted in this field.



Tip

When you specify authentication in a Web Service process element, these settings (Type, Username, Password, Host, Domain) will be overridden. Use Scripts like `in.user` carefully, since you might use this web service configuration in multiple processes with different data types.

The following attributes are available in the *Session Handling* section:

- Session handling** You can choose the session handling mode for current web service configuration. There are 3 modes available now:
 - **NO** there will be no session handling when you use this configuration
 - **WSELEMENT** invoking the same service from the same Web Service process element existing sessions will be reused
 - **APPLICATION** Within ivy applications whenever you invoke the same service existing session will be reused

The following attributes are available in the *Ports* section:

Ports The list of ports will become available when you have already generated web service client classes. (see: Generate WS classes)

The content of this list comes from the WSDL specified above, and being interpreted by Axis. So that your web service provider/architect can give more information about.

A port type is something like a group of operations according to some technical or business criteria. It has got nothing to do with TCP/IP, UDP or firewalls. Port types quite often reflect only SOAP versions used to access the same operations. In such case you normally see your port names ending with SOAP12, SOAP13, etc. If so, select the latest (biggest number)

Default endpoint

A URL is expected to be specified here. This is the URL where the current web service is located, and going to be accessed from. After client generation you might see URL-s here, because they can be specified in WSDL. If not, you have to set it up on your own. Value of this field depends on the "Port" selection, so that when you select another "Port" you might see another URL in this field.

Fallback endpoints

A list of endpoints can be specified here. They are used as fallback endpoints if any error happens during the web service request. First, the server at default endpoint will be requested. Servers on the list are queried one by one until a successful web service access can be made. You find error messages in runtime log when some endpoint invocations fail. If a service invocation is successful then the process continues as normal. This list is optional. When some servers are specified, then they are used, when not, then only the server at default endpoint will be accessed. If not even the default endpoint is specified then an exception is raised. (process continues with error handling). Like default endpoints the fallback endpoints depend on the "Port" selection.

Environments

In the configuration editor you can define different Environments. The environments can be added to existing web service configurations. Depending on the active environment the associated environment configuration will be used. If no environment configuration for the web service is available, the default configuration will be used. You can now define one web service with environment specific connection properties. For example it is possible to define different web service endpoints for your development, test and production environment.



Note

The web service configuration acts as the default configuration and must always be set



Tip

You don't must define environment configurations for every web service. If the web service configuration is the same for all environments, only the default configuration is necessary.

Roles and Users

Role is a widely used term in the computer industry and means a group of users of a certain system which share a common property. This enables an administrator to define configurations for this groups (roles) of users at once instead of defining it for each and every user individually. Axon.ivy incorporates a sophisticated role and user model to support:

- Authentication - Who may login into Axon.ivy
- Authorization - Control who is allowed to do what
- Task assignment - Decide who has to perform a task in a workflow
- User dependent UI elements - Configure who can see and operate on UI elements

In the following two sections you will learn how to create, edit and remove roles and users and how to link users with roles and vice versa.

Role Concept

The hierarchy of the roles is built upon the principle of specialization. Each child role specialises its parent role(s) meaning that a role *Team A* always implicitly contains its parent roles. The role *Everybody* is the root for all roles, all roles specialize this role. For example in the figure below, a member user of role *Team A* also has the roles *Development* and *Everybody*.

Member Role

A *Member Role* can be added as a child of an existing *Role* and links to another existing *Role*. While resolving the role tree, to collect the specializations of each role, the *Member Role* is handled like a normal child *Role*. Basically this simplifies the configuration and administration of roles.



Tip

The concept with *Member Roles* allows to create a sub-tree of roles with 'company roles' and a sub-tree of roles with 'permission roles'. The sub-tree of 'company roles' represents typically the structure of the company. The sub-tree of 'permission roles' represents the use or execution permission of a specific part or feature of an application. With a *Member Role* it is possible to assign a permission to a 'company role' by linking the 'company role' as a member role of a specific 'permission role'.

The following configuration illustration that users of group *First Level* and *Team B* have the permission for *Process N* and users of both support groups and of the *Support Group* itself has the permission for *Process M*.

```
+ Everybody
  + Support Group
    + First Level
    + Second Level
  + Development
    + Team A
    + Team B

+ Application Permissions
  + Process M
    - First Level (Member Role linked to role 'First Level')
    - Team B (Member Role)
  + Process N
    - Support Group (Member Role linked to role 'Support Group')
```

Role Editor

The role editor allows to create a new role and to edit and/or to remove existing roles and to structure them into a hierarchy. It can be started by double clicking on the *Roles* node in the Axon.ivy Projects view.

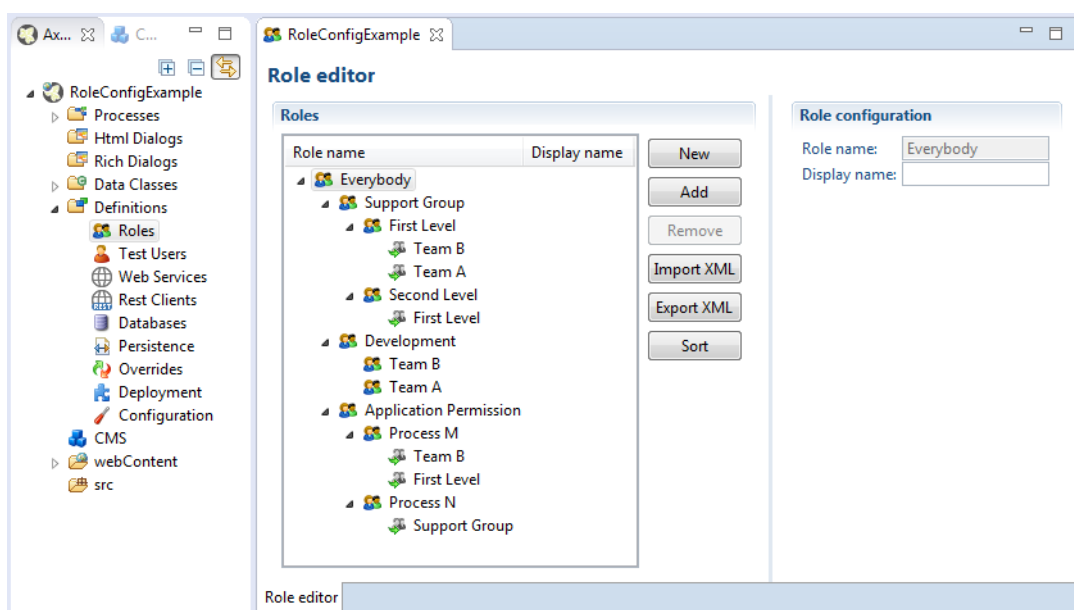


Figure 8.5. The Role Editor

The left side of the Role editor consists of a tree showing all the roles in a hierarchy order. A click on one of the roles will show the properties for the selected role on the right side of the editor. The role hierarchy can be manipulated by dragging a role and dropping it at the new place in the hierarchy.

New

A new role is created as a child of the selected role.

Add

A role is added as a linked member role to the selected role.

Remove

The selected role and all its children roles are removed. The user is obliged to confirm the removal. Note that the role *Everybody* may not be deleted.

Import XML

A *.roleconfig* file from another project can be imported for convenience reasons.

Export XML

All roles are exported in a XML file with the extension *.roleconfig* to easily re-use the role hierarchy.

Sort

The selected roles are sorted alphabetically, if desired this is performed recursively on the children.



Warning

Roles created in the designer are uploaded to the Axon.ivy Engine not until the deployment of the project. They are merged with all other roles in the same application context. Consider that the deployment will fail if the same role exist in more than one project in different role hierarchies.

Test User Editor

Process designers can create, edit and remove test users in the user editor. Users need a password in order to authenticate themselves and they need to be assigned to at least on role. The editor can be started by double clicking on the *Test Users* node in the Axon.ivy Projects view.



Note

Test users are only used in the process **simulation** within the Axon.ivy Designer and they are not uploaded to the engine at the deployment. Users for deployed processes on the Axon.ivy Engine need to be created and configured on the engine.

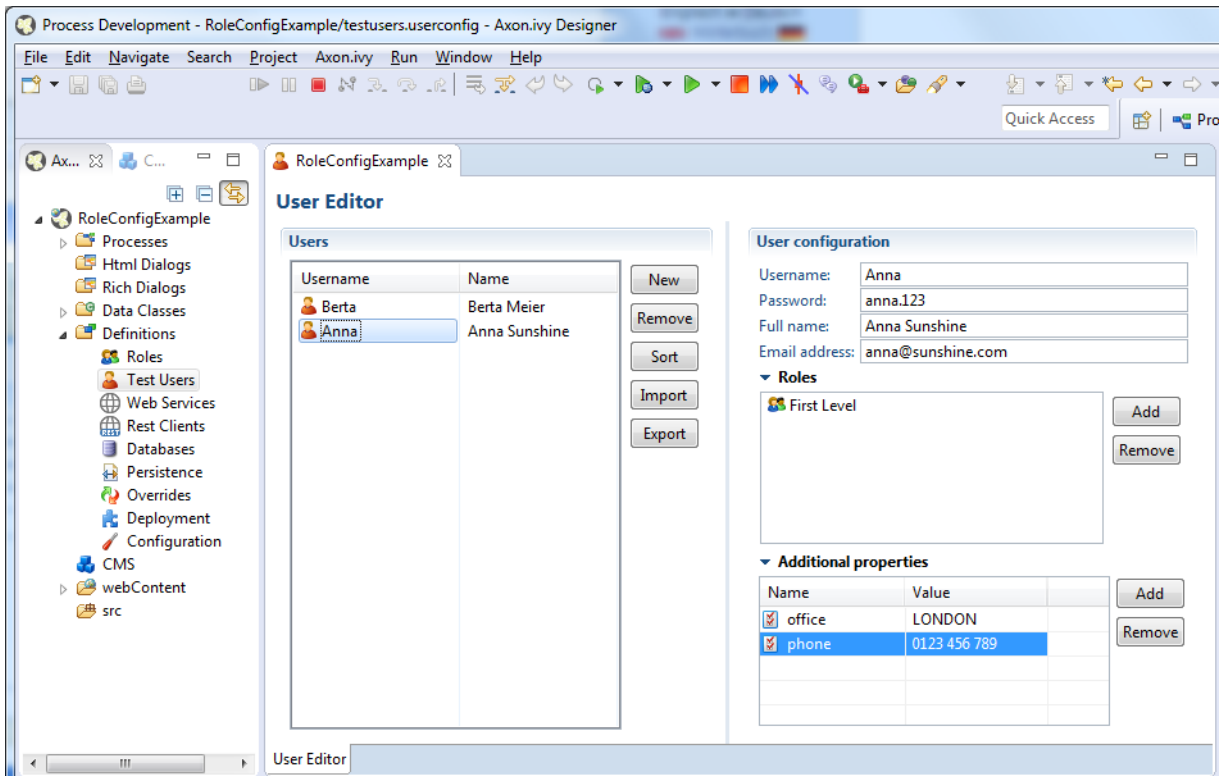


Figure 8.6. The User editor

The left side of the editor you see all users with their user names (i.e. login name) and their full names. On the right side the specific properties for the selected user on the left side is displayed. A user can be assigned to several roles and he is implicitly assigned to the parent roles of the roles, he explicitly is assigned to (see Role Editor for more precise details). Additional properties can be added to each user in terms of string key-value pairs and be reused within process steps in IvyScript.

New

Creates a new user with the specified name.

Remove

Removes a user from the list. The process designer is obliged to confirm the removal

Sort

Sorts the user list in alphabetical order

Import

A *.userconfig* file from another project can be imported for convenience reasons

Export XML

All users and their corresponding properties are exported in a XML file with the extension *.userconfig* to easily re-use the users in another project.



Tip

You do not see the user called *Developer*, because it is a built in user, that belongs to each groups, and owns all rights. The user is meant to be used for testing, so that it exists only in the Designer. The password of this user is *Developer* (in case you would like to log in using IvyScript)

Configuration files

This chapter deals with configuration files of the Axon.ivy Designer

Introduction

Configuration files are located in the *configuration* folder under the *%AXON_IVY_INSTALLATION%* folder. These files you have to edit out of Axon.ivy, e.g. using a system editor. It is recommended to stop your designer before you start changing these configurations.

There are some configuration files with extension *.xml. These are normal XML files. Meanwhile the *.any files are so called *any* files. There are a couple of any implementations - so that there is no real standard. The intension behind this is to have an easily to understand configuration with simple structure, and practically unlimited parameter types.



Tip

It is recommended to make a backup copy before you start changing configurations.

log4jconfig.xml

Using this file you can change logging preferences. This files contains a Log4j configuration. Log4j is an open source framework for logging. You can change logging format, logging destination (file, console, etc.), logging level and control which packages should be logged on which level.



Tip

The official web page of Log4j is <http://logging.apache.org/log4j/> Meanwhile all around the Internet you can find many articles and documentations.

jnlpcfg.any

Using this file you can change properties of the JNLP file that will be sent to client computers to start an Axon.ivy Rich Internet Application.

`jnlpcfg.any` is divided into tree sections: `information`, `java` and `application-desc`

`information` Configure general information about the application like title or its vendor.

`java` Configure parameters used for the Java virtual machine on client side like memory parameters, additional jvm arguments and the download-mode of jars.

`application-desc` Configure technical attributes of the application like client-site logging, tooltip behavior or the rich dialog session timeout.



Tip

Have a look into the file `jnlpcfg.any`. The configuration options are described in detail there.



Note

By default, the lazy download of jars is set to false. This means, that all jars required by the application are downloaded at startup. With a good internet connection, this will not be a problem. The application will start quite fast. In case the client is connected by a slow internet connection, it makes sense to enable lazy download of jars. But if lazy download is enabled, it will cause an overhead in the security certifacte validation - each jar will be verified individually. This also takes a certain amount of time.

ulclogconfig.any

Using this file you can change server side logging properties for ULC components. You can configure log level and log file. Sub-configurations can be defined per process model that receives the first request. If no log-file is defined, then System.err will be used as output of JNLP file that will be sent to client computers to start Axon.ivy application.

Chapter 9. Concepts

Adaptive Case Management

Classic BPM processes have a clear flow that defines how the process is executed. Within these strict processes the involved user has limited possibilities to improve the process while executing. Optimizations and flow changes often require a long modeling and re-deployment round-trip. Furthermore the process could get cryptic because every rarely occurring special case has to be modeled. Therefore, the process does no longer clearly show the most relevant business paths. Welcome to the world of spaghetti BPM.

Today the user has the need to adapt the process flow during execution. Optional side tasks are required in addition to the normal process flow or a set of tasks must be skipped because of a special condition. This brings back the power to the user which has often more knowledge about the domain and the current context of the process. For instance an important information could be received from a phone call, but the workflow system has no knowledge about this analog information.

Adaptive implementation

Invoking optional processes

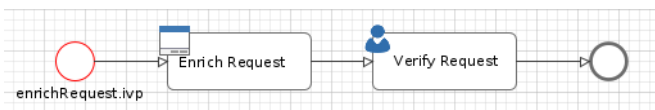
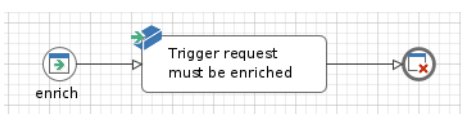
Think about a process where the purchase of an asset must be approved by a line manager. The line manager may want to ask the requester to provide more details why the asset is required. Therefore he'd add a side task to ask the requester for clarification. This optional interaction should not be wired into the main approval process as it obfuscates the most used business path. But it could be available as an optional side task that the line manager can start and then gets executed within the current process context.

In Axon.ivy processes with side tasks can be invoked through Triggers or Signals.

Triggers

It is possible to trigger a strictly defined process. Strictly defined means that the calling process knows the target process as it has to be implemented in the same or a required project. RequestStart events can be declared as trigger-able. While the Trigger activity is used to actually trigger such a start.

So in the request verification front end, a manager could simply trigger the process to enrich the request with a trigger call activity.

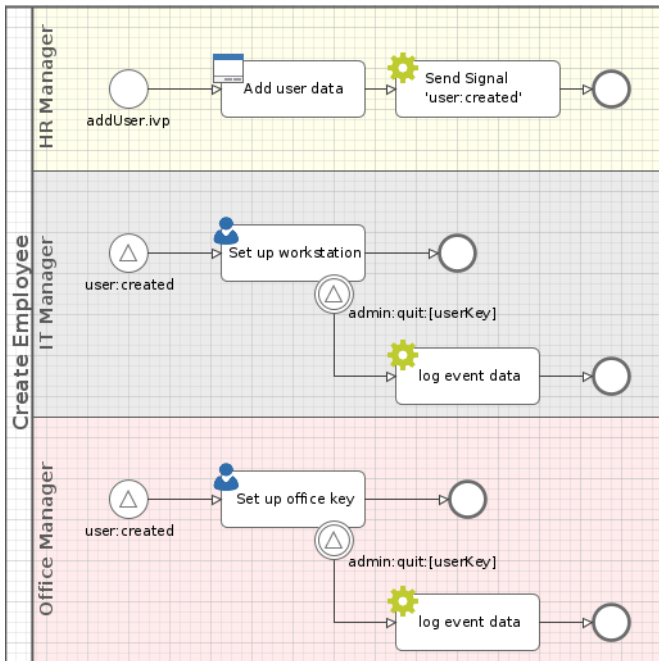


Signals

Most of the time you'd prefer a looser coupling between processes. This could be accomplished with Signals. A process that wants to integrate other processes simply fires a signal when a certain state within the process is reached. Multiple other processes in the same application could listen to this signal and all of them will be executed as soon as the signal is fired. A dependency between the firing and listening processes is not required.

As example think of an employee that starts to work in a company. When the employee is registered from HR, other processes can setup the environment for this employee. An IT responsible will setup a new desktop workstation while an

office administrator will get the personal keys for the employee. To do this tasks in parallel and loosely coupled signals are the first choice. The IT- and the office-process could listen to employee entry signals fired by the HR process.




Keeping loosely coupled processes in same context

A real world agile process execution can touch many different processes. But still the history and the context must be clear for anyone who is involved in a task. So the workflow needs to know whether an invoked process belongs to the invoking case. Or if the invoked process belongs to a completely new case.


The entity that can glue multiple process cases together is the Business Case. All cases and tasks that belong to the same Business Case are presented to the user of a workflow screen as related cases. Therefore, triggerable- and signalable-process start must define whether they belong to the same Business Case as the invoking process case. This can be done with a simple configuration on these starts. See the SignalStart and TriggerStart inscription for details.

Inscribe Signal Start Event

user:created 

Name Signal Output

Signal to receive

Signal Code 

Case

Attach to Business Case that signaled this process

You can also use the Public API to attach the current case to an existing Business Case.

```
if (in.departement.equals("HR")) // evaluate attachment by runtime conditions
{
    ivy.case.attachToBusinessCase(in.callerCaseId)
}
```

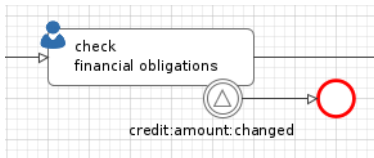
For workflow front end developers there exists API to list all tasks or cases of a Business Case. So showing the involved cases and tasks to a workflow user is a simple implementation. For more details see the Public API of `ch.ivyteam.ivy.workflow.businesscase.IBusinessCase`.

```
ivy.case.getBusinessCase().getActiveTasks(); // get involved tasks that are active
ivy.case.getBusinessCase().getTasks(); // get all involved tasks
```

Aborting tasks

A long running process could get into the situation that there are many open tasks that have to be done by human users. But eventually the environment of the case changes and it does no longer make any sense to accomplish the open tasks. For instance think about a car leasing process. If the customer decides shortly before contract signing that he requires leather seats instead of the furnished ones, the car will get more expensive. So the whole credit assessment process has to run again and open tasks become obsolete.

A UserTask can support abortion by listening to a signal. The UserTask activity can subscribe to an abortion signal by adding a Signal Boundary Event on it. When the signal, that the credit amount of the car changed, is fired from another process the listening UserTask will be aborted. And the process continues at the Signal Boundary Event. Classically after the Signal Boundary a clean up process follows.



Share data between processes

Often an initial larger process starts by gathering data that is later processed and enriched. This data is typically business relevant domain data that can be recognized by domain experts that contribute to the process. Think of bank employee that grants credits. The data for his processes could look like this when simplified:

IvyScript class 'CreditRequest' (workflow.credit)

Class
Specify the properties for the whole class.

Comment:

Annotations: BusinessCaseData

Attributes

Name	Type	Persistent	Comment
amount	Number	<input checked="" type="checkbox"/>	
reason	String	<input checked="" type="checkbox"/>	
salary	Number	<input checked="" type="checkbox"/>	
amountOfOtherCredits	Number	<input checked="" type="checkbox"/>	

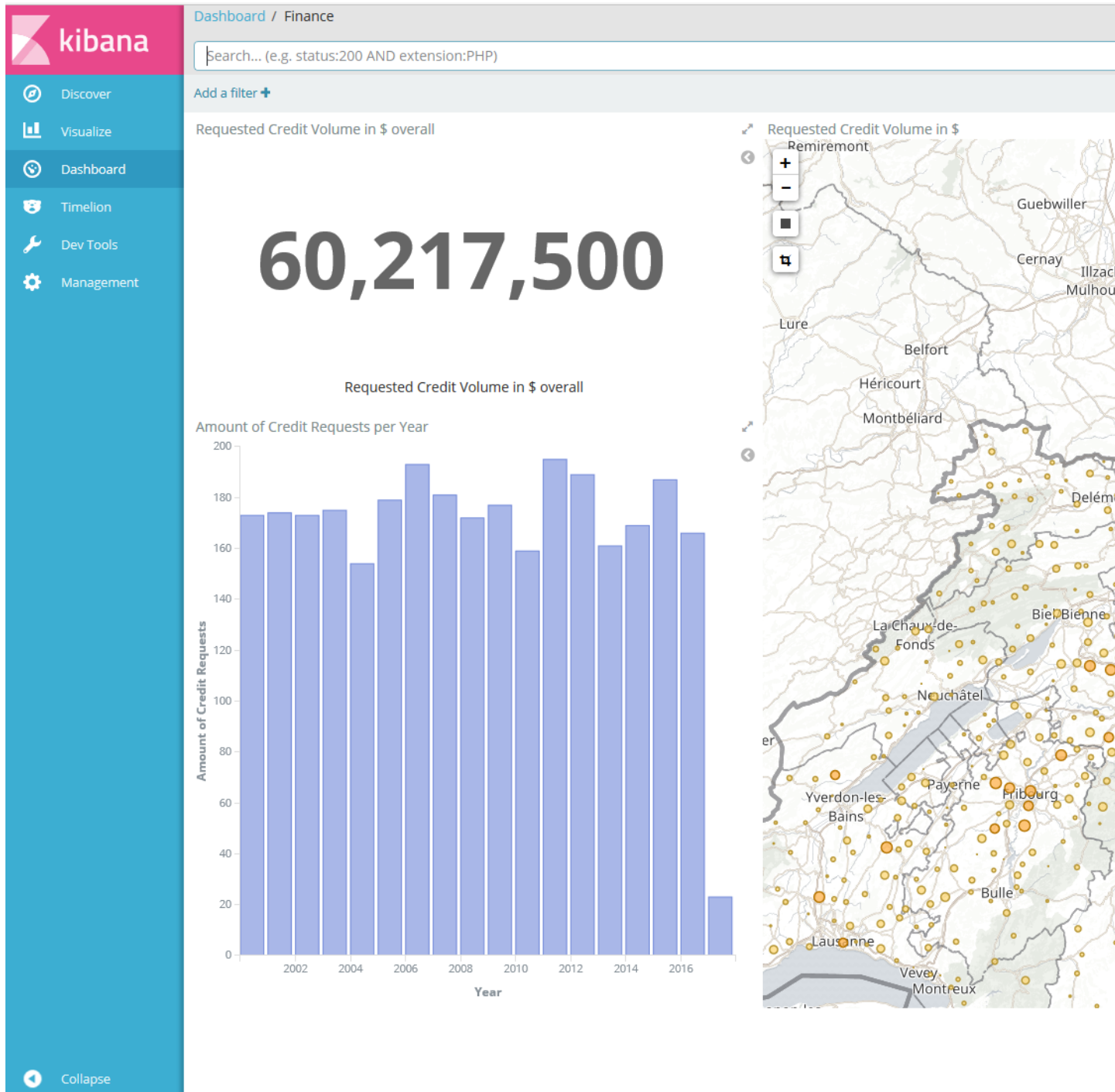
To store this kind of data Axon.ivy provides a simple repository that is called Business Data. This stored data can then be accessed by multiple processes instances during the lifetime of a long living complex process. The repository provides access to the data with simple store and load functions similar to well known other repositories such as the EntityManager from JPA. But in comparison to JPA and similar technologies this repository can be used without any database or environment configuration.

```
CreditRequest creditRequest = ivy.repo.find(in.businessDataId, CreditRequest.class); // load
creditRequest.amount = 30000; // modify a field
ivy.repo.save(creditRequest); // save the modified CreditRequest back to the Repo.
```

By annotating a data class with the `@BusinessCaseData` annotation, all values of the annotated data class are automatically associated with the context of the current business case. The data is then shared and accessible from all processes belonging to the Business Case. Multiple data classes of different types can be used inside a business case.

Business data analytics

Running business processes typically generate highly valuable data, that could influence critical business decisions. Based on the store data you will typically want to visualize KPIs on a management dashboard. In our credit sample, you may want to visualize the aggregated sum of all open credits. The data in the business data repository is stored in form that is easily accessible and explorable with a tool like Kibana.



Regaining the big picture

Real world BPM projects have shown that big processes tend to get increasingly complex and need to be split up into huge process landscapes, which leads to an intransparent main process flow. Users of the process often do not see how their work contributes to the bigger business process and therefore great opportunities for improvements are not taken. There is also a big

need for a unique view of adaptive case operations that can be used by process contributors. Like an overview of optional tasks that a clerk can start at any time.

The case map addresses the needs for flexible and agile business cases by providing a clear and simple view on the main process and its execution. With the Case Map you can easily orchestrate the main flow of processes and the business can identify and track the stages where a running process instance is.

Lending (Case Map)

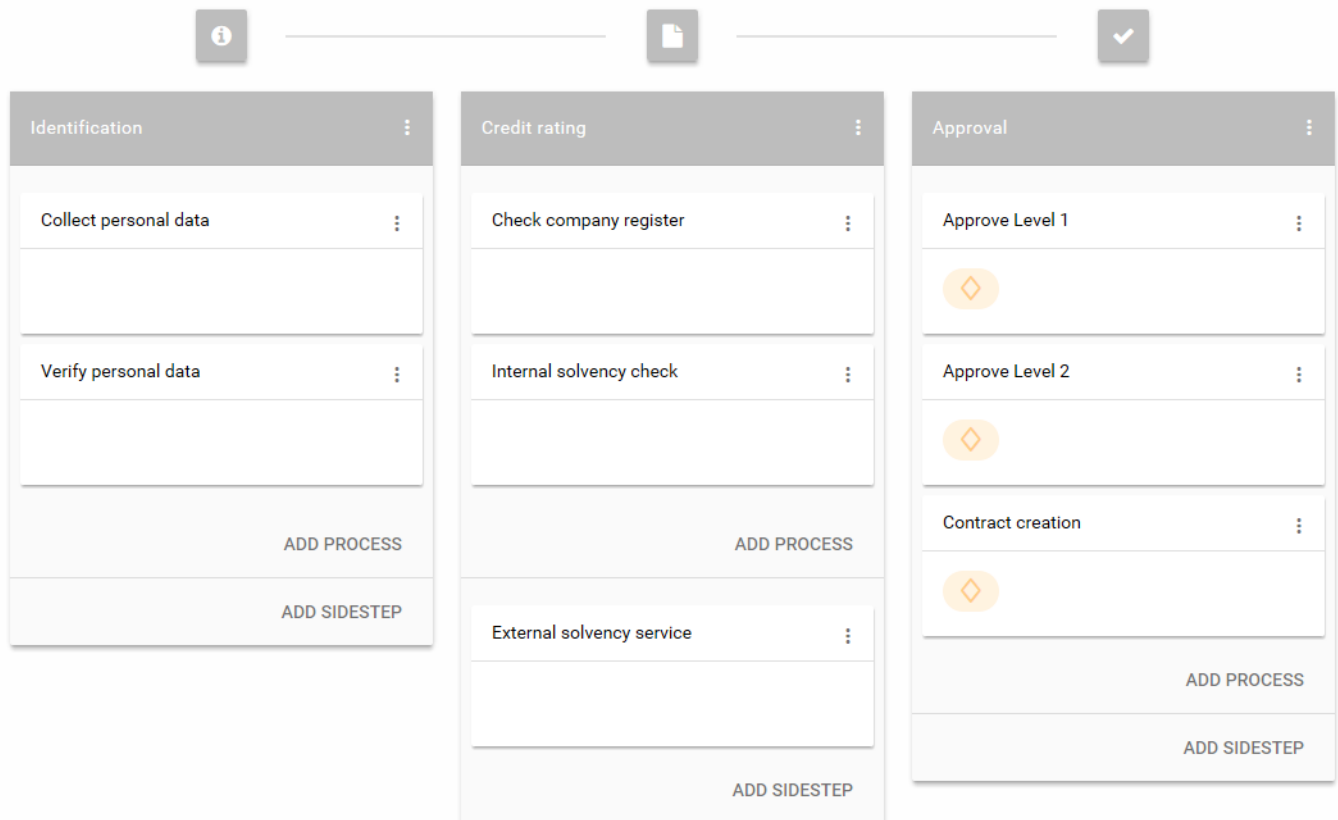


Figure 9.1. Case map of a lending process

A case map is divided into stages (in the sample above the stages are: Identification, Credit rating and Approval). Each stage defines a certain phase in the life cycle of a business process. A stage consists of processes (e.g. "Collect personal data"). The default flow or also known as happy path is from left to right and from top to bottom. If the last process of a stage has finished the flow continues on the stage to the right of the current stage. Stages typically have a name and icon. The idea is to reuse this icons in Workflow UIs and processes to give the end user a hint in which stage the current business case is.

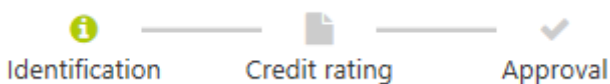


Figure 9.2. Actual stage of a business case in a case map

Besides processes a case map can contain side steps (e.g. "External solvency service" in the sample above). Side steps are discretionary processes. They can be defined within a stage. Side steps can be started manually during the ongoing business

case. A typical side step could be a process which aborts the business process (e.g. abort request). The use of side steps can reduce the time spent on technical round trips, for modeling rare and costly edge cases.

Processes started inside a case map create new cases inside the business case, which themselves contain tasks for the users. Data between processes can be easily shared using Business Data.



Figure 9.3. The relationship between Business Case, Business Data and Case Map.

Adaptive case management possibilities

To reiterate: signals and triggers can be used to loosely respectively tightly couple different processes. The innovative Case map brings order in to chaos of spaghetti BPM. A domain expert always has a simple graphical view on the business case where he contributes to. The case map empowers the domain expert to steer the process execution by starting optional side steps or gracefully skipping large parts of the pre-modeled standard flow.

The case map gives the developer and the user a common language to talk about a complex process landscape. The case map can be read and understood by anyone that contributes to the process without an introduction. This brings back the old BPM ideas that stood the test of time.

Signal Reference

Signals inform an unknown number of recipients of a specific event that has happened. Signals are sent application-wide without the need for project dependency between the sender and receiver.

Sending Signals

A Signal can be sent programmatically and consists of a Signal Code and optional signal data.



Note

Signal codes are defined as strings. Only letters and numbers [A-Za-z0-9] as well as the colon : as separator, are allowed characters for a Signal Code.

Valid: hr:employee:quit, flight:cancel:no:LXL398

Invalid: flight-cancel, LXL398, hr:employee:quit:, flight:cancel::no:LXL398

Send a Signal programmatically

A signal with a custom signal code can be sent using the following IvyScript code:

```
import ch.ivyteam.ivy.process.model.value.SignalCode;
// send simple signal
```

```
ivy.wf.signals().send("datarepository:updated");

// send signal with reference
ivy.wf.signals().send("order: canceled:"+in.order.id);

// send signal with signal data
ivy.wf.signals().send(new SignalCode("user:created"), in.employee.name);
```



Tip

It is not recommended to use data classes as signal data as not all receiving projects might have access to these data classes. Better send an id which references an object in a database or send payload data that is encoded as string (e.g. JSON).

Send a Signal manually in the Designer

While developing a process it is possible to send a Signal manually in the 'Signals' page of the Designer Workflow UI.

Receiving Signals

Signals can be received by Signal Boundary Events and Signal Start Events. Inscribed signal patterns can contain wildcards (*). Signal Boundary Events can react to a signal pattern containing macros.

Signal Boundary Event

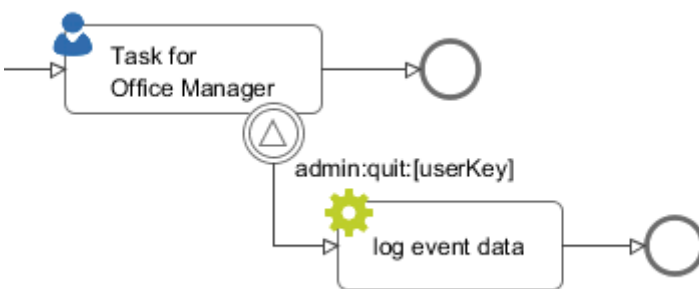
A “Signal Boundary Event” attached to a User Task Element destroys the task if a matching signal is received and the task is in SUSPENDED state (see also “Signal Boundary Event” in the Workflow chapter). The inscribed pattern on the Signal Boundary Event defines the filter for awaited signals codes:

Listening for a cancelled order signal with a specific id defined as macro:

```
order: canceled:<%=in.orderNr%>
```

Listening to any user created signals:

```
user:created:*
```



Signal Start Event

With a “Signal Start” a new process is started if a matching signal code is received.



Tracing Signals

Signals can be traced by either using the Designer Workflow UI or the JSF Workflow UI both Workflow UIs make use of the Public API for Signals (`ivy.wf.signals()`).



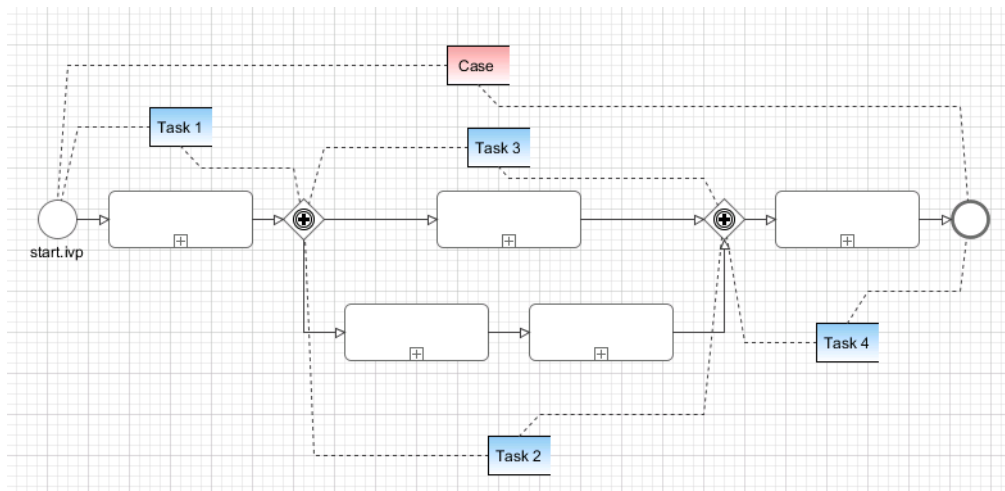
Tip

For debugging the signal data of a Signal event you can set a breakpoint on a Signal Start or Signal Boundary Event and inspect the signal variable in the 'Variables' view.

Workflow

Case and Task

The ivy workflow manages the execution of process instances. A process instance is represented through one Case and one or multiple Tasks. The Case exists from the first process step until the last process step and holds information of a process instance. When the Case gets finished even the process gets finished and backwards. A Case is processed through Tasks. Each Task defines an unit of work, which has to be done as one working step. Therefore a Task is assigned to a user or role which executes Task. A Task starts by a process-start element or a task-switch element and ends by the next task-switch element or an process-end element.



Business Case

Modern processes are loosely coupled and highly adaptive. Business processes can break out of the standard process flow and trigger asynchronous processes or send a signal that starts various other processes. As every running process creates a new Case instance it can get difficult for the workflow users to track the history and context of a task.

To clarify the workflow view, multiple Cases can be attached to a single Business Case. Triggered or signaled process-starts define in their inscription whether the started Case should be attached to the Business Case of the calling Case. Moreover, any Case can be attached to a Business Case by API.

Lifecycle

The first case of a process always acts as Business Case (Figure 9.4, “Initial Case”). If later an additional process case is attached to this initial case, this first initial case will be copied and treated as Business Case ((Figure 9.5, “Multiple Cases”)).

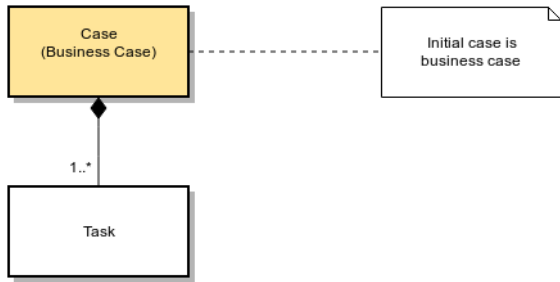


Figure 9.4. Initial Case

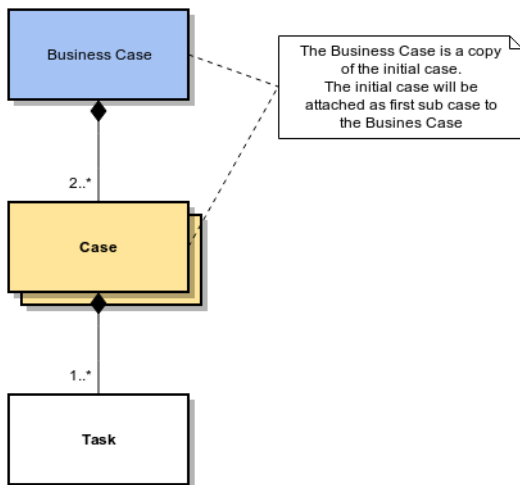


Figure 9.5. Multiple Cases

Case and Task Categories

A Case or a Task can be assigned to a category. A category is a structured String (e.g. Finance/Invoices) and categorize them into a hierarchical structure. It is beside the name of a Task (or Case) an important identification attribute of a Case or Task.

The Category API allows to get localized information from the CMS. E.g. the name of the category Finance/Invoices is stored in the CMS at /Categories/Finance/Invoices/name.

The following example shows a simple usage of a category on Case level. The API on Task level is identical.

```

ivy.case.setCategoryPath("Finance/Invoices");

String categoryName = ivy.case.getCategory().getName(); // EN: "Invoices", DE: "Rechnungen"
String categoryPath = ivy.case.getCategory().getPath(); // EN: "Finance/Invoices", DE: "Fi
    
```



Tip

The project WorkflowDemos demonstrates the usage of case and task categorisation. Typically the case category is used to categorize the over-all process (i.e. Business Case) and the task category is used to categorize a single or set of unions of work. Because the clear separation of case and task categorization even complex use cases could be handled.

E.g. in a midsized company the process to request an address from a customer change exists in multiple forms. There is one in the customer portal and one for partner agencies. The process executed from the customer

portal has the case category 'CustomerPortal/AddressChange'. The process executed by a partner agency has the case category 'Partner/Customers/AddressChange'. Both processes has involved a task to validate the address. Finally the address verification is done by the same department/user. So this task has in both cases the category 'AddressVerification'. This allows the user to filter those tasks no matter where they where created.

Workflow API

There are several APIs to manipulate and query workflow tasks and cases.

Task and Case queries

The fluent workflow query API makes queries against all existing tasks and cases possible. The queries can be written in a SQL like manor.

```
import ch.ivyteam.ivy.workflow.query.TaskQuery;
import ch.ivyteam.ivy.workflow.ITask;

// create a new query
TaskQuery query = TaskQuery.create()
    .aggregate().avgCustomDecimalField1()
    .where().customVarCharField1().isEqual("ivy")
    .groupBy().state()
    .orderBy().customVarCharField2().descending();
// resolve query results
List<ITask> tasks = ivy.wf.getTaskQueryExecutor().getResults(query);
```

To resolve all tasks that the current user can work on use the following code:

```
TaskQuery query = TaskQuery.create()
    .where().currentUserCanWorkOn()
    .orderBy().priority();
List<ITask> userWorkTasks = ivy.wf.getTaskQueryExecutor().getResults(query);
```

To execute a query an instance of a IQueryExecutor is needed. It can be retrieved trough the ivy environment variable.

```
// Application specific query executors can be retrieved from the application context
ivy.wf.getTaskQueryExecutor().getResults(taskQuery);
ivy.wf.getCaseQueryExecutor().getResults(caseQuery);
```



Warning

Queries over all applications can be executed on the global workflow context. But queries that involve the current session could deliver useless results as users are not shared over multiple applications.

```
ivy.wf.getGlobalContext().getTaskQueryExecutor().getResults(taskQuery);
ivy.wf.getGlobalContext().getCaseQueryExecutor().getResults(caseQuery);
```

Task and Case manipulation

The API to manipulate tasks and cases is available trough the ivy environment variable.

- ivy.case (ICase): represents the current process under execution
- ivy.task (ITask): represents the user's current work unit in the process under execution.
- ivy.wf (IWorkflowContext): addresses all workflow tasks and cases of all users for the application under execution.
- ivy.session (IWorkflowSession): gives access to all workflow tasks and cases of the current user.

REST API

There is a REST API available that uses HTTP, JSON (application/json) as content type and HTTP basic as authentication method. Over this interface the following services are available:

- HTTP GET /ivy/api/{application name}/workflow/processstarts Returns all process starts that can be started by the authenticated user.
- HTTP GET /ivy/api/{application name}/workflow/task/{taskId} Returns the task with the given task identifier.
- HTTP GET /ivy/api/{application name}/workflow/tasks Returns the tasks the authenticated user can work on.
- HTTP GET /ivy/api/{application name}/workflow/tasks/count Returns the number of tasks the authenticated user can work on.
- HTTP GET /ivy/api/{application name}/engine/info Returns the version and the name of the engine

Workflow States

During a process execution the corresponding case and tasks have various states. Normally, a case is started non persistent. This means it is stored in memory only. As soon as the process hits a task switch the case and its tasks will be made persistent by storing them to the system database. Only persistent cases and tasks can be resolved with the query API's above.

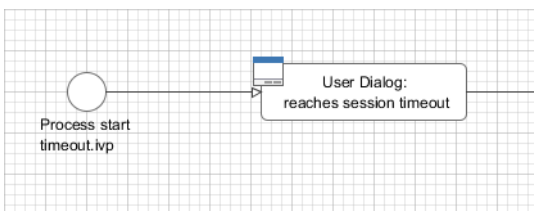
Process without Task switch



	Process start	Process end
Case state	CREATED	DONE
Task state	CREATED	DONE
Persistent	NO	NO

Table 9.1. Process without Task switch

Process with session timeout

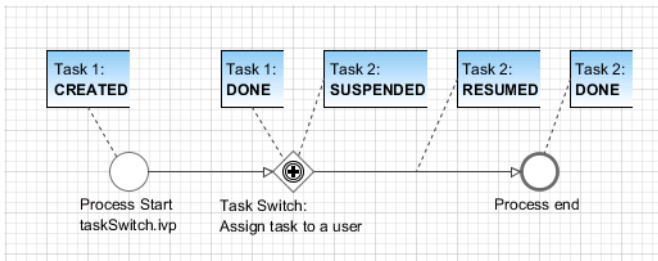


	Process start	User Dialog
Case state	CREATED	ZOMBIE

	Process start	User Dialog
Task state	CREATED	ZOMBIE
Persistent	NO	NO

Table 9.2. Process with User Dialog that reaches a session timeout

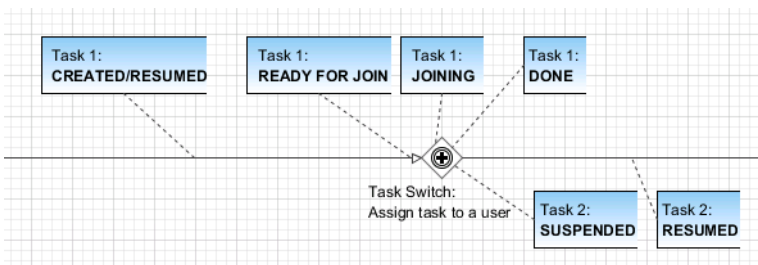
Process with Task switch



	Process start	Task switch	Process end
Case state	CREATED	RUNNING	DONE
Task state (Task 1)	CREATED	DONE	
Task state (Task 2)		SUSPENDED	DONE
Persistent	NO	YES	YES

Table 9.3. Process with Task switch

Task switch states in detail

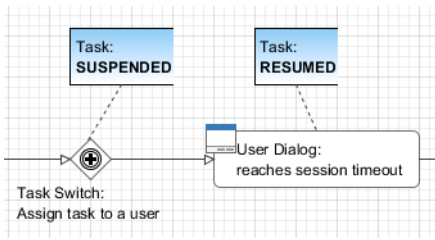


In detail the tasks are going to more technical task states inside of a task switch element. After a task reaches a task switch it is in state `READY_FOR_JOIN`. As soon as all input tasks have arrived at the task switch the state of all input tasks are switched to `JOINING` and the process data of the tasks are joined to one process data that is used as start data for the output tasks. After joining the input tasks are in state `DONE` and the output tasks are created in state `SUSPENDED`.

	Before switch	Task switch (reached)	Task switch (entry)	Task switch (done/output)	After Task switch
Case state	CREATED/ RUNNING	RUNNING			
Task state (Task 1)	CREATED/ RESUMED	READY_FOR_JOIN	JOINING	DONE	-
Task state (Task 2)	-	-	-	SUSPENDED	RESUMED
Persistent	NO/YES	YES			

Table 9.4. Process with Task switch

Task with session timeout

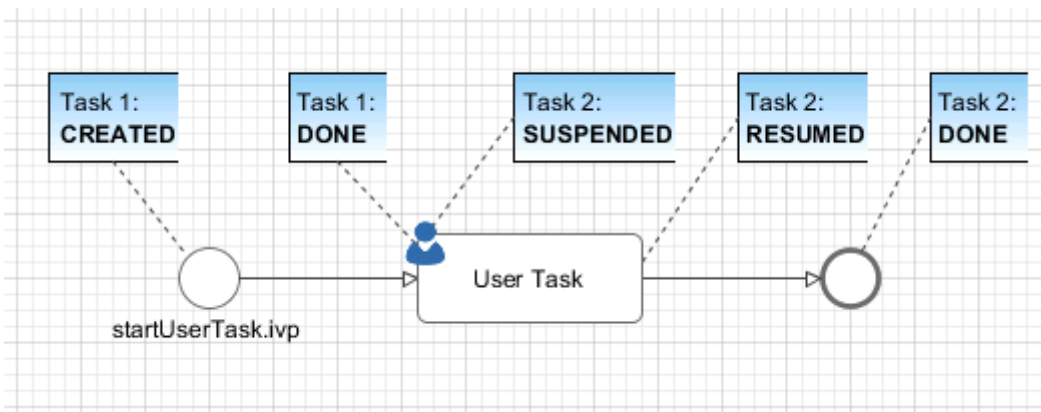


If a user resumes a task with an user dialog and then the session of the user timeouts then the task state is set back to state **SUSPENDED** and the process of the task is set back to the task switch element.

	Task switch	User Dialog	Task switch (after session timeout)
Case state	RUNNING	RUNNING	RUNNING
Task state (Task 1)	SUSPENDED	RESUMED	SUSPENDED
Persistent	YES	YES	YES

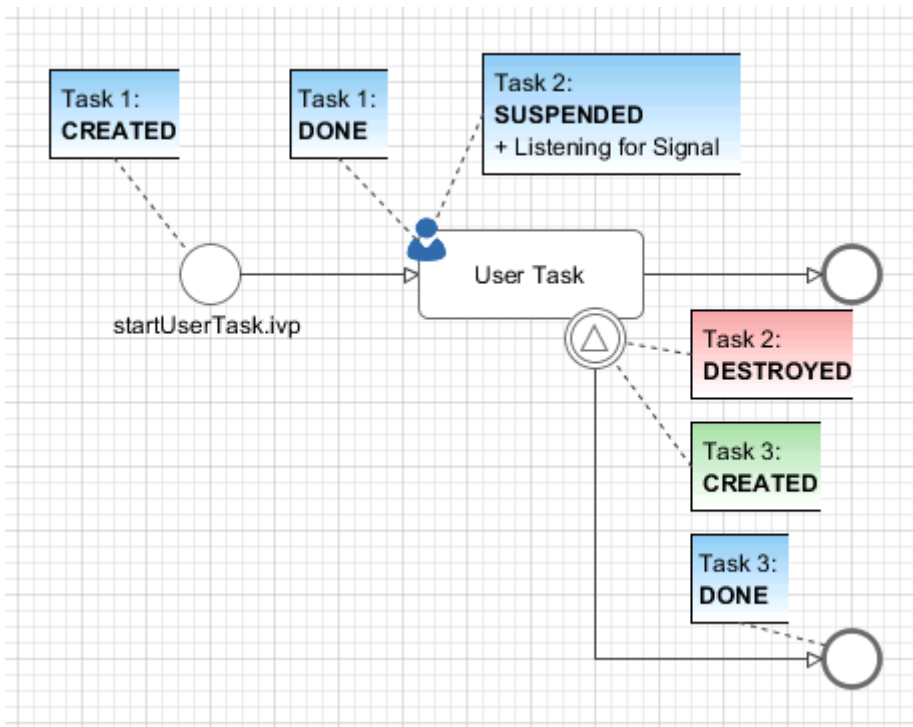
Table 9.5. Task with session timeout

User Task



A User Task is the combination of a Task Switch Event and a User Dialog. When the user start working on a normal Html User Dialog the task changes its state to **RESUMED**. In case of an 'Offline Dialog' the task state is not changed before the user submits the task form. Then the state changes from **SUSPENDED** to **RESUMED**. Subsequent steps are executed until the task is finally **DONE**. See also "Offline Tasks".

Signal Boundary Event



A User Task with an attached Signal Boundary Event is listening to a signal while its task is in SUSPENDED state. If the signal has been received the task is destroyed and the execution continues with a newly created follow-up task.

Other task states

There are more task states mainly for task synchronisation, error handling, intermediate events, or user aborts. To learn more about task states see enumeration `ch.ivyteam.ivy.workflow.TaskState` in public API.

Offline Tasks

Offline tasks are designed for use on a mobile device without connection to the Axon.ivy Engine. Typically the task data and the task forms are loaded during the synchronization with the Axon.ivy Engine and then handled locally by an app on a mobile device (e.g. a smartphone or a tablet). When the form is completed, the mobile app will transfer the entered form data to the server as soon as the connection to the workflow server is back again. In turn, this will resume the task and continue the process execution.

Offline Task in a Process

An Offline Task is generated, when the process engine executes a User Task element whereon an Offline Dialog is configured.

The User Task element provides a different task handling than normal task switches do. On a User Task, when the form data (the actual dialog page) is requested, the corresponding task remains in state *suspended*. The task state will be changed to *resumed* when the form data is submitted. Compared to this, a normal Task must be resumed first and after that, an Html Dialog element that follows in process flow, will return the form data.

Action	User Task element	Task Switch element followed by Html Dialog element
Task picked up from task list	<ul style="list-style-type: none"> Task state remains unchanged. Dialog page from configured dialog is returned. 	<ul style="list-style-type: none"> Task state changes to resumed. Process flow continues to Html Dialog element. Dialog page from configured dialog is returned.

Action	User Task element	Task Switch element followed by Html Dialog element
Form data submitted	<ul style="list-style-type: none"> • Task state changes to resumed. • Form data is mapped to dialog data. • Dialog is closed, dialog data is mapped to process data. • Process execution continues in context of the workflow user that submitted the form. 	<ul style="list-style-type: none"> • Form data is mapped to dialog data. • Dialog is closed, dialog data is mapped to process data. • Process execution continues in context of the workflow user that resumed the task.

Table 9.6. Comparison of the execution sequence



Tip

Offline Tasks can also be processed using a normal web browser as client. From a user's perspective they can be processed almost like normal tasks.

Because of the different task handling of a User Task element, the session can be interrupted/terminated after the form data was loaded. Then the form can be processed offline. After reestablishing the connection and creating a new session, the form data can be submitted. This would not work with normal tasks, since they are reset, as soon as the corresponding session (the one that resumed the task) expires.



Note

An offline aware application must manage the loading of the form data for required tasks, the presentation of the forms to the user during offline stages and the submission of the form data when the connection to the engine is established again. The ivy mobile app has full support for offline tasks.



Warning

Because the processing of an Offline Task may happen in parallel by several users, the task assignment should be set with caution. The form-submission of the first user will resume the task and continue the process. Subsequent form-submissions - from any user - will not be processed but responded with an error message.

Process elements that follow a User Task element will be executed in the context of the same task. An error during the execution of these elements will result as an error response to the form submission and the whole user task is set back to suspended.



Tip

Placing an Html Dialog element after an (Offline) User Task element is not a good idea, since it will not be handled correctly by an app that submitted the offline form. Generally it's best practise to place a task switch (Task Switch element of another User Task) as soon as possible after a User Task element.

Offline Dialogs

An Offline Dialog is a special kind of Html Dialog that warrants to be suitable for offline usage.

Ivy treats Offline Dialogs as separate view technology. Only when a User Task element is configured to use an Offline Dialog, it will generate Offline Tasks. Otherwise, normal tasks will be generated. From a technical point of view, an Offline Dialog is the same as a normal Html Dialog. They both are User Dialogs built on top of the JSF technology.

Even though there is no technical restriction - like a validation or similar - an Offline Dialog must omit any features that requires an active server connection before form submission. So, not all JSF features can be used. It's in the responsibility of the dialog developer to ensure the offline capability when developing an Offline Dialog.

Restrictions for the design of offline capable dialogs:

- The view should not rely on server side state e.g. session attributes, because it is executed solely on the client.
- The dialog data fields defined as persistent are held in the client view state. So, these fields, together with fields submitted in the form, are available in the dialog logic (methods and events). All other data fields are only available in the start method to initialize the dialog. Anywhere else in the dialog logic, they will be set to null.
- Owing to the offline capability, Ajax requests to the server are not possible. E.g. auto complete, lazy loading
- The entered data should be validated **before** form submission (client side validation). If only server side validation is performed, the user will get a late feedback, expressed as synchronization error when switching from offline to online.



Note

The layout and the styling of an offline capable dialog should consider the client device where it will run. Most probably it will be embedded in an mobile app on a device with a small touch-screen.



Tip

To avoid Ajax on form submission, a p:commandButton can be configured with the attribute ajax="false":

```
<p:commandButton value="Proceed" actionListener="#{logic.close}" ajax="false" />
```

Geo Location

The mobile app sends the current position of the mobile device to the server. This information is then stored in a location services that are available on the user that has worked with the mobile app and the tasks that have been worked on the mobile app.

Get latest position of a task:

```
import ch.ivyteam.ivy.location.GeoPosition;
GeoPosition taskPosition =
    ivy.task.locations().search().findLatest().getPosition();
```

Get latest position of a user:

```
import ch.ivyteam.ivy.location.GeoPosition;
GeoPosition userPosition =
    ivy.session.getSessionUser().locations().search().findLatest().getPosition();
```

The location service can also be used to store additional locations:

```
import ch.ivyteam.ivy.location.GeoPosition;
import ch.ivyteam.ivy.location.ILocation;
ivy.session.getSessionUser()
    .locations()
    .add(ILocation
        .create(GeoPosition.inDegrees(47.171573, 8.516835))
        .withType("User Home")
        .withNote("My Home is my Castle")
    );
```

More information can be found in the Public API in the package ch.ivyteam.ivy.location. It defines the location service and types to create, store and manipulate location information and geo-positions.

Data Storage

Axon.ivy provides multiple possibilities to manage and store project specific data. This chapter provides an overview of all the possibilities with their advantages and disadvantages. Which one should be used depends from case to case.

Content Management

Stores static multi language content like labels, texts, titles, images.

More information can be found in the chapter Content Management.

Web Content Folder

Stores static web files (CSS, JavaScript, Images, JSF-Templates) used in HTML User Dialogs.

More information can be found in the chapter HTML content in the Web Content Folder.

Filesystem

Data can be stores in files. Access and management has to be implemented in the project itself.

HTML User Dialog Resources

Stores static web files (CSS, JavaScript, Images, etc.) that are only used in the HTML User Dialog.

Database

Stores and access data in an external database systems. An own database server is necessary and the database schema must be managed outside of Axon.ivy.

More information can be found in the chapter Db Step.

Persistency (Java Persistence API)

Stores and access data in an external database systems. An own database server is necessary. The database schema can be generated. JPA is a Java standard that is well documented and widely used.

More information can be found in the chapter Persistence.

Web Service

Stores and access data in external systems by using web services.

More information can be found in the chapter Web Service Call Step.

Global Variables

Stores simple name/value configuration pairs. A global variable can have a different value per environment. On the engine there is a UI to change the values of a global variable.

More information can be found in the chapter Global Variables.

Application Custom Properties

Stores simple name/value pairs. Good alternative for storing small amount of data instead using a database.

More information can be found in the Public API ICustomProperties.

User Properties

Stores simple name/value pairs per user. Can be used to store user settings.

More information can be found in the Public API IUser.

Rich Dialog User Context

Stores UI settings per Rich Dialog and user. E.g. store the last choosen entry of a combo box to preselect it when the dialog is opened again.

More information can be found in the chapter Rich Dialog User Context.

Summary

Concept	Overriding	Project Dependencies	Environment	Public API	Web Accessable	Designer UI	Engine UI	Knowledge
Content Management	yes	yes	no	yes	(yes)	yes	no	Novice
Web Content Folder	no	yes	no	no	yes	yes	no	Novice
Filesystem	no	no	no	no	no	no	no	Advanced
Html User Dialog Resources	no	no	no	no	yes	yes	no	Novice
Database	no	yes	yes	no	no	yes	no	Advanced
Web Service	no	yes	yes	no	no	yes	no	Advanced
Persistency	no	yes	yes	yes	no	yes	no	Expert
Global Variables	no	yes	yes	yes	no	yes	yes	Advanced
Application Custom Properties	no	no	no	yes	no	no	no	Advanced
User Properties	no	no	no	yes	no	no	no	Advanced
Rich Dialog User Context	no	no	no	yes	no	(yes)	no	Expert

Table 9.7.

Overrides

This chapter deals with the concept of overrides and describes the Overrides Editor and the New Override Wizard. Overrides can be used to selectively redefine components of required projects so that replacement components are used at runtime instead. This is often a desired feature if an existing (possibly abstract) project or application is to be tailored for a specific customer or installation.

The Concept of Overrides

Applications are often implemented as a general solution for a problem and consist of multiple (dependent) projects. For many installations or customer projects it is desirable that certain parts of such a generic solution may be redefined in the context of a specific installation or customer.

To permit this, Axon.ivy knows various concepts of context-sensitive redefinitions:

- Regular redefinition (e.g. for Content Objects and/or Configurations): Simply define an already existing artifact with the same name again in a different project.
- Redefinition with environments (e.g. for Databases, Web Services, Global Variables): Redefine values and properties of global artifacts depending on the execution context.
- Redefinition with overrides (e.g. for Rich Dialogs and/or Sub Processes): Define a replacement component for an already existing component.

This chapter only deals with the third category of artifact redefinitions (overrides).

By defining overrides on project level, the lookup of a certain component can be redirected to a replacement component. When a component is referenced in a process model of that project then the lookup for this component will yield a different component (i.e. the replacement) at runtime instead of the originally referenced component.



Warning

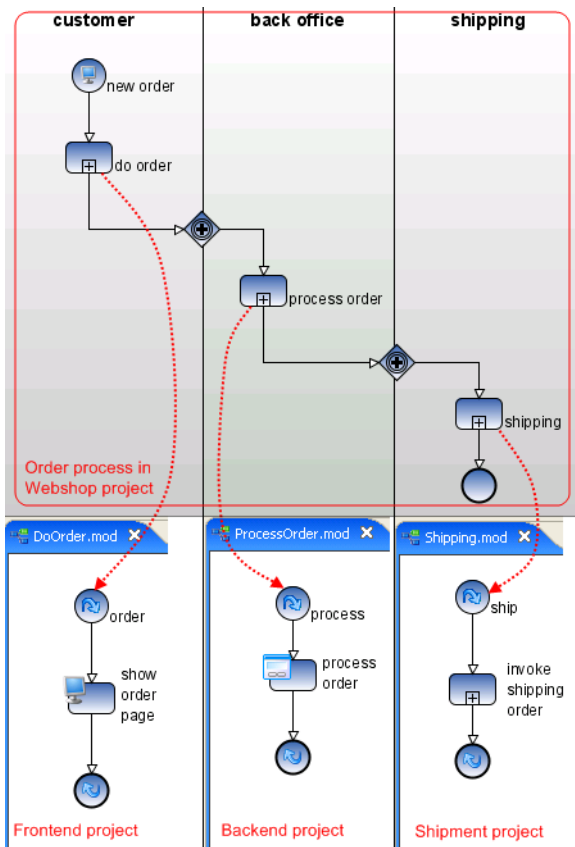
This happens completely independent of the original designers intention and will take place every time a component is looked up.

Case Scope

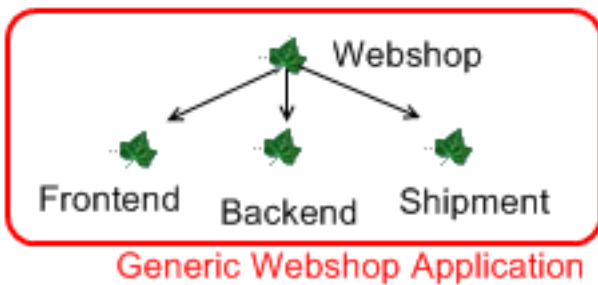
How is a component looked up? For the lookup of components at runtime, the so-called **case scope** is crucial. The case scope is determined by the project, in which the current case was started, e.g. where the start of the running business process was invoked. All component look-ups as well as configuration and content management references are processed within the case scope, i.e. the look up of such artifacts always starts at the project that defines the case scope.

Example: The Acme Web shop

As an example, imagine a web shop application. It contains the following (generic) business process:

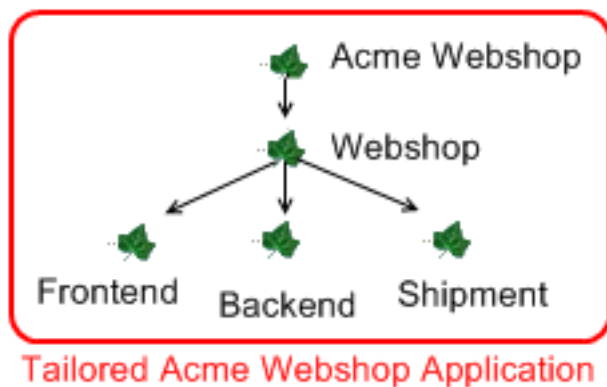


The main process itself (Order) and each of the depicted sub processes (DoOrder, ProcessOrder, Shipment) are defined in own projects. All of those projects together form a (generic) web shop application, depicted below. The web shop project contains the business process and it's start; the Frontend project contains the DoOrder sub process; the backend project contains the ProcessOrder sub process; the Shipment project contains the Shipping sub process.



In this scenario, regardless of the task that is currently being executed (customer, back office, shipping), the case scope will always be the web shop project, because the business process is started from there.

We now define an additional project, Acme web shop. The new project is dependent on web shop and the intention is to bundle all Acme-specific overrides and adoptions in this project. The already existing projects plus this new project form together a more specific and customized Acme web shop application, with the following project dependency tree:



If the main business process is copied from the *web shop* project to the *Acme web shop* project, and if it is ensured that the process request is issued through the *Acme web shop* project instead of the *web shop* project, then all tasks of an order case will consequently have *Acme web shop* as their case scope.

Knowing this, we can now specifically override and redefine Content Objects, Configuration entries, Rich Dialogs or Sub Processes from the original generic *web shop* application by redefining them inside the *Acme web shop* project. Afterwards, whenever a business process with case scope *Acme web shop* is started, then the overridden artifacts and components will be used instead of the original ones, due to the case-scope based lookup mechanism.

General Definition

The following figure illustrates the adaption of an application with overrides in a general way:

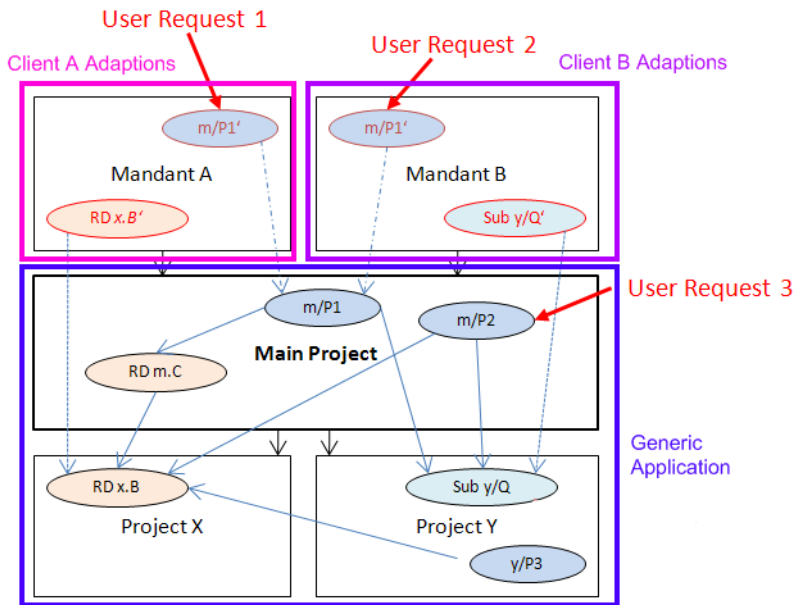


Figure 9.6. Adapting a generic application with overrides

It can be seen that multiple adaptations (Client A, Client B) may be created for a generic main project. Also, each adaptation may override different components.

Because Request 1 and Request 2 have different cases scopes, Request 1 (issued through the *Client A* project) will use the overridden Rich Dialog x.B' instead of the original x.B; Request 2, however, will use the original x.B Rich Dialog, because there is no redefinition within the case scope of the *Client B* project. Likewise the invocation of the Sub Process y/Q will result in the execution of the override y/Q' in Request 2, and the execution of the original y/Q in Request 1.



Note

If it should happen that the business process m/P2 is executed through the main project directly, then no overrides will be applied at all. Since such a "direct" invocation normally results in an unwanted case scope, it should be prevented. The easiest way to do so is the usage of a process facade as described below.

Process Facade

If the override mechanics are to work as intended, it must be ensured that processes are always and solely started from the adapted customer projects to ensure the proper case scope. This requires that all business processes (or rather their request start elements) must be copied to the adapter project.

To simplify this task and to reduce the work to the copying of a single file, it is recommended to employ the *process facade* design pattern.

Inside the main project of the generic application create a single process (e.g. Main) that holds the start elements of all the elementary business processes of the application. Factor the logic of those processes out into sub processes and call them from the facade process stubs, as illustrated below. With this approach, only one process (the facade) has to be copied to the top-level customer project.



Warning

When factoring out sub processes, please keep in mind that you should not use task switches in sub processes of required projects. As a general recommendation, any factored out sub process should roughly correspond to the contents of a task (or parts of such), but should not span multiple tasks.

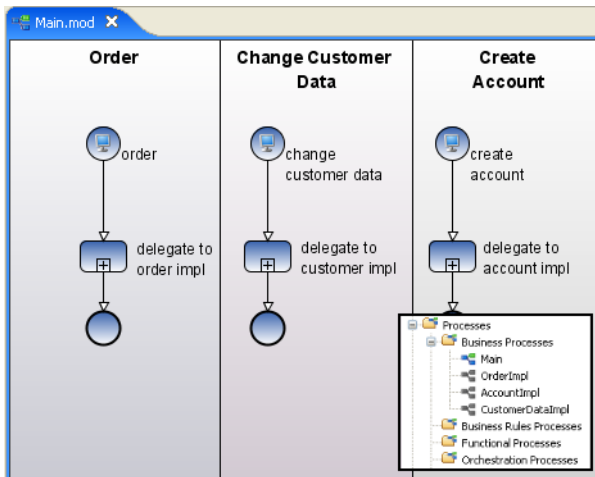


Figure 9.7. Implementing a process facade with process stubs

The portal website, the workflow UI or whichever other means that are used to start the application's business processes should only show the processes from the copied facade process. As all the out factored Sub Processes will also be available from the adapter project, no further changes have to be made.

Overrides Editor

Overview

The Axon.ivy *Overrides Editor* shows the registered and active overrides for a specific project. The overrides are listed in 4 different sections: Rich Dialogs, Sub Processes, Content Objects and Configurations.

Rich Dialog and Sub Process overrides require - for technical reasons - the registration of a mapping (this is done automatically by the New Override Wizard) which maps the original component's identifier to the replacement identifier. This mapping is displayed in the Override Editor and can be deleted by selecting an entry and subsequently clicking on the *delete* icon in the section's tool bar. When clicking on the *wizard* icon in the tool bar, a new override mapping of that category can easily be added by entering all necessary information into the opening wizard.

Overrides of Content Objects and Configurations, on the other hand, do not require a renaming and an extra mapping between the original and the overriding component. They are simply created by adding a new Content Object or Configuration entry with the name of a component that already exists in a required project. At runtime, the new component will be found first and thus shadow the original value. For this type of override no special actions are available from the editor; you should use the respective editors (Content Editor and Configuration Editor) to create or delete overrides. The editor shows the overrides of that type for reasons of a centralized overview and for convenience, rather than to provide an interface to edit them.

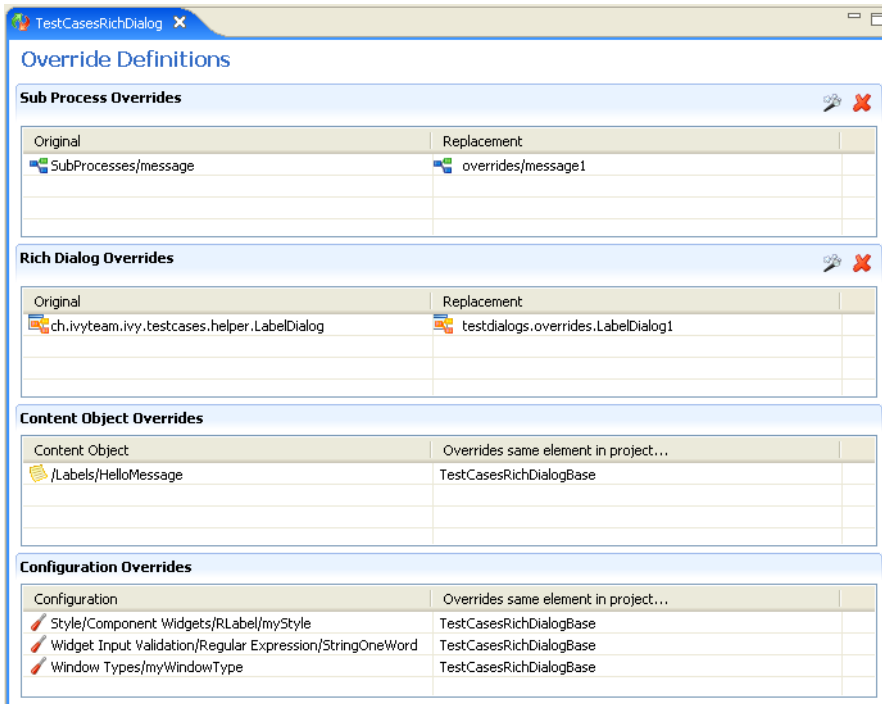
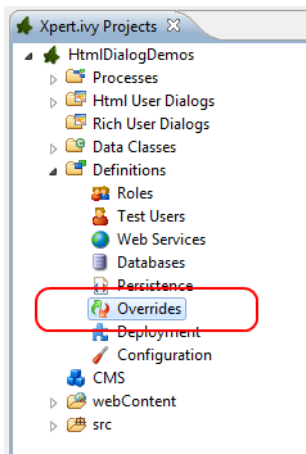


Figure 9.8. The Override Editor

Accessibility



Axon.ivy Project Tree > double click on the Overrides node.

Features

Sub Process Overrides

This section shows all Sub Process overrides that are registered for the selected project. You can delete an existing override by pressing the *delete* icon in the section's tool bar. This will only delete the mapping (and thus the execution of the override) but not the replacement Sub Process itself. You can add new Sub Process overrides by clicking on the *wizard* icon in the tool bar (this can also be used to "restore" a previously deleted mapping).

Rich Dialog Overrides

This section shows all Rich Dialog overrides that are registered for the selected project. You can delete an existing override by pressing the *delete* icon in the section's tool bar. This will only delete the mapping (and thus the execution of the override) but not the replacement Rich Dialog itself. You can add new Rich Dialog overrides by clicking on

the *wizard* icon in the tool bar (this can also be used to "restore" a previously deleted mapping).

Content Object Overrides

This section shows all Content Objects that are redefined in the selected project, i.e. the Content Objects for which there is an entry with the same URI in a required project. At execution time the redefined Content Object will be used.

You can delete overriding Content Objects directly from the list (multi-select a few lines and hit *Delete*) or use the Content Editor to add new overriding Content Objects.

Configuration Overrides

This section shows all Configurations that are redefined in the selected project, i.e. all Configurations for which there is an entry with the same name in a required project. At execution time the redefined Configuration will be used.

You can delete overriding Configuration entries directly from the list (multi-select a few lines and hit *Delete*) or use the Configuration Editor to add new overriding Configuration entries.

New Override Wizard

Overview

The *New Override Wizard* lets you create a new override. The wizard performs two tasks:

1. It will create an independent copy (snapshot) of the original component with a new name in the current project.
2. It will create and register a mapping <original,replacement> in the list of overrides that are known to the system. The list of those mappings can later be inspected and edited with the Override Editor.



Note

Please be aware that any Sub Process that is being overridden must have "use own data class" explicitly set in its inscription. The wizard will not let you create an override of a process if this is not the case, because the "use default data class" setting will result in a different data class inside the target project where the override will be created.

If the wizard refuses to create an override for this reason then you can set an explicit data class in the values tab of the original process's inscription mask.

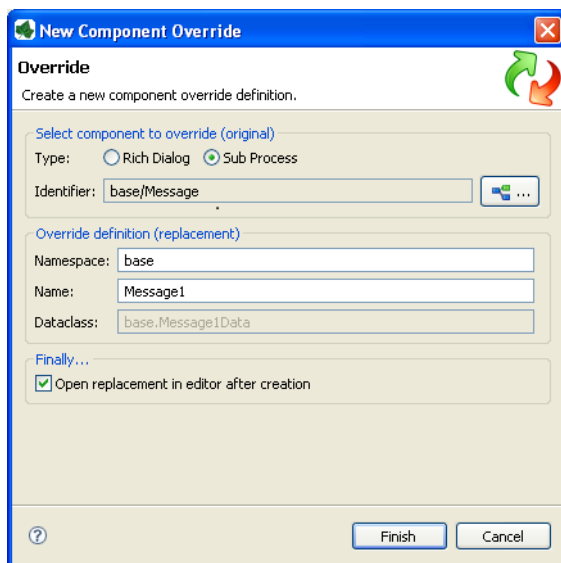


Figure 9.9. The New Override Wizard

Accessibility

File > New > Override

Features

Original Type	Choose the type of component for which an override replacement should be created (Rich Dialog or Sub Process).
Original Identifier	Specify the identifier of the original component that should be overridden at runtime. Use the button next to the text field to select from the available Rich Dialogs or a Sub Processes. Please note that only components from required projects can be overridden, there is no point in defining an override for a component in the same project (see override concept).
Replacement Namespace	Choose a namespace for the replacement component.
Replacement Name	Enter the name of the replacement component.



Note

If you create an override for a Sub Process, then a copy of the data class of the original component will be created (snapshot) and will be associated with the replacement process. The name of the copied data class will be inferred from the replacement component's identifier (namespace + name).

Finally...	Select whether you want the respective component's editor to open on the replacement component once the override has been created.
------------	--

Error Handling

Errors are used to model exceptional process paths. With an error the happy path of a process is left. An error is caught by an Error Boundary Event or Error Start Event if their Error Code pattern matches the thrown Error Code.

- Errors are divided into technical errors (e.g. database connection problem) or business errors (e.g. approval declined).
- An error is defined by an Error Code.
- The error may be caught by an “Error Boundary Event” attached to the activity or subprocess, or by an “Error Start”.
- An Error Boundary Event or Error Start Event with an empty Error Code catches every error.

Error Codes

Error codes are defined as strings. They can be refined by inserting a colon (:). Multiple sub Error Codes can be caught using wildcards (*). Trailing wildcards are optional so the string `custom:error` is the same as `custom:error:*`.

Example

If the error code `booking:failed` is thrown it can be caught with following error code patterns: `booking:failed`, `booking,*:failed`. Additionally it can be caught by an empty Error Code that catches all errors.

System Errors

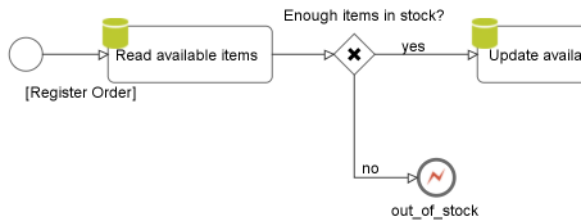
System errors are thrown by process elements like *Database Step* or *Web Service Call Step*. Their error codes are set by default and are prefixed with `ivy` (e.g. `ivy:error:database`).

Throwing Errors

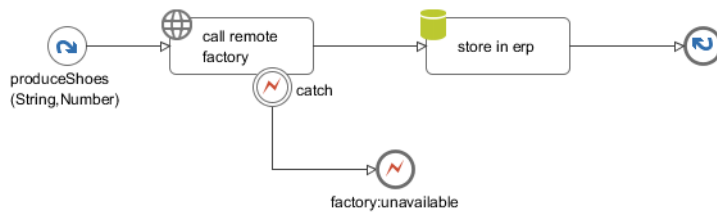
An error can be thrown explicitly by an Error End Event, or from code executed in IvyScript or Java. System errors (e.g. `ivy:error:database`) are implicitly thrown by the system.

Error End Event

The happy path of a process can be left by throwing an error with an “Error End”. (e.g. if an approval was declined). The Error End Event throws the error to upper process level, it can't be caught on the same process level.



Error End Events can also be used to re-throw a pre-defined ivy error with a specific error that has a meaning to the business. (e.g. if a webservice is not available)



Error handling in Html Dialog

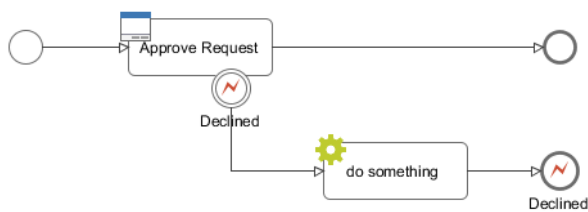
When an error happens inside of an Html Dialog the handling is slightly different than the default error handling.

Default Html Dialog Error Handling

Basically any thrown error (e.g. an Java exception) is handled inside of the Html Dialog itself. Therefore there is no propagation to the caller process or between Ivy/JSF composites. It is important to handle errors locally in the Dialog Logic to let the user work uninterrupted on the same dialog.

Exit an Html Dialog by an Error End Element

It is possible to exit an Html Dialog by an Error End Element. This is useful to leave the happy path of the calling business process. The throwing Error End Element must be located in the Html Dialog Logic of an Html Dialog Page (not an Component).



IvyScript or Java Code

Unhandled Script exception

If an unhandled exception occurs while executing IvyScript or Java code then the calling process element throws an error with the Error Code `ivy:error:script`. On the error object the causing Java exception is available as technical cause.

Throwing an error programmatically

An error with a certain Error Code can be thrown using the following IvyScript code:

```
import ch.ivyteam.ivy.bpm.error.BpmError;

BpmError.create("mystock:empty").throwError();
```

An error with a certain Error Code can be thrown using the following Java code:

```
import ch.ivyteam.ivy.bpm.error.BpmError;

throw BpmError.create("mystock:empty").build();
```

Elements throwing System errors

The process elements Program Interface, Database, Webservice and E-Mail throw system errors. If an exception or timeout occurs on these elements it can be caught using a matching Error Code or using a directly addressed Error Start Event. On the Error Start process element more information about the error can be accessed via the variable `error` and the legacy variable `exception`.

Catching Errors

Errors can either be caught by Error Boundary Events or Error Start Events.

An error is caught in the following order:

1. By an Error Start Event directly addressed in the element's inscription mask. (If available on the inscription.)
2. By an Error Boundary Event attached directly to the activity the error comes from.
3. By an Error Start Event on the same process level if not thrown by an Error End Event.
4. By an Error Handling on the next higher process level, starting there with step 2 until the top level process is reached.
5. By a Project Error Process in the top-level project.
6. If the error is not caught it is displayed to the user on the standard error page.

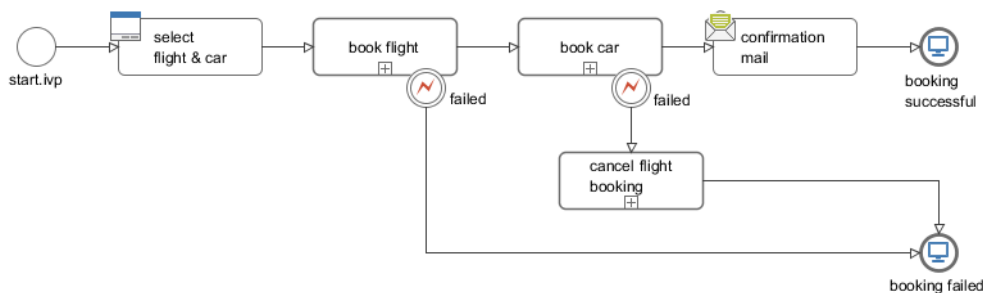


Note

Each process - including the embedded subprocess - is a separate process level.

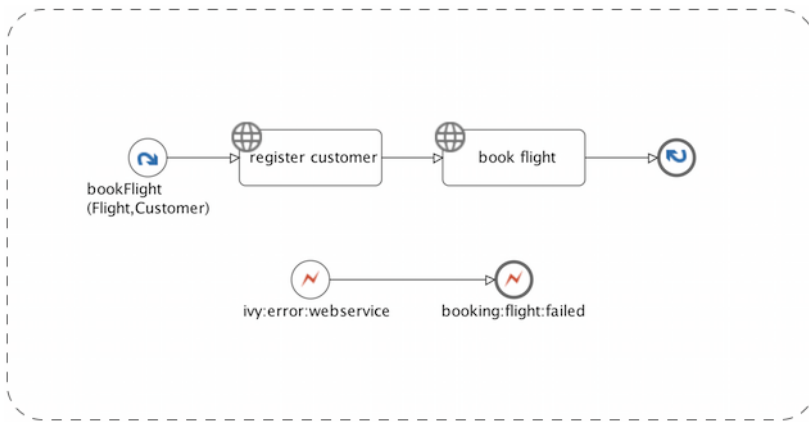
Error Boundary Event

An “Error Boundary Event” catches errors which were thrown from the attaching activity or subprocess if the configured Error Code matches the thrown error.



Error Start Event

An “Error Start” catches unhandled errors which were thrown in the same process or inside a subprocess if the configured Error Code matches the thrown error.



Loop Prevention

To prevent endless process execution through an inappropriate error handling, the ivy process engine detects loops during the error handling. If the engine detects a loop the error handling will be continued on the next higher process level with the new error code `ivy:error:loop`, to interrupt the cycle.

Loop detection is done on error catching elements (Error Start Event and Error Boundary Event). The engine checks if there was already an identical execution of the catcher at this process level. Identical means: Same process request, same throwing element (including its process callstack) and same catching element (including its process callstack).

Lets illustrate this with two use cases:

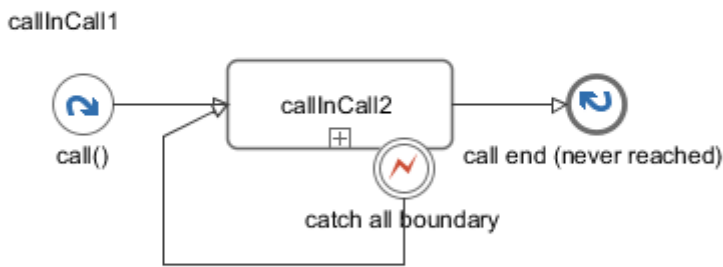
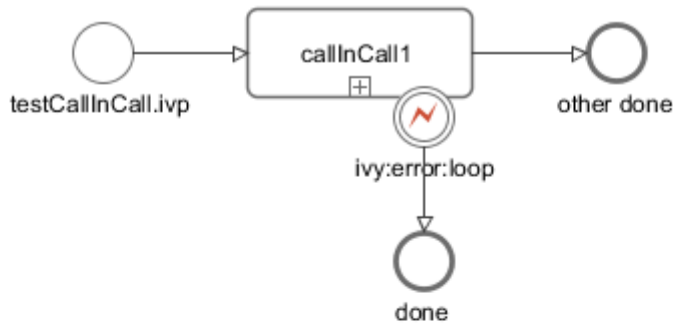
Use Case 1

The process element throws an `BpmError`. The Error Boundary Event will catch the error and call the process element again. In this case, the loop detection will interrupt the process when the Boundary Error Event was reached the second time. This would also be the case, when the throwing error element is located in a composite or callable process.



Use Case 2

In this case, the loop detection will interrupt the process 'callInCall1' after the second error handling. The process will be continue by the error handling on the caller process with the error code 'ivy:error:loop'. The process will end on the End Element named 'done'.



Project Error Process

A Project Error Process catches uncaught errors from the whole project. The name of a Project Error Process must start with **Error** and has to reside in the top-level process group *Processes*. It can contain one or more Error Start Events.



Note

The process data of the throwing process (i.e. the value of the `in` variable) is not available in the Error Start of a Project Error Process.

Error Object

The error object provides the following information about the error that was caught:

- Unique Error ID
- Error Code
- Technical Cause (Java Exception)
- Process element
- Process call stack
- User defined error attributes

For more information see the Public API of `BpmError`

Rule Engine

A Rule Engine is basically a software system that maintains and executes a given set of rules. More specifically, rules in a rule engine are usually described in a declarative way. Mostly in the notion of conditions and actions. Or in more developer friendly words, it's a bunch of if-then statements. For example, let's take a simple rule for computing the premium for a car insurance:

```
if owner.livesInDodgyArea then
  if car.price < 50000
    premium += 100
  else if car.price < 100000 then
    premium += 200
  else
    premium += 300

if owner.livesInBumpyStreetConditionArea then
  if car.type == SPORTSCAR then
    premium += 500
  else if car.type != SUV AND car.type != TRUCK then
    premium += 100;
```

You can imagine that with this approach you end up pretty soon into the Spaghetti-code anti-pattern. You have to take care of all the dependencies and relations between all the facts on your own leading to a forever-growing if-then statement that is almost impossible to maintain.

In a rule engine, you create simple standalone rules and you let the rule engine decide what to fire when. The subtlety is that rules can be written in any order, the engine picks the ones for which the condition is true, and then evaluates the corresponding actions. So instead of the massive if statement, you write the following rules:

```
if owner.livesInDodgyArea AND car.price < 50000 then premium += 100
```

```
if owner.livesInDodgyArea AND 50000 < car.price < 100000 then premium += 200
```

```
if owner.livesInDodgyArea AND car.price > 100000 then premium += 300
```

```
if owner.livesInBumpyStreetConditionArea AND car.type == SPORTSCAR then premium += 500
```

```
if owner.livesInBumpyStreetConditionArea AND car.type != SUV OR car.type != TRUCK then pre
```

Because of this simplification it might even be possible for non-developers to define or configure the rules (a little bit of tool support helps too).

In short, a rule engine helps you to decouple your production rules from the rest of the code and makes it much more maintainable for both developers and domain experts. But remember that everything has its flip side. Adding a rule engine means adding another level of complexity into your architecture (you replace plain code with a new system). And as the size of your rule sets grow, so does the potential impact to the performance. For more information about rule engines, please refer to one of the many available resources in the Internet.



Tip

In Axon.ivy, we integrate the open source rule engine Drools to give you the flexibility to use a rule engine if you want. We wrapped some of the most basic features of Drools into our own UI and API. If you need more than that, then simply use the normal full blown Drools API.

Decision tables and DRL files

We support two formats for defining rules: decision tables and rules written in the Drools Rule Language (DRL). A decision table is like the name says a table that can contain many rules. The columns usually make up the variables of the preconditions/ actions whereas each row in the table specifies one rule. Let's see the decision table for our example:

Pre-Condition					Action
lives in dodgy area	price min	price max	lives in bumpy street area	type of car	addition to basic premium
yes	0	50000	no	-	100
yes	50000	100000	no	-	200
yes	100000	-	no	-	300
no	-	-	yes	sports car	500
no	-	-	yes	sedan	100

Table 9.8. Decision table

Decision tables are simple to understand and maintain, especially for domain experts. On the other hand, the more variables and rules you use, the more your decision table bloats up and makes it hard to maintain.

The Drools Rule Language (DRL) on the other hand is more oriented towards developers. It is the native rule language of Drools. Let's see a rule in DRL:

```
rule "Luxury cars in dodgy areas cost a nice extra premium"
when
    c: Car( dodgyArea==true, price > 100000 )
then
    c.premium += 300;
end
```

Use the context menu entry *New*, the menu *File > New* or the tool bar button *New* to create either a new decision table or a DRL rule file. The rules will be shown in the project tree below their root folder *Rules*.

Execute rules

Before we see how you can use the rule engine to do something, let's once more make very clear the difference between logic in source code and using a rule engine. In code you have to explicitly define which part of the code to call, you are in control and responsibility to do the right things in the right order. In a rule engine this is different, you simply tell the rule engine to execute and it does find out itself which rules apply and which rules to fire.

To run the rule engine you have to use the Public API of the rule engine, e.g. in a script step or in a Java class. Use `ivy.rules.engine` to access the execution part of the rule engine API. First you will need to create a so called *rule base*. A rule base is a container in which you can load multiple rule files.

```
IRuleBase myRuleBase = ivy.rules.engine.createRuleBase();
myRuleBase.loadRulesFromNamespace("my.rule.name.space");
```



Tip

In the designer, the rule files are hot deployed into rule base. So when you are running your process and you change a rule file that is loaded in a rule base, then the Designer automatically unloads the old version of the rule file and loads the new one.



Tip

Use the namespace to group rule files that belong together and use the corresponding API to load all rule files of the same namespace together. You can also load the rule files from your dependent projects. And you can override rules and rules files by having a rule or rule file with the same name in the overriding project.

Now, what you need too is an instance of the data model that you used in the pre-conditions and the actions of your rules. You can either give the root object of your data or a list of objects. So, create a session, load the data into it and execute:

```
myRuleBase.createSession().execute(out.myDataForTheRules);
```

You should now see the result of the actions applied in the data that you passed into the rule engine before.



Tip

You can use the Public API in JUnit tests directly. Use this to test standalone rules and even groups of rules or all your rules with a pre-defined input and assert if the output matches your expectations. You must extend from `AbstractRuleEngineTest` and be aware that only rule files inside the same project can be resolved.

Demo project

To help you learn how to use the rule engine integration, we created a small demo project called *RuleEngineDemos* that is bundled together with the Designer.

Extensions

This chapter explains how Axon.ivy can be extended by customers or 3rd party software development companies.

Extendable Process Elements

Axon.ivy contains four process elements that can be extended with your own implementation. These process elements are:

- Call And Wait Element
- Start Event Element
- Intermediate Event Element
- Program Interface



Tip

Sample implementations of custom process elements can be found on GitHub in our open source repository. E.g.

<https://github.com/ivy-supplements/bpm-beans/tree/master/ldap-beans>

New Bean Class Wizard

Overview

With the *New Bean Class Wizard* you can create a Java class that implements the required interface for an extensible process element. It can also create an UI editor for the configuration of the event for the corresponding bean. The Java class that is created contains example code about how to implement the bean.

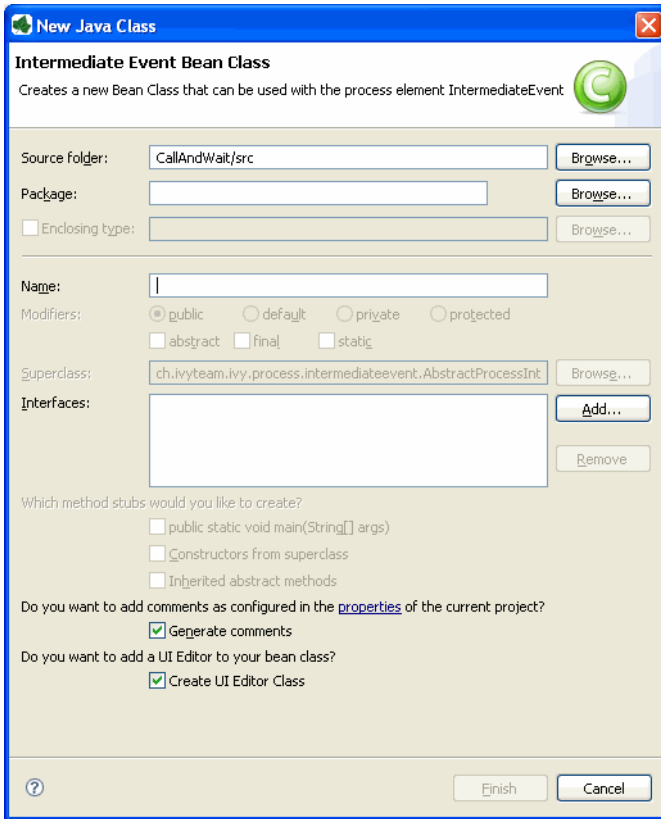


Figure 9.10. New Bean Class Wizard

Accessibility

Process Editor > inscribe > Inscription Mask > ... > Java Class to execute >



Features

Source Folder	The source folder the new Bean class should be created in.
Package	The Java package the new Bean class should be created in.
Name	The name of the new Bean class.
Interfaces	The interfaces the new Bean class should implement.
Generate comments	Should the new Bean class contain Javadoc comments?
Create UI Editor Class	Should an UI Editor class for the new Bean class be created?

Provide your own process elements

It is possible to provide your own process elements to Axon.Ivy. The process elements you provide are based on the standard extensible process elements Program Interface (PI), Start Event, Intermediate Event and Call& Wait. The bean class of your process elements are hard coded and cannot be changed on the inscription masks.

To define such process elements use the extension point `ch.ivyteam.ivy.process.element.IExtensibleStandardProcessElementExtension`. After you have implemented this extension point, Axon.Ivy knows about your process elements but they do not appear on the user interface.

To add your process elements to the process editor palette use the extension point `ch.ivyteam.ivy.designer.process.ui.editor.palette.IIvyProcessPaletteExtension`.

Now your process elements are visible on the palette, but the labels for the process elements are not yet defined. Use the extension point `ch.ivyteam.ivy.designer.process.ui.info.IProcessElementUiInformationExtension` to define the labels (short name, name and description) of your process elements.

Last but not least, you have to ensure that Axon.ivy can load the bean classes of your process elements. Use the extension point `ch.ivyteam.ivy.java.IIvyProjectClassPathExtension` to add the classes of your bundle to the Axon.ivy project runtime class path. You can now use your own process elements in Axon.ivy.



Tip

Sample implementations of custom process elements can be found on GitHub in our open source repository. E.g.

- <https://github.com/ivy-supplements/birt-reporting>
- <https://github.com/ivy-supplements/bpm-beans/tree/master/rule-beans>

Axon.ivy Extension Mechanism

There are other ways to extend Axon.ivy than with the process elements mentioned above. You may want to start your own code during the startup of the Axon.ivy Engine, for example to connect to your ERP system. You may have a product that administrates your users and you want to use the same users also in Axon.ivy. For such kind of extension or integration Axon.ivy provides an extension mechanism that allows the execution of 3rd party code on some points. These points are called extension points.

Build an Axon.ivy extension bundle (Eclipse plugin)

To provide an Axon.ivy extension you have to build an Eclipse plugin for the Axon.ivy Designer or Engine.

You can build your Eclipse plugin in the Axon.ivy Designer with the following steps:

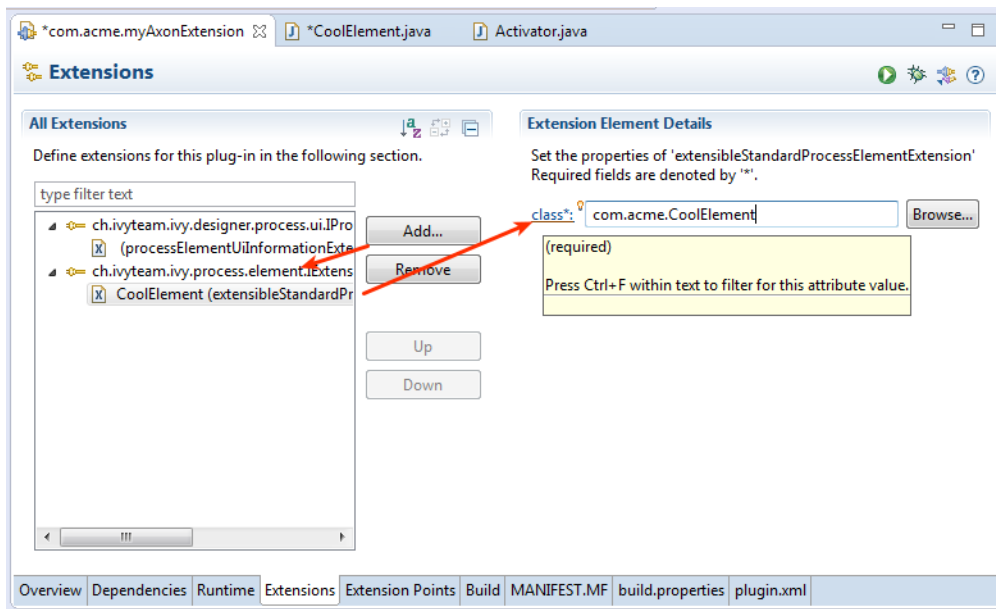
1. Start Axon.ivy Designer
2. Switch to the Plug-in Development Perspective. Menu: **Window > Open Perspective > Other... > Plug-in Development**
3. Create a new Plug-in Project. Menu: **File > New > Project ...**. In the appearing dialog:
 - Choose **Plug-in Project**.
 - Press the **Next** button.
 - Enter a project name.
 - Press the **Next** button.
 - Enter the **Plug-in Properties**.

Property	Description	Example
Plug-In ID	Identifier of the plugin. Must be unique. This identifier must be specified in the <code>*.extensions</code> file in the bundle attributes.	<code>ch.ivyteam.ivy.example</code>
Plug-In Version	The version of the plugin	1.0.0
Plug-In Name	The name of the plugin. The name is used for documentation only	Example

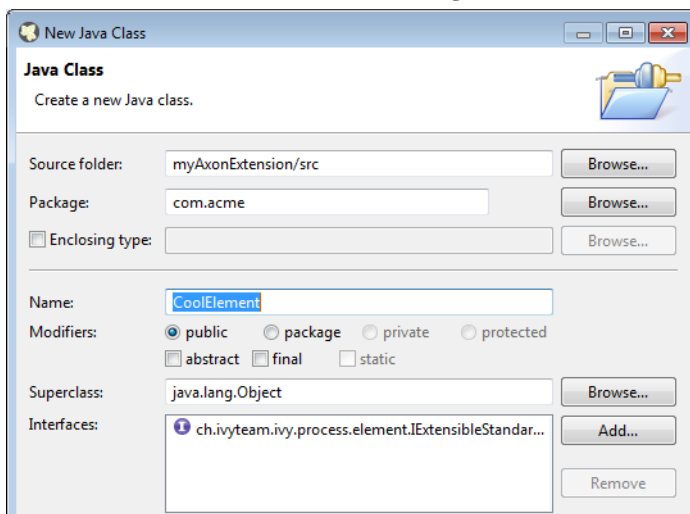
Property	Description	Example
Plug-In Provider	The provider of the plugin. The provider is used for documentation only	ivyTeam / Soreco Group

Table 9.9. Plug-in Properties

- Press the **Finish** button.
4. In the appearing editor click on the **Dependencies** tab. In the section **Required Plug-ins** press the **Add** button. From the list of plugins choose the one that provides the extension point you are going to extend right now. Press the **Ok** button.
 5. Switch to the **Extensions** tab. In the section **All Extensions** press the **Add** button. From the list of extension points choose the one you want to provide an extension for. Press the **Finish** button.
 6. Select the added extension point from the list in the section **All Extensions**. Select the added sub entry. In the section **Extension Element Details** click on the link **class***.



7. A **New Java Class** dialog appears. Enter the name of your extension class into the **Name** text field and the package where it should be located into the **Package** text field.



8. Write your extension class implementing the interface that the extension point requires (see Extension points)

- Switch back to the **META-INF/MANIFEST.MF** file editor. Choose the **Overview** tab and click on the link **Export Wizard**. As **Destination Directory** choose the *dropins* directory of your Axon.ivy Designer or Engine installation. Press the **Finish** button. Your plugin is created into the *dropins/plugins* directory.

Installation

You have to do the following steps to install your extensions in Axon.ivy Designer or Engine:

- Stop the running instance if any.
- Copy your plugin (bundle) that contains your extension classes to the *dropins* directory inside the Axon.ivy Designer or Engine installation directory.
- Start the Axon.ivy Designer or Engine.



Tip

If your extension is not active as expected, consult the *dropins/README.html*.

Extension Point Reference

Axon.ivy supports the following extension points:

Extension Point	Description
ch.ivyteam.ivy.server. IServerExtension	A Server extension can be used to start and stop your code when the Axon.ivy Engine is started or stopped . Server extensions can be accessed from Process Start Event and Process Intermediate Event Beans and also from every process element using the <code>ivy.extensions</code> environment variable.
ch.ivyteam.ivy.process.element. IExtensibleStandardProcessElementExtension	Element Extension can be used to define your own process elements based on the process elements Program Interface (PI), Start Event, Intermediate Event and Call&Wait.
ch.ivyteam.ivy.designer.process.ui.editor.palette. I IvyProcessPaletteExtension	Adds Extensions and process element entries to the process editor palette.
ch.ivyteam.ivy.designer.process.ui.info. IProcessElementUiInformationExtension	Information Extension (name, description) for your own process elements.
ch.ivyteam.ivy.java. I IvyProjectClassPathExtension	Adds libraries or classes from bundles to the ivy project class path. The extension point allows to add libraries or classes to the compile and the runtime class path. This extension point is useful if you want to provide your own classes in a eclipse bundle and want to access these classes from ivyScript or use them as Program Interface (PI), Start Event, Intermediate Event and Call&Wait bean.

Table 9.10. Axon.ivy Extension Points

Rich Dialog Client Side Libraries

Providing a Custom Certificate

This section shows you how to provide a custom certificate for your Rich Dialog clients.



Note

When a client starts a Rich Dialog for the first time, a security information dialog box pops up showing information about the publisher of Axon.ivy as well as the publisher's certificate. It may be confusing to the

users of your application that a certificate from Axon.ivy is displayed instead of a certificate of the implementor/vendor of the software they're using. This section shows you how to avoid this situation.

Install a Java Developer Kit (JDK)

Install a Java Developer Kit (JDK) version 1.7. You can download it from <http://java.oracle.com>

Creating a custom certificate

Use the `keytool` from the previously installed JDK to create a custom certificate in a Java keystore. You may also want to sign your certificate by a certification authority.

See the JDK keytool documentation for details.

Signing the Rich Dialog client libraries

To sign your custom Rich Dialog client extensions with a custom certificate, proceed as follows:

1. Configure the following properties:

```
keystore.file, keystore.password, keystore.alias and dir.jdk
```

in the file

```
[Axon.ivy Designer Installation Directory] / clientlib / build.properties
```

Save your changes.

2. Open the Ant build file

```
[Axon.ivy Designer Installation Directory] / clientlib / build.xml
```

in Axon.ivy Designer.

3. Open the *Outline View* (Use the menu *Window > Show View > Others... > General > Outline*)

4. In the *Outline View* select the root entry in the tree. Right click to bring up the context menu and select *Run As > Ant Build* to run the Ant build script (*build.xml*). The *Console View* will show what the Ant build is doing and if there are errors during the build, they will shown in the *Console View* as well.

5. If the Ant build was successful then a new directory has been created:

```
[Axon.ivy Designer Installation Directory] / clientlib / customSigned
```

This directory now contains the new Rich Dialog client libraries which have been signed with your certificate.

Axon.ivy Designer will automatically use libraries that are located in this new directory when a Rich Application is started, i.e. the provided client side libraries will automatically be deployed to the client when simulating the next time.



Note

If an error message like `jarsigner error: java.lang.RuntimeException: keystore load: DerInputStream.getLength(): lengthTag=109, too big.` appears, try setting the property `keystore.type` accordingly to the type used in the keystore.

Deploying custom signed Rich Dialog client libraries

To use the new custom signed client side extensions with another Axon.ivy Designer or Engine, simply copy the directory

```
Axon.ivy Installation Directory / clientlib / customSigned
```

with all its files to a directory with the same name of another Axon.ivy Designer or Engine Installation Directory.

Restrictions



Warning

Upgrading Axon.ivy Designer or Engine may install a new version of the standard client side libraries. If this is the case you have to resign your custom Rich Dialog client side libraries so that they are compatible with the newer standard client side libraries.

Axon.ivy Designer will fall back to the standard client side libraries if the custom signed client side library version does not match.

Axon.ivy Engine can only be started if the custom signed client side libraries have exactly the same version as the standard client side libraries.



Warning

All Rich Dialog client libraries located in the *clientlib/customSigned* directory must be signed by the same certificate. Otherwise the Rich Dialog client will not start because of security restrictions.



Warning

Because of performance optimizations (lazy loading) all client side libraries located in the *clientlib/customSigned* directory must be signed together. Do not mix new with previously signed libraries, otherwise the lazy loading index will be inconsistent, leading to errors on the client side when executing your application.

Providing custom ULC Widgets



Warning

Development of custom ULC widgets is only permitted with a valid ULC developer licence. Licences and more information about the ULC (Ultra Light Client) framework can be obtained from Canoo.

Licensed ULC developers may contribute their own ULC extensions to an *Axon.ivy* installation. This chapter explains how custom ULC widgets can be included by packing them into a *Axon.ivy Rich Dialog Client Extension*.

Using custom ULC Widgets

To use a custom ULC Widget the following steps are necessary:

1. The server side classes of the custom ULC Widget must be located in a Axon.ivy Project, so that they can be loaded on server side.
2. The client side classes of the custom ULC Widget must be located in a Axon.ivy Rich Dialog Client Extension that is installed in the directory *clientlib/extensions*.

Packing custom ULC Widgets to an Axon.ivy Rich Dialog Client Extension

To pack custom ULC Widgets to an Axon.ivy Rich Dialog Client Extension the following steps are necessary:

1. The client side classes of the custom ULC Widgets must be available in *.jar files.
2. In the *Axon.ivy Designer Installation Directory / clientlib / extensions* directory create a new sub directory with the name of your extension

3. Create another sub directory called *clientLibs* inside the previously created extension sub directory. The directory structure now looks like:

Axon.ivy Installation Directory / clientlib / extensions / myExtension / clientLibs

In the example the extension is called *myExtension*.

4. Copy the .jar files with the client side classes to the *clientLibs* sub directory created above.
5. Now, execute the same steps that are necessary to provide a customer certificate. Instead of using the file *Axon.ivy Designer Installation Directory / clientlib / build.xml* use the file *Axon.ivy Designer Installation Directory / clientlib / extensions / build.xml*. See section above.

If the Ant build is successful then a new directory was created:

Axon.ivy Designer Installation Directory / clientlib / extensions / myExtension / signed

It contains the jar files with the custom ULC Widgets signed with your certificate. The build will also create an extension configuration file if no such file is already available:

Axon.ivy Designer Installation Directory / clientlib / extensions / myExtension / extension.any

It contains information about your Axon.ivy Rich Dialog Client Extension like title, vendor, homepage, etc.. But also the version of the extension. This version should be increased and the extension rebuilt every time the client side classes of the ULC Widget changes.

Axon.ivy Designer will automatically use the new created extension for the Rich Dialog clients.

Deploying Axon.ivy Rich Dialog Client Extensions

To deploy an Axon.ivy Rich Dialog Client Extension simple copy the extension directory (e.g. *myExtension*) that contains the *extension.any* file and the *signed* directory inclusive the signed jars to the directory *clientlib / extensions* of another Axon.ivy Designer or Engine Installation.



Note

It is possible to install multiple Axon.ivy Rich Dialog Client Extensions from different vendors with different certificates.

Restrictions



Warning

All Jar files of one extension must be signed by the same certificate. Otherwise the Rich Dialog client will not start because of security restrictions.



Warning

Because of performance optimizations (lazy loading) all jar files of one extensions must be signed together. Do not mix new with previously signed jar files, otherwise the lazy loading index is inconsistent.

Troubleshooting Java Web Start Cache Problems

Rich Dialog Clients are using a technology called Java Web Start. Java Web Start caches all libraries it downloads from Axon.ivy. This can cause conflicts if you are developing and do not increase the version of your client side classes if they change or if you changing certificates. The following command line tools can help to solve problems.

Use the following command line to clean the cache of Java Web Start. This will remove all cached libraries from the cache:

```
javaws -uninstall
```

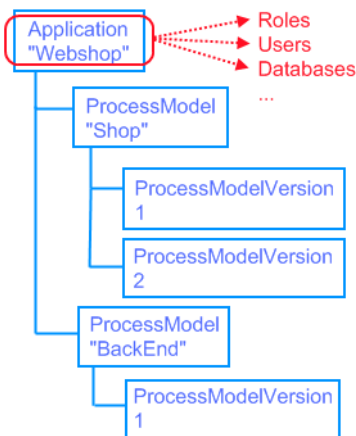
Use the following command to inspect the libraries that are cached by Java Web Start:

```
javaws -viewer
```

Deployment

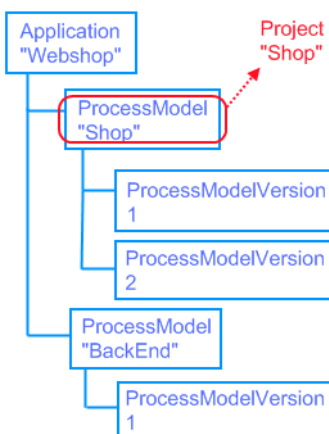
This chapter explains how an Axon.ivy Designer Project can be deployed to an Axon.ivy Engine. Before deploying an Axon.ivy project it is important to understand some major concepts and terms of the Axon.ivy Engine. The following chapter introduces these concepts and terms.

Application



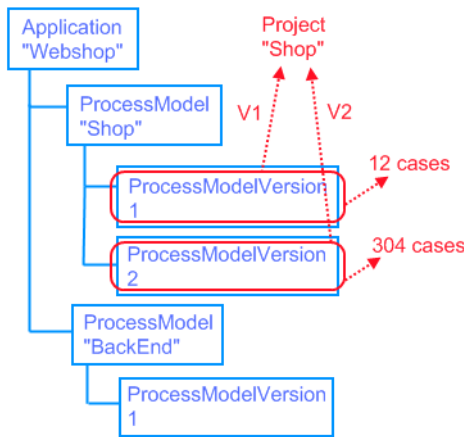
On the Axon.ivy Engine, applications can be configured. An application spans up an environment in which roles, users, external databases, tasks, cases and process models exist. Applications are completely independent of each other. E.g. a user of one application can **not** work on a task of another application.

Process Model



Within an application multiple process models can be configured. A process model on the Engine corresponds to an Axon.ivy project on the Designer. The difference is that a process model may hold multiple different versions of the same Axon.ivy project. A process model version - as its name suggests - is a version of an Axon.ivy project. In fact this version represents the state of an Axon.ivy project at the time it was deployed on the Engine.

Process Model Version



A process model can have multiple versions called process model versions. These versions allow to change an Axon.ivy project without worrying about the compatibility of currently running cases on the Engine. How does this work? When an Axon.ivy project has been finished or reached a milestone, it is going to be deployed as the first process model version. Users can use this project, they start processes. Some of the processes may last long time (weeks, months, or even years). While these processes (e.g. cases) are running, the project may be enhanced and might have now incompatible changes to the first version. Now the changed project can not be deployed to the first version but to a new configured version. Consequently old cases must not be stopped, they will be still executed within the first process model version. Meanwhile new cases are started from the new deployed version.

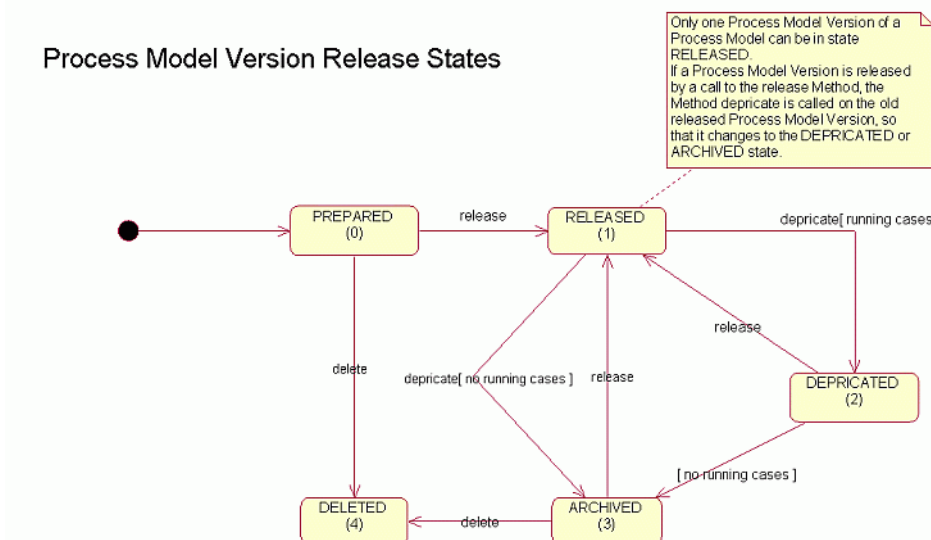
A process model version has a release state. The release state of a process model version is responsible how the version is used by the system. The most important release state is the state **RELEASED**. Within a process model only one version can be in this state. All processes that are started in a process model are started in the released process model version! A complete list of release state can be found in the following list:

Name	Description
PREPARED	The process model version has been created and the project may already have been deployed. However, the process model version is not yet used.
RELEASED	The process model version is the currently released version. This means that all new processes are started in this version. Program Starts and Web Service Processes are only active for process model versions in this state.
DEPRECATED	The process model version has previously been in state RELEASED , but then another version was released. Therefore, this version is now not in RELEASED state but in DEPRECATED state. All cases that were started in this process model version will continue to run in this version. As soon as all cases of this version have been ended, the state will change to ARCHIVED automatically.
ARCHIVED	The process model version has previously been in state RELEASED , but then another version was released, and running cases has been finished in this process model. Consequently, this version is now not in RELEASED state anymore but has been ARCHIVED . Actually the engine administrator can change a process model version from state ARCHIVED back to state RELEASED if necessary.

Name	Description
DELETED	The process model version has been deleted. All project data belonging to this version has been deleted.

Table 9.11. Release states of process model versions on Axon.ivy Engine

The following diagram shows all release states and state changes that are possible:



Configuration Example

The following table shows an example of how applications, process models and process model versions on an Axon.ivy Engine can be configured.

Application	Process Model	Process Model Version	Description
Company1			Application for company1. All users of the company are automatically imported to this application from the company's active directory server.
	HRM		Human Resource Management process model. Corresponds to the Axon.ivy project called "HRM".
		V1	The first version of the HRM project was released in February 2008. This version is deprecated. There are still cases running in this version
		V2	The second version of the HRM project was released in April 2008. This version is released. All new processes are started in this version.
		V3	The third version of the HRM project was created in January 2009. This version is prepared, but not used

Application	Process Model	Process Model Version	Description
			productive. It will be released on the first of September 2009.
	Finance		Finance process model. Corresponds to the Axon.ivy project Finance.
		V1	The first version of the Finance project was released in August 2007. This version is released. All new process are started in this version.
Company2			Application for company2. The users of the company are managed by the Axon.ivy Engine.
	HRM		Human Resource Management process model. Corresponds to the Axon.ivy project called "HRM".
		V1	The first version of the HRM project was released in April 2008. This version is released, so that all HRM processes of company2 run and are started in this version.

Table 9.12. Configuration Example

Axon.ivy Project Deployment

To deploy an Axon.ivy project to the Axon.ivy Engine execute the following steps:

1. Export all files of the project you want to deploy to a zip file using the Export wizard of Axon.ivy Designer (See next section).
2. Copy the zip file with your project files to the Axon.ivy Engine.
3. Start the Engine Administrator application on the Axon.ivy Engine
4. Choose or create an application
5. Choose or create a process model
6. Choose or create a process model version
7. Open the detail page of the process model version and find the section Deployment.
8. Press the **Deploy** button to start the deployment wizard.
9. On the first step of the deployment wizard choose the zip file with your project files and follow the wizard to deploy your project.



Tip

More information about the deployment of Axon.ivy projects or applications, process models and process model versions can be found in the Axon.ivy Engine Guide.

Export all Project Files to a ZIP-File

For the deploying of a project it is useful to export all files of a project to a zip file. This can be done with the export wizard of Axon.ivy Designer. Start the export wizard either by using the menu **File > Export ...** or by using the context menu **Export ...** in the Ivy Project Tree on a selected project.

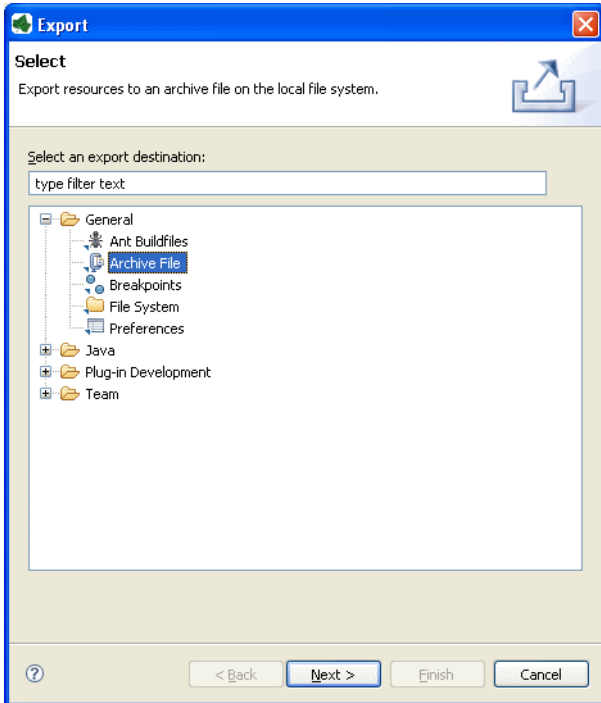


Figure 9.11. Export Wizard

On the export wizard select **General > Archive File**. Then press the **Next >** button.

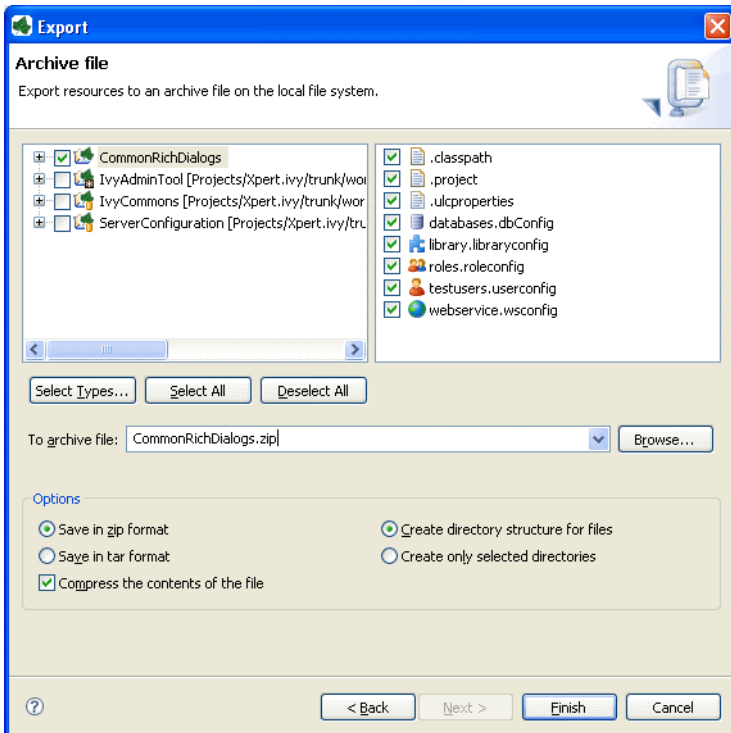


Figure 9.12. Export Wizard

Choose the project you want to deploy (export). Only select one project because the Deployment wizard can only handle one project in a zip file. Specify the zip (archive) file and press the **Finish** button. The created zip file can now be used to deploy your project to the engine.

Continuous Integration

Ivy Projects are designed to be built on a continuous integration (CI) server like Jenkins.

Maven build plugin

The project-build-plugin is a Maven plugin that can build Ivy Projects on a developer machine or on a continuous integration server. The plugin provides the following main features:

- Compilation of Ivy Projects
- Testing of unit tests against an Ivy Project or the Ivy core classes
- Packaging of built Ivy Projects as IAR (ivy archive) artifacts
- Installation of IAR artifacts into the local Maven repository
- Deployment of IAR artifacts to an Axon.ivy Engine

Runtime

The Designer has a built in Maven runtime that allows to start Maven with zero initial configuration effort. A local maven build can be started as follows:

1. Switch to the Java perspective
2. Expand an Ivy Project in the Ivy Project Tree view
3. Open the context menu of the file 'pom.xml' by right clicking it
4. Navigate to 'Run as' > 'Maven install'

Configuration

Ivy Projects declare its ids and dependencies in the “Project Deployment Descriptor”. This deployment descriptor can be easily edited with the corresponding ivy editor and is stored as Maven Project Object Model (POM.xml). Therefore each Ivy Project has by default the pom.xml which is needed by maven to build it.

However advanced Maven users can adjust this default configuration and use additional Maven plugins or dependencies in the pom.xml. But not all POM entries should be modified, some are required or limited in usage in Ivy Projects:

- `<groupId/>` and `<version/>` Must be set in every Ivy Project POM. It can not be inherited from a parent POM (even tough this is valid in plain Maven).
- `<packaging>iar</packaging>` Provides the custom Ivy Project lifecycle, must not be modified.
- Dependencies with `<type>iar</type>` will be manipulated by the “Deployment Descriptor Editor”. Therefore additional configurations like the `<scope>` could get lost trough the simple editor usage.
- Values that can be manipulated with the simple “Deployment Descriptor Editor” can not contain Maven properties. For instance `<version>${myVersionProp}</version>` is prohibited.
- The version must be qualified like `<version>5.0.0-SNAPSHOT</version>`. A version like `<version>5-SNAPSHOT</version>` is prohibited.

Technical documentation

- The detailed plugin goal and parameter documentation is on Github.io
- The source code of the ivy project build plugin is available on Github.com.

Continuous Integration Job with Jenkins


The following steps are needed to build an Ivy Project on a Jenkins CI server.

1. Install Jenkins as described in the Jenkins Wiki
2. Install a Maven runtime in Jenkins via *Manage > Configure > Maven > Maven installation > Choose auto installation*
3. Create a new Jenkins Job. Select "Maven-Project" as job style.
4. Provide a link to the source code of the Ivy Project in the *Source-Code-Management* section
5. Configure the goals `clean verify` in the *Build* section
6. Save the Job and Run it

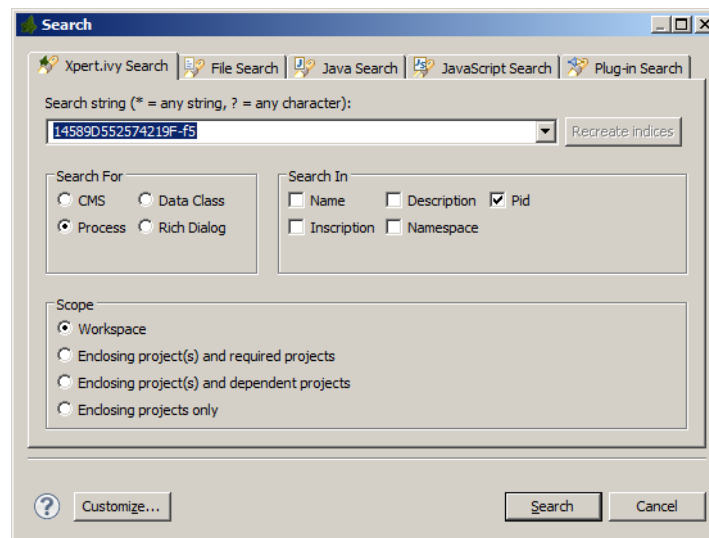
Miscellaneous

This chapter deals with several concepts and features that are integrated into Axon.ivy to leverage user convenience and experience.

Axon.ivy Search

In a workspace with many large projects it is sometimes hard to find specific Ivy elements. Then a powerful search mechanism can save the day. To use the Axon.ivy search, just click on the  symbol in the toolbar to open the search dialog. In the dialog that opens navigate to the Axon.ivy tab. At present, searching for CMS content objects, Data Classes / Entity Classes, Processes / Process Elements and Rich Dialogs (but not Html Dialogs) is supported by Axon.ivy.

The search page



Search string

Enter here the string you are searching for. You may use two wild-cards: The *** (*star*) for any sequence of characters (may be empty too). and the *?* (*question mark*) for a single

character (e.g. `a*b` matches each entity that starts with "a" and ends with "b" and has 0, 1 or more characters in between whereas `a?b` matches all strings with a length of three that start with an "a", end with "b" and has one character in the middle)

- Search For / Search In** Select for what kind of entities you are looking for. Depending on the chosen type, you can specify in which properties of the entity the *search string* (see above) is searched in. If you select more than one property, then be aware that the *search string* must occur only in one of the chosen properties.
- Scope** You can decide whether you want to search in the full workspace or only in the enclosing projects (the projects that are selected in the Ivy Projects View). If you choose *enclosing projects* you may select whether you want to include searching in dependent or required projects (see the Project Deployment Descriptor chapter for more details about how you can define and use project dependencies). The tool tip text tells which projects are currently selected.
- Recreate indices** The search indices in Ivy are automatically updated if you add, edit or delete entities. However, if you want to recreate the search indices hit this button and all indices are deleted and recreated from scratch in the background. Please be aware, that searching during the time of index creation may not return correct results.

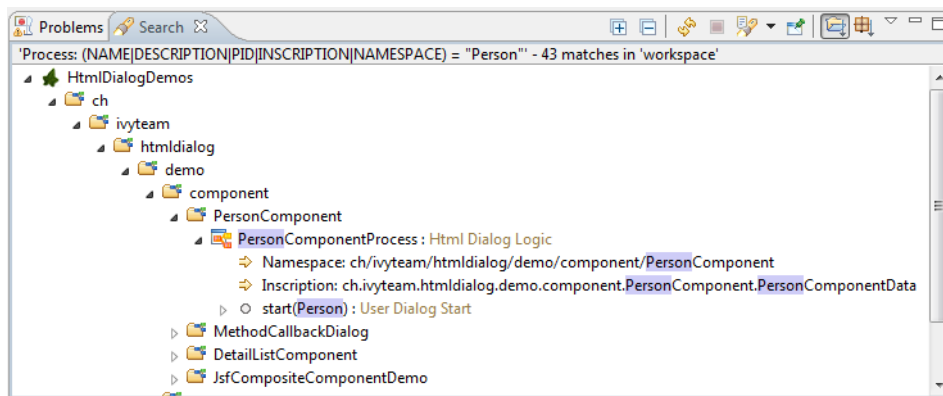


Note

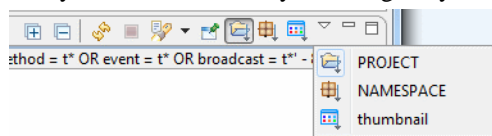
You may use as well other search facilities within this dialog to search for parts that are not covered by the Axon.ivy search page. e.g. if you write your own Java classes in the Axon.ivy Designer you may use the Java search.

The search result view

After clicking on the search button, the search results are collected in the search result view. Double-click on matching entries and the corresponding resource is opened in its editor.



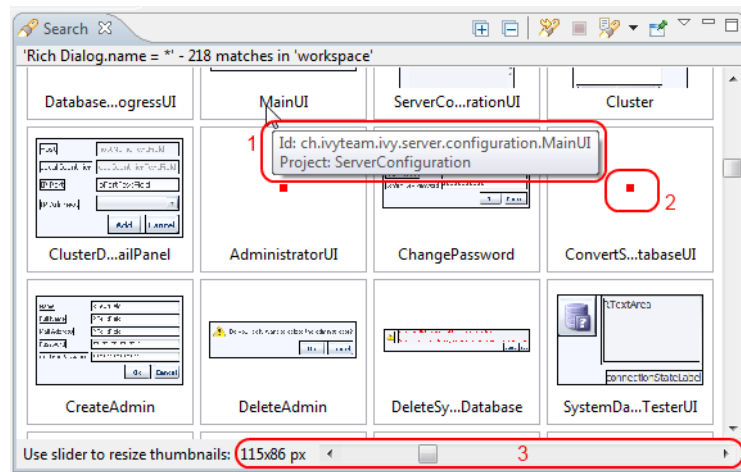
You can change the presentation layout for your search results by selecting a layout from the result view's menu:



For standard searches, only *Project* and *Namespace* grouping is available. For Rich Dialog searches the results can also be displayed as *thumbnails* (see next section).

Thumbnail Results for Rich Dialog Search

When searching for Rich Dialogs, the results are by default presented as *thumbnails*.



The result page shows the Rich Dialogs that have been found as a result of the search with a scaled image of the Rich Dialog's screen shot (if one is available) and with the Rich Dialogs simple name. The tool tip (1) offers additional information, such as the Rich Dialog's fully qualified name and the project it is located in. Rich Dialog's that don't have a screen shot are shown with a small red square (2).

Double clicking on a thumbnail will open the associated Rich Dialog's interface editor.

The size of the displayed thumbnails can be adjusted with the slider at the bottom of the view (3). Use this slider to set your preferred thumbnail size, it will be saved and used for all later thumbnail search results. The default size for the Rich Dialog thumbnails is set at 160 x 120 pixels, the minimum and maximum sizes being 16 x 12 and 480 x 360 pixels, respectively.



Tip

To automatically create a screen shot for a Rich Dialog you must first enable the feature on a Rich Dialog's interface editor (default is *yes*) and then open and save the Rich Dialog with the Rich Dialog editor. The screen shot will be updated every time you make changes to the Rich Dialog.

Update Notification

When newer Axon.ivy versions are available a dialog appears after starting Axon.ivy Designer. The dialog contains information about the new versions and where those can be downloaded.

Use the checkboxes provided on the dialog if you don't want to see the dialog again either for the same versions or for any new versions.

If you want to check for new versions manually use the menu *Axon.ivy > Check for Updates ...*



Note

While checking for new versions the following statistic information are sent to the update server. These information are only used to improve the product.

- Current designer version
- Operating system information (name, version, architecture, number of processors)
- Java memory information (maximum heap memory, maximum non heap memory)
- JVM (Java virtual machine) information (version, vendor, name)
- Host information (host name, SHA-256 hashes of IP address and MAC address to identify the host without being able to read the original IP address and MAC address itself)

Data Caching

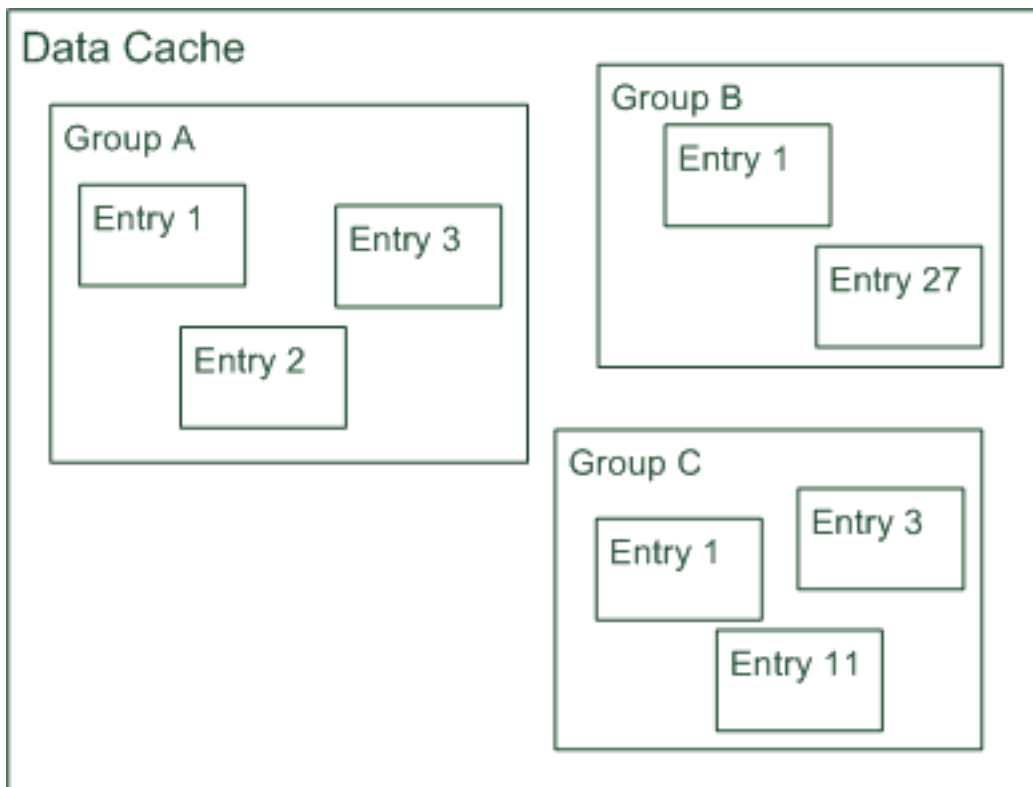
Axon.ivy offers a feature to store data temporarily into a *data cache* in the engine's memory. If you want to read data that stays unchanged for some time, you do not need to re-read the data every time you need to access it. If this data is read by long-running queries from a database or by calling a slow web service, you can gain a lot of performance by caching this data. The database step and the web service step natively support Data Caching (see the Data Cache Tab for more information), for other data you can access the Data Cache API by IvyScript.

Caches

Data that is cached is always stored in a data cache entry. This can be the result of a database query or of a web service call if you use the Data Cache Tab on the database step or on the web service step. But you can also store any arbitrary object into a data cache entry by using the Public API. Entries are identified by a textual entry identifier.

Entries are organised into groups. An entry always belongs to exactly one group, you cannot store the same entry in more than one group. In other words, the identifier of an entry must be unique in its group. If two entries in the same group have the same identifiers, then they are identical. Like entries, groups are as well identified by a textual group identifier. Use groups to store cache entries with similar data. This simplifies the invalidation of related data, see chapter invalidation below.

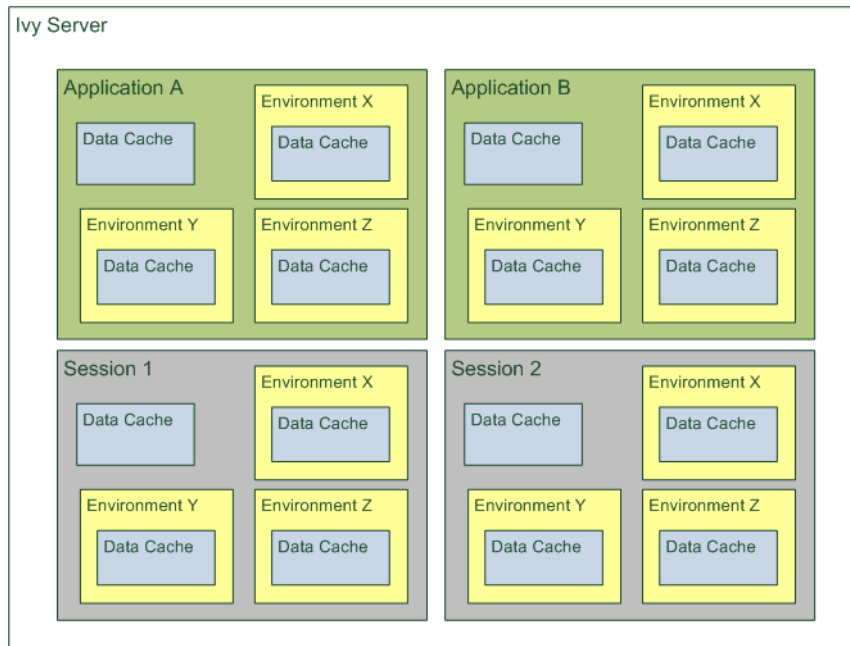
A *Data Cache* is a container for multiple groups. The identifier of a group must be unique in its data cache. If two groups in the same Data Cache have the same identifiers, then they are identical.



Data Caches always have a scope. A scope defines the boundaries of a specific Data Cache and as well the life cycle of the Data Cache depends on its scope:

Scope	Type of cached data	Multiplicity of Data Cache	Data Cache life cycle start	Data Cache life cycle end
Application	Global data that is related to the application	One per application	Application creation or engine start	Application deletion or engine stop
Environment	Global data that can vary for different environments, e.g. if you are using tenants or different configurations	One per application and environment	Environment creation or engine start	Environment deletion or engine stop

Scope	Type of cached data	Multiplicity of Data Cache	Data Cache life cycle start	Data Cache life cycle end
Session	Data that is related to interactions within the actual session	One per session and environment	Session start	Session end



Access and Life Cycle

Cache entries and groups are created the first time they are accessed - the first time the process step with the data cache entry is executed - and they are destroyed when the scope of cache entries or groups reach the end of their life cycle. For the scope *Session* this is the logout of the user of the session or the session timeout, for the scopes *Application* and *Environment* this is when the application is terminated or inactivated.

Cache entries and groups are resolved by their identifiers. You can put different cache entries into one group by using the same group identifier for all entries. You can use the same data cache entry for multiple steps by using the same group and the same entry identifier for all entries. This is very useful for data that almost never changes, you can load the data into the cache once at the beginning of the scope's lifetime and read it from the cache from every step in all processes thereafter.

Invalidation

In order to take into consideration changes in the data that is handled by the cache entries, it is possible to invalidate cache entries and as well whole groups either on request or after a configurable period of time. Thereby, invalidation means that only the value of the data cache entry is deleted, but not the entry itself. The next time a step referring to this data cache entry is executed, the value of the data cache is loaded again.

You can invalidate an entry, a group and even the whole cache explicitly in the Data Cache Tab of inscription masks of the process steps that use data caching or by an IvyScript API call. Otherwise you may specify a period as a lifetime or fixed time of day for invalidation. The lifetime period starts when the group/entry is loaded the first time. A background job is responsible to invalidate entries/groups when their lifetime has ended. If you set a fixed invalidation time, the corresponding entry or group is invalidated once per day at that time. By invalidating a group, all its contained entries are invalidated and similarly invalidating the whole data cache does invalidate all groups and therefore as well all entries.

How Data Caching works on an Axon.ivy Engine Enterprise Edition

An Axon.ivy Engine Enterprise Edition consists of multiple engine instances (nodes) that are running on different machines.

In an Axon.ivy Engine Enterprise Edition the *Application* and *Environment* data caches will be created on each node independently. However, if a data cache is invalidated on one cluster node either by timeout or explicitly, then it will be automatically invalidated on all other cluster nodes as well.

On the other hand, *Session* data caches will only be created on one node because sessions are always bound to a specific node in the cluster.

Eclipse Plugin Mechanism

You need a database frontend in Axon.ivy? Or editing support for any other programming or data declaration languages such as C/C++, PHP or XML? Or you have UML models to view? No problem at all.

Axon.ivy is based on the widely used Eclipse platform which offers a sophisticated plugin mechanism to integrate third-party modules. In these days, Eclipse which originally has been developed as an IDE for Java programmers evolved to a large and vibrant ecosystem and is used for a triad of different tools and systems in almost every work sector. Therefore a huge community exists that offers plugins (open source and commercial) and even web sites (Eclipse Marketplace) for browsing and searching these plugins arose in the past years.

And the conclusion, you can use all these plugins and integrate them into your Axon.ivy installation to interact seamlessly between your favourite plugin set and the built-in Axon.ivy features.



Note

Please follow the installation instructions of the specific plugin to integrate it into your Axon.ivy installation

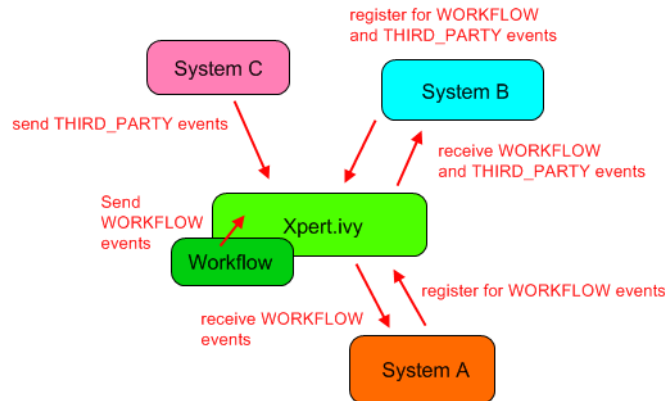
System Events

Axon.ivy offers the concept of system events, which can be understood as messages that are broadcasted across the Axon.ivy installation. While Axon.ivy itself (e.g. the workflow subsystem) generates events that interested participants may subscribe to (e.g. to be informed when a case is created or finished), it is also possible for implementors to define their own events and to broadcast them to any component that might be interested. Since this mechanism is session- and workflow independent, it can also be used to implement *inter-session communication* (within the same Application).

Concept and general usage

System events are messages that are broadcasted across the Axon.ivy system and that will be delivered to any interested party. System events have a name and are categorized, and they may carry an optional parameter object. System events can only be sent within the same Application on an Axon.ivy Engine.

Currently two categories are defined: `SystemEventCategory.THIRD_PARTY` and `SystemEventCategory.WORKFLOW`. The category `THIRD_PARTY` can be used to send (and receive) system events that are generated by integrated third party applications (or processes). The category is reserved exclusively for this purpose; i.e. the Axon.ivy Engine will never generate any events of this type.



The Axon.ivy system itself currently only generates events of the category WORKFLOW. Inside this category, events with the following names are generated:

- `WorkflowSystemEvent.TASK_CREATED`
- `WorkflowSystemEvent.TASK_CHANGED`
- `WorkflowSystemEvent.CASE_CREATED`
- `WorkflowSystemEvent.CASE_CHANGED`

All of those events carry a parameter object of the type `WorkflowSystemEventParameter` which gives access to the identifiers of the workflow objects that have been created or modified. More system defined categories and events can be expected in the future.

To send system events, client and/or third party applications must first create a `SystemEvent` object and then get a hold of an `IApplication` object, which offers the method `sendSystemEvent(SystemEvent event)`. Only events of the category `THIRD_PARTY` can be sent by process applications, attempts to send system events of different categories will result in an error.

To receive system events, clients must implement the interface `ISystemEventListener` and must then register themselves on an `IApplication` object using the method `addSystemEventListener(EnumSet<SystemEventCategory> categories, ISystemEventListener listener)`. It is strongly recommended, that the similar `remove` method is used, as soon as clients are no longer interested in a specific event category.

Clients should only listen to system events they know the name of, all other events should be ignored. Clients should handle received events as fast as possible, because handling will block the delivery of events to other receivers. Also the received parameter object should never be modified (it shouldn't be modifiable in the first place), since this may affect the handling by other receivers which will consequently receive a modified event object.

In Java, the handling of system events generally results in code similar to the following:

```

/**
 * Registers this participant for workflow system events.
 */
public void registerForWorkflowEvents(IApplication application)
{
    application.addSystemEventListener(EnumSet.of(SystemEventCategory.WORKFLOW));
}

/**
 * Unregister this participant for all system events.
 */
public void unregister(IApplication application)
{
    application.removeSystemEventListener(EnumSet.allOf(SystemEventCategory.class));
}
  
```

```

/**
 * Implementation of ISystemEventListener.handleSystemEvent(...)
 * Events will only be delivered for the categories that this listener registered for
 */
public void handleSystemEvent(SystemEvent event)
{
    String eventName = event.getName();
    if ("thirdparty.mysystem.myevent".equals(eventName))
    {
        // do something
    }
    else if (WorkflowSystemEvent.TASK_CHANGED.equals(eventName))
    {
        // do something
    }
    // else: ignore event
}

/**
 * Distribute a new system event to all interested/registered listeners of my event.
 * MyEventParameter can be of any (serializable) type, the type is part of the event definition,
 * clients will have to cast accordingly.
 */
public void sendMyEvent(IApplication application, MyEventParameter param)
{
    SystemEvent event = new SystemEvent(SystemEventCategory.THIRD_PARTY, "thirdparty.mysystem.myevent", param);
    application.sendSystemEvent(event);
}

```

Rich Dialogs can also send and receive `SystemEvents` via the broadcast mechanism. Read the section below to learn how.

How System Events work on an Axon.ivy Engine Enterprise Edition

An Axon.ivy Engine Enterprise Edition consists of multiple engine instances (nodes) that are running on different machines.

Distribution of system events is handled in two ways on a Engine Enterprise Edition, depending on their category:

- `THIRD_PARTY` system events are distributed as cluster messages across all nodes, i.e. from the node that generates the event to all other cluster nodes
- `WORKFLOW` system events are generated on each cluster node in parallel and then distributed locally only

Important implementation notes:

Since `THIRD_PARTY` events are distributed as messages in a Cluster, all custom event parameter objects *must be serializable*.

Please be aware of the fact that having multiple running instances of a system event sender may lead to race conditions. If you use system events for message exchange between e.g. processes and/or User Dialogs and third party systems that are integrated via the Server Extension mechanism, you should ensure that a certain event is only sent once. This may require that the third party system (e.g. an ESB) is only started on one node in the cluster. Otherwise a received message from the external system may be injected into the Axon.ivy Engine Enterprise Edition system n times (once for each node) instead of being sent only once.

How to send and receive System Events in Rich Dialogs

By using the event broadcast mechanism in combination with the System Events framework, Rich Dialogs can receive and send system events as broadcast events.



Note

Check out the *Chat Demo* in the *IvyDemos* project (available from your Axon.ivy Designer installation directory under *applications/samples/IvyDemos*) to see how System Events can be used to implement an inter-session information exchange.

The demo implements a simple chat application which makes usage of `THIRD_PARTY` system events to send or receive messages from chat users, respectively.

Receiving System Events

Before the Rich Dialog can receive any events, it must register itself for the respective categories. This is done by calling `ivy.rd.subscribeToSystemEvent(SystemEventCategory)`. Typically this registration should happen in the start method process of the Rich Dialog as shown below. Explicit de-registration is not necessary, all Rich Dialogs will automatically be unregistered as soon as their panel is unloaded (i.e. when the Rich Dialog executes a Rich Dialog End element).

Furthermore, the Rich Dialog must declare all the names of the system events it is interested in as accepted broadcast events on it's interface. The category of the system event does not matter (they are defined in the registration call as described above). However, the name of the accepted broadcast event must be exactly identical to the name of the system events that the rich dialog is interested in. Also the type of the parameter must be identical to the type of the system event's optional parameter. If names or parameter type do not match, then the system event will not be delivered.

The handling of the declared accepted broadcast events is then implemented with Rich Dialog event handler processes as usual.



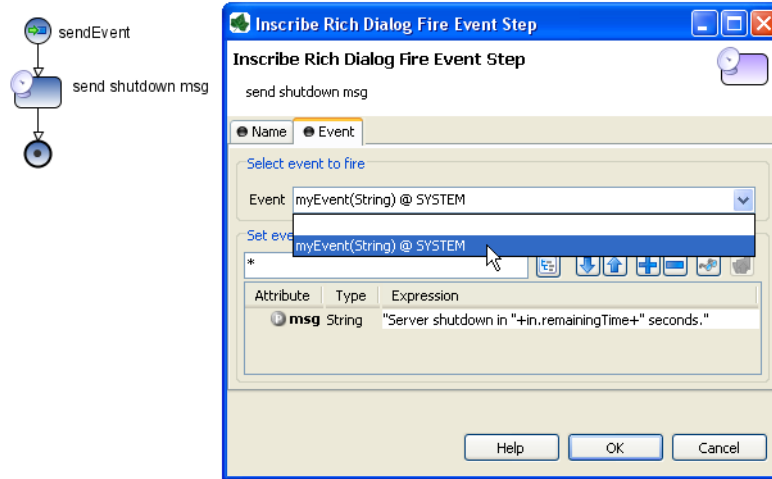
Warning

Due to technical reasons, the name of an accepted broadcast event has to be a valid *Java* identifier. Therefore the original name of a system event may be an illegal name for an accepted broadcast event (e.g. the system event name `foo.bar.baz` will not be accepted because it contains punctuation). In such cases the name of the accepted broadcast event should substitute all illegal characters with underscores, e.g. `foo_bar_baz` for the given example)

Sending System Events

Sending System Events from a Rich Dialog is fairly easy. There are two possibilities:

1. On the Rich Dialog interface declare a fired event with scope `SYSTEM`. Then use the Fire Event process element to send the event. The category of the system event will always be `THIRD_PARTY`, the declared event's name will be used as the system event's name and the (optional) parameter will become the system event's parameter.



2. Use IvyScript/Java to send a system event through the `IApplication.sendSystemEvent(SystemEvent)` API as described in the general usage section above.

Chapter 10. Troubleshooting

Introduction

Here you will find solutions to some of the most common problems related to Axon.ivy Designer. If you can't find your solution here there are some other sources which could help:

Axon.ivy Q&A	The Axon.ivy Q&A contains a considerable amount of questions and answers related to Axon.ivy Designer and Engine.
Stack Overflow	Problems related to common technologies like Java, JSF, Primefaces are answered on the web, e.g. on Stack Overflow.
Support	You can get support via support@soreco.ch (support may be subject to charging, depending on your licence agreement).

Error Dialogs

Error Id

Error Dialogs shown to the user contain a unique error id. This error id can be used to search the log files for more information about this error.

Error Report

On Error Dialogs it is possible to generate an Error Report. This report contains information about the error, the designer machine and the current state of the Axon.ivy Designer.



Note

If you need to contact the support because of an error, please generate an Error Report for it. Most of the time it contains the necessary information, to find and solve the problem.

On the Rich Dialog error dialog use the *Copy* button to generate an error report and copy it to the clipboard.

On the HTML error page use the *Open Error Report* button at the end of the error page to open the Error Report. Use CTRL-A and CTRL-C to copy the Error Report to the clipboard.

The Event Details Dialog of the Runtime Log View provides a button to generate and save an error report for a given log entry.

Moreover, you can also use the menu *Axon.ivy > Debug > Save Debug Report* to generate and save such a report without an error.

Startup Problems

Start of Designer fails

Immediately after the start of the Axon.ivy Designer you get the error dialog below and the Designer does not start.

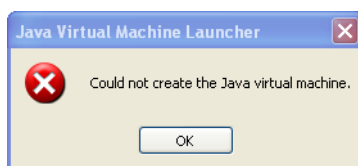


Figure 10.1. Error dialog when Axon.ivy Designer.ini broken

Most probably the configuration file *Axon.ivy Designer.ini* for the Designer is broken. Ensure that all parameters you added are correct. Furthermore, ensure that you do not have empty lines in the file. No whitespace characters (such as space or tab) except line feed and carriage return are allowed at the end of a line.

Start of Elasticsearch / Business data manager fails

If Elasticsearch cannot be started make sure that the ivy Designer is not running in a folder that has special Windows UAC settings.

Alternatively, try creating a `plugins/` folder inside the `elasticsearch/` folder in the main designer directory.

Memory Problems

OutOfMemoryException: Java heap space

If this error occurs, then the Designer requests more memory than it is allowed to use. This can happen when a lot of data is used during the process simulations. Ensure, that your computer has enough memory. Then you can increase the maximal memory consumption of the Axon.ivy Designer. Just open the *Axon.ivy Designer.ini* and change the value in the line after `-Xmx` to a higher one.



Note

If you have a 64 Bit computer use the 64 Bit version of the Axon.ivy Designer. This allows you to allocate more than 2 GB memory.

Graphics Problems

Superimposition of UI elements when scrolling

With a combination of certain Java 1.6, Windows, DirectX and graphics adapter driver versions, some strange UI effects may occur. Mostly happening when the UI must be redrawn very frequently (e.g. scrolling tables, moving windows), it looks like some UI elements are superimposed by some other UI elements.

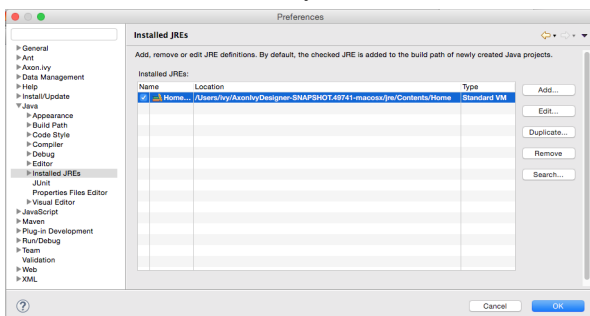
First try to update the driver of your graphics adapter. If this does not work, you may want to switch off DirectX usage for the Designer. Just add the following line to the *Axon.ivy Designer.ini* and the problem should be gone.

```
-Dsun.java2d.d3d=false
```

OS X Problems

No Java Execution Environment is set (Problem: "Unbound classpath container: 'Default System Library'")

Solution: In the Preferences tree `Java/Installed JREs` add a Standard VM located in the Axon.ivy Designer `jre/Contents/Home` directory and select it.



Key bindings in the process editor don't work sporadically

Solution: Close the process and reopen it.

Logging

Configure ULC logging on server and client

It is possible to log ULC communication and status messages both on client and server side. Both a default log level and a log level per known Ivy test user can be configured, as well as a log file path/name where the log of an ULC session should be stored permanently.

To configure ULC logging, the files *configuration/jnlpcfg.any* and *configuration/ulclogcfg.any* within the Axon.ivy Designer installation folder must be edited. The former one controls the client-side logging, the latter one the server-side logging. The log configurations are similar in both cases and edited with *Anything* notation.

To modify the *client logging* properties locate the following section in *jnlpcfg.any*:

```
...
/log-level {
  /user-specific {
    /johndoe FINER
  }
  /default WARNING
  /log-to-file "C:/temp/xpervivy_ulc_client.log"
}
...
```

To modify the *server logging* properties locate the following section in *ulclogcfg.any*:

```
{
  /user-specific {
    /johndoe FINER
  }
  /default WARNING
  /log-to-file "xpervivy_ulc_server.log"
}
```

To enable user-specific logging on server side and/or client side, enter the known name of the Ivy test user to log for in the */user-specific* section, followed by a valid log level (one of SEVERE, WARNING, INFO, FINE, FINER, FINEST) just as specified in the example configurations with the user *johndoe*. Use double quotes around the Ivy test user name if it contains special (i.e. non-ASCII) characters or white space.

To set the default log level (which will be used for all sessions) set the log level for the key */default*.

Finally, to enable logging to a file, enter a filename **in double quotes** for the key */log-to-file* as shown above. If the */log-to-file* slot is omitted altogether or if its value is a ***, then no log file will be created.



Note

Changes in log configurations take place immediately for all new sessions or client application starts; a restart of the designer is not necessary. However, you have to restart any running client applications for the new log settings to take effect.

The created log files will automatically include the user name (if user-specific logging is enabled), the HTTP session ID and an ID for the application. The two ID's mentioned are part of the client and the server log file, so it is easy to find the two logs related to a client-server communication.



Tip

For reasonable results a default level of `WARNING` or `INFO` is recommended. For error analysis, `FINER` on the client and `FINER` on the server is recommended, on a per-user basis.

If you enable ULC logging, it is recommended to specify the `/log-to-file` parameter. This way the log output is redirected to a file, otherwise the logs are written to the console.



Warning

Please bear in mind that the log file path that you specify may not be in correct format for every client platform (Linux/Windows). If this should be the case then the log files will be created in the client system's default temporary directory.

Be aware that log files can become very large (up to several hundred MB per day and user if `FINE` /`FINER` /`FINEST` is used as log level).

Chapter 11. References

Conventions used in this book

This section covers the conventions used in this book.

Typographic Conventions

Constant width	Used to indicate source code, e.g. Java or IvyScript class names, properties or methods.
<i>Italics</i>	Used to <ul style="list-style-type: none">• introduce terms• for URLs• for email addresses• for filenames and directory paths• define the navigation within Axon.ivy application menus• for referencing GUI elements (e.g. Open the <i>Code</i> tab for editing the properties).
bold	Used to highlight important terms in a text.
{replacable text}	Names or parts of strings that are dependent on the user environment, are marked with brackets. See the following example: Navigate to <i>{Designer Install Directory}/configuration/demo.lic</i>

Displays



Note

The **note** designates a note relating to the surrounding text.



Tip

The **tip** designates a helpful tip relating to the surrounding text.



Warning

The **warning** designates a warning relating to the surrounding text.

Screen

A **screen** is used to show keyboard input. e.g. script samples

```
out.x=in.y;
```

Sidebar

Sidebar

In a **sidebar** you will find background information.

Reference

This chapter provides a linked reference of the Axon.ivy application parts.

Editors

- Case Map editor
- Configuration editor
- Content Object editor
- Database Configuration editor
- Data Class editor
- Entity Class editor
- Java editor
- Project Deployment editor (formerly known as *Library* editor)
- Overrides editor
- Process editor
- REST Client Configuration editor
- Rich Dialog Panel editor
- HTML Dialog Panel editor
- User Dialog Interface editor
- Rich Dialog Interface editor
- Role editor
- User editor
- Web Service Configuration editor
- Persistence Configuration editor

Views

- Breakpoints view
- CMS Tree view
- Data Binding view
- Error view
- Event Mapping view

- Expressions view
- History view
- Java Beans view
- Axon.ivy Outline view
- Process Performance view
- Process Outline view
- Process Templates view
- Problems view
- Axon.ivy Projects view
- Properties view
- Reference view
- Runtime Log view
- Tasks view
- Variables view
- Widget Configuration view
- Web Browser view

Wizards

- New Bean Class Wizard
- New Case Map Wizard
- New Data Class wizard
- New Entity Class wizard
- New Event Mapping wizard
- New Override wizard
- New Process wizard
- New Process Group wizard
- New Axon.ivy Project wizard
- New User Dialog wizard
- New Html Dialog View wizard
- Rename wizard
- Move wizard
- Copy wizard
- Delete wizard
- Export Axon.ivy Archive wizard

- Import Axon.ivy Archive wizard
- Import Axon.ivy Modeler Processes wizard
- Import into CMS
- Export from CMS

Perspectives

See also the Perspectives section in the *Introduction* chapter.

- Process Model Perspective
- Process Development Perspective
- Rich Dialog Perspective

Process Elements

See also Process Elements reference chapter.

- Request Start
- Web Page
- Alternative
- Call & Wait
- Split
- Join
- Task
- Task Simple
- “Error Start”
- Event Start
- Intermediate Event
- Process End
- Script Step
- Database Step
- E-Mail
- PI (Programming Interface)
- Web Service
- REST Client
- Independent Sub-Process (Call Sub)
- Embedded Sub Process
- Note

- User Dialog
- Init Start
- Method Start
- Event Start
- Broadcast Start
- Rich Dialog Script Step
- Fire Event
- UI Synchronization
- Process End
- Exit End
- Generic
- User
- Manual
- Script
- Receive
- Rule
- Send
- Service

Widgets

See Rich Dialog panel widgets reference chapter.

- ButtonGroup
- RBorderLayoutPanel
- RBoxPane
- RBrowser
- RButton
- RCardDisplay
- RCardPane
- RCheckBox
- RCheckBoxMenuItem
- RCloseableTabbedDisplay
- RCollapsiblePane
- RComboBox
- RDatePicker

- RFiller
- RFlowLayoutPane
- RGridBagLayoutPane
- RGridLayoutPane
- RHtmlPane
- RHyperlink
- RLabel
- RList
- RListDisplay
- RLookupTextField
- RMenu
- RMenuBar
- RMenuItem
- RMenuSeparator
- RPasswordField
- RPopupMenu
- RProgressBar
- RRadioButton
- RRadioButtonMenuItem
- RScrollBar
- RScrollPane
- RSlider
- RSplitPane
- RTabbedDisplay
- RTable
- RTableTree
- RTaskPane
- RTaskPaneDisplay
- RTaskPaneContainer
- RTextArea
- RTextField
- RToggleButton
- RToolBar

- RTree

IvyScript

See IvyScript section for more information

- IvyScript language
- IvyScript reference
- Public API

Miscellaneous

- Rich Dialog User Context
- Rich Dialog User Context Examples
- Rich Dialog UI State

Glossary

This chapter provides an alphabetized glossary for specialized expressions and terms that are employed in this book.

Application	<p>On the Axon.ivy Engine one or more Applications can exist. The Application defines the container wherein the Process Models can be deployed.</p> <p>Also Users and Roles are defined and Tasks and Cases are stored in an Application.</p> <p>See also section Application in chapter Deployment.</p>
Case	<p>A Case is one concrete instance of a Process. It must not necessarily run through all Steps of a Process. A Process may define a different handling for different Cases depending on the information of a Case. For example by using the Alternative Element .</p> <p>A Case holds the information used to carry out the Process. This is on the one hand information about the Case like the current position in the Process. On the other hand this is information collected during the Case, which is passed from one Step to the next in the form of the Process Data</p>
Connector	A Connector connects two Steps of a Process. This defines the sequence of these two steps.
Form Field	A Form Field is a JSF code snippet which serves content for a specific data class field type (e.g. a Label and a Datepicker for a Date). The New User Dialog Wizard create forms with Form Fields.
Html Dialog	A Html Dialog is an implementation of a User Dialog. The Html Dialog is implemented with (HTML/JSF).
Layout	A Layout contains the main structure of a Html page (e.g. a header, content and footer section). For Web Pages the layouts are defined in the CMS. For Html Dialogs the layouts are defined in the webContent folder.
Process	A Process is an abstract description how a group of Cases will be handled. It consists of Process Elements connected with each other. Every time a Process is started, a Case and a Task is created.
Process Data	The Process Data is the data passed from one Step to the next. Its represented by a data class used for the whole process. Even though every Step can create a new instance of this data class to be passed to the next Step it will always be an instance of the same data class.

Process Element	<p>Process Elements are the bricks a process is built of. Ordered with Connectors they become the Steps of a Process.</p> <p>There are 3 groups of process elements.</p> <ul style="list-style-type: none">• Activities do something. Like running a script or let somebody else do something for example by showing a User Dialog.• Gateways structure a Process. For example a Alternative decides which way a Case runs through the Process.• Events are notifications of things that happen outside the process. <p>The chapter Process Elements describes all the Process Elements available in <i>Axon.ivy</i></p>
Process Model	<p>A Process Model on the Engine corresponds to an Axon.ivy project on the Designer. The difference is that a Process Model may hold multiple different versions of the same Axon.ivy project. These are called Process Model Version.</p> <p>See also section Process Model in chapter Deployment.</p>
Process Model Version	<p>A Process Model can have multiple versions called Process Model Versions. These versions allow to change an Axon.ivy project without worrying about the compatibility of currently running Cases on the Engine.</p> <p>See also section Process Model Version in chapter Deployment.</p>
Rich Dialog	<p>A Rich Dialog is an implementation of a User Dialog. The Html Dialog is implemented with (Java/ULC).</p>
Role	<p>A User has one or multiple Roles which define what the user is allowed to do.</p>
Signature	<p>In computer programming, especially object-oriented programming, a method is commonly identified by its unique method signature. This usually includes the method name, and the number, types and order of its parameters, but usually excludes the return type(s) of the method.</p> <p>Within <i>Axon.ivy</i>, signatures act as unique identifiers for specific start elements (e.g. method starts, request starts, trigger starts, call sub starts), within the same process, only one element with the same signature may exist. The same holds true for signatures of start methods and events on a User/Rich Dialog interface.</p>
Step	<p>A Process Element placed in a Process becomes a Step of this Process. The Connectors define the order of the Steps in a Process.</p>
Task	<p>A Task is a unit of work which is indivisible. It has to be carried out by one user in one piece. If anything goes wrong during the execution of a task, we must return to the beginning of the task.</p> <p>It's not possible to work on a Case without a Task. Every time a new Case is started a Task will be created. While working on a Case / Task new Tasks can be created. This allows to interrupt the work on a Case and to hand it over to another user if necessary.</p> <p>A Task consists of one or multiple Steps. It begins for example with a Request Start or a Task Switch Element. And ends for example with the next Task Switch Element or at the Process End.</p> <p>Task can be assigned to a specific User, a Role or to Everybody.</p>
User	<p>A User is a person interacting with a Case. The user is identified by a unique user name.</p> <p>If a User is not identified, we speak of an anonymous User.</p>

User Dialog	A User Dialog is a concept of an User Interface. User Dialogs can be implemented as Html Dialog (HTML/JSF) or as Rich Dialog (Java/ULC).
View Type	A View Type defines the default content of a User Dialog View. Axon.ivy has predefined View Types, i.e. Page and Component.