

FuegoBPM Studio 5 Documentation

Fuego, Inc.

FuegoBPM Studio 5 Documentation

by Fuego, Inc.

Published January, 2005 - Version 5.5. Revision 10 - June, 2006.

Copyright © 2001-2006 Fuego, Inc.

FuegoBPM Studio 5 Documentation

Copyright 2001-2006 Fuego, Inc. All rights reserved.

This documentation is subject to change without notice. This documentation and the software described in this document contains proprietary trade secrets and confidential information of Fuego, Inc. and is also protected by U.S. and other copyright laws and applicable international treaties. Use of this documentation and the software is subject to the license agreement between you and Fuego, Inc. If no such license agreement exists, you may not use this documentation and software in any manner whatsoever. Unauthorized use of the documentation or software, or any portion of it, will result in civil liability and/or criminal penalties. U.S. Patent Pending.

Fuego, Fuego 4, Component Manager, Process Designer, Work Portal, Orchestration Engine, Execution Console, Process Analyzer, Organization Administrator are trademarks or registered trademarks of Fuego, Inc.

FuegoBPM 5, FuegoBPM Studio, FuegoBPM Designer, FuegoBPM Enterprise Administration Center, FuegoBPM Work Portal, FuegoBPM Portal Console, FuegoBPM Archive Viewer, FuegoBPM Logviewer, FuegoBPM Express Server, FuegoBPM Enterprise Server, FuegoBPM Application Server Edition, FuegoBPM Web Console, FuegoBPM Process Analyzer, FuegoBPM Data Store, FuegoBPM Dashboard, FuegoBPM BAM, FuegoBPM Portlets, FuegoBPM Suite, FuegoBPM Deployer, FuegoBPM Failover, FuegoBPM VCS, FuegoBPM Ant Tasks, FuegoBPM FDI, FuegoBPM Help Viewer, FuegoBPM Server are trademarks or registered trademarks of Fuego, Inc.

InstallAnywhere is a registered trademark of Zero G Software, Inc. Solaris and Java are trademarks of Sun Microsystems, Inc. Windows is a registered trademark of Microsoft Corporation.

All other trademarks, trade names, and service marks are owned by their respective companies.

Table of Contents

1. FuegoBPM Basics	11
Business Services Orchestration	11
What's FuegoBPM	14
System Requirements	16
Architecture	17
Internationalization	22
FuegoBPM Studio Development Workspace	26
FuegoBPM Studio Preferences settings	47
Contextual Help	55
Introduction to FuegoBPM Studio Administration Guide ...	56
FuegoBPM Studio Update	59
Applying a Service Pack to the FuegoBPM Studio	60
Applying a Hotfix to the FuegoBPM Studio	68
2. Defining an Orchestration Project	73
Project Preferences	85
Project Synchronization	89
Localizing and using multiple languages	91
Documenting a Project	92
Generating the Project documentation	99
Saving the Project	105
3. Designing a Process	108
Instance	109
Process	111
Process Group	137
Process Exception Flow	137
Importing a process from Visio	144
Importing a process from Aris	149
Importing a process from Workflow Management Coalition - WfMC	153
Generating automatically a Subprocess, Procedure and Screenflow	159
Searching for a process, procedure or screenflow	168
Notes within a Process	169
Generate Events	171

Audit Trail - Work Portal	174
Audit Trail Information Columns - Work Portal	178
Roles within a Process	180
Process as Web Services	183
4. Activities	189
Initiating a Process	202
Begin Activity	202
End Activity	208
Human Interaction	211
Interactive	211
Grab	215
System Interaction	220
Automatic	220
Process Controls	224
Split-Join Circuit	224
Split N-Join Circuit	239
Conditional	246
Organizations Interaction	248
Subprocess	248
Process Creation	252
Termination Wait	257
Process Notification	259
Notification Wait	262
Dynamic Process Call	269
Global actions	281
Global Creation Activity	281
Global Automatic	289
Global Automatic Examples	308
Global	316
Other activities	330
Connectors	330
Measurement Marks	330
5. Tasks	335
BP-Method Overview	346
Tasks Execution Timeout	350
Procedures	359
Procedure examples	367
Screenflows	367

Screenflow examples	387
External Tasks	392
Using Relay to	394
Procedure and Screenflows versus IPC	401
6. Argument Mapping & Correlations	402
Argument Mapping	402
Correlations	423
7. Groups	441
Groups and Grab activities	450
Groups Examples	451
8. Transitions	454
Conditional Transition	458
Unconditional Transition	461
Due Transition	462
Exception Transition	473
Compensate Transition	474
Message Based Transition	475
9. Using Variables	481
Instance Variables	485
External Variables	487
Business Variables	489
Argument Variables	492
Predefined Variables	495
Local Variables	509
Business Parameters	510
10. Exception Handling	514
How to handle Exceptions	518
Exception Examples	539
Global Automatic Activities and Exceptions	550
Exceptions	551
ClassNotFoundException	553
NumberFormatException	553
NegativeArraySizeException	553
ArithmeticException	553
CloneNotSupportedException	554
NullPointerException	554
ArrayStoreException	554
IllegalThreadStateException	554

StringIndexOutOfBoundsException	554
Exception	555
ArrayIndexOutOfBoundsException	555
IllegalAccessException	555
UnsupportedOperationException	555
InterruptedException	555
InstantiationException	556
IllegalArgumentException	556
IndexOutOfBoundsException	556
RuntimeException	556
SecurityException	557
IllegalMonitorStateException	557
ClassCastException	557
IllegalStateException	558
11. Compensation Handling	559
Compensate Activity	564
Compensate Activity Examples	569
12. HTML Process API	578
HTML Process API Examples	598
13. BPEL Processes	628
BPEL activities	632
14. Simulation	650
Simulation	650
15. Defining Business Objects levels	692
Introducing Business Objects into FuegoBPM Project Catalog	696
Modules	702
Components	704
Implementing Business Objects Using Fuego Objects	709
Fuego Objects Attributes	720
Fuego Objects Groups	754
Fuego Objects Methods	760
Presentables and Non-Presentables Fuego Objects ..	790
Fuego Objects Presentations	791
Designing Using Fuego Objects	947
Handling Fuego Objects	950
Fuego Objects as Instance Variables	957
Using JSP and Fuego Objects	958

Transformations	987
Implementing inheritance	1010
Fuego Objects Example	1024
Implementing Business Objects Using external components	1099
COM Components	1099
.Net Assemblies Components	1114
Enterprise JavaBeans ComponentsEJB	1123
Web Services	1130
XML Components	1134
Java Components	1141
Java Naming Directory Interface Components JNDI	1147
SQL Components	1155
SQL Queries	1172
CORBA Components	1197
Cataloguing SAP BAPIs	1233
Enumerations	1241
Cataloging Exceptions	1249
Implementing Business Objects Using Fuego Blocks	1250
External Resources Configuration	1256
CORBA	1263
SQL Database	1267
COMBridge	1281
Naming and Directory Service	1282
J2EE Application Server	1282
JMS Messaging Service	1283
Enterprise JavaBean	1284
Java Class Libraries	1285
Server	1291
Web Service	1291
16. Project Portal Administration	1293
Default Views and Presentations	1293
Default Views	1293
Default Presentations	1294
Custom Views	1297
Custom Presentations	1305
Editing a presentation	1306
Deleting a presentation	1306

17. FuegoBPM Dashboard	1308
FuegoBPM Dashboard Components	1309
FuegoBPM Dashboard Standard Methods	1323
Methods Invoked from Dashboard Components	1326
Changing Business Rules from a Dashboard	1328
Building some simple Fuego Dashboard Examples	1328
Building a BAM Dashboard using the Wizard	1347
Building a BAM Dashboard Manually	1360
Using the "ClientBusinessProcess" block to build a FuegoBPM Dashboard	1376
How to drill-down to Instance Data from a BAM Dashboard	1380
Recommendations	1385
18. Version Control System	1386
Common Catalog	1420
19. Data Store and Business Activity Monitoring	1426
BAM Configuration	1441
20. Defining Organizational Resources	1443
Organization	1443
Organizational Units	1452
Organizational Roles	1459
Organizational Groups	1470
Participants	1477
Participant's permission for instance assignment ...	1488
Holiday Rules	1494
Calendar Rules	1500
21. Defining the Server and Runtime Properties	1508
Server	1508
Server Properties	1511
Runtime Properties	1526
Deployment	1531
Deployed Processes	1539
Referrals	1540
BAM Properties	1546
Refreshing the Server Data	1548
Inter-process Communication	1550
22. Executing Processes	1554
Publish and Deploy	1554
Starting the Server	1556

Stopping the Server	1559
23. Log Viewer	1561
The Work Environment	1562
The Logged Information	1569
Filters	1575
Connector Rules	1583
Bookmarks	1586
Working with Log Files	1588
24. Appendixes	1589
Quick Definitions	1589
Appendix A - Implement FuegoBPM Studio VCS using VSS	1590
Appendix B - IBM WebSphere 5.1 EJB integration with Fuego	1598
Appendix C - JMS Integration Examples	1622
Integrating FuegoBPM and TIBCO	1622
Integrating FuegoBPM and IBM MQSeries	1628
Integrating FuegoBPM and Weblogic	1641
Appendix D - FuegoBPM Portlets	1642
What is a Portlet	1642
FuegoBPM JSR-168 compliant Portlet	1642
Appendix E - Other Documents of Interest	1645

Chapter 1. FuegoBPM Basics

Business Services Orchestration

The FuegoBPM (TM) Suite embraces and extends the concept of Business Process Management (BPM) through its vision of Business Services Orchestration (BSO.)

BPM is a discipline that includes many different types of tools and methodologies. A simple process modeling tool, such as Visio, can be considered a BPM utility. Business Intelligence tools can be considered BPM utilities. True, in today's market more people are starting to see BPM as a new category of software that **automates business processes**. The problem is: what do we really understand by automating business processes?

- For the creators of BPEL, it is the organization in time of web services invocation
- For EAI fans, it is a state server that coordinates messages on a proprietary bus
- For some ERP vendors, it is the business logic embedded in an ERP system
- For traditional workflow vendors, it is the organization of the collaboration between people

FuegoBPM can be used to fit in any of the above visions, but they fall short of what FuegoBPM was meant to do.

For FuegoBPM, automating business processes consists of **managing the behavior of people, systems and organizations to orchestrate a repeatable business service**.

Therefore,

- FuegoBPM sees organizing the invocation of web services as managing the behavior of systems, and not all systems: only those exposed as web services.
- FuegoBPM sees a state server to coordinate messages as managing the behavior of systems, and not all systems: only those that have adapters into a proprietary messaging bus.
- FuegoBPM sees the business logic embedded in an ERP system as a service that manages the behavior of organizations limited by the rules in the ERP system. This service can be reused in the context of a cross application enterprise process.
- FuegoBPM sees the organization of the collaboration between people as managing the behavior of people.

Fuego's vision of BPM includes all the above visions in one single holistic vision: Business Services Orchestration. FuegoBPM sees anything a person, system or organization does within an enterprise as a **Business Service**. FuegoBPM provides all the necessary tools to **Orchestrate** composite business services using existing ones, manages and measures the service levels of those composite business services and continuously improves them.

This is what we call *Full Lifecycle Management of Orchestrated Business Services*.

To be able to do this, FuegoBPM provides the full set of tools that enables companies to:

1. Model Processes.
2. Transform Process Models into executable designs.
3. Simulate the execution of designs to study the feasibility of a service level.

4. Harmonize and catalog business services from existing systems to be able to use them regardless of what technology is used to expose them.
5. Catalog the different services from people that can be rendered by the organization and their availability in time.
6. Expose composite services that orchestrate services from systems people and organizations to be reutilized.
7. Monitor the orchestration in production according to the parameters set forth in the simulation.
8. Measure the performance of the process from a historical perspective.
9. Use statistical data to refine future simulations.

FuegoBPM can be used to manage the full spectrum of business processes, from the mostly automated (like BPEL) to the more collaborative processes like those that involve specialized workers and creative activities.

When designing with FuegoBPM, it is critical to understand that the Server was conceived to manage **behavior** rather than just to pass data. When working with a business service, the invocation of the service provokes behavior, when presenting a user with a work portal, the Work Portal suggests the adequate behavior to the user. Obviously, the user is free to do as he or she wishes, but it is very convenient not to need to remember the adequate behavior in each intervention in each process in which a user is involved. And, whatever gets done in effect by people, systems and organizations is logged into a process log that allows the tracking, tracing and measuring of performance.

Without any doubt, Business Services Orchestration is the most complete way to automate the management of a business process designed, for example, as a result of a six sigma exercise, ISO

compliance exercise or BPR exercise. Why?

Because the FuegoBPM Enterprise Server will elicit behavior that otherwise would have implied months of training and convincing, and eons of application integration.

Moreover, Business Services Orchestration is the easiest way to build composite apps that integrate existing ones and expose them as web apps or web services.

To provide the ideal Orchestration platform FuegoBPM has centralized all the design and development tools in a single environment: FuegoBPM Studio. As well the design can be previously defined in the FuegoBPM Designer and the development can be completed using the FuegoBPM Studio.

The orchestrations created in Studio run on an orchestration server that comes in two categories: Express and Enterprise.

The Express category of servers is designed for quick deployment of departmental and small business orchestrations that will require no administration or for proof of concept projects in their pre-rollout stage.

The Enterprise category of servers is designed for full featured Enterprise security, scalability and failover capabilities as well as to run inner-departmental and inter-enterprise processes.

What's FuegoBPM

FuegoBPM is a full-life cycle development and runtime environment for managing business processes from a Business Services Orchestration (BSO) perspective. This means that FuegoBPM focuses on managing the behavior of people, systems and organizations (through a process metaphor) to fulfill a measurable and repeatable business service that may span departments, divisions and company boundaries.

The full-life cycle development environment is FuegoBPM Studio.

Studio provides all the necessary functionality for a BSO approach towards BPM.

The full-life cycle runtime environment is provided through two runtime server editions:

- FuegoBPM Express - an entry level server that requires zero administration, fit for self-contained business services or for proof-of-concept projects.
- FuegoBPM Enterprise - the full fledged enterprise edition to run processes that span departments, divisions and enterprises with all the scalability, security and flexibility features you would expect from an enterprise grade product.

FuegoBPM caters to the needs of our customers in terms of TCO (Total Cost of Ownership) and ROI (Return on Investment). This is why we can really improve the way businesses run. FuegoBPM helps businesses increase operational efficiencies, reduce costs and increase profitability with an agile BPMS that can adapt to any budget and manpower. FuegoBPM allows companies to take control and tangibly optimize enterprise assets—applications, people and core business functions – and how they work together. With FuegoBPM, companies can quickly fill the gap between business strategy and execution in order to gain immediate payback.

FuegoBPM provides a BMPS software that makes the critical enterprise assets work the way you do and change as you change. By orchestrating applications, people and partners into executable, end-to-end processes that can be exposed as new composite business services, FuegoBPM fills the gap between business strategy and business execution.

FuegoBPM shields the process logic from the differences that arise from location (timezone, holidays, vacations, language), from IT infrastructure (MS, Unix, Legacy), from IT strategy (J2EE, .NET, Websphere, CORBA) and from the applications that contain reusable

services (SAP, Peoplesoft, I2, Siebel, legacy, etc.). Therefore, allowing non-specialized business analysts to model, design and change processes with no need to be domain experts.

FuegoBPM reduces complexity, enhances productivity and makes any company as competitive as its creativity allows (not limiting process automation to that which their enterprise software vendors provide.)

System Requirements

System Requirements for FuegoBPM Studio (Development Environment)

Operating Systems

FuegoBPM Studio runs on the following operating systems:

Windows

- NT 4.0 Workstation (Service Pack 3 or higher)
- NT 4.0 Server (Service Pack 3 or higher)
- NT 4.1 Workstation
- NT 4.1 Server
- 2000 Professional or Adv. Server
- Windows XP
- Win 2003

UNIX

- Sun Solaris ver. 2.6 or higher (Java 1.4.2 support)

- Linux RedHat distribution ver. 6.x or higher
- Linux SUSE distribution ver. 6.0 or higher
- Compaq Tru64 (Java 1.4.2 support)
- UNIX (Java 1.4.2 support)
- HP-UX 11.00 (Java.1.4.2 support)

Disk Space and RAM

Successful installation of the FuegoBPM Express in a development environment requires the following:

- 350 MB of free disk space.
- 256 MB RAM minimum. 512 MB recommended.

System Requirements for FuegoBPM Express (Production Environment)

Successful installation of FuegoBPM Express runtime environment requires:

- 350 MB of free disk space.
- 256 MB RAM minimum. 512 MB recommended.

Architecture

FuegoBPM is a full-life cycle development and runtime environment

that provides the complete functionality to achieve a seamless solution to integrate, design, deploy and evolve your most important enterprise activities.

FuegoBPM Designer is the entry point to start developing your business processes. Process designers begin creating a project and model the processes but they don't have to focus on the technical issues to implement them.

FuegoBPM Studio is another entry point to start developing your business processes by creating a project.

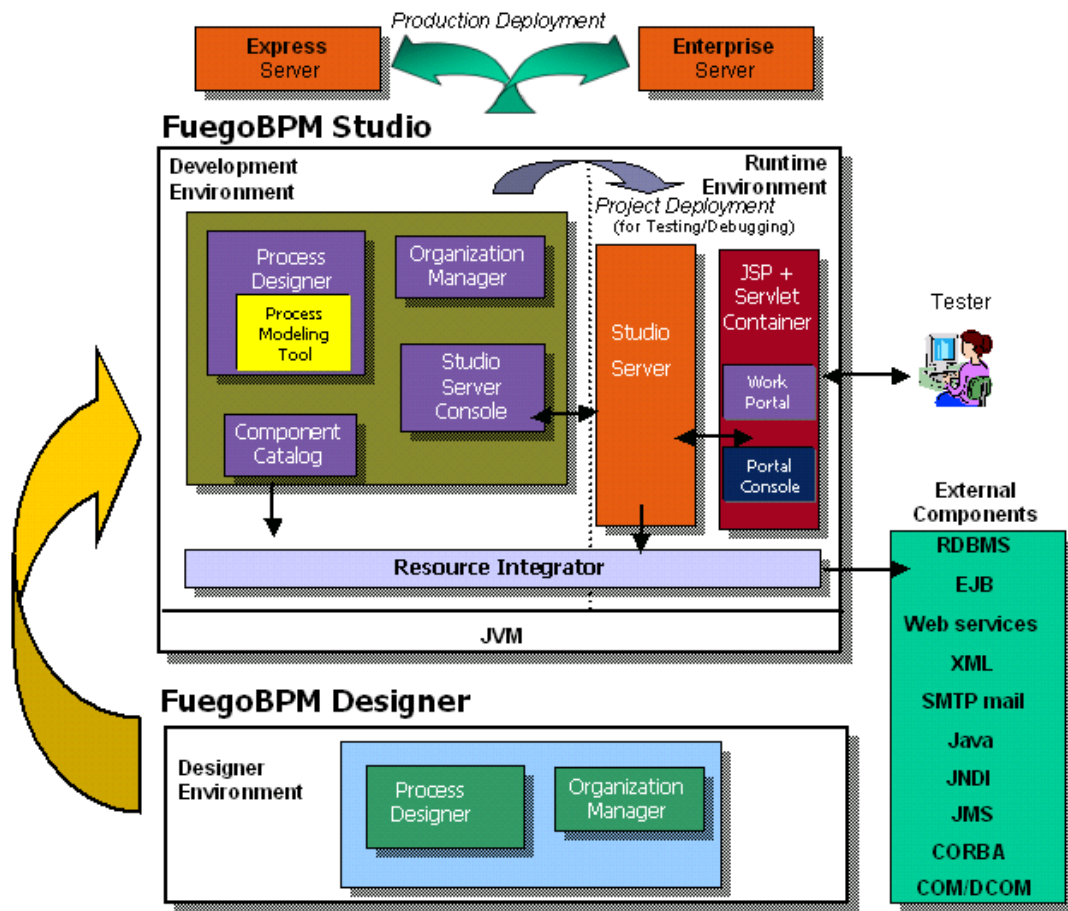
It can be easily installed and provides the most complete **development environment** that allows developers to model processes.

Once the project has been developed, with no additional installation steps or third party products needed, it can be deployed in a **runtime environment**.

FuegoBPM Express provides the full-life cycle runtime environment through a runtime server edition that requires zero administration. It is tailored for self-contained business services or for proof of concept projects.

FuegoBPM Enterprise the full fledged enterprise edition to run processes that span departments, divisions and enterprises with all the scalability, security and flexibility features you would expect from an enterprise grade product.

The following graph shows the environment elements and the interaction between them.



The designer environment

FuegoBPM Designer capabilities allow a **business analyst** to create a project to model the appropriate business processes, including their activities, the transitions between each activity and the roles associated to each of them. No scripting tool is needed at this point.

To manage process participants, FuegoBPM Designer allows you to define the **Organization**, any divisions or organizational units, process roles, users and any calendar rules that may apply. This enables organizations to manage what people participate in a process, when they participate, and the scope of authority they have. For processes that span corporate boundaries, directory service referrals are performed.

The development environment

FuegoBPM Studio has the same capabilities to create a project to model the appropriate business processes as the FuegoBPM Designer.

As part of its development environment, for each activity within the process, the business analyst uses **Methods**, a simple scripting tool, to define the appropriate business rules.

FuegoBPM Studio also manages process participants, and allows you to define the **Organization**.

For processes requiring integration with applications, FuegoBPM processes communicate with these underlying application services through components. Components are also cataloged for use through **FuegoBPM Studio**. Separately licensed "technology adapters" are used to connect to common industry standard technologies such as Java, EJB, COM, CORBA/IDL, JDBC/ODBC, XML, JMS and other middleware. The technology adaptors connect to this standard technology instead of a particular application. This allows the component **Catalog** to connect to any object. It has the ability to introspect any object technology and read its methods and properties to create a "wrapper" that directly interfaces with it.

The runtime environment

FuegoBPM Express is a runtime environment designed in such a way that:

- It is self-contained and installs from a single installation object,
- It doesn't require early intervention of IT specialists such as the DBA, the security expert, the webmaster and others to configure the server correctly,
- It does not require a dedicated BPMA (Business Process Manager Administrator) to tune and control a server,

- It does not require a dedicated system administrator to manage users, roles, etc.

FuegoBPM Enterprise is the full runtime environment designed to run processes that span departments, divisions and enterprises.

Once the project modeling stage is complete, the project can be published and installed in the **runtime environment** where the modeled processes start executing.

The runtime environment runs over a different Java Virtual Machine to keep project execution isolated and separated from development changes.

The runtime environment is initiated when the Server is started or when a **Publish & Deploy** operation is performed. From that moment on, to keep the runtime environment updated with the last changes made to the project model, FuegoBPM provides functions that synchronize the **runtime environment** with the **FuegoBPM Studio development environment**.

When the project is **Published and Deployed**, the business rules written in **Fuego Business Language (FBL)** are transferred into Java classes. The business process model is interpreted by the Server directly.

The resultant Java classes are the executable business processes referred to as *supervisory applications*. Then, processes are deployed to the **FuegoBPM Server**, which ensures that each process is executed. The Server communicates with the directory service to determine which processes it will run, which participants will be involved and which components it will use.

When the FuegoBPM Server is started, it is ready to run the supervisory applications to perform the business process by connecting process participants, third party applications and data.

The FuegoBPM Server maintains the state of each executing process

instance, regardless of whether it runs for a few minutes or for months at a time.

When a process activity requires human participation, the FuegoBPM Server *pushes* work to the Organization Participants in charge of doing the job. Participants will have access to the pending work and may access Work Portal through any Internet browser. Work Portal enforces the roles and permissions as defined in the Organization settings and only displays activities relevant to the participant who is currently logged into Work Portal. Additionally, users may interact with or start a process from third-party applications.

FuegoBPM Server and **FuegoBPM Work Portal** execute in the **runtime environment**. All changes to the project model are applied to this environment during Publish and Deploy and every time the server is started.

Changes made to the Organization settings are applied to the **runtime environment** provided that **Refresh Server Data** option is performed from **FuegoBPM Studio**. This function also forces all the changes made to FuegoBPM Work Portal Views to be applied with no delay to the runtime environment in all FuegoBPM Work Portal sessions.

Internationalization

FuegoBPM supports multiple languages for **FuegoBPM Studio** or **FuegoBPM Designer** as well as for the **project definition and design**. The internationalization (i18n) follows the standards to internationalize software.

FuegoBPM Studio and FuegoBPM Designer

You can configure FuegoBPM Studio and FuegoBPM Designer either in English or Spanish (default languages). The language is set at installation time.

If any other language is required, contact your Fuego representative.

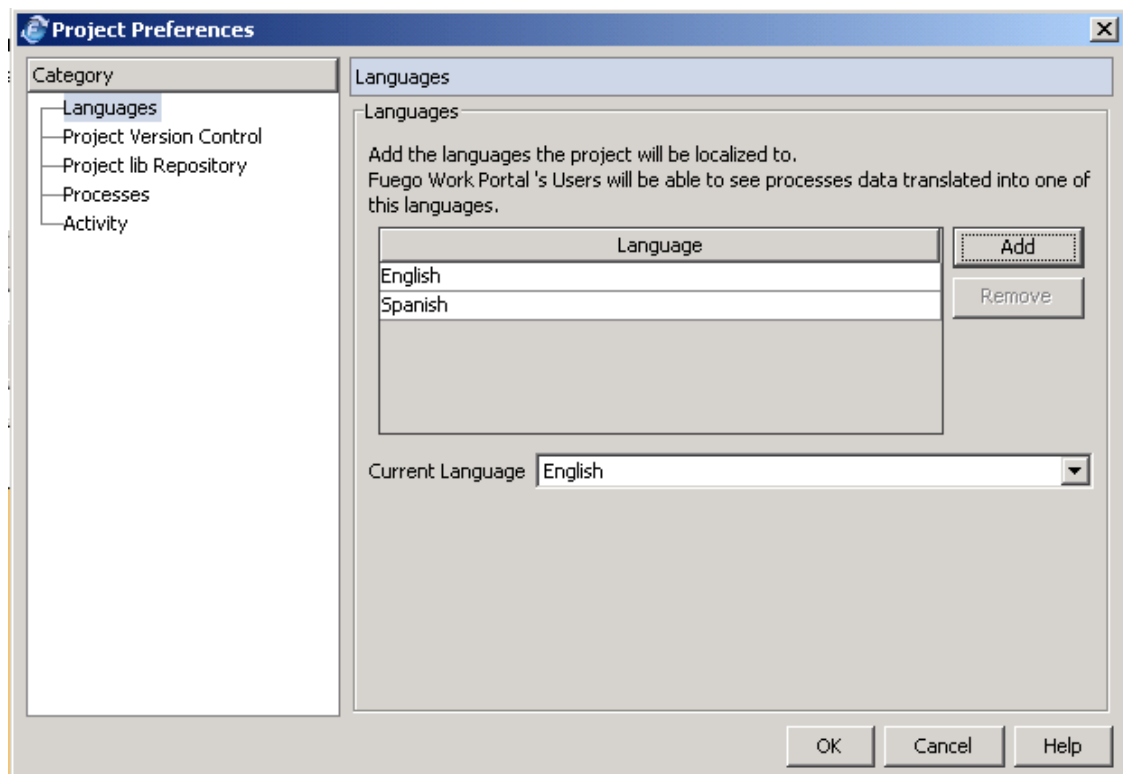
Once FuegoBPM Studio or FuegoBPM Designer are installed, if you want to switch between languages, select from the **View** menu, the **Language** option.

The project languages

In FuegoBPM Studio/Designer, the project can be internationalized; this means that you can write information in different languages. For example, for a *process* you can internationalize the label, description, documentation, use cases, as well as for *Activities* and their corresponding documentation.

The available languages for internationalization are those enabled in the **Project Preferences, Languages** category.

You have to add all the languages in which the designers need to localize all definitions.



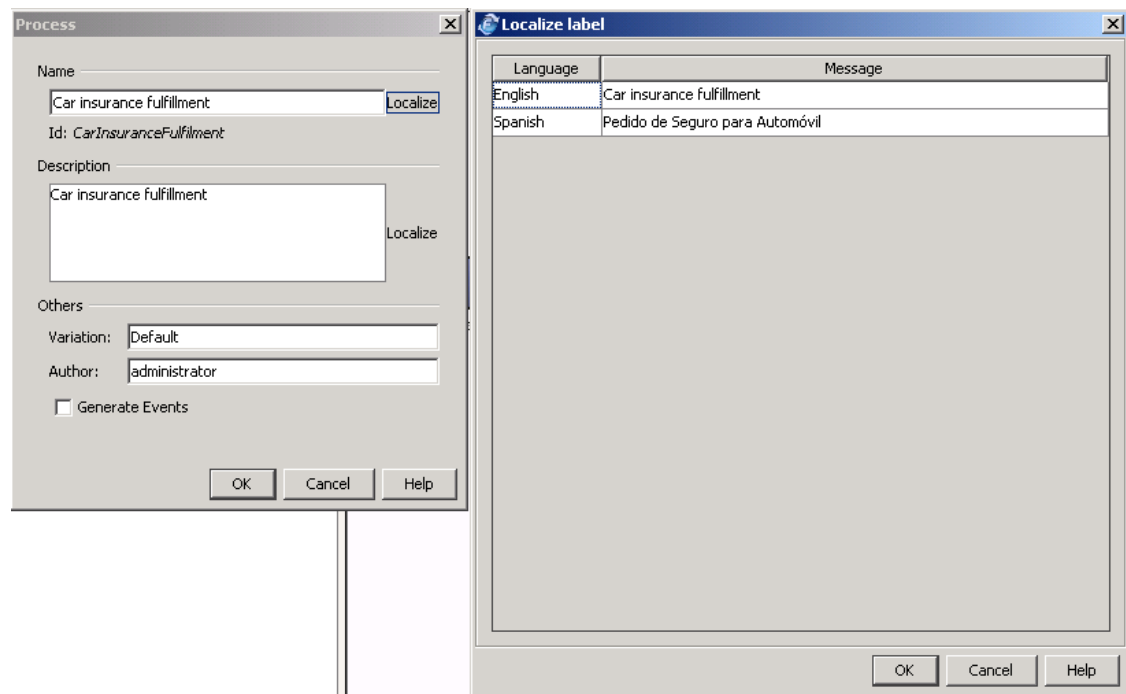
The **Default Language** indicates which language will be used to display labels and information on the developer workspace.

Whenever you see a **localize** option, you will be able to write in the languages defined in the project.

For each project, you need to define what languages the people using the project will require and enable such languages in the project preferences.

For example, if you have Spanish-speaking participants you need to add Spanish to this list.

If enabled, as the designer defines names, descriptions, and so on, they can be **localized**. The list of all enabled languages populate and they can be completed in the different languages.

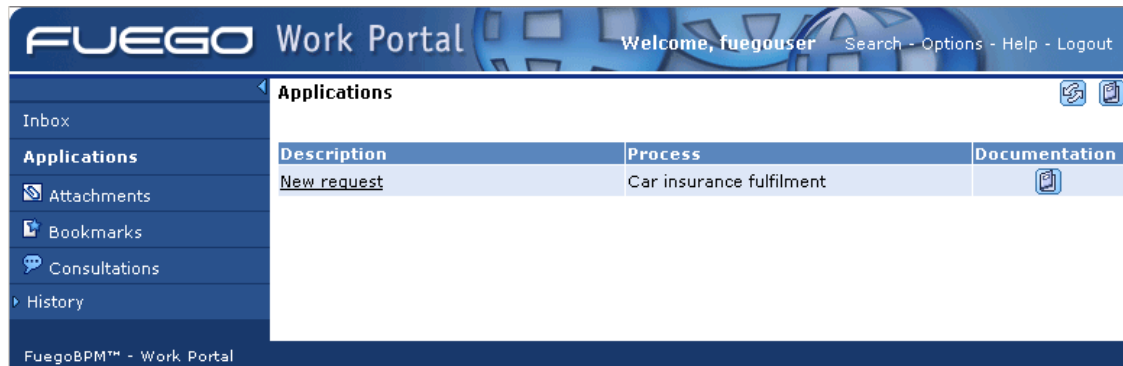


Elements that can be localized are the ones that are visible for participants in the Work Portal. When a participant changes his or her language in the Portal, all elements are displayed in the corresponding language. This is why it is very important to localize all the elements of the project for all participants.

For example, you set your language option in the Work Portal to English:

Options		Help
User Information		
Full Name:	fuegouser	
Login Name:	fuegouser	
E-mail:		
Browser settings		
Enable Flash version menu:	<input type="checkbox"/>	
Enable DHTML support:	<input checked="" type="checkbox"/>	
Settings		
Sort instances by:	Received	
Instances order:	Ascending	
Instances date format:	10:40 AM, 8 Oct, 8 Oct 1980	
Show hidden views:	<input type="checkbox"/>	
Follow the Instance:	<input type="checkbox"/>	
Notify me by e-mail when new instances arrive:	<input type="checkbox"/>	
Keep instance view:	<input type="checkbox"/>	
Enable applet for attachment management:	<input type="checkbox"/>	
Enable remote scripting for FuegoObject presentations:	<input checked="" type="checkbox"/>	
Show applications:	In a folder	
User Working Directory:	/temp/	
	(Including last path separator, ie.: 'c:\temp\').	
Maximum number of searches in history:	10	
Display options		
Number of instances:	10	
Language:	English	
Country:		
TimeZone:	GMT-03:00	
<input type="button" value="Save"/> <input type="button" value="Close"/>		
FuegoBPM™ - Work Portal		

Therefore, all the activities names as well as the process name appear in English in your WorkPortal:



Now, if you change the language to **Spanish**:

Display options

Number of instances: 10

Language: español

Country:

TimeZone: GMT-03:00

Save Close

Then the Work Portal is shown in **Spanish** as well as all the design elements that you localized during project development. In the example below the activity name was changed from **New request** to **Nuevo pedido** as well as the **process name**:



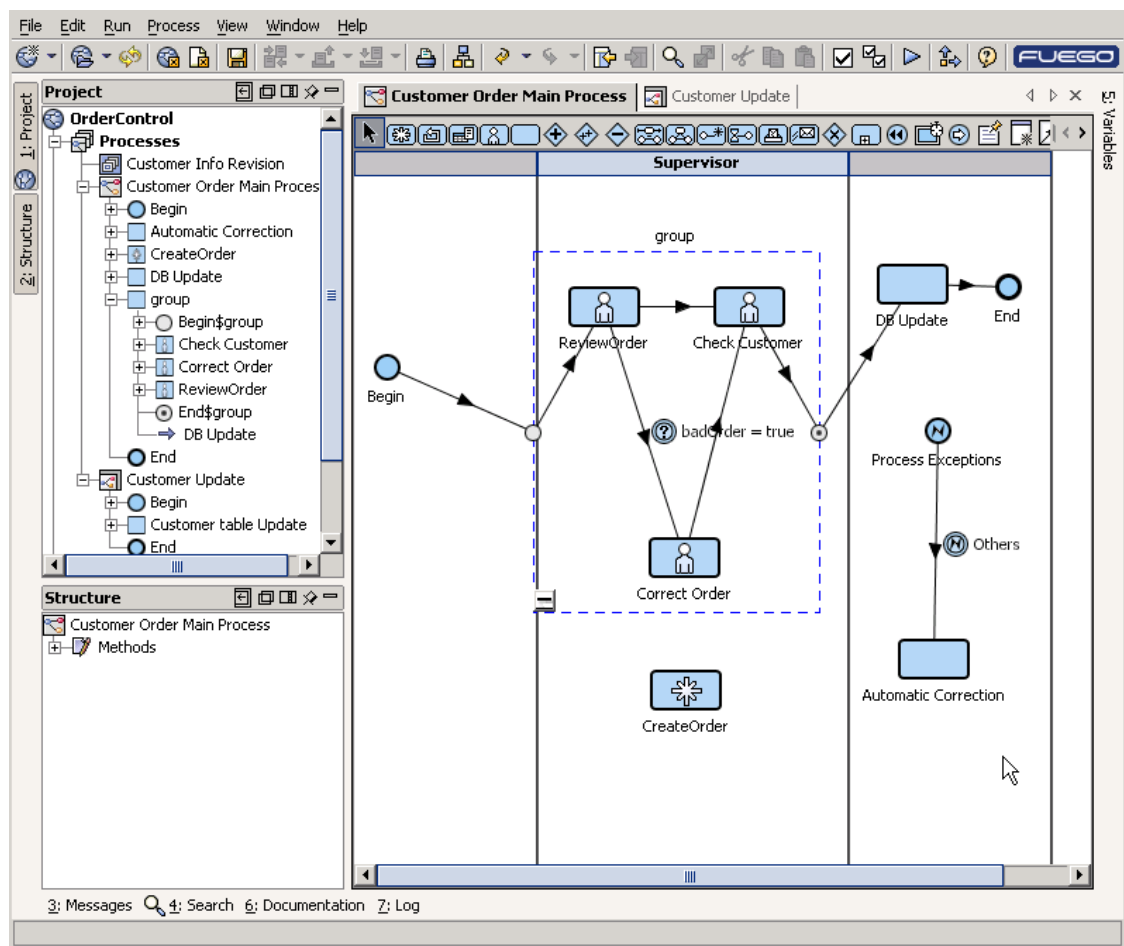
FuegoBPM Workspace

Studio

Development

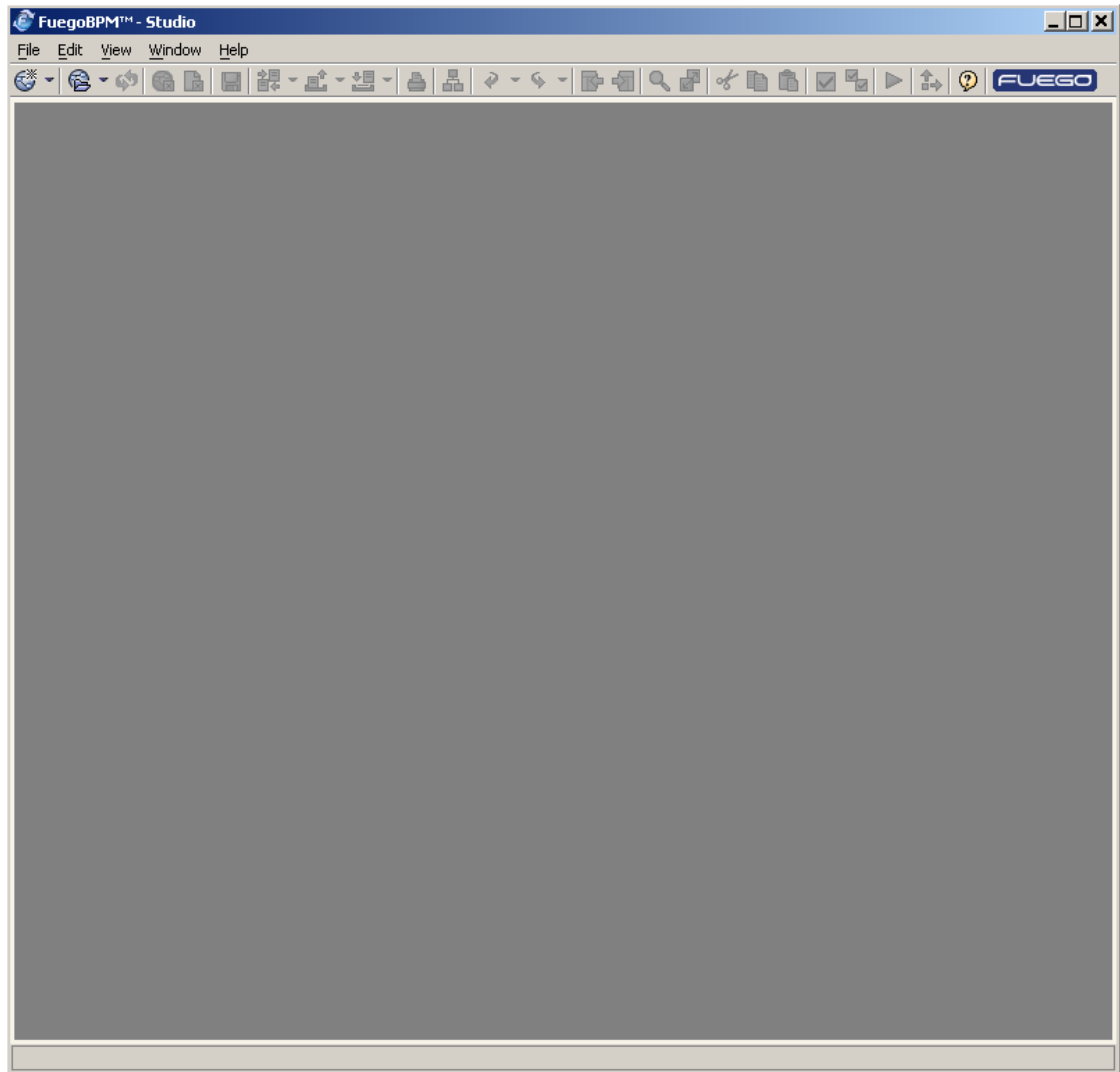
Overview

The user interface of FuegoBPM Studio has been designed to display a main window - where the developer will focus - and several other windows that can be minimized and expanded as needed. The window panels can even be moved from one area to another depending on your preferences. All these windows are stored in an area called 'Desktop'. An example of desktop is shown in the image below.

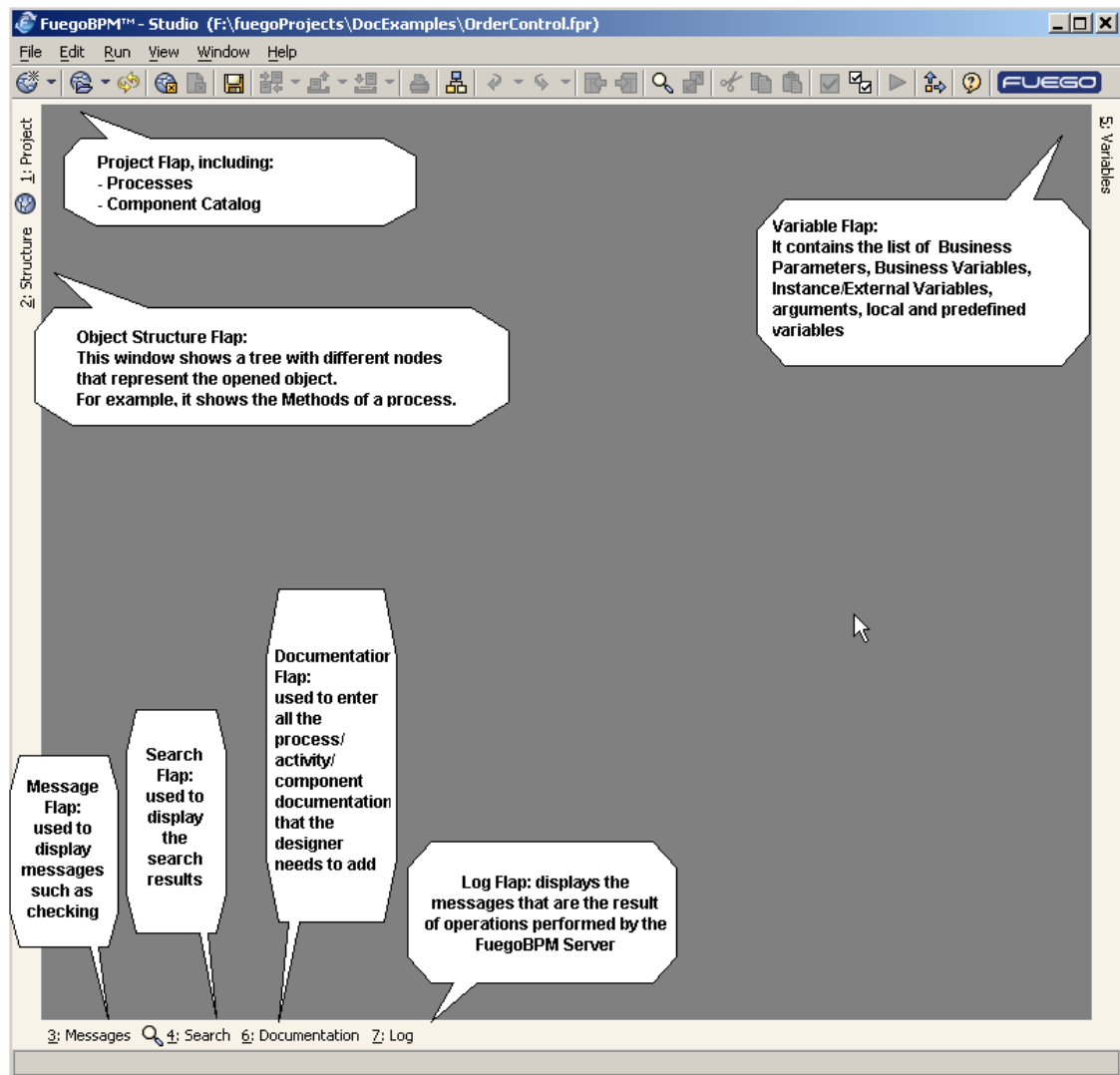


Desktop

The desktop is the area where all the panels are located. When running FuegoBPM Studio for the first time (if no project is opened), the desktop is empty as shown in the following image.



Once a project is opened, the desktop is filled in with flap windows, which allow access to project components. Each flap is described in the following image.



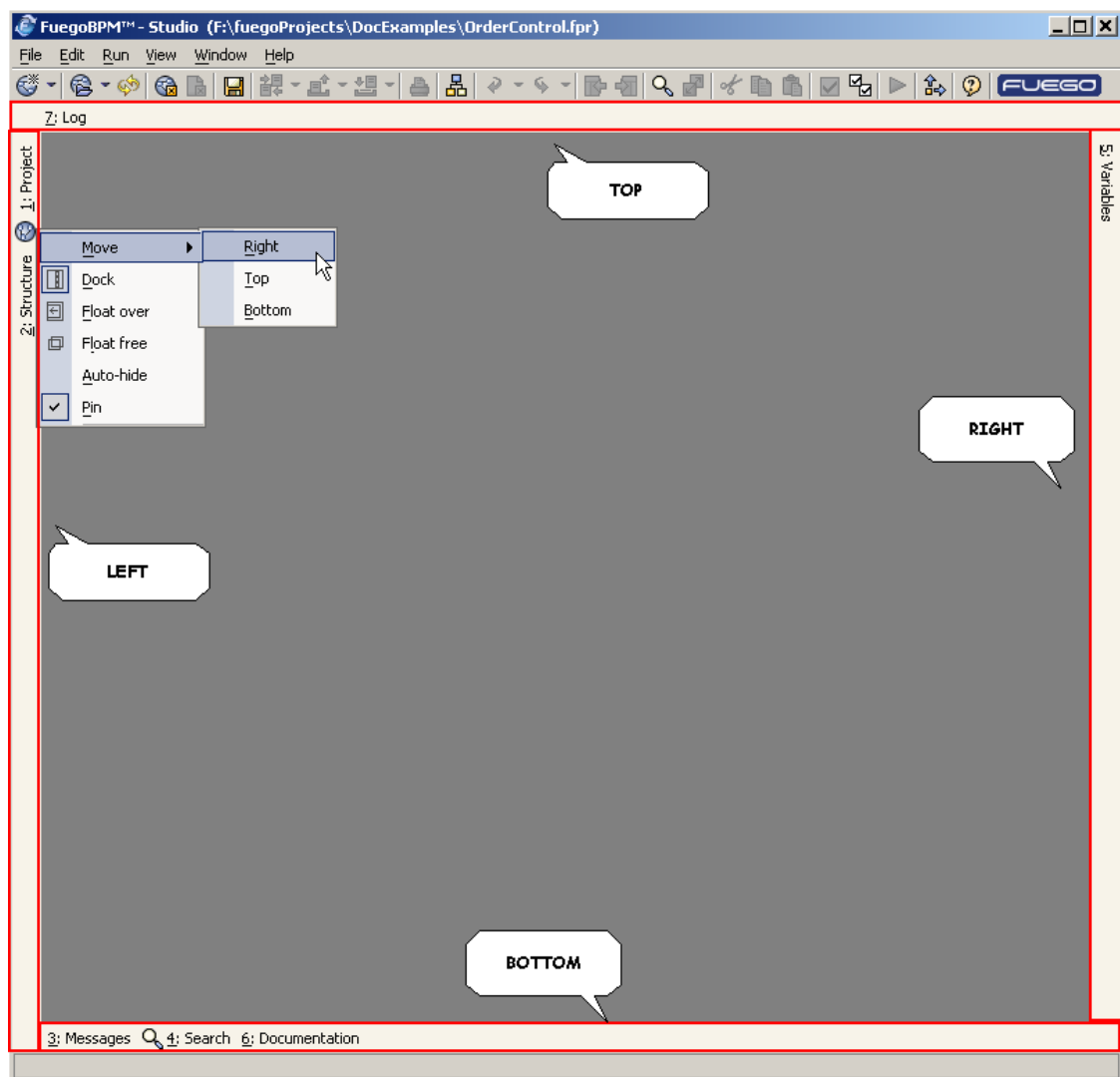
The flaps are the basic panels of the desktop. They can be accessed through the shortcut key 'Alt' followed by the number of the flap. For example, "Alt+1" will open the Project flap. Additional flaps can be opened based on the user action. For example, if a user tries to debug a Method (using the "Play" button in the toolbar), a new flap called "Run" is opened with the debugger. This flap disappears once the user has finished using the editor. Hence, no shortcut key is provided for it.

Flaps (Desktop Windows)

A **flap** is a panel that can be opened when needed and has the ability to be located on the different docking areas of the desktop.

Each **docking area** has an associated "toolbar" displayed on the border where the flap titles are shown. When these buttons are selected, if the flap is currently visible, it will be hidden; and if the tab is hidden, it will appear in its current dock area.

The docking areas are highlighted in the following image.



You can rearrange the default layout of the desktop to suit your preferences. All modifications made to the desktop layout are automatically saved.

Flaps also provide different modes to aid the user when building a process, writing methods, etc.

Flaps Properties:

- Move
- Docked
- Float over
- Float Free
- AutoHide
- Pin
- Minimize

Flap possible positions and states:

- *Docking Area*
 - Top
 - Bottom
 - Right
 - Left
- *Flap positioning "layer"*
 - Component layer: it is placed in a split panel in the same layer

as the central component.

- Floating over (upper layer): the component is placed in a different layer from the central component and is displayed over it.
 - Floating free (different window): the component is set "free" and displayed in a non-modal dialog.
-
- Autohide: when a flap is set to "autohide" mode, it will disappear whenever a component of a different flap is selected. This hierarchy is defined as
 - The central component.
 - The docked flaps/components and the floating free flaps/components.
 - The floating over flaps/components.
 - Pin or lock: The docking area is a shared space, when a flap is not pinned, it is hidden whenever another flap is selected. When a flap is pinned, it will not be hidden, but it will remain visible and share the area with other flaps that are selected.

Each docking area has a toolbar or shortcut bar. These toolbars contain buttons that act as shortcuts to view or hide the panel.

User Interface Features: The selected flap label is highlighted for visible components.

Note



When a flap is set to "floating over", it is always set to work as auto hide. The auto hide button is not shown.

Flaps Documentation

To access the documentation of each Flap, select it and press F1.

FuegoBPM Studio navigation

You can simultaneously open multiple panels in the central panel. Each panel might contain a process, a BP-Method, etc.

You can navigate through the panels that are already opened. Options on the **Window** menu—"Backward" and "Forward"—to go back and forward from the current panel.

You can also move from one panel to the other by clicking on the panel's top tab or right-clicking and selecting the **Previous** or **Next** options.

Navigation is particularly useful when editing multiple processes from the same project as well as BP-Methods from the process.

Studio opens only one script editor for each process - or component. Navigation buttons allow you to return to a previous script for editing.

FuegoBPM Studio Menus

File

- **New**
 - Project: See Defining a Project.

- Folder: To organize your project and processes, you can create a folder in which you group processes, screenflows or procedures. Be aware that if you delete a folder all processes within it are also deleted. If you want to keep the processes, **move** them to another folder or to the *Processes* entry within the tree.
 - Process: See Process.
 - Procedure: See Procedures.
 - Screenflow: See Screenflows.
 - BPEL Process: See BPEL Processes.
 - Module: See Modules.
 - Fuego Object: See Fuego Objects.
 - Enumeration: See Enumerations.
 - Exception: See Cataloguing Exceptions.
 - Transformation: See Transformations.
 - Catalogue Component: See Components.
 - External Resource: See External Resources Configurations.
-
- **Open**
 - Project: See Defining a Project.
 - Process: See Process.
 - Component: See Components.

- **Open Recent:** All recently opened projects are listed so that you can choose one of them. If you select **Clear list**, this list is cleared.
- **Import**
 - Project (from repository or from file): See Defining a Project.
 - Designs: See Process.
 - Enumeration: See Enumerations
 - Fuego Object: See Handling Fuego Objects.
- **Close Project:** See Defining a Project.
- **Close:** Closes, from the main panel, the selected tab (process, BP-Method editor, etc.)
- **Save:** See Saving a project.
- **Revert to saved:** See Saving a project.
- **Synchronize Project:** See Synchronizing the project

- **Store project in repository:** See Version Control System - Store Project to the repository.
- **Organization:** See Organization.
- **Export project:** See Defining a Project.
- **Project Report:** See Generating Project documentation.
- **Project Preferences:** See Project Preferences.
- **Page Setup:** Define the setting for printing.
- **Print/Preview:** Displays a Preview of the selected process.
- **Print:** Prints the selected process.
- **Studio Update:** Allows an automatic update of Service Packs. To update FuegoBPM Studio, download the Service Pack (*.upd* extension) to your computer. Launch the Studio and close any opened project. Run the Studio Update. Once the *.upd file is selected and confirmed, the update takes place.

- **Check for Updates:** When you start FuegoBPM Studio, you can set the Studio preferences to automatically check for new updates. If you prefer to update manually, you can **Check for Updates** at any time by selecting this option.
- **Preferences:** See Preferences.
- **Exit:** Exit from FuegoBPM Studio

Edit

- **Undo:** select to undo any editing change performed.
- **Redo:** select to redo any editing change undone.
- **Cut:** select the object and select **Cut** to remove it to the clipboard. You can then paste the object in another location.
- **Copy:** select the object and select **Copy** to copy the object to the clipboard. You can then paste the object in another location.
- **Paste:** paste the cut or copied object.
- **Find:** find a string in any label within the project, as well as a component name or a word within a method. In some case, within the right-click menu in the left tree, for example on Processes or Catalog, you can search using the *Find in path*. The search and find is performed restricted from that point of the tree.
- **Replace:** the Replace option is enabled when editing a Method. Find the selected string and replace it with a new one.

- **Go to:**
 - Open: select a process to open.
 - Find Component: find a component.
 - Go to line: if you are editing a Method, you can go to the selected line.
- **Properties:** open the properties dialog for the selected object.
- **Backward:** goes to the previous opened process tab.
- **Forward:** goes to the next opened process tab.
- **Open Previous Member:** open previous Fuego Object member (attributes and methods.)
- **Open Next Member:** open next Fuego Object member (attributes and methods.)

Run

- **Check All:** check all processes. See Process Overview for further information.
- **Publish & Deploy:** See Publish & Deploy for further information.
- **Start Server:** start the server. See Server for further information.
- **Stop Server:** this property is only available if the Server is running.
- **Clean Project Database:** deletes the Directory Service, FuegoBPM Server and BAM database transactions. This option is

useful if you are testing a process that adds participants or roles defined in your organization. Each time you want to test the process you need to clean the directory database.

- **Refresh Server Data:** this property is only available if the Server is running. It updates any Organizational data in the Server.
- **Launch Work Portal:** launches the FuegoBPM Work Portal. FuegoBPM Work Portal is a web interface tool that allows you to interact with a process in relation to your assigned role or roles within your company. See **FuegoBPM Work Portal documentation** for further information.
- **Server Preferences:** See Server Properties, Runtime, Deployment, Deployed Processes (you will see this option only if the Server is running), Referrals.

Process

See Process Menu for detailed information.

- **Run**

If you are editing a

- **Process:** runs the Simulation of the process (See Simulation for further information.)
- **Screenflow:** see a virtual run of how screens will flow.
- **Method:** you run the debug of the method.
- **Fuego Object:** you get a preview of the presentation.

View

The View menu options vary according to what you are working with.

Options enable when working in the process designer:

- **Show Roles Horizontally:** change the orientation of the process graphic as the Roles columns are shown on the right.
- **Activities**
 - Show Grab activities: if checked, it shows Grab activities. See Grab activity for further information.
 - Show Notes: if checked, it shows Notes within a process. See Notes for further information.
- **Groups**
 - Begin and End Orientation: See Groups (notes) .
- **Transitions**
 - Show Grab Transitions: if checked, it shows transitions coming out and in a Grab activity. See Grab activity for further information.
 - Show Unconditional Transitions: if checked, it shows Unconditional Transitions.
 - Show Conditional Transitions: if checked, it shows Conditional Transitions.
 - Show Due Transitions: if checked, it shows Due Transitions.
- **Exceptions**

- Show Exceptions: if checked, it shows Exceptions Flows. See Process Exception Flow for further information.
- **Themes:** visualize all activities in the designer as Classic, UML, Business Analyst or BPMN (Business Management Process Notation).

- Classic



- UML



- Business Analyst



- BPMN



- BPMN Color



- **Grid**

- Show Grid: show the grid on the designer workspace.
- Snap to Grid: center the activities in the nearest grid.

- **Grid Settings:** set the distance (pixels) in between the grids.

Only the common options are enabled when working with the project catalog. If you are working with a Fuego Object presentation, the following options are displayed:

- **Show symbolic view:** if selected, it displays the components within the presentation with the icon that represents it instead of its real aspect.
- **Show dotted lines:** if selected, delimits each element of the presentation by dotted lines. Dotted lines are very useful during design to help visualize the location of elements in the presentation.
- **Presentation Preferences:** you can configure the default preferences that are used when a presentation is created.

See [Designing a Form](#) for further information about presentation views.

See [Adding presentations to a Fuego Object](#) for the detailed options to set the presentation preferences.

Common options:

- **Language:** set the FuegoBPM Studio's language.
- **Show Hidden components:** Shows or hides the project catalog components set as hidden.
- **Full Screen:** maximize FuegoBPM Studio to a full screen.

Window

- **Forward:** goes to the next opened process tab.
- **Backward:** goes to the previous opened process tab.

Help

- **Contents:** opens the FuegoBPM Help Viewer with FuegoBPM Studio documentation.
- **Tip of the day:** see all tips of the day.
- **Component Index:** you get a list with all the Fuego standard components, the category or module to which it belongs to, and a brief description about it. If you double click on the component, a new tab in the main panel opens for that component.
- **About:** information about the product and the company.

Customizing the Studio toolbar

You can customize the Studio toolbar.


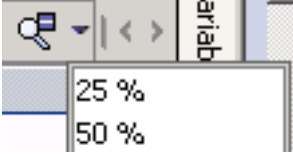
Right click in between the **Help icon** and the **Fuego logo**. The **Customize** option is displayed. Enable or disable the functions you want to see in the toolbar that is opened.

FuegoBPM Studio Designer toolbar

You can choose to add any activity, group, connector, note, role or transitions from the Designer toolbar directly.

You can also change the size of the process design by doing one of

the following:

- selecting the **zoom in** or **zoom out** icons  at the end of the toolbar, or
- selecting the size of the picture from the drop-down menu at the end of the bar , or
- pressing Ctrl + wheel mouse to zoom in or out.

Project Flap

See Defining a Project.

Process / Procedure / Screenflow

All processes, procedures, and screenflows are displayed in the Project flap.

See Process.

See Procedures.

See Screenflows.

Project Catalog

All modules and their components are displayed in the Project catalog. Default modules, **java** and **Fuego** and their components are included.

See Introducing Business Objects into FuegoBPM.

See Modules.

See Components.

Fuego Object

See Implementing Business Objects Using Fuego Objects to learn what is and how to create a Fuego Object.

See Fuego Objects Attributes.

See Fuego Objects Methods.

See Fuego Objects Presentations.

Structure Flap

Within this flap, contextually, different structures can be displayed:

Method's structures

If you are working in the main panel with a process, all BP-Methods and their structures are displayed in this flap.

Component's structures

If you are working in the main panel with a component, then the Structure tab displays all the component's elements as attributes, methods and presentations (if the component is a Fuego Object.)

If the opened component is a Fuego Object, and you are editing one of its presentations, you can select to either the Fuego Object or the presentation structure. In order to do this, select **Structure** or **Presentation** from the drop-down on the top right corner of this flap. If you select *Presentation*, you can collapse or expand the structure of the component by clicking the icon on the left top corner of this flap. If you select *Structure*, then you can group or ungroup the inherited elements by clicking the icon on the left top corner.

If you are visualizing a heir Fuego Object, (a Fuego Object implementing "behavior inheritance"), keep in mind that the

inherited elements are displayed as virtual elements, meaning they cannot be removed or edited.

Messages Flap

All messages are displayed in this flap. For example, the checking results.

Search Flap

Searching results are displayed in this flap.

Variables Flap

Within this flap, contextually, different variables can be displayed:

Processes Variables

See Using Variables.

Fuego Objects Variables

If you are working on a Fuego Object, the variables flap contains the list of attributes of the Fuego Object. See Fuego Object Attributes.

Properties Flap

This flap is only available when working with components.

Components: See Components.

Fuego Objects: See Fuego Objects Overview

Fuego Objects Methods: See Fuego Object Methods

Fuego Objects Presentations: See Fuego Object Presentations - Defining the Structure

Fuego Objects Presentations elements: See Adding Components to the Presentation

Documentation Flap

See Documenting a Project.

Log Flap

Log results are displayed in this flap. See Logviewer for detailed information.

Simulation Flaps

See Simulation for detailed information.

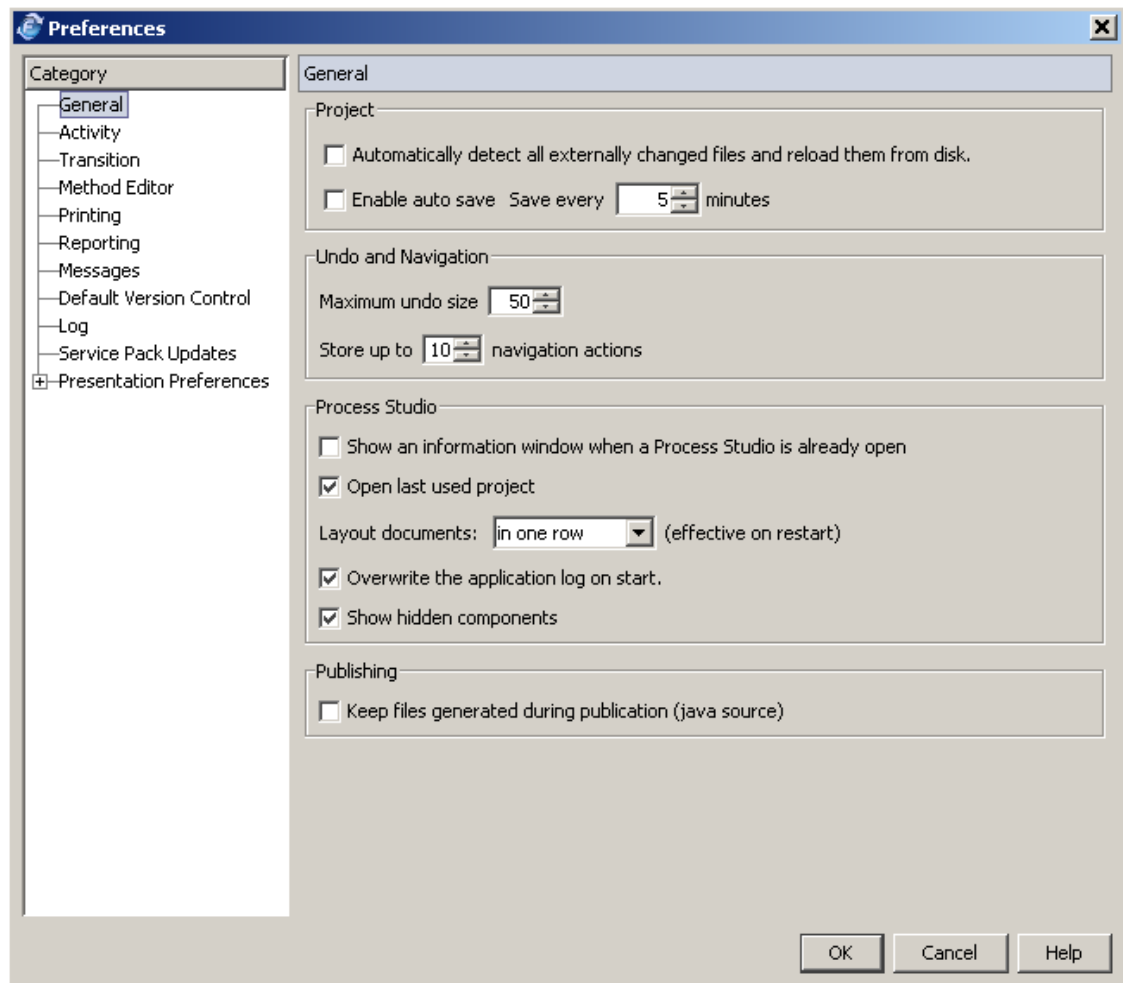
Tips

- You cannot open more than one code editor per process.

FuegoBPM Studio Preferences settings

Select **File** and **Preferences** from the menu options to open the **Preferences** dialog box. Preferences can be set in a variety of ways as explained below.

General



- **Automatically detect all externally changed files and reload them from disk:** the project synchronization is done automatically. See Synchronizing the project.
- **Project / Enable autosave:** See Saving a project.
- **Undo and Navigation:** Set the number of operations you can undo as well as the number of navigation operations to store.
- **Process Studio:**
 - Show an information window when the Studio is already opened: If this option is selected, when trying to open a second FuegoBPM Studio, a warning message appears: *The*

Process Designer is already running.

- Open last used project: opens the last project and component you were working with, for example, a Fuego Object, a method, processes.
- Layout documents: Define the way in which you want the tabs that display the processes, method's editor, Fuego Object, etc. in the central panel.
- If you select **in one row**, it is possible that the tabs may not all be visible in the central panel. Scroll to the right or left using the arrows shown to the right in the image below.



- If you select **in multiple rows**, the tabs are displayed in one or more rows and all tabs are shown in the central panel.

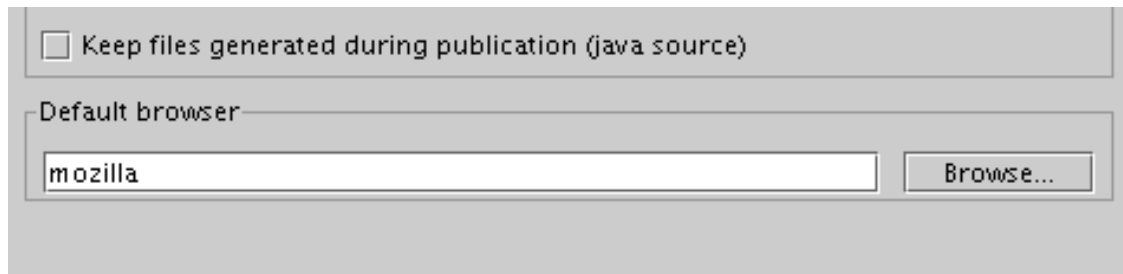


- Overwrite the application log on start: The studio log files are deleted and re-generated with all the new logs or logs that are appended to the existing files (within the FuegoBPM Studio installation directory, log/studioStderr.log and log/studioStdout.log)
- Show Hidden components: Shows or hides the project catalog components set as hidden. This preference can be changed by selecting/unselecting the option *Show hidden components* in the *View* menu of the Studio.
- **Publishing:** Keep files generated during publication: when you publish a process, a set of Java files is generated in the project

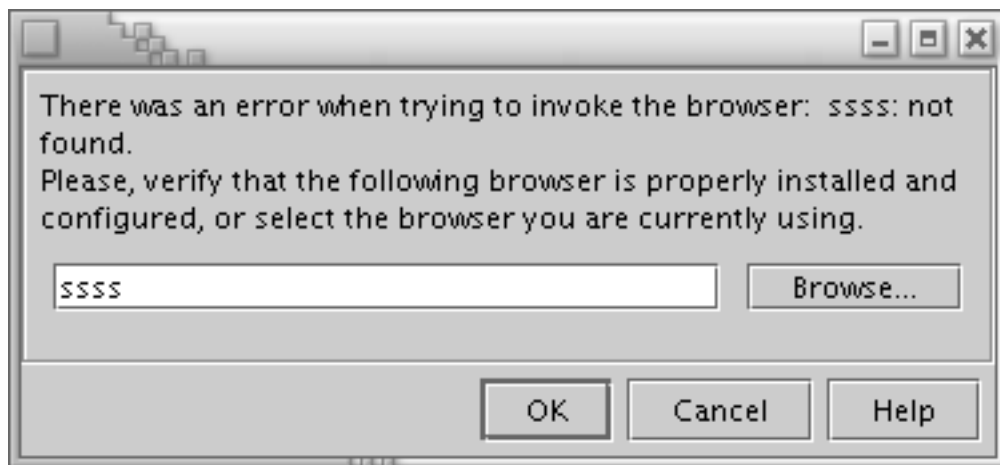
directory in build/output. These files are kept if you enable the preference. This preference is applicable only if you require any special support, these files are requested to review the problem.

- **Default browser:** Only available in Unix.

In Unix, it is possible to configure the browser to use when launching Work Portal and Portal Admin, or when opening documentation files and Help contents. This is done in the File -> Preferences menu, by selecting the command name that will be executed when a browser is needed.



If the execution of this command fails, a dialog box is displayed asking you to change the command until a valid command is specified or the execution of the browser is cancelled.



Activity

- **General**

- *Show properties automatically when adding a new object:* If enabled, when you add a new activity in your process design, the **Properties** dialog box automatically opens allowing you to complete the activity's property information.
- *Keep adding activity mode:* If enabled, you can keep on adding the same kind of activity you have chosen. Each time you click on the selected activity, it is placed on the designer workspace. To stop this mode, press the Esc key on your keyboard. In this mode, the Properties dialog box will not appear after adding the activity.
- *Show tooltips:* When you allow the mouse pointer to hover on an activity, a tooltip yellow window displays showing some information about the activity. To turn off this function, disable this preference.

- **Messages**

- *Show confirmation when replacing activity task (using drag and drop):* You can drag a BP-Method from the left panel (Structure panel) and drop it on an activity. The task is automatically set to run this Method. In addition, you can drag a defined Screenflow or Procedure in the Project from the project panel. If you drag and drop to an **Automatic** activity, the task implementation is replaced with the new selected action. If you enable this preference, you are asked for confirmation to proceed with the replacement in Automatic activities.
- *Automatically insert activity when dropped over a transition:* When you drop an activity over a transition, it is automatically inserted into that position. If this preference is enabled, no confirmation

is requested.

- **On double-click show:** When you double-click an activity, you can choose to automatically open the **Properties** window or the **Main task** window. If the activity is an Interactive or Automatic one, and the main task is not present, the double click creates a Method task and opens it.
- **Show:** The activity's title within the process designer can display either the activity's **Name** or the activity's **Description**.

Transition

See Transitions.

Method Editor

- **Language / Style:** The method is shown in Fuego-BP, Java or Visual Basic .Net syntax.
- **Auto complete:** The list of possible options to complete a component invocation is displayed one second after the "." (dot) is written.
- Others

- **Allow the debugger to commit transactions:** When running a method in debug mode, if this property is checked, then the transactions embedded in your Method will be committed. If not, all transactions will be rolled back when the method stops running.

Printing

- **General / Complete page size with last role:** The process is printed completing the page with the last role or it prints the last role and the page is kept in blank until the end.
- **Transitions / Print conditional transition information:** Shows the transition condition on the designer graph.

Reporting

- **Include use cases:** As you design a process you can document the use cases. At reporting time you can choose to include it or not. See Documenting a Project.
- **Include methods resources:** See Generating Project documentation.
- **Include variables:** See Generating Project documentation.
- **Confirm when report file exists:** to avoid overwriting an existing file, a confirmation is requested.
- **Always show options before report:** Before you generate the documentation, you can choose what to include and what not.

See Generating Project documentation for further information.

Messages

Determine what confirmations you want to be displayed when you delete certain objects in your design (process, folder, method, resources, variables or transformations.)

Default Version Control

See Version Control System.

Log

See Log viewer

Service Pack Updates

Check for updates on start up: You can set Studio to automatically check for new updates each time you launch it. The updates are downloaded from a Fuego URL. If any problem arises with the URL, a log is posted to log/studio.log

Always show the option of checking for new service pack availability: If this option is enabled, when starting the FuegoBPM Studio, a dialog box appears. Click **Yes** to check for updates.

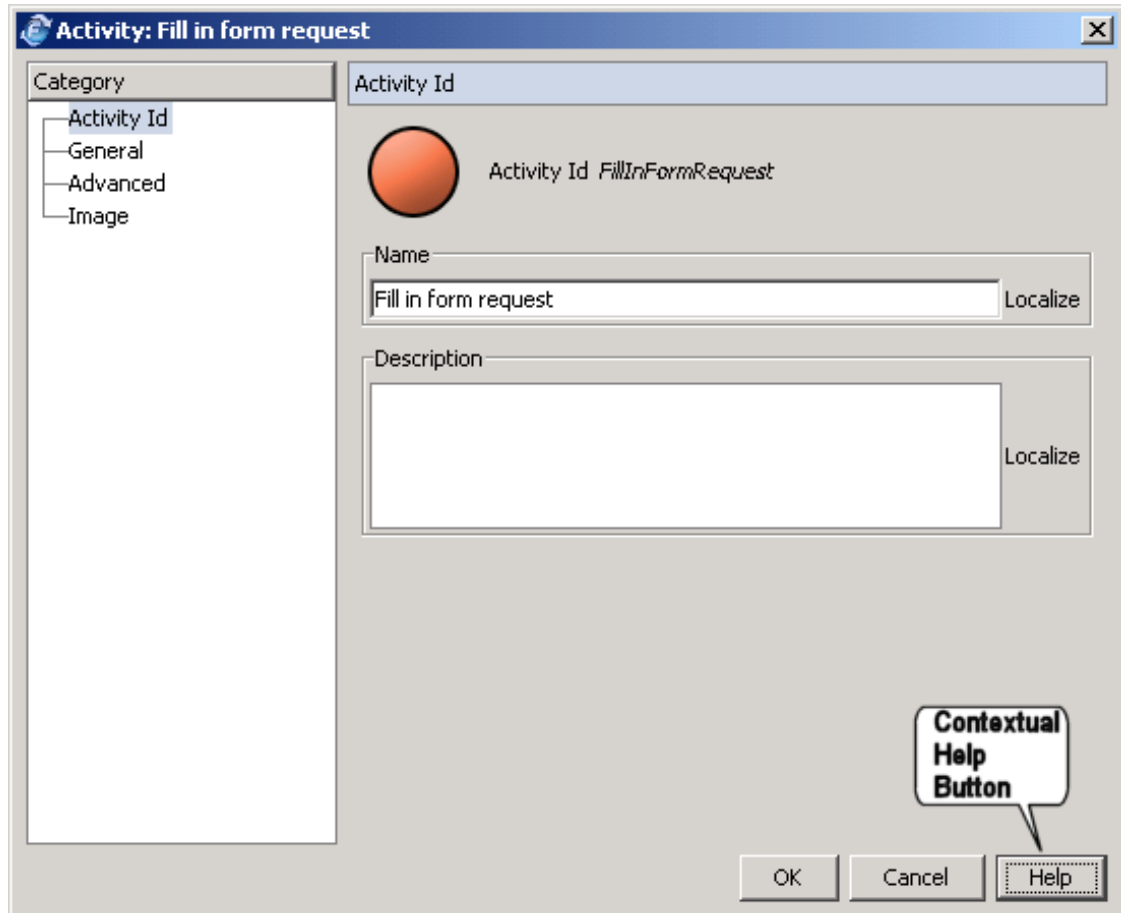
Server URL: Indicates the URL from where new updates are downloaded.

Presentation Preferences

See Adding Presentations.

Contextual Help

As you work with FuegoBPM Studio, you will find contextual help available. Find the **Help** button within the screen and click on it. The FuegoBPM Help Viewer is opened with the information on the feature you are working with.



Flaps Documentation

When you are working within any of the Workspace Flaps such as the **1. Project Flap**, **2. Structure Flap**, **3. Messages Flap**, etc; to access the documentation of each Flap, select it and press F1.

Introduction to FuegoBPM Studio

Administration Guide

FuegoBPM Studio's Installation

After installing FuegoBPM Studio some directories are generated. The most relevant are defined in the following sections.

bin directory

This directory contains all the executable files to run FuegoBPM Studio. The most important one is the *fuegostudio* executable that runs FuegoBPM Studio.

log directory

All FuegoBPM Studio log files are saved in this directory. The Fuego support team might require these log files while providing you support.

There are normally three files in this directory:

- **studioConsole.log** : these are the console output of the Studio program, these files are overwritten each time Studio is started.
- **studio.log**: this is the application log. This file contains warnings, etc., generated by the application. This file is by default overwritten each time Studio starts. You can change the default to append to the file by selecting **File -> Preferences -> General window**. Clear the check mark from the **Overwrite the application log on start** check box.
- **Fuego_Studio_5.x_InstallLog**: the log file generated during Studio installation.

studio.log file

Possible logs posted to this file:

- **Problems with the Studio update URL**

If you have set an automatic update when starting FuegoBPM Studio and there is any problem with the server defined in the URL then the following log is posted:

```
Error trying to connect to update server.
    Caused by: Server returned HTTP response
                code: 405 for URL:
                http://www.fuego.com/sp/servlet/updaterServlet

fuego.patcher.applier.client.RemoteUpdateException:
    Error trying to connect to update server.
    at fuego.patcher.applier.client.RemoteUpdate.
        getAvailablePatchs (RemoteUpdate.java:115)
    at fuego.patcher.applier.client.PatcherUpdateAction.
        downloadServicePacks
        (PatcherUpdateAction.java:62)
    at fuego.designer.action.CheckForUpdates.
        updateUsingAvailableServicePacks
        (CheckForUpdates.java:98)
    at fuego.designer.DesignerApplication$1.run
        (DesignerApplication.java:465)
    at java.awt.event.InvocationEvent.dispatch
        (Unknown Source)
    at java.awt.EventQueue.dispatchEvent
        (Unknown Source)
    at java.awt.EventDispatchThread.pumpOneEventForHierarchy
        (Unknown Source)
    at java.awt.EventDispatchThread.pumpEventsForHierarchy
        (Unknown Source)
    at java.awt.EventDispatchThread.pumpEvents
        (Unknown Source)
    at java.awt.EventDispatchThread.pumpEvents
        (Unknown Source)
    at java.awt.EventDispatchThread.run
        (Unknown Source)
    Caused by: java.io.IOException: Server returned HTTP
        response code: 405 for URL:
        http://www.fuego.com/sp/servlet/updaterServlet
    at sun.net.www.protocol.http.HttpURLConnection.
        getInputStream (Unknown Source)
    at fuego.patcher.applier.client.RemoteUpdate.
        getAvailablePatchs (RemoteUpdate.java:110)
    ... 10 more
```

To correct the problem, review the **Server url** field defined in FuegoBPM Studio Preferences in the Service Pack Updates window.

Studio Log Configuration

Studio log configuration is independent from the server log configuration. The Studio log is fixed while the server logs can be configured.

Studio logs are written in the *[installdir]/log/studio.log file*, while the server logs are in the project's directory in the system directory.

Studio will log all severities with the exception of DEBUG in a default installation.

- Java asserts are disabled in a default installation.
 - Enabling Java asserts implies that Studio DEBUG messages will also be logged.
 - To enable Java asserts you must manually edit the `fuegostudio.lax` file and add the following line:

```
lax.nl.java.option.additional =-ea
```

The Studio Project log files

In FuegoBPM Studio, project log files are saved in the directory created when you saved your project under the directory called *system*. For example, project log files can be found in *YourProject.fpr/system*.

The *system* directory will contain up to five files named **YourProject.log.0** , **YourProject.log.1**,..., **YourProject.log.4**.

These files are used to save all the logs the Server, the Portal and the Studio generate. The files are revolving. When the first file fills to capacity, a second file is created. When all five files fill to capacity, the first file is overwritten and re-used.

This log is used by any component within the Studio to log its actions, for example, when publishing.

Generating a Full Thread Dump file

Full thread dumps are only required occasionally for **support purposes**. To generate a **Full Thread Dump**, press the Ctrl-Shift-F12 keys. The dump is saved in the **studioStdout.log** file located in the Studio log directory.

FuegoBPM Studio Update

FuegoBPM can be updated by applying a Service Pack (SP) or a Hotfix.

It is recommended that all Fuego Applications are updated to the same Service Pack or Hotfix. This will prevent eventual compatibility problems when migrating and deploying a project implemented with a FuegoBPM Studio version and deployed into a different one.

Another recommendation prior to applying a Service Pack or Hotfix is, when updating an Enterprise for J2EE is to backup the Application Server as well as the deployed Applications.

Service Pack

A Service Pack (SP) is an update of the software that contains bug fixing and minor improvements. This is a packaging mechanism used by Fuego to deliver these bug corrections and improvements to the Fuego Customer Base.

The Service Pack packaging **ONLY** contains the delta of the modified files based on the previous Fuego Service Pack or release.

They are not acumulative, therefore, for example, to apply SP3, you first have to apply SP1 and SP2.

In most of the cases, the Service Pack packaging is significantly smaller than the Installer package.

FuegoBPM can be configured to point to a URL at Fuego where new service packs will be published when available. However, this is not the recommended approach for getting service packs for UAT or Production Environment. Manual Service Pack application is recommended for UAT or Production environments.

Hotfix

Fuego HotFixes is a packaging mechanism used by Fuego to deliver corrections to the customer before the closure date of a Service pack. HotFixes (HF) contain corrections for Production Halted situations and also corrections that require very complicated workarounds. Essentially, the HotFix allows Fuego to release a correction ahead of service Pack release time. All corrections included into a HotFix will be included in the following Service Pack based on internal procedures enforced by Fuego. The Fuego customer can install safely a HotFix knowing that these corrections will not be lost when applying the next available Service Pack released by Fuego.

They are acumulative. That means that when you apply the latest hotfix, you are applying every other HotFix released previously for your particular version.

A HotFix can only be applied manually after getting it in the Fuego's Customer Support Download page.

Applying a Service Pack to the FuegoBPM Studio

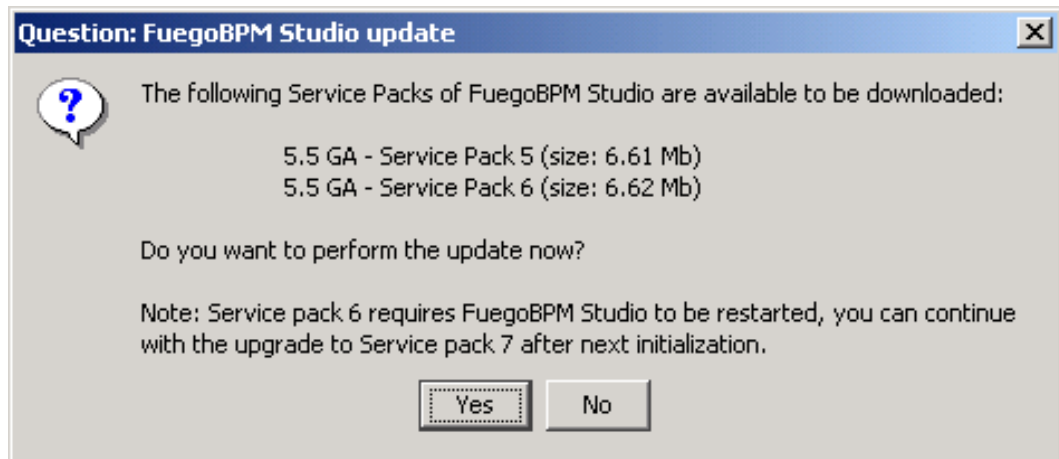
Configure Service Pack preferences

Refer to FuegoBPM Studio Preferences, to learn how to configure the *Service Packs Update*.

Applying a Service Pack OnLine

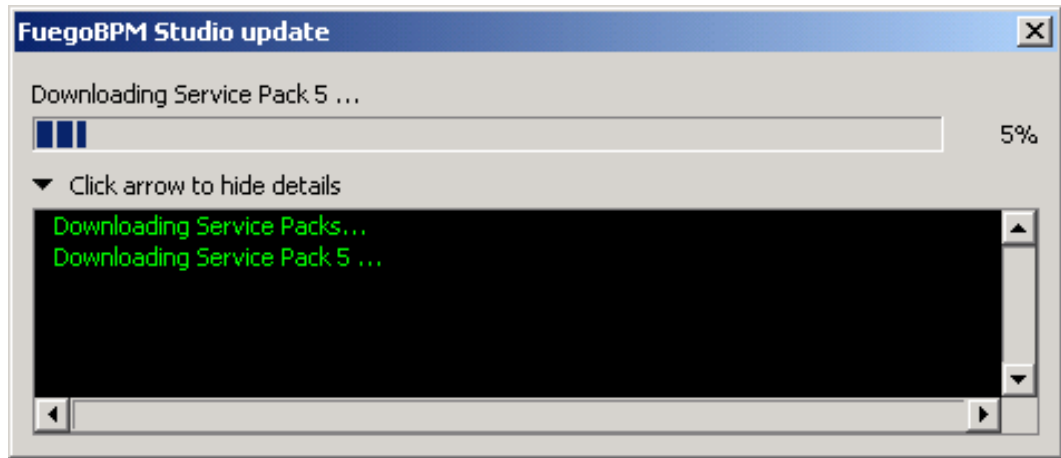
To apply a Service Pack online,

1. Run the option *Updates/Check for Updates* from the main menu **File** of the FuegoBPM Studio,
2. The *Check for Updates* detects which are the possible SPs to apply to your installation.

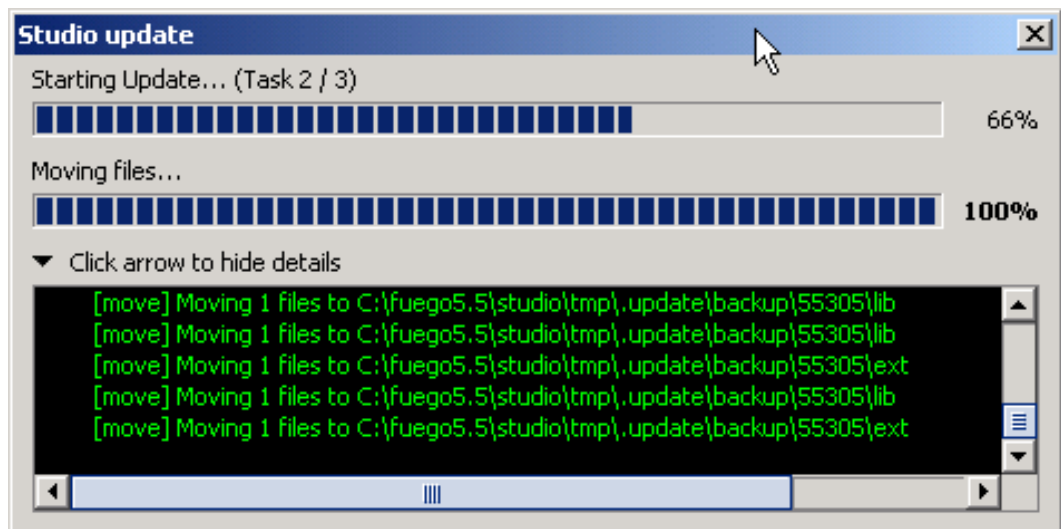
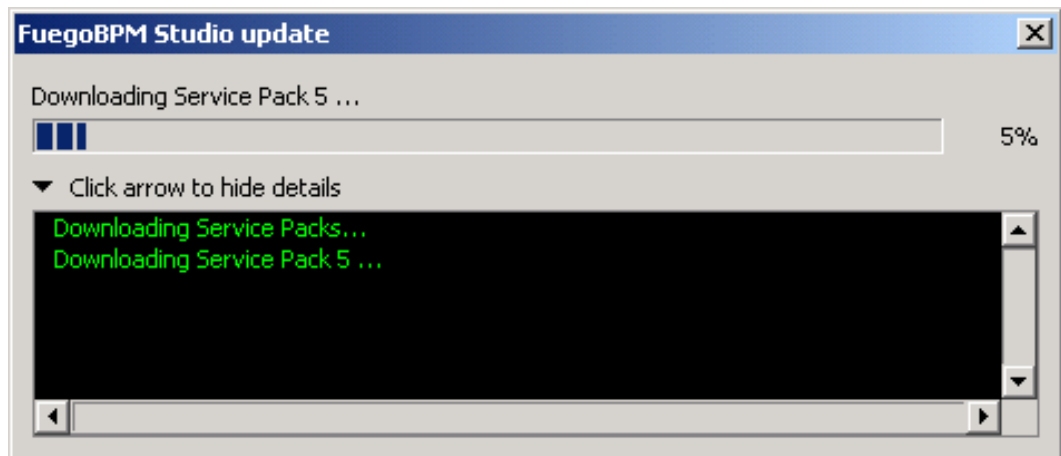


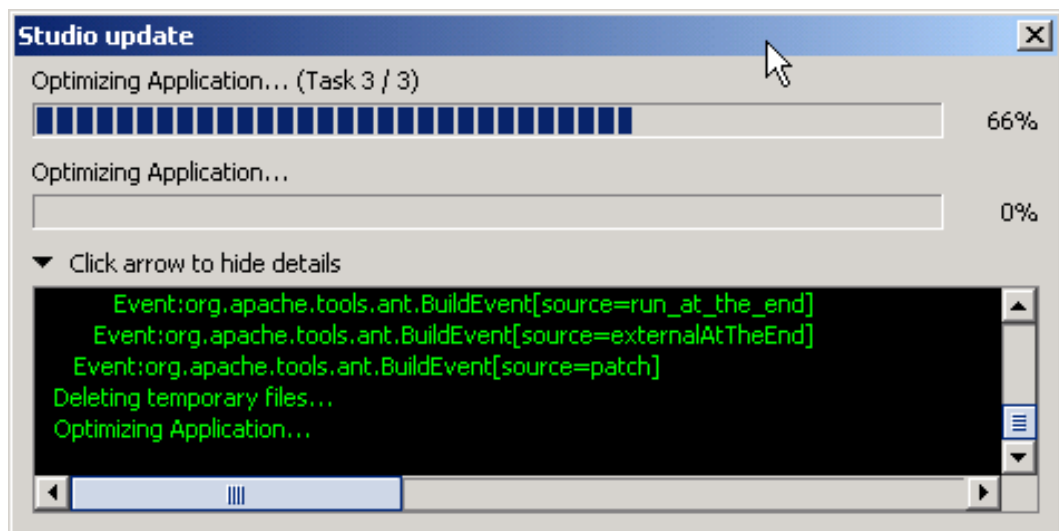
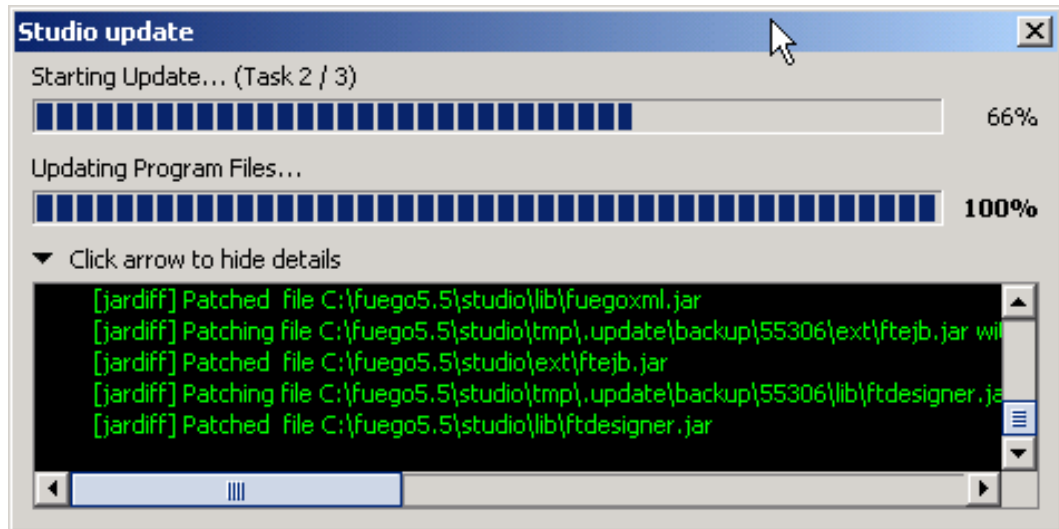
In this example the installation is a FuegoBPM Studio SP4. The *Check for Updates* detects SP5 and SP6 can be updated. And let's the user know that the SP7 would have to be done after restarting the FuegoBPM Studio.

3. Click **Yes** to begin the update or **No** to cancel the operation. Once the execution is confirmed a dialog showing the progress is displayed. Click the arrow to *show* or *hide* the update details.

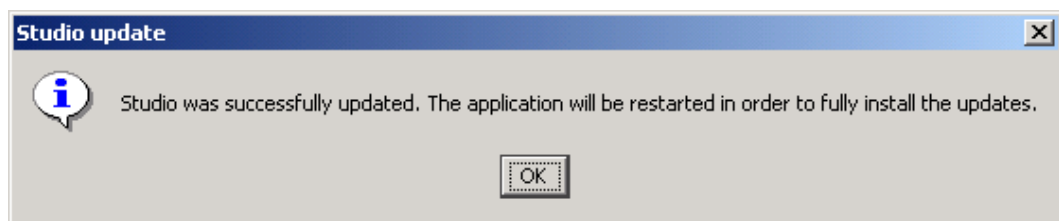


4. Some of the steps you will see while updating:






- Once the installation update ends, it restarts the FuegoBPM Studio.



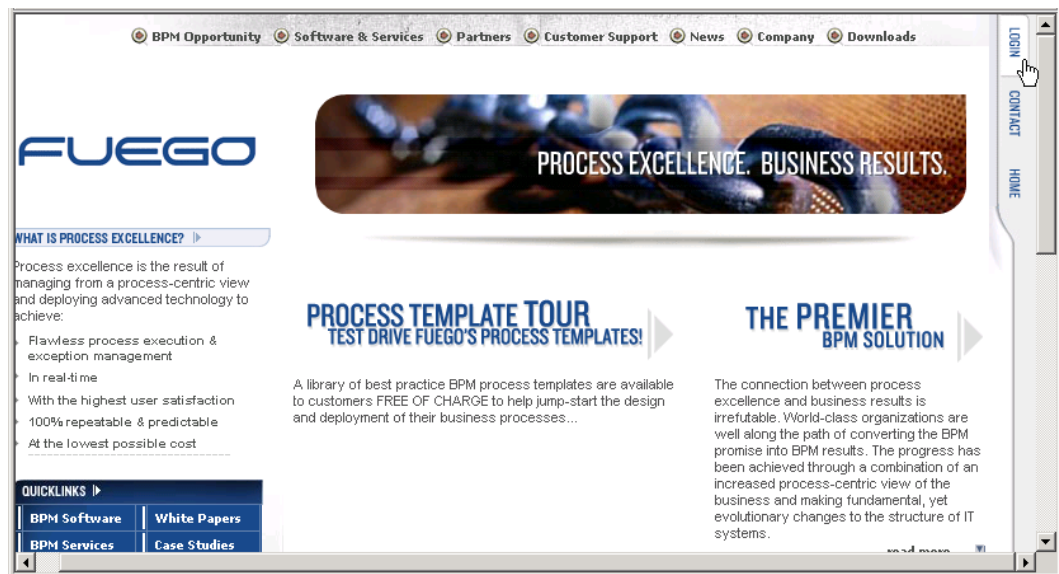
Note

 When you start FuegoBPM Studio, you can set the Studio preferences to automatically check for new updates.

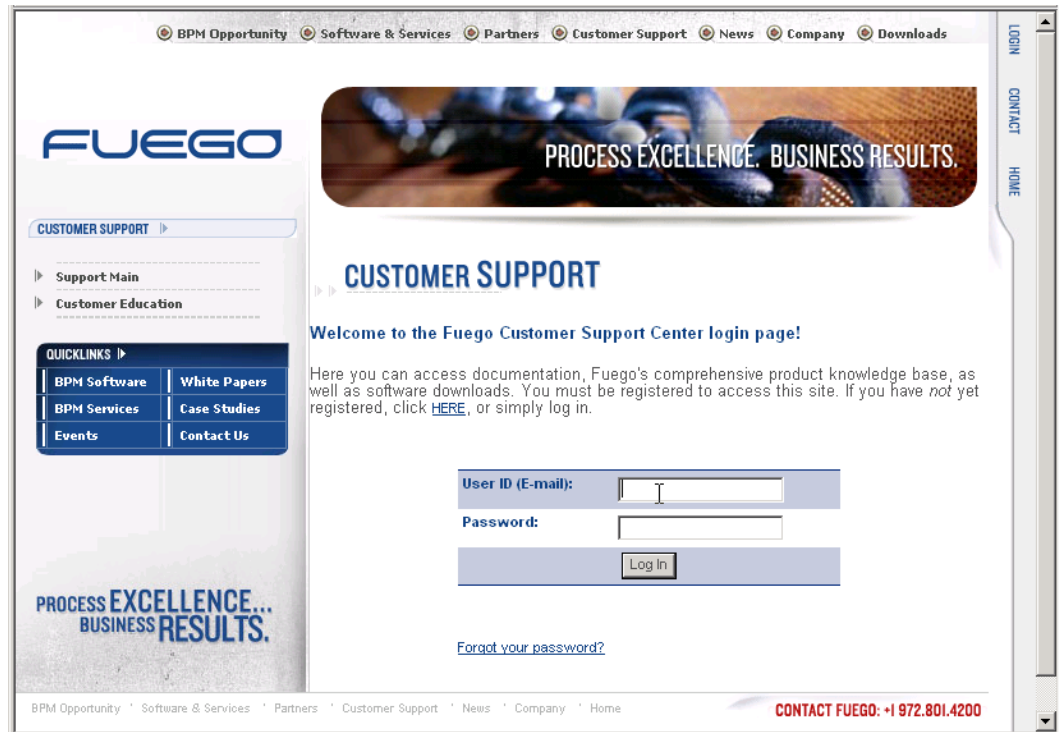
Applying a Service Pack Manually

To apply a Service Pack manually,

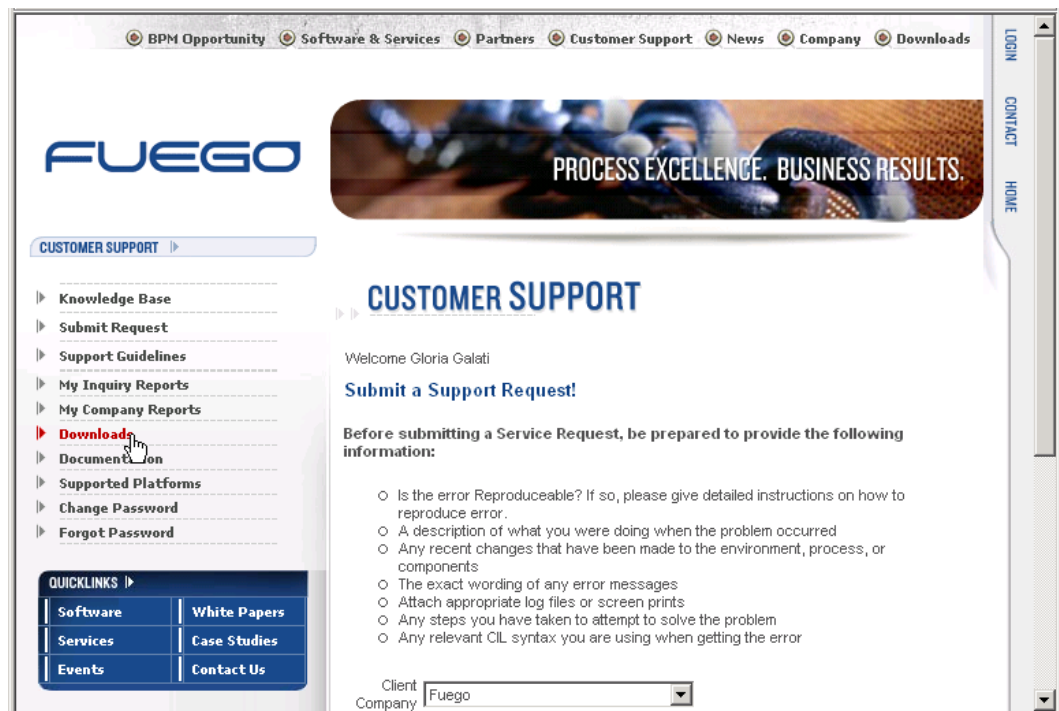
1. Go to the Fuego homesite and select the *Login* tab on the right top of your browser,



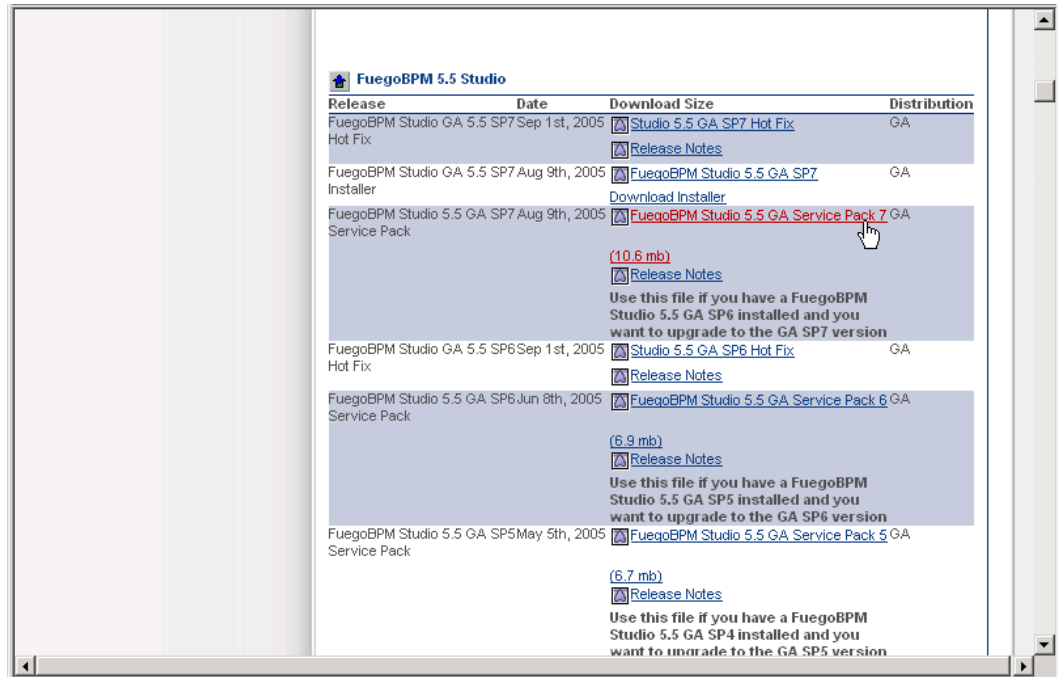
2. Type your email and password,



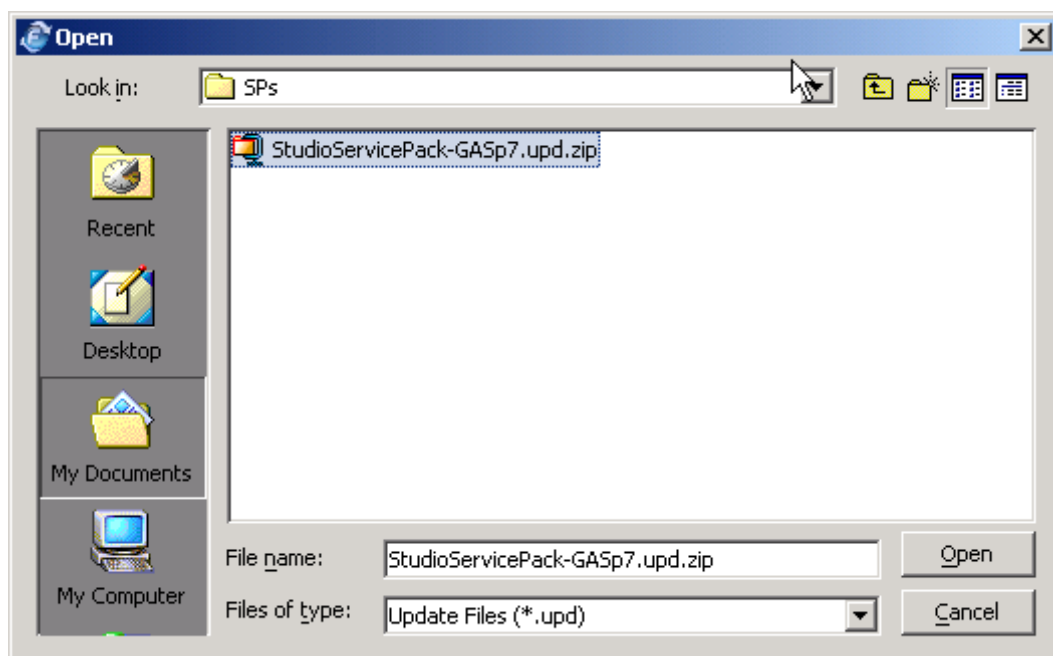
3. Click on the link *Downloads* on the left pane of the screen,



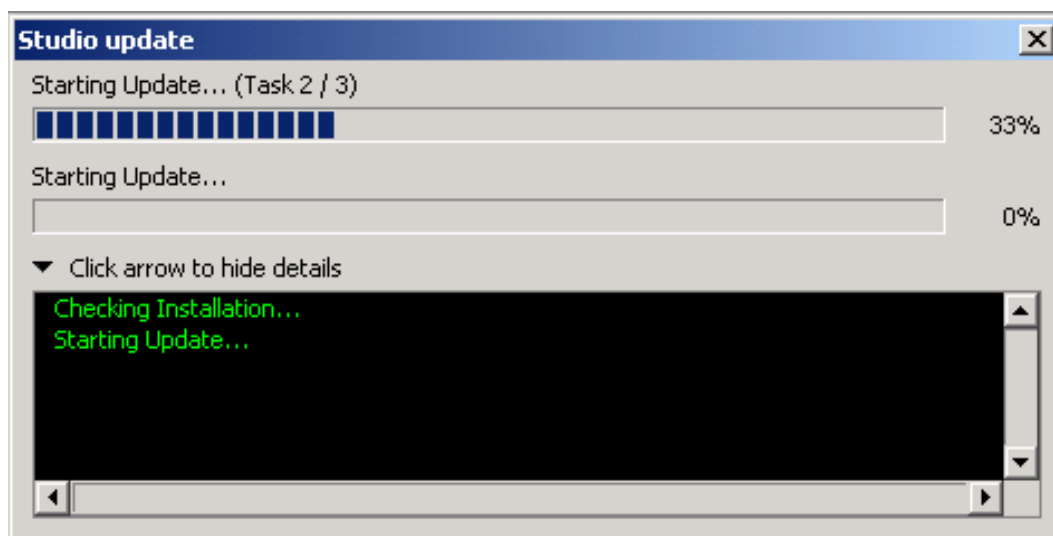
4. The list of versions is listed according to your customer profile. Select the Service Pack you are looking for and begin the download .



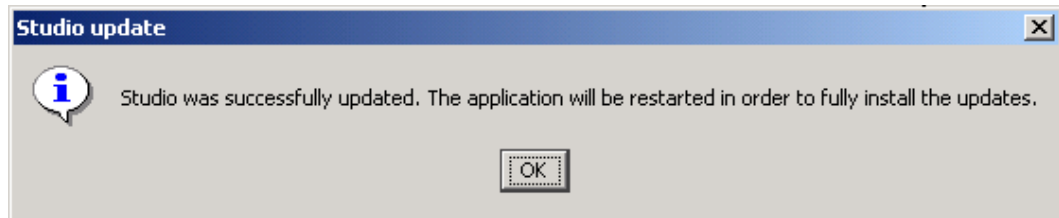
5. Once the download has finished, launch the FuegoBPM Studio and close any opened project.
6. Run the *Updates/Studio Local Update* option from the main *File* menu. Browse to the location where the Service Pack was downloaded, select the file and click **Open**.



7. The Service pack installation begins.



8. Once the installation update ends, it restarts the FuegoBPM Studio.

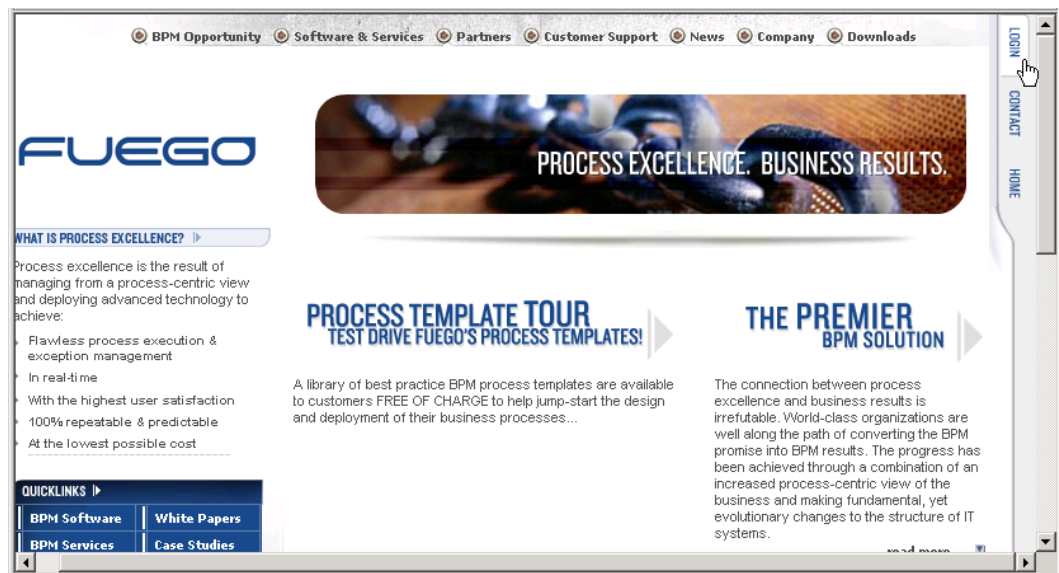


Rollback Service Pack

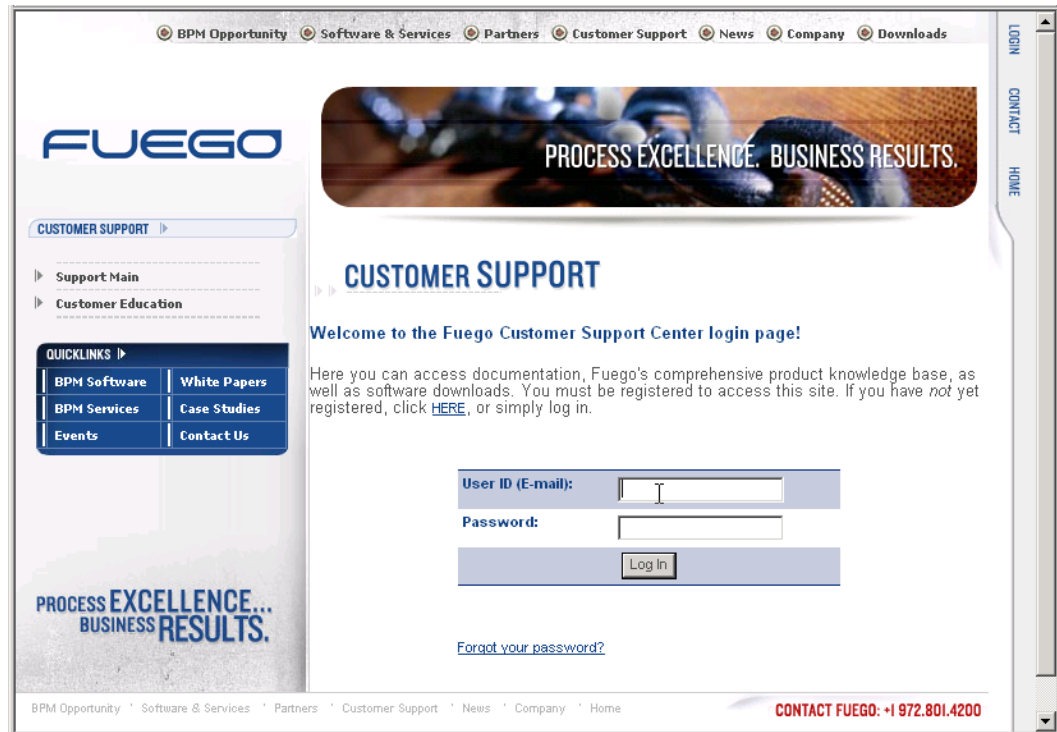
If a Service Pack has been applied and needs to be rolled-back, run the option *Updates/Rollback service pack* from the main menu **File** of the FuegoBPM Studio

Applying a Hotfix to the FuegoBPM Studio

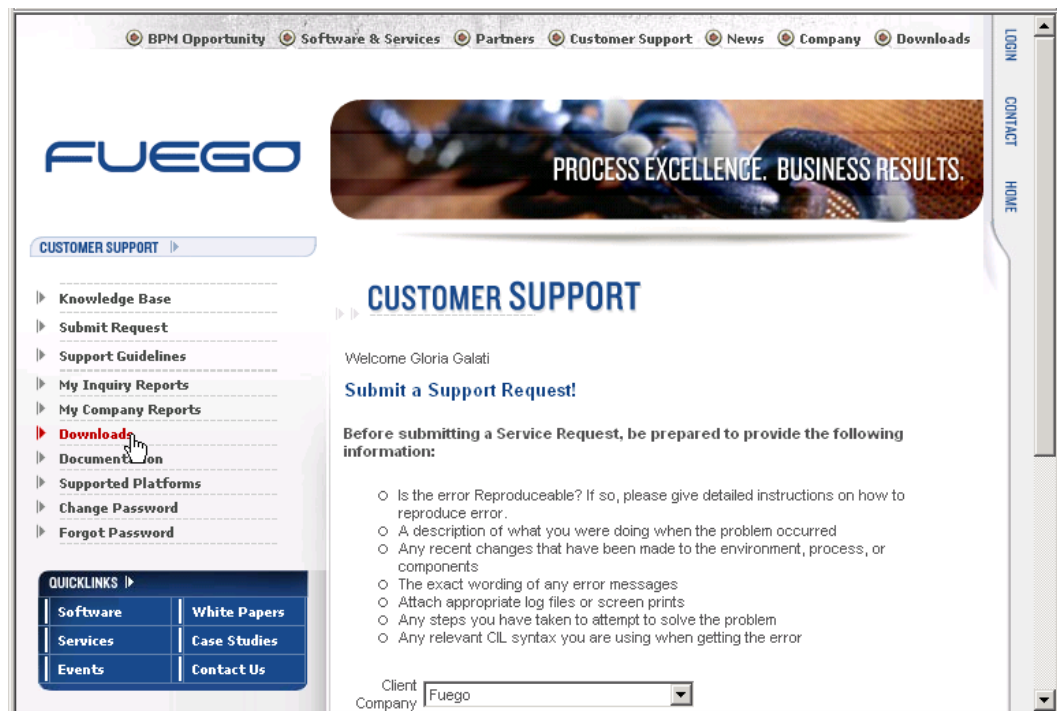
1. Go to the Fuego homepage and select the *Login* tab on the right top of your browser,



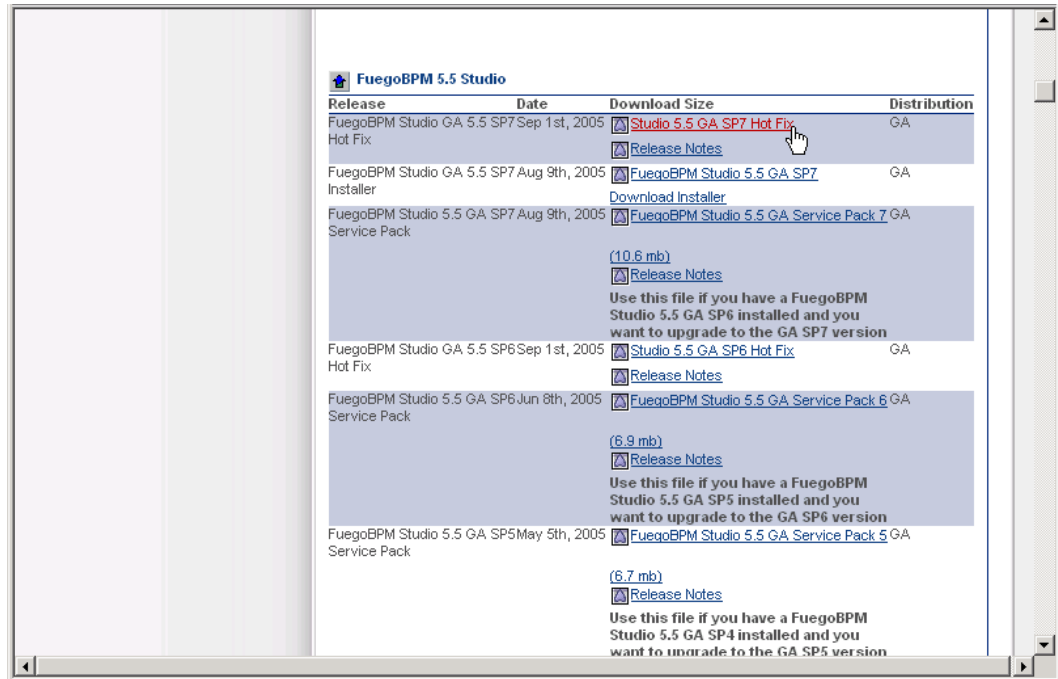
2. Type your email and password,



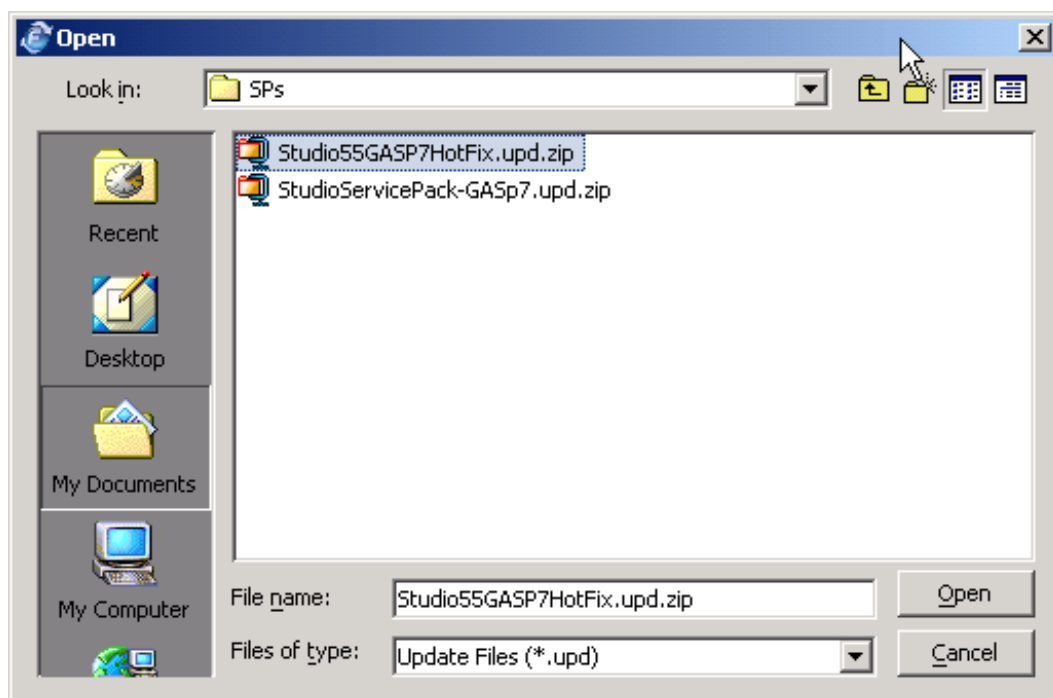
3. Click on the link *Downloads* on the left pane of the screen,



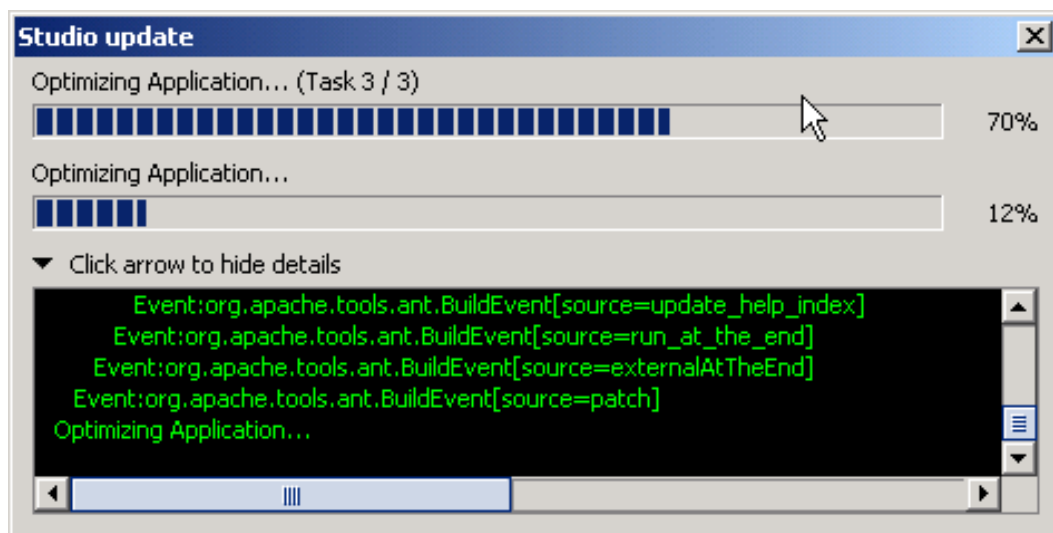
- The list of versions is listed according to your customer profile. Select the Hotfix you are looking for and begin the download .



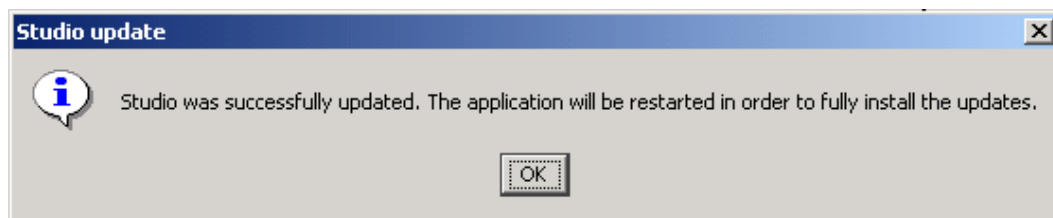
- Once the download has finished, launch the FuegoBPM Studio and close any opened project.
- Run the *Updates/Studio Local Update* option from the main *File* menu. Browse to the location where the Hotfix was downloaded, select the file and click **Open**.



7. The Hotfix installation begins.



8. Once the installation update ends, it restarts the FuegoBPM Studio.



Rollback Hotfix

If a Hotfix has been applied and needs to be rolled-back, run the option *Updates/Rollback hotfix* from the main menu **File** of the FuegoBPM Studio

Chapter 2. Defining an Orchestration Project

Orchestration Project

Introduction

A business project is the combination of a series of actions or operations pursuing a common business purpose. These activities, either human or automated, need to be executed in order to deliver a product or service. Business requirements may involve functional integration across the company or organization.

An Orchestration Project involves not only the representation of all the elements that are part of a business, the human resources, the organization, the processes and the systems execution but also the way in which all of them interact.

General Overview

FuegoBPM Projects are a way to organize, develop and manage different processes, their users, component or systems catalogs. One of the main goals of projects is to enable you to group processes that are related in some way and separate them from other groups.

Each project has its own component catalog so that you will be able to separate components used in some processes but not in others by grouping them in different projects. The project also contains all the abstract user roles used in it and its own Organization information required in order to deploy the project.

Project Structure

When you save a project, it is saved to a new directory named after the project. The project and its directory contain:

- Name
- Processes root directory: The base directory where the processes are saved.
- Catalog root directory: The base directory where the catalog is saved.
- Build directory: The directory where all classes, such as Fuego Objects and processes classes are generated.
- Library directory: The directory from which external library jars or files are taken to be used in the processes and components.

Creating a Project

You can create a new project by doing one of the following:

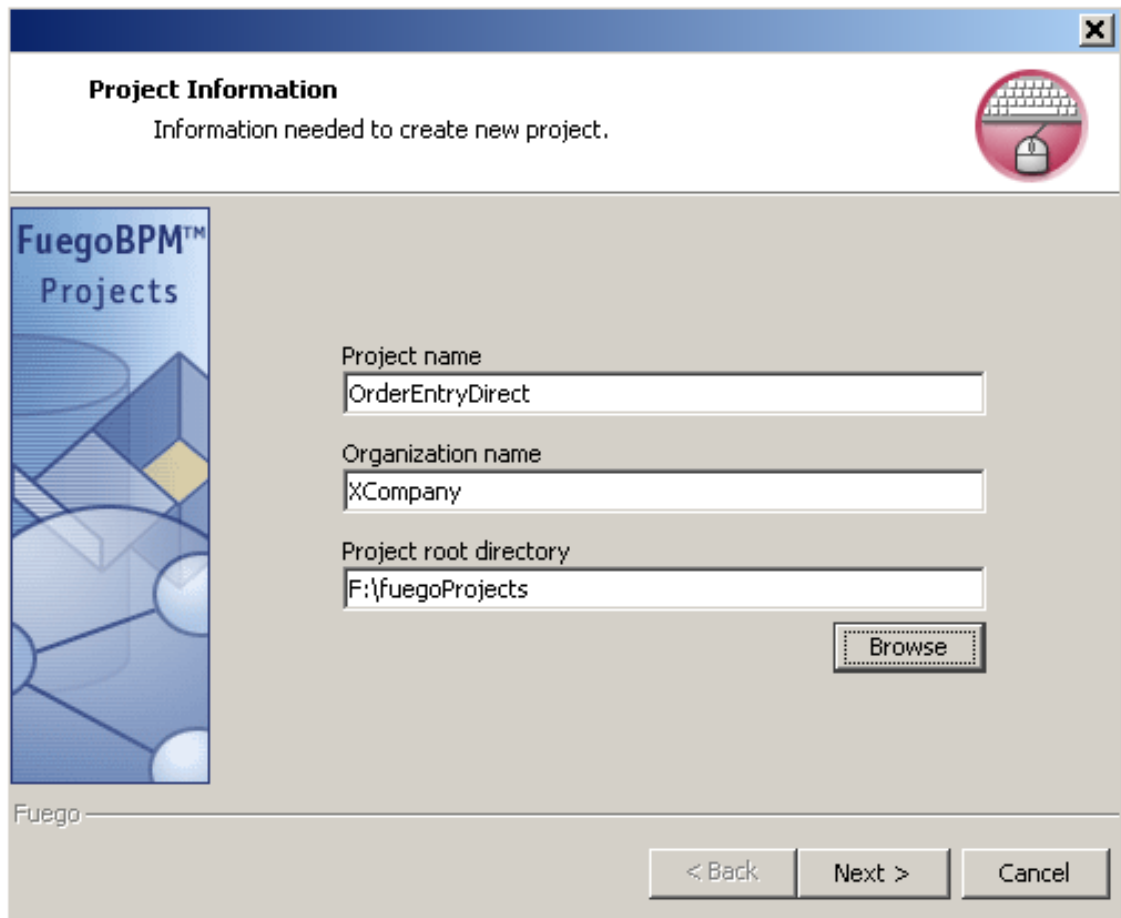
- From the **File** menu, select **New** and **Project** as shown below.
- Clicking the **New Project** button on the main toolbar, or
- Clicking the arrow to the right of the **New Project** button and selecting the New Project option.

No matter which method you choose to create a new project, a wizard is shown in which you must type all the necessary properties to create the project.

The information requested to create a new project is:

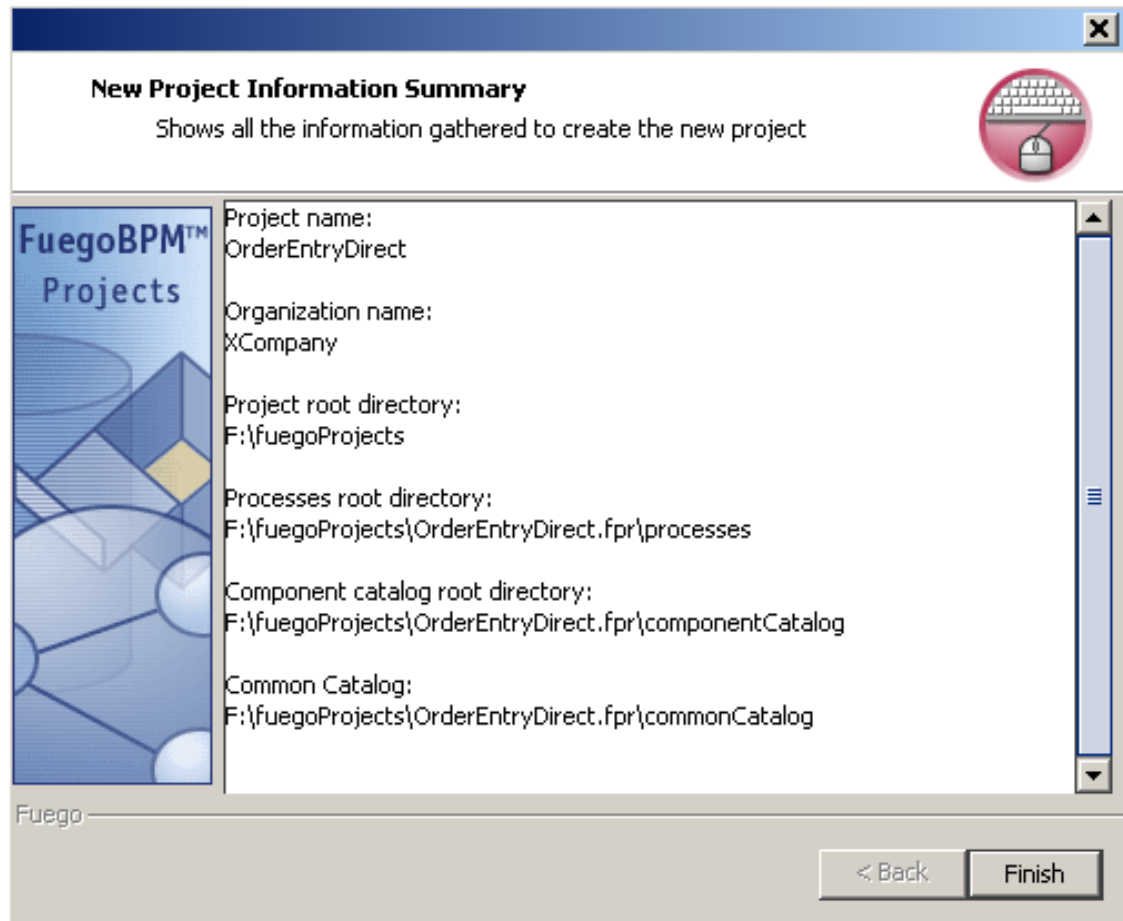
1. Project Name,
2. Organization Name,
3. Project Root Directory. By default, the directory where the last

project was created is displayed. You can select an alternate location by clicking the **Browse** button.

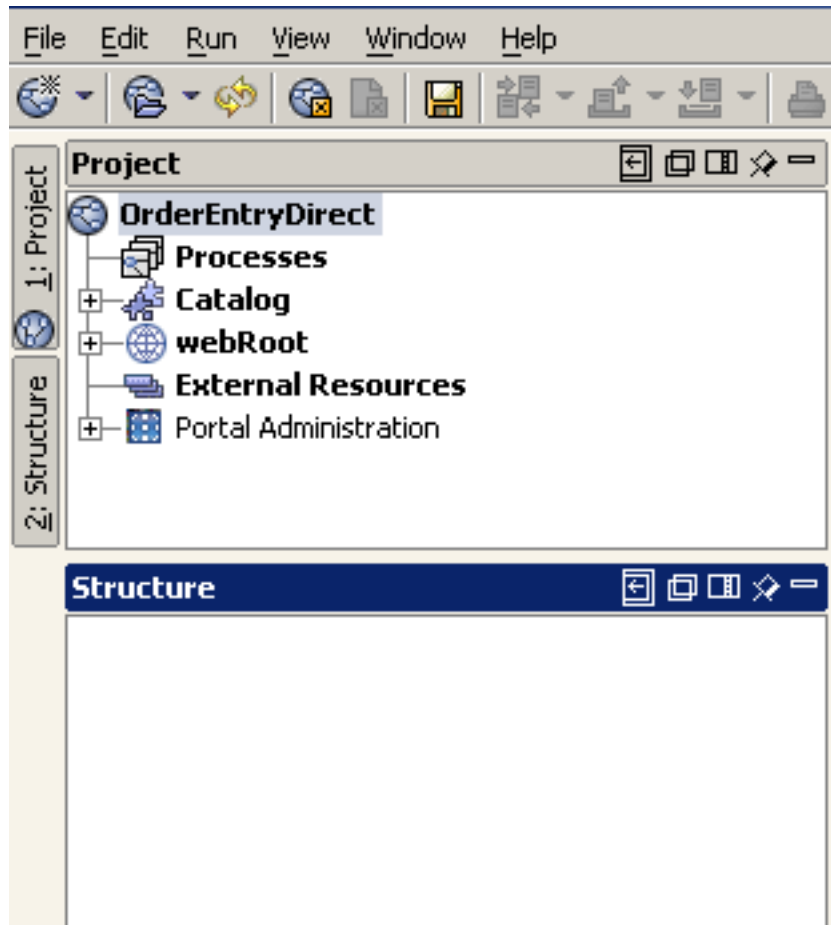


The screenshot shows a Windows-style dialog box titled "Project Information" with a subtitle "Information needed to create new project." and a keyboard icon. On the left is a vertical banner for "FuegoBPM™ Projects" with a blue geometric design. The main area contains three text input fields: "Project name" with the value "OrderEntryDirect", "Organization name" with the value "XCompany", and "Project root directory" with the value "F:\fuegoProjects". A "Browse" button is positioned to the right of the third field. At the bottom are three buttons: "< Back", "Next >", and "Cancel". The "Fuego" logo is visible in the bottom left corner of the dialog area.

The wizard shows an information summary that is used to create the project.



Once created, a new node appears in the Designer's navigator (click on the **Project**Tab). All the processes and subdirectories are shown under this node.



If another project was open when the new project was created, it is saved and closed.

By right-clicking on the project, a popup will show several options:

1. Add a new directory, creates a physical directory under the project's processes directory.
2. Create a new process, creates and adds a new process to the project. See Create Process.
3. Import a process or directories from other locations.
4. Perform VCS operations, VCS - Version Control System.
5. Close the Project. Saves any changes asking for a confirmation

and then closes the opened project.

Note



Options 1, 2, 3 and 4 are also available by right-clicking on a directory node.

When a new project is created, the Server database is created as well.

Opening a Project

To open a project,

- From the **File** menu, select **Open** and then **Open Project** or
- Press the shortcut keys Ctrl + Shift + J, or
- Click the **Open Project** button, on the main toolbar, or
- Select the **Open Project** option from the popup menu displayed when you click the arrow next to the **Open Project** button.

Closing a Project

To close a project,

- Use the shortcut Ctrl + Shift + W, or
- Click the **Close Project** button on the main toolbar, or
- Select **Close Project** from the File menu.

Reopening a Project

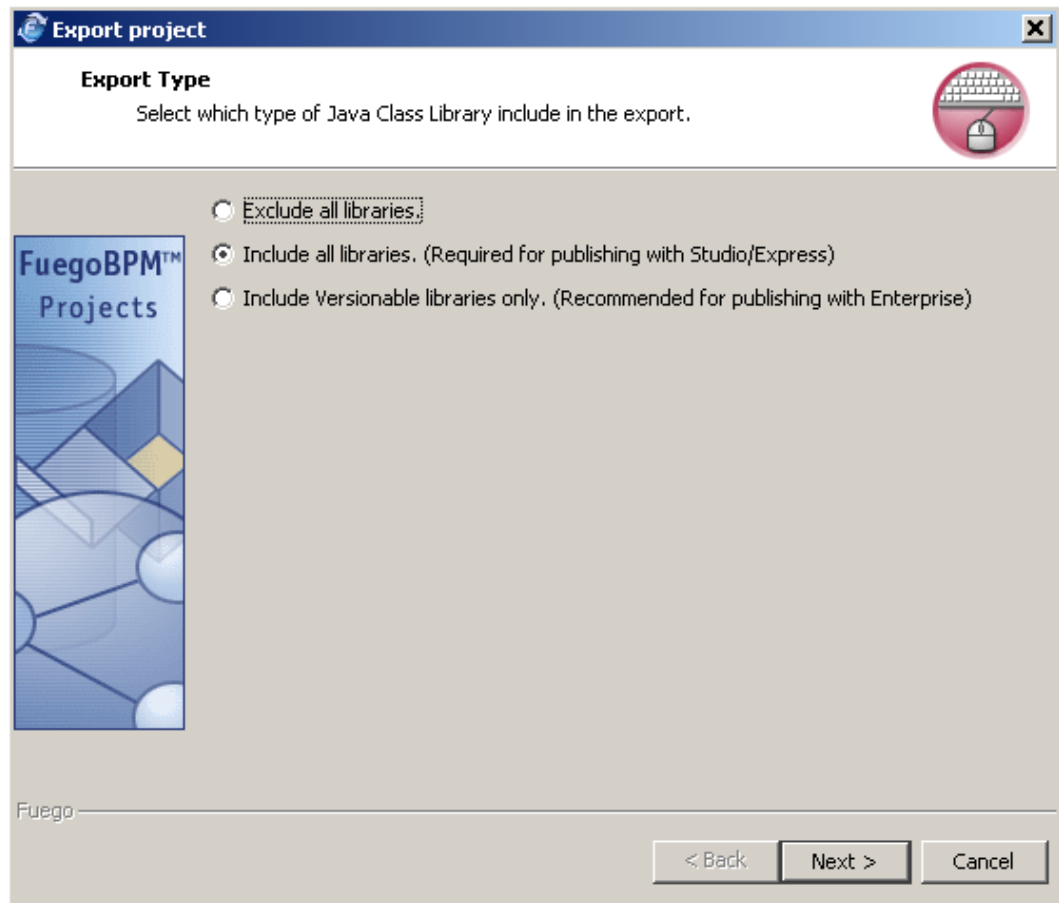
To reopen a project, select the **Open recent** option and then select a project from the list. The list can be cleared by selecting the **Clear List** option.

Export a Project

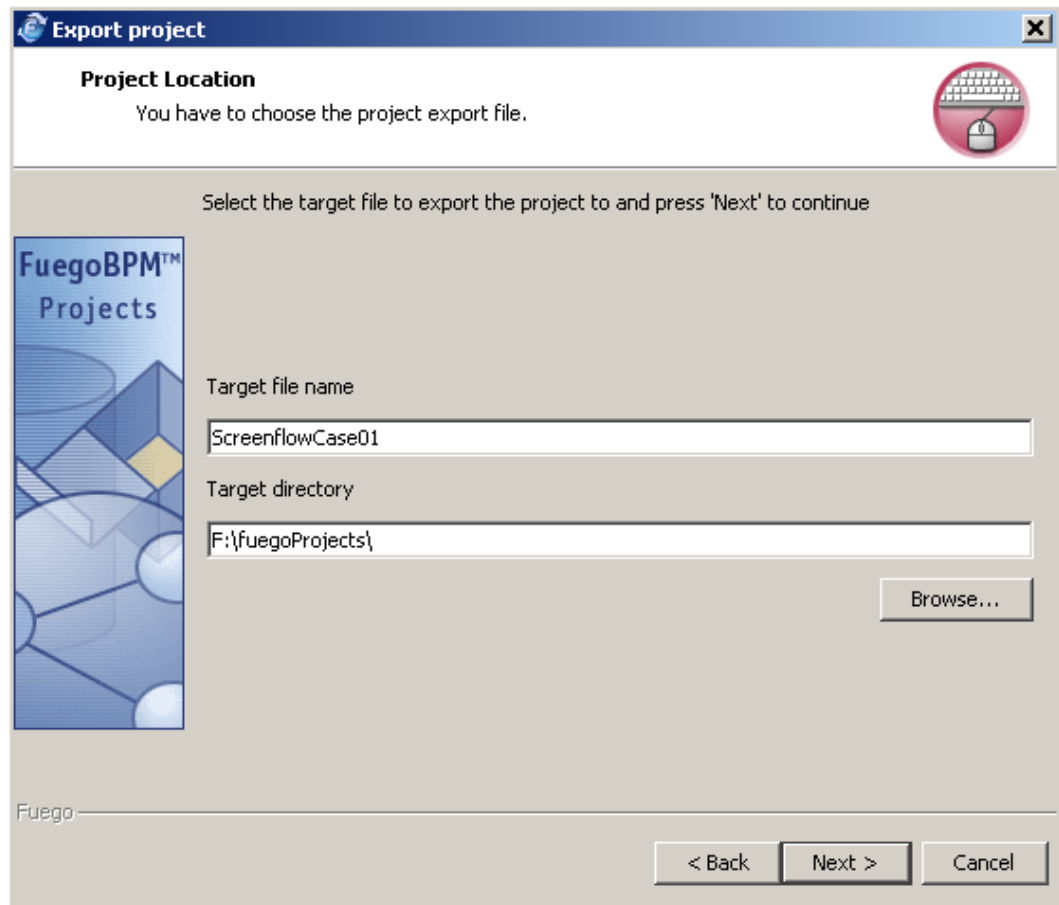
You can export a project to a file to share it or to use it in another environment.

To export a project, select:

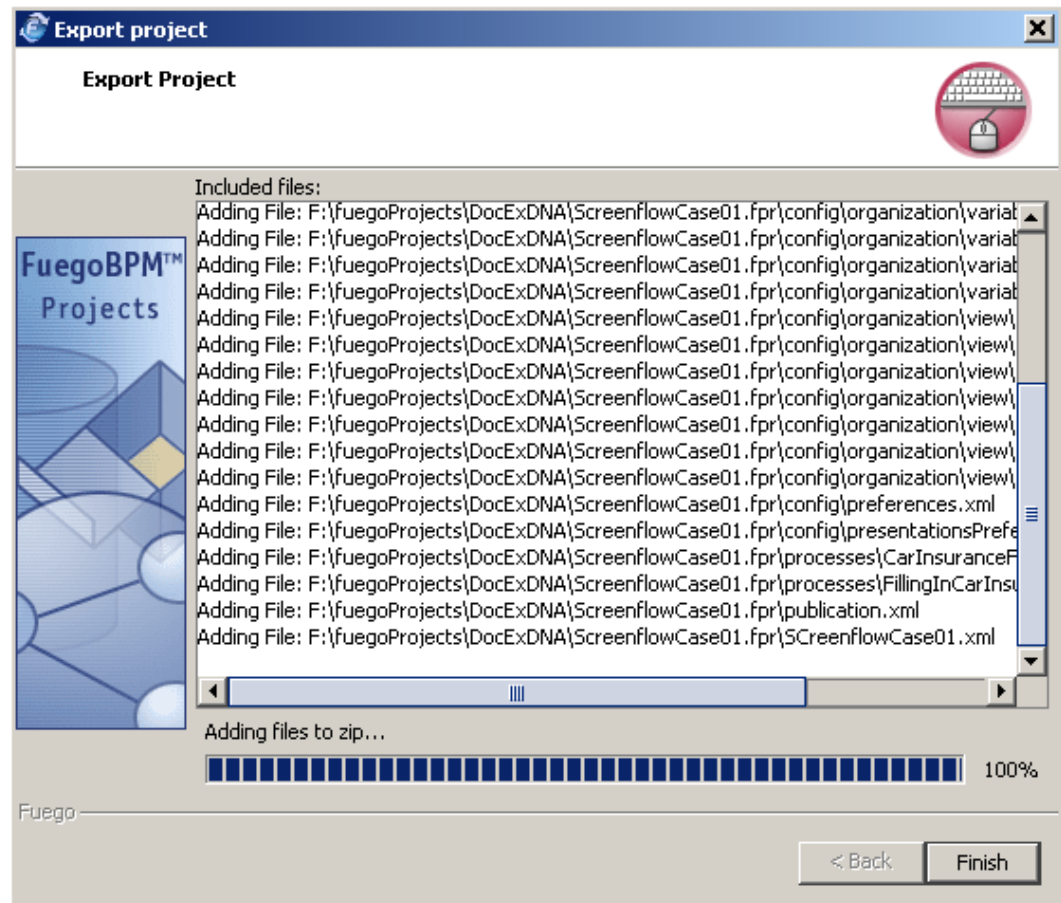
- The **Export Project** option from the **File** menu, or
 - Right-click on the project Name on the root of the navigation tree in the Project tab and select the **Export Project** option. The *Export Project* wizard is opened.
1. In the first wizard step, indicate if the project libraries are being included in the exported project. Select one of the options and click **Next**.



2. Browse to the directory where you will export the project and type a name for the file. By default, the project file is named as the exported project. Click **Next**.



3. The expanded files are displayed. Click **Finish** to end the export process.



Import a Project

To import a project into your FuegoBPM Studio, go to the **File** menu in the main menu bar of FuegoBPM Studio.

You can import a Project from an *file* or from the *Repository*.

From a Archive

To import a project from a file,

1. Select the **Import** and **Project** and then **From Archive** from the **File** menu. The import wizard opens.

2. Click the **Browse** button and select the location of the *Exported Project File* you want to import.
3. Type the *Project Root Directory*. By default, the last directory used is displayed. Change it by clicking the *Browse* button to browse to the location. Click **Next** to continue.
4. The name of the Project contained in the file you are about to import is displayed in the *Project Name* field. You can change the project name you are importing. Click **Next** to continue.
5. The project is imported. Click **Finish** to end the wizard.

From the repository

See Version Control System - Import Project from Repository.

Creating a New Folder

It is possible to create any level of nested folders under the Project, thus providing an easy way to organize the project by any criteria. For example, the Project may be the Order Fulfillment for a company and you may organize it under different folders for the different modules that compose the whole project, such as Order Entry, Billing, Delivery, Customer Service, etc.

A folder can be created by right-clicking on the Project category in the Navigator panel.

If you right-click on a folder, a popup menu with all the possible actions is displayed.

- Create a New Process,
- Create a New BPEL Process,
- Create a New Procedure,

- Create a New Screenflow, from the File menu
- Import Designs,
- New Folder,
- Delete Folder,
- VCS.

Deleting a Folder

To delete a Folder, right-click on the folder in the Navigator panel and select the **Delete** option. A confirmation dialog is displayed and everything contained in the folder is deleted.

Creating a New Process

See Creating a Process.

Creating a New Procedure

See Procedures.

Importing Processes

See Importing a process.

Checking a Project

You can check the whole project selecting the Check all icon. Each process is individually checked. Also, all processes and the catalog are cross-checked to verify any inconsistency.

If one process changed and has an impact on another process, if they are no longer compatible, the check all function detects this and warns you about the problem.

See Process - Checking the process design for further information.

VCS - Version Control System

See VCS – Version Control System.

Setting Project Preferences

Refer to the Project Preferences topic to learn more about how to configure the preferences for the Project about:

- *Languages,*
- *Project Version Control,*
- *Processes,* and
- *Activity.*

About the Component Catalog

Each project contains its own catalog of components, so that all the processes inside the project check and compile using this catalog. Each time a process is published, the catalog Fuego Object's jar file is also published in the Directory Service. For further information, see Implementing Business Objects using Fuego Objects.

The catalog taken for this operation is the catalog of the project to which the process belongs.

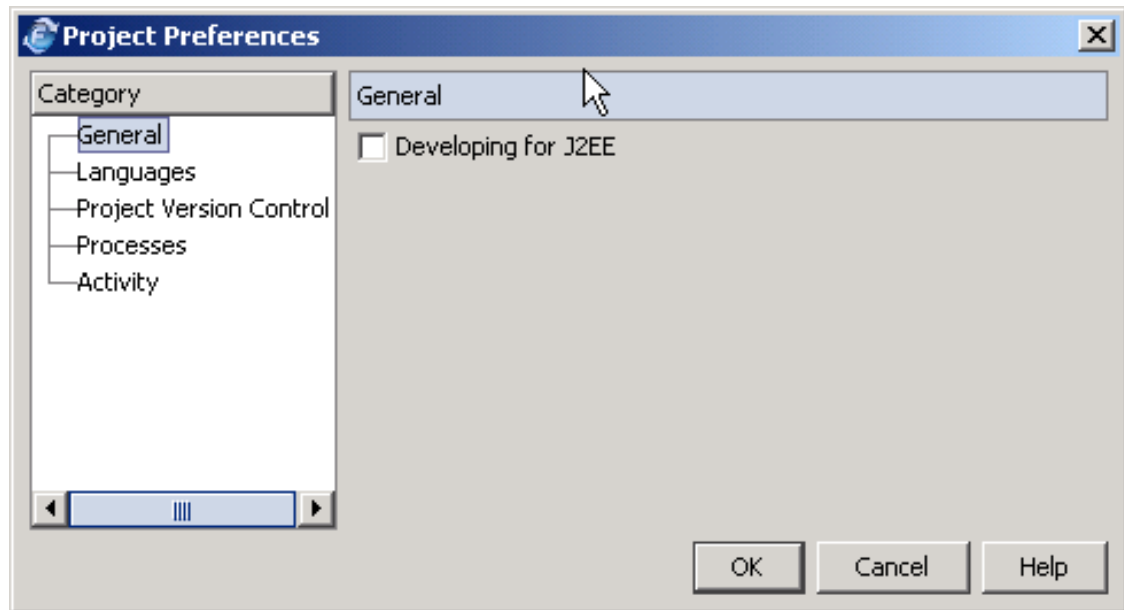
For further information on the Project Catalog, please refer to Implementing Business Objects with FuegoBPM.

Project Preferences

By selecting **Project Preferences** from the **File** menu, you are able to configure the preferences for the Project. The preferences that can be configured are as follows:

- *General,*
- *Languages,*
- *Project Version Control,*
- *Processes, and*
- *Activity.*

General

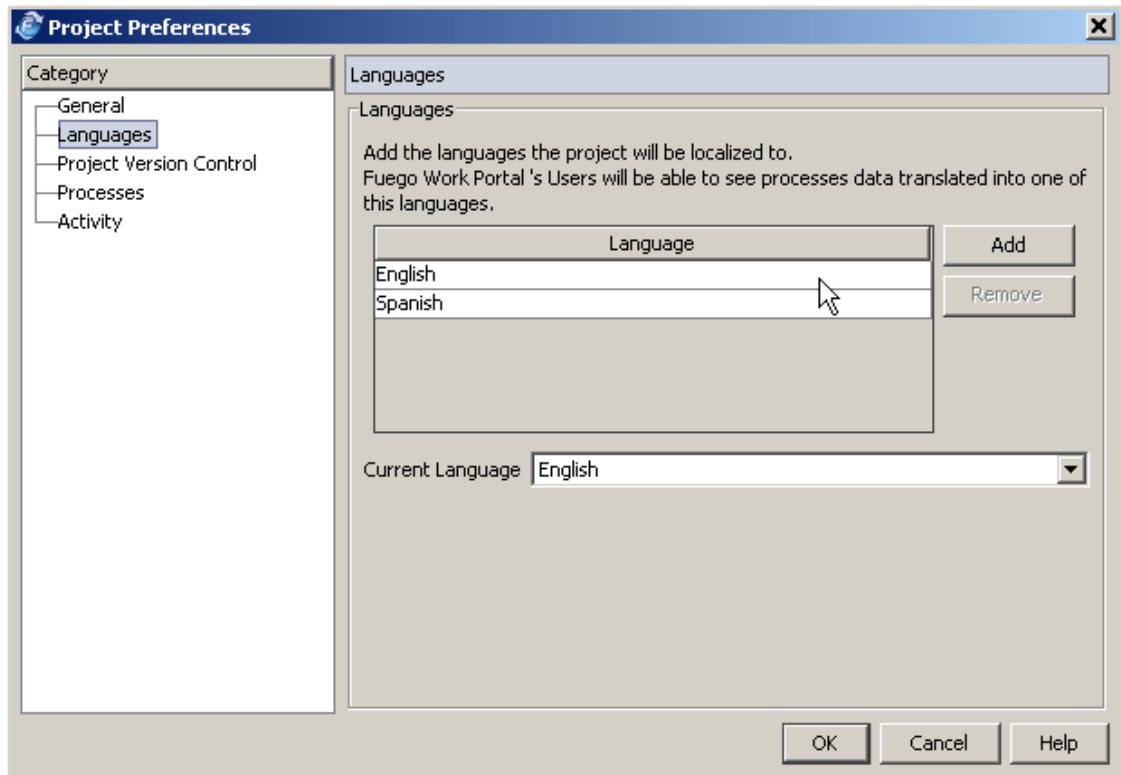


If you select this property, when you check the project, some implementation rules apply. For example:

- No *input* statements are accepted. You must use Screenflows.
- If you define a **Global Automatic activity** as **Executes when an event occurs** you have to use the Fuego components specially designed for JMS.

Languages

You define the set of languages in which you will localize the Project.



When a project is created, the language set by default in the Project preferences is the one set in the FuegoBPM Studio preferences. For example, if you create a new project and Studio preferences are set to *English*, then *English* is the only language in the project preferences and it is set by default.

The *Localize* button in the FuegoBPM Studio dialog boxes will remain disabled until you have another language in the set of possible languages of the Project Preferences. Once you define another language, the *Localize* button becomes available.

To change the language in which you are seeing the design of your project, change the default language in the Project Preferences to the one you desire.

The Documentation flap, set by default at the bottom of the

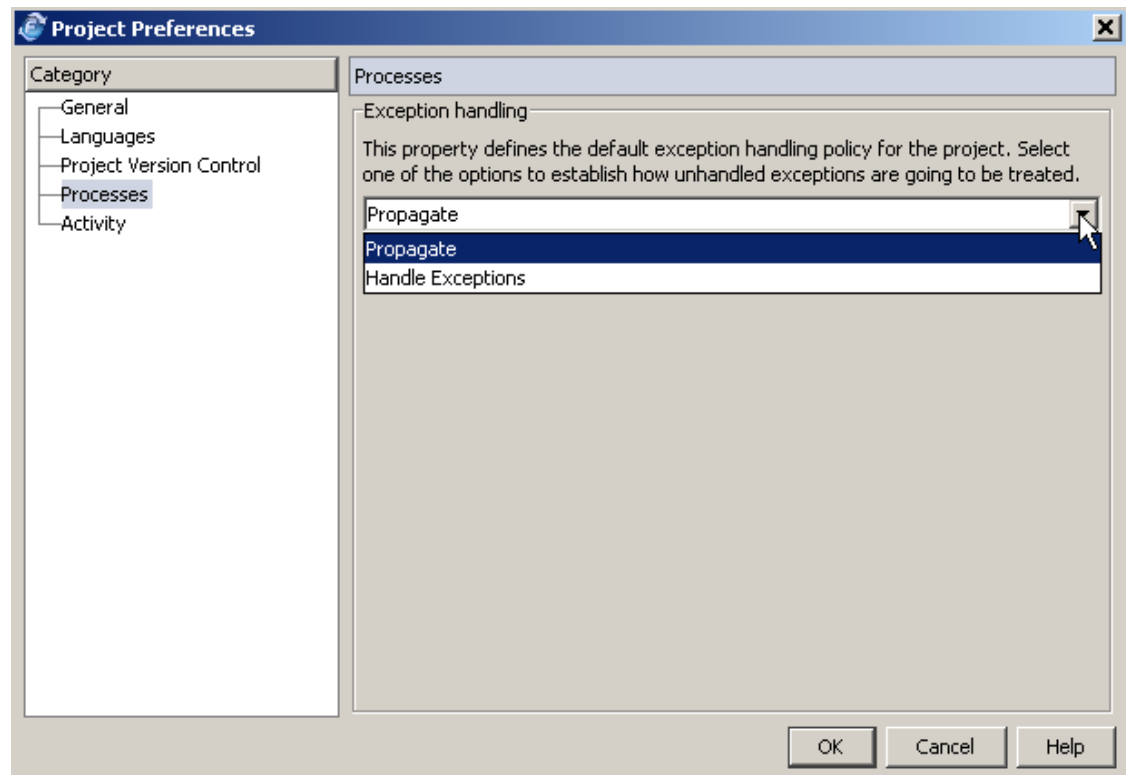
FuegoBPM Studio workspace, does not have a *Localize* button. The documentation you write in corresponds to the *default language* set for the project. You have to change the default language to add the documentation in each possible language in which you are localizing your project.

Project Version Control

Please, refer to the Version Control System topic for details.

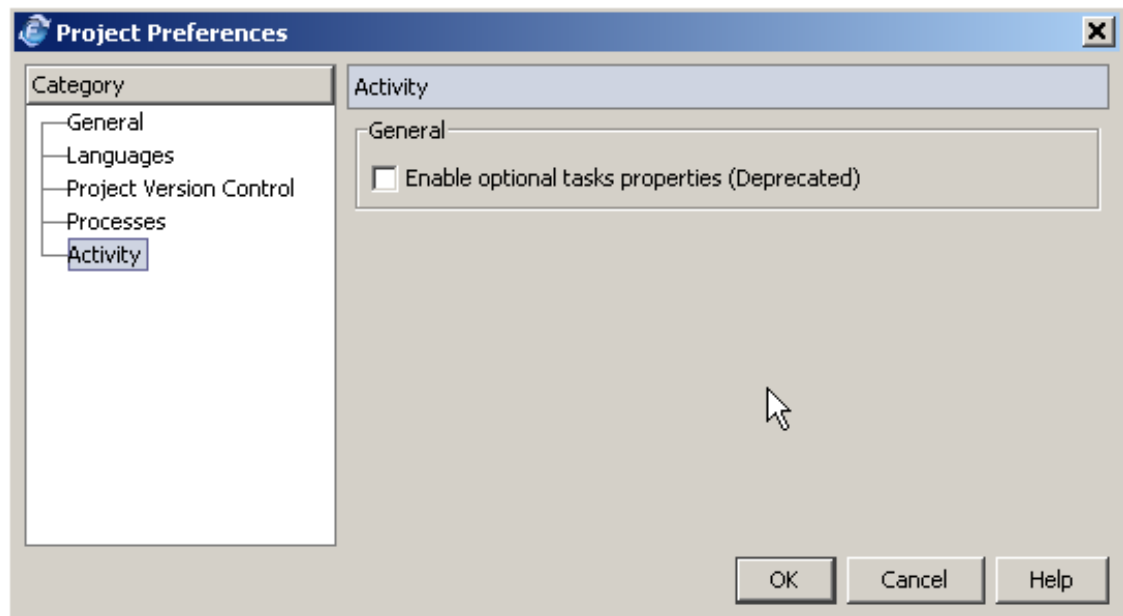
Processes

If you want all your processes within the project to manage the **Process Exception Flow** on a mandatory basis, you need to define the following preference:



See Process Exception Flow

Activity




By default, optional tasks are *repeatable* and *read only*. By enabling this preference, you are able to edit optional tasks' properties. Therefore, an optional task can become *mandatory* and/or *repeatable*. This preference might be *Deprecated* in the future and the optional tasks properties will no longer be editable.

Project Synchronization

Synchronizing the Project

Even though FuegoBPM Studio/Designer tightly integrates with the Concurrent Version System (CVS) that helps you managing revisions of your projects, you might have chosen to use a different VCS provider having to administrate your project from outside FuegoBPM Studio or FuegoBPM Designer.

In such cases, the **synchronize project** button  helps to update the project you are currently working with, with all the changes externally introduced.

If none of the resources reloaded by the **synchronize** was locally modified using FuegoBPM Studio/Designer, then the changes will be reflected immediately. Should you have any of the changed resources opened for viewing, you might need to close and reopen the panel or dialog to see the updated changes.

If you click on the **Synchronize Project** action and some of the externally modified resources conflict with your changes made using FuegoBPM Studio/Designer, then you will be asked to decide how to solve the problem. You can then either keep your last version in the Studio/Designer, or load the file system changes discarding all your changes and reload the panels with the file system version of the resource.

Note




Note The same happens if studio attempts to autosave a resource that was externally modified

Read Only Management

The use of a VCS provider other than the one that is the integrated with FuegoBPM Studio/Designer, might also cause that FuegoBPM Studio and FuegoBPM Designer show some project source elements as read-only. This scenario is present since some VCSs allow to get files with read only access.


The project source elements that might appear as read only include :

- Processes
- Components
- Views and Presentations
- All organization resources
- Simulation models

All the resources marked as read-only  in the main project tree are shown with the read only icon next to the resource identification indicating that you cannot change that resource.

You can keep on viewing all the settings and preferences for those resources opening the panel or dialogs that show the information but you are not able to do any modification.

Note

 **Note** Changing the read-only status of a resource is not handled by the **Synchronization** action. You need to reopen the edition panel or dialogs to refresh the new read only state

Localizing and using multiple languages


FuegoBPM Studio/Designer allows you to adapt all definitions to different languages. To do so, all processes can be internationalized. You can write information in different languages. For example, for a *process*, you can i18n (internationalize) the label, description, documentation, use cases; the same may be done with *Activities*, etc.

For example, an activity in your process called *Review Order* in English can be defined as *Revisar Pedido* in Spanish.

The available languages depend on the enabled languages in your project. Please refer to Internationalization for further information about the available languages.

The Process designer will see all the labels in the language set by default in the Project Preferences. And each process participant will see the titles in the language he or she selected through the Work Portal Options.

Note

 Any object that can be localized will contain a **Localize** button in its **Properties** dialog box. If the **Localize** button is disabled, it is because the

project does not have multiple languages available. Add languages in the **File** menu, **Project Preferences** option, in the **Language** category. See Internationalization for detailed information.

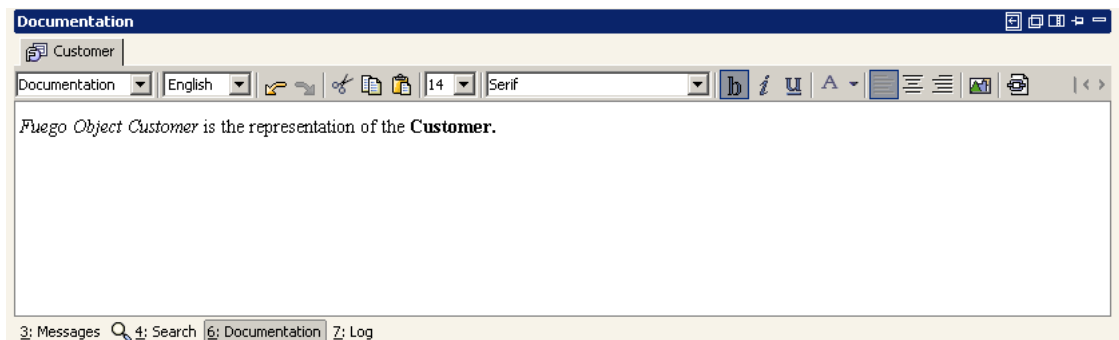
To localize an object in the project

1. Click **Localize**. The **Localize** dialog box appears.
2. All available languages are listed.
3. Enter the translation in the **Message** column.
4. Click **Ok** to close the dialog box.

Documenting a Project

Documentation Editor

When you select the **Documentation** flap, a simple editor is opened by default at the bottom of the FuegoBPM Studio workspace.



Documentation Type

The editor displays one or two tabs based on the context you are working with. For example, if you are within a **Process**, **Screenflow** or **Procedure** you will see two tabs: one of them to document the process/procedure/screenflow as a whole and the other to document the selected activity in the main panel.

If you are working with a component, such as a Fuego Object, you will only see one tab with the name of the Fuego Object unless you open a method within the Fuego Object. In this case, you will see two tabs: the main Fuego Object documentation tab and the Fuego Object's method documentation tab.

Each tab to be documented has two documentation options: *Documentation* or *Use Cases*.

Text Format and Style

Available Fonts:

- Fonts installed in the PC where you are working.

Available Font sizes:

- 8
- 10
- 12
- 14
- 18
- 24
- 36.

Available Font Formats:

- Black
- Italic

- Underlined.

Text Editing:

- Copy or Ctrl+C
- Cut or Ctrl+X
- Paste or Ctrl+V
- Select All or Ctrl+A
- Undo
- Redo
- Text can be left, center or right justified.
- Text can be color formatted using the color pallet.

Link

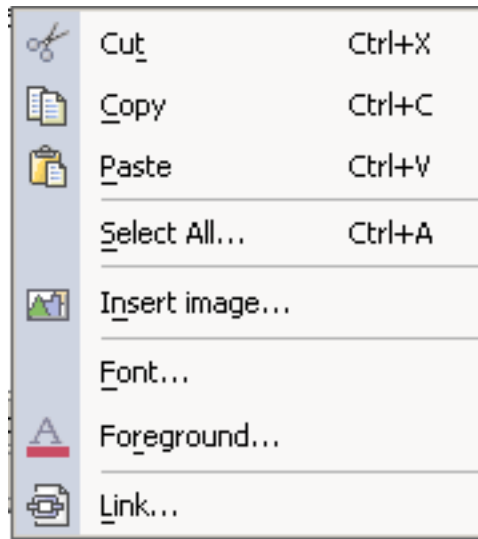
Links can be added to your documentation by selecting the Link icon.

Images

Images can be added to the documentation by selecting the Images icon or by selecting the *Insert image...* option from the shortcut menu that opens when right-clicking on the editor panel.

Edit Options

By right-clicking in the editor panel, a shortcut menu with edit options is displayed.



- Cut or Ctrl+X
- Copy or Ctrl+C
- Paste or Ctrl+V
- Select All or Ctrl+A
- Insert image
- Font
- Foreground
- Link.

In the **documentation** flap, when you are writing and then you press the **Enter** key, you are creating a new paragraph. When you press **Shift + Enter** you are just inserting a new line in the same paragraph.

Paragraphs can be aligned left, right or center independently. When you insert a new line, the next line will be in the same paragraph as the previous line.

Documenting a Process / Procedure or Screenflow

FuegoBPM Studio allows you to document your processes, procedures and screenflows in two ways:

- You can create documentation for the entire process/procedure/screenflow by selecting the first tab within the Documentation Flap, and
- Documentation can also be created for each individual activity by selecting the second tab, if enabled (for the tab to be enabled, you should select the activity in the main panel).

The documentation is available to end users in Work Portal and is also included when you generate a process report. See the Process documentation for information on how to **Generate a Process Report**.

General Documentation

- **Documentation:** you can briefly document the process/procedure/screenflow as a whole. This documentation is included if you generate the project report. For example, you can document the purpose of a process.
- **Use Case Documentation:** you can describe the Use Cases of the process. This documentation is included in a separate section within the project report. For example, you can describe the scenario in which the process is used and the interaction the participant has with the process.

Documentation can also be written in different languages and the description will be displayed in the corresponding language within the Work Portal.

Documenting an Activity

- **Documentation:** you can briefly document the activity. This documentation will be included if you generate the project report. For example, you have an Interactive activity with a main task and multiple optional tasks and you want to explain each task briefly.
- **Use Case Documentation:** you can describe the Use Cases of the activity. This documentation will be included in a separate section within the project report. For example, you can describe the daily work and match the activities and tasks used to perform it. Describe the scenario in which this activity is performed and the interaction between the participant and the process.

Documenting the Catalog

FuegoBPM Studio provides the ability to create, modify and delete documentation for each component. Documentation consists of component comments (defined by designer at component creation), methods, attributes, etc. (as with JavaDoc).

It provides the process designer with the ability to automatically generate component documentation for components in a local catalog. This documentation is in HTML format.

Documentation that was generated for a single component can be added. It can be built for

- a selected group of components,
- all components in an entire Catalog.

Each component has default documentation including descriptions, methods, arguments and attributes.

You can modify and improve this documentation as described below for each component, method or attribute.

Adding Documentation to a Component

To Document a Component:

1. Select the Documentation flap, set by default at the bottom of the FuegoBPM Studio workspace,
2. You can either write the:
 - **Documentation** of the component explaining, for example, what it represents, which its attributes and methods, attribute valid values, etc. are, or
 - **Use Case Documentation**: you can describe the Use Cases in which the component can be used. This documentation will be included in a separated section within the project report.

FuegoBPM Studio Component Documentation

If you open a Fuego Block in the main panel and you open the Documentation flap, the component documentation displays. Within the Use Cases tab, you will find some common uses of the component.

Generating the Project Documentation Report

See Project Report.

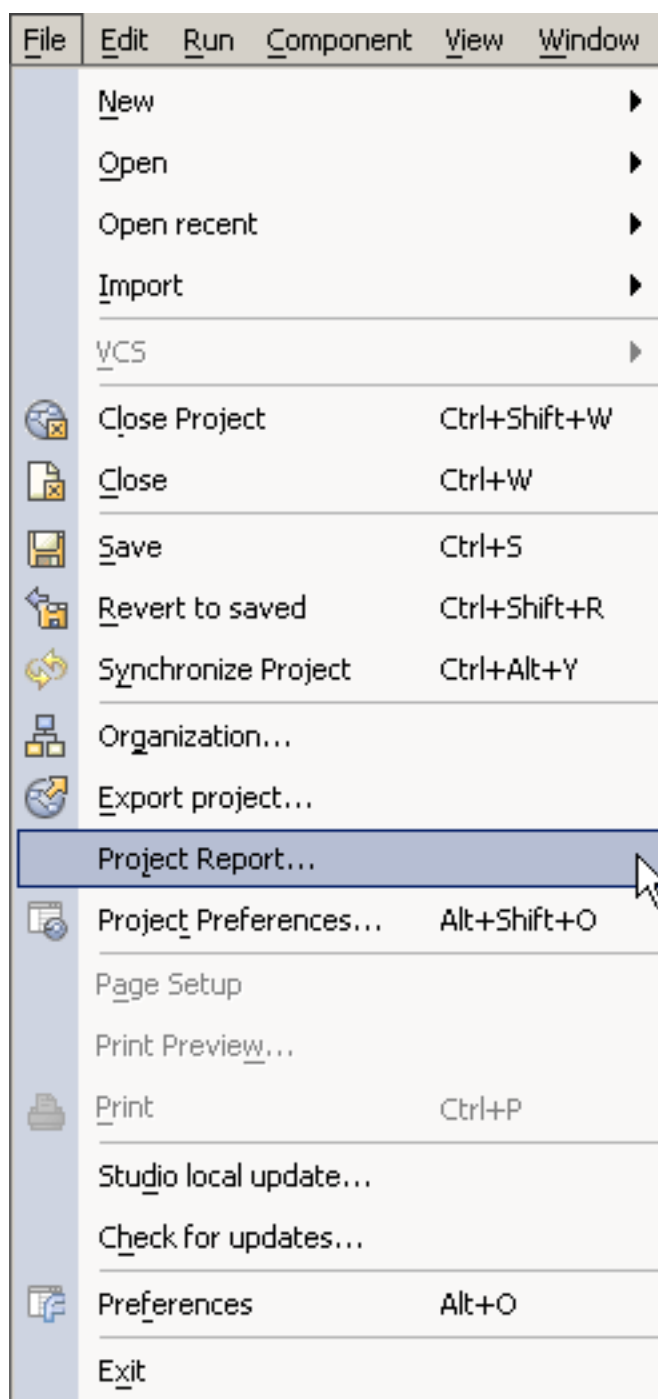
Generating the Project documentation

You can print the whole **Project documentation**, a **Process documentation**, the **Catalog** documentation or a **Component** documentation.

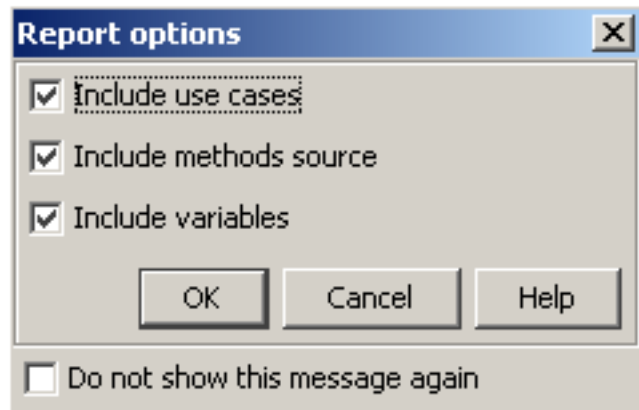
This report includes all the information written for the project while Documenting the Project.

Project Documentation

To generate the Project Report, select the **Project Report** report from the **File** menu. Or right-click on the Project name in the left tree and select **Project Report**.



Each kind of report (project, process, catalog or component) has different **options** as shown below:



The Project Documentation includes all other documentation in different sections.

- Project Documentation
- Process Documentation
- Catalog Documentation
- Component Documentation

Process Documentation

The **Process** Documentation contains all information and adds documentation of

- Roles
- Activities
- Variables
- Methods

Catalog Documentation

The **Project Catalog Documentation** shows the list of modules that compose the Project Catalog.

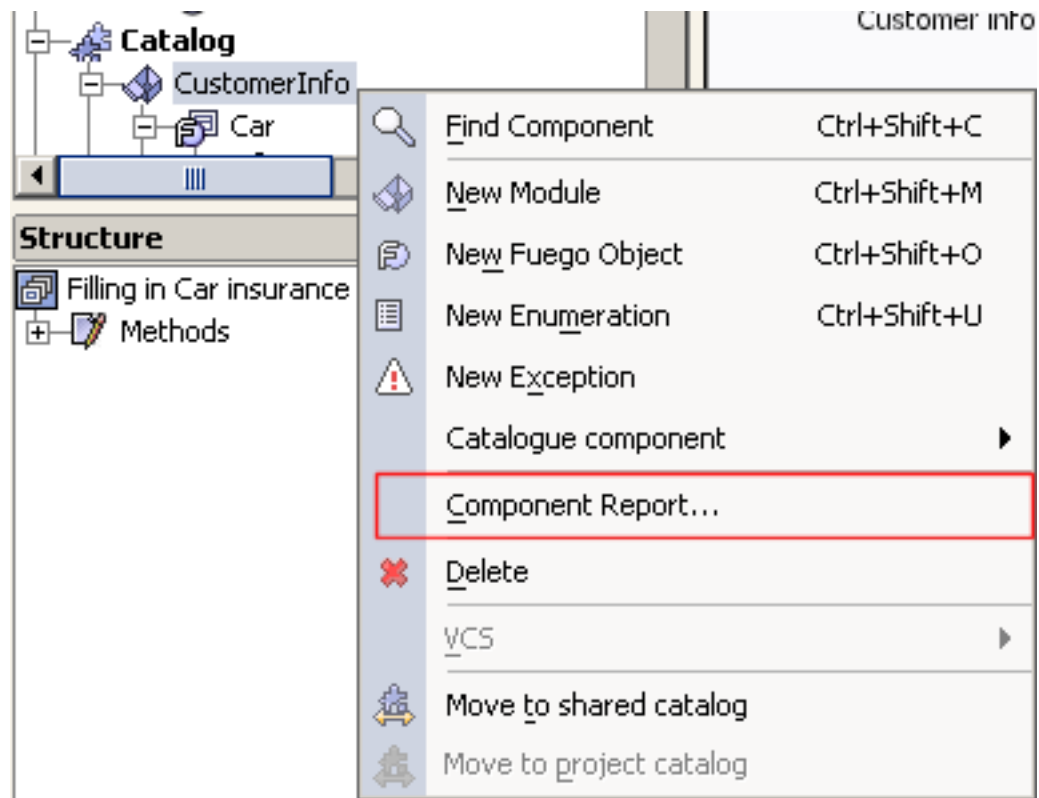
The **Module Documentation** shows the list of components or submodules contained in it and grouped by type, such as, *java*, "enumeration", "Fuego Object", etc.

The generated documentation is a javadoc type and is grouped by type of component, such as *Module*, *Enumeration*, *Fuego Objects*, *Javas*, etc.

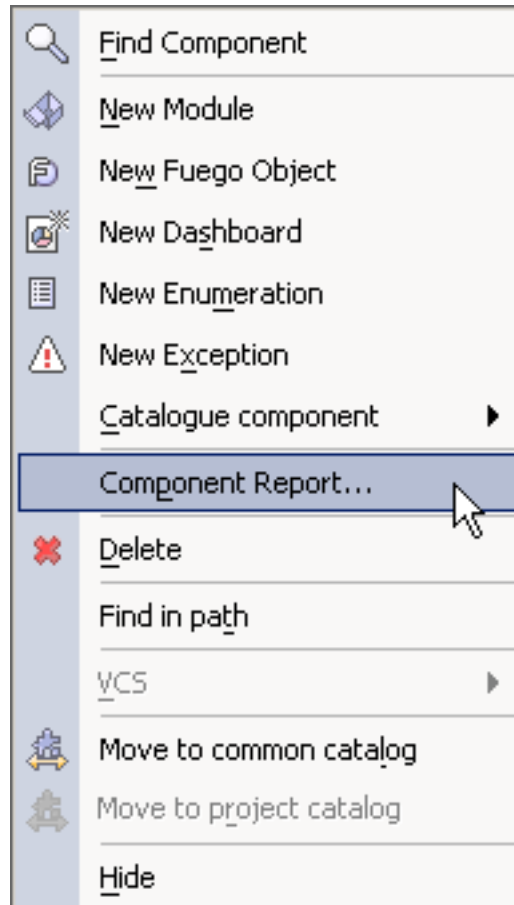
Components

To document a Group of components,

1. Right-click on the module or component in which you want to build the documentation.
2. Select the Component Report Option" from the shortcut menu,



3. The *Reporting options* dialog is opened.
4. A dialog to select the documentation output directory is opened.



5. Select or create the directory where you will save the documentation for the selected group of components.
6. Click **Generate Report**.
7. HTML format documentation is created under the selected directory.

Component Documentation includes:

- the documentation manually added by the process developer,
- a summary list of its attributes,

- a summary list of its methods, *type*, *name* and *arguments*.
- attributes detail, and
- methods detail.

Attribute Documentation, *type* and *name*.

Method Documentation, *type*, *name* and *arguments*.

Customizing the generated Documentation

When the documentation is generated, a cascading style sheet, *css file*, is generated as well.

You can define colors, fonts and other style attributes in this file to override the defaults values like:

- Page background color
- Table colors
- Font used in left-hand frame lists
- Navigation bar fonts and colors

Documenting a Project

See Documenting a Project.

Saving the Project

Saving Project's Entities

FuegoBPM Studio provides automatic saving for all the modified entities in the project: processes (.xpld), components, Fuego Objects, and so on. Before saving changes, FuegoBPM Studio keeps the first version of the entities in a backup directory.

FuegoBPM Studio **saves** all modifications automatically when:

- the developer closes a panel with a modified object (process, Fuego Object, and so on)
- if the autosave action is enabled
- before any Version Control System operation

All changes can be **reversed** unless you perform a **permanent save**.


Setting the autosave interval time

The autosave action is performed only if it is enabled and you define the **interval of time** the autosave operation is executed.

Select the **Preferences** option from the **File** menu. Within the **General** category, **enable the autosave** and set the **Save every minutes** property.

Permanently saving

To save all changes permanently, you have to do one of the following:


- explicitly **SAVE** the project by clicking the  **Save** button. This confirms the changes made up to that point and all backup files are deleted. The Save button is always enabled. Or,
- if you close the project without saving, a dialog box appears

where you can confirm changes.

This point is called **Checkpoint**.

All changes after **permanently saving** can be reversed, but no changes made before the permanent save.

Note

 when you publish and deploy a project, you need to confirm the permanent save as well.

How can you reverse the changes since the last permanent save?

If you decide to ignore all changes performed from the last checkpoint, you can do one of the following:

- confirm **not to save changes** when closing a project
- perform the **Revert to saved** operation. Select this option from the **File** menu

These actions copy the backup files to their original source and replace all modified files. In addition, reverting closes and reopens the project, except when done in the FuegoBPM Studio exit operation.

Chapter 3. Designing a Process

Designing a Process

In order to **design** or **model** a Business Process, it is important to understand the basics of Business Services Orchestration.

Automating business processes consists of managing the behavior of people, systems and organizations to orchestrate a repeatable business service.

FuegoBPM Studio allows you to quickly and easily model a process workflow scenario for your business. By creating roles, adding activities to the roles and linking activities to transitions, you can visually see how your business process flows.

The types of activities within FuegoBPM respond to the different participants of the process and their behavior.

- People
- Systems
- Organizations

The whole process has an Init or Begin activity and an Exit or End activity. All the action of the set of activities that compose the process is within both activities. This set of activities is called Process group as well.

The visual representation of your business process enables you to spot workflow redundancy and correct it on-the-fly while your process is up and running. The ability to quickly modify processes empowers you to make educated business decisions and to quickly implement them.

Instance

An instance is a single enactment of the process. The creation of instances is generally triggered by an event such as a customer order. Some examples of instances are:

- In a Hiring process, a prospective new hire.
- In an Order Management process, a new order.
- In a Patient Insurance Care process, a patient's insurance claim.





Some means must be built into the process to create an instance. There are several internal and external ways to create an instance. Instances are created in the Begin activity and stopped when they reach the End activity of the process. Once an instance has been created, it begins its flow through each activity in the process according to transition rules and Business Process (BP) Methods logic.

Creating an instance

There are several ways to create an instance, either internally or externally to a process. The following sections deal with some of the ways. As you create an instance, incoming arguments can be passed to the process.

Internal instance creation

An internal instance creation is generally started by an activity type that is designed to create instances. The following table lists activities that can create instances in processes.

Activity	Icon	Creation Method
Global Creation		The Global Creation activity is the most common way an instance is created in a process. It appears in Work Portal where an end user can manually click it, complete its BP-method and send the instance on in the process. No special BP-method is required to create an instance.
Global Automatic		The Global Automatic activity is another common way to create an instance. However, this activity type does not create instances automatically. The create method of the Process Instance component must be invoked from a BP-method in order to create an instance.
Subflow		The Subflow activity creates instances in the subprocess indicated in its Activity Properties dialog box. Instance creation is automatic.
Process Creation		The Process Creation activity also creates instances in the

Activity	Icon	Creation Method
		subprocess indicated in its Activity Properties dialog box. Instance creation is automatic.

ProcessInstance component

The *ProcessInstance* component contains the method **create**, which allows you to create an instance in the process indicated by the parameters of the method's template. The **create** method can be called from any BP Method. This component comes with the FuegoBPM Studio installation. It is located in the **fuego** module on the **Component Browser** in the BP Method Editor.

External instance creation

Instances can also be created by external events. The external event triggers a Global Creation activity in a process. One of the most common ways in which this happens is by integrating an existing Web Application with a process. For example, if a customer places an order through a Web Application, an instance can be created in the corresponding process. See [Creating instances from a Web Page](#) for further information.

Warning



In FuegoBPM Studio, the maximum number of instances to be created at a time is 5000. The server is able to execute that maximum number simultaneously. Therefore, it cannot create more instances than that number.

Process

Use FuegoBPM Studio to create, build and modify a business process. Each process targets specific business needs.

Each process is broken down into logical steps called activities. For example, for an Order Management process you might create activities called *Create Order*, *Check Inventory*, *Select Shipping Route*, *Check Customer Credit*, *Pick Product*, *Pack Product*, *Create Billing*, *Create Invoice*, *Print Invoice* and *Ship Product*.

Each activity is assigned to a role. Roles indicate who will perform the specific activity. You connect the activities to transitions in order to define the process workflow sequence from activity to activity. Transitions may be unconditional, conditional, due or exception depending on the type of workflow that is desired.

Version control built into FuegoBPM lets you modify processes on-the-fly, even if instances already exist in published and deployed processes.

Creating a process

To create a process, you need to:

- Right-click on the project tree node and select new process.
- Or Select the **New** from the **File** menu and choose **New Process**.

In both options, a dialog asking for the process information is displayed. In this dialog you must fill in the process name and other process properties.

The process is automatically stored in the project processes root directory.

Process

Name Localize

Id: *NewProcess*

Description Localize

Others

Variation:

Author:

☐ Generate events interactive activities

☐ Generate events for all activities

☒ Do not generate events

☐ Propagate unhandled exceptions

OK Cancel Help

1. Enter a process **Name**. This name will appear in the process design and in the Work Portal. This field is not limited.
2. Optionally, click the Localize button to enter the name in alternate languages. See Internationalization for further information on available languages.
3. Enter a process **Description** . Localize to enter the description in other languages.

4. Enter a process **Variation** . A variation is a label indicating that the process is a variation of another. For example, if you were creating a new copy of a process, you might give it a variation label. Variations are like same processes with different implementations.
5. Enter the process **Author**.
6. Select any of the **Generates events** check boxes if you want to enable the Audit Trail in Work Portal. See Generate Events.
7. Enable the **propagate unhandled exceptions** if the project has defined that all processes within it must handle exceptions but this new process is an exception to that rule. See Process exception flow for further information.
8. Click **OK** and the new process with the default **Begin** and **End** activities is displayed.

The Process Exception Flow that represents the most outer Exception Flow that will manage all non managed exceptions within the process also appears. See Process Group for further information on how to manage the Process Exception Flow.



Opening a process

There are two ways to open a process:

- Double-click on the process within the tree node in the left panel of the window (the default location).
- Select from the **File** menu, the **Open** option and then **Open a process**. Select one of the processes from the list. This dialog has an incremental search feature. Therefore, by typing the first letters of the process the list will automatically focus on the process that matches the search.

When a process is opened, a new panel is added to the central panel with the name of the process. This panel will contain the process design (graphical design) and the process toolbar where you can add activities, roles or transitions.

Note

 If the  icon is displayed next to the process in the project tree, this means that the process is read-only and you cannot modify it.

Importing a Process

- Select from the **File** menu, the **Import/Designs** options; or
- Right-click on the Project name or any Folder and select the **Import Designs** option.

You can import different kind of processes or graphics depending on the extension of the file (Visio: .vdx, ARIS: .xml generated by Aris, BPEL: .bpel)

New Folder

You can create folders within a project in order to group processes within it under certain criteria.

Folders help you organize your project if you have a large number of processes.

In the *Project* tree, right-click on the **Processes** title and select the **New Folder** option. Then **Move** (see this option below) all the processes to be grouped into this Folder.

Process Menu

There are additional functions and information you can specify to the process.

Run

Select this option to run the Simulation of the process.

See Simulation for detailed information.

Properties

Properties allows you to change and display general information about the process in FuegoBPM Studio. Select **Process** and **Properties** from the menu options and the **Properties** dialog box appears. Fields include the process' name, description, variation and author. **Variation** refers to a generation of the original or base process that differs in some way from the original process. For example, a company may want to use a variation of their Supplier Order Management process for their Order Management process.

The **Generates events** check box is selected when you want to enable the Audit Trail in the Work Portal.

The Audit Trail displays all events that have occurred for an instance.

Click the **Localize** button to set the process name in different languages. See Localizing processes for further information.

New Process Method

You can define a Process Method to be used in any of the activities of the process. See Process Method overview for further information.

New Variable

See External Variables

or,

See Instance Variables

or,

See Business Variables

or,

See Business Parameters

Open a Method

The list of process methods (Business Process Methods or BP-Methods) are shown in the **Structure** flap (by default on the left side of the FuegoBPM Studio.)

If you double-click on any of these BP-Method tree nodes, then the BP-Method Editor will display as a new panel in the central panel. Each process has one BP-Method Editor that is located next to the design panel of the process.

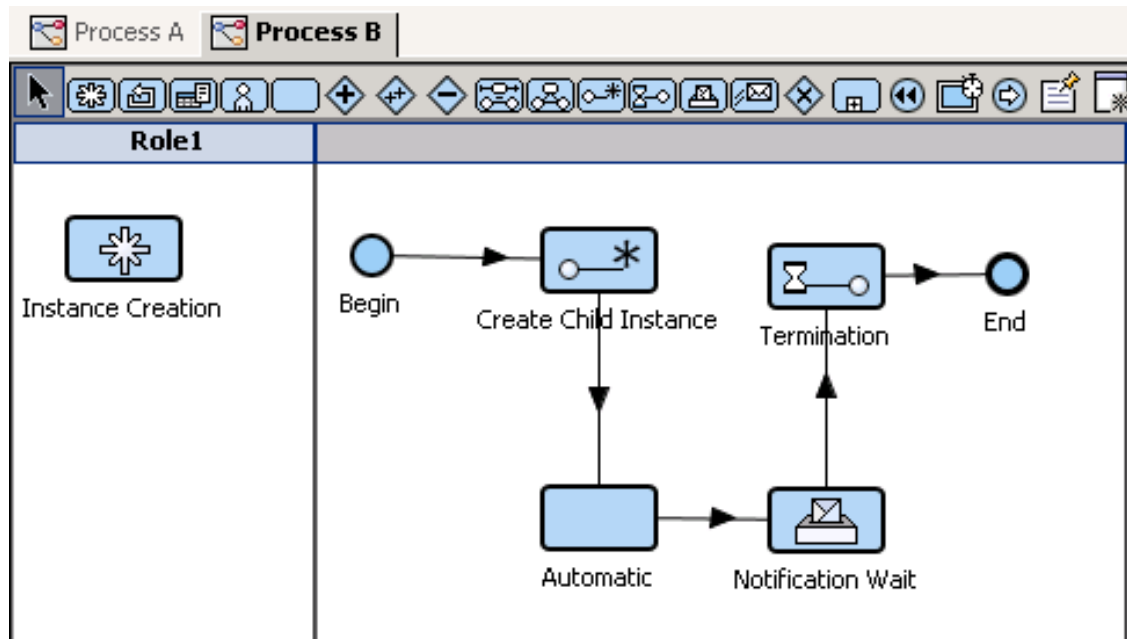
Once the BP-Method has been associated to an activity task, the BP-Method is also shown in the activity context menu.

The other way to open a BP-Method is through the **Process Menu**.

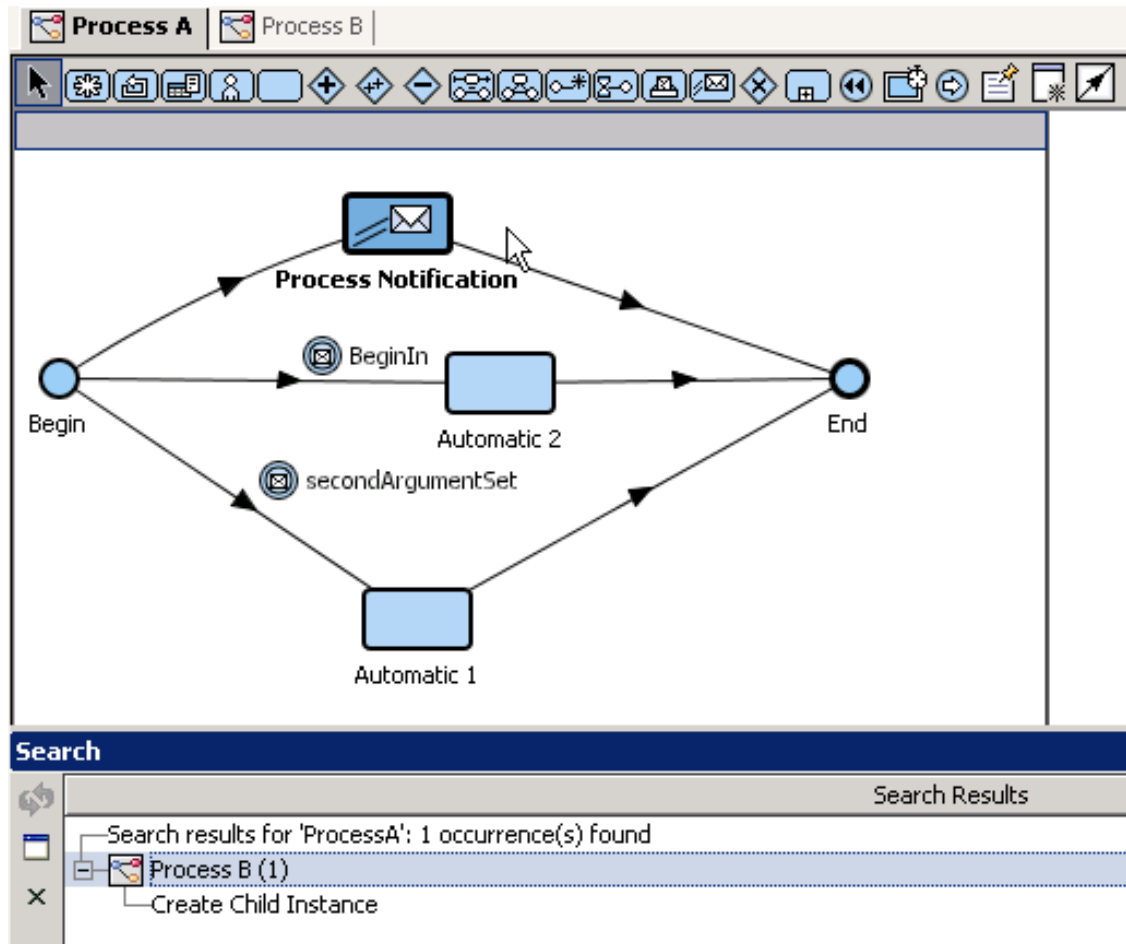
Find Usages

The **Find Usages** option on the **Process** menu searches within the project where the selected process is located. The process is *used* by another process if it contains IPC activities or it is a Procedure or a Screenflow. Therefore, it might be calling or being called by other processes. In the **Search tab** (in the bottom of the screen), all the related processes appear. The same with the activity to which the process is related.

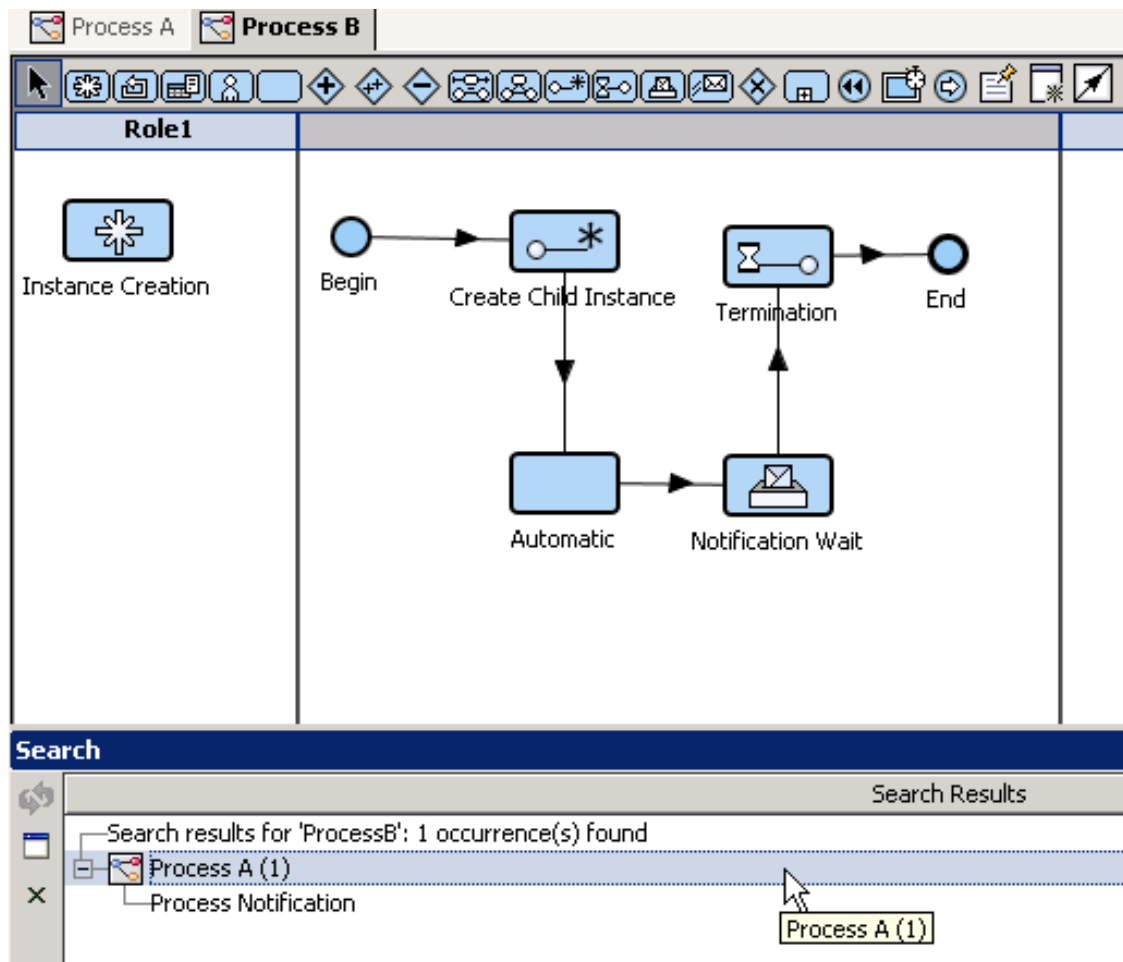
For example, in a **Process B**, you have a Process Creation activity that creates instances in a **Process A**.



Therefore, when you *find usages* for *Process A*, the search results will populate *Process B* and its *Process Creation* activity.



On the other hand, **Process B** has a Notification Wait activity that waits for the **Process A** Process Notification activity notification. Therefore, when you *find usages* for *Process B*, the search result will populate *Process A* and its *Process Notification* activity.



Checking the process design

FuegoBPM Studio uses a robust design checking feature to ensure that your process design will function correctly when it is published and deployed. The **Check Design** feature analyzes your business process in the following two ways:

- Validates the design.
- Syntactically and semantically checks the BP-methods.

Checking the project

You can check the whole project selecting the **Check all** icon. Each process is individually checked. Also, all processes and the catalog are cross-checked to verify any inconsistency.

If one process changed and has an impact on another process, if they are no longer compatible, the **check all** function detects this and warns you about the problem.

What does check design verify?

Check Design verifies the following, among other rules:

- Each activity has at least one transition and only one outgoing unconditional transition.
- Activities have only one due transition.
- Activities do not have an unconditional and a conditional transition to the same destination activity at the same time.
- There is not a Join activity without a corresponding Split activity.
- There is/are no transition(s) from/to activities of different levels of split/join.
- There are no incoming transitions to the Begin activity.
- There are no outgoing transitions from the End activity.
- Subflow and Process Creation activities refer to a valid subprocess (XPDL file).

A syntax check verifies the following, among other rules:

- All BP-methods are checked and compiled.
- All transition BP-methods expressions (due and conditional) are checked and compiled.

Use the **Check Design** option to analyze the process in the Process Designer by validating the graphical design and syntactically and semantically checking the process methods.

To check a process design:

1. Select **Process and Check Design from the menu options**. The check process begins. Or, you may click the **Check Design** button.
2. If there are any errors, the **Messages** tab appears with the errors listed at the bottom of the Process Designer screen.

Results

Any results from the check design process appear in the **Messages** flap at the bottom of the screen.

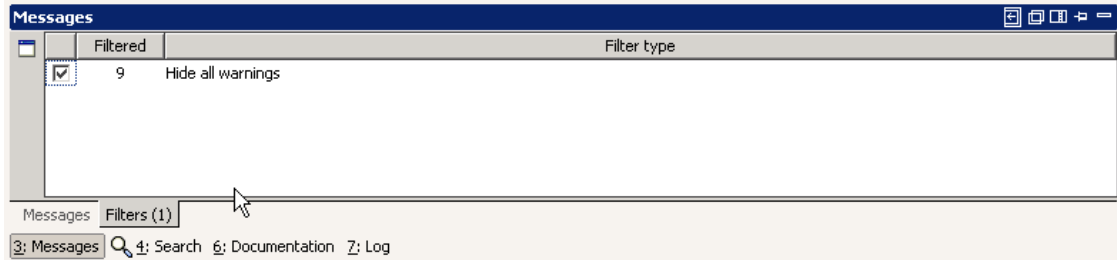
Double-click on each error to launch the BP-method Editor for the method associated to the error. If the error is in an activity or transition, the **Properties** dialog box opens for the object than contains the error. Correct the errors and repeat the steps above. Some errors may have a tip on how to fix them. For this kind of errors, a small light bulb will appear at the most right side of the error message. By right-clicking on the error, a fix option containing the different suggested ways to fix the problem will be shown .

After correcting every error, **minimize** the message tab or click on the tab below.

Filters

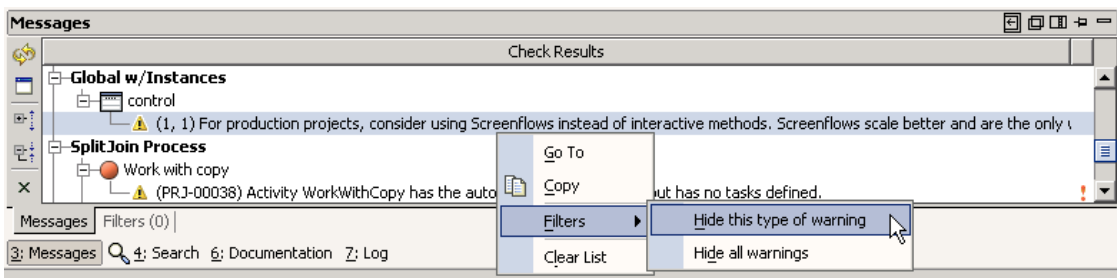
Filtering applies for warning messages. If you get a large number of warnings, you can hide them using the filter option.

By default you can choose to hide **all** warning messages. Select the **Filter** tab within the **Message** flap and check the *Hide all warnings* check box.

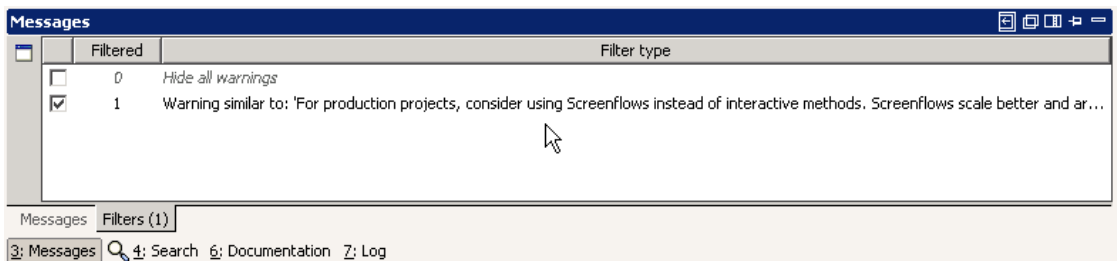


You can also filter any of the displayed warning messages by type as long as the warning message applies to a BP-Method .


From the list of warning messages displayed on the Message tab, select one. Right-click and select the **Filters** option. Choose to **hide this type of warning**.



A new filter is added to the list of filters and you can enable or disable it as needed.



Note

 If you select the *Hide all warnings* then all other filters are disabled.

Copy to

Allows you to duplicate the selected process in the same project, change its name and modify it as required.

This is typically used if you have a new process to design which is very similar to an existing one, so you copy it and work on the new copied one, as necessary.

Move to

Generates a new process based on an existing one and deletes the existing one. It is used to rename a process or to move existing processes into a new **Folder**.

Export Process

Generates a file (*.xpd file*) where all the process information design is stored. This file can be later *Imported* into another project in order to re-use it.

Process Report

The process report describes all the process documentation as well as other activities and elements within it.

To complete the Process documentation, see Process Documentation.

The options to generate this report are:

1. **Include use case documentation in reports:** includes the use case documentation for the process and the activities.
2. **Include methods source:** includes the BP-Methods statements in each activity in the report.
3. **Include variables:** lists all instance variables defined for the process and links them to the correct activity in the report.

1. Make the appropriate selections and then click **Ok**. The Generate Process Report dialog box appears.
2. The **File name:** field auto-populates with the name of your process. You can change this as appropriate.
3. Select the location where you want to save your process report.
4. Click **Save** .
5. Open your Internet browser.
6. Select **File** and then **Open** from the menu options.
7. Browse to the location where you saved your report and select it. The process report will appear in your Internet browser window.

The **Process report** produces a detailed report of the process in HTML format which can be viewed in a web browser or published to an Internet/Intranet site.

The generated report is composed of the following sections:

- Diagram of process. Click on any item in the diagram of the process design and you will jump to the appropriate topic later in the process report.
- **Process Information** lists any global information about the process. This information is entered from the **Process** menu by selecting **Process info** [Alt+I].
- Documentation lists any global documentation on the process. This information is entered from the **Documentation** panel.
- Use Case Documentation lists any global use case documentation on the process. This information is entered from the **Process** menu by selecting **Use Case Documentation and Default or the appropriate language**.

Contents

The Contents section is basically a Table of Contents for your process report.

- **Role List** lists all roles included in your process. The role names are in hyperlink format (highlighted and underscored). Click on a role to view a section of the report that describes the role and the activities associated to it.
- **Activity List** lists all activities included in the process. Like the roles in the Role List, each of the activities is linked to a later section of the report that describes the activities and their attributes.
- **Exception List** lists any exceptions in the process. The exceptions are also linked to a later section of the report that describes the exception and its attributes.
- **Variables** lists each role in the process and each activity associated to that role. Each activity is followed by any activity description, BP-Method, transition and/or argument associated to that activity. Next, the process variables are listed by name and type. Each BP-Method and BP-Method description ends the report.

Generate Process Interface

A process interface is a version of a process that displays only the Begin, End and any Notification Wait activities. It is used in Business to Business (B2B) scenarios where one company must share their process design with an external company.

All process argument variables are available in the process interface design which enables the process interface design to be called as a subprocess.

To create a process interface

1. Open the process in Process Designer.
2. Select **Process** and **Generate Process Interface** from the menu options. The **Save** dialog box opens. By default "**_Interface**" is appended to your process name in the **File name:** field.
3. Browse the location where you want to save the interface.
4. Click the **Save** button.
5. The process interface can be e-mailed to an external company.

To import the Process Interface in the other project, use the **Import Design option**.

Searching for a process

See Searching for a process, procedure or screenflow.

Saving a process

Processes are saved in Extensible Markup Language (XML) format. The saved file uses the file extension .xpdI appended to the process name. XPDL stands for XML Process Definition Language.

Processes must be saved before you can publish and deploy.

Saving the process

To save the process

1. From the FuegoBPM Studio menu, select **File** and then **Save**, or
2. Click **Save** on the FuegoBPM Studio toolbar.

See Saving the project for further information on how to save or revert changes of the process.

Process deadlines

There are two types of process deadlines that can be used in a process design: due transitions and instance expiration deadlines.

Due transitions

A due transition is a transition that connects two activities to force an instance to flow between the activities within a certain interval. When the due transition is added, code is defined in the Due Interval field to indicate the interval.

If the due interval is set to '5m', the server forces the instance to flow through the due transition after five minutes even if any mandatory BP-method in the first activity is not completed.

See Time in the Due Transition topic for other examples of BP-method time syntax.

Instance expiration deadlines

An instance expiration deadline time is set by utilizing the predefined variable *processDeadline*. **processDeadline** can be set to the same interval in which an instance must complete, to a maximum, its flow through a process. If the deadline expires, the server automatically sends the instance to an Instance Expiration exception handler (for further information, see Exception handling).

The *processDeadline* variable has to be set in the **immediate activity** after the Begin activity.

For example, the BP-Method's of this first activity after the Begin activity dictates that an instance has 20 minutes to flow through the process to the End activity. The instance automatically moves to an *Instance Expiration handler* if the deadline is exceeded.

First, set the *processDeadline* to the current time ('now') plus 20 minutes ('20m') to ensure that an instance completes its flow from the Begin to the End activities in 20 minutes.

```
/* Set process deadline to 'now' (current time) plus
20 minutes.*/
processDeadline = 'now' + '20m'
```

Second, design the *Instance Expiration* exception handler. The expired instances are directed to a new Interactive activity to ensure that instances exceeding the 20 minute deadline continue processing. The *Handle Exception* activity catches the instance that has exceeded the process deadline and, for example, sends an e-mail to somebody or a new interactive activity receives it, waiting for a manual solution. For example, in a Shipping process, consider that the instance is sent to a Supervisor in order to determine what to do.

```
/* Give the Shipping Supervisor the ability to expedite
the order if it has exceeded
the processDeadline.*/

display "Do you want to ship the product now?"
using title = "Ship Product?",
type = "question",
options = ["Yes", "No", "Cancel"],
default = "Yes"
returning selected = selection

/* action = BACK sends the instance back to the activity
where it was when the
processDeadline was reached.
You must first reset the processDeadline*/

if selected == "No" then
    processDeadline = 'now' + '10m'
    action = BACK
elseif selected == "Cancel" then
end
```

Note



An Instance Expiration exception must flow to an activity that is not in the main flow of the process or to the End activity. For further information, see Exception overview.

Printing a process design

The **Print** option prints the process design as it appears on your screen. Select Print [Ctrl+P] from the file menu and the process prints using default print settings (8 1/2" x 11" paper in Portrait orientation). To change these default settings, use the Print/Preview function.

Print/Preview displays a view of the process design before it is printed and allows you to make changes to printer settings.

To modify printer settings

1. From FuegoBPM Studio, select **File** and then **Print/Preview** [Ctrl+Shift+P]. The **Print/Preview** dialog box appears.
2. Scale the width of process by selecting the arrow on the scale bar and moving it to the left to shrink the process or to the right to stretch the process. You may also manually increase and decrease the percentage value in the Zoom: field.
3. Click **Print** to print the process as it appears in the Print/Preview dialog box or proceed to step 4 to make further alterations to the print options.
4. Click **Setup** to open the Page Setup dialog box.
5. Choose different paper sizes, orientations and adjust margins in the Page Setup dialog box. Click **OK** to print. The **Printer** button allows you to select alternate printers or options associated to your printer.

Finding an object in the process

The **Find** function searches for text in the process and finds any text in all elements in the process design.

To find an object in a process

1. From FuegoBPM Studio, select **Edit**, then **Find** and the Find dialog box appears.
2. Type the search text in the **Find words** drop-down field.
3. Select the **Match Case** check box to find text exactly as typed in the **Find words** field.
4. Select the **Regular expression** check box if you are using any regular expression character in the **Find words** field.
5. Click **Find** .
6. The search results appear in a Search Results folder in the lower part of the Process Designer window.
7. To display the object associated to the search term, double-click on any of the results.

Subprocesses

Overview

Subprocesses are processes that are called by an initiating process from certain activity types. These activity types that call a subprocess:

- Make a complex process more easily understood by using the activity to abstractly represent the underlying process.

- Enable reuse. The subprocess can easily be reused by many calling processes.
- Enable Business-to-Business (B2B) communication between processes. The subprocess can be run in another company behind their firewall.

Subprocess activity types

The activity types that can call a subprocess are listed in the following table.

Activity	Type	Description
Subflow	Synchronous	When an instance reaches a Subflow activity in a process, the Creation Subflow code is run. Argument variables expected by the subprocess' Begin activity are set by the Argument Mapping. After the Creation Subflow code has finished, an instance is created in the subprocess. The original instance waits in the Subflow activity until the instance passes through every activity in the subprocess. When it reaches the subprocesses' End activity, it is returned to the calling process'

Activity	Type	Description
		Termination Subflow code. Variables defined in the Argument Mapping are updated to reflect any changes made in the subprocess.
Process Creation	Asynchronous	A Process Creation activity can be used to call a subprocess asynchronously. This means that when an instance reaches the Process Creation activity, an instance is created in the subprocess. Both instances continue in the main process and in the subprocess simultaneously. Argument variables expected by the subprocess's Begin activity are defined in the Argument Mapping.
Process Notification	Synchronous	Process Notification activities send notifications to Notification Wait activities in different processes. Instance ID information is created and tracked by the FuegoBPM Server for each instance created in a process. Instance ID

Activity	Type	Description
		information is stored in the predefined variable instance Id. The Process Notification activity uses instance Id to communicate with a specific corresponding instance that is waiting in a Notification Wait activity. Argument variables are passed to the Notification Wait activity and the instance variables are updated accordingly. The instance is then free to continue its flow through the process.

Creating a subprocess

When do you decide to create a subprocess? Basically, at any time you have a set of activities that can be repeated in more than one process. For example, in most companies, a finance department or accounting group is responsible for verifying and maintaining a yearly budget. To keep within the budget, this group must know how much money is being spent by other groups in the company.


Since requesting money from a finance department is a repeatable process that is accessed by several other processes, it is an ideal candidate to become a subprocess. That being said, how do you create a subprocess? There are two different ways: by leveraging an existing process and by creating a subprocess on the fly.

Calling an existing process

To create a subprocess by calling an existing process

1. Decide whether your process and the subprocess can work asynchronously or synchronously. This will tell you what type of subprocess activity you need to use.
2. Add an automatic role lane to your process design to hold the subprocess activity.
3. Add the required activity, either a Subflow activity (synchronous processing) or a Process Creation activity (asynchronous processing) to the role lane. The **Activity Properties** dialog box is displayed.
4. Click on the **Related Process Category** .
5. Browse to the location where the subprocess exists and select it.
6. Click **Open**. Fields in the Related Process Category automatically populate with the process name and location.
7. Click **Ok**. The activity appears in the role lane.

Note

 Processes interface by passing arguments in between them. See Argument Mapping for detailed information on handling arguments.

Creating a subprocess from a group of activities

In the middle of process development, you may see that some parts of your process are complex and make the overall process design cumbersome. You can send some of the more complex roles in your process to a subprocess.

You may decide to move some activities to a subprocess in order to reduce the complexity of your main process. You can easily do this by using your mouse to draw a box around the selected activities

To select a group of activities and create a subprocess


1. Left-click the mouse on the upper-left corner of the area you want to select.
2. Drag the mouse to the lower-right corner of the area you want to select.
3. Release the mouse button. The activities are surrounded by a box. Notice that selected activities' names are in a bold font.
4. Now you can right-click on the selected area and choose **Move selection to a subprocess** or **Create a process with selection** from the shortcut menu.
5. The **New Process** dialog box appears requesting new process information.
6. Enter a name in the **Name** field and a description in the **Description** field. The **Description** field auto-populates with information from the **Name** field. You can select this information and type over it if you want to.
7. Click **Ok** . The **Save** dialog box appears.
8. If you selected **Move**, in the original process, the activities you selected are turned into a single **Subflow** activity having the same name as the one you gave the subprocess in the step above. If you selected to **Create** a process, the original group of selected activities will remain as they are in the original process.
9. You must now connect your Subflow activity to the rest of the process by adding transitions. See Transitions for further information.

10. In the new created process, you will notice that the Begin and End activities and automatic role lanes appear in the subprocess as well as the activities you selected from the original process.
11. You will have to add the appropriate roles and transitions and any additional activities to the process. See Roles, Transitions and Activities for further information.

Process Group

The **process group (PG)** is the group that contains **all the activities within the main flow of the process** as well as any activity within the process exception handler flow. The Process Group is not displayed as a group but it is the most outer Group of the process.

If an exception occurs within any activity of the main flow of the process and no Exception handling for that activity is defined, the **exception flow of the Process** will handle the exception.

When a new process is created, Studio can automatically create the Process Exception represented as  or you will need to create it in each process.

FuegoBPM generates the Process Exception Flow automatically if you have the **Handle exceptions** property enabled in the Project preferences.


See Process Exception Flow for further information.

Process Exception Flow

In the event that an exception occurs within any activity of the process and no Exception handling is defined for that activity, the **Process exception flow** will manage such exception.

It is recommended to define different levels or Groups of activities for which you can design your own Exception Handler Flow. If an

exception occurs in any activity and no group handles that exception, the **Process Exception Flow** will manage it.

When a new process is created, FuegoBPM can automatically create the Process Exception flow initiated by a .

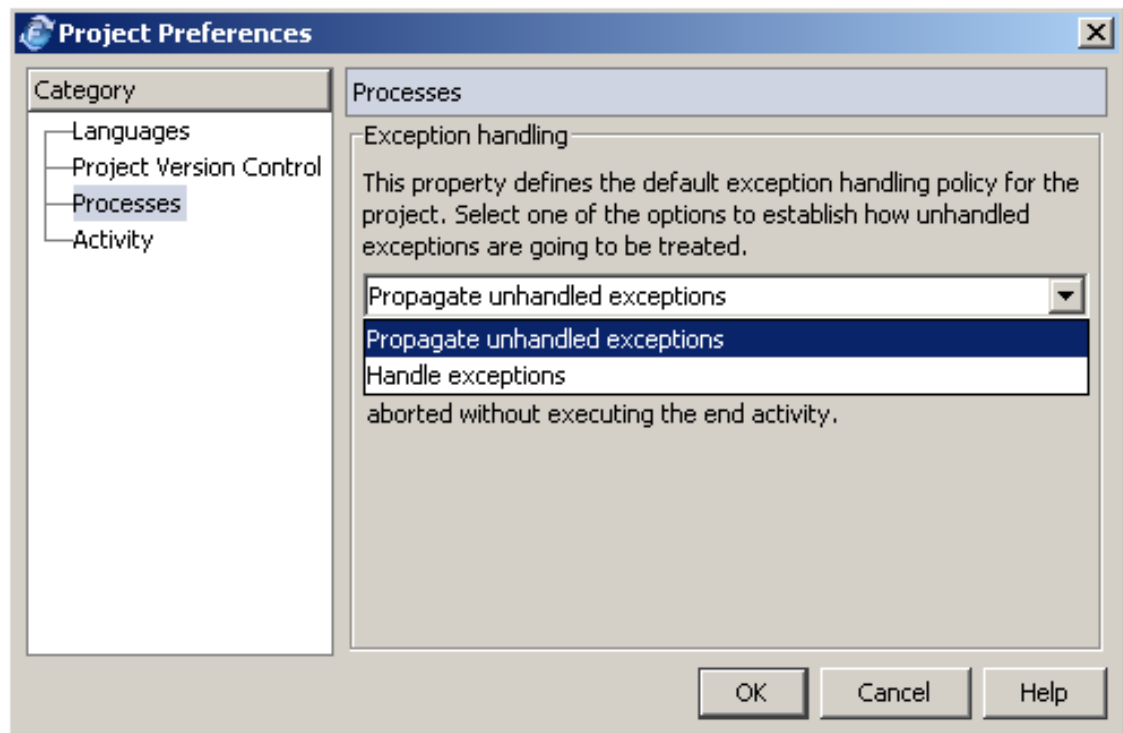
It is highly recommended that you always define the Process Exception Flow. Thus, you can make sure that all processes in your project handle exception flows.

Defining the Process Exception flow

If you want that all your processes within the project manage the **Process Exception Flow** on a mandatory basis, you need to define the following preference:

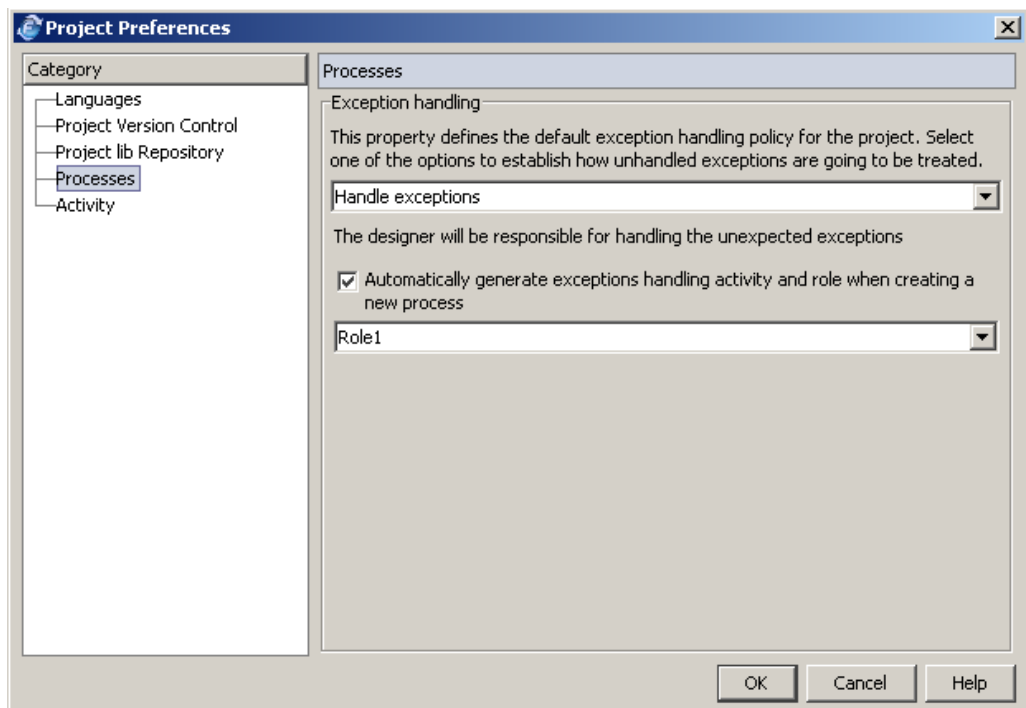
- From the **File** menu, select **Project Preferences** and the **Processes** Category.

You have 2 options:

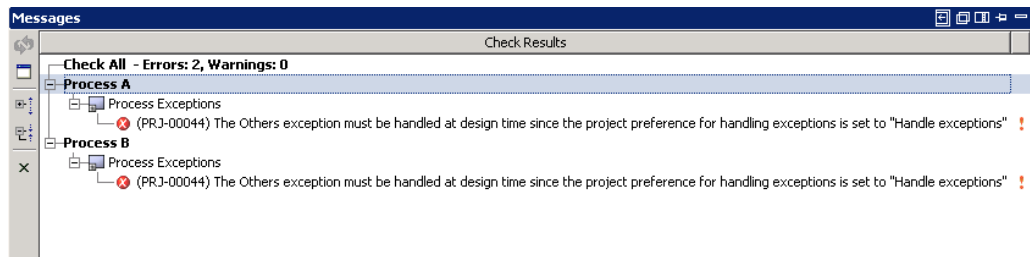


- **Propagate unhandled exceptions:** handling exceptions is not mandatory, so no default flow is generated.
 - If the process corresponds to a subprocess, called by a parent one, then the exception is propagated to the parent, which is expected to handle the exception.
 - If the process has no parent, then all instances that throw a non-handled exception cannot be managed by any flow and are aborted (bypassing the End activity.)
 - The process is compensated.
 - Even though it is not mandatory for the project, you can define the Process exception flow for any of the processes. See below.

- **Handle exceptions:** if you determine that all processes have to handle the Process Exception flow, then FuegoBPM can automatically generate it or you will need to create it in each process.
- If it is automatically generated, a Interactive activity is created. Therefore, you need to define the **role** in which it is created. This role represents who can manually take care of the exception. If required, you can later modify the automatically created Process Exception flow.
- A compensate activity is generated as well.




- If no Process Exception Flow is created, as it is defined mandatory, an error will populate at check design time.



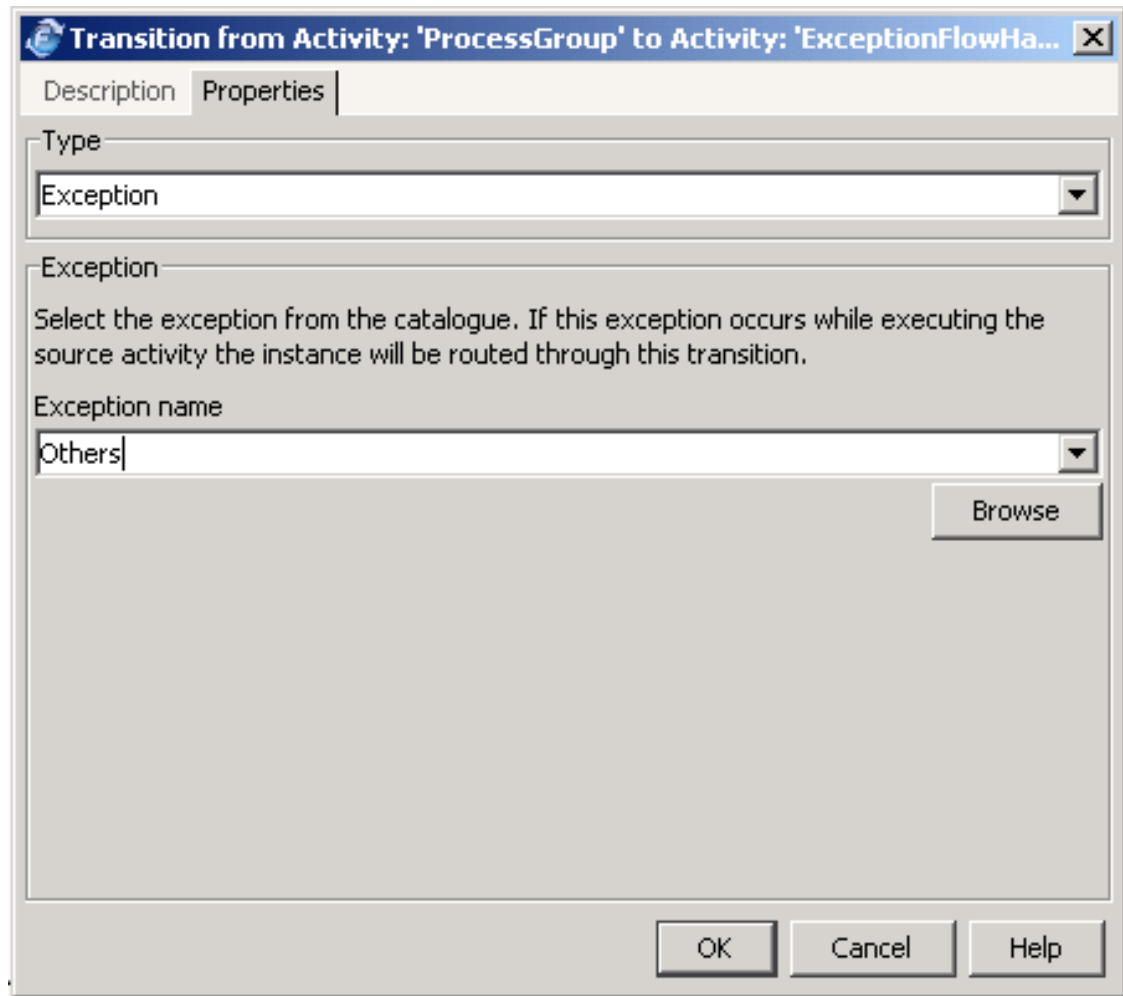
- You can select to fix it. A dialog is displayed for you to complete the required information.

Note

 If a specific process within a project that **handles exceptions** requires no **Process exception flow** definition, then at process creation time you can enable for that process the **propagate unhandled exceptions** property. Be aware that all exceptions that occur within the process are expected to be managed by the parent process.

Creating a Process Exception Flow

- To manually create a Process Exception Flow, you need to create the first activity in the flow.
- Next, in the designer workspace, right click and select **Add exception transition to**.
- Select the created activity.
- The exception dialog populates. In the properties tab select the **Exception name: Others**.



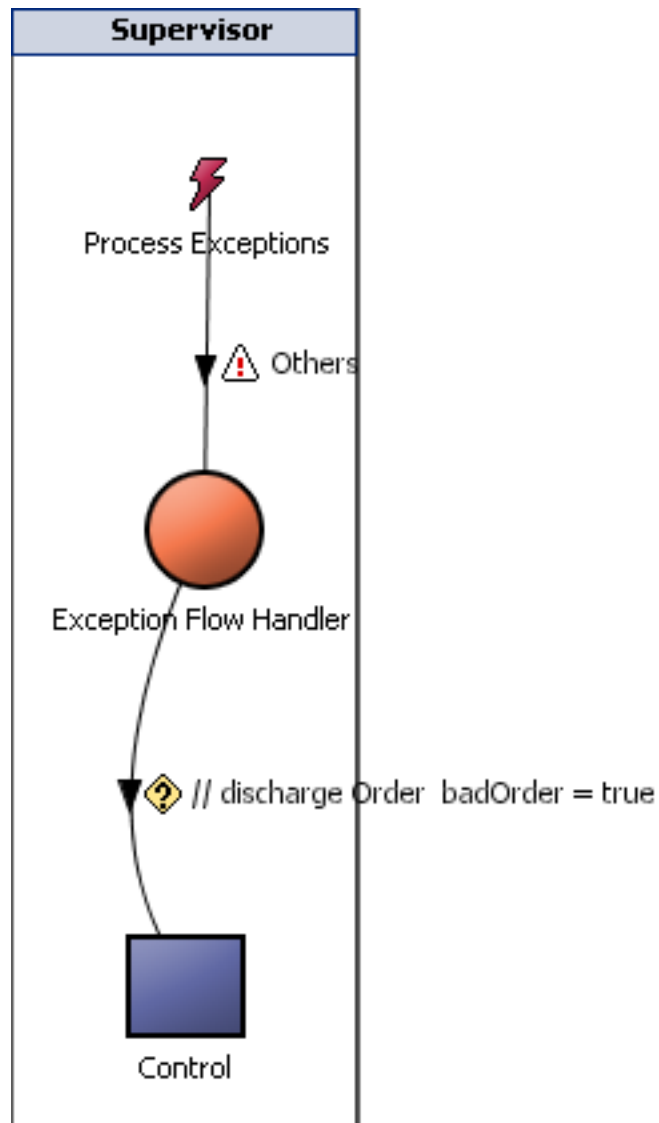
- The Process Exception flow is created.

It is very important to add the correct exception handling flow for those instances that are thrown out of the main flow of the process into the Process Exception Flow.

If you do not handle them correctly, all exceptional situations not considered at designing time will cause the instance to terminate with no further treatment (bypassing the End activity.)

For example, add at least an Interactive activity so that all instances for which a non-expected exception occurs will appear in a

supervisor task to be treated.



In the example above, the Supervisor has the option to *send back* the order to the main flow (this decision is made as one of the activity's tasks) or to send it to an Automatic activity for further control because the order is discharged.

Part of the task's BP-Method is:

```

if(!badOrder)then
    action = BACK

```

end

Note



If you do not want the Exception Process Flow to display in the process design, then, from the **View** menu **Exceptions** option, disable the **Show Exceptions** preference.

What happens if there is an Exception within the Process Exception flow?

The instance will be submitted to the **Outest Default Exception Flow**. This is a **Default Exception flow** managed directly by the **Server** (not drawn.)

In this case, the compensation for the whole process is executed and the instance is terminated bypassing the End activity.

For further information on **Exceptions** and **Exception Handler Flow**, see Exception Handling.

Importing a process from Visio

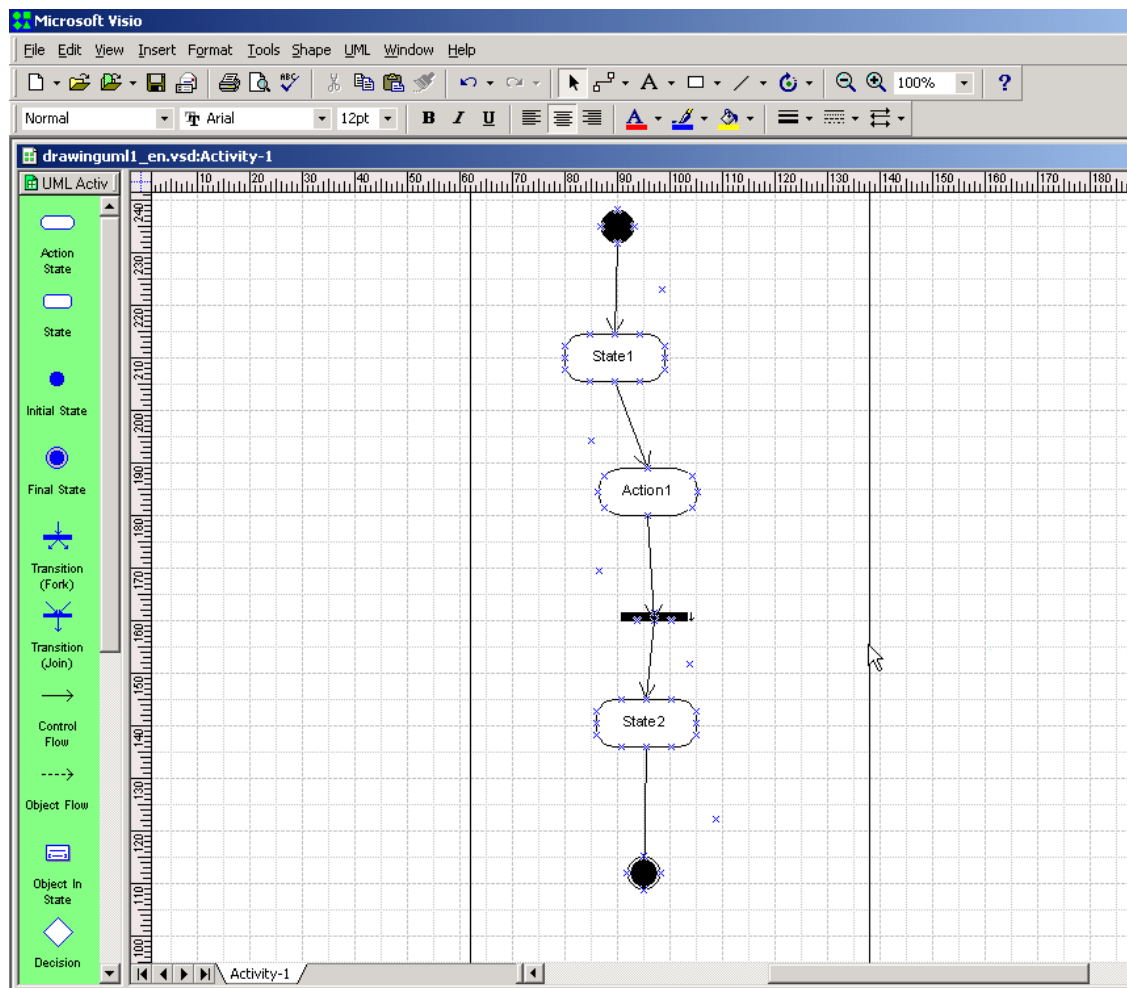
You can import a Visio process drawing into the designer as a new process.

Process to import a Visio Drawing

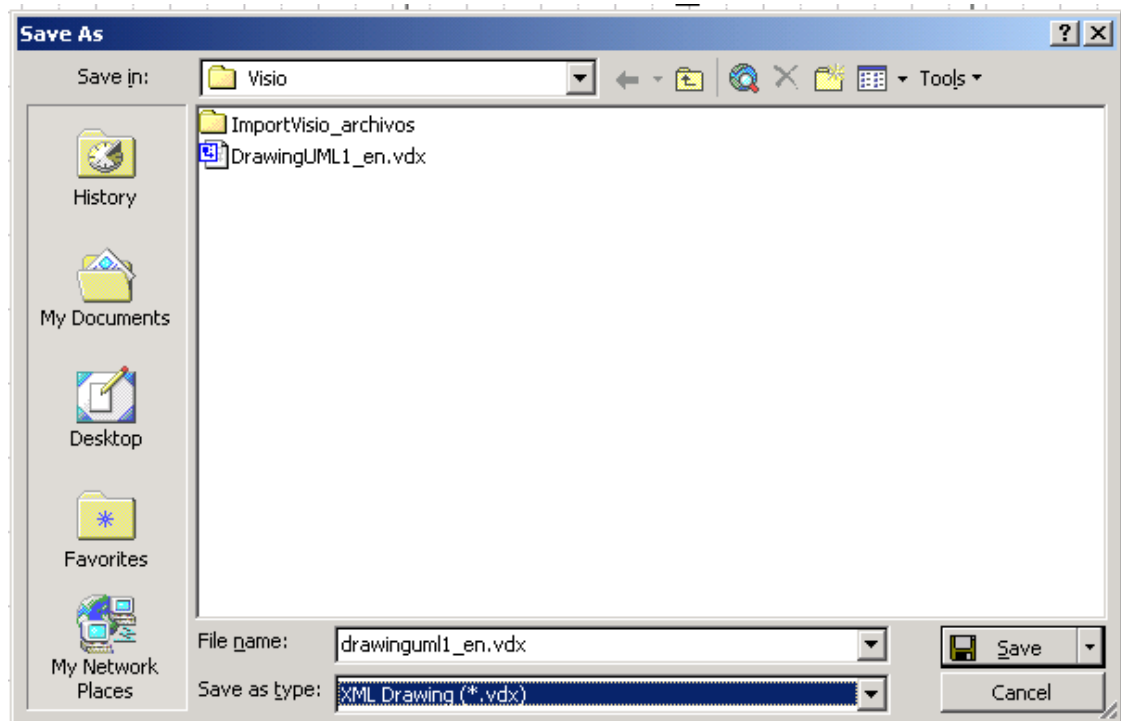
Obtaining a Visio schema

Once the drawing has been created and edited with Microsoft Visio, it must be stored as a XML Drawing with the *.vdx* extension. This is the only format that is enabled for import into FuegoBPM Studio/Designer.

The following image illustrates a Visio drawing:



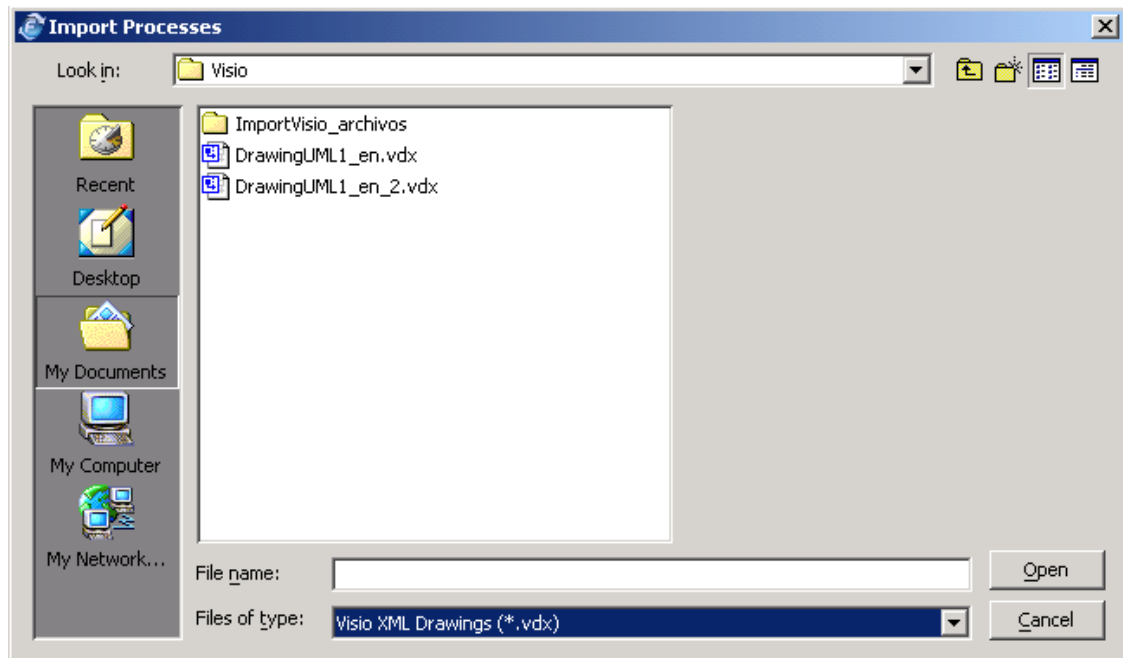
While saving, select the *.vdx* format:



Starting to import the file

Once the *.vdx* has been generated, open FuegoBPM Studio/Designer and select from the **File** menu the **Import** option and then the **Designs** option. Or from the Project tree, right click and select the **Import Designs** option.

A dialog box is displayed. Select to see all *.vdx* files:




The file to be imported can be chosen from this dialog.

Editing Import Rules

In the following step, the **Import rules** have to be defined. These are a set of Pairs (Shape, Activity type) that determine the mapping from *Shapes* (Visio objects) to *Activities* (FuegoBPM Studio/Designer).

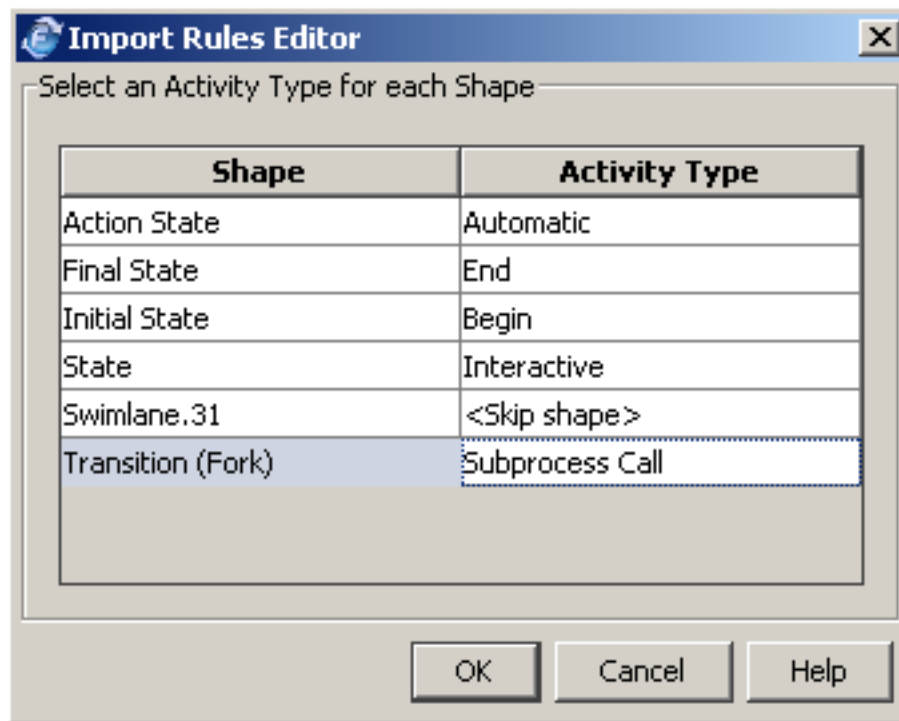
These rules should be set carefully because incorrect matching will produce an inconsistent or even, meaningless process.

Note

 You must assign an activity to every shape. This will prevent you from continuing with the import.

To ease future imports, the rule for each shape is stored. Therefore, whenever a new drawing with that shape is imported, the suggested rule for that shape will be the last one stored.

As an example, the following shows the rules editor with six import rules:



Creating the Process

After setting the import rules, the process is automatically created. A new process is created with the same name as the file that was imported.

Select **Process** and **Check Design** from the menu options to check the process. Some manual adjustments might be required.

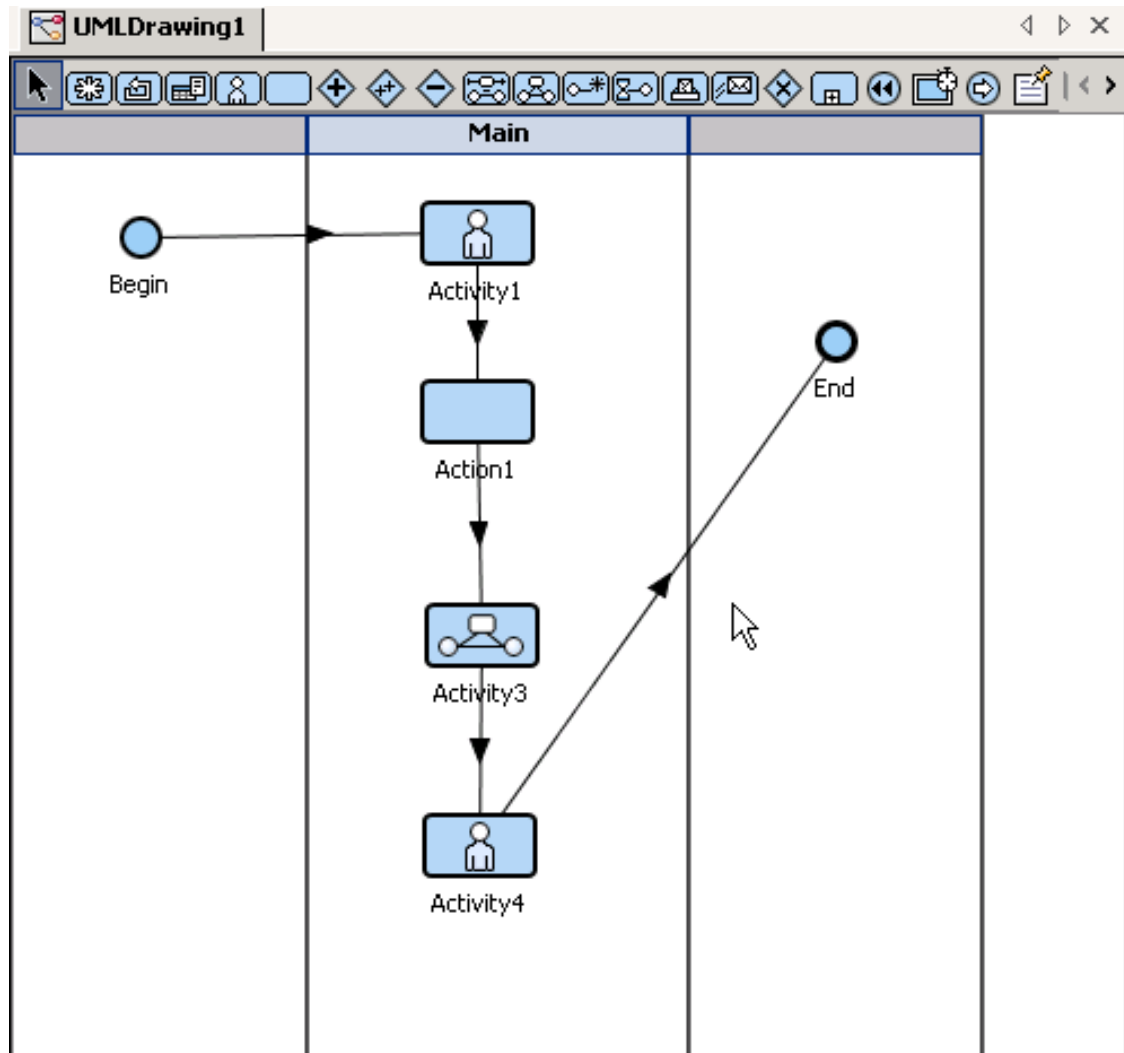
Saving the Process

The process is created from where the **Import Process** option was selected. If it was selected from the **File** menu, then the new process will appear below the Project Name. If it was created from the project structure, it will appear where the import option was executed (for example, within a folder).

Import result

For the example given above, the FuegoBPM Studio/Designer

process appears as follows:



Importing a process from Aris

You can import an Aris process drawing into the designer as a new process.

Process to import an Aris Drawing

Obtaining an Aris schema

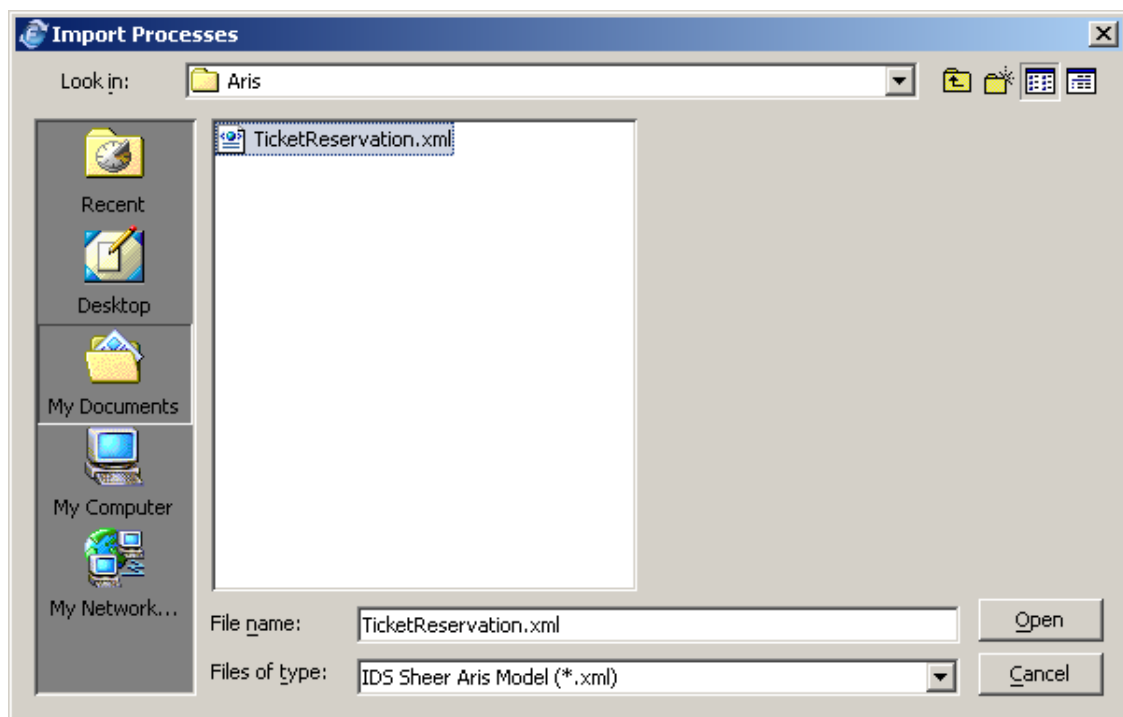
Once the drawing has been created and edited with Aris, it must be stored as a XML file. This is the only format that is enabled for

import into FuegoBPM Studio/Designer.

Starting to import the file

Once the *.xml* has been generated, open FuegoBPM Studio/Designer and select from the **File** menu the **Import** option and then the **Designs** option. Or from the Project tree, right click and select the **Import Designs** option.

A dialog box is displayed. Select to see all *.xml* files:



The file to be imported can be chosen from this dialog.


Editing Import Rules

In the following step, the **Import rules** have to be defined. These are a set of Pairs (Shape, Activity type /Role) that determine the mapping from *Shapes* (Aris objects) to *Activities* (FuegoBPM Studio/Designer) or *Role*.

By default, the corresponding FuegoBPM activity type is suggested for some of the Aris objects.

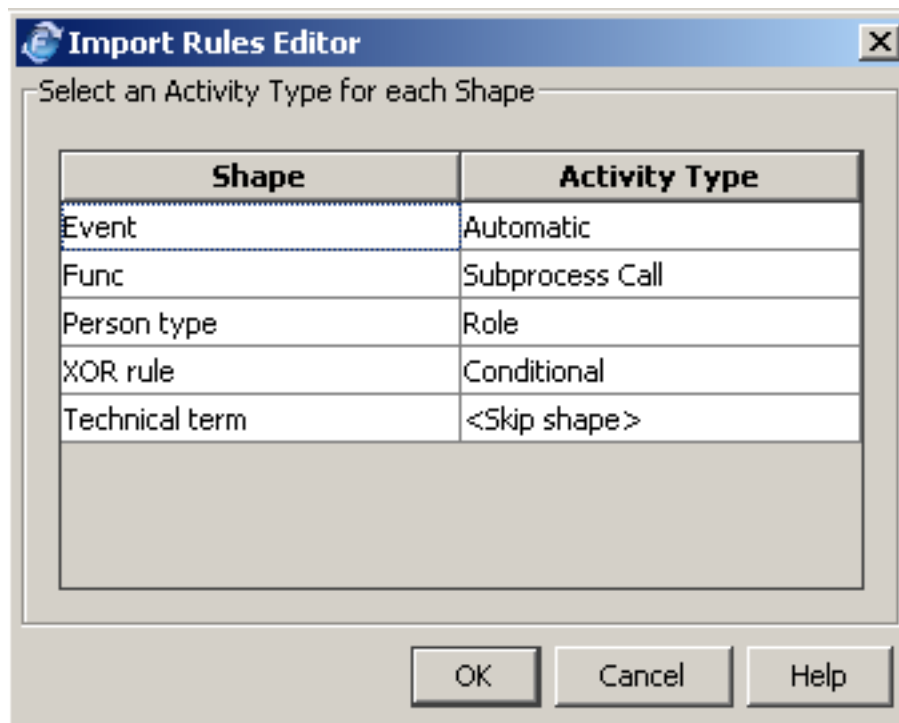
These rules should be set carefully because incorrect matching will produce an inconsistent or even, meaningless process.

Note


 You must assign an activity to every shape. This will prevent you from continuing with the import. If the Aris shape is not relevant to import into FuegoBPM, select the *Skip Shape* option

To ease future imports, the rule for each shape is stored. Therefore, whenever a new drawing with that shape is imported, the suggested rule for that shape will be the last one stored.

As an example, the following shows the rules editor with five import rules:



Note

 If any of the Aris objects represents a role, match it as **Role**. In this case no FuegoBPM activity type is defined.

Creating the Process

After setting the import rules, the process is automatically created. The main new process is created with the same name as the file that was imported.

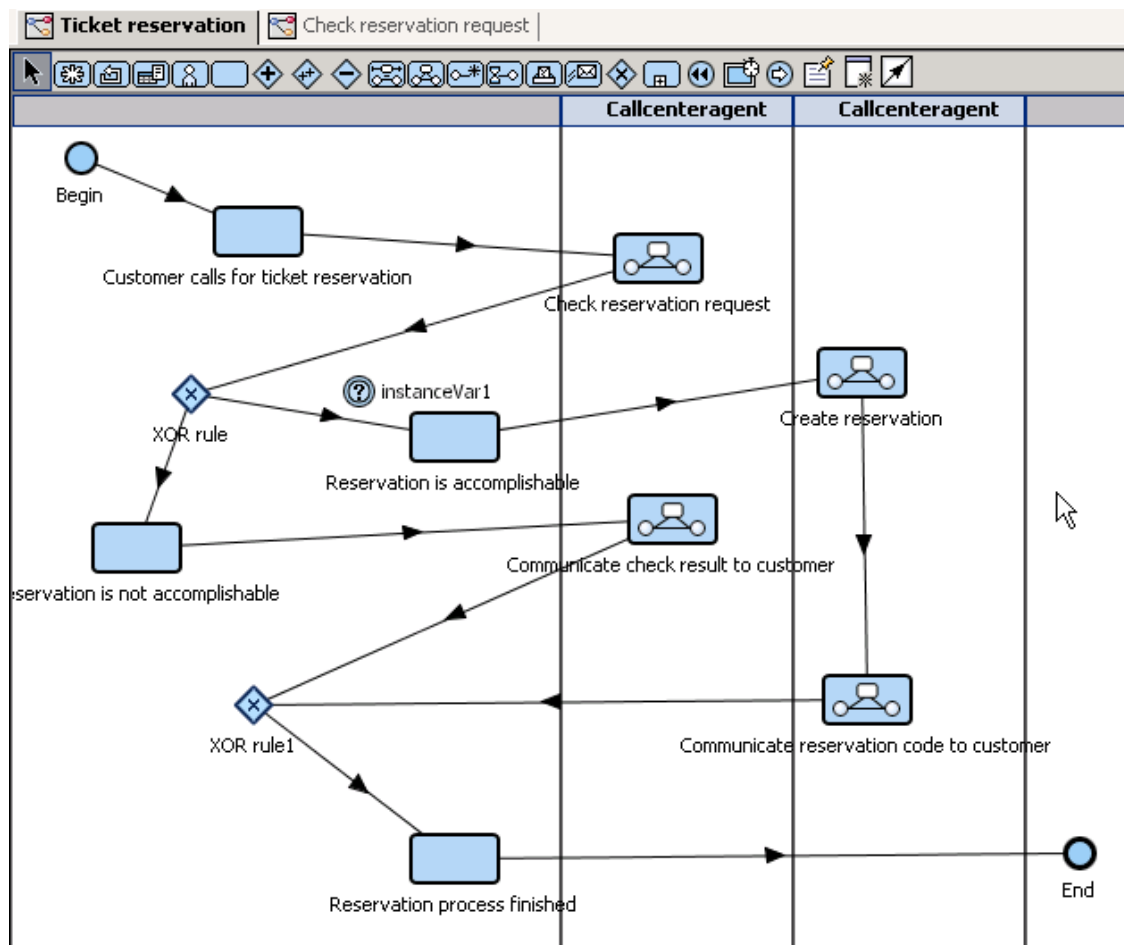
Select **Process** and **Check Design** from the menu options to check the process. Some manual adjustments might be required. You will need to add the transition from the **Begin** activity to the first activity in the process. As well you need to add the transition/s to the **End** activity.

Saving the Process

The process is created from where the **Import Process** option was selected. If it was selected from the **File** menu, then the new process will appear below the Project Name. If it was created from the project structure, it will appear where the import option was executed (for example, within a folder).

Import result

In FuegoBPM Studio/Designer, the process appears as follows:



Importing a process from Workflow Management Coalition - WfMC

You can import a WfMC process drawing into the designer as a new process.

Process to import a WfMC Drawing

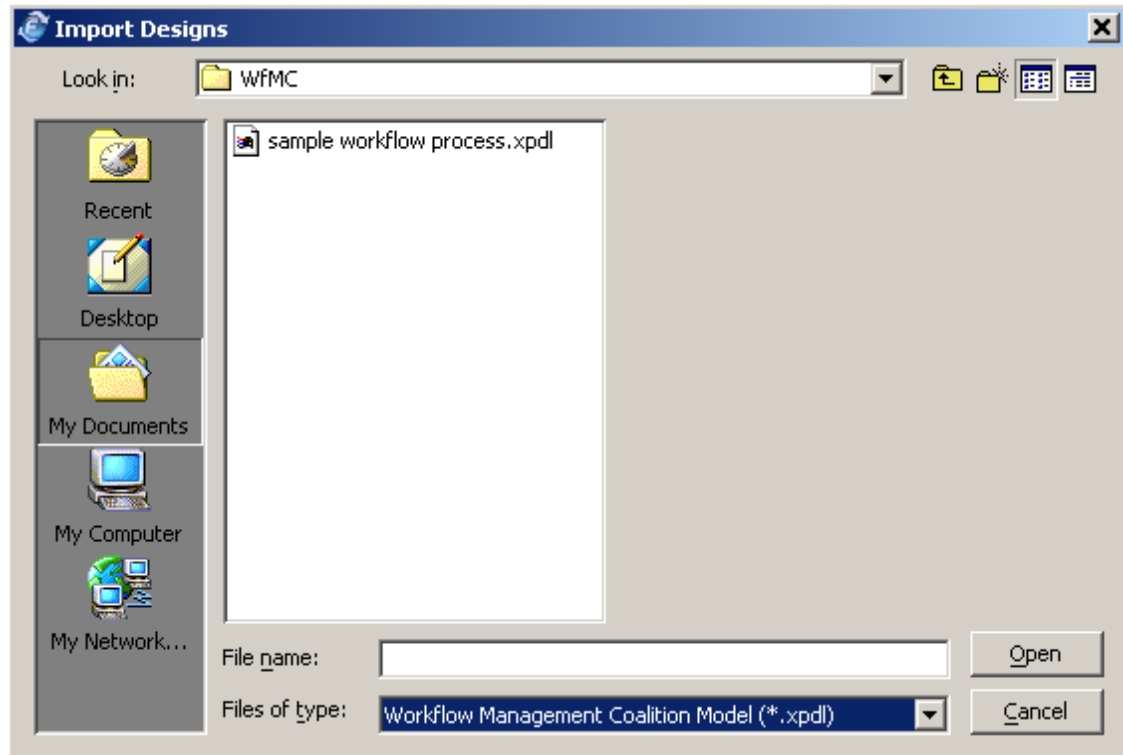
Obtaining a WfMC schema

Once the drawing has been created and edited as WfMC, it must be stored as a XPD file.

Starting to import the file

Once the *.xpd* has been generated, open FuegoBPM Studio/Designer and select from the **File** menu the **Import** option and then the **Designs** option. Or from the Project tree, right click and select the **Import Designs** option.

A dialog box is displayed. Select to see all *Workflow Management Coalition Model .xpd* files:



The file to be imported can be chosen from this dialog.

Import Rules

When the import is completed, one or multiple processes in FuegoBPM Studio are generated.

There are automatic rules that do the mapping for all the objects in between WfMC and FuegoBPM:

- Basic objects mapping

- Process mapping
- Activity mapping
- Transition mapping

Basic objects mapping

Workflow Management Coalition XPDL	FuegoBPM's Model
Workflow process	Fuego process
Participant /Performer	Role
Activity	Activity
ActivitySet	Group
Transition	Transition
Formal Parameter	Begin's or End's argument mapping
Data Fields	Process Instance variables

Process mappings

The importer assumes that each Workflow process included in the package is a FuegoBPM process and maps the following data:

- Id
- Name
- Description
- Formal Parameters:
 - IN as Begin's argument mapping

- OUT as End's argument mapping
- INOUT as both Begin and End's argument mapping
- Data Fields are mapped to FuegoBPM process instance variables. Types are mapped as follows:

WFMC Data Type	FuegoBPM Type
STRING	String
FLOAT	Real
INTEGER	Int
BOOLEAN	Bool
DATETIME	Time
REFERENCE	Any
PERFORMER	Any
NON-BASIC TYPES	Any

Activity mappings

The import process keeps the activity id, description, location and performer.


Route

Route type is normally mapped to the **Conditional activity** in FuegoBPM.

But if the **Route type** has an **AND Split transition restriction**, it is mapped to a FuegoBPM's **Split activity**.

And if it has an **AND join transition restriction**, it is mapped to FuegoBPM's **Join activity**.

Note

 FuegoBPM's model check might fail in some cases since FuegoBPM does not support having transitions going outside a split-join circuit from activities inside it.

Implementation

WfMC Implementation	FuegoBPM Activity
<i>Subflow</i>	Subprocess (SYNCHR) or Process Creation (ASYNCHR). Current parameters are mapped to an argument mapping (in or out)
<i>No</i>	Automatic activity
<i>Tool</i>	Automatic activity

Block Activity

Block type is mapped to **FuegoBPM's groups**. The associated activity set is created as a group in the FuegoBPM model.

Start Mode

WfMC Start Mode	FuegoBPM Activity
<i>Automatic</i>	The activity is mapped to FuegoBPM's Automatic activity unless one of the above conditions has been met.
<i>Manual</i>	The activity is mapped to FuegoBPM's Interactive activity unless one of the above conditions has been met.


Transition mappings

The transition properties are mapped to FuegoBPM's transitions properties as follows:

- From
- To
- Condition (in case of a Conditional Transition)
- Description
- Type (as follows) :

WfMC Transition Type	FuegoBPM Transition Type
CONDITION	Conditional
OTHERWISE	Unconditional
EXCEPTION	Exception
DEFAULTEXCEPTION	Unconditional

Note

 *WfMC Circular transitions* are not allowed in FuegoBPM model. They are ignored.

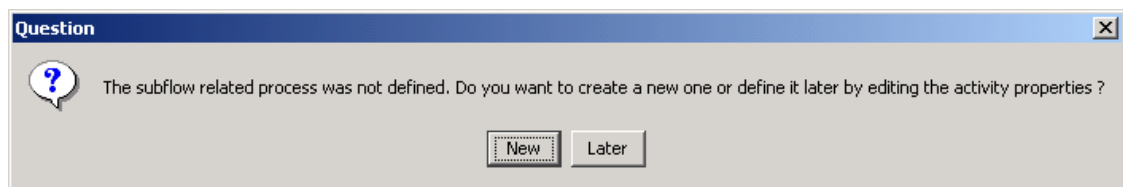
Mapping that are not yet implemented

- Applications
- Activity Tool implementation (usually invokes an application)
- Activity Simulation information
- Activity deadline

Generating automatically a Subprocess, Procedure and Screenflow

When you are designing your process, you can choose to define a Subprocess, a Procedure or a Screenflow to better match your design requirements.

When you choose one and if it does not exist, you can create a **New** one:

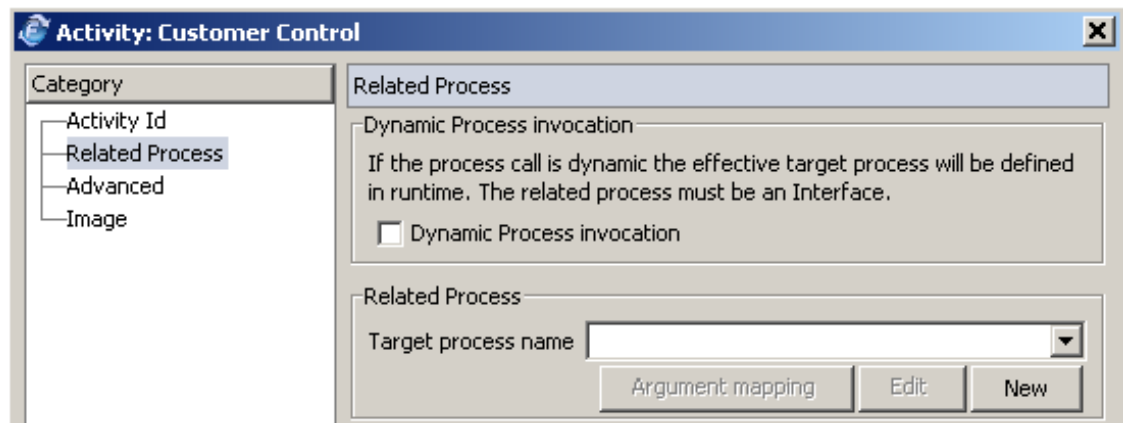


When creating them through the **New** option, FuegoBPM provides a wizard to facilitate the argument mapping, the instance variables generation in the called process, and to automatically link the calling and the called process.

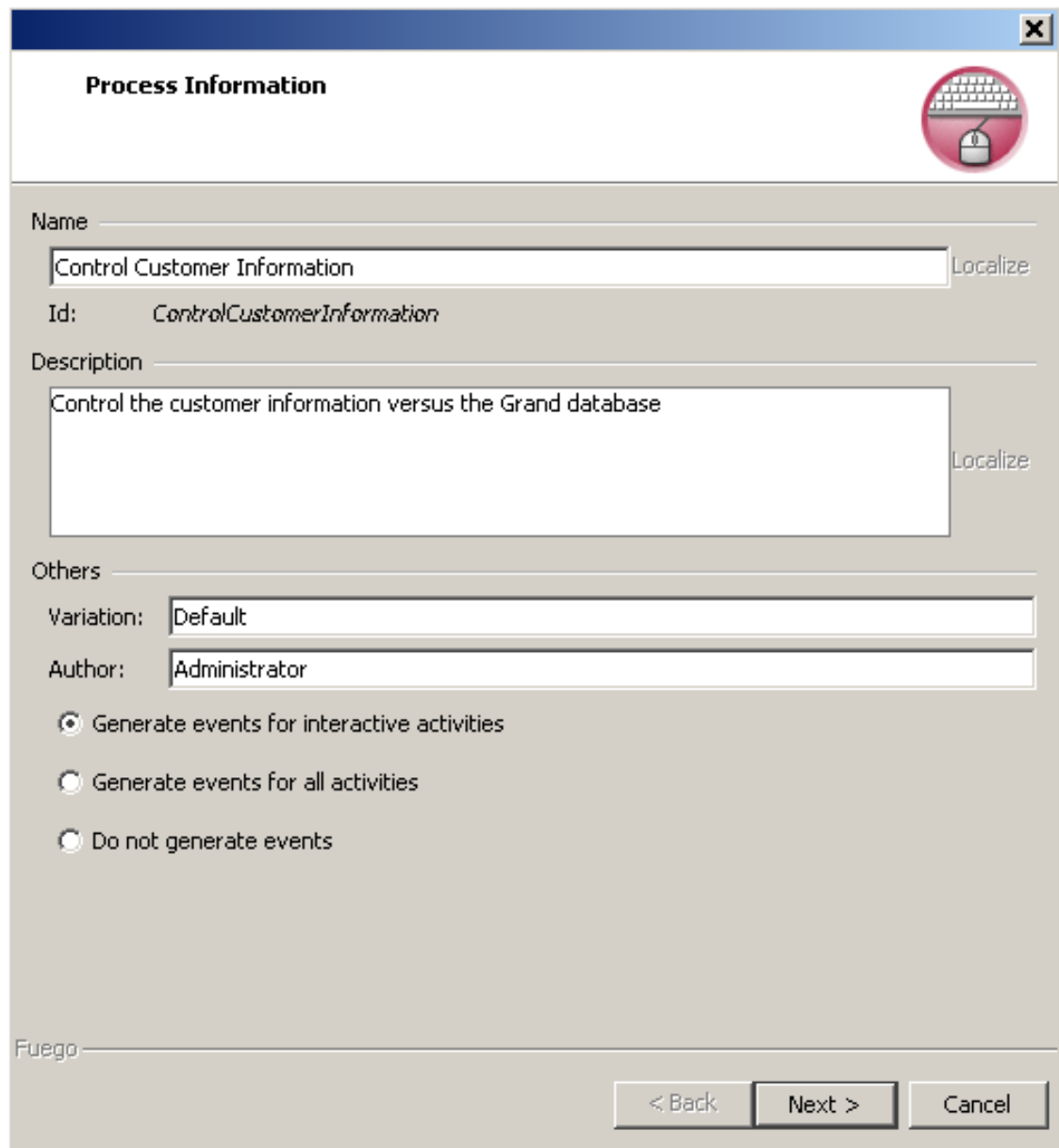
If you decide not to create a Subprocess, a Process Creation, a Procedure or a Screenflow at that moment, you can do it **Later** and still choose to create it by using the wizard. If you manually create any of them, you will have to manually implement the argument mapping and process relationships.

How can you easily create a Process?

Select the **New** option that populates or if you have decided to **Later** create it, you can select the **New** option as you match the **Related Process**.



The first step is to define the **Process Information**



The image shows a 'Process Information' dialog box with a blue title bar and a close button. It contains several sections: 'Name' with a text field containing 'Control Customer Information' and a 'Localize' button; 'Id' with a text field containing 'ControlCustomerInformation'; 'Description' with a text area containing 'Control the customer information versus the Grand database' and a 'Localize' button; 'Others' with 'Variation' (Default) and 'Author' (Administrator) text fields, and three radio buttons for event generation: 'Generate events for interactive activities' (selected), 'Generate events for all activities', and 'Do not generate events'. At the bottom is a 'Fuego' label and three buttons: '< Back', 'Next >', and 'Cancel'.

Process Information

Name: Localize

Id:

Description: Localize

Others:

Variation:

Author:

☒ Generate events for interactive activities

☐ Generate events for all activities

☐ Do not generate events

Fuego

< Back Next > Cancel

and then set the argument mapping. All instance variables and valid predefined variables are listed for you to choose them as **In** arguments or **Out** arguments.

Select the instance variables to use

Choose from this list the variables that will be used in the corresponding mappings as in and out arguments.

Instance variable	in	Out
children	<input type="checkbox"/>	<input type="checkbox"/>
creation	<input type="checkbox"/>	<input type="checkbox"/>
currentException	<input type="checkbox"/>	<input type="checkbox"/>
customerID	<input checked="" type="checkbox"/>	<input type="checkbox"/>
customerName	<input checked="" type="checkbox"/>	<input type="checkbox"/>
customerVIP	<input type="checkbox"/>	<input checked="" type="checkbox"/>
deadline	<input type="checkbox"/>	<input type="checkbox"/>
description	<input type="checkbox"/>	<input type="checkbox"/>
id	<input type="checkbox"/>	<input type="checkbox"/>
notes	<input type="checkbox"/>	<input type="checkbox"/>
organization	<input type="checkbox"/>	<input type="checkbox"/>

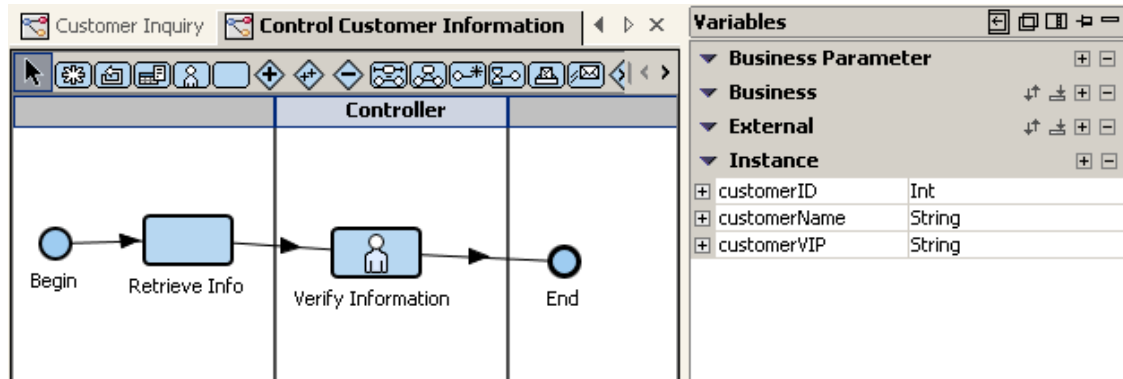
Fuego

< Back Next > Cancel

The **In** arguments (*CustomerID* and *CustomerName*) become *in* arguments in the corresponding Begin activity of the called process, as well as the **Out** arguments (*CustomerVIP*) become the arguments sent by the End activity.

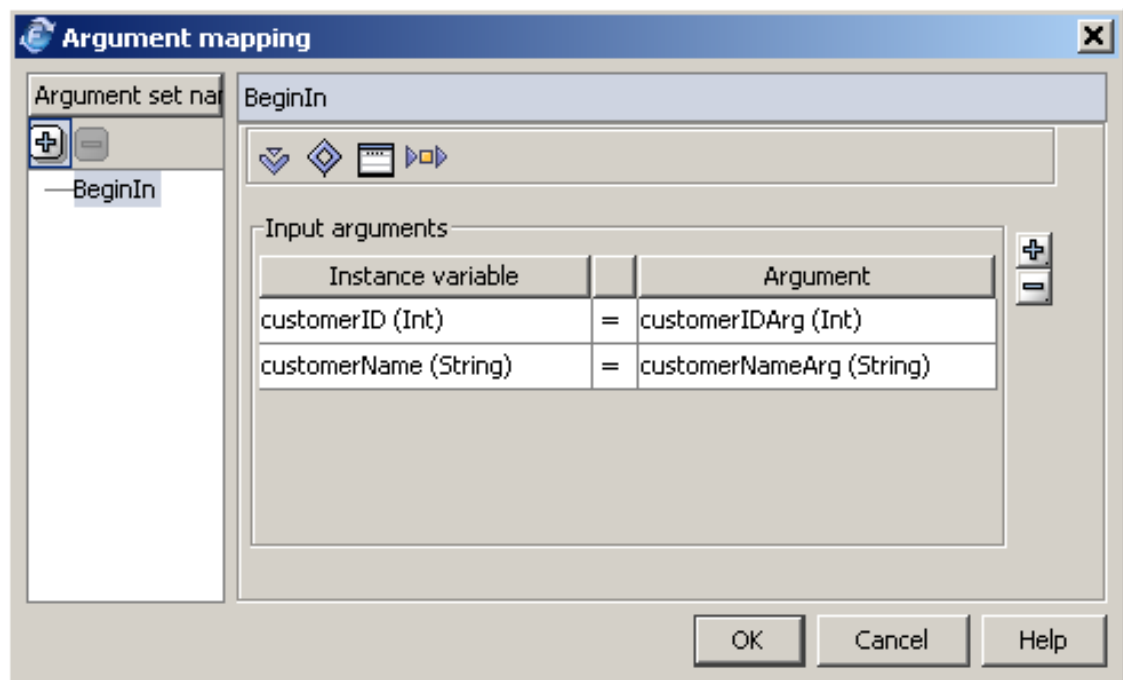
If you implement the Argument mapping at this time, then:

- The **instance variables** are automatically generated in the **called process** (subflow, procedure or screenflow). All the instance variables from the **calling process** used in the argument mapping, are created in the **called process** to receive the arguments in the Begin Activity, or return them from the End Activity.

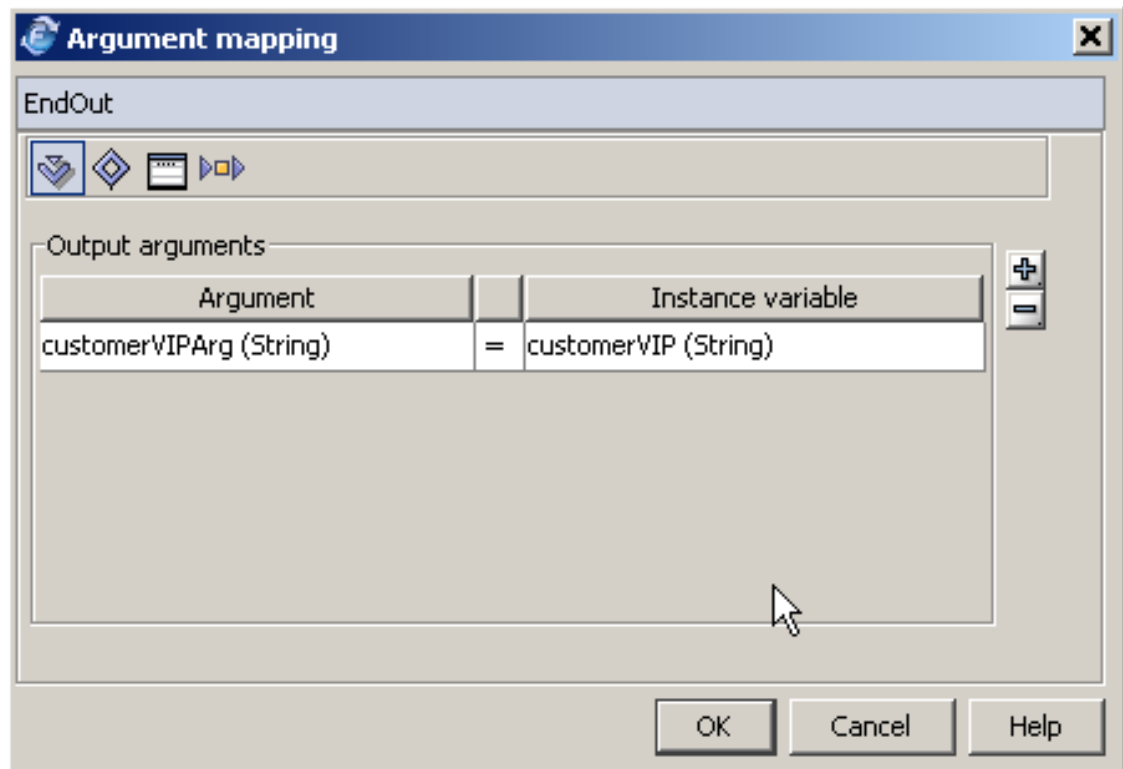


- All **in** or **out arguments** in the **called process** are automatically generated with the name of the instance variable ending with *Arg*.
- The Argument mapping is set.

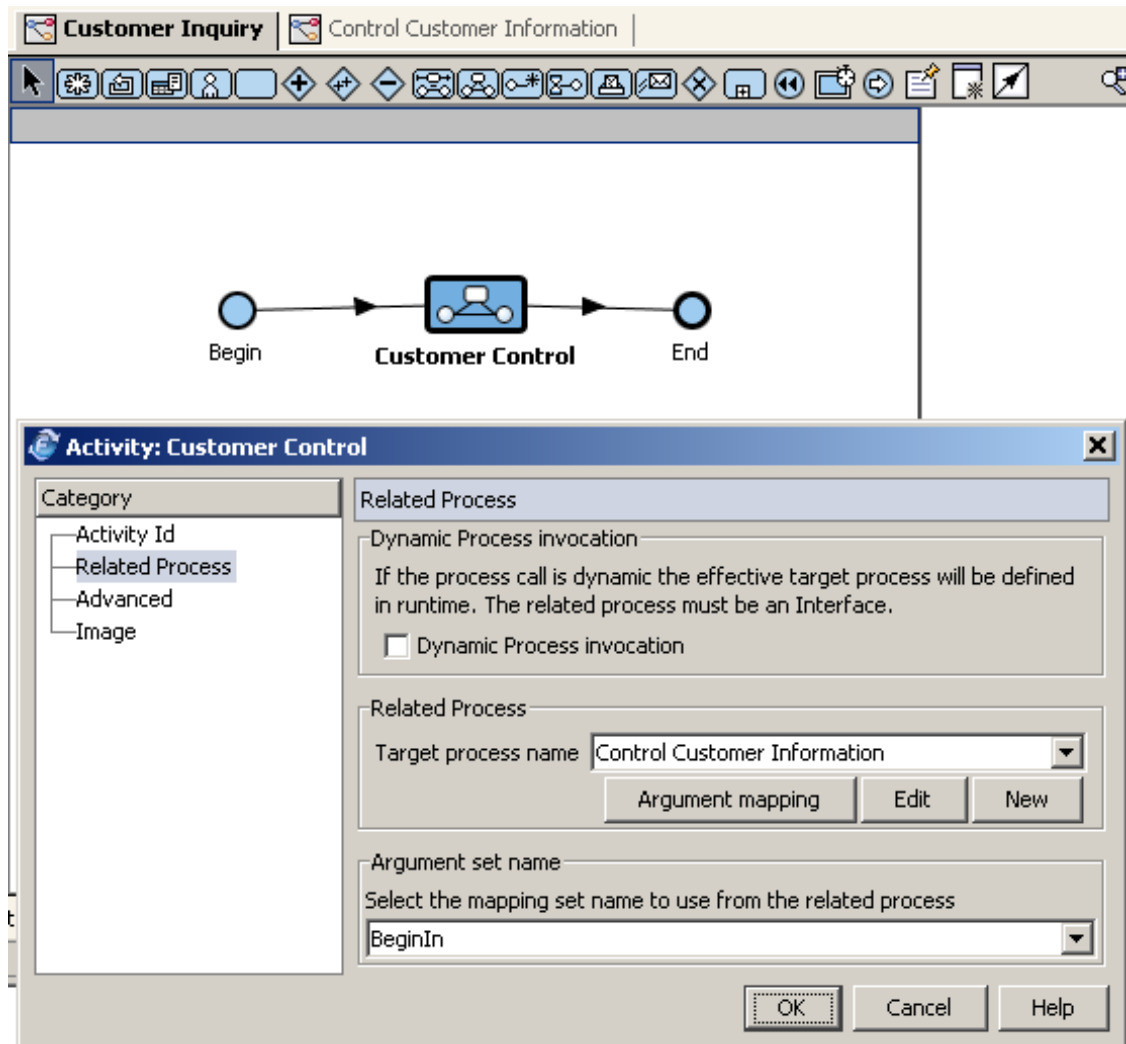
Begin Argument Mapping



End Argument Mapping



- The Related Process is set.



Process Creation and Termination Wait

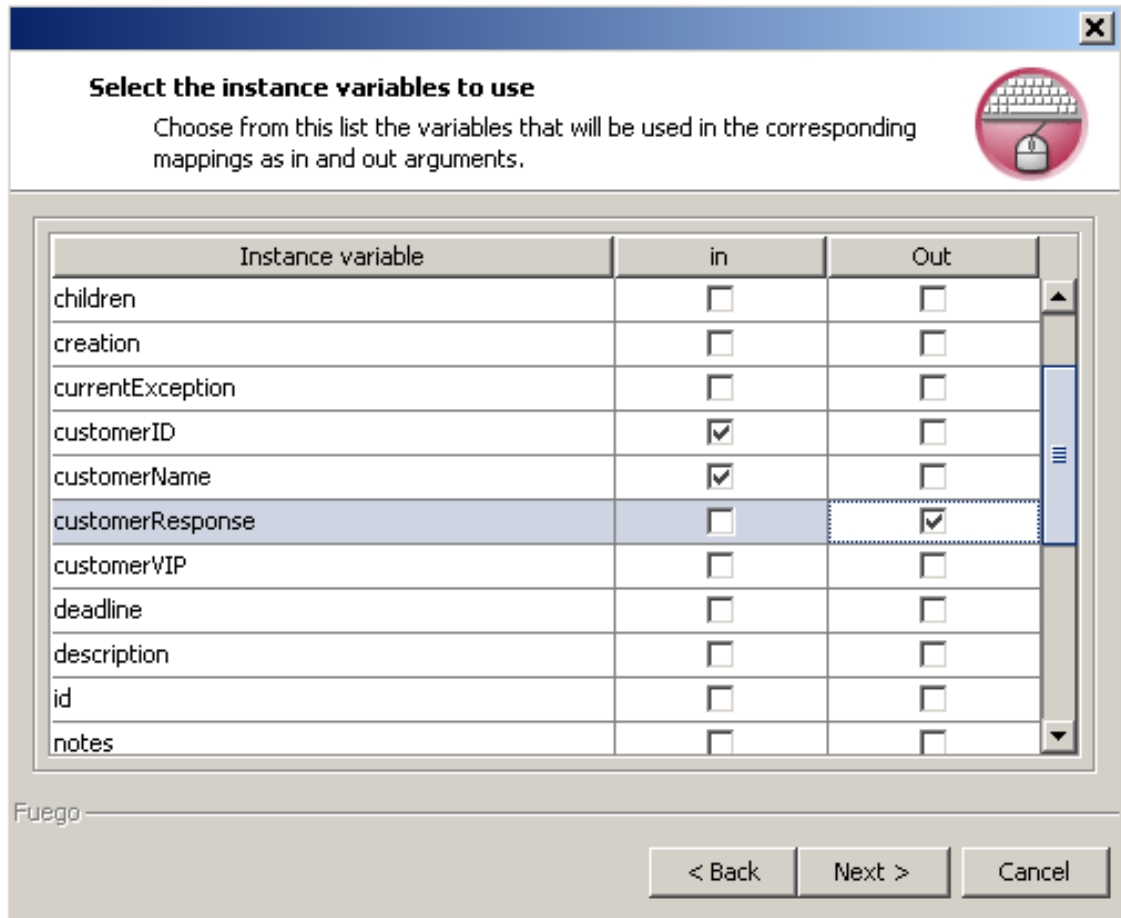
If you are implementing a Process Creation and creating the related process using the wizard, the arguments and instance variables are created in the **called process**.

If the **called process** returns arguments, the **Termination Wait** activity receives them. Therefore, you have to **manually match the returned arguments** (defined as the out arguments when creating the Process Creation), to the **corresponding instance variables of the calling process**.

In the above example, we add an asynchronic process to submit the

inquiry to other departments within the company.

As we create the **Process Creation** activity, we also generate the new process **Resolve Inquiry** and set the argument mapping as:



Select the instance variables to use

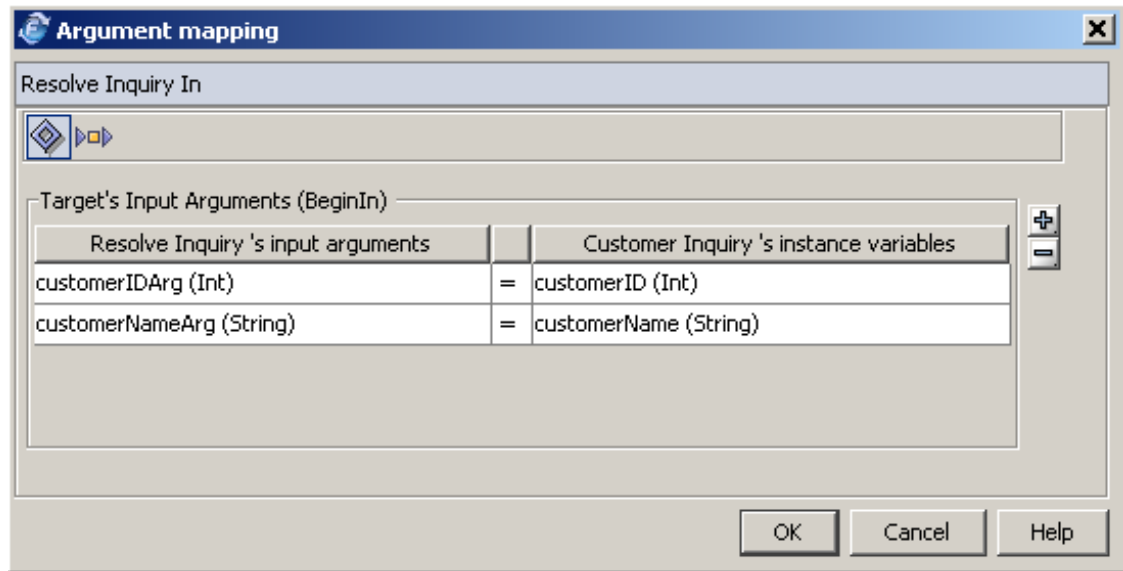
Choose from this list the variables that will be used in the corresponding mappings as in and out arguments.

Instance variable	in	Out
children	<input type="checkbox"/>	<input type="checkbox"/>
creation	<input type="checkbox"/>	<input type="checkbox"/>
currentException	<input type="checkbox"/>	<input type="checkbox"/>
customerID	<input checked="" type="checkbox"/>	<input type="checkbox"/>
customerName	<input checked="" type="checkbox"/>	<input type="checkbox"/>
customerResponse	<input type="checkbox"/>	<input checked="" type="checkbox"/>
customerVIP	<input type="checkbox"/>	<input type="checkbox"/>
deadline	<input type="checkbox"/>	<input type="checkbox"/>
description	<input type="checkbox"/>	<input type="checkbox"/>
id	<input type="checkbox"/>	<input type="checkbox"/>
notes	<input type="checkbox"/>	<input type="checkbox"/>

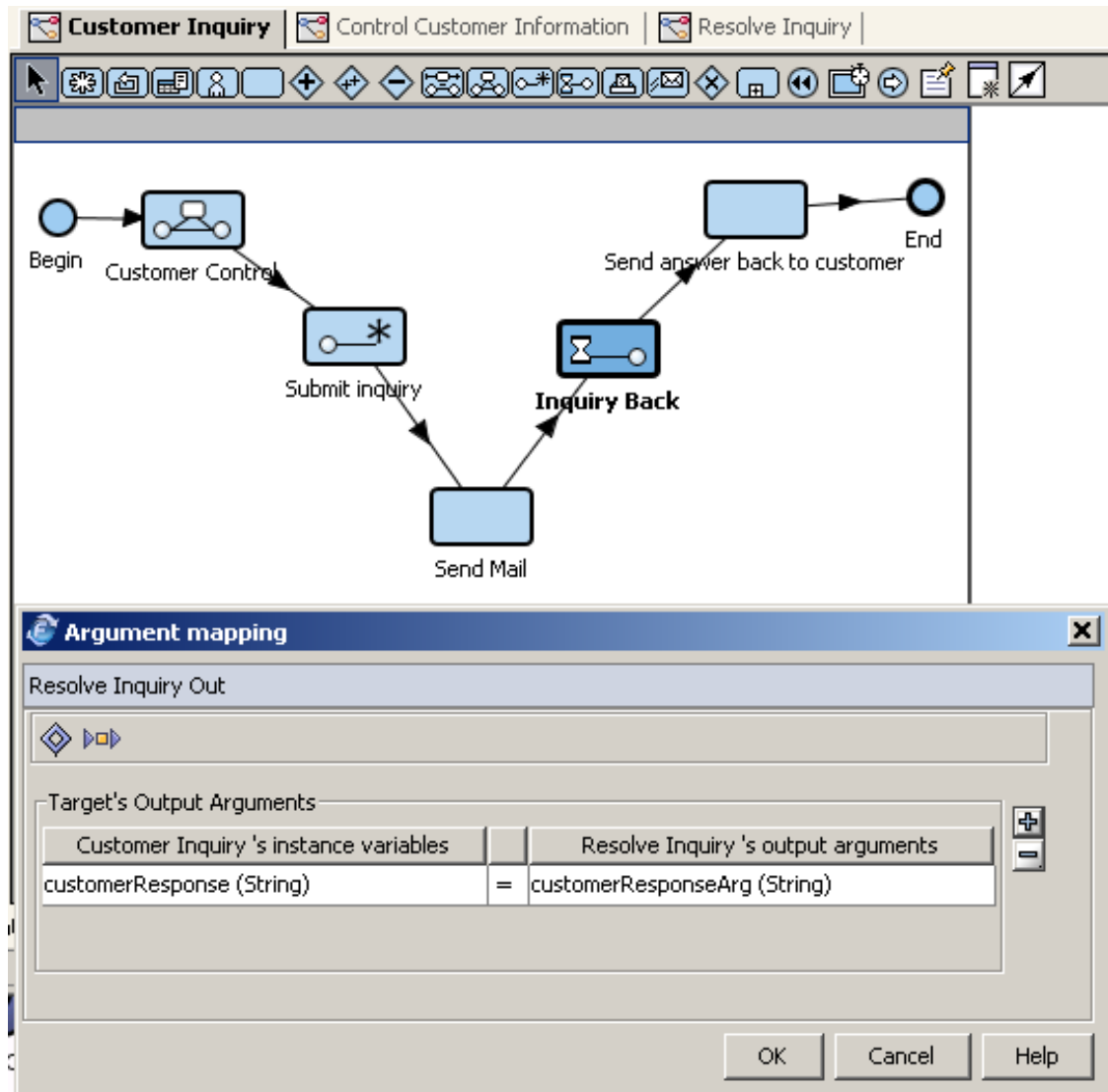
Fuego

< Back Next > Cancel


So, if you later see the **Process Creation** argument mapping, you get:



But you need to manually define the **instance variable** that is going to receive the **out argument** of the called process ("customerResponse**Arg**").



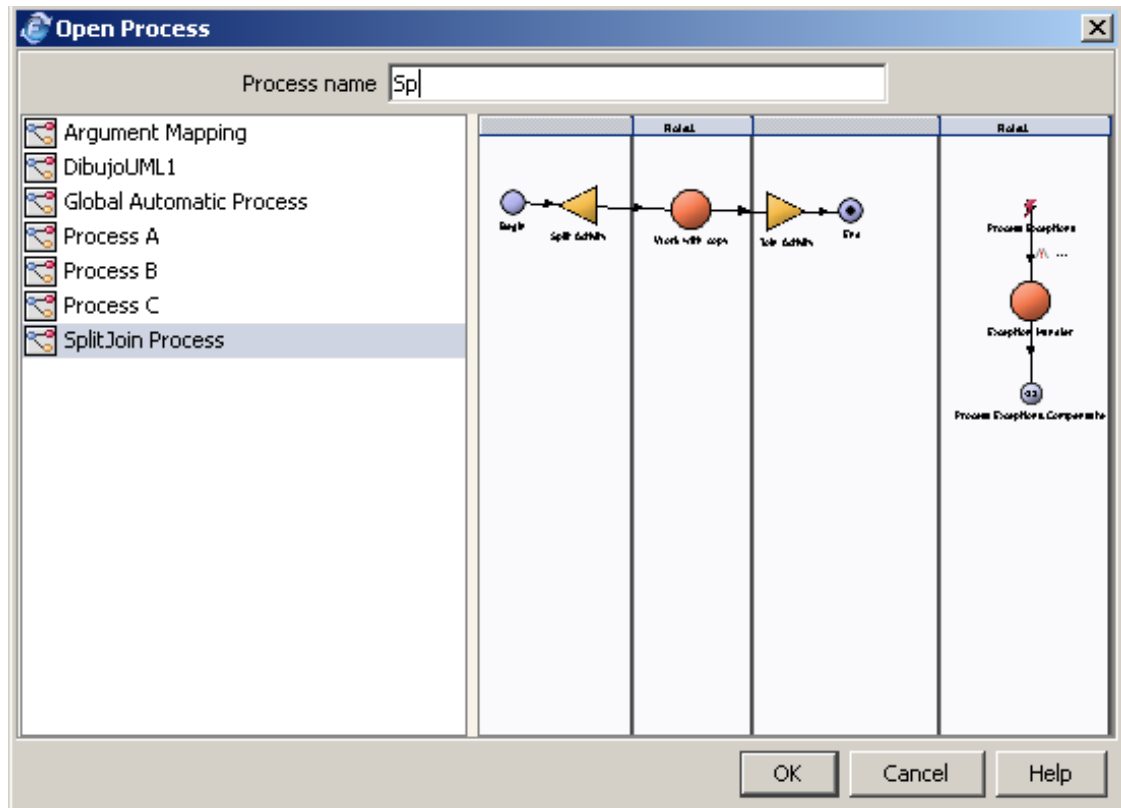
Note

 If you don't map the returned arguments in the Termination Wait, you will not have this information available in the calling process.

Searching for a process, procedure or screenflow

Within your project you may have a large number of processes, procedures or screenflows. You can search for one of them by name using an **incremental search**.

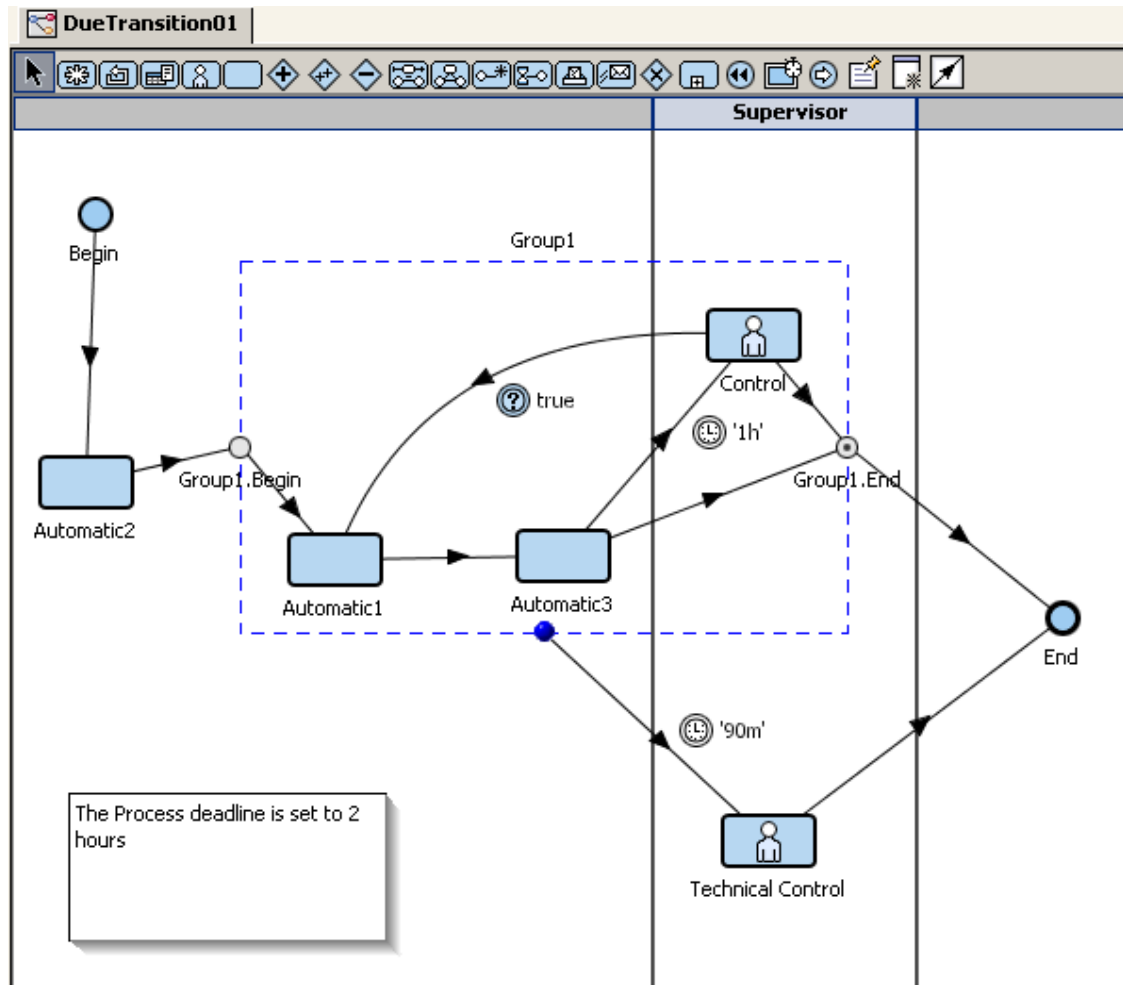
Right click on the **Processes** option in the project tree and select **Open**.



Type the **Process name**. The first process that matches, as you type the name, is displayed and highlighted.

Notes within a Process

While designing, you might need to *attach* short **notes** to the activities or to the process. These notes help you remember or make remarks on anything special about the activity or process.



Creating Notes

To add a **Note**, click the *note icon*  in the Designer toolbar, move the cursor to the desired target location and click again.

If the note is to be attached to an activity, drop it near the activity. Enter the text for the note, click off the note and then right-click on the Note. Select **Point to** and select the activity that the note will be attached to.

If you do not select an activity, the note will remain as a *process note*.

To edit the note, double-click it.

Optionally, you can write the note in different languages. Right-click on the note and select **Localize** to enter the note in alternate languages. See *Internationalization* for further information on available languages.

To **delete** the note, right-click and select the corresponding option.

Viewing Notes

To hide the **notes**, select the **Activities** from the **View** menu and disable the *Show Note* option.

If you print the process design, the notes will be printed as well.

Generate Events

You can keep track of the events that happen while an instance is flowing through the process. The events represent each time the instance entered or exited an activity (simple activity, group, procedure). The server generates one event per action.

When are events stored?

There are four levels that can define when events are stored:


1. Defined per **Server**.
2. Defined per **Process**.
3. Defined per **Group**.
4. Defined per **Activity**.

Per Server

The options for the server to store events are the following:

- Depends on the process, the server will store events only if the **generates events** property is enabled in the process.
- Always stores events, no matter whether the **generates events** property is enabled in the process or not.
- Never stores, even if the **generates events** property is enabled in the process.

Note

 These options are available in **FuegoBPM Enterprise** and can be set through the Web Console. In **FuegoBPM Studio** it always depends on the process properties


Per Process

- If the Server stores events (option 1 or 2, or FuegoBPM Studio) and if any of the **generates events** properties are enabled in the process.

The Process can define to generate events for:

- All activities, or for
- Interactive activities only.

Note

 Although the events generation can be disabled for a process, you can enable it for a specific group or activity. See below.

Per Group

- If the Server stores events (option 1 or 2, or FuegoBPM Studio),

you can define to:

- keep the **Default** definition. The Default option always refers to choose the immediate superior level *generate event* definition. In this case what was defined at the process level or if it is an inner group, what was defined for the immediate outer group.
- generate events, although, at a process level the event generation is disabled; or
- do not generate events.

Per Activity

- If the Server stores events (option 1 or 2, or FuegoBPM Studio), you can define to:
 - keep the **Default** definition (for interactive activities only, for all activities or no events generation). The Default option always refers to choose the immediate superior level *generate event* definition. The activity will generate events depending on the group to which it belongs to (if applicable) or the process event generation definition; or
 - generate events, although, at a process or group level the event generation is disabled; or
 - do not generate events.

What events are generated?

- All the activities generate the same events (IN, OUT, EXECUTE,

SELECT, UNSELECT, among others.)

- The Begin activity has no Activity IN event as the instance is created in it.
- The End activity has no Activity OUT event as the instance terminates there.
- The *Join* activity generates events only if the *Split* associated activity generates events.
- When an instance is created, a CREATION event is generated instead of an Enter event. This event is always automatically generated if the Server stores events. All original instances (copy 0) have the CREATION event.
- When an instance is finished, an END event is generated. This event is always automatically generated if the Server stores events. All terminated original instances (copy 0) have the END event.
- Interactive activities have additional events that occur between the Enter and End events.

By default, the option 'Generate Events' is *on* in Interactive activities and *off* in Automatic activities.

If you have any **Generates events** check box selected, the Audit Trail in Work Portal is enabled. The Audit Trail displays all events that have occurred for an instance.


Audit Trail - Work Portal

Audit Trail lets you see all past activities associated to a particular process instance. This includes events, activities, the participants (users) who processed a particular event, date and time for each event and all copies of events, and the process image showing the flow path of the instance through the process and the actual location

of the instance.








The audit trail information might not be available because process designers determine whether it is necessary for each business process to generate the audit information.

To view the audit trail for an instance

1. In Work Portal, click the appropriate view listed in the left column. The instances associated to that view are listed in the instances panel.
2. Click the description of the appropriate instance.
3. Click the **Audit Trail** button  at the top-right corner of the Work Portal. The *Audit Trail* window displays in a separate browser window.

Audit trail
Help


[Marine Supply Order Fill](#) > [Check Freight](#) > [Diving Supply OrderFill5](#)

Activity	Event	Responsible	Date	Copy
 Create Order 	Completed		Feb 22, 2005 3:18:56 PM	0
 Review Order 	Completed		Feb 22, 2005 3:18:56 PM	0
 Check Freight 	Processing		Feb 22, 2005 4:43:06 PM	0
	Enter	Shipping Clerk	Feb 22, 2005 4:43:06 PM	0
	 Task Check Freight executed	John Smith	Feb 22, 2005 4:43:37 PM	0

▼ Process Image

Close

FuegoBPM™ - Work Portal

4. Click the plus sign  next to an activity to view further detailed information on that activity.

Audit trail
Help

Marine Supply Order Fill > Check Freight > Diving Supply OrderFill5

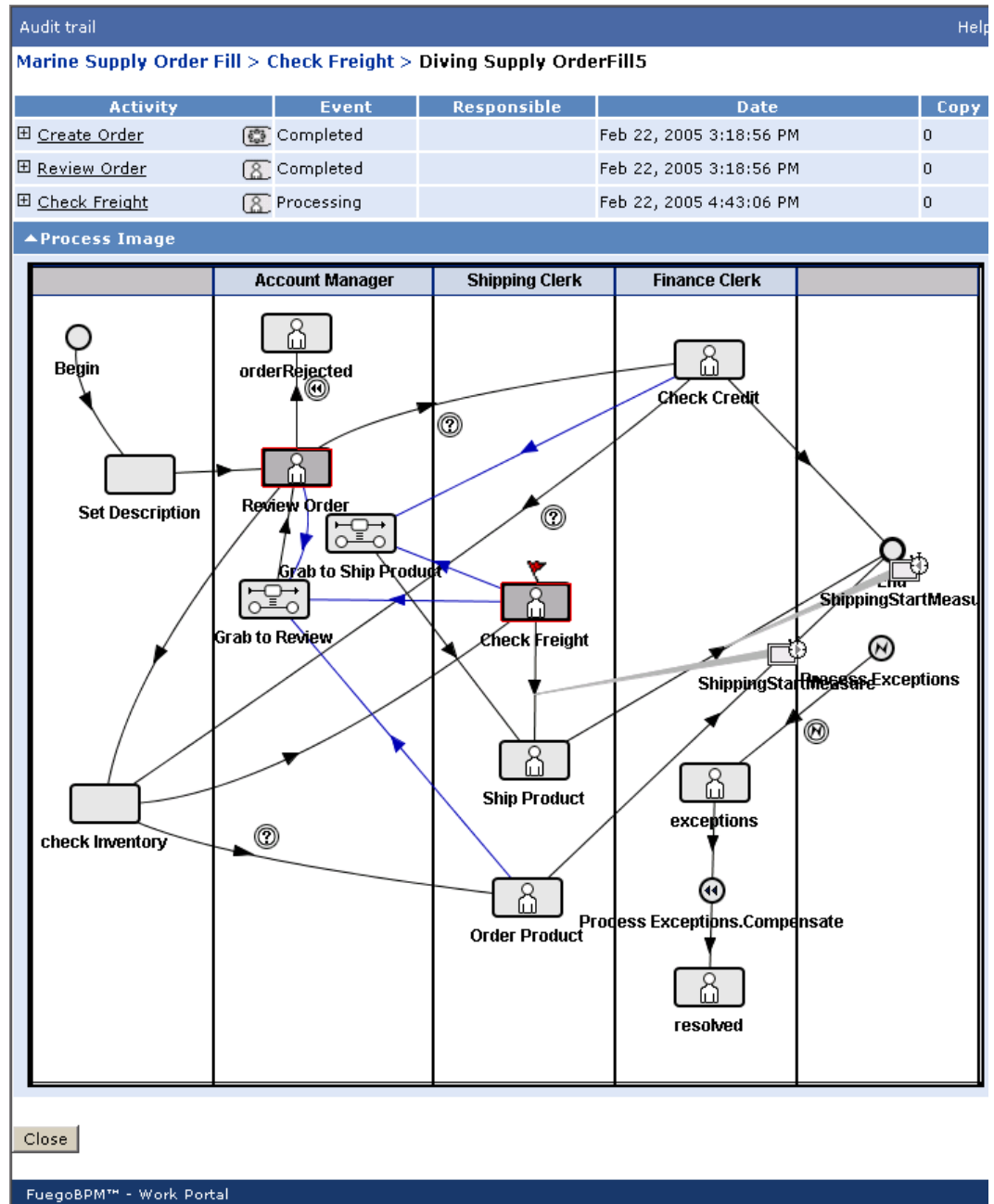
Activity	Event	Responsible	Date	Copy
<input type="checkbox"/> <u>Create Order</u>	Completed		Feb 22, 2005 3:18:56 PM	0
	Creation	John Smith	Feb 22, 2005 3:18:56 PM	0
<input type="checkbox"/> <u>Review Order</u>	Completed		Feb 22, 2005 3:18:56 PM	0
	Enter	Account Manager	Feb 22, 2005 3:18:56 PM	0
	✓ Task Review Order executed	John Smith	Feb 22, 2005 4:43:02 PM	0
	Exit	John Smith	Feb 22, 2005 4:43:05 PM	0
<input type="checkbox"/> <u>Check Freight</u>	Processing		Feb 22, 2005 4:43:06 PM	0
	Enter	Shipping Clerk	Feb 22, 2005 4:43:06 PM	0
	✓ Task Check Freight executed	John Smith	Feb 22, 2005 4:43:37 PM	0

▼ Process Image

Close

FuegoBPM™ - Work Portal

- Audit trail also includes the Process Image, which shows all past activities and transitions associated to the instance. Activities included in the instance path are outlined in a different color. A red flag indicates in which activity the instance is in that moment.




6. Measurement Mark activities are included in the Audit trail as shown in the picture below.

Audit trail					Help
Marine Supply Order Fill > End > Diving Supply OrderFill5					
Activity	Event	Responsible	Date	Copy	
[-] Create Order	Completed		Feb 22, 2005 3:18:56 PM	0	
[-] Review Order	Completed		Feb 22, 2005 3:18:56 PM	0	
[-] Check Freight	Completed		Feb 22, 2005 4:43:06 PM	0	
[-] ShippingStartMeasure	Completed		Feb 22, 2005 4:50:48 PM	0	
[-] Measurement Start	Measurement Start	John Smith	Feb 22, 2005 4:50:48 PM	0	
	• orderAmount = 23000.00		Feb 22, 2005 4:50:48 PM	0	
[-] Ship Product	Completed		Feb 22, 2005 4:50:48 PM	0	
[-] ShippingStopMeasurement	Completed		Feb 22, 2005 4:51:13 PM	0	
[-] Measurement Stop	Measurement Stop	John Smith	Feb 22, 2005 4:51:13 PM	0	
	• Elapsed Time = 25s		Feb 22, 2005 4:51:13 PM	0	
	• orderAmount = 23000.00		Feb 22, 2005 4:51:13 PM	0	
[-] End	Completed		Feb 22, 2005 4:51:13 PM	0	
▼ Process Image					
<div>Close</div> <div>Measurement Mark started</div> <div>Measurement Mark stopped</div>					
FuegoBPM™ - Work Portal					

In this example, the measurement marks for collecting Key Performance Indicator data (KPI) about the shipping, also store the business variable *orderAmount*. The business variables are shown both, in the measure start and stop, and the elapsed time is shown in the measurement mark stop. Whether you would see, measure starts separately from the stop would depend on the business process design.





- Click **Close** to exit the **Audit Trail** window and return to the Work Portal.

Audit Trail Information Columns - Work Portal

When you click on the **Audit Trail** button , the **Audit trail** window appears and detailed information is provided for the instance.

Audit trail Help

Marine Supply Order Fill > Check Freight > Diving Supply OrderFill5

Activity	Event	Responsible	Date	Copy
 Create Order	Completed		Feb 22, 2005 3:18:56 PM	0
 Review Order	Completed		Feb 22, 2005 3:18:56 PM	0
 Check Freight	Processing		Feb 22, 2005 4:43:06 PM	0
	Enter	Shipping Clerk	Feb 22, 2005 4:43:06 PM	0
	 Task Check Freight executed	John Smith	Feb 22, 2005 4:43:37 PM	0

▼ Process Image

Close

FuegoBPM™ - Work Portal

Audit Trail Info Column Descriptions

Column Name	Description
Activity	Displays each activity, in order, where each event took place.
Event	Displays the event associated with each activity on the left. An Enter and an Exit event is associated to every activity. Note: The Begin activity will have a Creation instead of an Enter event. Interactive activities will have additional events that occur between the Enter and Exit events.
Responsible	Displays the person or role that processed or selected the activity for the associated instance.
Date	Displays the completion date and time for the associated event.
Copy	Displays the number of copy where the associated event occurred. If several copies of the instance were

Column Name	Description
	created as the result of a split/re-join process, all events for each copy are displayed. Each event has a different copy number.

Note

In some processes, Audit trail functionality may not be provided. Check with your system administrator.

Roles within a Process

A role defines a job function for work being done in a process. Roles are displayed in the Process Design Area as "swim lanes." Roles act as handlers for different types of activities. Process Designer supports two types of roles: automatic and user-defined.

Automatic roles have no heading name because no end user is required to run the activities in automatic roles. User-defined roles have a heading name defining the role of the end user fulfilling the activities in these roles. See Roles for further information.

End users are assigned to user-defined roles while defining the Organizational Resources. See Creating a Participant for further information.

The display of activities within roles helps you visualize the role's responsibilities and sequence in the overall process. You can move the role lanes by left-clicking on the role lane heading and dragging the role lane to its new position. When the role lane is where you want it to reside, release the left mouse button.

You can also view the roles horizontally or vertically by selecting the appropriate option from the **View** menu options.

Automatic roles

Whenever an activity requires no end user intervention, the activity is added to an automatic role lane. Automatic role labels are not displayed in Process Design Area.


The types of activities which have no user intervention are: Begin, End, Split and Join. Automatic, Global Automatic, Process Creation, Notification Wait and Process Notification. These activity types generally reside in automatic role lanes.

The Process Designer automatically creates two automatic roles and two activities, Begin and End, when you create a process. Begin is displayed in the first automatic role and End is displayed in the second automatic role. You add automatic and user-defined roles between these two lanes to hold the activities required in your process design.

Create an automatic role

Create an automatic role lane when you need to add activities that require no end user intervention.

To create a new automatic role

1. Right-click on any role lane and select **Add role**, or drag the role icon  and drop it on the correct lane so the new role can be inserted.
2. The **Role** dialog box appears. Select **Automatic Handler** from the pop-down menu.
3. A new role lane appears with no heading at the top.

User-defined roles


When user intervention is required for an activity, you add the activity to a user-defined role lane. User-defined role labels are displayed in Process Designer with a title in the row lane header.

These roles are the ones created as part of the organizational resources within the company. See Roles for further information.

Types of activities that require user intervention are Interactive, Global Creation, Grab and Global.

Create a role from an existing process

To add an existing role from another process within the project:

1. Right-click on any role lane and select **Add role**, or drag the role icon  and drop it on the correct lane so the new role can be inserted.
2. The **Role** dialog box appears. You can create a new role or select an existing one. The listed roles are those from all processes within the project.
3. A new role lane appears with the existing role name heading.

Parametric roles

User-defined roles can also be Parametric. Parametric roles are those roles in which two or more sets of employees complete essentially the same work except for minor differences. Examples of employees in Parametric roles are shift workers who perform the same work on different working schedules.

A process instance variable can be implemented and set through a distribution algorithm so that the balancing and distribution is achieved based on some programmatic business rules.

See Roles for further information.


Hiding roles

Hiding roles allow you to reduce visual complexity and to ease process navigation when you design a process. Hidden roles allow users to view roles that are physically far apart from each other because all other roles apart from the roles of interest are not displayed.

To hide and un-hide existing roles

1. Right-click on the role name you wish to hide.
2. Select **Hide** from the shortcut menu. To hide multiple roles, repeat steps 1 and 2. The role(s) and associated swim lane(s) are hidden. A bi-directional arrow appears when you hold your mouse over the hidden roles, indicating that the role(s) are hidden.
3. To un-hide roles, click on the bi-directional arrow.

Tip

 If you click on the middle of an existing role (or drag & drop the toolbar role icon) FuegoBPM is going to analyze whether you have activities after and before this position. Depending on the above, it will insert the role before or after the existing role. If necessary, it will split the existing role into two and add the new role in the middle.

Process as Web Services

Overview

FuegoBPM allows you to expose portions of your business process as a web service in Web Service Definition Language (WSDL). The WSDL can be shared with external partners and organizations as appropriate allowing web service aware applications to interact with processes deployed in a FuegoBPM Server.

Process Web Services can provide several operations. Two types of operations are defined by Process Designers:

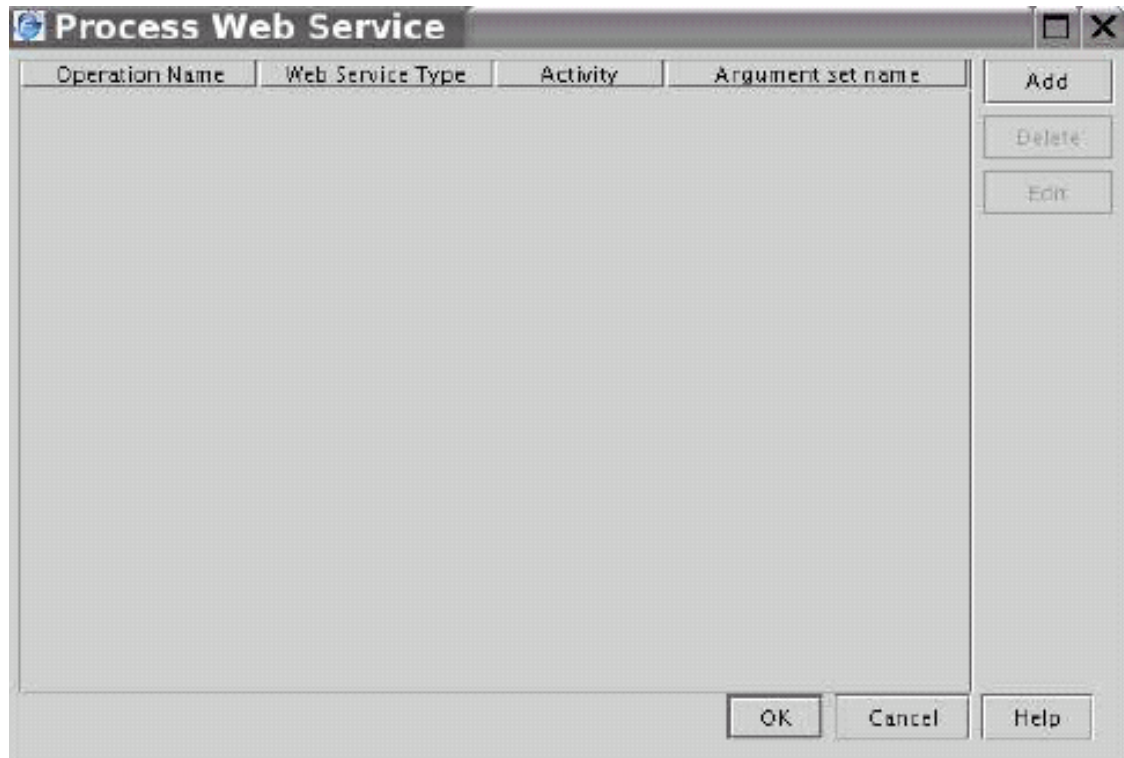
1. *Process Creation*: Allows creation of new instances in the process. Process creation is the ability to create an instance in a process externally (from outside the process). External partners or organizations can create instances in your process by leveraging the WSDL file with their own existing Web applications or Web services.
2. *Process Notification*: Allows applications to send notifications to the process. Instances waiting in Notification Wait activities can be notified by external Web applications or Web services via the WSDL file.

Additionally, when any of the previous types are defined for a process, the following operations are also included in the corresponding web service definition:

1. *Get Instance Status*: Allows applications to find out the status of a given instance.
2. *Start Session*: Applications that want to execute the exposed operations must authenticate first. This operation must be called to start a session before other operations can be executed.
3. *Discard Session*: Applications should call this operation when the session is not needed anymore.

Defining Operations

To define the operations open the **Process Web Service** dialog by right-clicking the process on the *Project Tree* or open it from the **Process Menu**.

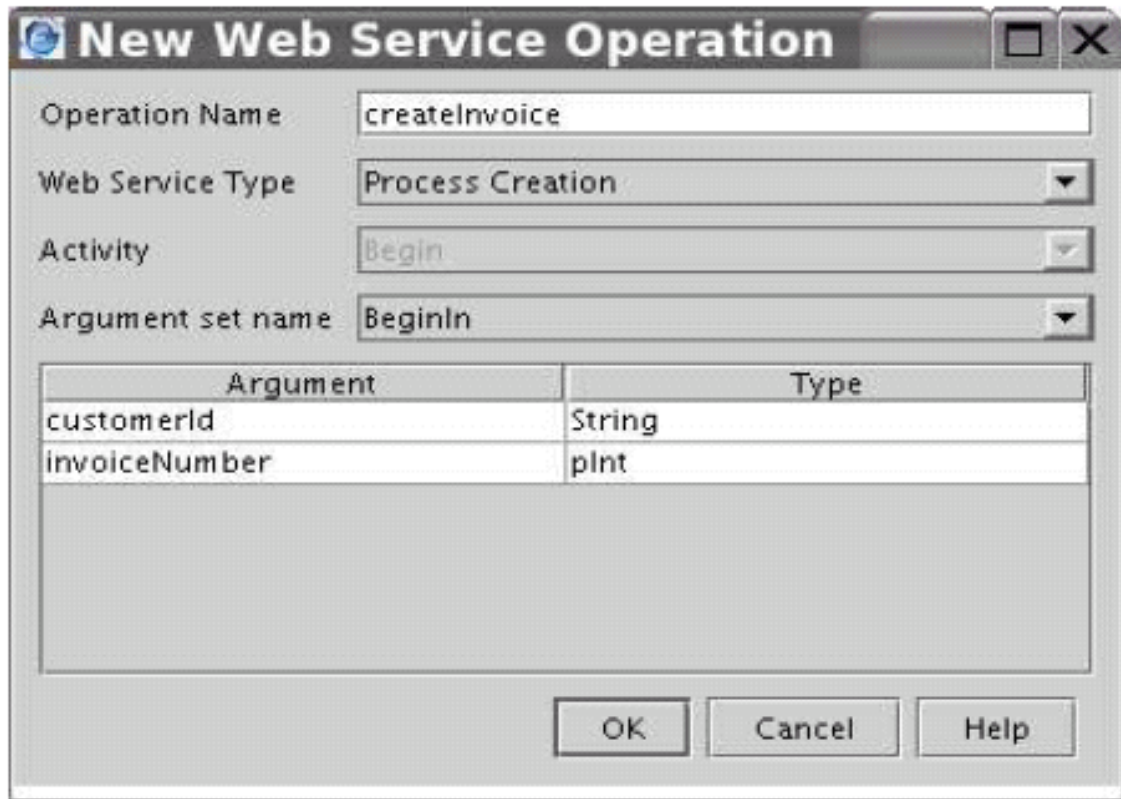


The dialog shows the *Process Creation* and *Notification* operations defined for the process. From here you can **Add**, **Delete** or **Edit** operation definitions.

The following operation information is displayed:

- **Operation Name:** The name given to the operation. For example: *createOrder*.
- **Web Service Type:** The operation type (Process Creation or Process Notification).
- **Activity:** For *Notification* operations this column displays the activity that will receive the notification. Note that *Wait Activities* receiving notifications in this way must be marked as *Receiving External Events*.
- **Argument set name:** The argument set required for the operation.

To add an operation simply click on the **Add** button. A new dialog appears:



Argument	Type
customerId	String
invoiceNumber	plnt

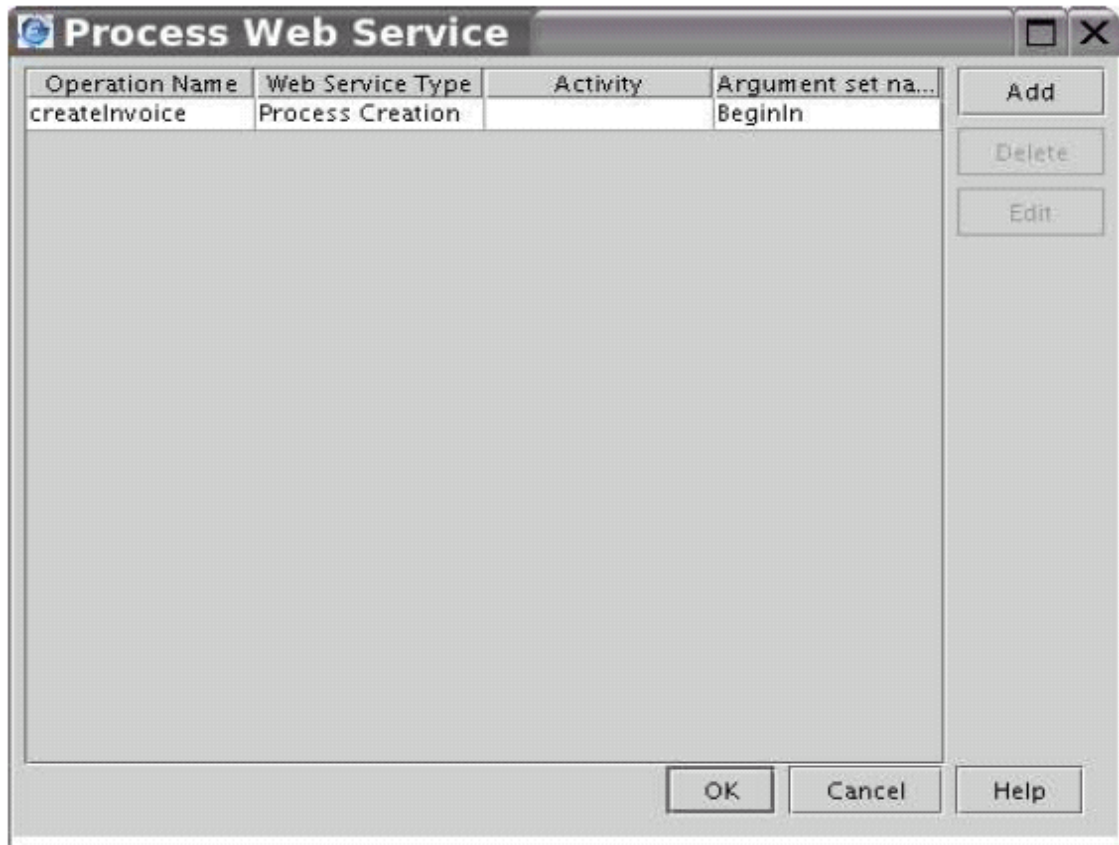
As an example we are creating here an operation of type *Process Creation* called *createInvoice*.

To do it we fill the **Operation Name** field with the *createInvoice* text and select the **Process Creation** option in the **Web Service Type** dialog.

The Argument Set Name combo lets you select any of the available argument mappings in the *Begin* (for Process Creation) or *Wait* (for Process Notification) activity. This allows you to define more than one operation of the same type, each one receiving different arguments.

The table below the **Argument Set Name** combo shows the arguments that the operation will receive.

After creating the operation the **Process Web Service** dialog looks as follows:



Process Notification operations are defined in the same way.

URL to access a process defined as a Web Service

When using FuegoBPM Studio Server, to access any process defined as a Web Service, the URL is:
`http://localhost:9000/fuegoServices/ws`

When using FuegoBPM Enterprise Server, the Port is defined in the Web Console as a Server-Services property called *Web Services Listener Port*. Therefore, instead of the default port *9000* used in the Studio, the URL is composed by this redefined port.

When using FuegoBPM J2EE Server, the port number is configured at the J2EE application server. Normally, the port number is the same port used for the Work Portal. For example:
<http://localhost:9080/fuegoServices/ws>

Chapter 4. Activities

Activities

Activities define a manual or automated task that conforms one step within a process design. Adding a new activity allows you to create a new step and assign it to a role in a process. A manual activity requires end user intervention whereas an automatic activity can be automatically completed by the FuegoBPM Server. An activity can include one or more tasks.

Examples of activities in an Order Management process might be *Create Order, Check Inventory, Get Shipping Route, Check Customer Credit, Pick Product, Pack Product, Create Billing List, Create Invoice, Print Invoice* and *Ship Product*.

Based on the Business Service Orchestration, FuegoBPM's activities can be categorized as activities that have

1. **Human Interaction (people)**
2. **System Interaction**
3. **Organizations Interaction**
4. **Processes (Initiating a Process and Process Controls.)**

Additionally, some activities are defined as "**Global activities**" which are part of the process, but are not executed, based on an instance.

Naming activities



FuegoBPM convention suggests that you name your activity with a verb followed by a direct object specifying the activity's role in the process. For example, *Create Order, Ship Product, Check Credit* are all valid activity names.

Once the activity name has been set, it cannot be changed. But you can change the activity label (displayable name.)

The following tables list and describe activities used in Process Designer.











Initiating a Process

These activities are automatically generated and they define the scope of the process.

Classic / UML / Business Analyst / BPMN & Color	Activity Name	Definition
	Begin	Provides an entry point into every process and is created with new process creation. Every process contains a Begin activity.
	End	Provides an exit point from every process. Always the last activity in a process. Created with new process creation. Every process contains an End activity.






Human Interaction

These activities are the ones that allow people to interact with the process.

Classic / UML / Business Analyst / BPMN & Color	Activity Name	Definition
 /  /  /  / 	Interactive	Directly brings end users into a process by displaying information to or by requesting information from the end user. This information is used later to set process instance variables.
 /  /  /  / 	Grab	Provides process supervisors with flexibility to deal with exceptional conditions and redistribute instances accordingly. Used to easily move instances from one activity's queue to another activity's queue to prevent a backlog of work or to handle other not expected situations.

System Interaction






This activity is the automatic one and runs with no human interaction

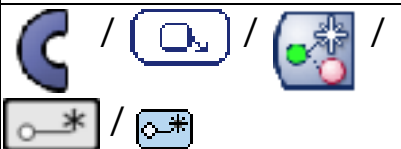

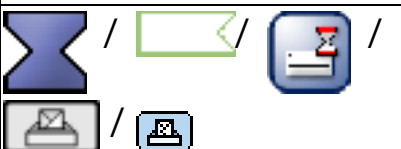
Classic / UML / Business Analyst / BPMN & Color	Activity Name	Definition
 /  /  /  / 	Automatic	Does not require any direct end user interaction. Performs many of the behind - the - scenes work on behalf of the process, such as batch program runs .

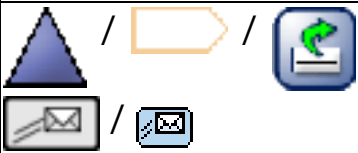
Organizational Interaction

These activities are also called the Inter Process Communication activities or IPC activities.

They work in pairs and allow communication from one process to another. Processes can be in the same Server or across Servers. Furthermore, they can belong to the same company or across companies (Business to Business environment.) There is always one calling activity and its corresponding called activity, or the notifying activities and the notified activity.





Classic / UML / Business Analyst / BPMN & Color	Activity Name	Definition
 /  /  /  / 	Subflow	Used to invoke a subprocess synchronously, so that the parent processing stops at this activity and awaits completion of the subprocess. Useful for making complex

Classic / UML / Business Analyst / BPMN & Color	Activity Name	Definition
		processes more easily understood and to enable B2B communication between different processes.
	Process Creation	Invokes a child process asynchronously, so that the instance in the parent process can continue without waiting for completion of the child process.
	Termination Wait	Gives an optional synchronization point in the calling process for a called subprocess using a Process Creation activity.
	Notification Wait	Synchronizes processing of parent and subprocesses. Instances wait in the Notification Wait activity until notification. However, if the activity has an outgoing due transition or a process deadline and time has elapsed, workflow continues without waiting for the notification. Process Notification and Notification Wait

Classic / UML / Business Analyst / BPMN & Color	Activity Name	Definition
		activities work together to allow communication from a called subprocess to calling process. Notification Wait can also be notified by the Notification component or an external component via PAPI.
	Process Notification	Used to alert a Notification Wait activity in a parent process that a subprocess instance is ready to pass results as required. Sends notification to the destination Notification Wait activity (defined in the properties dialog box) so the two activities will synchronize their flow.

Process Control
















These activities enable you to control the instances flow or to generate copies of it in order to let the instance flow through different paths simultaneously.

Classic / UML / Business Analyst / BPMN & Color	Activity Name	Definition
	Split-Join	Allows an instance to travel several parallel paths at the same time. Copies of the instance are created for the concurrent processing of the instance. Instances entering a Split have a corresponding Join to complete the circuit and resume process flow. Split is added to automatic roles.
	SplitN-Join	Used to build a Split/Join circuit that can create a variable number of copies of the instance. Requires a Join to complete the circuit. Split-n is added to automatic role lanes.
	Join	Instances entering a Split or Split-n activity have a corresponding Join activity to complete the circuit and resume the process flow. Scripts re-integrates copies created in the Split or Split-n activity.
	Conditional	The Conditional Activity helps centralize different paths of the process

Classic / UML / Business Analyst / BPMN & Color	Activity Name	Definition
		into one activity and re-distribute the instances based on conditional transitions.

Global Activities

These activities work globally and not with an instance. They are used to add tasks to the process but not referred to an instance. They can be used to generate instances but not to work with them.

Classic / UML / Business Analyst / BPMN & Color	Activity Name	Definition
 /  /  /  / 	Global Creation	One of the ways to start a new instance in a process. There is an implied transition to the Begin activity in a process.
 /  /  /  / 	Global Automatic	Does not have any end user interaction. Can be set to perform its Script on a set schedule or upon occurrence of a specific event. One way to start a new instance in the process. Used in automatic roles.
 /  /  /  / 	Global	Allows end users to run applications or database queries that are not an

Classic / UML / Business Analyst / BPMN & Color	Activity Name	Definition
		integral part of the process. Any applications that are invoked from the Global activity are not scheduled and can be run on an "as needed basis". Global activities do not have any interaction with instances.

Adding and removing activities from a process

You can add activities to a process by right-clicking on the desired role lane or clicking on the desired activity in the activity toolbar and dragging it to the desired role lane.

To add an activity

1. Right-click within the role column where you want to add an activity.
2. Select Add Activity and select the type of activity you wish to add

OR

1. Select an activity type from the Activity toolbar and then click

again on the role lane to which you wish to add an activity. The Activity Properties dialog box appears.

If you drag an activity from the toolbar and drop it on a Transition (on the arrow), the Transition will split, thus creating two transitions:

1. One from the original *from* activity, *to* the new inserted activity.
2. The second one *from* the new inserted activity *to* the previous activity destination.

If you drag an activity from the toolbar, as you move the activity, the nearest Transition (the one into which the activity will be inserted) changes its color. If you do not want the activity to be inserted into the flow, then select the activity from the toolbar while pressing the Ctrl key. As well, when you delete an activity the flow is automatically reconnected. All the activity's incoming transitions are sent to the activity that was receiving the unconditional transition.

Note



If the Activity Properties dialog box does not appear, right-click on the activity and select Properties from the shortcut menu. To change your preferences, select File and then Preferences from the Process Designer menu options to display the Preferences dialog box. Select the Activity tab and select the Show properties automatically when adding an object option.

Activity properties

- Type a name in the Name field. This name will appear in the process design and in Work Portal. This field is not limited.
- Optionally, click the Localize button to enter the label in alternate languages. See Internationalization for further information on

available languages.

- Optionally, type a Description for the new activity in the Description field.
- Depending on the type of activity, there may be additional properties to set.
- The activity image is the image that is displayed in the process designer to represent the activity type. There are two ways to change activity images. One way is to modify the **Theme**. You can also change an activity's image from the **Activity Properties** dialog box.

To change an activity image

1. Right-click on an activity and choose **Properties** from the shortcut menu. The **Activity Properties** dialog box appears.
 2. Click the Image folder.
 3. Click **Local image**. The **Open** dialog box appears.
 4. Browse to the location of the image.
 5. Select the image and click **Open**. The Activity Properties dialog box is displayed with the new image.
 6. Click **Ok** to accept the changes or click **Reset image** to reset the image to its default appearance.
- Click OK to accept the properties.

Note



GIF and JPEG formats are supported for images.

Tip

you can change the activity label by selecting the activity and clicking on the displayed label. You can directly edit the label from there.

To remove an activity

1. Right-click on the desired activity to be removed.
2. Select Cut from the shortcut menu or press the Delete key on your keyboard.

The activity is removed, its icon is erased from the Designer workspace and any transitions to or from the activity are removed as well. Cut lets you remove any activity except Begin, End and Process Group.
















Note

If you delete an Activity with reusable scripts within the designer and it contained a Script, the Script will not be automatically deleted. You must manually delete it.




Working with activities

Depending on the type of activity, some of the following options will appear if you right click on an activity:

- **Properties:** they define the behavior of the activity when executed. See Activity Properties.
- **Subflow:** if the activity is related to another process/activity because it is an IPC activity, you have the chance to move quickly to the related process by clicking in *Subflow*. This option is enabled for the Subflow, Process Creation, Termination Wait, Process Notification, Notification Wait.

- **Add thread:** only available for Split activities.
- **Add Unconditional Transition.** See Unconditional Transition.
- **Add Conditional Transition**  / : See Conditional Transition.
- **Add Due Transition**  / : See Due Transition.
- **Add Compensate Transition**  / : See Compensate Transition.
- **Add Exception Transition**  / : See Exception Transition.
- **Add Message Based Transition**  / : See Message Based Transition.
- **Main Task:** Some activities have a Main task that completes the activity. The implementation defines whether a Method, a Component Method, a Screenflow or a Procedure will run to achieve the task. See Tasks
- **Optional Tasks:** Some activities, such as **Interactive activities** or **Grab activities** can additionally have tasks that complete the activity. See Tasks.
- **Screenflow** : If the main task or any of the optional ones are implemented with as a screenflow, the screenflow's name will appear on the shortcut menu next to . If you double click on it, the screenflow design will open in the main panel.
- **Procedures** : If the main task or any of the optional ones are implemented with as a procedure, the procedures's name will appear on the shortcut menu next to . If you double click on it, the procedure design will open in the main panel.
- **Business Process Methods** : Some activities have BP-Methods or tasks that are implemented through scripts. Each BP-Method defined will populate in the BP-Method section by its

name. See Methods. Fuego Business Language (FBL.)

- **Argument Mapping:** Some activities receive or send arguments between activities (for example, Inter Process Communication.) The Argument Mapping defines and controls this communication.
- **Create group with selection:** a set of activities can be grouped. This group has its own properties. See Groups.
- **Remove from group:** this option unselects an activity within a defined group. It is only available if the activity belongs to a group. See Groups.
- **Copy** : this option copies and sends it to the clipboard.
- **Cut** : this option will delete the activity and send it to the clipboard so it can be later *pasted* where required.
- **Delete** : this option will delete the activity and will not be sent to the clipboard so it cannot be recovered. You can **undo** a delete action.

Documenting activities

As part of the project documentation, you can document the activities and its *use cases*. This documentation will be included in a separated section within the Project and process report - Generate documentation. See Documenting the project for further information.

Setting Preferences

See Setting Studio Preferences - Activity section.

Initiating a Process


Begin Activity

The Begin activity provides an entry point into every process. The Begin activity is where incoming instances finish the creation process. Instance variables are set for each instance as it flows through the Begin activity.





When a new process is created, the FuegoBPM Studio automatically creates a Begin activity. There can be only one Begin activity in a process. Instances can enter a Begin activity with arguments initialized by any of the following:

- A Global Creation activity in the process.
- A Global Automatic activity in the process.
- A Subflow activity in another process.
- A Process Creation activity in another process.
- An external application or web page (through a servlet) that uses the Process and ProcessInstance components to initiate an instance in a process, using PAPI or WAPI.

Note

 There is always an implied (but never drawn) path from any Global Creation to the process' Begin activity.

Begin activity characteristics

Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			

Work Portal and the Begin activity

The Begin activity is an entry point into the process. It will not

appear in Work Portal.

Roles and the Begin activity

The Begin activity automatically appears when you select **File** and **New Process** from the FuegoBPM Studio menu options.

The Begin activity can be moved around on the Process Designer workspace. The Begin activity can be either in an automatic or an interactive role.

Variables and Begin activities

The mapping between instance variables and process arguments is specified at the Begin activity.

The Instance variables will be automatically completed with the received arguments, as designed in the Argument Mapping.

Preconditions

Argument variables come into the Begin activity from an external component or application, a Global Creation activity, a Global activity, a Global Automatic activity executing ProcessInstance component or from an activity type that starts a subprocess.

These arguments respond to the **Argument Mapping** defined in the Begin activity.

Post-conditions

An instance is created within the Begin activity. Instance variables are initialized from the incoming arguments, as mapped. The instance flows to the next activity according to transition rules.

Properties

The Begin **Activity Properties** dialog box allows you to enter a description for the work the activity is to perform.

Activity: Begin

Category

- Activity Id
- General
- Advanced
- Image

Activity Id

Activity Id *Begin*

Name

Begin Localize

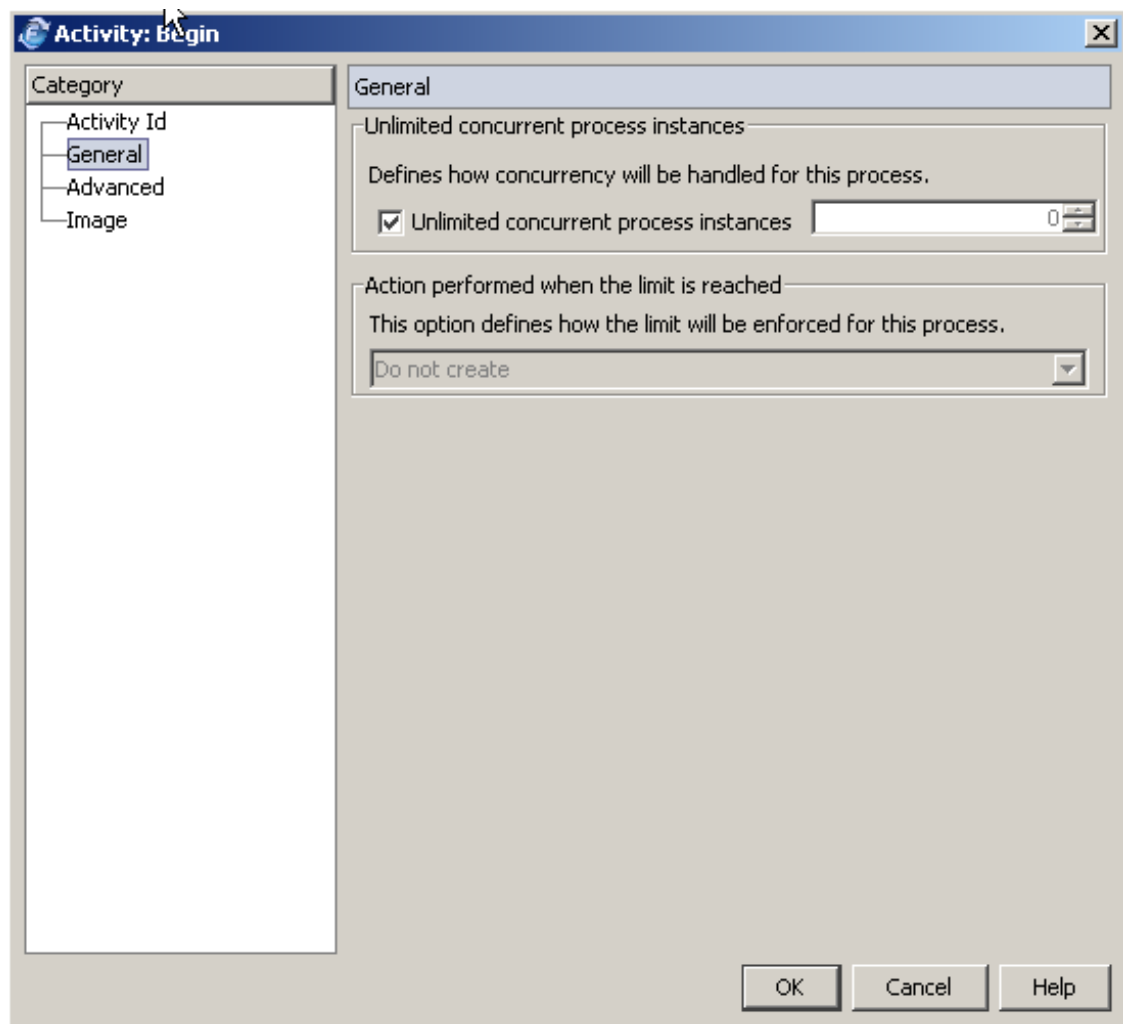
Description

Begin Localize

OK Cancel Help


General Category

To control the number of instances that can enter the process at the same time, complete the following:



If you disable the **Unlimited copies** check box, the field next to it is enabled for you to define the **Maximum number of active instances** that are allowed to run simultaneously in the process. The instances above this number will be treated as defined in **Action performed when the limit is reached**.

Note

 If the Begin activity is set to a maximum number of instances, no Interactive Activity can exist within the process. This situation mostly applies when the process employs non-human resources.

The **Action performed when the limit is reached** drop-down box is enabled when you designate a maximum number of instances for

the Begin activity.

If your process is receiving more instances than the number you indicate in the **Maximum amount of instances** field, you can choose any of the following from the **Action performed when the limit is reached** drop-down menu:

- **Do not create:** If you have a monitoring process that is simply monitoring a service, you may have a situation in which several messages that contain the same information are being generated. For example, if your database suddenly stops working and you have a polling activity listening to the service, the polling Global Automatic may try to create several instances with "FAIL". It is not necessary to waste FuegoBPM Server space with the same message over and over again. If you wish, you can choose **Do not create** to cancel instance creation.

The instances that are not created are logged in the FuegoBPM Server log available from the FuegoBPM Logviewer.

- **Allow creation but limit concurrent executions:** If all instances coming into your process are critical, you must choose to **Allow creation but limit concurrent executions**. All incoming instances that exceed your **Maximum amount of instances** number are created but kept in the Begin activity. When an instance within the process finishes the process (it may be sent to the End activity or aborted), an instance from the Begin activity begins to flow.

Tip



If you are creating instances from an external component or method, and these instances are created altogether in a single transaction, whenever the maximum number is reached, the transaction will fail. Therefore, NO instance is created.

Advanced Properties

Some extra properties can be defined:

Generate Events : See Generate Events.

Transitions and Begin activities

No incoming transitions are allowed to the Begin activity. One or more outgoing transitions must leave the Begin activity. Begin activities cannot transition to a Grab activity.

If the Begin activity has multiple sets of arguments, the Message based transitions are available. See Message based Transition.

Tasks

No tasks are available.

Business Process Method's considerations

In order to pass arguments between processes, the Argument Mapping function is available. See Argument Mapping.

Advance scripting is available. The most common use is to validate the passed arguments.

After setting the argument as defined in the mapping, the advance script is run.

Each argument mapping has an advanced script to run independently.

Note



If any of the instance variables used in the argument mapping are modified in the script, the values passed in the mapping are lost and the new ones become valid.





End Activity

The End activity is always the last activity in a process. It also provides an exit point from it. When you create a new process, FuegoBPM automatically creates an End activity. There can be only one End activity in a process.

The End activity transforms instance variables into arguments (See Argument Mapping) that can be then passed to another process or to an external application. Upon completion, the End activity can return flow to any of the following:

- A Subflow activity that called the process.
- When a process was started by another process using a Process Creation activity, the called process' End activity can return instances to the calling process' Termination Wait activity.
- An external application.

End activity characteristics

Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			

Work Portal and the End activity

The End activity is an end point of the process. It will not appear in the Work Portal.

Roles and the End activity

The End activity automatically appears when you add a new process. It can be moved around on the process design workspace, but it must reside in an Automatic role lane.

Variables and the End activity

The End activity can define instance variables to determine the argument mapping. See Argument mapping.

The End activity transforms instance variables into outgoing arguments.

Preconditions

Instances come into the End activity from the process.

Post-conditions

When an instance completes the End activity, it is completed within the process. If the process was called by a Subflow or Process Creation activity, the instance variables are set to outgoing arguments, as defined in the Argument Mapping, and are passed back to the calling process from the End activity.

Properties

The End Activity Properties dialog box allows you to enter a description of the activity.

Advanced Properties

Some extra properties can be defined:

Generate Events : See Generate Events.

Transitions and the End activity

One or more incoming transitions are required. End activities can only transition to a Grab activity (outgoing transition). Instances can be grabbed from the End activity unless the instance status is aborted or terminated.

Tasks

No tasks are available.

Business Process Method's considerations

To pass arguments between processes, the Argument Mapping function is available.

Arguments are passed to the calling process to return information to it

Advance scripting is available. The most common use is to proceed with clean up actions or to free resources.

The script is run first and then the argument mapping is set.





Human Interaction

Interactive

Interactive activities bring end users into the process. Within the BSO (Business Service Orchestration), it facilitates the interaction of **human participants**.

It manages multiple Methods to automatically invoke components that require end user interaction. Its Method can be as complicated as needed in order to accomplish a task, and each Interactive activity can contain multiple tasks. Tasks are a list of functions that can be performed while an instance is in the activity. Each task may or may not be required or repeatable. See Tasks for further information.

Interactive activity characteristics

Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			

Work Portal and the Interactive activity

The Interactive activity is visible in Work Portal, but only to the participant(s) assigned to the role where the Interactive activity exists.

Roles and the Interactive activity

Interactive activities must reside in user-defined roles.

Tip



If you add a new interactive activity and drop it in an automatic role, FuegoBPM will allow you to select an existing user-defined role or generate a new one (the Role dialog box populates.) The column-role is added to the design and the new interactive activity will automatically appear in the selected role.

Variables and Interactive activities

Interactive activities can access argument, instance, local and predefined variables from the Method.

Preconditions

An Incoming instance from another activity in the process with the necessary instance variables set.

Interactive activities can also be used as an exception handler. In this case, the Interactive activity receives an incoming instance from an activity in which the BP-Method has thrown a user-defined exception. See Exception overview for further information.

Post-conditions

Instance moves to the next activity in the process via one of the outgoing transition lines.

Properties

Activity ID

See Creating an Activity.

General Category

Suspendable: Process participants can take instances off the clock in Work Portal by selecting **Suspend**. This means that due transitions and process deadlines are temporarily disabled for an instance once it has been suspended.

If an instance has been suspended, it will not expire.

User selects transition: If there is more than one unconditional transition exiting from an activity, the process participant can select which one should be taken.

Auto complete: The instance automatically flows to the next activity in the process if:

- all mandatory tasks have been completed in Work Portal, or
- there are no mandatory tasks and a read-write task has been executed.

Abortable: End users can eliminate instances from the process. Aborted instances flow immediately to the End activity.

Warning



Aborted instances cannot be retrieved.

Assignable: This property enables the participant to reassign any of the instances that are in the Interactive activity to another participant that belongs to the same role where the activity is located. The assignment to another participant depends on the category that each participant has for the role. These categories are defined through the Web Console.

If the instance is originally assigned to the role (no participant has

grabbed it yet), any participant of this role can select it and reassign this instance to any other participant of the role.

Refer to Participant's permission for instance assignment for further information.

Advanced Properties

Unlimited copies: Defines the number of instances that are allowed in the Interactive activity at the same time. If you clear the Unlimited copies check box, you can enter a number of instances in the **Maximum amount of parallel processing instances** field. Any instances exceeding such number will be queued until an instance is completed and leaves the Interactive activity.

Generate Events: See Generate Events.

Transitions and Interactive activities

One or more incoming and outgoing transitions are required.

Tasks

The default behavior for Interactive Activities consists in having one single Implementation or task.

This greatly simplifies the understanding of Interactive Activities.

Furthermore, this simplifies the operation of Work Portal since executing an instance is ALWAYS executing its implementation.

However, you can have **Optional** (or support) tasks that the participant may need to execute in some cases in order to help him do the required work (that is, executing the main task.)

But if you look at optional tasks in this way, they make sense ONLY if you allow the user to execute them while executing the main task.


To do so, these optional tasks must be *read-only*, meaning that they must not modify the instance data in order to avoid inconsistencies.

The tasks have to be defined by the developer. The interactive activity has then a main task. Moreover, optional tasks can be added. See Tasks.

Business Process Method Considerations

Interactive activities can have one or more BP-Methods defined as tasks. Applications invoked from Interactive activities are sent incoming parameters (set by BP-Method variables) and return outgoing parameters (which set BP-Method variables).

Note




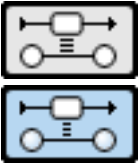



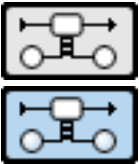



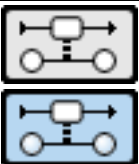
 Human participants are unpredictable. If an end user begins a task in Work Portal but does not complete it, a thread for that task remains open in the FuegoBPM Server. Too many open threads can cause poor server performance.

To ensure that end users do not lock threads, use *relay to* in your BP-Method to immediately close threads and relay end user's response to another BP-Method task when the end user completes a task. See Controlling threads with relay to for further information.

Grab

Grab activities give processes the flexibility to deal with slowdown conditions and to redistribute instances as appropriate to alleviate such conditions. Grab activities are most commonly used in supervisory roles within the process. This enables supervisors to monitor instance flow. They can use the Grab activity to easily move instances from one activity's queue to another activity's queue.

Grab activity characteristics

Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
 Defined			 /
 From All - To All			 /
 From All			 /


Work Portal and the Grab activity

All Grab activity types are visible in Work Portal to the user(s) assigned to the role.

Roles and the Grab activity

All Grab activities must reside in a user-defined role.

Tip

 If you add a new grab activity and drop it in an automatic role, FuegoBPM will allow you to either select an existing user-defined role or to generate a new one (the Role dialog box populates.) The column-role is added to the design and the new grab activity will automatically appear in the selected role.

Variables and Grab activities

All Grab activity types can access argument, instance, local and predefined variables from the BP-Method Editor.

Preconditions

Grab activities require incoming grabbed instances from other activities in the process. **Defined** Grab activities can only receive instances from activities to which they are attached by transition.

Grab **From All** and Grab **From All/To All** activities can grab instances from any activity in the process.

Post-conditions

Defined Grab: Once marked complete, instances are sent to the next activity in the process via one of the outgoing transition lines.

Grab From All : Once marked complete, instances are sent to the next activity in the process via one of the outgoing transition lines.

Grab From All/To All: Once marked complete, instances are manually sent to the appropriate activity in the process by the end user.

Note



Instances that have been grabbed to one Grab activity and remain in this activity cannot be grabbed from another Grab activity.

Properties

Activity ID

See Creating an Activity.

General Category

Grab Type:

- **Defined Grab** activities have one incoming transition line or several incoming transition lines. This means that this type of activity can only grab from the specific activities that transition to it. They may or may not have outgoing transitions. If there is an

outgoing transition, the **Defined Grab** can only redistribute instances to these activities.

- **From All Grab** activities have no incoming transition lines because they can grab from any activity in the process. They may or may not have an outgoing transition to, at least, another activity. This means that the From All Grab can grab instances from any activity's queue, but can only redistribute instances to activity queues which are connected to the Grab by a transition.
- **From All/To All**, this Grab activity is the most powerful one. It can grab instances from any activity and redistribute them to any other activity. People assigned to the role of a **From All/To All Grab** should understand all activities in the process and the consequences that arise when moving instances between activities.

Suspendable : End users can take instances off the clock by suspending them in Work Portal. This means that Due transitions are temporarily disabled for an instance once it has been suspended.

Abortable : End users can eliminate (abort) instances from the process in Work Portal.

Warning



Aborted instances cannot be retrieved.

User selects transition : If there is more than one transition coming out of an activity, the end user can select which one should be taken.

Note



From All/To All Grab types are very powerful. Since an instance can be grabbed from any activity and directed to any other activity, no transitions are required. Extreme care must be taken when adding a From All/To All Grab type to your process. Grabbed instances will not be able to follow the

process flow until they are ungrabbed or rerouted to activities that ensure that instances are being completed according to business and process rules.

Generate Events: See Generate Events.

Transitions and Grab activities

Defined Grab: One or more incoming transitions are required. An outgoing transition is not necessarily required. There is an implied *back* transition that takes instances back to the activity from which they were grabbed. This is accomplished by clicking the **Ungrab** button in Work Portal.

If a Defined Grab activity has transitions to or from activities inside of a Split-Join circuit, this same Grab activity cannot have transitions to or from activities outside the Split/Join circuit. Similarly, a Grab activity with transitions to or from activities outside of a Split/Join circuit cannot have transitions to or from activities inside a Split/Join circuit.

Grab From All: No incoming transitions are required. The Grab From All activity can grab instances from any activity in the process. An outgoing transition is not necessarily required. There is an implied transition that takes instances back to the activity from which they were grabbed.

Grab From All/To All: Grab from All to All can take instances from any activity and can send them (re-deploy) to any activity.

Note



If you do not want to see the **Grab transitions**, disable the **Show Grab Transitions** property from the **View** menu, **Transitions** option.

Tasks

Tasks have to be defined by the developer. The grab activity has a main task. Moreover, optional tasks can be added. See Tasks.

Business Process Method Considerations


Grab activities only have access to another activity's instances. Grab activities cannot access the BP-Method for the activity it is grabbing. However, you can assign the same BP-Method to the Grab activity that is assigned to the activity that you are grabbing instances from. For example, if you have an Interactive activity called **Review Order** that has one BP-Method called **enterCustomerInfo**, you can apply the same BP-Method to the Grab activity.

You can add tasks to the Grab activity to process the BP-Method according to business rules. You can also ungrab the instance and allow it to go back to the activity from which it was grabbed in order to continue processing.

Groups and Grab activities

See Groups and Grab activities.

Note




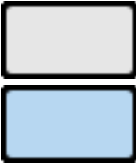
 If you do not want to see the grab activities in your process design, you can hide them. Select **View** and **Activities** from Studio's main menu. Clear the **Show Grab activities** check box.

System Interaction

Automatic

Automatic activities do not require any direct end user interaction. Applications and components interfaced with Automatic activities should not require any end user intervention. Applications/components typically run on a remote server and perform work behind the scenes. For example, database maintenance, e-mail notification and so on.

Automatic activity characteristics

Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			 /

Work Portal and the Automatic activity

Automatic activities do not appear in Work Portal because the Automatic activity does not require any end user intervention. Any BP-Method in Automatic activities is automatically executed.

Roles and Automatic activities

Automatic activities can appear either in automatic roles or in the user defined role types. However, if the Automatic activity is in a user-defined role, it will not appear in Work Portal.

Variables and Automatic activities

The Automatic activity can access predefined, local or instance variables.

Preconditions

The Automatic activity is activated by an incoming instance from another activity in the process.

Post-conditions

After the BP-Method tasks in the Automatic activity have been completed, the instance flows to the next activity in the process according to transition rules.

Properties

Activity ID

See Creating an Activity

Advanced Properties

Unlimited copies: allows you to control the number of instances that may flow through an activity at the same time. As you design your process, think about which activities may take longer to process instances due to component connections or resource allocations. You may want to slow down some of the fastest activities by limiting the number of instances that can be concurrently processed.

Maximum amount of instances: If you clear the **Unlimited copies** check box, the **Maximum amount of instances** field will be activated and will allow you to enter a number of instances. Instances exceeding the maximum amount limit are queued. The queued instances will not start processing until the first group of instances is complete.

Generate Events: See Generate Events

Transitions and Automatic activities

One or more incoming and outgoing transitions are required.

Tasks

The task has to be defined by the developer. The automatic activity can have only one task to be completed, that is, the **Main Task**. See Tasks.

Business Process Method considerations

Automatic activities can have only one BP-Method task. The BP-Method is automatically performed.

Components invoked from Automatic activity BP-Method tasks are sent incoming parameters (set by BP-Method's variables) and may return outgoing parameters (which set BP- Method's variables.) Components accessed from Automatic activities should not require any end user intervention. If user interface is required, use an

Interactive activity instead.

You can use a display Method statement in the BP-Method of an Automatic activity for testing purposes. Instead of being displayed to an end user, the display statement writes to the FuegoBPM Server log.

Typical uses

Automatic activities are used in the process design where work can be performed without human intervention. Some typical uses are as follows:

- Database updates.
- Running batch programs.
- Sending e-mail notifications.
- Sending e-mail confirmation to customers.

Automatic activity example

In the following BP-Method, the **Send E-mail** activity sends e-mail notifications to the system administrator's e-mail inbox depending on the value in the **serverDown** variable. The whole BP-Method is automatically executed.

```
if serverDown == true then
  send Mail using
    to = "sysAdmin@fuego.com"
    message = "Venus server is down. Please check!"
end
```

In the following example, a database is updated with new employee's information. This BP-Method is in the Automatic activity because it does not require any end user intervention to update the

database.

```
INSERT INTO employee (fname, lname, salary, title)
VALUES (firstName, lastName, salary, jobTitle);
```

How to rollback Automatic activities actions

If an automatic activity fails, the Server will try to execute it as many times as defined in the Server properties **Retry times** and **Retry interval**.

If the execution is still not successful, then an exception is thrown.

If any action of those that have been completed in the Automatic activity needs to be reverted, you must manually define the rollback in an Exception Handling Flow for that Automatic activity.

The best way to implement the rollback is to define an Exception Flow for each possible known exception. Therefore the Automatic activity will have multiple exception transitions, one for each known exception that might occur. To contain all other not expected exceptions, design the Exception Flow for the *Others* exception.

Process Controls

Split-Join Circuit

A Split activity allows an instance to travel over several parallel paths at the same time.

There are two ways to accomplish several paths flow:

1. Copies of the instance are created for the concurrent processing of the instance as it flows through the different paths.
2. The original instance is the one that flows through the different

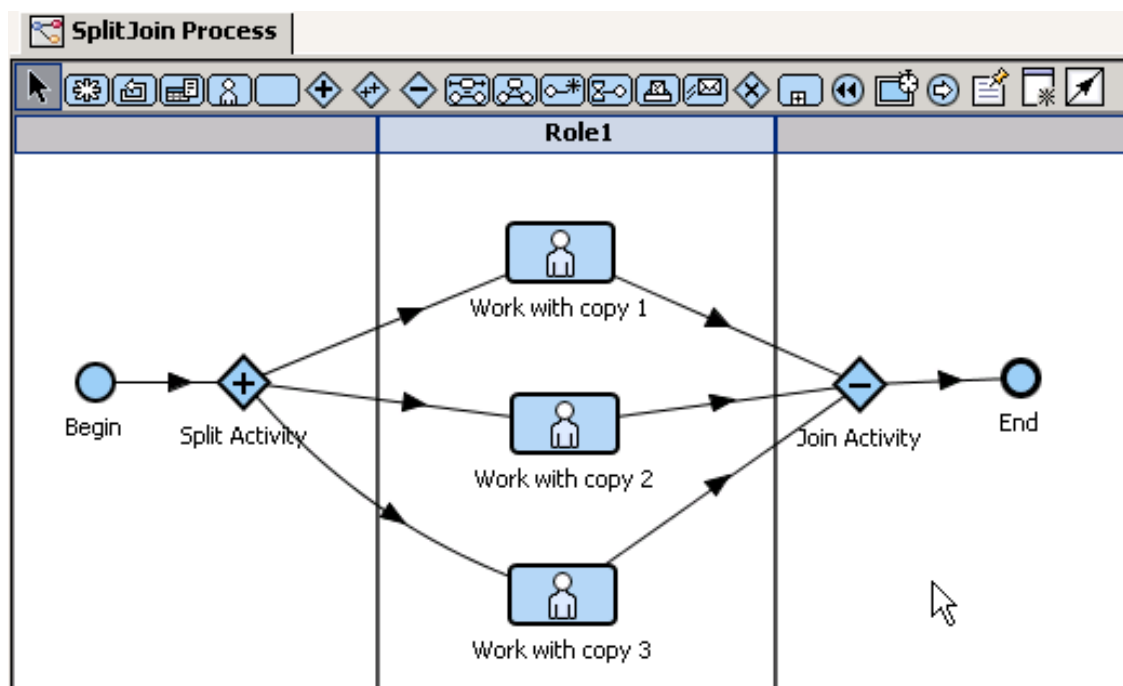
paths.

The number of copies that the Split generates is the number of outgoing unconditional transitions plus any conditional transitions that evaluate to **true** for the Split activity.


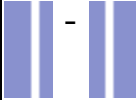
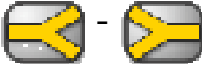

Split activities must have a corresponding Join activity in order to complete the circuit and resume the process flow.

Activities within the Split / Join circuit cannot have any transitions to or from activities outside it. An exception to this rule is when you have a Grab activity to handle overrun conditions within the Split / Join circuit. However, in this case, grabbed instances can only be sent back to the activity from which they were grabbed or to the End activity and never to another activity outside the Split / Join circuit.

When an instance reaches a Split activity, the original instance immediately flows to the corresponding Join activity. Copies of the original instance travel through the activities inside the Split / Join circuit.



Split and Join characteristics

Classic Icons	UML Icons	Business Analyst Icons	BPMN Icons
 Split and Join Activities			

Work Portal and the Split and Join activity

Split and Join activities are not visible in Work Portal.

Roles and the Split and Join activity

Split and Join activities can reside in user-defined and automatic roles. However, if the Split and Join activity is in a user-defined role, no activity appears in Work Portal.

Variables and Split and Join activities

Split and Join activities can access instance, local and predefined variables from the BP-Method Editor.

Join activities can also reference copies of instance variables. If the variable is a copy, it is referenced as follows:

```
// Set the original variable with the copy
// variable.
orderTotal = copy.orderTotal
```

Arguments: the Join activity has the **copy** argument available and it is used as "**copy**".

If the Split activity **does not generate copies**, and thus, the original instance is the one following in the parallel paths, the Join

activity will have no BP-Method available.

Preconditions

Split activities require an incoming instance from another activity in the process.

Join activities require incoming copies from one of the parallel paths reaching the Join activity. If the original instance is still in the Join activity, the copies are consolidated with the original instance.

Post-conditions

Split: When an instance reaches a Split activity, the original instance is automatically sent directly to the corresponding Join activity. While still in the Split, instance copies are created and each copy flows across one of the parallel paths in the Split / Join activity circuit.

Join: The original instance can leave the Join activity due to one of these four reasons:

- If the **number of copies to wait to release** property is set, the original instance leaves the Join activity after that number of copies have arrived to this activity. If it is not set, then,
- After all copies have reached the Join activity, the instance moves to the activity following the Join activity according to transition rules.
- When a copy reaches the Join activity and the Join's Method sets the predefined variable *action* to *RELEASE* (*action* = *RELEASE*).
- When the original instance expires because of a due transition or when the process' instance expiration exception handling has occurred.

Properties

All Split activity must have the corresponding Join activity.

SPLIT

Activity ID

See Creating an Activity.

If you right click on a Split activity, you have the option **Add thread**. If you select it, a new unconditional transition is generated opening a new path for a new copy within the Split-Join circuit.

Advanced Properties


Generate copies: If this option is selected, copies of the original instance are created as well as the instances variables are generated at Split time. Then, each copy flows independently until they arrive at the Join activity.

If you need to share the instance variables between the different copies within the Split-Join circuit, then do not select **Generate Copies**.

If copies are not generated, then the original instance is the one that will hold the instance variables.

If any of the activities within the Split-Join circuit modify the instance variables, the change will be immediately available for all the parallel instances that are flowing.

Note

 Not generating copies becomes useful when all the outgoing paths are not modifying instance variables but using the existing information (for example, to update other systems). As you do not need to consolidate data, there is no need to generate additional code in the SPLIT and JOIN nodes. Keep in mind that under this scenario if you do modify the instance variables, the FuegoBPM Server will lock them to update any information

Generate Events: See Generate Events.

JOIN

Activity ID

See Creating an Activity.

Advanced Properties

Aborted Copies: You can select where the instance copy will go if it is aborted.

The instance copy will be routed to the chosen activity (Join or End) if:

- The instance copy is aborted within the split-join because of process design.
 - The instance copy is aborted by the user within the split-join.
 - The instance copy throws an exception in the split-join and then process participant aborts it.
-
- If the instance is sent to the Join, the Join BP-Method will be executed for the copy.
 - If the instance is sent directly to the end, it will not go through the Join activity. Therefore, no BP-Method is executed.

Amount of copies to wait to release : you can set how many of the instances created in the Split activity should be waited for before the instance waiting in the Join activity moves forward in the process.

Generate Events: See Generate Events. In particular, the Generate events property is automatically enabled if the associated Split activity is also enabled.

Transitions and Split and Join activities

Split: At least one or more incoming transitions are required. At

least two or more outgoing transitions are required.

Join: The number of incoming transitions must equal the number of outgoing transitions from the Split activity. One or more outgoing transitions are required.

If you add a transition from the Split activity, it will automatically be connected to the corresponding Join activity.

Tasks

No tasks can be generated. A BP-method will be automatically created with the same name as the activity.

The Join Activity associated script will be automatically generated after you check the design. At that moment the Split-Join circuit is consolidated.

Business Process Method considerations

If the Split activity **generates copies**, then each parallel path between the Split/Join activity circuit creates its own copy of the original instance. The Join activity acts as a marshaling point where the copied instance is used to update variables in the original instance.

The Join's BP-Method MUST only be used to merge the copies' instance variables into the original instance's variable.

To differentiate copy instance variables from the original instance variables, "copy." is used in front of the instance variable.

```
creditStatus = copy.creditStatus
```

Some means should be used in order to keep track of the copy instances. This allows only certain instance variable updates in the original instance. The instance variable *copy.activity.source.name* is used to determine from which activity the copy instance came. For

example, the Method in the Join activity may appear as follows:

```
// Check where the copy instance came from

switch copy.activity.source.name in
case "CheckCredit":
  creditStatus = copy.creditStatus

case "CheckInventory":
  shipStatus = copy.shipStatus

case "OrderProduct":
  shipStatus = copy.shipStatus

end
```

If the Split **does not generate copies** and the original instance is the one that effectively flows, then there will be NO Join BP-Method available.

Split and Join example

In a human resources hiring example process, a new employee is hired in a company. Several company departments need to be notified of the new employee to ensure that he or she is entered into the company's databases and systems. All departments can be notified at the same time by using a Split / Join circuit. After the instance completes the Split / Join circuit, information from all departments is collected and can be sent to the new employee via e-mail.

The different copies will flow to:

1. The IT department has to perform a number of activities so that the first copy flows to a Subflow activity *Notify IT Department*. Therefore, the copy instance is sent to a subprocess where all IT services, such as computer purchasing and network passwords are set.

2. Additionally, an Automatic activity called *Add Employee to Payroll* is performed. This activity adds the employee's information to the company's payroll system.
3. The Travel department needs to enable this employee for any traveling issues. An Automatic activity *Add Employee to Travel* receives one of the copies to assign travel privileges to the employee in the company's external business travel software application. The employee receives a travel ID and a password to access the travel system.
4. Finally, some Security issues need to be performed. An Automatic activity *Add to Security Database* assigns security clearances to the employee and updates the Security database.

Join Activity

As in this example, the Split activity **generated copies**, the Join BP-Method must consolidate the copies instance variables used while they flew within the Split/Join circuit.

The Join activity rejoins the copy instances with the original instance. Note that the use of "copy." and the name of the activity from which the instance came are used to set the instance variables in the original instance.

```
// Check to see where the copy came from
switch copy.activity.source.name in

case "NotifyITDepartment":
// update appropriate instance variables
    pc = copy.pc
    eMail = copy.eMail
    networkID = copy.networkID
    password = copy.password

case "AddEmployeeToPayroll":
// update appropriate instance variables
    employeeID = copy.employeeID
    payStatus = copy.payStatus
```

```

case "AddEmployeeToTravel":
// update appropriate instance variables
    travelUserID = copy.travelUserID
    travelPassword = copy.travelPassword

case "AddToSecurity":
// update appropriate variables
    security = copy.security

end

```

Continuing with the Process

To complete the Hiring process, the employee is notified of all the updated information about him.

A mail is automatically sent to him through an Automatic activity called *Send Mail to Employee*.

The Automatic activity *Send Mail to Employee* generates an e-mail with the combined information from each department. The e-mail communicates all user IDs and passwords given to the employee.

```

hrMessage = "Dear " + firstName + "," + "\n\n" +
"Welcome to our company, we are glad to have you
    working with us.
    The following information contains user IDs
    and passwords to help you log in to the
    various company systems." + "\n\n" +
"PC ID: " + pc + "    Use this ID to log trouble tickets
    with the IT department." + "\n" +
"Network ID: " + networkID + "    Use this ID to log
    into your PC." + "\n" +
"Network Password: " + networkPassword +
    "Use this password to log into your PC."
    + "\n\n" +
"Payroll Status: " + payStatus + "\n\n" +
"Travel User ID: " + travelUserID + "Use this ID to
    book business travel in
    the travel application." +
"Travel Password: " + travelPassword + "Use this
    password to log into
    the travel application."

```

```
        + "\n\n" +
"Security Level: " + security + "      You will receive your
        badge within a week." + "\n\n" +
"If you experience any problems with this information,
        call HR at 1-800-555-1212." + "\n\n" +
"Welcome aboard!"

// use Mail standard component to send the message
send Mail using
    message = hrMessage
    to = eMail
    from = "hr@ourcompany.com"
    subject = "Welcome Aboard"
```

Keeping track of updated instance variables

If the instances to flow within the Split-Join circuit are **copies**, when the copy instances reach the Join activity in a Split/Join circuit, the BP- Method should be programmed to keep track of which instance variables are to be updated. If this BP-Method was omitted, the instance variables would update to the values held in the last copy instance to reach the Join activity. There are two ways to handle this: using Boolean variables or using a switch statement.

Boolean variables

As each copy instance flows into the Join activity, the original instance variables are updated with the values associated to the copy instance. Therefore, Split/Join circuits must use Boolean variables to keep track of the copy instances. All Boolean variables are set to *false* in the Split activity and then set to *true* in the activities between the Split and Join circuit.

For example, in an order management process, some parallel activities occur between the Split and Join activities: *Check inventory, Calculate Freight, Check Credit/*

- Split activity: Three Boolean variables are set to *false*.

```
splitShipping = false
splitCredit = false
splitFreight = false
```

- Check inventory activity: a shipping clerk checks to ensure that the order is in inventory. If the items are not in inventory, the shipping clerk orders the products.

```
// Set Boolean variable to true
splitShipping = true
// This display statement has 3 buttons and a default.
// The selected button value gets returned to the
// local variable.
display "Is the product in inventory?"
      using title = "Check Inventory",
type = "plain",
options = ["Yes", "No", "Cancel"],
default = "Yes"
returning selected = selection

// See what the end user selected and set the instance
// variables.
// Note the use of elseif statement below.
if selected == "Yes" then
  shipStatus = "OK"
elseif selected == "No" then
  shipStatus = "BackOrder"
else
  shipStatus = "Invalid"
end
```

- Calculate Freight activity: a shipping clerk calculates the freight cost for the order.

```
// Set Boolean variable to true
splitFreight = true
// return shipping freight cost
```

```
freightCost = 400
```

- Check Credit: A finance clerk checks the customer's credit if the order is a credit order.

```
// set Boolean variable to true
splitCredit = true
display "Approve credit for this order?" + "\n" +
    "Customer: " +
    customerName + "\n" + "Order Amount: $" +
    orderAmt using title = "Approve Credit",
type = "question",
options = ["Yes", "No"],
default = "Yes"
returning selected = selection
// check the status and update the instance variable
if selected == "Yes" then
    creditStatus = "Approved"
else
    creditStatus = "Declined"
end
```

- Join activity: The copy instances rejoin with the original instance in the Join activity. See how Boolean variables in the BP-Method are used to update the appropriate instance variables in the original instance.

Note



To refer to instance variables in the copy instances, use **"copy."** in front of the instance variable name as shown in the example BP-Method below.

```
// Did this copy of the instance come from
// the "Shipping" branch?
```



```

if copy.splitShipping then
// set the instance variable shipping status
//from the copied status.
    shipStatus = copy.shipStatus
// Did this copy of the instance come from
// the "Credit" branch?
elseif copy.splitCredit then
// set the credit status.
    creditStatus = copy.creditStatus
// Did this copy of the instance come from
// the "Freight" branch?
elseif copy.splitFreight then
// set the freight status
    freightStatus = copy.freightStatus
else
    shipStatus = "Error"
    freightStatus = "Error"
    creditStatus = "Error"
end

```

Switch statement

Using the switch statement in BP-Method is an elegant way to discover from which path an instance arrived and then to update the appropriate instance variables. This way does not require any additional BP-Method in the Split activity or any activity residing within the Split/Join circuit.

Within the BP-Method below, the predefined variable **sourceActivity** and its property **name** are used to determine from which activity the copy arrived at the Join activity.

```

// This log statement is useful for debugging during runtime
logMessage "previous Activity was: " +
copy.activity.source.name using severity = DEBUG
// Determine which branch of the Split / Join copy
// came in from by discovering the Activity's name
// the copy came from.
switch copy.activity.source.name in

// Came in from Check Credit activity
case "CheckCredit" :
    creditStatus = copy.creditStatus

```

```
// Came in from the Check Inventory activity
case "CheckInventory" :
  shipStatus = copy.shipStatus

// Came in from the Order Product activity
case "OrderProduct" :
  shipStatus = copy.shipStatus

// Came in from the Check Freight activity
case "CheckFreight" :
  freightStatus = copy.freightStatus

end
```

Note



The **copy**. is Instance type, therefore it has all the "ProcessInstance" component methods and attributes available. As well it has all its parent instance variables defined.

Precedence transition

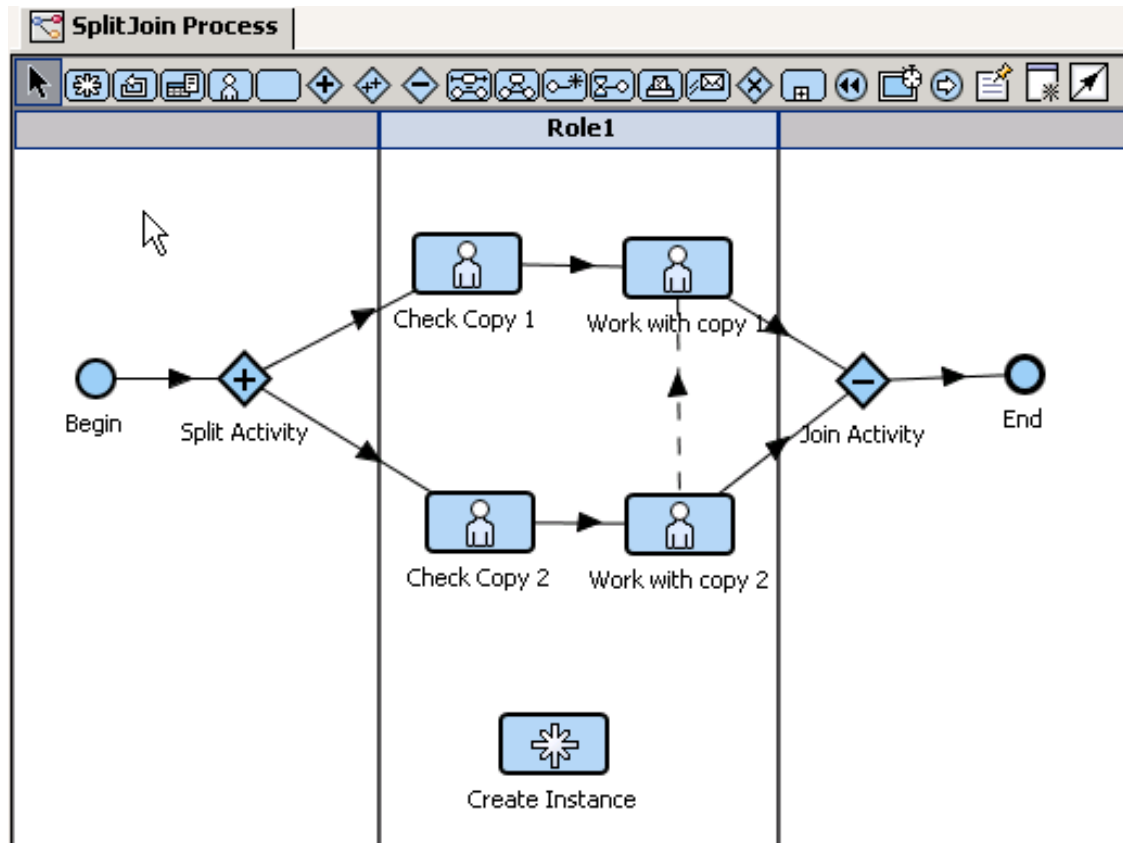
Copies within a Split-Join circuit can have a synchronization or a precedence.

If you generate 2 copies, but at one point within the Split-Join circuit, you need a task to be executed over one copy before another task is executed over another copy, you can define a **precedence** between them.

The precedence is implemented using a **Precedence transition** (represented by a dotted transition).

The precedence transition goes from the notifier activity to the notified activity. The copy will not be available in the notified activity until the notification is received from the activity you need to synchronize (the activity you need to be run first).

For example:



In the **Split** activity, 2 copies are generated.

Copy 1 flows first to the **Check Copy 1** activity but once this task is executed it will not appear in the **Work with copy** activity until the **Copy 2** is processed by the **Work with copy 2** activity.

Once this happens, a notification (following the precedence-dotted transition) is sent to the **Work with copy** activity and Copy 1 appears available in the activity.

In the Work Portal, Copy 1 cannot be processed until the notification is received from the **Work with copy 2** activity. It can only be seen if you search for it and it appears in an activity called **WMN_Work with copy** (Wait Multiple Notifications).

Split N-Join Circuit

The Split-n is used to copy an instance n times for processing

purposes. The easiest way to visualize how the Split-n activity operates is to picture a process that solicits bids from external vendors. The company wants to get multiple bids from different vendors and uses the BP-Method to select the lowest bid that meets the company's specifications.

BP-Method statements in the Split-n activity create the individual copy instances and set their respective instance variables. As an instance flows into a Split-n activity, the original instance automatically flows to the corresponding Join activity, while copies of the original instance are created. The original instance stays in the Join activity until all copy instances arrive. However, there are four exceptions to this rule:

- If the **number of copies to wait to release** property is set, the original instance leaves the Join activity after that number of copies have arrived to this activity.
- If there is a due transition leaving the Join activity, the original instance must follow the logic in the due transition.
- If there is a deadline for the entire process, the original instance will be rerouted to the Instance Expiration exception if such deadline expires.
- If the **action** variable is set to *release* in the Method of the Join activity. *action = release* releases the original instance from the Join activity.

The copy instances automatically inherit all the attributes of the original instance as they leave the Split-n activity. If end users add or change attachments of the copy instances, the attachments are automatically associated to the parent instance once they have reached the Join activity.

Note




You should be careful with the values of instance variables that may be











changed within the Split-n / Join circuit since the original instance will take the value of the variable from the last copy instance to reach the Join activity, unless the Method in the Join activity dictates otherwise.

Activities within the Split-n / Join circuit cannot have any transitions to or from activities outside it. An exception to this rule is when you use a Grab activity that resides outside the Split-n circuit to handle overrun conditions within the Split-n / Join circuit.

Note

 Grabbed instances can only be sent back to the activity from which they were grabbed or to the End activity. They can never be sent to another activity outside the Split-n / Join Circuit.

Split-n and Join characteristics

Classic Icons	UML Icons	Business Analyst Icons	BPMN Icons
 -  Split-n and Join Activities	 - 	 - 	 -  /  

Work Portal and the Split-n and Join activity

The Split-n and Join activities are not visible in Work Portal.

Roles and the Split-n and Join activity

Split-n and Join activities reside in automatic roles. They can also reside in user-defined roles. However, no activity will appear in Work Portal.

Variables and Split-n and Join activities

Split-n and Join activities can access instance, local and predefined variables from the BP-Method Editor.

Copies have to be created manually and not graphically as it occurs in the Split activity where each outgoing transition will produce a copy creation.

To create copies, you have to generate them in the **Split-N** BP-Method with the following code:

```
i = 0
while i < numOfCopies
do
    // create a copy of the process instance
    copy= clone(this)

    // Get ready for next loop
    i = i + 1
end
```

Join activities can also access instance copies by using **copy.** as shown below:

```
// Set the original instance variable with the
// copy instance variable
bidTotal = copy.bidTotal
```

Preconditions

Split-n activities require an incoming instance from another activity in the process.

Join activities rejoin each instance copy reaching the Join activity (if the original instance is still there).

Post-conditions

Split-n: When an instance reaches a Split-n activity, the original instance is automatically sent directly to the corresponding Join activity. While still in the Split-n activity, instance copies are created

and each copy flows across the path in the Split-n/Join activity circuit. The number of generated copies will depend on the number of copies generated using the variable "*copy*".

Join: The original instance can leave the Join activity due to one of these three reasons:

- If the **number of copies to wait to release** property is set, the original instance leaves the Join activity after that number of copies have arrived to this activity. If it is not set, then,
- After all instance copies have reached the Join activity, the instance moves to the activity following the Join activity according to transition rules.
- When a copy reaches the Join activity and the Join's Method sets the predefined variable *action* to *RELEASE* (action = RELEASE).
- When the original instance expires either because of a due transition or the process' Instance Expiration exception handling has occurred due to a missed process deadline.

Properties

All Split activities must have the corresponding Join activities.

SPLIT N

Activity ID

See Creating an Activity.

Advanced Properties

Maximum number of simultaneous copies: determines the number of copies that can flow simultaneously within the circuit. And at runtime *any number of copies above this maximum will be queued and later released when one of the already flowing copies reaches the Join activity or is aborted.*

For example, if you set the maximum number of copies to 10, and then the FBL generates 100, the last 90 will be queued until a copy arrives at the Join activity or is aborted. When this happens, one of the 90 copies will be released. A use case on when to define limits is if your process needs to upload **100** files and has only **5** connections available. Therefore, each file to upload is a copy (**100** instance copies are generated) but the number of simultaneous uploads is limited to the number of connections, so the *Maximum number of simultaneous copies* is set to **5**.

Note



Any Interactive activity added within the circuit can not be *Suspendable*.

Generate Events: See Generate Events.

JOIN

Activity ID

See Creating an Activity.

Advanced Properties

Aborted Copies: You can select where the instance copy will go if it is aborted.

The instance copy will be routed to the chosen activity (Join or End) if:

- The instance copy is aborted within the split-join because of the process design.
- The instance copy is aborted by the user within the split-join.
- The instance copy throws an exception in the split-join and then process participant aborts it.

- If the instance is sent to the Join activity, the Join BP-Method will be executed for the copy.
- If the instance is sent directly to the end, it will not go through the Join activity. Therefore, no BP-Method is executed.

Amount of copies to wait to release : you can set how many of the instances created in the Split-N activity should be waited for before the instance waiting in the Join activity moves forward in the process.

Generate Events: See Generate Events. In particular, the Generate events property is automatically enabled if the associated SplitN activity is also enabled.

Transitions and Split-n and Join activities

Split-n activities require at least one or more incoming transition(s). Only one outgoing transition is allowed.

Join activities must have only one incoming transition in a Split-n / Join circuit. One or more outgoing transition(s) are required.

Tasks

No tasks can be generated. A BP-method will be automatically created with the same name as the one of the activity.

The Join Activity associated script will be automatically generated after you check the design. At that moment the Split-Join circuit is consolidated.

Business Process Method considerations

The Split-n / Join activity circuit creates copies of the original instance based on BP-Method logic. The Split-n activity creates the copies as shown in the following BP-Method:

```
i = 0
while i < numOfSuppliers
do
    // create a copy of the process instance for
    // this supplier's quote
    copy= clone(this)
    copy.supplier = "Supplier" + String(i)
    copy.supplierNum = i

    // Get ready for next loop
    i = i + 1
end
```

Note



If the instance has associated a Separated Instance variable, its value will not be automatically copied to the copies separated instance variable. Within the Split-N BP-Method, you have to assigned it manually (e.g., *copy.separatedvariable = separatedvariable*).

The Join activity acts as a marshaling point where each copied instance is used to update variables in the original instance. For example, the Join activity's BP-Method will contain lines like the following:





```
// Take the copy instance variables and use them to set
// original instance variables. Use 'copy.' .

supplierName = copy.supplierName
costQuote = copy.supplierQuote
```

Conditional

The Conditional Activity helps you centralize different paths of the process into one activity and re-distribute the instances based on conditional transitions.

Conditional activity characteristics

Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			

Work Portal and the Conditional activity

Conditional activities do not appear in Work Portal because the Conditional activity does not require any end user intervention.

Roles and Conditional activities

Conditional activities can appear in either automatic roles or the user defined role types. However, if the Conditional activity is in a user-defined role, it will not appear in Work Portal.

Variables and Automatic activities

No variables are available.

Preconditions

The Conditional activity is activated by an incoming instance from another activity in the process.

Post-conditions

The instance flows to the next activity in the process according to transition rules.

Transitions and Automatic activities

One or more incoming and outgoing transitions are required. Only Unconditional and Conditional transitions are available.

Tasks

No tasks are available.

Business Process Method's considerations

No BP-Method is available.

Organizations Interaction




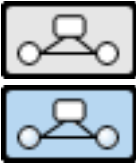
Subprocess

Subprocess activities are used to call a subprocess, which is a different process that can exist internally to your organization or in a separate organization. See IPC activities - Organizational interaction for further information.

Subprocess activities:

- Make a complex process more easily understood using the Subprocess activity to abstractly represent the underlying process.
- Enable reuse - Subprocess activities (and therefore the underlying processes they represent) can be easily reused by many calling processes.
- Enable Business-to-Business (B2B) communication between processes. When a process calls another process through a Subprocess activity, the called process can be run in another company behind their firewall.
- Balance load, you can design subprocesses to distribute their execution among different servers to improve the performance and balance resources load.

Subprocess activity characteristics

Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			

Work Portal and the Subprocess activity

The Subprocess activity is not visible in Work Portal as any other automatic activity.

Roles and the Subprocess activity

Subprocess activities can reside in user-defined and automatic role lanes.

Variables and Subprocess activities

Instance variables can be used to fill in the arguments, as constant values or expressions can be used.

Preconditions

Subprocess activities expect an incoming instance from another activity in the process. The Subprocess activity's "create part" sets the called subprocess' incoming argument variables through the argument mapping. A new instance is created in the called subprocess.

Post-conditions

The Subprocess activity's "termination part" sets the returning argument into instance variables through the argument mapping.

When the *child* instance finishes at its end activity, a notification is received by the subprocess activity and the *parent* instance variables are set based on the Argument Mapping. The *parent* instance moves to the next activity according to transition rules.

Properties

Activity ID

See Creating an Activity.

Related Process

Dynamic Process Invocation: the target process can be dynamic. In this case you do not define a Process as a target but a Process Interface instead. The real process to be called must match the process interface and its name is passed as the argument **targetProcessName** in the Argument Mapping Set. See Dynamic Process Call for detailed information.

Target Process Name: specifies the process to be used for the subprocess. The selected process determines the arguments to pass at creation time and the arguments to receive at termination time, as defined in the Argument Mapping.

If the process has not been created yet, click **New** to create a new process or you can define it later on. At *Design check* time, an error will populate indicating that the Related Process has not been defined and you can add it from the error panel to fix it.

If you select **New**, you are guided to create the process through a wizard. This option facilitates all instance variables creation, arguments creations and argument mapping. For further information see Generate Processes, Procedures and Screenflows automatically.

In Business to Business (B2B) scenarios, the external company must send the subprocess or the process interface to your company. If the external company wishes to keep its process private, they can simply send a Process Interface, which displays only the IPC activities as the Begin, End and any Notification Wait activities and the expected incoming and outgoing argument variables associated to each activity.

Argument set name: select within the different mapping names defined by the Related process to be used when calling it.

Tip

If you drag a **Process** from the *Project tree* and drop it on a Subprocess activity, you can dynamically define this process as the Target process of the Subprocess.

Advanced Properties

Attachments can be visible to related processes: select this option if you wish bound attachments to be visible to the subprocess. This option is disabled by default.

As each activity is processed in Work Portal, the user has the option to append file attachments to the instances as they flow through the process. These attachments may be then opened and edited by each subsequent participant in the process. When an activity receives an instance, the end user may check to see whether an attachment has been added. Any added attachment will be displayed in Work Portal showing the name of the attachment, the version, who created the attachment, a modified file name and whether the attachment has been locked by a previous user.

Process notification immediately: this is considered when the called process finishes and notifies the Subprocess activity that it has finished. If this property is **enabled**, the called process will wait until the Subprocess activity sets the instances variables as defined in the argument mapping and routes the instance. Once these actions are successful, the child instance in the called process is completed. If any of them fails, the Server will retry as many times as defined in the Server's properties.

If the notification immediately property is **disabled**, then the child instance will only send the notification and will not wait for the Subprocess success. It is set as completed. The notification will be stored in a queue and the arguments will be processed as soon as the Server is able.

Generate Events: See Generate Events.

Transitions and Subprocess activities

Subprocess activities have one or more incoming transitions and one or more outgoing transitions.

Tasks

No tasks are available.

Business Process Method's considerations

No scripting is available. To pass arguments between processes, the Argument Mapping function is available.

Process Creation




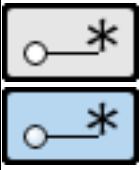
Process Creation activities are used to invoke another process asynchronously. Process Creation activities are used for the following reasons:

- To speed completion of tasks by simultaneously running more than one process.
- To make a complex business process more easily understood by using a Process Creation activity to abstractly represent the underlying process.
- To enable reuse - called processes can easily be reused by many calling processes.
- To enable Business-to-Business (B2B) communication between processes. As a process calls another process through a Process Creation activity, the called process can be run in another company behind a firewall.
- To Balance load. You can design subprocesses to distribute their execution among different servers in order to improve the performance and balance resources load.

Though a Process Creation activity is similar to a Subflow activity in that they both call a subprocess, the Process Creation activity is slightly different from a Subflow activity in the following ways:

- It invokes a called subprocess asynchronously, which means that the calling process does not have to wait for the called process to finish before moving forward.
- It is responsible for invoking the called subprocess and the called sub process has no requirement to return to the calling process. If this is desired, a Termination Wait activity is used to capture the return.

Process Creation characteristics

Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			

Web Portal and the Process Creation activity

The Process Creation activity is not visible in Web Portal.

Roles and the Process Creation activity

Process Creation activities can reside in an abstract and in automatic role lanes.

Variables and Process Creation activity

Process Creation activity can define instance variables to determine the argument mapping.

Preconditions

Incoming instance from another activity in the process. The Process Creation activity's Argument Mapping can set the called subprocess' incoming arguments, if applicable, with the local instance variables. A new instance is created in the called subprocess.

Post-conditions

As the called subprocess is invoked and the 'child' instance is successfully created, the instance moves to the next activity in the calling process according to transition rules.

Properties

Activity ID

See Creating an Activity.

Related Process

Dynamic Process Invocation: the target process can be dynamic. In this case you do not define a Process as a target but a Process Interface instead. The real process to be called must match the process interface and its name is passed as the argument **targetProcessName** in the Argument Mapping Set. See Dynamic Process Call for detailed information.

Target Process Name: specifies the process to be used for the subprocess. The selected process determines the arguments to pass at creation time and the arguments to receive at termination time, as defined in the Argument Mapping.

If the process has not been created yet, click **New** to create a new process or you can define it later on. At *Design check* time, an error will populate indicating that the Related Process has not been defined. You can add it from the error panel in order to fix it.

If you select **New**, you are guided to create the process through a wizard. This option facilitates all instance variables creation, arguments creations and argument mapping. For further

information, see Generate Processes, Procedures and Screenflows automatically.

Argument set name: select within the different mapping names defined by the Related process to be used when calling it.

Tip



If you drag a **Process** from the *Project tree* and drop it on a Process Creation activity, you can dynamically define this process as the Target process of the Process Creation.

General

Attachments can be visible to related processes: Any attachments added to an instance in the parent process are visible to users in the subprocess if this check box is selected.

If you check this option you have to consider that:

- Both, parent and child we be looking at the same attachment, therefore any modification to it will be visible for both.
- Since the parent and subprocesses run asynchronously, if the instance in the parent process finishes the End activity before the instance in the subprocess, all attachments will immediately no longer be available. This also happens the other way around with subprocess attachments, if the subprocess finishes before the parent process.

If you have Archiving enabled and set to archive attachments at deployment time, the parent/child attachments will be available in the archiving.

If you do not check this option and pass the attachment to the parents/child as an argument, you have to consider that:

- If the parent instance is completed and gets to the end, the child

instance will still have the attachment available. The same happens the other way around, if the child creates the attachment and later on finishes its flow.

- As the attachment was "copied", any change to any of both copies will not be available to the other instance.
- If the attachment is big, then as it was duplicated, the process will deal with two heavy attachments.

Keep relation with child process: Maintains a link between corresponding instances in the parent process and the subprocess. This check box needs to be selected if you are including a Notification Wait or Termination Wait activity to rejoin the instances at some point later in your process design. If the parent process does not need information from instances in the subprocess, you can clear this check box.

If this field is enabled, the parent will only keep relation to **one** child for that activity. There is no need to indicate the instance while notifying or waiting for notification as there is only one possibility, and the Server is aware of it.

If this field is disabled, it works as "fire and forget". The instance is created but no relationship is kept. The parent can generate as much children as required and it is not notified as its children arrive at the end in their own process. If certain communication is required between them, parent or child will have to handle external notifications by using the Notification Wait activity and the Process Notification activity.

Generate Events: See Generate Events.

Transitions and Process Creation activity

Process Creation activities have one or more incoming transition(s) and one or more outgoing transition(s).

Tasks

No tasks are available.

Business Process Method's considerations




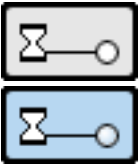
No scripting is available. To pass arguments between processes, the Argument Mapping function is available.

Termination Wait

The Termination Wait activity gives an optional synchronization point in a calling process for a called subprocess only if the "keep relation with child" property is enabled in the Process Creation activity. It is always used in combination with the Process Creation activity.

The combination of having Process Creation and Termination Wait activities in a process is very similar to that of using a Subflow activity. The advantage to the combined Process Creation and Termination Wait activities is that several activities can occur between the two of them while the subprocess is running. Termination Wait activities are optional and only need to be used if you wish to halt processing an instance until the called subprocess completes.

Termination Wait Characteristics

Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			

Work Portal and the Termination Wait Activity

The Termination Wait activity is not visible in Work Portal.

Roles and the Termination Wait Activity

Termination Wait activities can reside in user-defined and automatic role lanes. However, if the Termination Wait activity is in a user-defined role, it will not appear in Work Portal.

Variables and Termination Wait Activity

The Termination Wait activity can define instance variables to determine the argument mapping received from the End activity of the called process.

Preconditions

Somewhere in the process previous to a Termination Wait activity, there must be a corresponding **Process Creation** with *keep relation with child* enabled.

The Termination Wait activity has defined which Process Creation activity it is related to.

Termination Wait activities expect an incoming instance from another activity in the process. Once reached, the instance will wait there until the subprocess called from the Process Creation activity reaches its End activity.

Post-Conditions

When an instance reaches the End activity in the called subprocess, it is returned to the Termination Wait activity in the calling process. Instance variables are updated based on the defined Argument Mapping. The instance then continues on through one of the transitions leaving the Termination Wait activity.

Properties

Activity ID

See Creating an Activity.

General Category

Creation Activity: select the Process Creation activity which the Termination activity corresponds to.

The subprocess associated to the Process Creation determines the arguments returned (found in the subprocess' End Argument Mapping.) Based on the Argument Mapping defined in the Termination Wait activity, the received arguments set the instance variables defined and become available within the process.

Process notification immediately: speeds up the notification process. When this option is selected, the receiving activity's argument mapping will be run upon receipt of the notification. If it is disabled, the notification will be stored in a queue and its argument mapping will be processed in the order that it was added to the queue. This option is selected by default.

Generate Events: See Generate Events.



Transitions and Termination Wait Activity



Termination Wait activities have one or more incoming transition(s) and one or more outgoing transition(s).

Business Process Method Considerations

No scripting is available. To pass arguments between processes, the argument mapping function is available.

Process Notification

Process Notification  and Notification Wait  activities work together to allow communication between processes.

Instances wait at a Notification Wait  activity until they receive notification from a corresponding Process Notification activity .

The Process Notification activity informs the instance waiting in a Notification Wait activity when the related instance has arrived at the activity. This notification releases the instance from the Notification

Wait activity to continue its flow in the process. The instances in both processes can either be related (sent by a Process Creation activity with the **Keep relation with child process** property selected) or not. If the instances are not related, some tracking means must be used (for example, a relational database) in order to match the instance ID of the instance you want to notify.





There are three scenarios where you can use the Process Notification and Notification Wait combination:

- Parent/Child relationship.
- Child/Parent relationship.
- External relationship.

See Notification Wait activity for further information.

Process Notification activity characteristics

The Process Notification activity's *Notification Category*, found in the Activity Property dialog box, specifies the name of the process and the specific Notification Wait activity to communicate back to. Right-click on the Process Notification activity and select **Properties** from the shortcut menu. Select the **Notification Category** and then select the parent process. You can then select the name of the Notification Wait activity from the **Activity** drop-down menu.

Classic Icons	UML Icons	Business Analyst Icon	BPMN Icon
			 /

Work Portal and the Process Notification activity

The Process Notification activity is not visible in Work Portal.

Roles and the Process Notification activity

Process Notification activities can reside in user-defined and automatic role lanes. However, if they are in user-defined roles, they will not appear in Work Portal.

Variables and Process Notification activity

The Process Creation activity can define instance variables to determine the Argument Mapping.

Note



Arguments are created in the Notification Wait activity.

Preconditions

There must be at least one incoming transition to the Process Notification activity.

Post-conditions

When an instance reaches a subprocess' Process Notification activity, it immediately sends a notification message to the corresponding Notification Wait activity. The message contains the arguments the Notification Wait activity is expecting (defined in the Argument Mapping. The subprocess' instance then continues on through one of the transitions leaving the Process Notification activity.

The notification will wait for the success of the delivery before going through the right transition to the next activity.

Properties

Activity ID

See Creating an Activity.

Notification

Target Process Name and Activity: specifies the process to be notified by the Process Notification. Within this process, select the Notification Wait activity to notify.

The selected activity determines the arguments to receive while notification is defined in the Argument Mapping.

If the process has not been created yet, click **New** to create a new process or you can define it later on. At *Design check* time, an error will populate indicating that the Target Process/Activity has not been defined and you can add this information from the error panel fixing it.

Argument set name: select within the different mapping names defined by the Wait Notification activity to be used when notifying it.

Tip



If you drag a **Process** from the *Project tree* and drop it on a Process Notification activity, you can dynamically define this process as the Target process of the Process Notification.

Advanced Properties

Generate Events: See Generate Events.


Transitions and Process Notification activity

There can be one or more incoming transition(s) and one or more outgoing transition(s).

Business Process Method considerations

No scripting is available. To pass arguments between processes, the Argument Mapping function is available.

Notification Wait


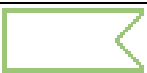

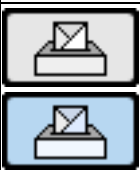
 activity waits in a process until it receives a notification from one of the following:

- A Process Notification activity in an alternate process based on a Child/Parent Relationship or a Parent/Child Relationship.
- An activity using the *send* method from the *Notification* standard component.
- An external program using the Process Application Program Interface (PAPI). For further information, see *External Notification* on this page.

Notification Wait activity characteristics

Instances wait in a Notification Wait activity for a Process Notification activity or an external event to send a message to the Notification Wait activity before work flow continues. The exception to this rule is that due transitions can be added to a Notification Wait activity. This means that when the time for the due transition expires, the instance flows through the due transition path instead of continuing to wait for the message or event.

Any notification for an instance that has not reached the notification wait activity yet will wait there until the target instance arrives.

Classic Icons	UML Icons	Business Analyst Icon	BPMN Icon
			 /

Work Portal and the Notification Wait activity

The Notification Wait activity is not visible in Work Portal.

Roles and the Notification Wait activity

Notification Wait activities can reside in user-defined and automatic role lanes.

Variables and Notification Wait activity

The Notification Wait activity can define instance variables to determine the Argument Mapping.

The different argument mappings that can be defined are the incoming information from the Process Notification activity in another process or a notification from an external application using PAPI to communicate with the Notification Wait activity.

The chosen argument set (see Message Based transition.) can be the one that will decide the next activity for the notified instance.

Preconditions

An instance will wait at the Notification Wait activity until a subprocess' Process Notification activity sends a message back to the waiting instance or the Notification Wait activity's due transition logic expires. The subprocess' Process Notification activity should set the required arguments defined in the Argument Mapping as expected by the Notification Wait activity.

A Notification Wait activity can also wait for an external notification, which could be from either of the following:

- Any activity in any process that uses the *Notification* component to send notification to the Notification Wait activity.
- An external program that uses PAPI to notify the Notification Wait activity.

Post-conditions

When an instance leaves a Notification Wait activity, it continues on through one of the transitions leaving the Notification Wait activity following transition rules. (see Message Based transition)

Properties

Activity ID

See Creating an Activity.

General Category


Creation activity: select the Process Creation activity to which the Notification Wait activity will respond. It applies if the expected notification comes from a Child process.

Waits for (type of event):

- Parent process: The Notification Wait activity waits for some kind of notification from a Process Notification activity in a parent process ("Keep relation with child" property is enabled in the Process Creation to which the Notification Wait corresponds.)
- Child process: The Notification Wait activity waits for some kind of notification from a Process Notification activity in a child process. The **Creation activity** determines the Process Creation activity within the process. To achieve this scenario, the **Notification Wait** activity has to be located between the **Process Creation** activity and the **Termination Wait** activity -this last one, if applicable- as **Notification** is expected from an instance generated in a child process (the process which the Process Creation calls) ("Keep relation with child" property is enabled.)
- External event: The Notification Wait activity waits for notification from an external program using PAPI (for further information, see the PAPI Javadoc distributed with the product) or an activity in an external process using the *send* method from the *Notification*

component (For further information, see the **Notification Component** documentation.)

Allows interruptions: the Notification Wait activity will behave as an exception catching activity with a remote control connected to the instance. An activity's BP-Method in the subprocess will send a notification with unique information to identify the correct instance to notify. This notification will be taken as an interruption and the instance will move from wherever it currently is and will go directly to the Notification Wait activity unless it has already reached the End activity. For further information, see *Allowing interruptions* on this page.

If this check box is selected, the Notification Wait activity appears with an *interruption* sign such as a small circle in its center in the Classic theme or a thunderbolt in the image (BPMN theme - ).

This Activity can't be in the main flow.

Process notification immediately: speeds up the notification process. When this option is selected, the receiving activity's argument mapping will be run upon receipt of the notification. If it is disabled, the notification will be stored in a queue and its argument mapping will be processed in the order that it was added to the queue. This option is selected by default.

Once the Notification has been processed, any other notification for the same instance is dismissed when the instance arrives at the end. It means that if two notifications for the same instance are sent, the first one to arrive will notify the instance and the second one will have no effect, unless the instance passes through this activity again.

Generate Events: See Generate Events.

Transitions and Notification Wait activity

Notification Wait activities typically have one or more incoming transitions and one or more outgoing transitions. They may have an

outgoing Due transition to allow instances to automatically transition out of the Notification Wait if it does not receive a notification within a specified time period.

If the Notification Wait activity has multiple sets of arguments, the Message based transitions are available. See Message based Transition.



If the Notification Wait allows interruptions, it will reside in its own flow and will not have an incoming transition.

Business Process Method's considerations



No scripting is available. To pass arguments between processes, the Argument Mapping function is available.

Parent/Child Relationship


In a Parent/Child relationship, the parent process contains a Process Creation activity that calls the Child process.

- The Process Notification  activity always implicitly communicates with a Notification Wait  activity in the calling process. It can only communicate back to the process that called it.

Child/parent relationship

This relationship is essentially the same as the Parent/child relationship with the exception that, by adding a new Notification Wait  activity to the child process and a Process Notification  activity to the Parent Process, the two processes can communicate with each other. This way, communication can be bidirectional between processes.

Note

 As the **Notification Wait** will expect the notification from an instance in an activity in a child process, the **Notification Wait** activity should come after a **Process Creation** activity generating an instance in the child process.

External notification


In an external notification relationship, Notification Wait activities can wait for notification from the following:

- An external program connecting through PAPI. PAPI is the Application Program Interface that allows external services to communicate with FuegoBPM processes. For further information, read the PAPI javadoc document distributed with the product.
- An activity in a process that is not a subprocess (child process.) This activity does not need to be a Process Notification activity. Using the Fuego standard component **Notification** and its **send** method, you can send a notification to any process instance from any activity type.

In the external program scenario, a process includes a Notification Wait activity that has been created using the **External event** option on the Activity General Category.

In the **Waits for:** field, **External event** is selected to indicate that the activity is waiting for an event that is external to the process.

Note

 A process that is not a subprocess (child) of the process which contains the Notification Wait activity is also considered an **External event**.


Allowing interruptions


A Notification Wait activity in your process indicates that all instances should wait in the activity for a specified time period only

moving forward in the process if a notification is received or if due transition logic is enacted. This may not always be the best process design. What if you want to notify the process design only under certain conditions? For example, only in the event that customer has changed or cancelled an order? This kind of scenario does not require a Notification Wait directly in the main flow of the process design.

The **Allows interruptions** check box in the **Activity General Category** dialog box is used when you need to pull instances from a process flow at any time under certain conditions.


Note

 If you are performing an external notification, you must indicate the instance Id you want to pull. If the notification is generated from a Process Notification, define the **peerInstanceID** with the Instance ID in the Argument Mapping.

Notification Wait activities that allow interruptions do not reside in the main process flow (between the Begin and End activities.) Instead, the activity resides in its own flow. The allows interruptions Notification Wait is shown with an *interruption* sign such as a small circle in its center in the Classic theme or a thunderbolt in the image (BPMN theme). .

When a notification is received by the Notification Wait activity, it is receiving all information about the instance that should be notified. The FuegoBPM Server finds the instance that matches the information received in the notification. The instance is then pulled from the main flow and routed to the Notification Wait flow.

Note

 Remember to return the instance routed to the Notification Wait flow back to the main flow. In the last activity within this parallel flow use the **BACK** or **SKIP** actions in the BP-Method

Dynamic Process Call

When you design a process and it needs to call a subprocess or a procedure you can choose which one to execute at runtime.

At designing time you define the process interface to be used but not the specific process by its name.

Dynamic Process Call applies to **Subflow** and **Process Creation** activities, as well **Dynamic Procedure Call** applies to **Procedures**.

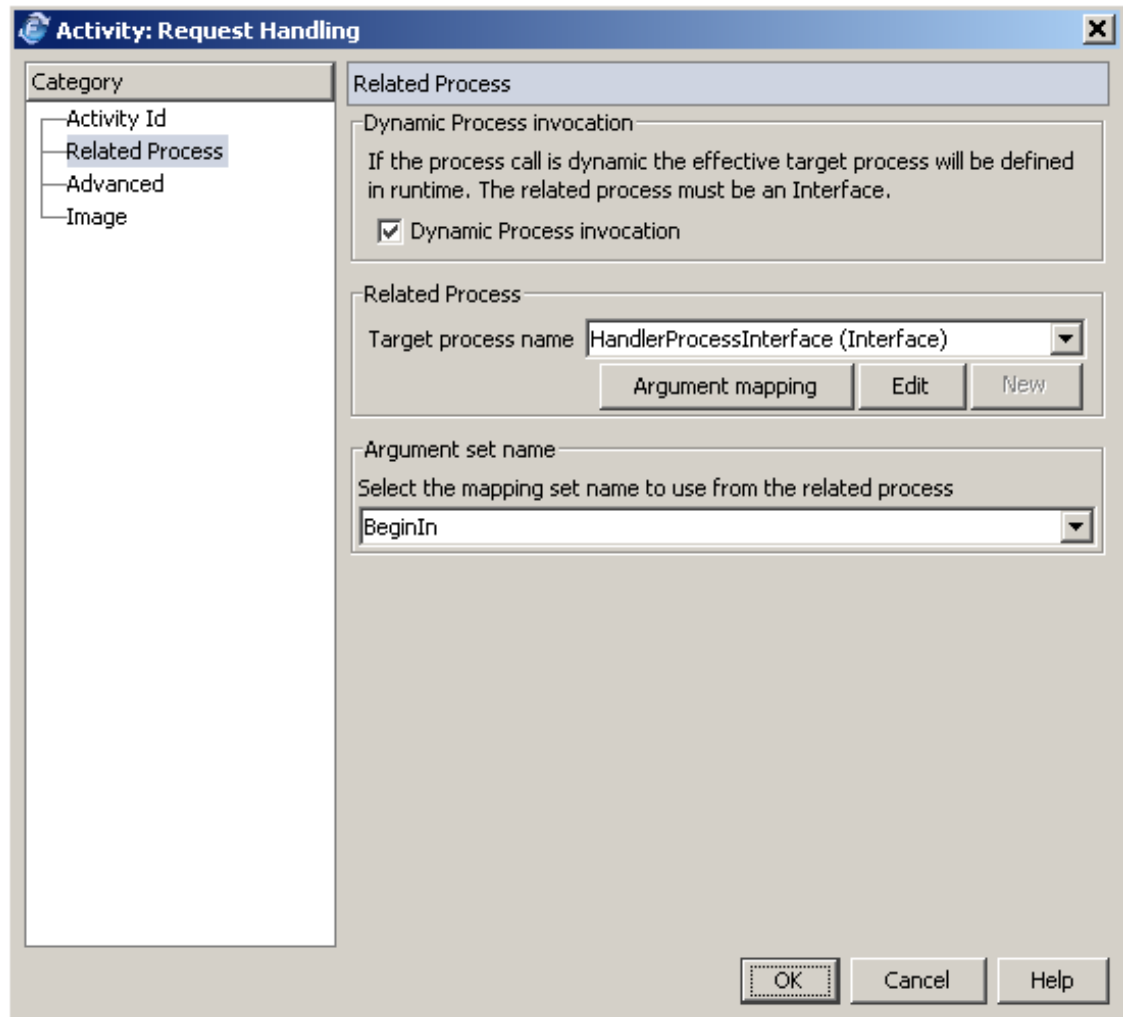
The related process/procedure in any of the above can be static or dynamic.

The **static** option requires you to define the target process/procedure name when you are designing.

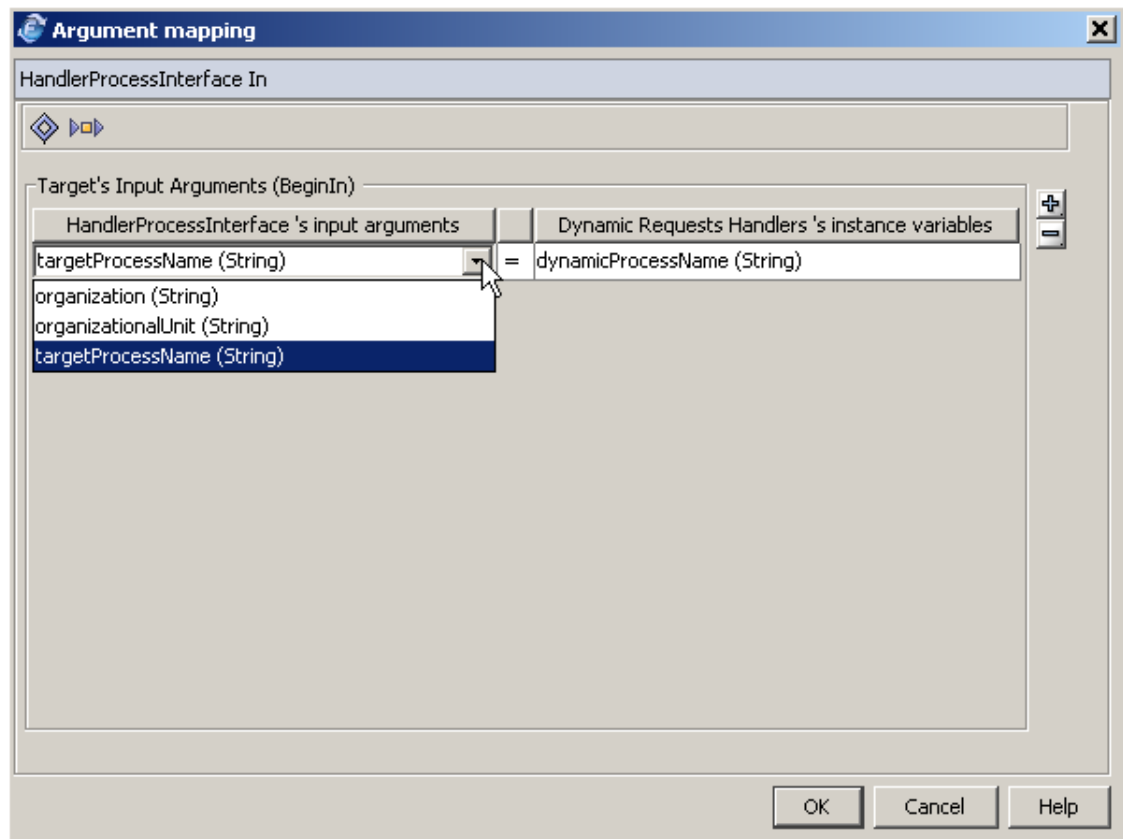
The **dynamic** option requires you to define the target process interface. The specific process to call is determined dynamically through an argument that represents the name of the process/procedure. This argument is the **targetProcessName** available in the Argument Mapping.

Subflow and Process Creation

When you define the **Related Process**, turn on the **Dynamic Process invocation** check box. Complete the **Target process name** with a process interface (you need to previously import the interface to be used).



Then map the argument **targetProcessName** to the instance variable that will contain the name of the process at runtime.



Procedure

When you implement an automatic activity as a Procedure, and you define the **Related Process**, turn on the **Dynamic Process invocation** check box. Complete the **Procedure name** with a procedure interface (you need to previously import the interface to be used).

Main task - Activity Preprocess request

Preprocess request

Implementation type
Procedure

Dynamic Process invocation
If the process call is dynamic the effective target process will be defined in runtime. The related process must be an Interface.
☒ Dynamic Process invocation

Related Process
Procedure name ProcedureInterface (Interface) Edit New

Choose the argument set to use in the selected procedure
Argument set name
BeginIn Argument mapping

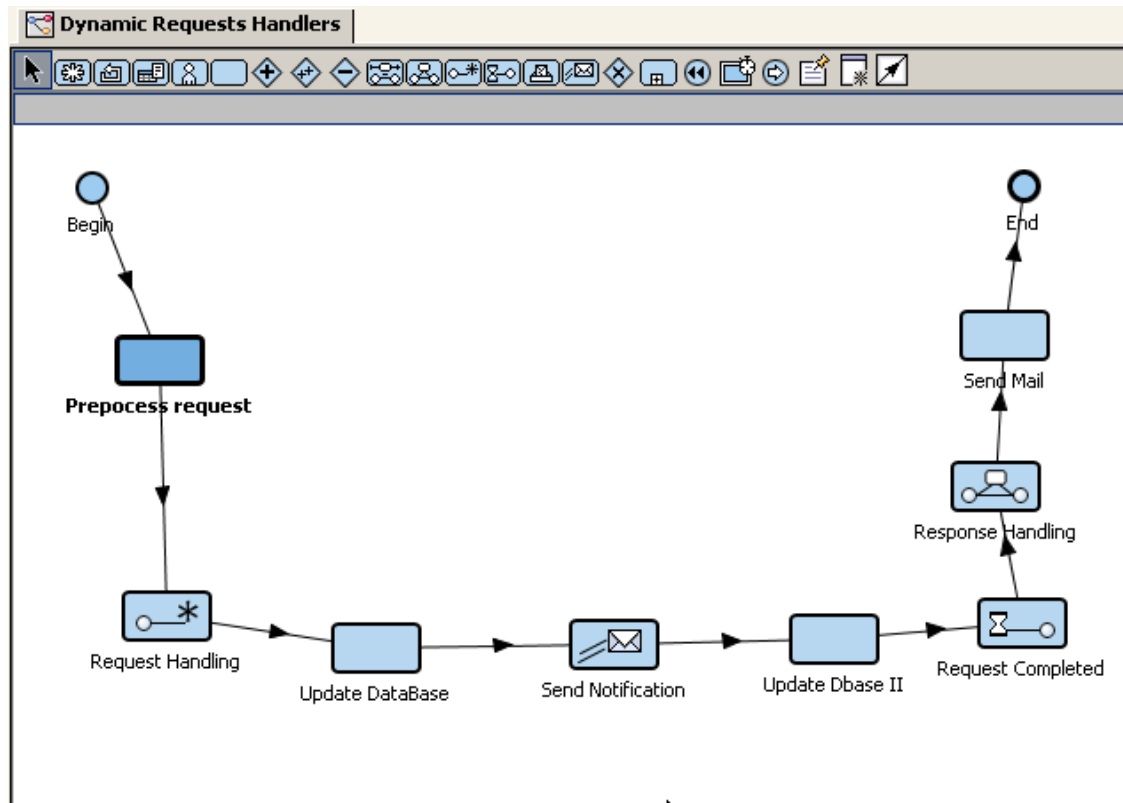
OK Cancel Help

Then map the argument **targetProcessName** to the instance variable that will contain the name of the procedure at runtime.

Dynamic Call Examples

Request Controller and Dispatcher

This example is about a call center that receives requests from the customers.



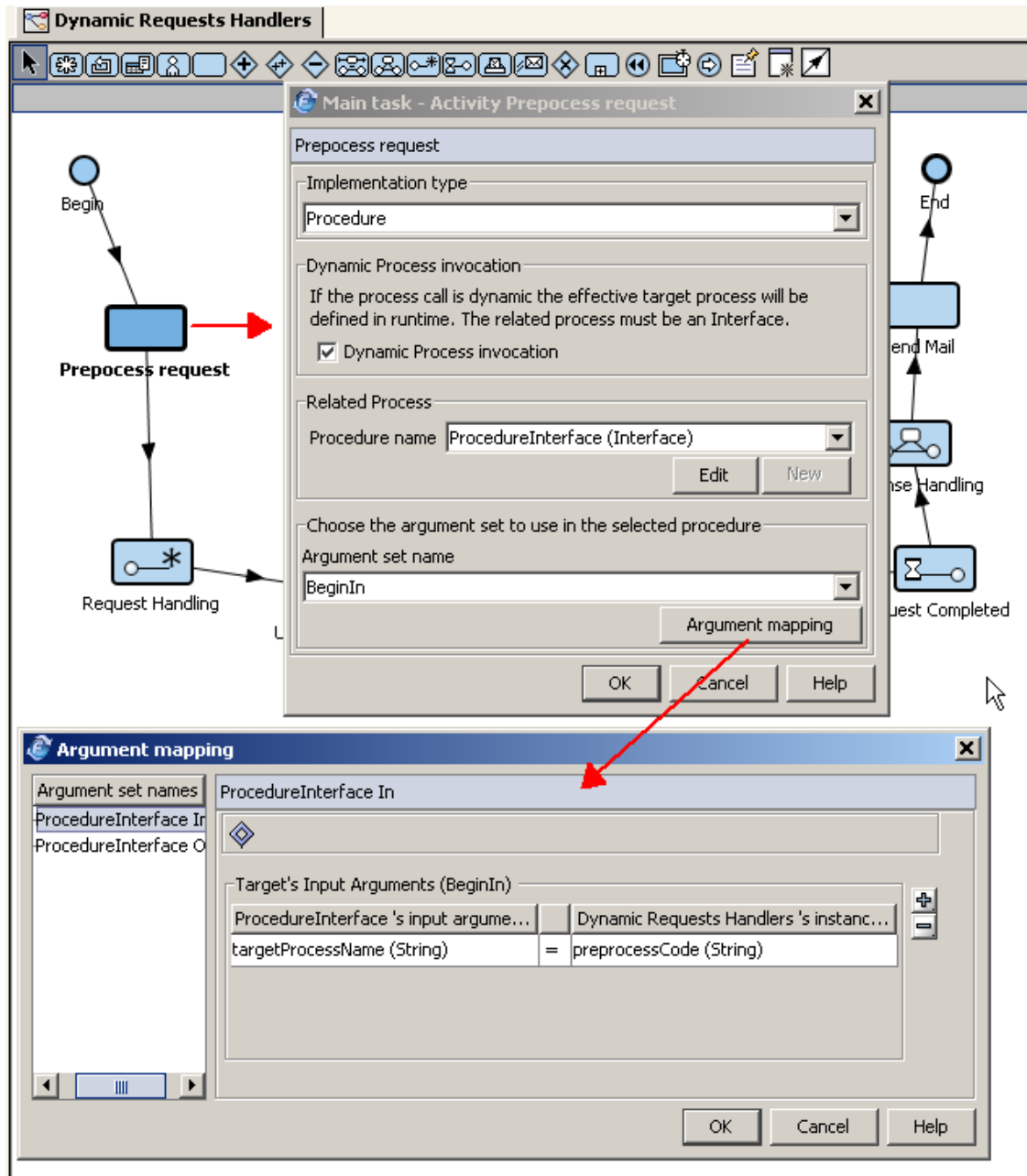
Each request initiates an instance that is preprocessed to complete certain information that is needed along the main process.

Preprocess Request activity: Procedure example

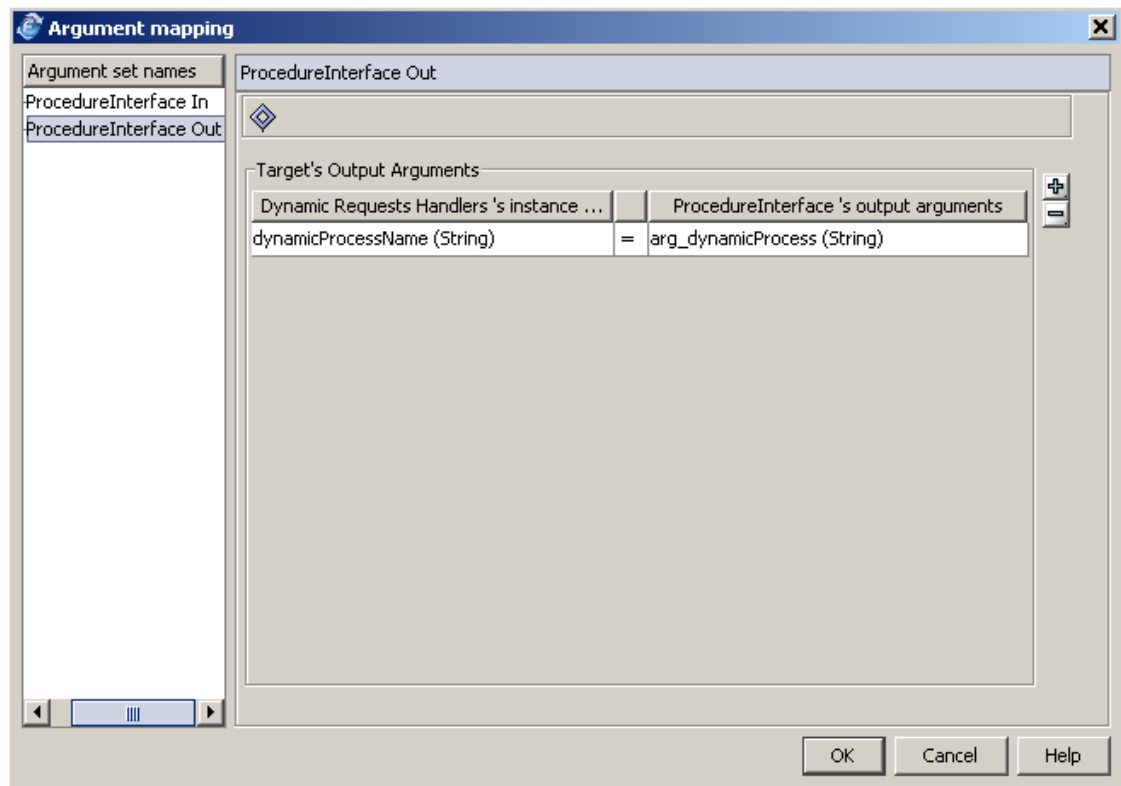
The **Preprocess request** activity's main task is a procedure. Different procedures are executed based on certain code that comes in the request (instance). The procedure returns the name of the process that later will be called.

The procedure is implemented with **Dynamic Process Invocation**.

The different implemented procedures are called based on the instance variable **preprocessCode** that contains the name of the procedure to execute.



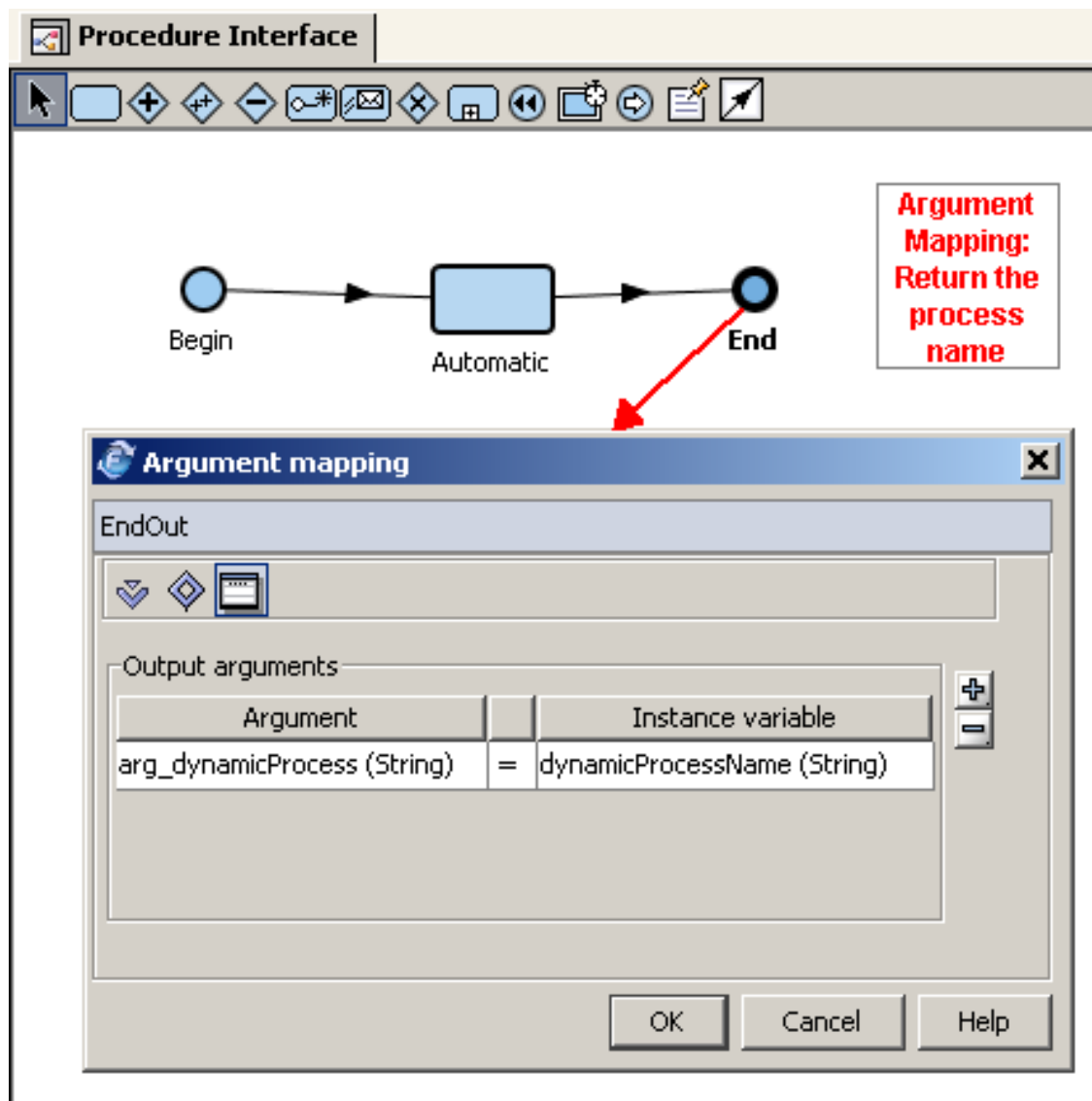
The procedure returns the name of the process that later will be called.



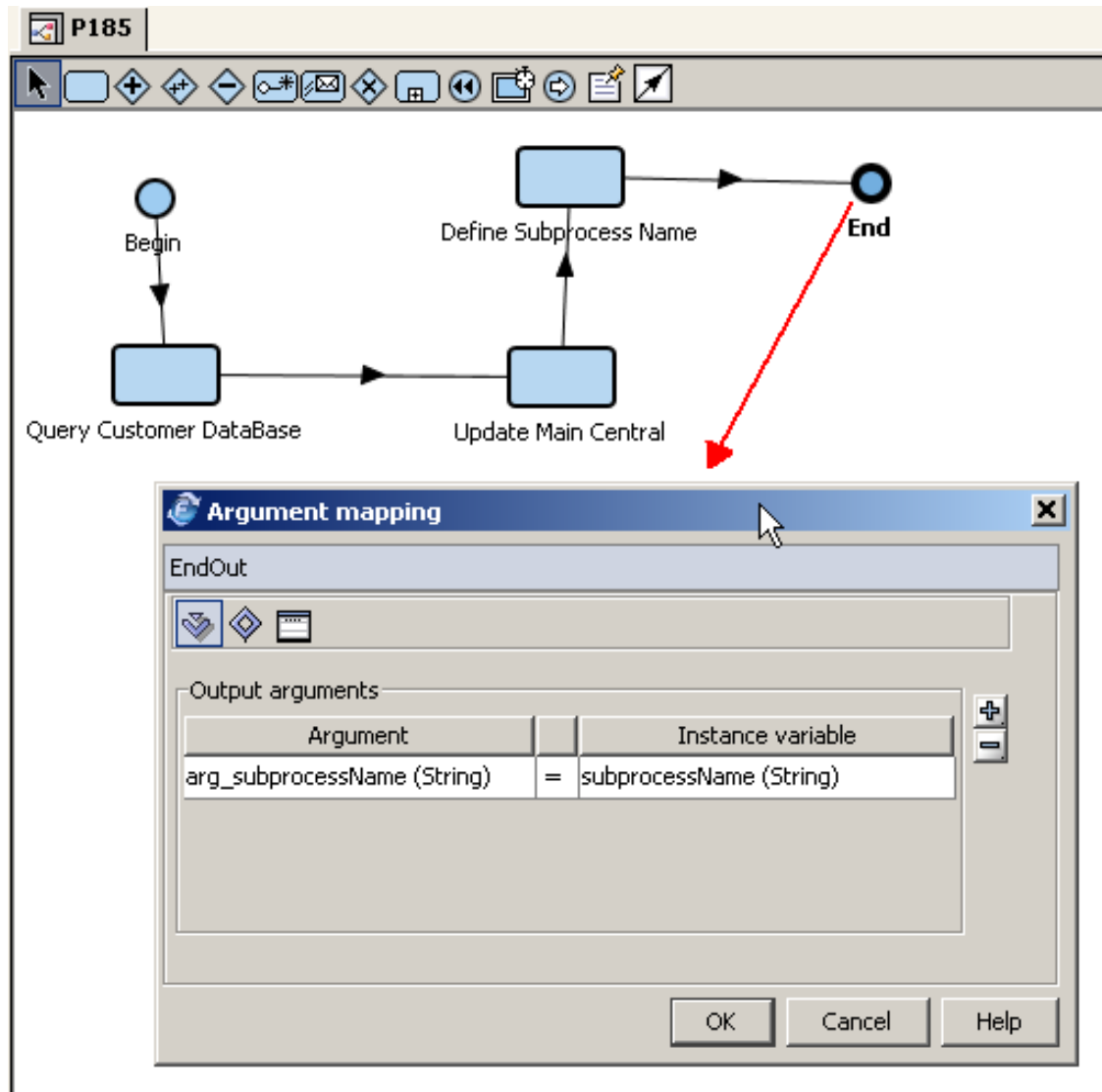
In the example the **preprocessCode** instance variable can contain the value "**P185**" that indicates that procedure P185 has to be executed.

You have to design the Procedure template, generate the Process interface for it and import the interface into the project. Each invoked procedure can do different things as long as they match the interface.

Procedure Interface:



Procedure P185 that matches the Procedure Interface:



The **Procedure P185** executes different tasks but has to match the defined interface returning the Subprocess Name.

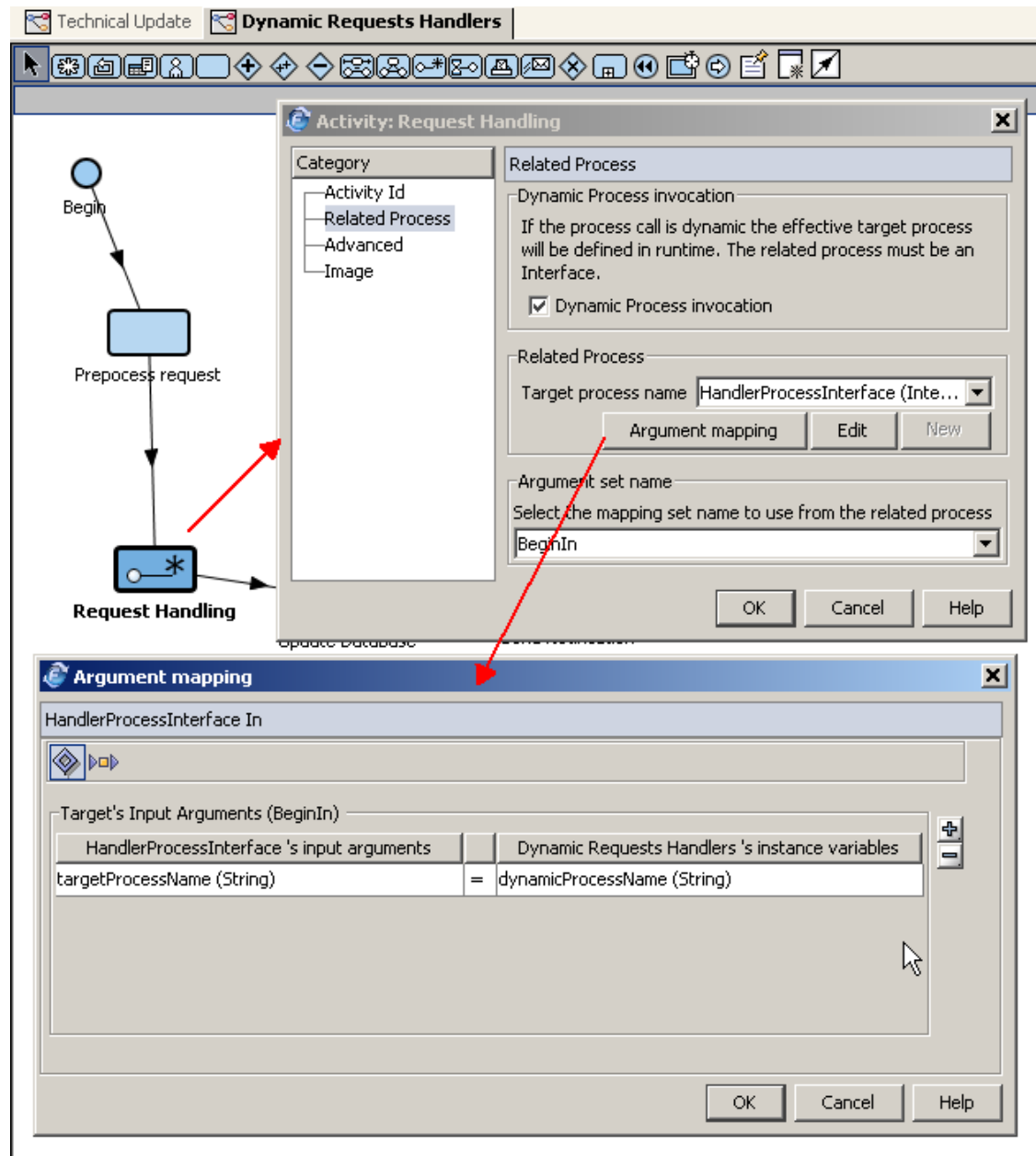
The advantage of implementing this case with a Procedure Dynamic Call is that if new procedures need to be implemented all you need is to add the procedures to the project but the main process does not need to be modified.

Request Handling activity: Process Creation example

The **Request Handling** subprocess also dynamically calls to different processes based on the returned value of the procedure.

The Process Creation activity is implemented with **Dynamic Process Invocation**.

The different implemented processes are called based on the instance variable **dynamicProcessName** that contains the name of the process to execute.

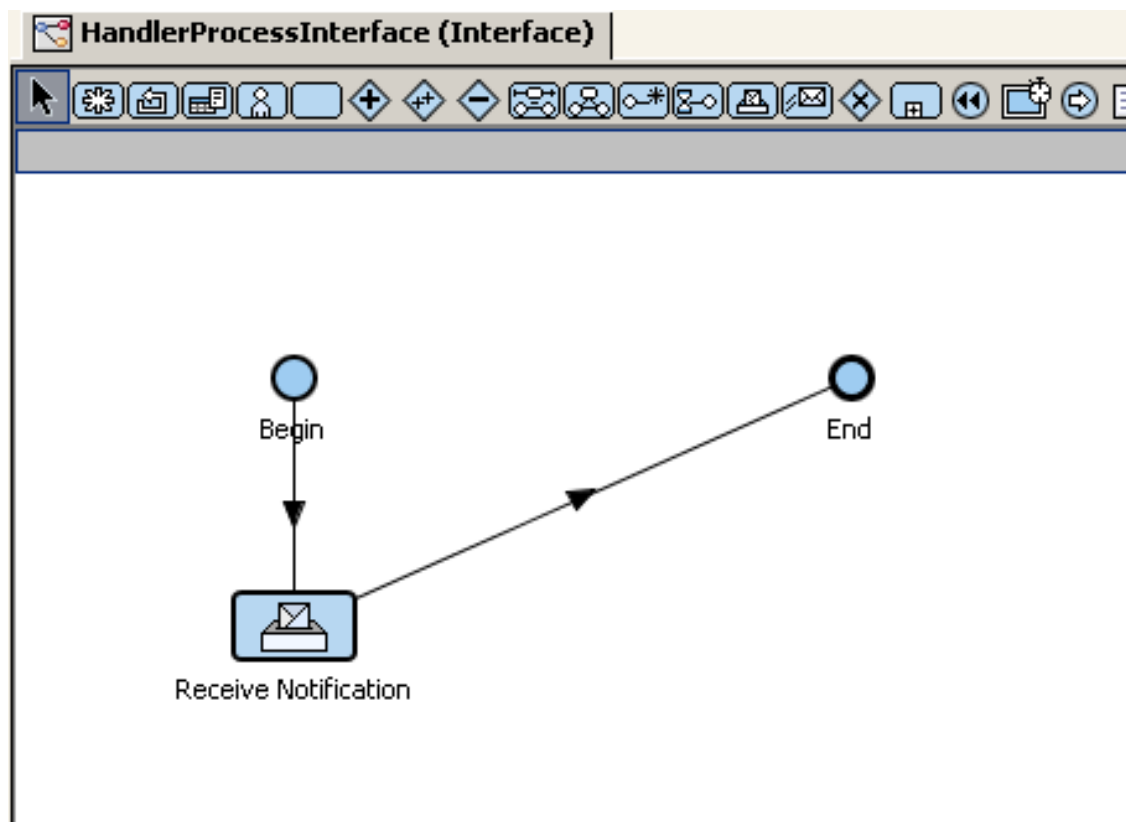


In the example the **dynamicProcessName** instance variable can

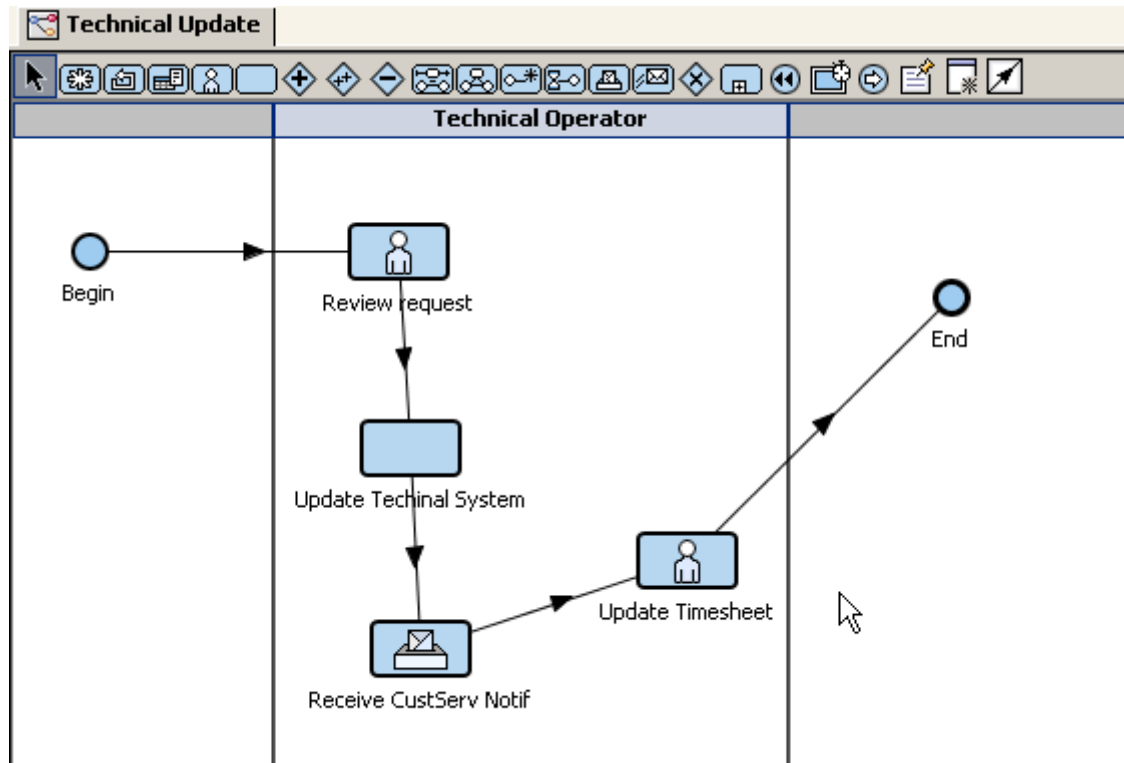
contain the value "**Technical Update**" that indicates that process Technical Update has to be executed. This name was returned in the previous called procedure.

You have to design the Process template, generate the Process interface for it and import the interface into the project. Each invoked process can do different things as long as they match the interface.

Process Interface:



Process Technical Update that matches the Process Interface:



The **Process Technical Update** executes different activities but has to match the defined interface. It expects the process notification that sends the main process (**Receive CustServ Notif**).

Global actions

Global Creation Activity





The Global Creation activity is one way to create new instances in a process. When the Global Creation activity executes, an instance begins creation in the process. The Begin activity finishes the instance creation.

Any user-defined role in a process can contain Global Creation activities. This means that an end user can run the activity by selecting it in Work Portal.

The Global Creation activity has an implied transition to the Begin activity in the process. The implied transition will not appear on the

Process Designer workspace.

Global Creation characteristics

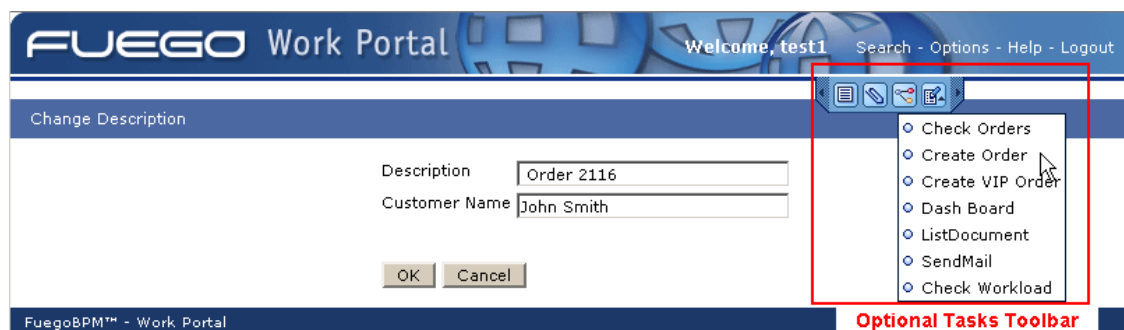
Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			

Work Portal and the Global Creation activity

The Global Creation activity is visible in Work Portal and is used to initiate an instance in the process.

All Global Creation activities appear in the Optional task toolbar as well.

When you execute a Main task you have enabled the optional toolbar.



In the picture above, the **Create Order** task is a Global Creation activity.

Roles and the Global Creation activity

The Global Creation activity must reside in a user-defined role lane.

Variables and Global Creation activities

Global Creation activities can access the arguments (defined in the corresponding Argument Mapping of the associated Mapping name), the predefined variables that do not require an instance (such as a process, a participant, etc.) and local variables.

Preconditions

There are no preconditions for the Global Creation activity.

Post-conditions

After execution, an instance flows to the Begin activity in order to finish creation.

Properties

Activity ID

See Creating an Activity.

General Category

Auto complete: allows instances to automatically flow to the Begin activity after the script is processed.

If you disable **Auto complete**, Work Portal process participant must click the **Send** button in order to send instances to the Begin activity after any script is processed.

Argument set Name: The Global Creation activity generates instances in the Begin activity. Therefore, it uses one of the sets of Argument mapping defined in it. See Argument Mapping & Message Based Transition.

Transitions and Global Creation activities

There are no transitions to or from the Global Creation activity. There is an implied (but not drawn) transition from the Global Creation to the Begin activity.

Tasks

A task will be automatically created with the same name as the one of the activity. The task can be implemented as a Method or as a Screenflow.

Business Process Methods considerations

You should set the arguments that the Begin activity expects to receive. Based on the chosen **Argument set name** within the Global Creation properties, the BP-Method completes the arguments that will be passed to the Begin activity as defined in the Argument Mapping in order to be converted into **instance variables**.

Global Creation example

Case 1: Creating an Instance

Use Global Creation as a means of initiating an instance in a process.

Create a Global Creation activity and assign it to a user-defined role (for example, Customer). Global Creation activity can appear in the process in any number of roles, but the created instance is always submitted to the Begin activity to complete its creation.

Within the Global Creation properties, in the **General** tab, the **Argument set name** defines the argument variables required by the Begin activity.

In the example below, custName and custNumber are defined as argument variables in the Begin activity within the Argument Mapping used by the Global Creation activity. Here, the Global Creation activity prompts the end user to enter a customer name and number, which will be sent to the Begin activity once "OK" has been selected. After the end user clicks the OK button, an instance begins creation. The instance completes creation in the Begin activity. If the Argument Mapping is set for both arguments, the received information is kept in instance variables.


```
//Prompts the end user to enter the customer name and
// number.
// The defined arguments names have to be referenced
// with "arg."
//
input "Customer Name: " arg.custName,
"Customer Number: " arg.custNumber
using title = "Enter Customer Information"
```

The Begin activity argument mapping will transform the received arguments into instances variables.

Case 2: Creating multiple Instances

Within a Global creation, you can create multiple instances by using the *create ProcessInstance* statement.

```
i = 1

input "Quantity of instances: " op

while i < op do
  create ProcessInstance
  i = i + 1
end
```

Warning



In FuegoBPM Studio, the maximum number of instances to be created at the same time is 5000. This maximum number can be simultaneously executed by the Server. Therefore, it can not create more than such number.

Case 3: Implementing a Screenflow

You can implement a Global creation activity with a screenflow.

Create a Global Creation activity and select the **Main Task** as a

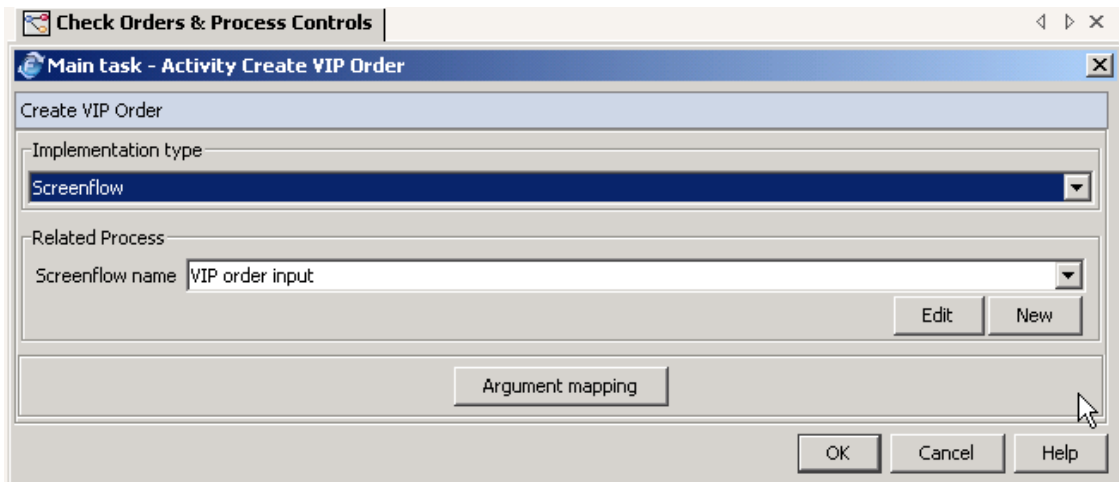
Screenflow.

Create the screenflow and select the in and/or out instance variables that map to the arguments. See Screenflow automatic generation for further information.

The Argument Mappings are automatically built for the Screenflow and the output Argument Mapping for the Global Creation activity.

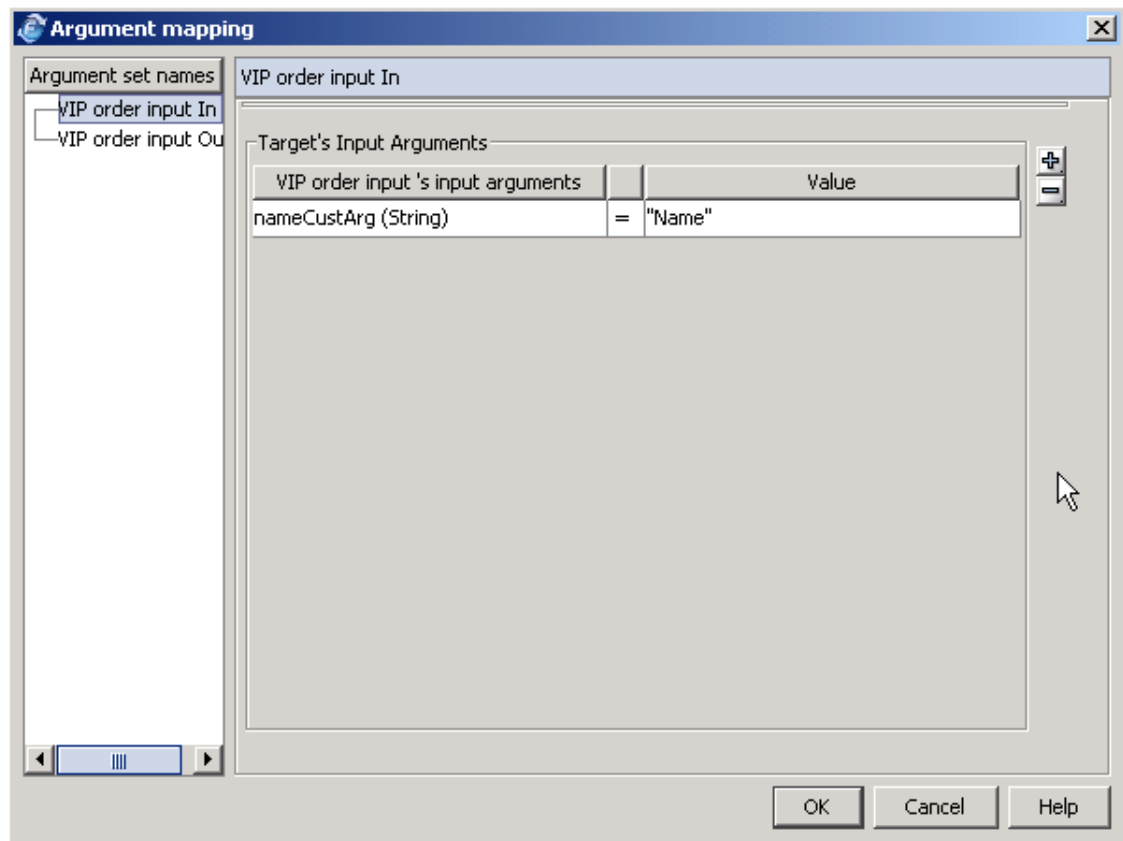
In the example:

- The process name is: **Check Orders & Process Controls**
- The global creation activity is: **Create VIP Order**
- The screenflow name is: **VIP order input**

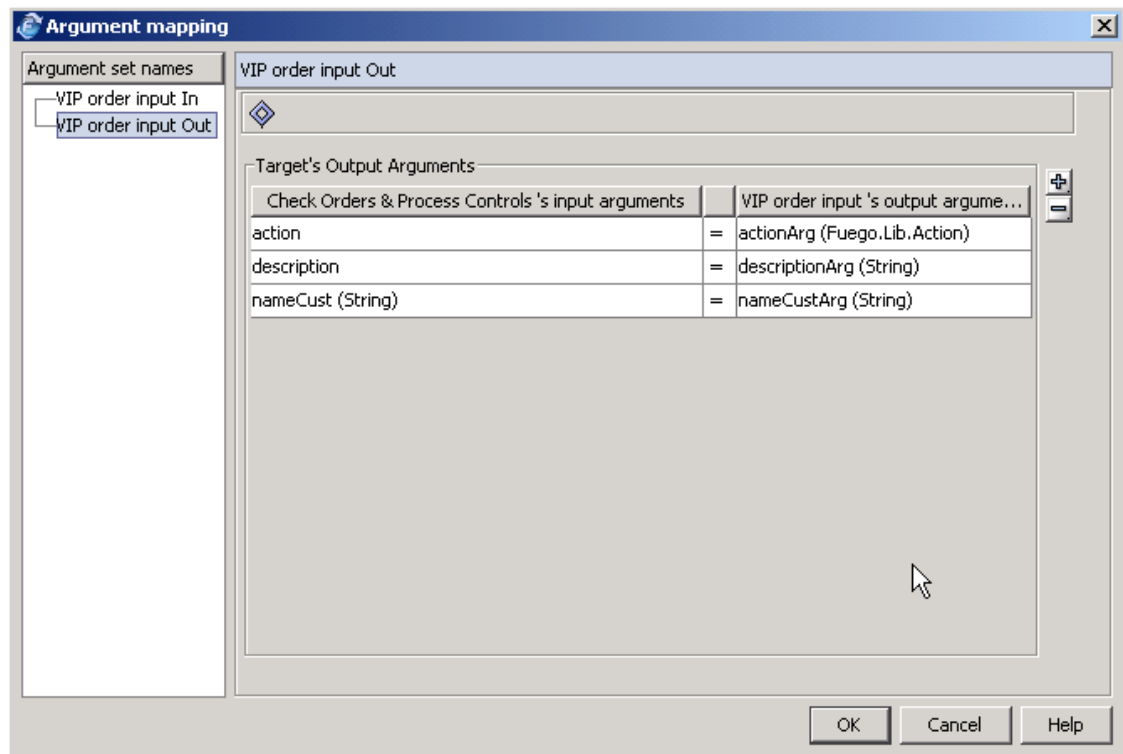


The Global Creation Argument Mapping is defined as:

- In arguments: correspond to the **Screenflow** expected input arguments. In the example, **nameCustArg** is the **Screenflow's (VIP order input) Begin** activity input argument. These argument are those received by the screenflow.



- Out arguments: maps the expected **in arguments** defined in the **Main process Begin activity** and the **Screenflow output arguments**. In the example the first column, you can define the **Check Orders & Process Controls** Begin arguments and in the second column, you map them to the out arguments of the screenflow **VIP order input**. **The Global creation output mapping is the bridge between the called screenflow and the main process begin activity.**



In the example, the **action** input argument that the main process receives corresponds to the **result** of the screenflow. This result has to be passed as an argument in the Screenflow's End activity (actionArg(Fuego.Lib.Action)). This mapping has to be manually added if you need in the main process (Begin activity) the result of the implemented screenflow in the Global creation activity.

Trouble Shooting

1. You are defining the Global Creation BP-Method and no arguments to pass to the Begin appear.
 - a. You have not set the **Argument set name** for the Global Creation (Properties/General tab), or
 - b. No argument mapping has been defined in the Begin activity

Global Automatic

Global Automatic activities do not have any direct end user interaction. The applications/components invoked by the Global Automatic activity typically run on a remote server.

Global Automatic activities are useful for processing batch reports or downloading batch files at scheduled times. Global Automatic activities can also be used as event listeners in the process. They can be programmed to listen to a port or to a specific event, such as an end user mouse click or a broken connection to a remote component. And then, based on such event, they perform some type of action.




Global Automatic activities do not appear in the Work Portal tasks list. These activity types can invoke a component or application that creates instances. But the instances are automatically created without user intervention.

Instances flowing through the process do not have any interaction with the Global Automatic activity. However, the Business Process Method in the Global Automatic activity may be run because of some action caused by an instance (Executes when an event occurs) or it may create an instance in the process.

Global Automatic activities:

- Do not have any relationship with an instance.
- Automatically execute the Business Process Method without receiving an instance, even though they can interact with an instance in the BP-Method. These kinds of BP-Methods do not work directly over an instance but, for example, they can send notifications to instances.
- Cannot be manually launched.

Global Automatic activity characteristics

Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			 /

Work Portal and the Global activity

The Global Automatic activity is not visible in Work Portal because the activity does not require any end user intervention.

Roles and the Global activity

Global Automatic activities can reside in a user-defined or an automatic role lane.

Variables and Global activities

The Global Automatic BP-Method can access predefined and local variables.

Preconditions

- **Polling by Interval:** waits for a specified interval before launching the BP-Method.
- **Executes when an event occurs (event listener):** waits for a specified event before launching the Listening BP-Method.
- **Automatic Scheduled:** waits for a scheduled date and time before launching the BP-Method.
- **Automatic JMS Listener:** This type is applicable for FuegoBPM Application Server Edition. Listens to JMS message before launching a BP-Method.

Post-conditions

Post-conditions vary depending upon the type of Global Automatic activity and the BP-Method that is launched.

- **Polling by Interval:** BP-Method runs after an interval. Once the Global Automatic task finishes, the new interval of time begins to count. Any action required by the BP-Method, such as instance creation, is completed.
- **Executes when an event occurs** (event listener): BP-Method runs when a predefined event occurs. Any actions required by the BP-Method, such as instance creation, are completed.
- **Automatic Scheduled:** BP-Method runs according to a time schedule. Any actions required by the BP-Method, such as instance creation, are completed.
- **Automatic JMS Listener:** This type is applicable for FuegoBPM Application Server (EJB based Servers). BP-Method runs when a predefined event occurs. This BP-Method has as argument a `Fuego.Msg.JmsMessage` that is a wrapper of the `javax.jms.Message`. Any actions required by the BP-Method, such as instance creation, are completed.

Properties

The Global Automatic Activity Properties dialog box allows you to enter a description of the activity.

General Category

Global Automatic type: defines when and how the Global Automatic BP-method will run.

- **Polling by Interval:** The Polling by interval Global Automatic activity runs its BP-method on a time schedule you create in the

Wait interval field. The default is **1d**, which means that the BP-method will run every 24 hours approximated from the time the Server to which the process is published and deployed is started. Actually the "one day" period of time begins after the task finishes its execution, therefore the total time in between one task run and the other one is the **Wait interval + the task execution time**. See Polling by Interval on this page for further information.

- **Executes When an Event Occurs:** The Executes when an event occurs Global Automatic activity runs its BP-Method when an event defined in the **Listener class** field executes.

You can use one of the three events included in the FuegoBPM installation, in the Component Catalog within Fuego/Msg (fuego.MessageQueueListener, fuego.ServerLogListener or fuego.TopicListener) or you can create your own listener class. See Executes when an Event Occurs on this page for further information on the Listener Class. The **Component valid return values** drop down box contains the valid return values for the Java class file indicated in the **Listener class** field. The drop-down box auto-populates with information from the class file. All possible data types that the component can retrieve are populated. This information is obtained from the component when it was introspected into the Component Manager.

Note



You should catalog your Java class file in the Component Manager if you choose to use one of your own. You cannot select your file unless it has been catalogued in the Component Manager.

- **Automatic Schedule:** The Automatic Schedule Global Automatic activity runs its BP-Method on the exact date and time you specify. You can specify **Daily**, **Weekly** and **Monthly** and the exact time at which the BP-Method should run. See Scheduled

type on this page for further information on how to set the schedule. You can also disable the **Runs also on holidays** check box to make sure that the BP-Method does not run on holidays. The holidays are set in the Organization Administrator. If the scheduled time falls on a holiday, the BP-Method will only run if this check box is enabled.

- **Automatic JMS Listener:** This type is applicable for FuegoBPM Application Server Edition. The Automatic JMS Listener activity listens to JMS message and executes a BP-Method. This BP-Method has as argument a Fuego.Msg.JmsMessage that is a wrapper of the `javax.jms.Message`.

Stop running when process is deprecated: Ensures that the BP-Method is not running when a process version has been replaced by a new version. This field is selected by default. If the **Stop running when process is deprecated** check box is not checked, the activity will still run after a new version of the process is published. If the Global Automatic activity uses an instance creation behavior (*create ProcessInstance*), the BP-Method will create an instance in the active process.

Instances flowing through the process do not have any interaction with the Global Automatic activity. However, the BP-Method in the Global Automatic may be run because of some action caused by an instance (Executes when an event occurs). Or the BP-Method may create an instance in the process.

Transitions and Global activities

There are no transitions to or from the Global activity.

Tasks

No tasks can be generated. A BP-method will be automatically

created with the same name as the activity. Or, if the Global Automatic activity works *when an event occurs*, two BP-Methods will be generated. See below for further information.

Business Process Methods Considerations

Global activities can contain only one BP-Method if the activity is defined as "Polling by interval" or "Automatic Schedule". This BP-Method is automatically created with the same name as the activity.

If the activity "Executes when an event occurs", then two BP-Methods are automatically generated. One of them has the same name as the activity and the other ends with "_Listening". The last BP-Method is the one that will be launched each time the event occurs. The first BP-Method will only execute once (at the server start or at the deploying time), which is used for the listener component to get initiated.

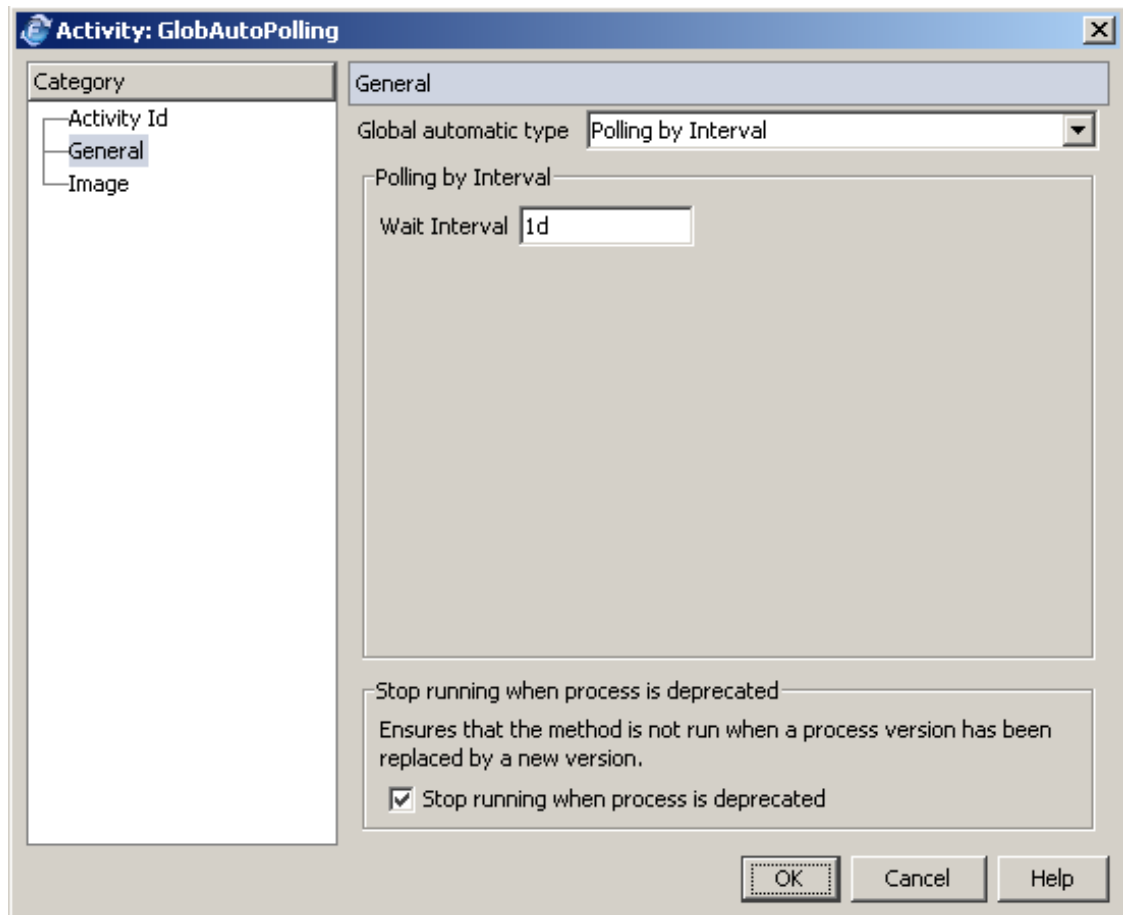
Any application can be invoked from a Global activity, but the information should not impact on the movement of instances through your process.

Polling By Interval

The **Polling By Interval** option on the Global Automatic Activity Properties dialog box indicates that the BP-Method executes according to an interval, which is indicated by the **Wait Interval** field.

The interval begins to count once the Global Automatic task finishes, therefore the total time in between one task run and the other one is the **Wait interval + the task execution time**.

As shown in the example below, "1m" means that the Script will execute once the task finishes, after one minute period time occurs.



The syntax is:

- *d* for day
- *h* for hour
- *m* for minute
- *s* for seconds

Examples of valid intervals include:

- '3d ' for three days

- '1h' for one hour
- '4m' for four minutes
- '2m30s' for two minutes and 30 seconds

Stop running when process is deprecated

The **Stop running when process is deprecated** check box is selected by default. It is used to tell the Server to stop running the activity when a new version of the process is published. When this occurs, the old version is *deprecated*. (It is either replaced by a newer version or undeployed in the Web Console.)

Global Automatic activities do not create instances by themselves. If instance creation is required, the behavior should be explicitly written in the activity's BP-Method with a create *ProcessInstance* statement.

The Global Automatic activity FBL can manage two out arguments: **retryIn** and **retryAt**.

Both can be used to reschedule next polling. If any of them is set within the FBL, the next polling will occur based on the values of these arguments. If these arguments are not set during the FBL execution, next polling will happen based on the polling interval defined within the activity property.


retryIn: you can set it with a new interval.

retryAt: you can set it with a defined time.

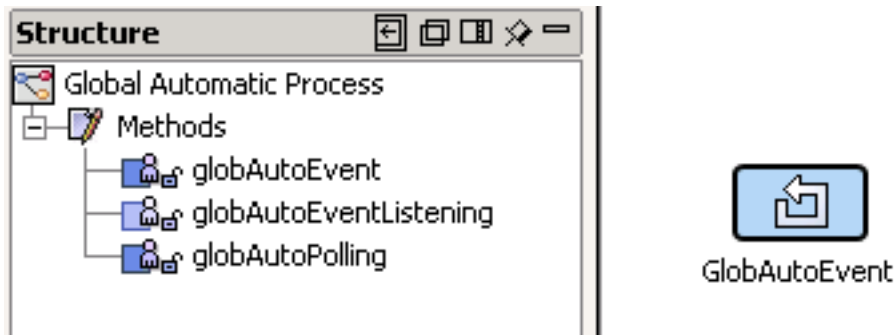
Executes when an event occurs

The main function of the **Executes when an event occurs** option is to act as an event listener in the process. When a certain event occurs, the BP-Method ("Listening") executes.

Note

 This option is not applicable with EJB based servers. You must use the Automatic JMS Listener.

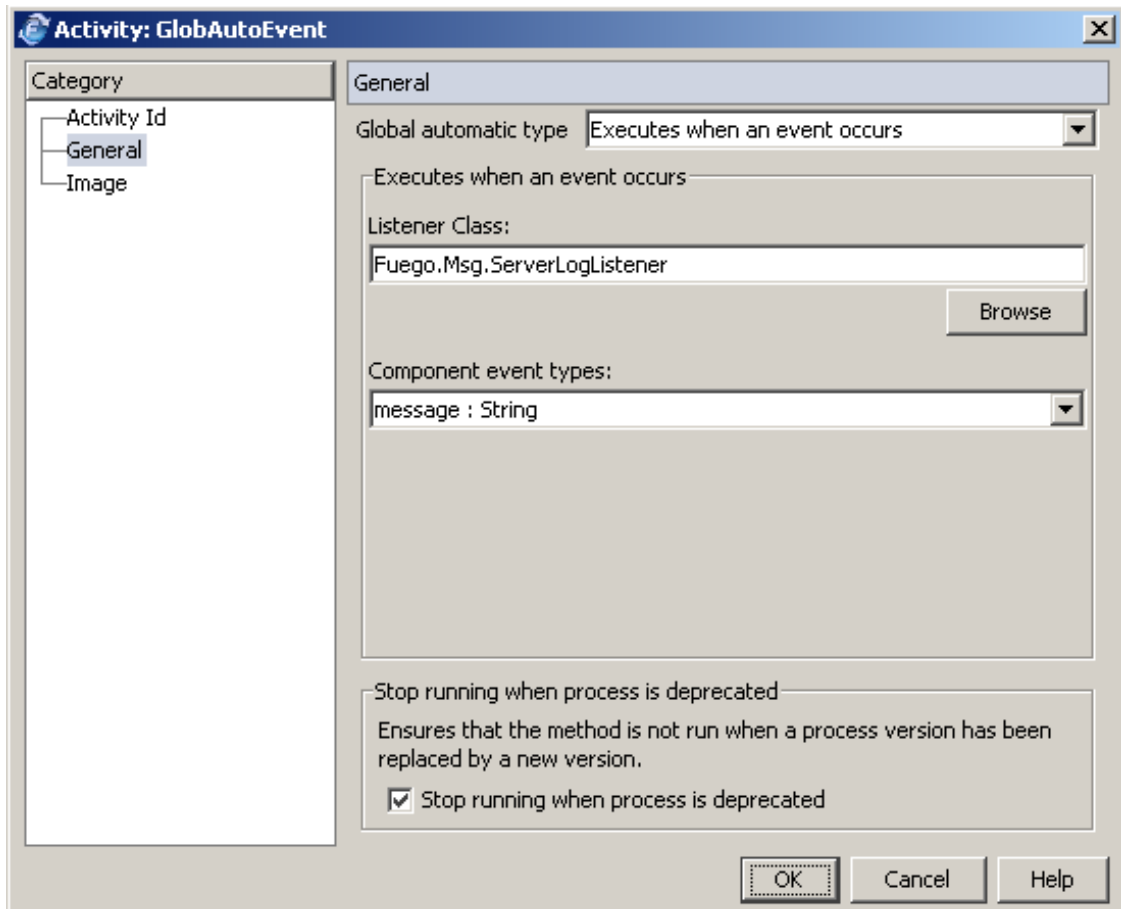
In this case, the activity manages two BP-Methods. These Methods are automatically generated once the **Global Automatic** activity has been defined:



1. **Init** BP-Method (same name as the activity): this script prepares the listener component in order to run. It is in charge of passing the arguments required by the external listener component. The **init** Script executes when the FuegoBPM Server starts or right after the process is deployed.
2. **Listening** BP-Method (same name as the activity and ended with "_Listening"): this script receives an argument from the Listener component. The **listening** BP-Method is invoked by the listener component through the FuegoBPM Server. Upon occurrence of the specified event, the listener component notifies the FuegoBPM Server. Then, the server runs the listening BP-Method. The listener component passes arguments to the listening BP-Method. The argument type should be selected from the **Component valid return values** field, as shown in the image above.

When you choose **Executes when an event occurs** from the **Global Automatic type** drop-down menu, a new field called

Listener class appears in the activity properties dialog box.



Listener classes are Java classes that are specifically designed to listen to a particular event. FuegoBPM provides a standard listeners that you can use. Or you may develop your own one and add it to the catalog of components through the Component Manager.

The listener component is actually a standard component included within FuegoBPM:

- **ServerLogListener:** The ServerLogListener component listens to events in the Server log. The server's logs can be accessed through the Web Console. See ServerLogListener component documentation for further information.

An example of the use of this component is provided in the GlobalAutoListCase01.fpr.exp project in FuegoBPM's installation directory, in the studio/samples directory.

Import this project into the FuegoBPM Studio, and use the **LogListener** process as an example.

There are 2 other listeners provided as standard components but they are no longer recommended.

- TopicListener: The TopicListener listens to events in a JMS compliant Topic.
- MessageQueueListener: The MessageQueueListener component functions as a Java Message Service (JMS) MessageListener and as a ServerEventSource. It listens to events in a JMS compliant MessageQueue.

Instead of using these components you should implement the **Global Automatic** activity as a **JMS listener** (see below) for Queues or for Topics.

BP-Method

"Executes when an event occurs" type of the Global Automatic activities requires two different Scripts, an **init** BP-Method and a **listening** Method. The **init** Method executes when the FuegoBPM Server starts or at deployment time. It is invoked once in order to initialize the listener component.

Writing the BP-Method

The first BP-Method, called *ActivityName* should set the argument variable **componentArguments** (automatically generated) with all the values that will be passed to the listener component. This BP-Method is only invoked once at server startup or when the process is deployed.

An example of init BP_Method is as follows:

```
arguMap["interval"]=5  
componentArguments = arguMap
```

This example shows how to send the value '5' meaning five minutes, like the interval needed by the listener component "Timer" to init the interval value.

The second BP-Method, called *ActivityName_Listening*, receives—as a parameter—a variable with the same name as the name of the argument chosen when setting the component in this activity properties. The external component source will publish the possible values to be used as arguments. All the values start with ARG_ and are followed by a 'real name'. This argument contains the argument passed by the component to the BP-method, when the event occurs.

Note



The type of this variable should be the same type as the variable set in the **Component valid return values** field on the Activity Properties dialog box.

The listening BP_Method is called each time the component invokes the *processEvent* method. An example of Listening BP_Method is as follows:

```
create ProcessInstance using argumentsSetName = "BeginIn",  
arguments = ["customer":ARG_CustomerString]
```

This example shows how to create an instance each time the listening component sends the event to the FuegoBPM Server using the argumentSetName (see **Argument Mapping** and how to use the received data sent by the listener component) as the argument for the instance creation.

Building your own external event listening component

When building an external event listening component, the component should implement the *fuego.components.ServerEventSource* interface. By implementing *ServerEventSource*, the class file appears in the **Listener class** drop-down menu on the activity properties dialog box.

The listener component through the public fields *ARG_* will set the possible arguments. The argument types should be catalogued first.

In the definition of the global automatic listening, when introspecting the Listener class, the types of fields named *ARG_* will be searched to be a possible return type of the class. Thus, the class programmer should add non-private fields with a name starting with *ARG_* for each type that the class may return.

This field name will be passed as an argument to the activities in the second BP-Method.

An example of an external event listening component is provided in the *GlobalAutoListCase01.fpr.exp* project in FuegoBPM's installation directory, in the *studio/samples* directory.

The component that works as the external listening component is called *TimerComponent*.

- Import the project into the FuegoBPM Studio and deploy it.
- Find in the project's root directory, some java files that are Listener components to use as an example or a guide:

1. *TimerComponent.java*: This component is an example of a Listener that needs to be compiled and catalogued. The component activates the second FBL in the Global automatic

activity (ListenerbyInterval) every 5 minutes. This FBL creates a logmessage.

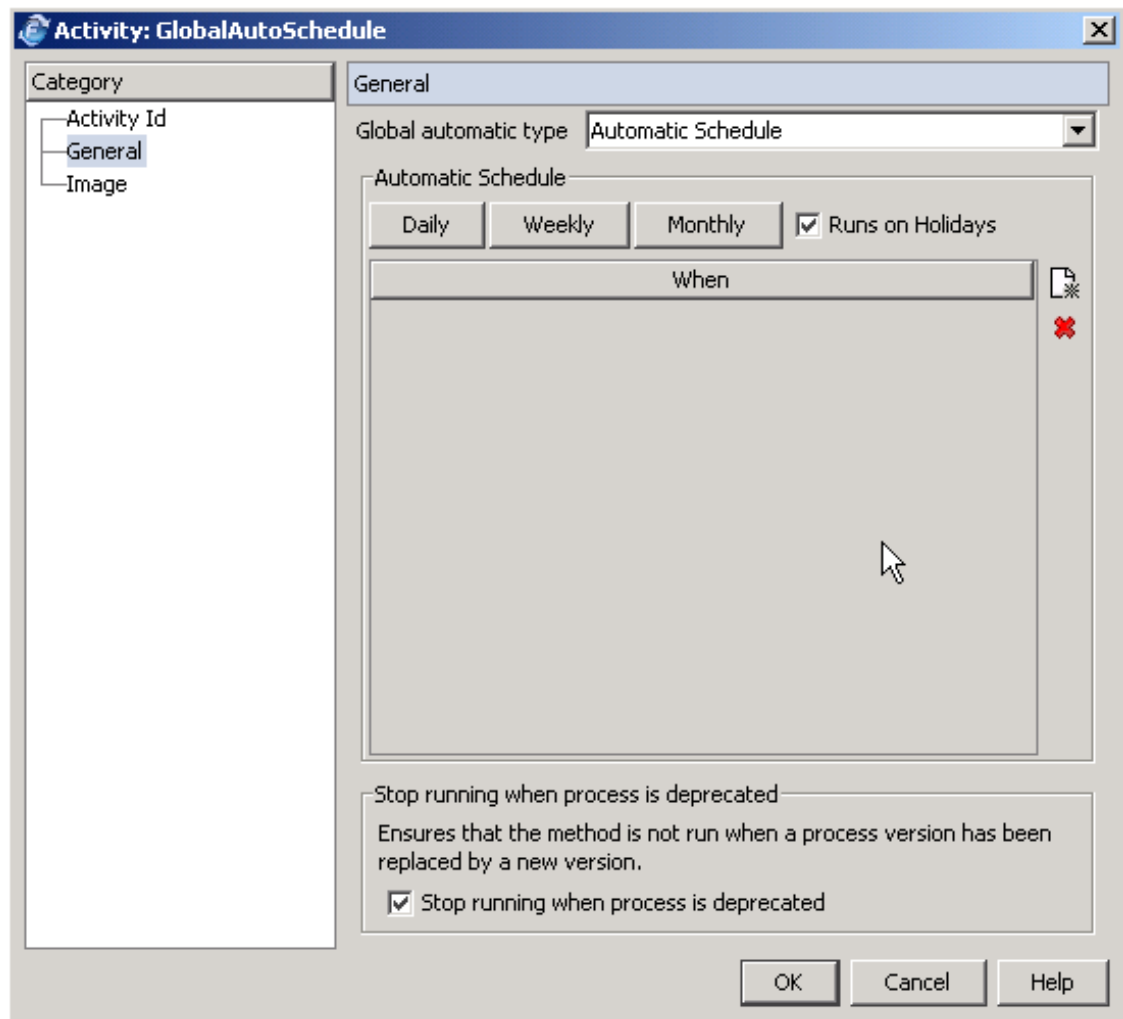
2. MyEventListener: This is a template java component that shows the different methods you can implement in your own listener.

Cataloguing the component

After development, the external component cannot be used in the process design until it is catalogued in the Component Manager application. Refer to the documentation included in Component Manager for further information on cataloguing components.

Global Automatic scheduled activity

The Scheduled option on the **Global Automatic activity property dialog box** indicates that the activity is set to run on a specific date at a specific time. Use the option buttons in the **Automatic Schedule Type** field to change the time as deemed appropriate.



The **DAILY** choice allows you to select a time at which you want the BP-Method to run every day.

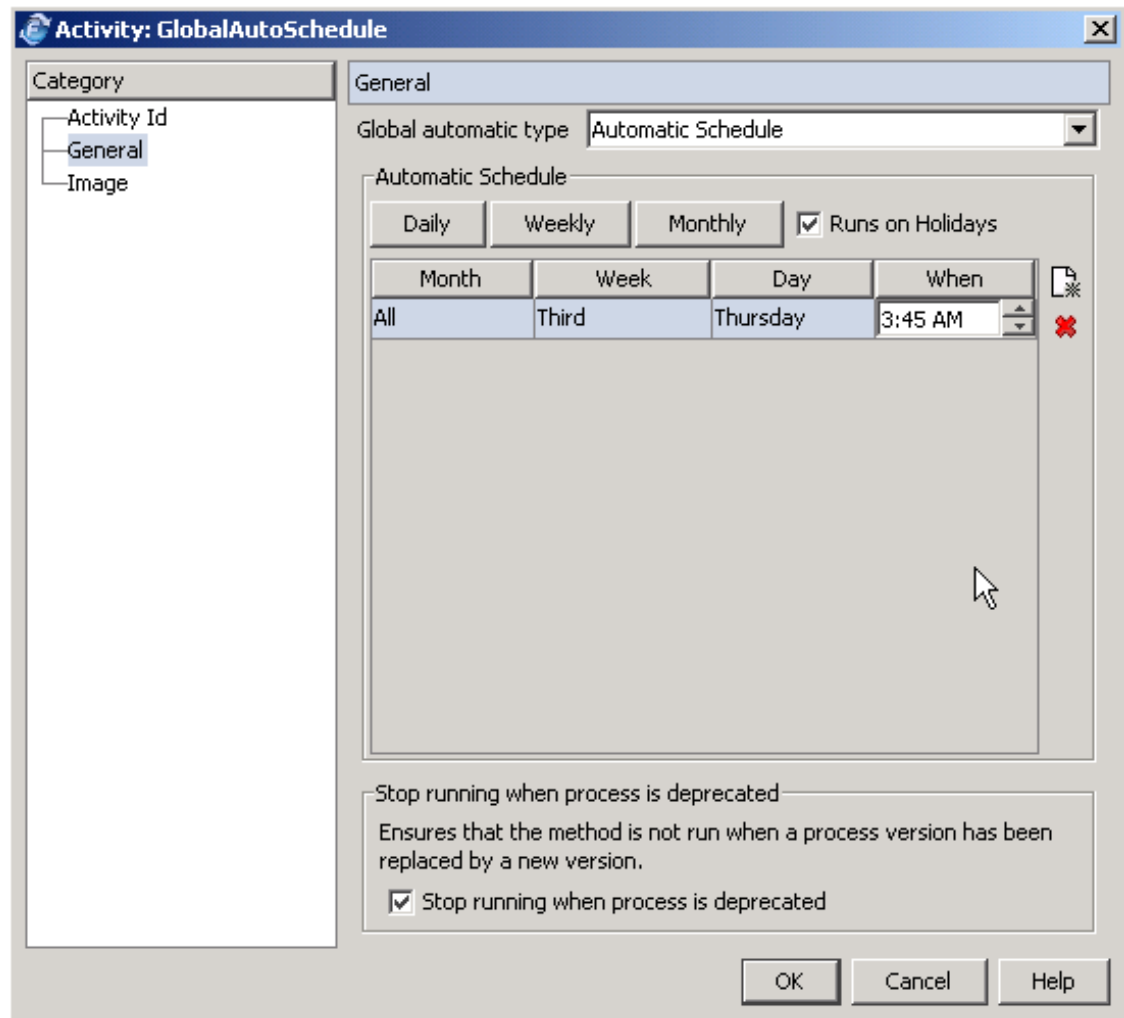
The **WEEKLY** choice allows you to select a "**day of the week**" and a time when the BP-Method will run every week. For example, the BP-Method can run every "**Monday at 10:18 A.M.**"

The **MONTHLY** choice allows you to select

- the *month*,
- the week of the month (First / Second / Third /Fourth / Last) or the day of the week,

- the day (Sun-Sat) or (1-31),
- the time at which the BP-Method will run.

For example, you can choose to have the BP-Method running on the third (3rd) Thursday of every month at 3:45 P.M.



Or, the 15th of January, April, July and October at 11:00 AM.

Activity: GlobalAutoSchedule

Category: Activity Id, General, Image

General

Global automatic type: Automatic Schedule

Automatic Schedule

Daily Weekly Monthly ☒ Runs on Holidays

Month	Week	Day	When
January	Day of month	15	11:00 AM
April	Day of month	15	11:00 AM
July	Day of month	15	11:00 AM
October	Day of month	15	11:00 AM




Stop running when process is deprecated

Ensures that the method is not run when a process version has been replaced by a new version.

☒ Stop running when process is deprecated

OK Cancel Help

Note

 To add more lines to the table schedule, click on . To delete lines, click on .

The **Runs on Holiday** check box is selected to indicate that the BP-Method will run even though the scheduled day falls on a holiday. Holiday rules are defined and scheduled in the Organization Administrator.

What happens if, at the scheduled time, the component fails?

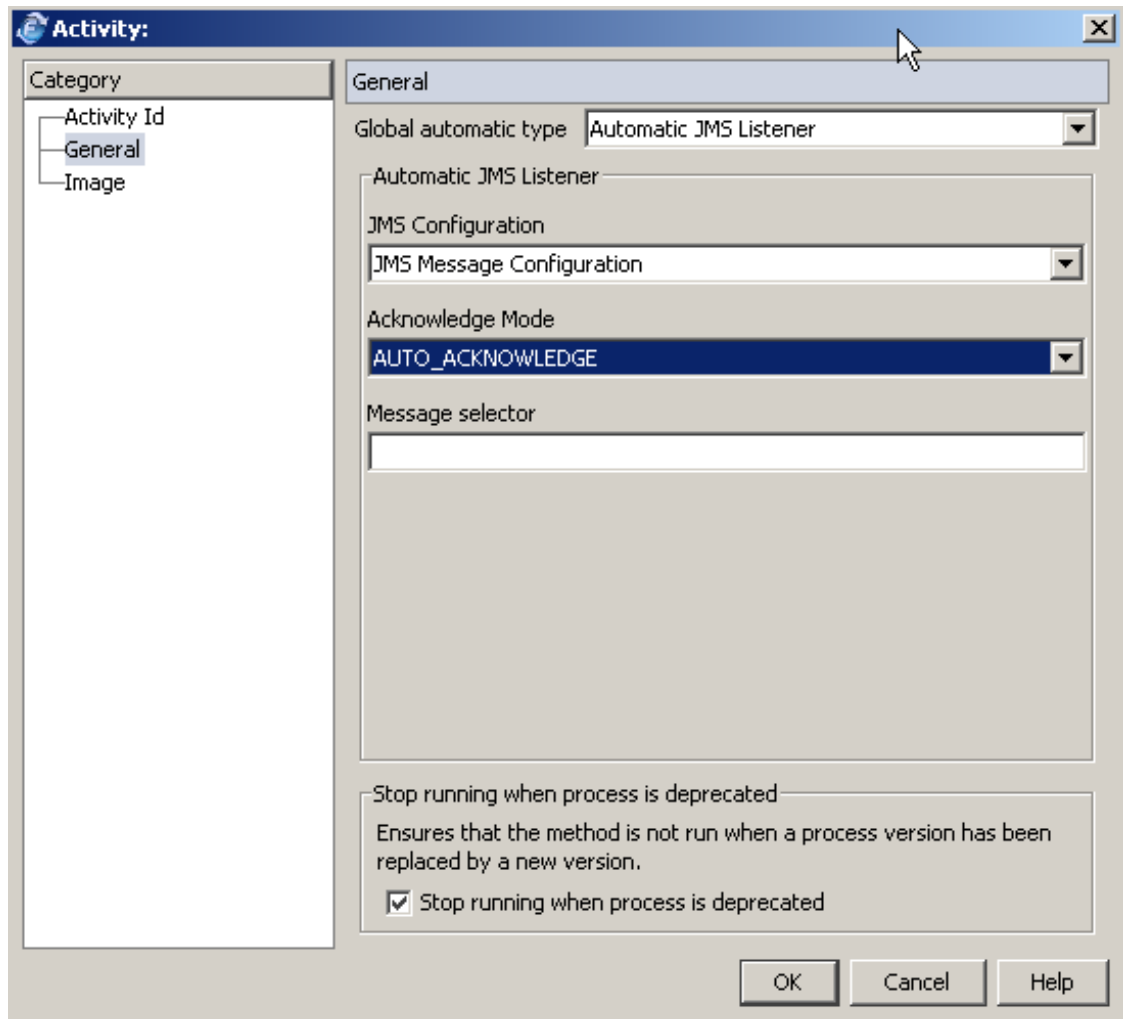
If an unforeseen event prevents the component from running at the scheduled time, the component will run as soon as it can do so. If the time frame between the *original scheduled time* and the *time it can do so* exceeds the time between the *original scheduled time* and the *next scheduled time* in more than 10%, the activity will not run and you will have to wait until the *next scheduled time*.

For example, if you schedule the Global Automatic activity to run each **Monday and Friday at 10:00AM** and on **Monday 10:00 AM** it fails, it will retry as soon as possible. As the next scheduled time (**Friday 10:00AM**) is **120 hours** far away from this scheduled time (**Monday 10:00 AM**), if the problem is solved in the next **12 hours** (10% of 120 hours), then the Global Automatic activity will be executed. If it exceeds 12 hours (after Monday 10:00 PM), then it will directly run on the **next scheduled time, Friday 10:00 AM**.

To understand how to create an instance from a Global Automatic activity, see the **Process Instance component** documentation.

Automatic JMS Listener

This type of activity must be used when deploying the process in a FuegoBPM Application Server Edition. But it can also be used for FuegoBPM Enterprise Server Edition.



Select the JMS Configuration previously implemented in External Resources.

The **acknowledge mode** can be:

- **AUTO_ACKNOWLEDGE**: no message acknowledgment needs to be managed.
- **DUPS_OK_ACKNOWLEDGE**: the Fuego BP-method returns a boolean to determine the action to take.

If you choose not to stop listening when the process is deprecated,

then be aware that in the case the activity is listening from a Queue, the message can arrive to a global activity of any of the current versions of the process that are deployed. It is convenient to use a selector to filter the messages.

If the activity is listening from a Topic, the message is received from all the versions including the active one.

Global Automatic Examples

Global Automatic Examples.

Global Automatic Examples

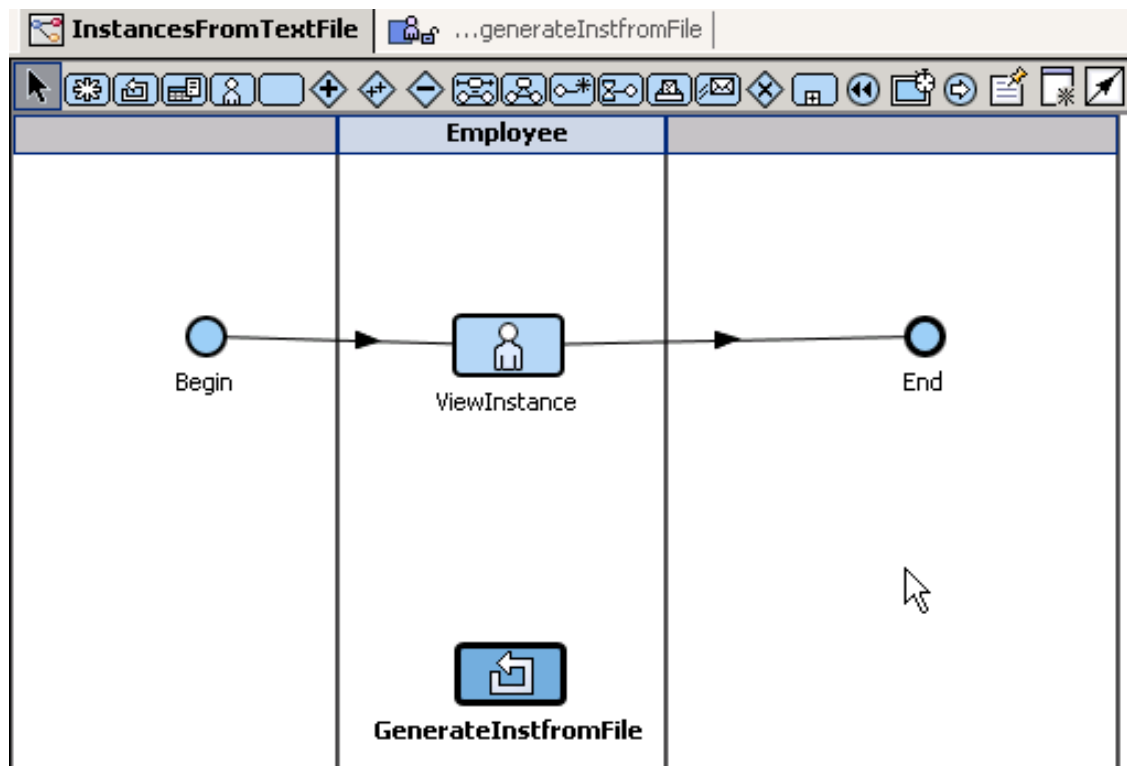
Using the Global Automatic Activity

Case 1: Generating multiple instances from a file (GlobalAutoCase01.fpr)

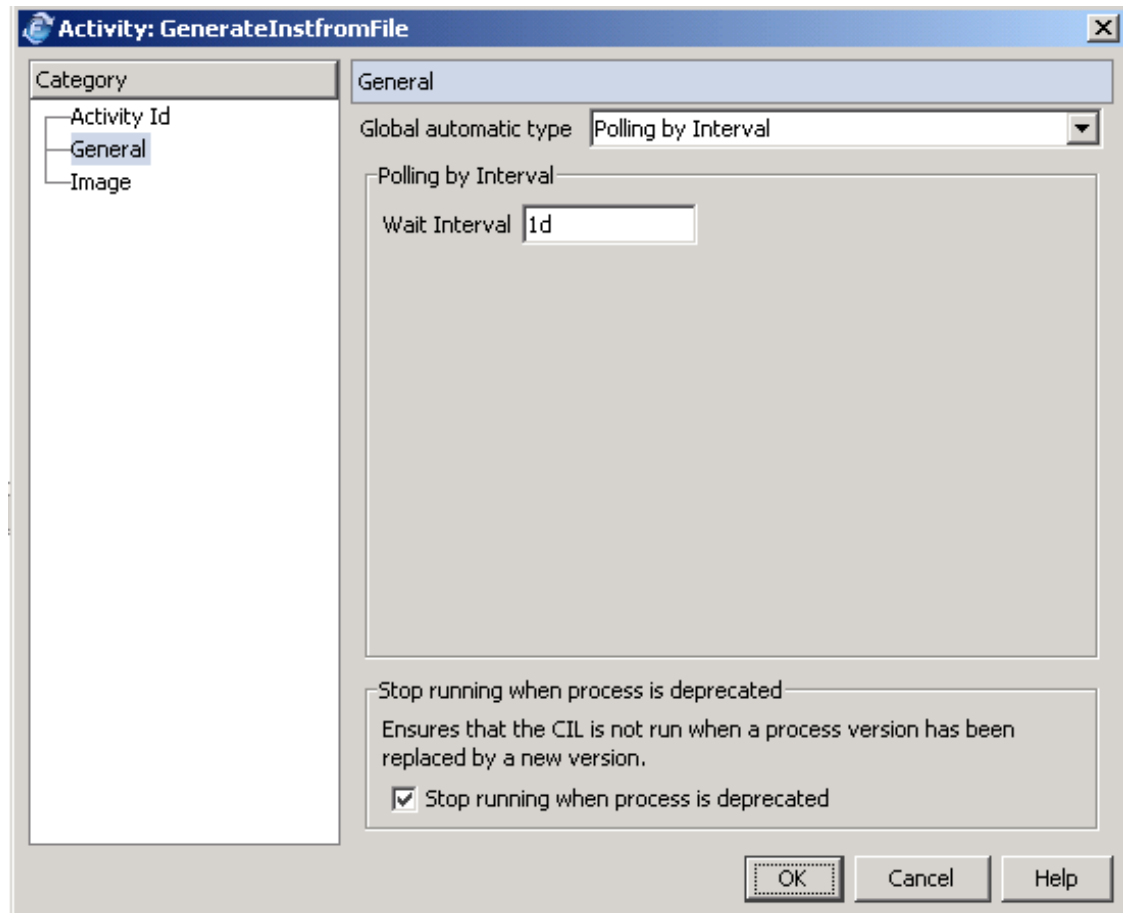
Your department receives on a daily basis, a file with all the invoices generated the day before. Each invoice is an instance in the process.

The file is copied to a certain location (c:\tmp) and needs to be processed automatically.

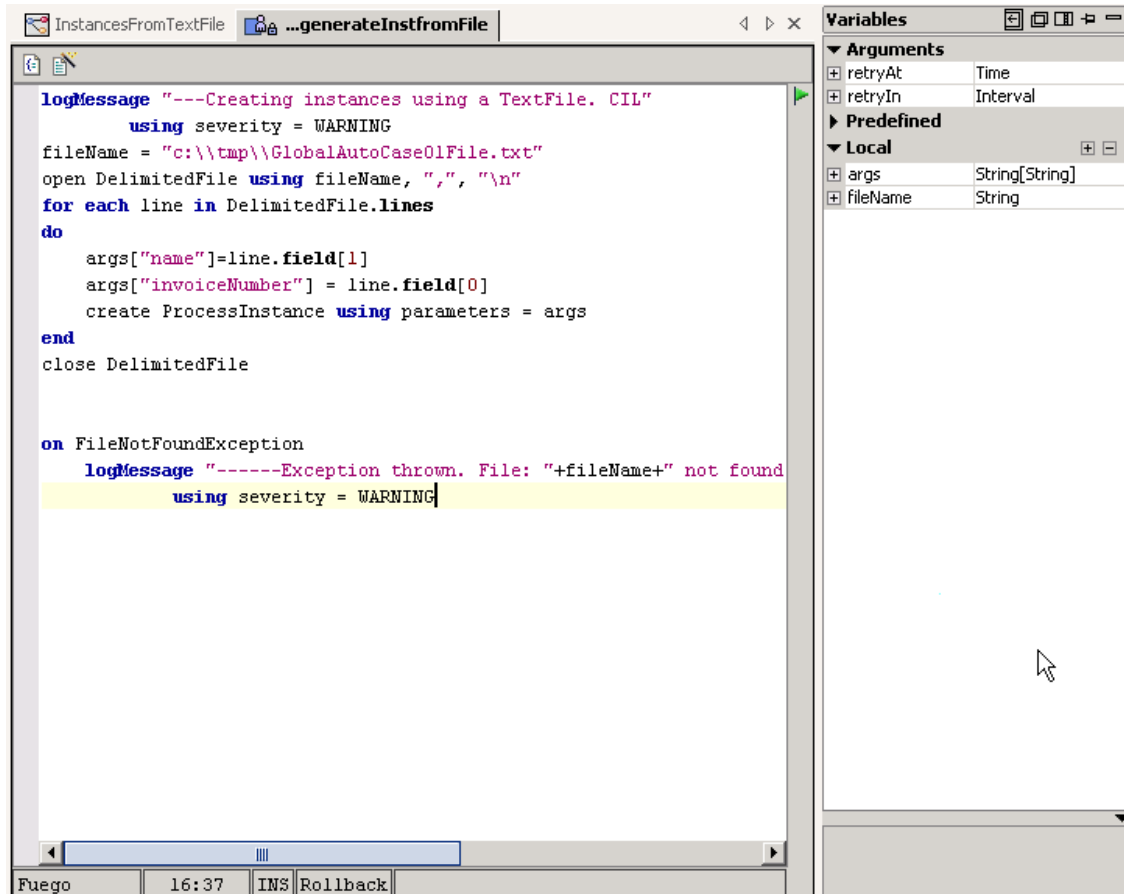
The following process automatically reads the file and for each line in it (each invoice) generates an instance that flows through the process.



The Global Automatic activity is defined to be executed as Polling by Interval every day:



And the Global Automatic activity has the following BP-Method:



The file has the following format: (invoice#, name)

1111,Jhon Perez


2222,Raul Mendez

3333,Esteban Deluca

4444,Alex Rivera

For more details find the project **GlobalAutoCase01.fpr** in FuegoBPM's installation directory, in the studio/samples directory to study the example further.

Note

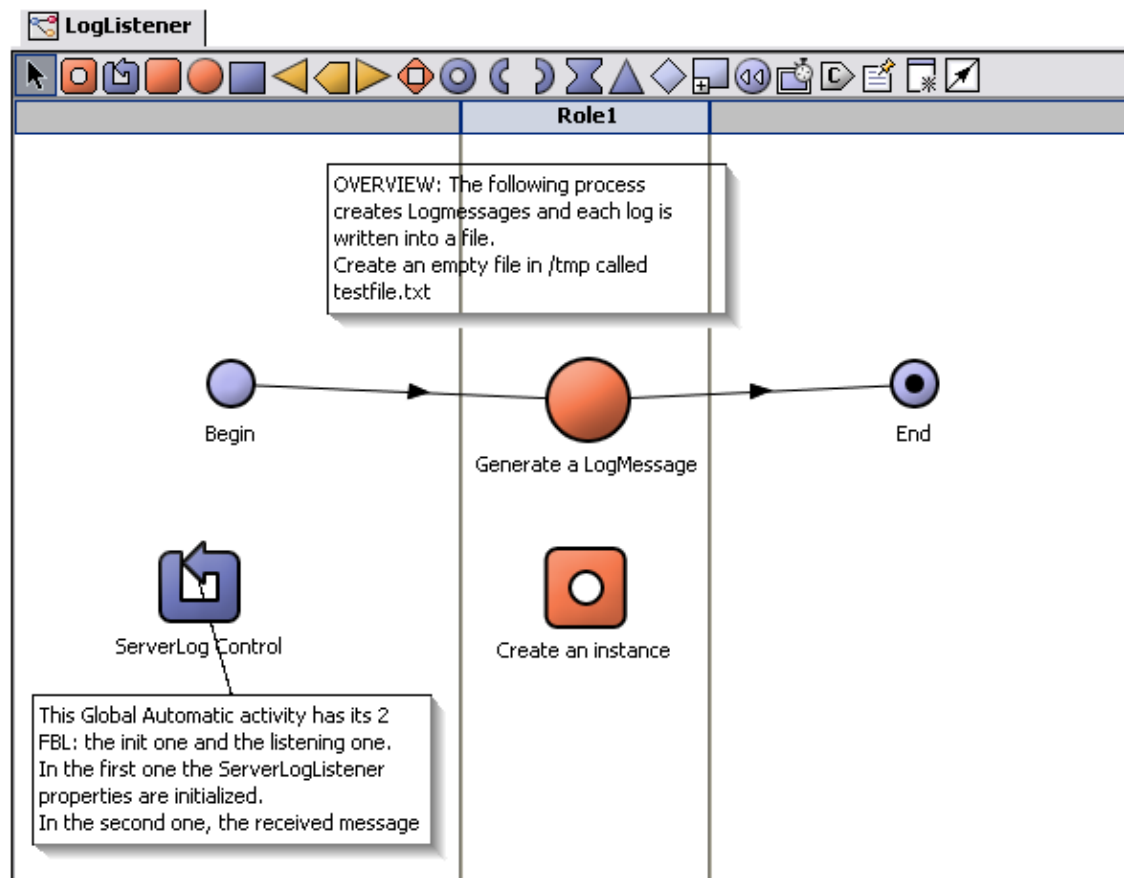
 This example was built to run on Windows as the text file from which the instances are created is in c:/tmp/GlobalAutoCase01File.txt. Change

this location and name in the Global Automatic activity Method if necessary.

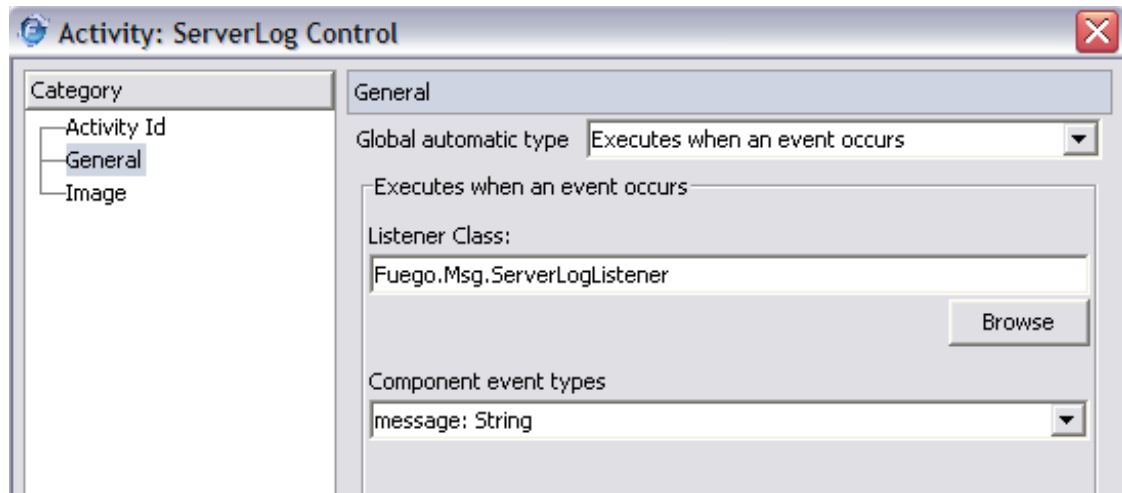
Case 2: Implementing the ServerLogListener component (GlobalAutoListCase01.fpr)

The following example implements FuegoBPM standard component **Msg.ServerLogListener**.

Every log that is posted, is copied to a file.

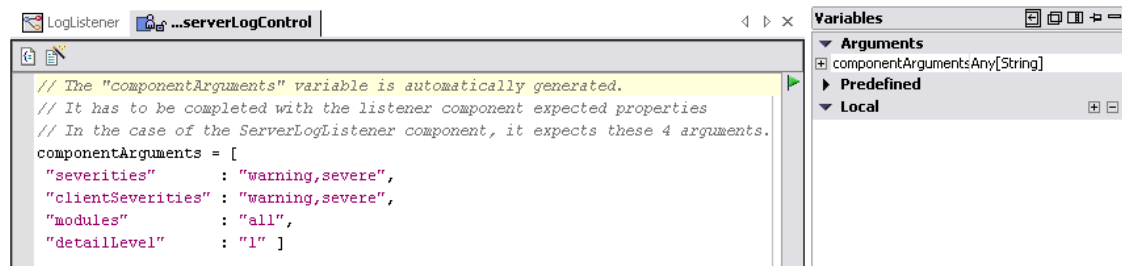


The Global Automatic activity, **ServerLogControl**, implements the **ServerLogListener** component:



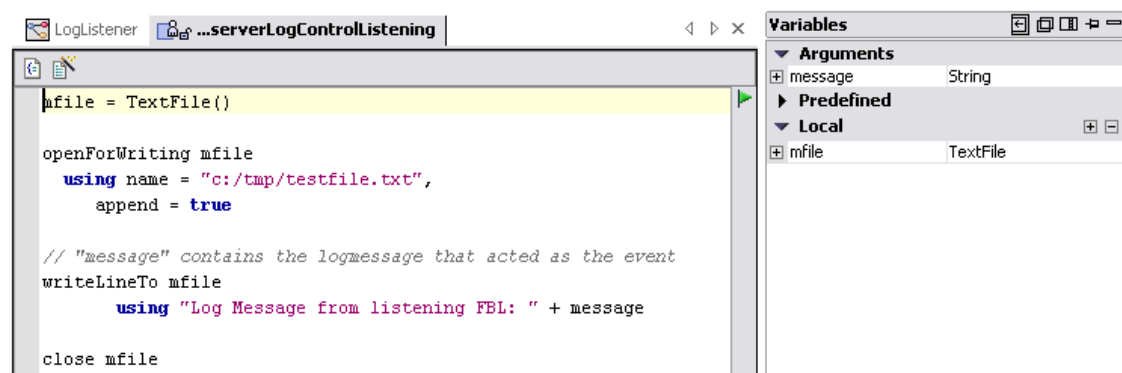
It has its two FBL: the init one and the listening one:

Init FBL



Initializes the properties required by the component

Listening FBL



On each event (when a log is posted), the logmessage is copied to a

file.

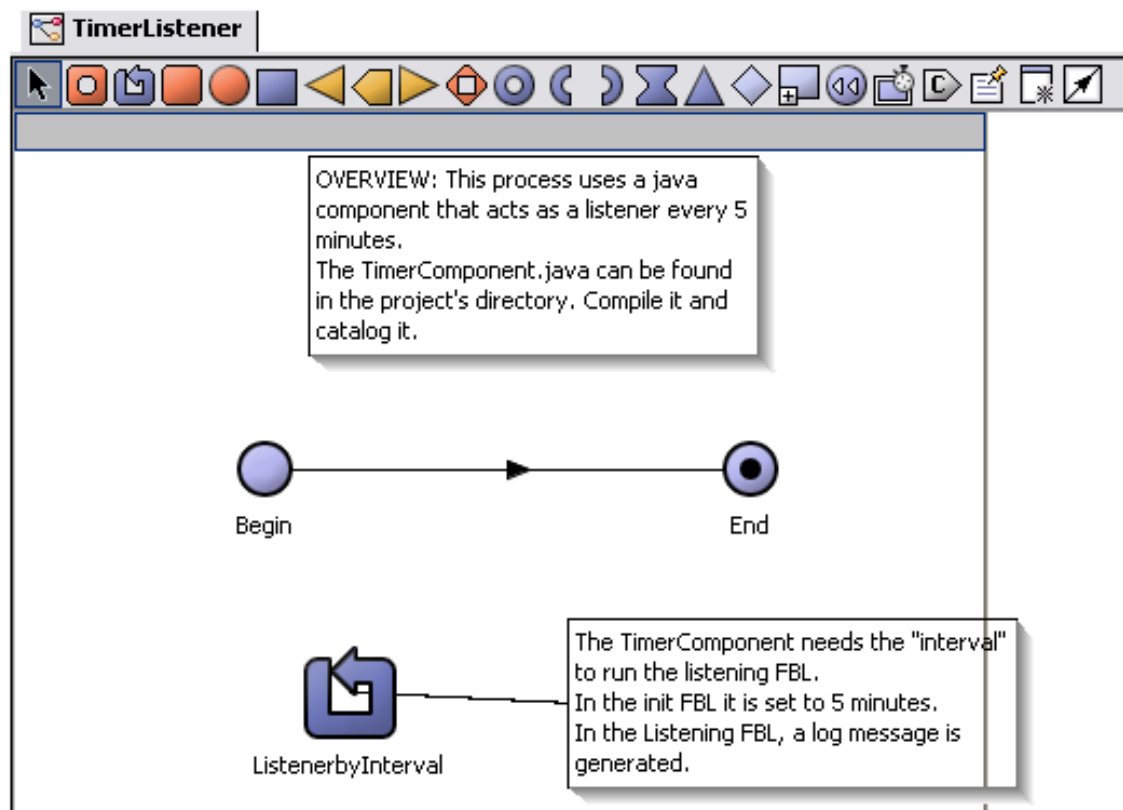
Find the project **GlobalAutoListCase01.fpr** in FuegoBPM's installation directory, in the studio/samples directory to study the example further.

The FuegoBPM's participant name is: **test1**.

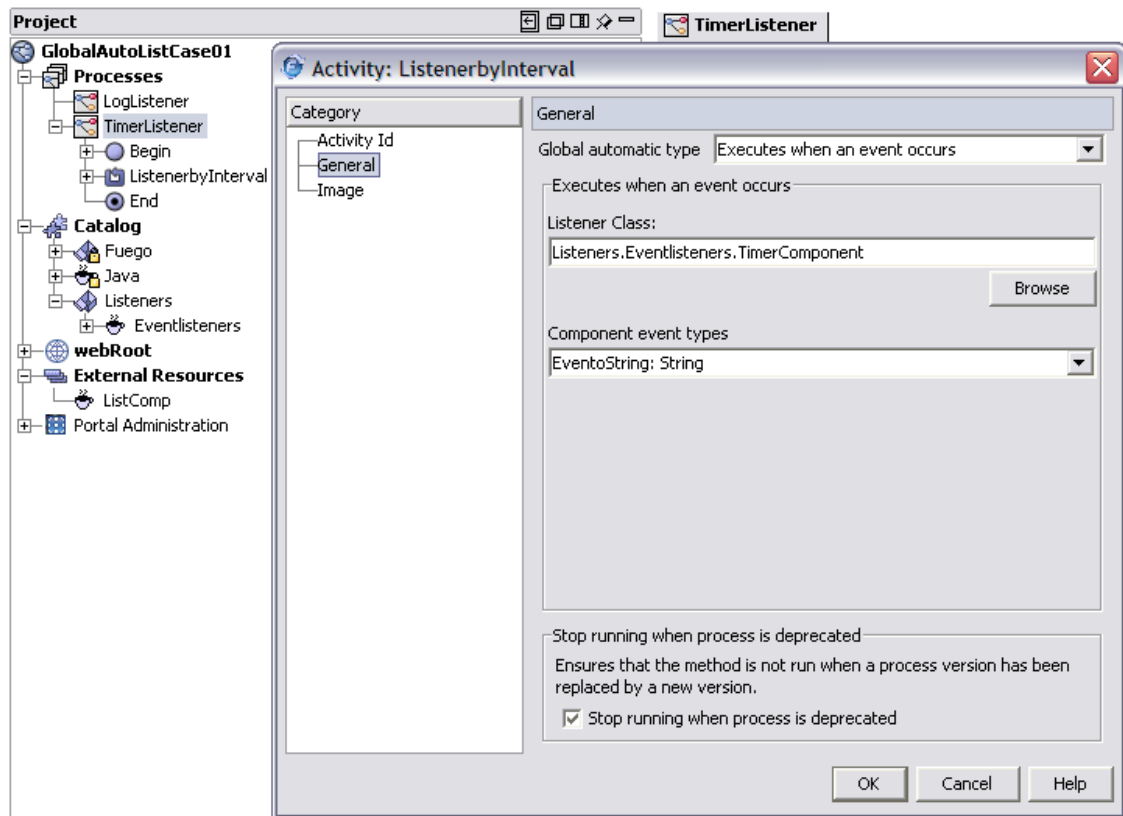
Case 2: Implementing an external event listening component (GlobalAutoListCase01.fpr)

The following example implements an external event listener called **TimerComponent**.

Every "n" seconds, the listening FBL is executed.



The Global Automatic activity **ListenerbyInterval** implements this external component:



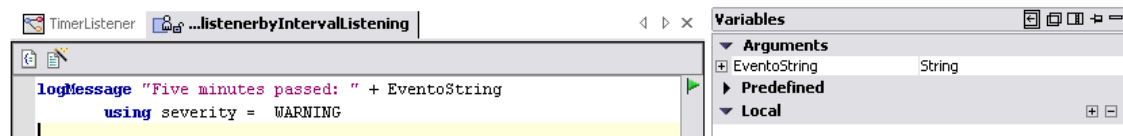
Init FBL

It passes the *interval* in between each event.

```
componentArguments = ["interval" : 300]
```

Listening FBL

It is executed every "n" seconds depending on the passed *interval* and creates a log message.



In this case the external component passes the "EventoString"

argument with a message that can be used in the FBL.

Find the project **GlobalAutoListCase01.fpr.exp** in FuegoBPM's installation directory, in the studio/samples directory to study the example further.

Find, as well, in the project's root directory, the TimerComponent java file that works as the Listener component:

This component is an example of a Listener that needs to be compiled and catalogued.

The FuegoBPM's participant name is: **test1**.

Global





Global activities are primarily used to allow end users to run applications or database queries only when needed. These applications are not an integral part of the process, but they contain information that can be accessed on an "as needed" basis. As a result of this, Global activities can be assigned to any user-defined role. Global activities never have transition lines going to or coming from them. They do not have interaction with instances either.

Global activities are useful to easily run lookup queries on databases, to quickly send e-mails or to invoke any applications that will help the end user as he or she interacts with the process.

As well you can implement certain predefined functions using a **Global activity** as:

- Process image: shows the instance within the process graphic
- Display instance: you can custom the instance panel.
- Workload: shows the process workload
- Dashboard: shows business activity monitoring based on stored information.

Global activity characteristics

Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			 /


Work Portal and the Global activity

The Global activity is visible in Work Portal, but only to the user(s) assigned to the role where the Global activity exists.

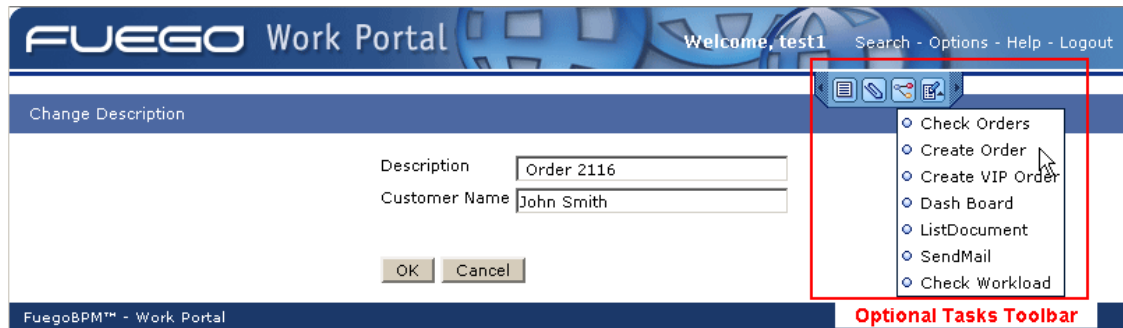
If the Global activity does not have access to the instance, it can be executed from the Work Portal's **Application view**.

If the Global activity has access to the instance, it can be executed from the Work Portal next to the instance.

Additionally, some Global activities can be executed from the **Optional tasks toolbar**:


- All Global activities that do not have instance access, for example the Workload global activity
- Global activities that have instance access but defined as Read-Only.
- Global activities defined as Process image, the  icon appears directly in the toolbar.

When you execute a Main task you have enabled the optional toolbar.



In the picture above, the **Check Orders** and **List Document** tasks are Global activities with instance access and set as read-only.

Check Workload and **SendMail** are Global activities with no instance access.

Finally, the  icon that appears in the toolbar indicates that a Global activity has been implemented as **Process Image**

Roles and the Global activity

Global activities reside in user-defined roles.

Variables and Global activities

Global activities can access predefined, local and argument variables from the BP-Method Editor.

Preconditions

No preconditions are required for the Global activity. It is an "as needed" activity that can interface with an instance. or be an independent activity with no relation to an instance.

Post-conditions

There are no post-conditions for the Global activity.

Properties

The Global **Activity Properties** dialog box allows you to enter a

description of the activity. See [Create an Activity](#) for further information.

Has instance access: the Global activity can apply to an instance or not. Based on this property, the activity can implement different types of tasks.

Use activity for instance presentation: if the activity has instance access, you can customize the instance information when you select to work with it. It indicates that the instance display information will not be the default one but built with a Component, Method, Screenflow or the instance's variables.

Transitions and Global activities

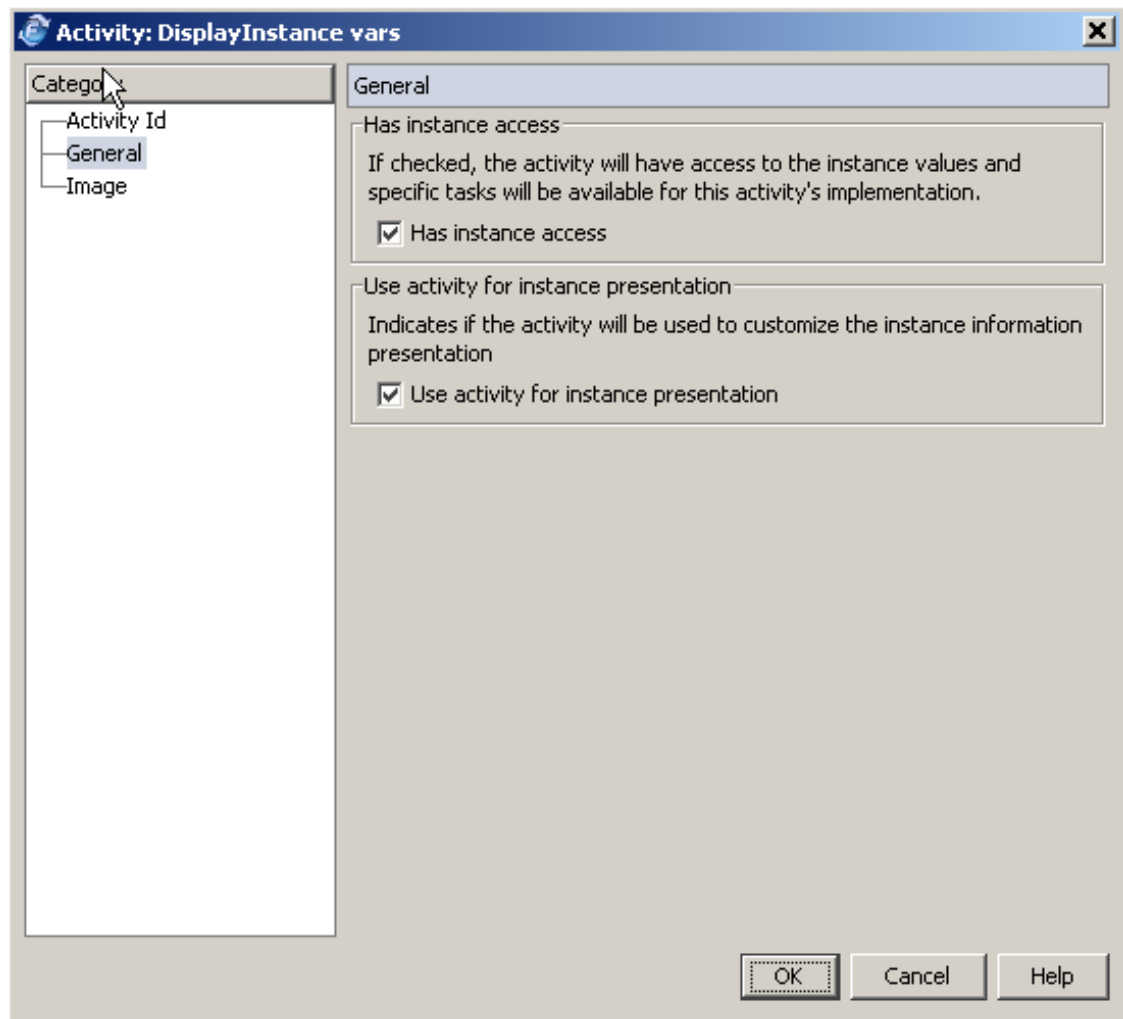
There are no transitions to or from the Global activity.

Tasks

The task has to be defined by the developer. The global activity can have only one task to be completed, that is, the **Main Task**.

The Global activity can have **instance access** or no interaction with an instance. Based on this property, the task can have different type of implementation.

Has instance access

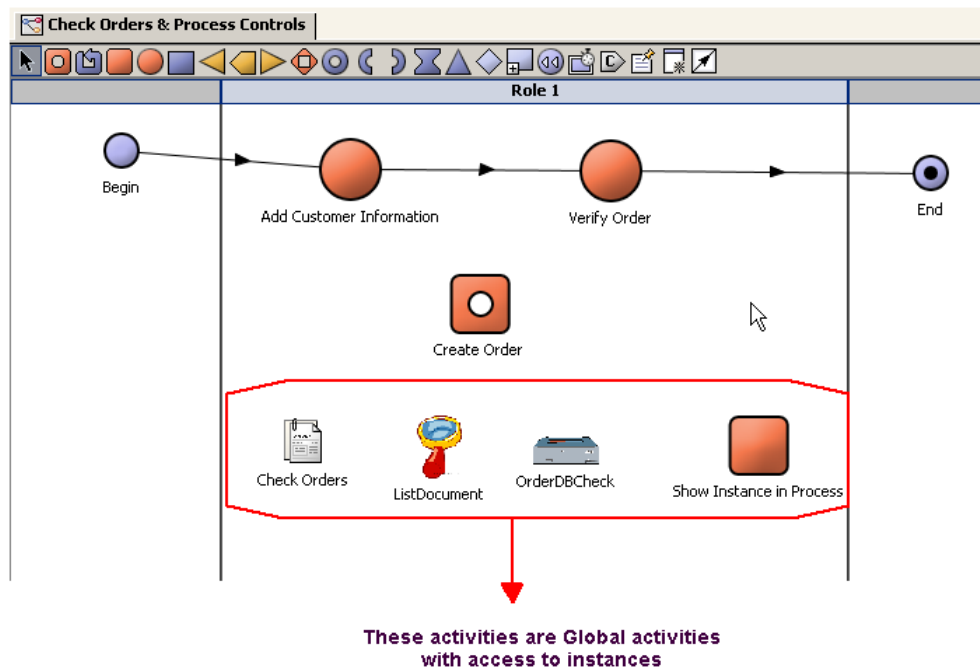


- Method: executes the defined method.
- Component: executes the defined component.
- Screenflow: executes the defined screenflow.
- Show Process Image: shows the instance within the process image.
- Display instance variables: when you double click on an instance in the Work Portal, you can implement a different display than the one that FuegoBPM provides as default and display the instance's variables.

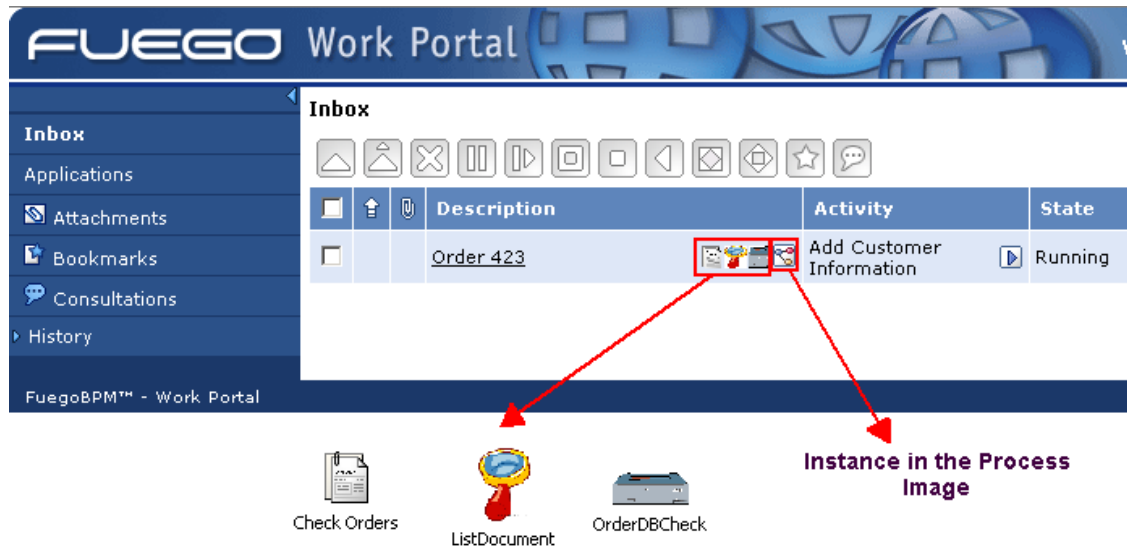
As these tasks have access to the instance, they can be executed based on certain information of the instance.

These type of global activities appear next to the instance (excluding the *Display instance variables* option).

For example, if you define the following Global activities as *has instance*:

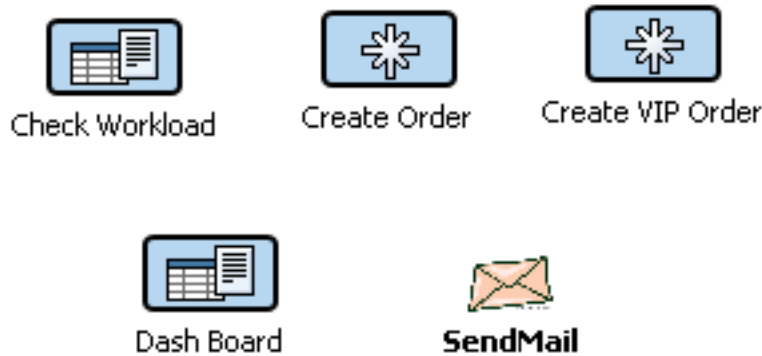


In the Work Portal you see icons next to the instance. You can run these task at any time:



No instance access

- Method: executes the defined method.
- Screenflow: executes the defined screenflow.
- Show Workload: shows the number of instances within each activity
- ShowDashboard: you can define a Dash Board and implement a Global activity to run it in the Portal.



These type of global activities appear as Applications.

FUEGO Work Portal Welcome, test1 Search - Options - Help - Logout			
Applications			
Inbox			
Applications	Description	Process	Documentation
Attachments	Check Workload	Check Orders & Process Controls	
Bookmarks	Create Order	Check Orders & Process Controls	
Consultations	Create VIP Order	Check Orders & Process Controls	
History	Dash Board	Check Orders & Process Controls	
	SendMail	Check Orders & Process Controls	

Method

The following example shows how to use the Global activity *List Customers*, which performs a database query that returns a list of customers in order to let any corporate account manager (CorpActMgr role in the example below) do a quick lookup to find a customer's name.

In the following Method, local and argument variables are used to query the database to find a customer's name. The database column name is *customer_id*.

Note

The name of this database is CUSTOMER. In order to try this Method, change the database name to the name of a table that exists in your

database. Define a local variable "customerNumber". Typing the database name all in capital letters is an SQL convention. It is not necessary for the BP-Method, but it can make the BP-Method more readable.

```
// Prompt for a customer to select  
  
input "Enter the customer number to retrieve" customerNumber  
    using title = "Select Customer"  
load CUSTOMER using customer_id = customerNumber  
display "The customer name is: " CUSTOMER.name
```

Component

Refer to the Tasks documentation to implement the Global activity with a Component.

ScreenFlow

Refer to the Screenflow documentation.

Show Process image

Global activities can be implemented to show where an instance is within the process.

To enable this function, the Global activity has to be defined as **Has instance access**.

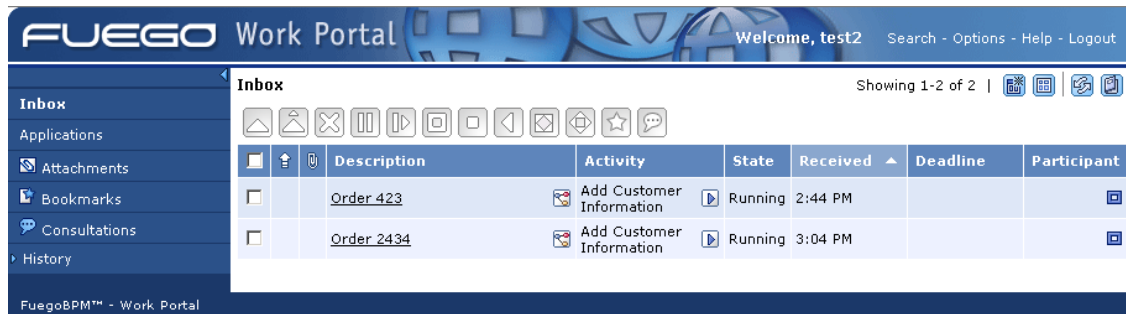
Within the Global activity **Properties**, in the **General Category**, enable this option.

The Main task is implemented as **Show Process Image**





All the participants that belong to the role where the Global activity is defined, can see where the instance they are working with is.







In the Work Portal, next to the instance, you see the Process image

icon .



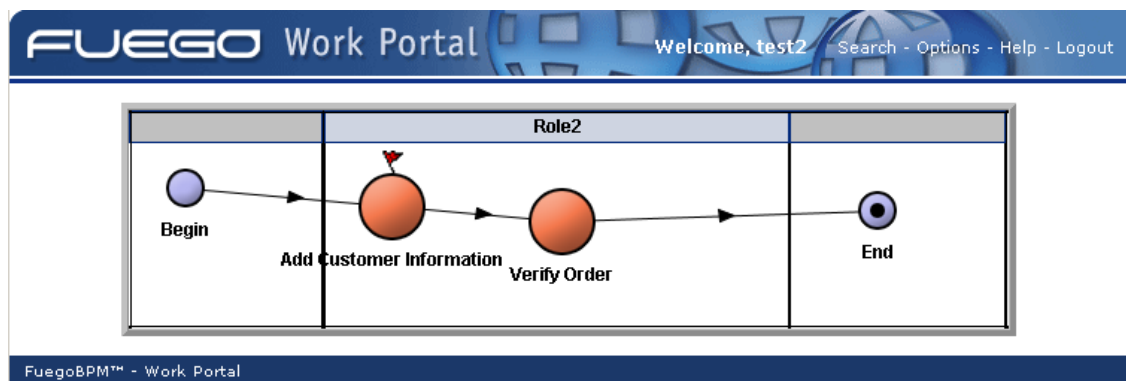
FUEGO Work Portal Welcome, test2 Search - Options - Help - Logout

Inbox Showing 1-2 of 2 |    

	Description	Activity	State	Received	Deadline	Participant
<input type="checkbox"/>	Order 423	 Add Customer Information	 Running	2:44 PM		
<input type="checkbox"/>	Order 2434	 Add Customer Information	 Running	3:04 PM		

FuegoBPM™ - Work Portal

Click on it to see the Process Image and a flag indicates where the current instance is.



Display Instance variables

If you check the property *Use activity for instance presentation*, you can customize the information to display with the selected external variables.

In Work Portal when you double click on an instance, you see the default panel for the instance. If you implement this option, the selected instance's variables are additionally shown in the Details section.

Select the external variables you want to see:

Main task - Activity DisplayInstance vars

DisplayInstance vars

Properties

☐ Read Only

Implementation type

Display Instance Variables

Choose instance variables to show

Instance variable	Choose
nameCust	<input checked="" type="checkbox"/>
numCust	<input checked="" type="checkbox"/>

OK Cancel Help

when you select an instance in Work Portal the variables are shown as well:

Inbox > Order 3

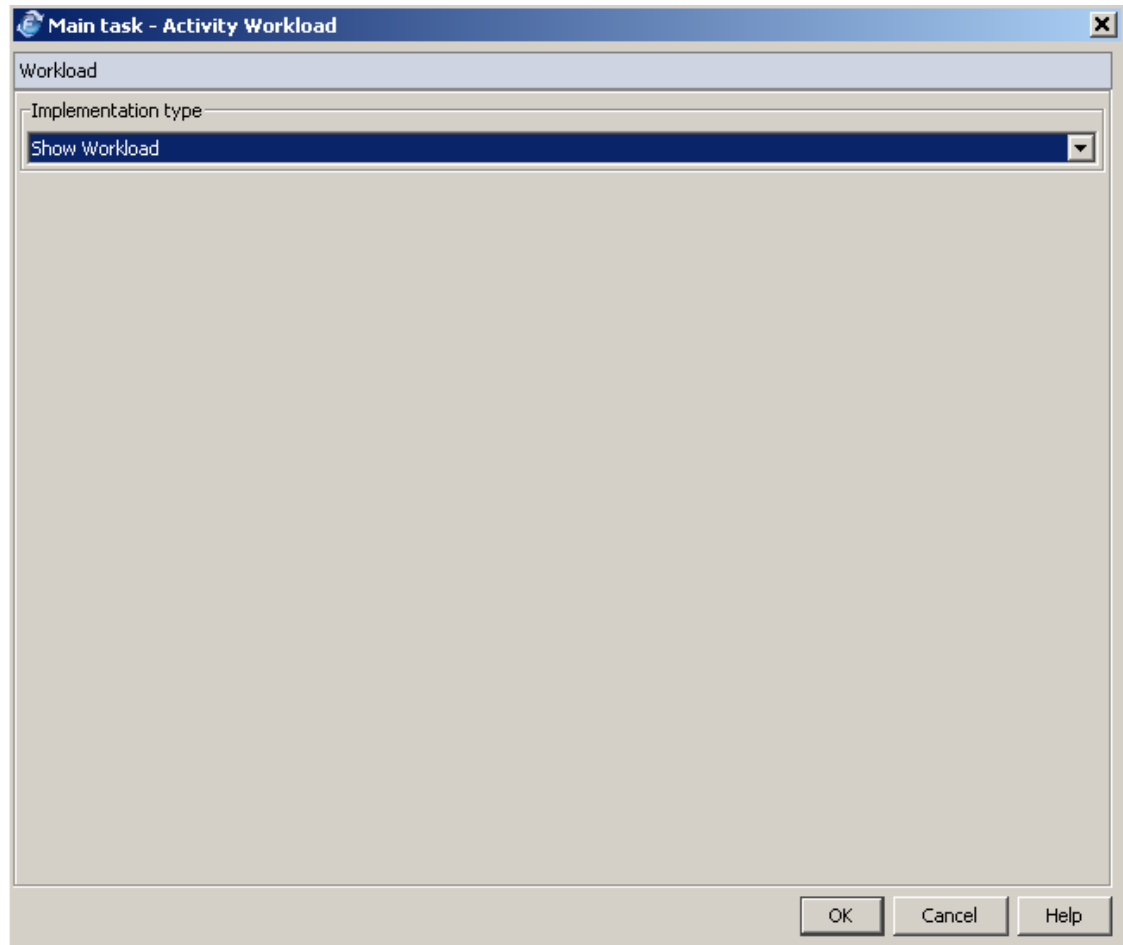
Details			
Process:	Check Orders & Process Controls	Activity:	Add Customer Information
Priority:	Normal	Status:	Running
Received:	Jan 13, 2005 4:21:41 PM	Deadline:	
Participant:		Copy:	0
nameCust:	Kate Winston	numCust:	1,425

Show Workload

To be able to see the instances workload within the process, you can define a Global activity implemented with this function.

To enable this function, the Global activity has to have the **Has instance access** property **disabled** (turned off).

The Main task is implemented as **Show Workload**



All the participants that belong to the role where the Global activity is defined, have this activity enabled in the **Application Folder** in the Work Portal.

In the example the new application called **Check Workload** is displayed:

FUEGO

Work Portal

Welcome, test1

Search - Options - Help - Logout

Inbox

Applications

Attachments

Bookmarks

Consultations

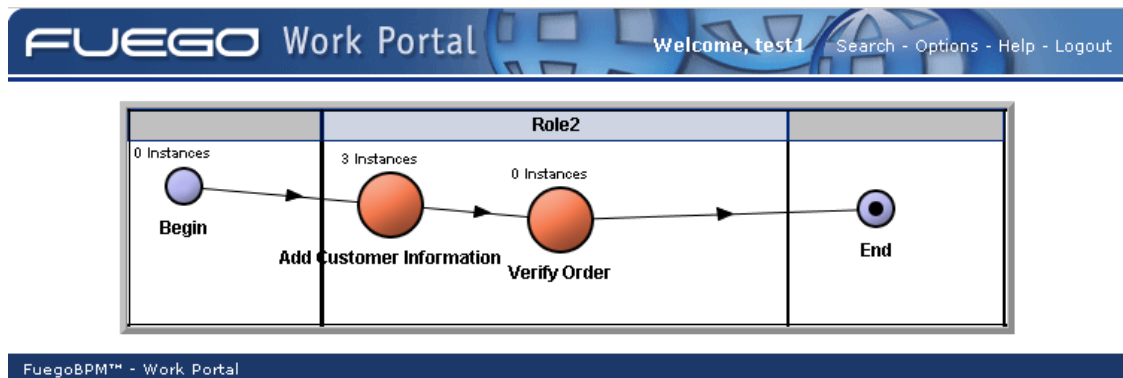
History

Applications

Description	Process	Documentation
Check Workload	Check Orders & Process Controls	
Create Order	Check Orders & Process Controls	
Create VIP Order	Check Orders & Process Controls	
Dash Board	Check Orders & Process Controls	
SendMail	Check Orders & Process Controls	

FuegoBPM™ - Work Portal

Click on it to see the Process Image and the number of instances that are in each activity.



Show Dash Board

You can implement a Dash Board to define a certain monitoring function based on stored information.

To enable a Dash Board, you define a Global activity that has to have the **Has instance access** property **disabled** (turned off).

The Main task is implemented as **Show Dashboard**

Main task - Activity Dash Board

Dash Board

Implementation type
Show Dashboard

Dashboard Display Properties
Allows to show the dashboard in a new window of the browser
☒ Open dashboard in new window
Shows the full control toolbar of the browser for the new window (only if Open in New Window property is enabled)
☐ Show browser's full control toolbar

Component method
Fuego Object Name
Select a Fuego Object and a presentation to be shown as a Dashboard in the Work Portal.
Component name:
FuegoObjects.CustMgmtProcessDBoard
Browse

Fuego Object Presentation
Presentation name
CustomerAverageProcessTime

OK Cancel Help

You can choose to open the dashboard in a new window and show the browser with full toolbar control.

To implement this kind of Global activity, you need first to implement a Fuego Object and a presentation to be shown as a Dash Board.

- **Component name:** select a Fuego Object created to show a Dash Board.
- **Presentation name:** select the presentation.

All the participants that belong to the role where the Global activity is defined, have this activity enabled in the **Application Folder** in






the Work Portal.

Click on the Global activity to see the defined Dash Board.

For further information about Dash Board implementation, see Dash Board Example

Other activities

Connectors

When two activities are not close enough to each other to be displayed at the same time in the designing workspace, you can add a *connector*,  /  /  /  / , which represents a shortcut to an activity. Connectors eliminate the need to draw a transition line across a large process design to connect two activities.

To add a connector

1. Right-click on any role and select **Add connector to**. Or select the connector from the designer toolbar. The list of available activities in the process is displayed.
2. Select the activity which the connector will refer to.
3. The connector icon is displayed on the design area with the name of the activity that it is representing.
4. Draw a transition from the source activity you want to connect to the connector. Remember, the connector represents another activity in the process design - it is now the target activity for the transition.

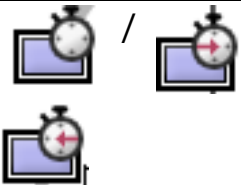



Measurement Marks

Measurement marks are checkpoints in the process to measure time or business variables.







When the Server routes the instance through a transition with a Measurement Mark, it performs all the checkpoints associated with the transition including the list of business variables which values the user wants to persist.

The information generated can be retrieved in the Audit Trail in the Portal or used for the Datawarehouse.

Measurement mark characteristics

Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			

There are different types of measurement marks:

- Snapshot Start  / : indicates the start of a measurement. Time begins to run. As well, business variables are persisted.
- Snapshot Stop  / : indicates the end of the measurement. Elapsed time is marked. As well, business variables are persisted. All Stop Measurement marks are referred to a Start one.
- Snapshot Start & Stop  / : persists business variables values.

Work Portal and the Measurement activity

Measurement activities do not appear in Work Portal. The information that generates can be retrieved in the Audit Trail.

Roles and Measurement activities

Measurement activities can appear either in automatic roles or in the user defined role types. However, if the Measurement activity is in a user-defined role, it will not appear in Work Portal.

Properties

Drag the **Measurement mark** icon from the Studio toolbar.

Add the **name** and the **description** . Define the type of required measurement. If it is **Snapshot Stop** mark, you have to define to which **Start measurement mark** it corresponds to.

All the **business variables** of *Measure* object type, that are defined as instance variables (external) in the process, are displayed. **Include** those you want to persist its value each time an instance passes through that mark.

Note



If a business variable is not listed, verify if it is of *measure* type and if it has been declared as instance variable within the process.

Transitions and Measurement marks

Measurement marks are associated to one transition.

When the server routes the instance through that transition it performs all the checkpoints associated with the transition.

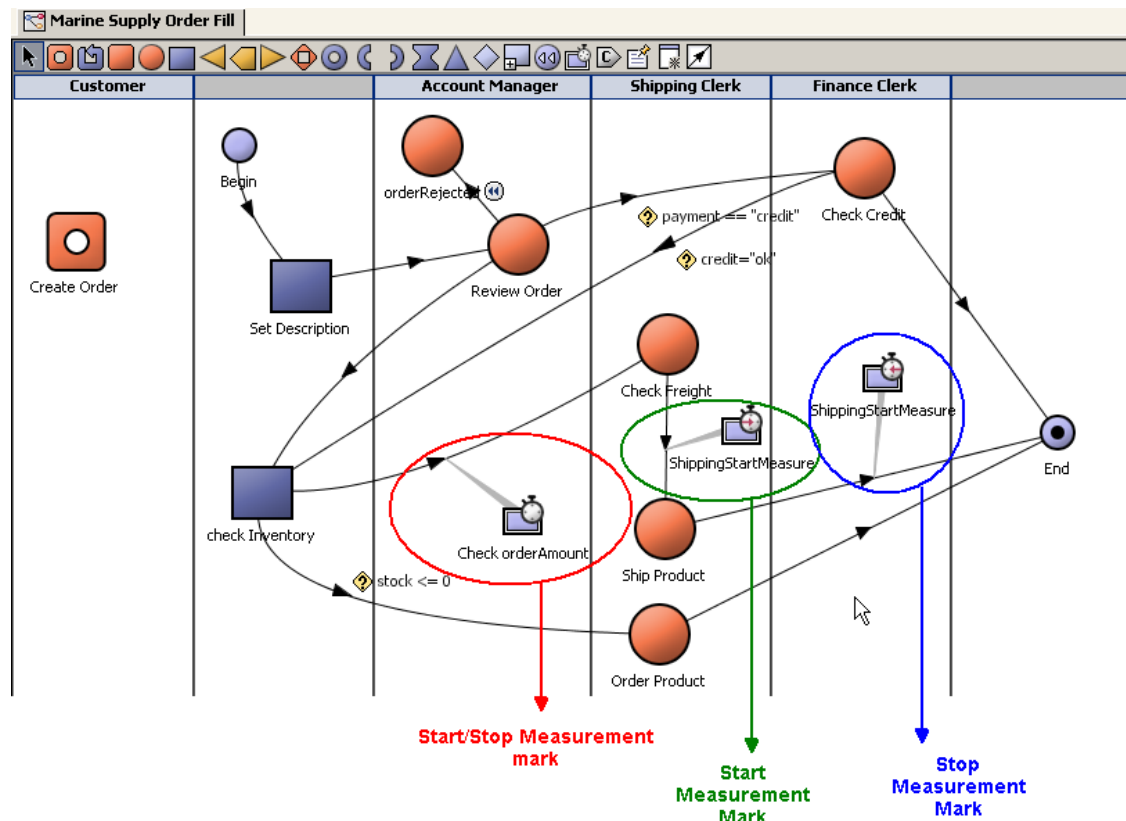
Audit Trail in the Portal

Measurement marks are registered in the Audit Trail.

They show the values of the defined business variables.




The Stop marks show also the elapsed time since the Start mark.

For example, for the following process:



The Audit Trail is:

Audit trail					Help
Marine Supply Order Fill > End > Scubapro Dive Shops OrderFill6					
Activity	Event	Responsible	Date	Copy	
[-] Create Order	Completed		Nov 22, 2004 3:28:12 PM	0	
[-] Review Order	Completed		Nov 22, 2004 3:28:13 PM	0	
[-] CheckOrderAmount	Processing		Nov 22, 2004 3:28:28 PM	0	
[-] Measurement Start Stop	Measurement Start Stop	Engine	Nov 22, 2004 3:28:28 PM	0	
	'orderAmount' = 156.35		Nov 22, 2004 3:28:28 PM	0	
[-] Check Freight	Completed		Nov 22, 2004 3:28:28 PM	0	
[-] ShippingStartMeasure	Processing		Nov 22, 2004 3:28:44 PM	0	
[-] Measurement Start	Measurement Start	John Smith	Nov 22, 2004 3:28:44 PM	0	
	'orderAmount' = 156.35		Nov 22, 2004 3:28:44 PM	0	
[-] Ship Product	Completed		Nov 22, 2004 3:28:44 PM	0	
[-] ShippingStopMeasurement	Processing		Nov 22, 2004 3:29:38 PM	0	
[-] Measurement Stop	Measurement Stop	John Smith	Nov 22, 2004 3:29:38 PM	0	
	'Elapsed Time' = 54s		Nov 22, 2004 3:29:38 PM	0	
	'orderAmount' = 156.35		Nov 22, 2004 3:29:38 PM	0	
[-] End	Completed		Nov 22, 2004 3:29:38 PM	0	

 **Measurement Mark Started**
 **Measurement Mark Stopped**
 **Measurement mark Star/Stop**

See Datawarehouse and Business activity monitoring for additional information.

Datawarehouse

These measurement marks are consolidated into the datawarehouse.

See Datawarehouse and Business activity monitoring for further information.

Chapter 5. Tasks

Task

A task consists of one or more actions that need to be executed in order to achieve an activity's goal. Each type of activity within FuegoBPM can contain only one task, multiple tasks or no tasks at all.

There is a **Main task** and, additionally, some **Optional tasks** can be added. Activities that require human intervention can have *optional tasks*, such as the Interactive activity and the Grab activity.


The **Main task** is automatically generated and the optional tasks have to be added and defined by the developer. The associated BP-Methods will also be generated.

Some tasks are **mandatory**, which means that they should be processed before the instance can be sent to the next activity in the business service. Additionally, a task can be defined as **repeatable**, which means that this task can be performed as many times as necessary until the instance moves to the next activity.

The first task in the list is the **Main task**, and the order of all **Optional tasks** is the order in which they appear on the list within the dialog box where they are defined.

Tasks appear in a list in Work Portal. A Work Portal user selects the task to run in the order that he or she chooses.

Properties

Main tasks as well as **Optional tasks** have properties that define their behavior. Their dialog boxes are very similar, the only difference is that you can define multiple **optional tasks**. Therefore, these tasks can be added by using the left panel displayed in the dialog box. Add additional tasks by using the  icon.

Moreover, you can define the execution order of the optional tasks by sorting them in the correct order. Use the up and down arrows at the right corner of the left panel.

MAIN TASK

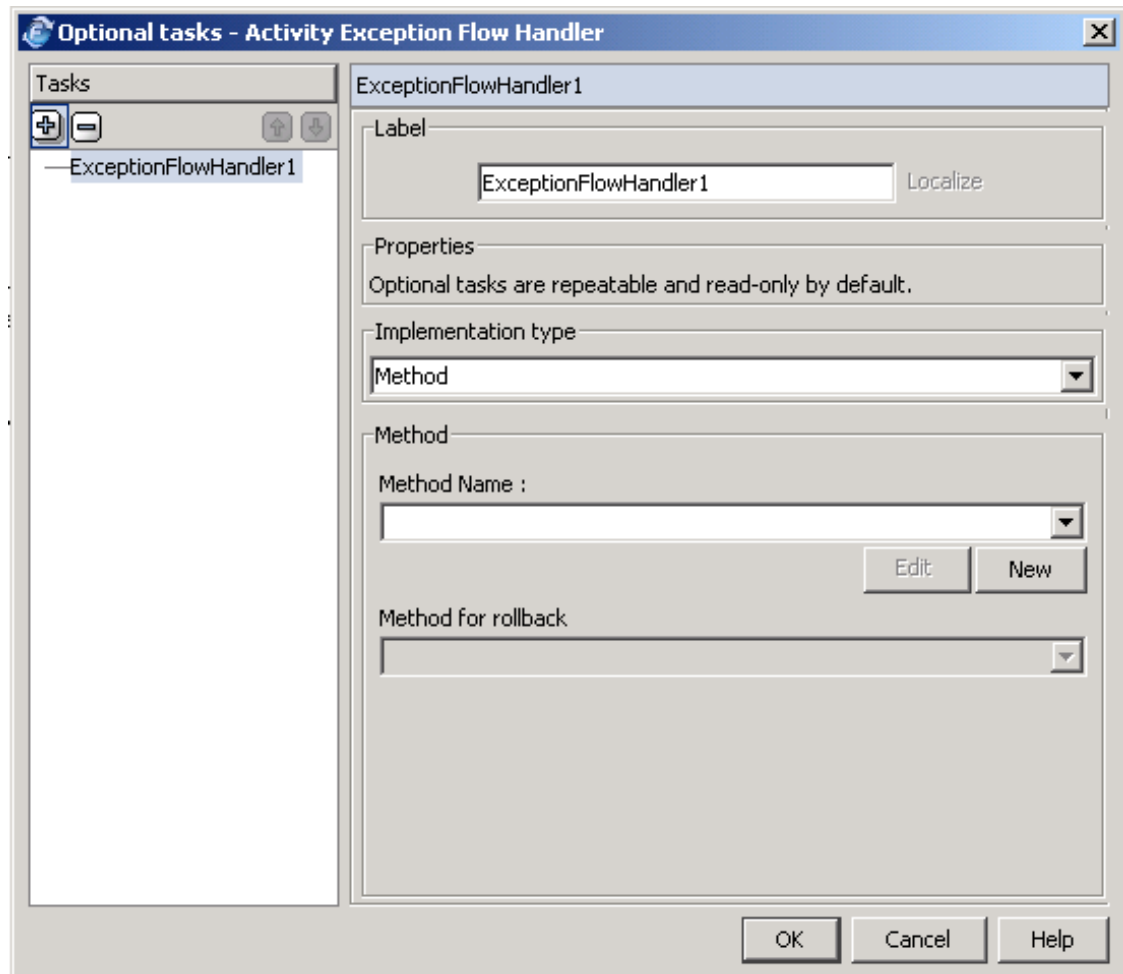
The screenshot shows a dialog box titled "Main task - Activity Exception Flow Handler". It contains the following elements:

- ExceptionFlowHandler**: The name of the handler.
- Properties**: A section with two checkboxes: ☒ **Mandatory** and ☐ **Read Only**.
- Implementation type**: A dropdown menu currently set to **Method**.
- Method**: A section containing:
 - Method Name :** A text field with the value `exceptionFlowHandler` and a dropdown arrow.
 - Buttons**: **Edit** and **New** buttons.
 - Method for rollback**: A text field.
- Buttons**: **OK**, **Cancel**, and **Help** buttons at the bottom right.

OPTIONAL TASK

Optional tasks by default are **repeatable** and **read-only**. In order to change the optional tasks properties, you should enable the **Enable optional tasks properties** in the **Project Preferences / Activity** (within the **File** menu.)

With no properties enabled



With properties enabled

Label: represents the name of the task.

Repeatable: the task can be repeated more than once.

Mandatory: the task is mandatory for the instance and should be completed before the instance can leave the activity.

Read only: any change to any instance and the variables (instance variables, predefined variables, etc) within the BP-Method will be ignored. On the other hand, as the task is on **read only** mode, the instance will not be locked.

Note

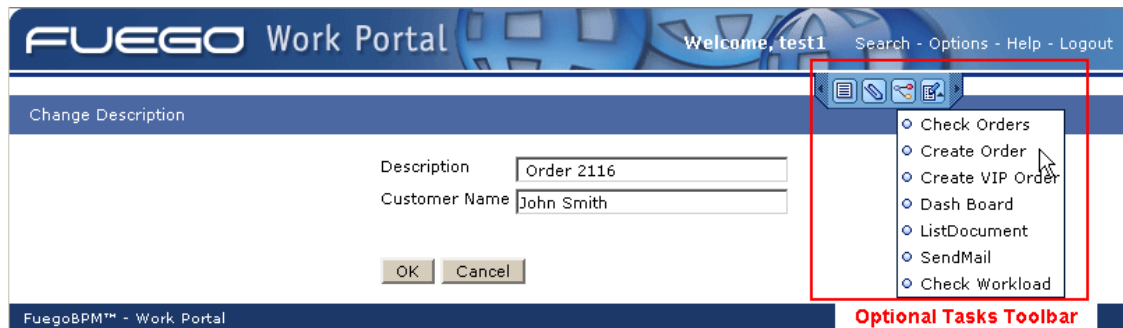


If the **Repeatable**, **Mandatory** and **Read only** properties for

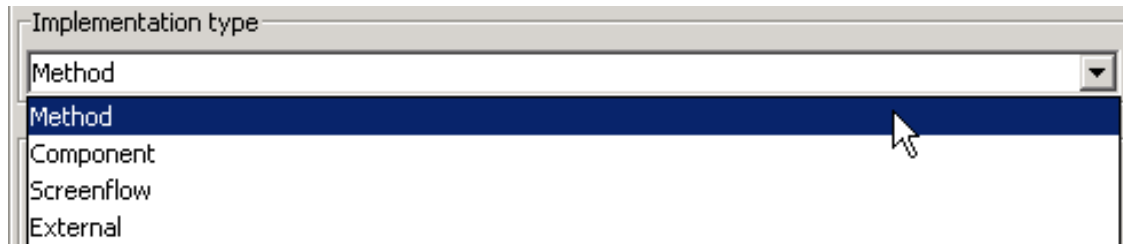
Optional tasks do not appear, set the corresponding preference. Select the **File** menu and the **Preferences** and **Activity** options. Select **Enable optional tasks properties**.

Optional Tasks and the Work Portal

The Optional tasks appear in the **Optional tasks toolbar** that appears when you execute the **Main task**.



Implementation type:



- **Business Method** (See Business Method Overview).
 - *BP-Method name*: Defines the BP-Method that will be executed when the task is performed.
 - *Edit/New*: You can edit the selected BP-Method or create a new one and associate it to the task.
 - *BP-Method for rollback (available only for Interactive or Grab activities)*: Defines the BP-Method to execute in case of a rollback situation. See BP-Method for Rollback on this page.

Tip

You can drag a BP-Method from the left panel (Structure Pane) and drop it on an activity. The task will be automatically set to run this Method. You can also drag a defined Screenflow or Procedure in the Project from the project panel. If it is an **Automatic** activity, the task implementation will be replaced with the new selected action. If it is an **Interactive** or a **Grab** activity, a new task (optional task) will be created.

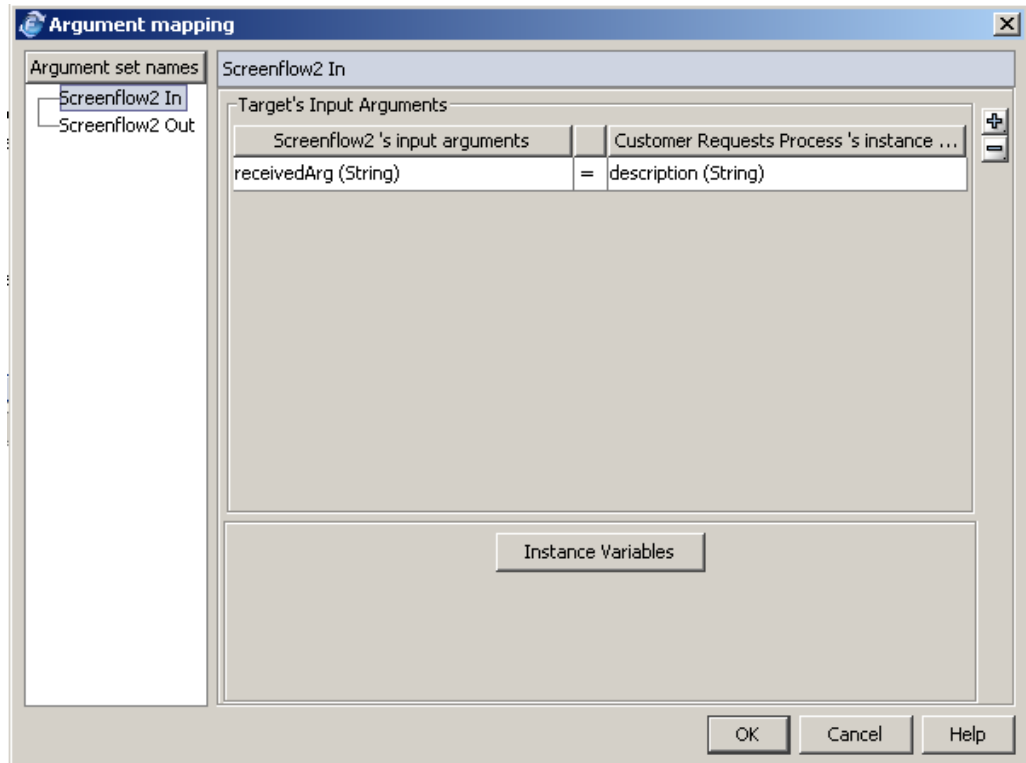
- **Component:** The task will run the defined Component method. (See Component.)
 - *Activity task execution timeout:* set the timeout for the component call execution (minutes:seconds). After this time occurs, the server cancels the component execution. See Execution timeout for detailed information.
 - *Instance variable:* Indicates that the invocation of the method will be applied to the selected variable. By selecting a variable, the Component name is implied and taken from the variable type. A typical example of instance variable to use is a Fuego Object instance variable.
 - *Component Name:* Represents the type of component to be invoked. If an instance variable is selected, this value will be filled out from the variable type and cannot be edited from here. Complete the component name or browse to choose the component within the Project Catalog. If the task corresponds to an Interactive activity, you can only select a component defined to run on the client-side.
 - *Method name:* Indicates the method to execute (defined within the component).
 - *Presentation name:* In the event that the selected component is a Fuego Object, a presentation can be selected. In this case, the method name is grayed out since an input statement will be used on the Fuego Object. All existing information in the

selected variable will be displayed in the presentation.

- *Argument Mapping*: If the method requires any in or out arguments, these arguments have to be mapped to instance variables or predefined variables that contain the value/s to be passed. Additionally, if you are using an instance variable (for example a Fuego Object instance var) and the component updates any of the attributes within it, you need to explicitly map the Fuego Object instance variable to an argument variable. For example, map the Fuego Object to the *currentComponentInstance* variable.
- **Screenflow** (See Screenflow.)

The screenshot shows a software configuration window titled "Main task - Activity Input Customer Info". The window is divided into several sections. The top section is a header "Input Customer Info". Below it is a "Properties" section with three checkboxes: "Repeatable" (unchecked), "Mandatory" (checked), and "Read Only" (unchecked). The next section is "Implementation type" with a dropdown menu currently set to "Screenflow". Below that is the "Related Process" section, which includes a text field labeled "Screenflow name" containing the text "Screenflow2", and two buttons labeled "Edit" and "New". A dashed rectangular box labeled "Argument mapping" is positioned below the "Related Process" section. At the bottom of the window are three buttons: "OK", "Cancel", and "Help".

- *Screenflow name*: Indicates the screenflow defined within the project to be executed.
- *Argument set name*: Defines the argument mapping defined by the screenflow to invoke it.



- **External**: External tasks are tasks that are completely implemented outside FuegoBPM. These tasks are accessible from PAPI by calling the methods: `prepareExternalActivity(...)` `commitExternalActivity(...)` For further information see the PAPI's JavaDoc documentation. The External task implementation allows you to implement the interactive activity (more precisely the GUI presented to the user) outside FuegoBPM (for example using ASP pages). When you select the External implementation you have to specify two methods:
 - The `prepareExternalActivity()` method can be used to get any

values needed by the presentation before actually displaying it. This method extracts and processes information from the instance variables or Fuego Objects, and makes it accessible through its output arguments.

- The *commitExternalActivity()* method should be invoked once the user finished with the GUI and processing can continue in FuegoBPM. This method completes the task (and the instance if necessary), and maps its input arguments to instance variables. Both methods can declare input/output arguments that will be passed along whenever they are invoked. As well both methods need to be invoked through PAPI or through PAPI-WebServices. For example: The instance arrives to the activity and waits there until someone sends the *prepareExternalActivity()* method to it. It executes the method and answers back a set of arguments (valid values lists or predefines/default values to be used by the GUI). After the user finishes with the GUI (which could be implemented anyway you want), the *commitExternalActivity()* method is invoked and the values entered by the user are passed along so that FuegoBPM can use them (set them into a Fuego Object for example). Then the instance moves forward in the process (unless you finish the *commitExternalActivity()* method with an Action different than OK or RELEASE).
- *Configuration*: you can optionally define an URL. When the task is executed from the Work Portal, the Work Portal redirects the execution to the URL.

Activities and tasks

Activity	Number of Tasks	Automatically generated	Description
Begin	None	None	.

Activity	Number of Tasks	Automatically generated	Description
End	None	None	.
Global Creation	1	Yes	.
Global Automatic	1	Yes	.
Global	1	If it is a Method	This can be a Method and therefore it is created with the same name as the activity.
Interactive	Multiple: Main and optional tasks.	No	The developer should define as many tasks as required in the activity.
Automatic	Main one	No	The developer should define as many tasks as required in the activity. No repeatable or mandatory properties will appear for this type of activity.
Split	1	Yes	.
Split N	1	Yes	.
Join	1	Yes	.
Grab	Multiple: Main and optional tasks.	No	The developer should define as many tasks as required in the activity.
Subflow	None	None	.
Process Creation	None	None	.

Activity	Number of Tasks	Automatically generated	Description
Termination Wait	None	None	.
Notification Wait	None	None	.
Process Notification	None	None	.

BP-Method for Rollback

Tasks can have an associated BP-Method. Each BP-Method task is a single transaction to the FuegoBPM Server.

All changes that cannot be repeated with consistent results should be recorded to allow the rollback BP-Method to undo what the incomplete task started. This also means that, while the task is running, some indication mechanism should reside within the BP-Method task to track the portions of the BP-Method that have been completed so that, upon BP- Method failure, these changes can be rolled back.

Apart from databases, there are other application components that may or may not support transactions. For applications that do not support transactions, an understanding of any changes made is required to be able to roll back changes in the event of an incomplete BP-Method task.

In BP-Method tasks that connect to applications which support transactions, failing BP-Method will encounter the line of code that performs the commit, leaving an incomplete transaction. The rollback BP-Method can then use that transaction reference to tell the application to abort the transaction.

Note



BP-Method for Rollback must not contain interactive statements nor any function that requires a participant interaction as it runs in background and is triggered automatically

BP-Method Overview

A BP-Method is a high-level scripting language used to define the business rules and the logic of activity types and certain transitions within a process. Each activity type initiates a very specific action and the BP-Method provides the script for this action. BP-Method includes statements that integrate variables, expressions, operators and components.

See Methods Types for further information.

Editing a BP-Method

The BP-Method Editor is a freeform text editor that allows you to copy, cut and paste partial or entire BP-Methods objects, including statements, expressions, arguments, variables, operators and components.

BP-Method timeouts

Timeouts apply to BP-Method. Timeouts have a **default** value (5 minutes) that can be redefined using the predefined variable **timeout**. As well, no matter the defined timeout for each of them, they are all **limited** to the maximum specified in the Server property **Maximum BP-Methods timeout**.

See Execution timeout for detailed information.

Using relay to

The preferred method for dealing with timeouts is the *relay to* BP-Method statement. See Controlling threads with relay to for further information.

BP-Method execution results


When activity BP-Method tasks are executed, several results may occur. The following information provides a complete list of possible


instance behavior according to the result of the BP-Method. The predefined variable *action* is used to indicate the BP-Method result.

The FuegoBPM Server acts according to this variable and saves or undoes (commits or rollbacks) the changes.

The **action** variable is an enumeration that accepts the following valid values:

action =	Description	Result
OK	Indicates that BP-Method execution was successful. This is the default result.	BP-Method changes to "completed" status. If the activity is marked auto-complete on its Activity Properties dialog box, the instance flows to the next activity according to transition rules. Otherwise, the instance waits in Work Portal for further processing in the activity or for the user to click the Send icon.
FAIL	Indicates that the BP-Method has failed its execution. The BP-Method must be executed again, if it is so required.	The error is logged in the Web Console log (in the FuegoBPM Enterprise Product) or it can be visualized at the bottom of FuegoBPM Studio, in the Log tab. If a rollback BP-Method is included, it is executed. If the rollback BP-Method fails, the server will retry the original BP-Method until

action =	Description	Result
		<p>it succeeds or invokes the maximum number of retry times and is routed to an exception activity. The maximum number of retry times is set in the Web Console (FuegoBPM Enterprise Product) or in the Server Properties in FuegoBPM Studio. See Server Properties.</p> <p>Note</p> <p> A component exception is treated like Action=FAIL by the server.</p>
CANCEL	BP-Method execution is aborted.	The instance is rolled back to the point before BP-Method execution. No trace of the BP-Method failure or execution appears in the Audit Trail.
REPEAT	Indicates that the BP-Method execution is ignored.	The instance is committed. However, the task's status remains in a pending state. BP-Method should be executed again.
RELEASE	Ends the BP-Method execution.	The instance is released to the next activity without processing any of the BP-Methods in the current activity,

action =	Description	Result
		even if they are marked mandatory.
ABORT	Ends BP-Method execution and aborts the entire instance.	The instance is not processed and is sent directly to the End activity.  Warning: Instances that are aborted cannot be recovered.
BACK	Ends BP-Method execution and sends the instance back to the activity where the exception occurred.	Used in an exception handling activity to send an instance back to the activity where the exception occurred. If the exception can be corrected, fix it in the exception handling activity before sending it back.
SKIP	Ends BP-Method execution and sends the instance back to the activity where the exception occurred and skips it.	Used in an exception handling activity to send an instance back to an activity in the process. The instance goes to the point of BP-Method failure and is released to the next activity without re-performing the BP-Method that caused the failure, even if the BP-Method task is marked mandatory on the Implementation dialog box.

Note

A user-defined exception rolls the instance back and routes it to the appropriate exception handler.

Calling components from BP-Methods

A number of standard components (Fuego Blocks) are available at the time of FuegoBPM installation for you to use in your process design. You may also add your own Java, SQL, EJB, JNDI, Web Service and Fuego Objects.

The syntax for calling a component is the same, irrespective of the type of component you are using. In *FuegoBPM* skin, it will be as follows:

```
Syntax:  
[method name] [component name]
```

Example:

For example, a method name is *calculateTotal* and the component name is *Pricing* .

calculateTotal Pricing

See COMPONENTS for further information.

BP-Methods

For further information, see Business Method

Tasks Execution Timeout

Timeouts

Timeouts apply to **BP-Method** and **Component tasks**. Timeouts have a **default value** that can be redefined. As well, no matter the defined timeout for each of them, they are all **limited** to the maximum specified in the Server property **Maximum BP-Methods timeout**.

BP-Method timeout

Default timeout

By default the BP-Method has the timeout set to 5 minutes. You can change this value using the predefined variable *timeout*.


Extending the timeout

This method is not recommended for all situations. Instead, it should be used only occasionally. During the BP-Method execution, several Server resources are locked, therefore if you extend the timeout for each BP-Method, the risk of having all the resources locked increases and it may produce a bottleneck.

The syntax is as follows:

```
// timeout is an interval
// Increasing it to 20 minutes
timeout = '20m'
```

Note

 If you set the BP-Method timeout to be greater than the server property "Maximum BP-Methods timeout", the BP-Method will fail at runtime and a Server log message is generated

If you set the **timeout** predefined variable to **null**, then the Server property (**Maximum BP-Methods timeout**) value applies.

Timeout Limit

The FuegoBPM Administrator can limit the timeout to a maximum for all processes deployed in a specific Server using the Server property **Maximum BP-Methods timeout**. **Maximum BP-Methods timeout** is a tool for the FuegoBPM Administrator to ensure that the Server resources are enough to serve all deployed processes. If a BP-Method timeout is greater than the "Maximum BP-Methods timeout" property, such BP-Method will fail.

The **Maximum BP-Methods timeout** can be set from FuegoBPM Studio Server Properties as well as from FuegoBPM Web Console Server Properties.

Integrated Components within a BP-Method

When you use FuegoBPM Components within a BP-Method, you can set a timeout as one of the component attributes. This timeout applies to the component execution time.

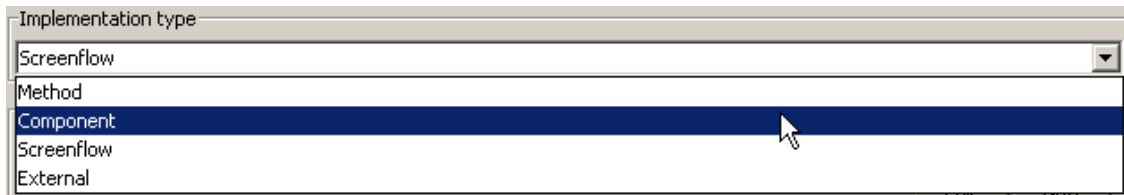
If the BP-Method runs one or more component, although they can have individual timeouts, the BP-Method timeout rules the full processing.

For example, the maximum BP-Method timeout is set to 3 minutes. You define a BP-Method that runs 2 FuegoBPM components (Component A and Component B), and both components have the *timeout* attribute set to 2 minutes.

Component A begins running and finishes in 1 minute, 45 seconds. Although Component B has an individual timeout set to 2 minutes, its execution will not last more than 1 minute, 15 seconds as passed that time the BP-Method execution is aborted.

Component task timeout

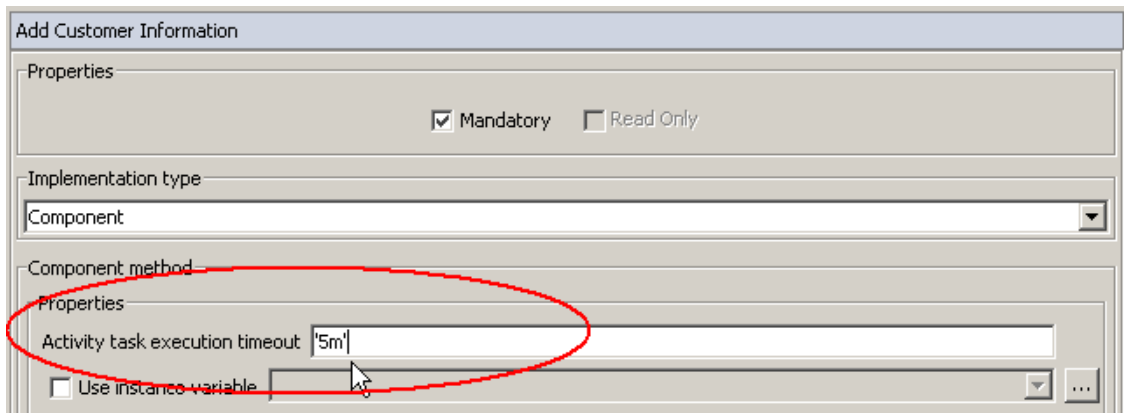
When you implement a task, it can be defined as a Component.



Default timeout and how to extend it

The timeout is set in the **Activity task execution timeout** that works the same way as the **BP-Methods timeout** but applicable to the tasks implemented through components (not through a BP-Method).

By default the **Activity task execution timeout** property has the timeout set to 5 minutes. You can change this value if required (minutes:seconds).



Setting Timeout limits

The FuegoBPM Server sets limits to the defined timeouts in two of its properties:

- Maximum BP-Methods timeout, and
- Interactive component timeout

Maximum BP-Method timeout

The **Maximum BP-Methods timeout** (seconds) is a property that ensures that the Server resources are enough to serve all deployed processes. If a BP-Method timeout is greater than the "Maximum BP-Methods timeout" property, such BP-Method will fail.

The timeout of a BP-Method is set by default to 5 minutes and the developer can change it using the predefined variable **timeout**. But although you can increase any individual BP-Method timeout, this Server property will limit that time. The new set timeout cannot exceed the Maximum BP- Method timeout set in this field. If you set a greater value, the BP-Method execution is aborted.

To determine this value, you must consider all the methods for all the processes running on the Server. The timeout must always be as short as possible and defined based on the standard situation (not considering exceptional cases). If the method does not complete in this period of time, the Server will abort the task's execution and all resources taken by this task will be freed and rolled back, given that they provide transactional control.

If the BP-Method times out, the Server will interrupt the execution of the BP-Method and assume that it has failed.

Examples that apply are:

- Automatic activities,
- BP-Methods that don't use relay-to,
- Methods from a Fuego Object that run on server, etc

Interactive component Timeout

The **Interactive component timeout** (minutes) limits the time the

Server will wait for a component that runs on the client to be completed. In general, it doesn't need to be changed, since it applies only to interactive components

implemented as **screenflows** or **BP-Methods that use relay to** (two ways of achieving the same result).

While such a call is in progress, the Server temporarily selects the task for the currently running participant. This timeout specifies the maximum time this type of task selection can be held. Note that this timeout can be safely set to several hours (or even days) because no resources are locked in the Server side while the component is running in the client side.

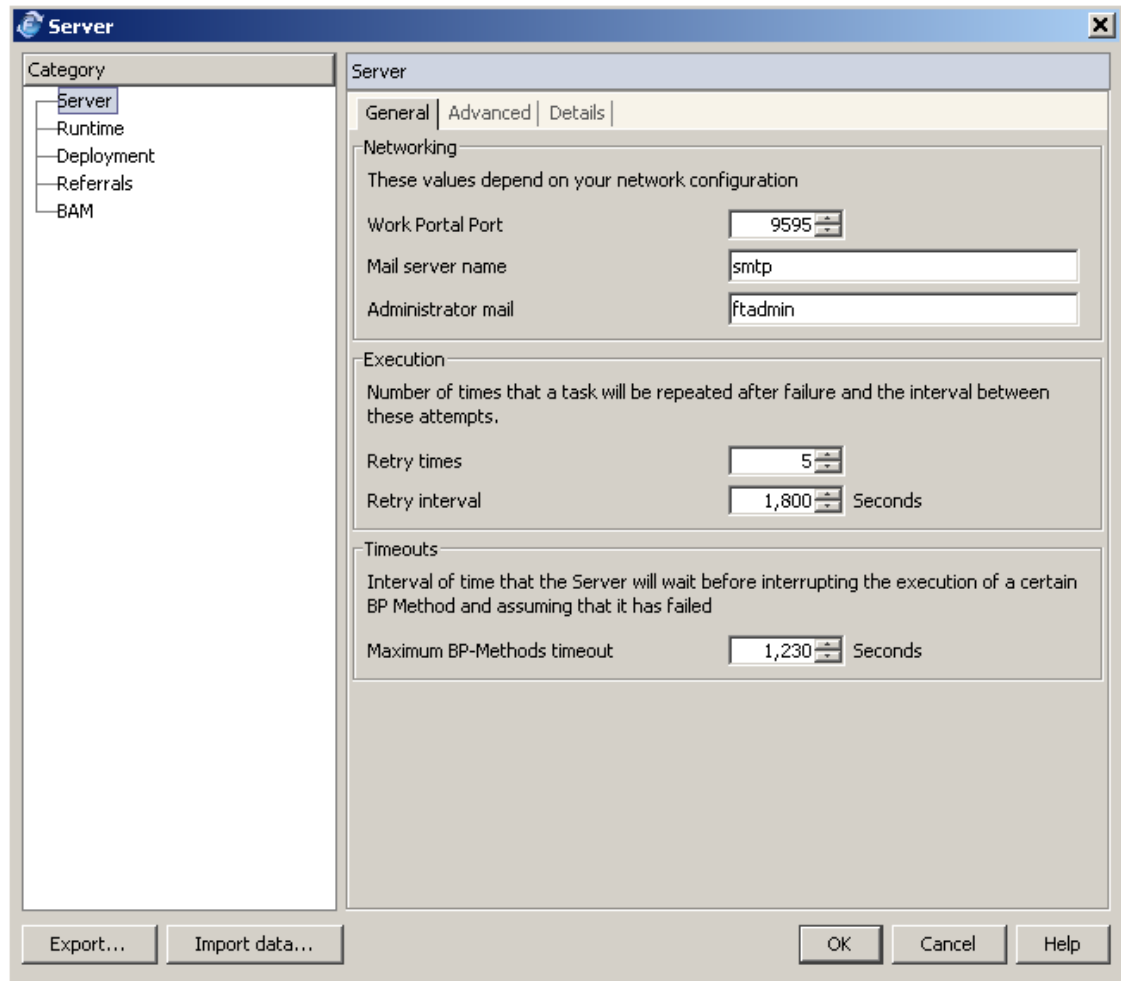
Examples where this timeout applies are:

- Fuego Object Presentation,
- Interactive component call within a Screenflow,
- Within a BP-Method, a relay-to execution. For example, a process activity task may be composed by just one BP-Method but can also be a BP-Method calling many other BP-Methods through the *relay to* feature. A task composed by many BP-Method joined by relay to, can also have set a timeout (regardless that each BP-Method timeout will keep working as specified above).

FuegoBPM Studio Server Properties

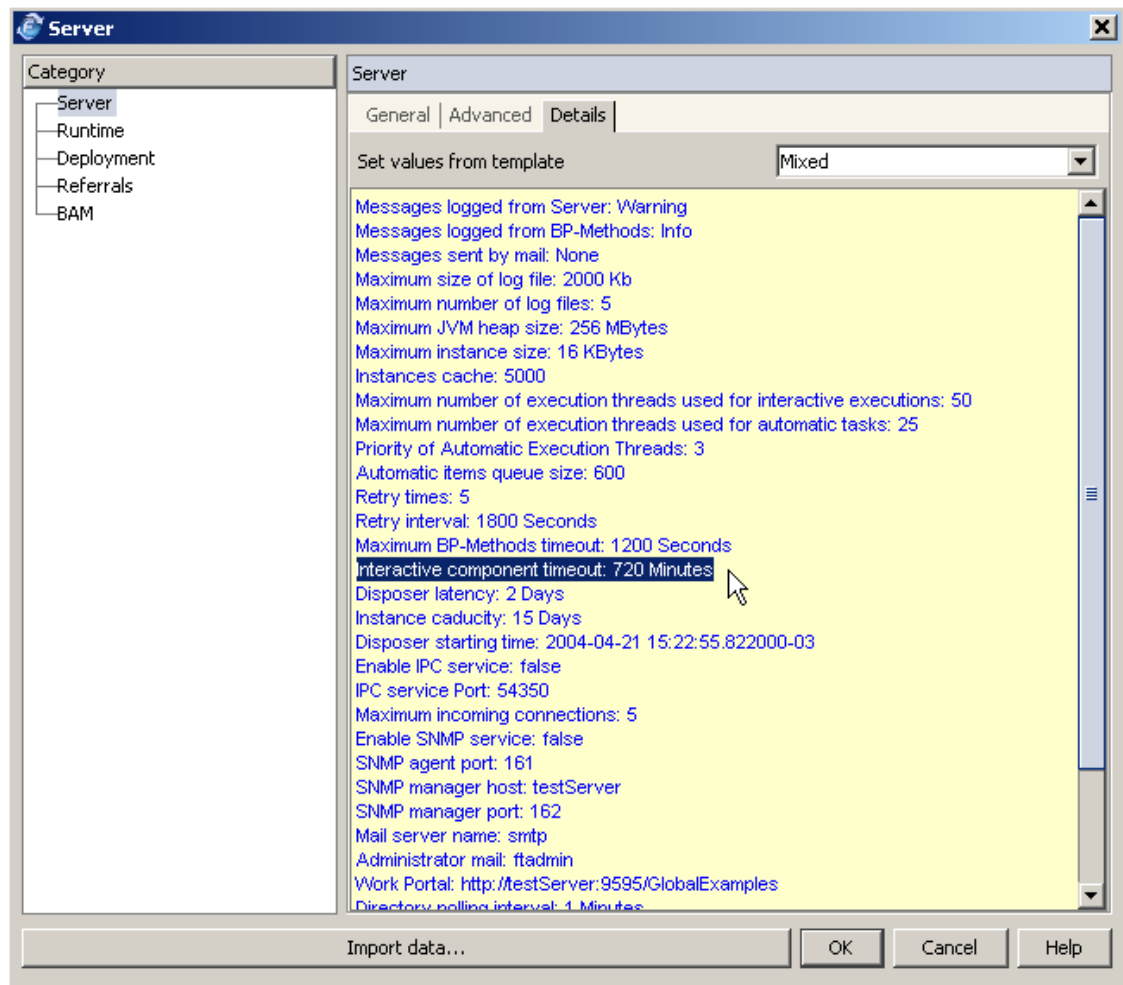
Maximum BP-Methods timeout

The **Maximum BP-Methods timeout** can be re-defined from the **FuegoBPM Studio Run** menu. Select **Server preferences** menu item. The Server properties window appears. Choose **Server** category from the list of categories displayed on the left panel. Within the **General** tab, set this property in the **Timeout properties** section



Interactive component timeout

The **Interactive component timeout** property can not be redefined. You can see its definition in the **Details tab**.



FuegoBPM Web Console Server Properties

Timeout settings are available from the Server configuration **Execution Tab / Timeout** section.

Basic Configuration	Log	Execution	Services	Networking	Others
-------------------------------------	---------------------	---------------------------	--------------------------	----------------------------	------------------------

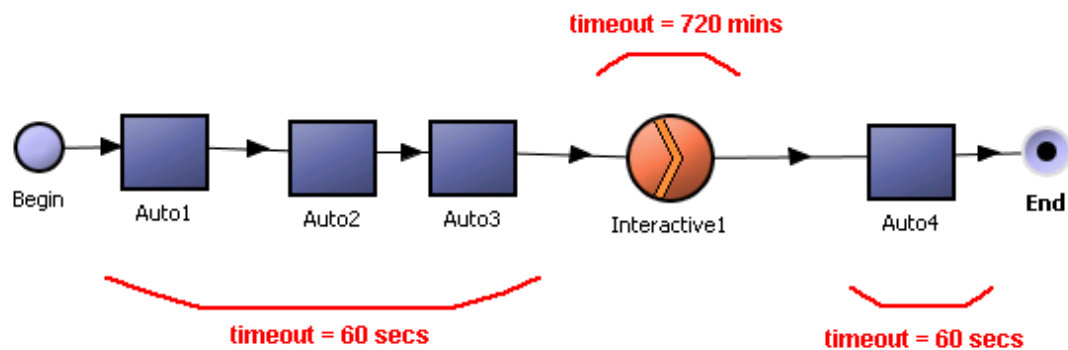
Startup	
Start automatically during Web Console initialization	<input type="checkbox"/>
Additional arguments used in startup	<input type="text"/>
Additional java arguments used in startup	<input type="text" value="-ea"/>
Memory	
Maximum JVM Heap Size	<input type="text" value="256"/> MB
Maximum Instance Size	<input type="text" value="16"/> KB
Instances Cache	<input type="text" value="5000"/>
Execution Threads	
Maximum number of execution threads used for interactive executions	<input type="text" value="50"/>
Maximum number of execution threads used for automatic tasks	<input type="text" value="5"/>
Priority of Automatic Execution Threads	<input type="text" value="5"/>
Automatic Items Queue Size	<input type="text" value="1000"/>
Retry Times	<input type="text" value="5"/>
Retry Interval	<input type="text" value="1800"/> Seconds
Request Queue Size	<input type="text" value="300"/>
Request Queue Timeout	<input type="text" value="5"/> Minutes
Timeouts	
Maximum BP-Methods Timeout	<input type="text" value="1800"/> Seconds
Interactive Component Timeout	<input type="text" value="720"/> Minutes
Maximum Process Web Service Session Timeout	<input type="text" value="300"/> Seconds
Debugger	
Trace Components	<input type="checkbox"/>
<input type="button" value="Save"/> <input type="button" value="Cancel"/> <input type="button" value="Reset"/>	
Changes to these field values will be ignored while the Server is running. They will take effect only after restarting the Server.	

Screenflow and timeout behavior

If you deploy in an FuegoBPM Enterprise edition, a process that has a screenflow, you have to consider that some timeout settings are

available in the FuegoBPM WebConsole. You can set the **BP-Method timeout** and the **Interactive component timeout**.

In a **screenflow** you can combine different types of activities and timeouts apply for each activity or group of them. In the following example the **BP-Method timeout** is set to 60 seconds and the **Interactive component timeout** is set to 720 minutes.



Once the screenflow begins its execution, for the first 3 automatic activities (**Auto1**, **Auto2** and **Auto3**), the total timeout will be **60** seconds. Once an interactive activity is reached, the **Maximum BP-Methods timeout** is reset.

Then the Interactive component call activity, **Interactive1**, begins its execution. The user has **720** minutes maximum to work with the presentation.

To complete the screenflow, the **Auto4** activity executes and has **60** new seconds to complete its execution.

Procedures

Procedures are applicable to graphically define component methods that do not have interaction with end users by using familiar graphical syntax from the FuegoBPM Studio.

A **Procedure** is a set of activities that is executed by a single participant. It also has a reduced set of activities that can be used.

No roles are allowed because a procedure is limited to automatic activities.

Scope

Procedures are designed to be used in any part of a process. A procedure cannot be used outside the project that it belongs to, but it can be reused among Processes in the same Project.

When do you have to use Procedures?

Procedures should be used in order to reuse part of a process. They are the right way to share process behavior between more than one process or inside the same process (calling the same procedure several times) instead of using IPC.

A procedure is a set of automatic activities that does not define a Process, but it defines some automatic instance behavior. For example,

1. Having a set of automatic activities that perform several notifications to different third-party applications and this behavior is repeated by most of the processes within the project.
2. A set of automatic activities that should be used several times within a Process. Therefore, adding several Procedure calls to the same Procedure would clarify the design.
3. If your BP-Method has a large number of lines, you should consider the possibility of moving the BP-Method to be implemented as a **procedure** and break down the method into several activities in a graphical design.
4. To graphically show a sequence of components methods to improve its comprehension.
5. To hide certain details of the implementation and push some logic to another level of the design.

What distinguishes procedures from a sub-process?

See Procedures versus Sub-processes.

Process instance and procedure instance

Once the process instance reaches an activity that executes a procedure, it remains there until the procedure finishes. The process instance is never *in* the procedure. The process instance actually remains in the process therefore, whatever you "apply" in the procedure does not affect the process instance.

The procedure instance is a separate instance and the operations performed over the instance within the procedure will not apply to the process instance. For example, neither adding notes nor attachments will apply to the process instance. If you need to use any of the process instance variables data, they need to be passed as arguments.

Characteristics

As it has an automatic behavior, a procedure does not have roles and it only includes automatic activities. That is, activities of the following types: Begin, End, Process Creation with no Termination Wait activity, Process Notification, Automatic, and Split-Join). It also includes Groups, Compensate Transitions and Exception Transitions.

Automatic Activities within the procedure can have:

- Fuego Methods: Fuego Business Language (FBL)
- Component call (Runs on server components only.)
- Procedure call.

Creating a Procedure

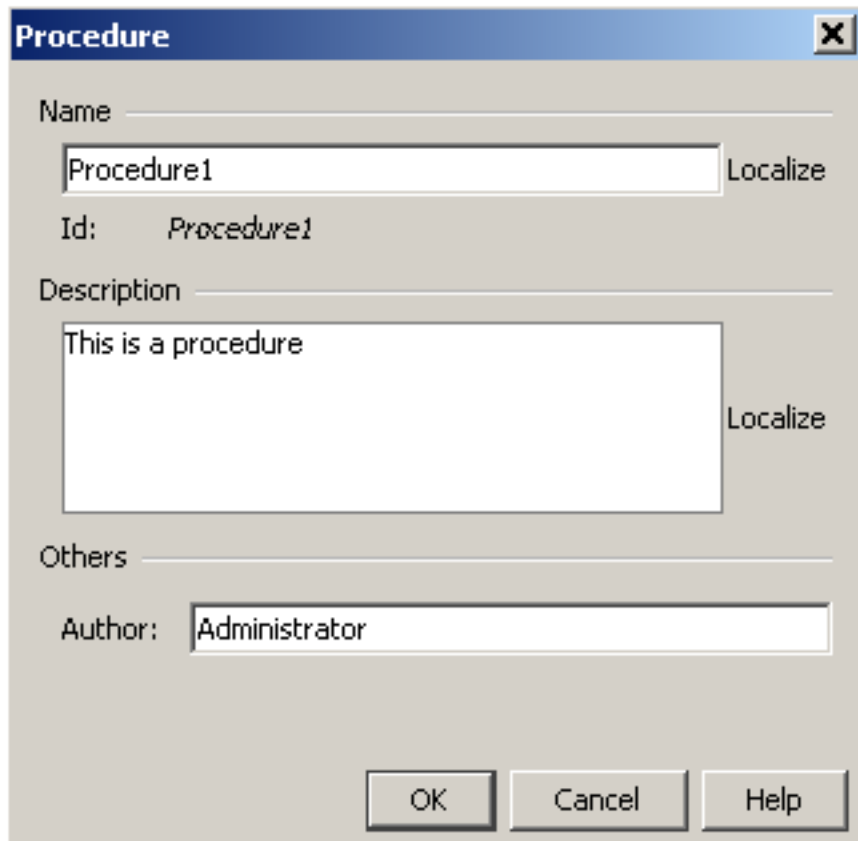
In order to create a Procedure, you can design it or import it from a file.

To **import a Procedure**:

1. Select the **Import/Processes** options from the **File** menu, or
2. Right-click on the Project name or on any folder and select **Import Processes**.
3. Finally, choose the corresponding file ending with *.xidl*.

To **create a Procedure**:

1. Select from **File** menu the **New** option and select **New procedure**, or
2. Right-click on the Project in the tree node and select **New procedure**, or
3. From the task you are implementing as a Procedure, select the **New** option. In this case, you are guided to create the procedure through a wizard. This option facilitates all instance variable creation, argument creation and argument mapping. For further information, see *Generate Processes, Procedures and Screenflows* automatically.



Procedure

Name Localize

Id: *Procedure1*

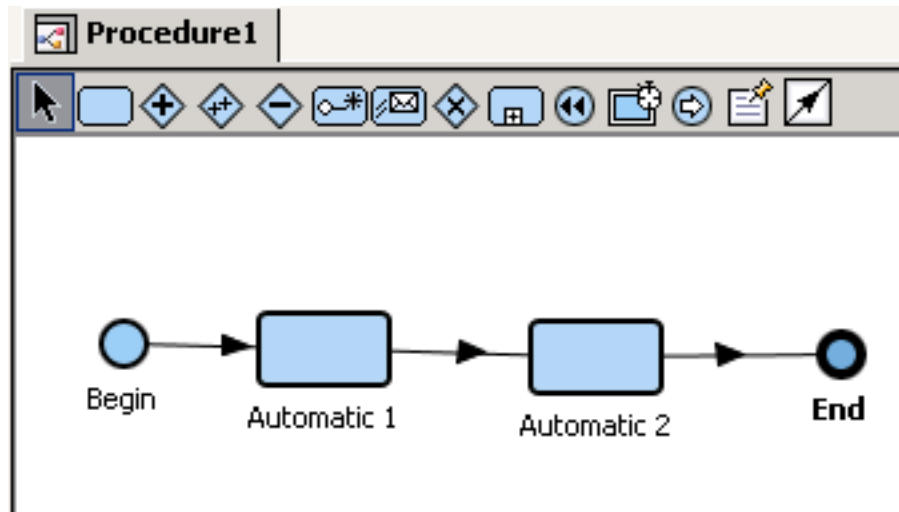
Description Localize

Others

Author:

OK Cancel Help

1. Enter a procedure **Name**. This field does not accept blank spaces. Anything that appears in this field will be displayed on the FuegoBPM Studio workspace and in Work Portal. This field accepts blank spaces and it is not limited.
2. Click **Localize** to include the procedure name in other languages. See Localization for further information.
3. Enter a procedure **Description**.
4. Enter the procedure **Author**.
5. Click **OK** and the new procedure appears displaying the default **Begin** and **End** activities.
6. Add all required activities within the procedure.



Note

 No **Process exception Flow** is available in a Procedure.

After creating the Procedure, it appears in the Project tree and is available for any process within the project.

To open the procedure, double-click on it or right-click and select the **Open** option.

Variables

It has input and output arguments and instance variables.

Using Procedures

Main task - Activity Preprocess request

Preprocess request

Implementation type
Procedure

Related Process
Procedure name P185 Edit New

Dynamic Process invocation
If the process call is dynamic the effective target process will be defined in runtime. The related process must be an Interface.
☐ Dynamic Process invocation

Choose the argument set to use in the selected procedure
Argument set name
BeginIn Argument mapping

OK Cancel Help

The Procedures are used in a similar way as a process invocation from a Subflow activity. You need to define the Begin Argument Mapping and the End Argument Mapping.

See Argument Mapping for further information.

A procedure can be invoked from an automatic activity's tasks. Define the task as a procedure type and apply from there.

See Tasks for further information.

Dynamic Process Invocation: the target procedure can be dynamic. In this case you do not define a Procedure as a target but a Procedure Interface instead. The real procedure to be called must match the procedure interface and its name is passed as the argument **targetProcessName** in the Argument Mapping Set. See Dynamic Process Call for detailed information.

Procedures do not generate events.

Searching for a procedure

See Searching for a process, procedure or screenflow.

Deployment

The procedure is deployed following the *deploy logic of a component*. It is not listed between processes at deployment time and it is stored in the repository at the same level as the catalog. Procedure code is directly embedded into the Process at publish time.

The server loads all associated Procedures as new Processes. This means that the server will work as any Subflow activity when working with a Procedure.

Any Procedure will be seen by the Server as a real Process with only one role.

Changes to a Procedure are not dynamically included in Processes that use the Procedure. As a consequence, the project needs to be redeployed to pick up the changes to the Procedure.

Rollback Techniques

Procedures are defined as **atomic**. The rollback is automatically done by the FuegoBPM Server. Exceptions cannot be treated inside a Procedure, they are thrown to the immediate outer group (or parent group).

FuegoBPM and the Standards

In the FuegoBPM Studio environment the right place for BPEL processes is as procedures.

FuegoBPM is a process SERVER: it invokes *services* from *people, systems and organizations*. One of the **services** it can invoke is an **automated**

procedure built in BPEL or in FuegoBPM. It can invoke discrete components as well as services composed of several web services (BPEL) and services composed of a sequence of calls to ANY component in ANY technology (FuegoBPM procedures) and screenflows.

Procedures Examples

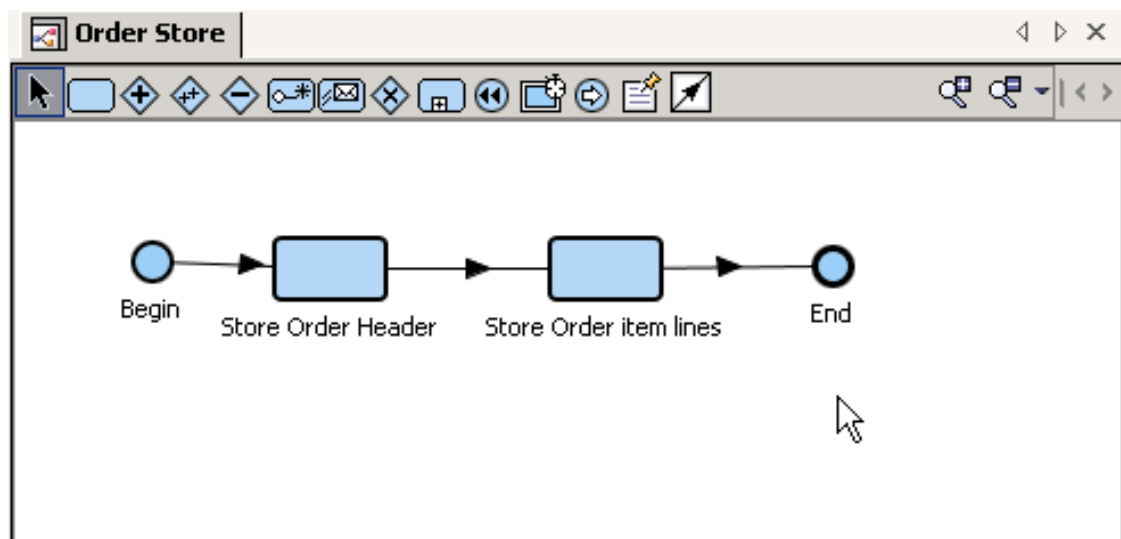
See Procedures Examples for examples of procedures.

Procedure examples

Using Procedures

Loading or storage of a Purchase Order

The Order is composed of a header and line items. The first procedure activity can be the storage of the header and the second one can be the storage of each individual line item. You can implement the loading of the order in the same way—a procedure activity will load the header and another activity will load the order line items.



Screenflows

Screenflows are similar to automatic procedures but they are intended to model complex human interactions. A typical use of screenflows is to design the interaction of different presentations of one or several Fuego Objects.

Screenflows are used to graphically define complex form interactions using the familiar graphic syntax from the FuegoBPM Studio.

Scope

Screenflows can be used from any interactive activity of any process within the project they belong to. They are used to implement interactive tasks. Therefore, they have no role because a single user executes the entire screenflow.

When do you have to use Screenflows?

You should use screenflows in most interactive tasks whenever possible. They have several advantages over Fuego Objects with hand written interactions:

- Virtually no code is necessary to connect the different presentations.
- The flow between the screens is explicit and graphically shown.
- They require less resources at run-time because they use **relay to** automatically, which results in improved scalability.

What distinguishes Screenflows from a sub-process?

The most important differences are the following:

- Screenflows are executed by a single participant to complete a process task.
- They are designed to be used as a single task within a process.

For further information, see Screenflows versus Sub-processes.

Characteristics

Screenflows should be thought of as components with a process-like interface.

They only include the following activities:

- Begin
- End
- Interactive Component Call
- Automatic

These activities have similar behavior as those in the process, but they are much simpler. In addition, screenflows support three types of transitions:

- Unconditional
- Conditional
- Exception
- Due

Screenflows automatically manage the allocation of server

resources, thus, freeing the developer of the need to deal with *relay to* expressions. Thus, it is very convenient to use them in order to define any interactive task. You should see interactive activities as a point in the process where instances get into an inbox for something to be done. The execution of this activity task will involve the invocation of interactive components. This sequence of interactive component invocations should be mapped to the screenflow's activities. Later on, this screenflow should be mapped as the implementation of the interactive activity.

Creating a Screenflow

To create a Screenflow, you can design it or import it from a file.

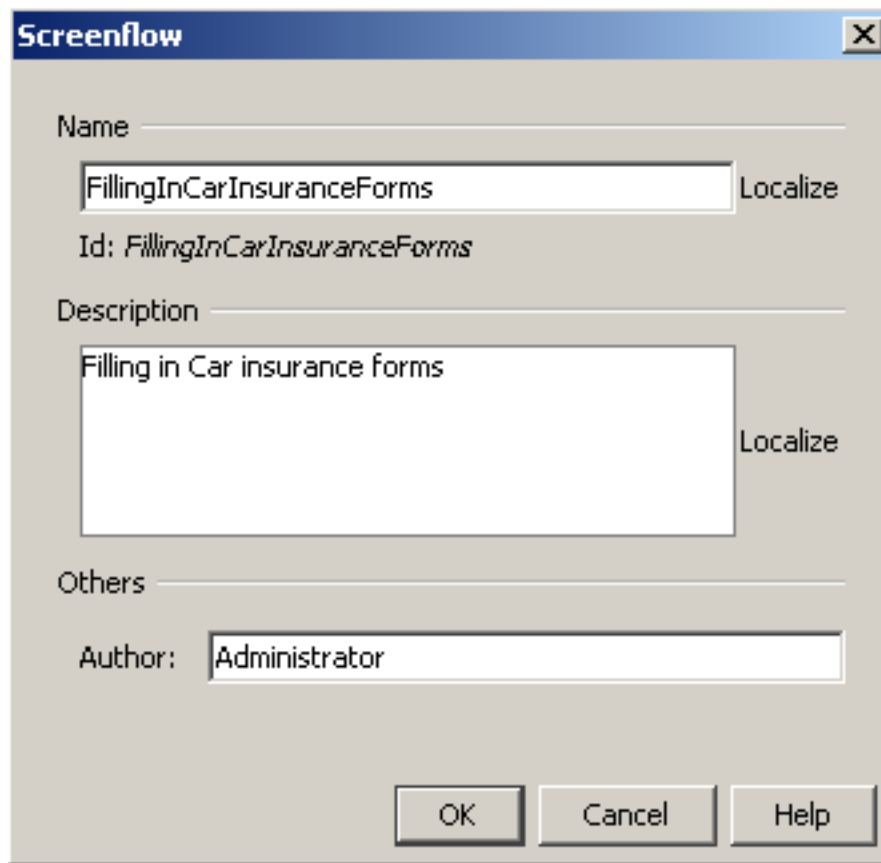
To import a Screenflow:

1. Select from the **File** menu the **Import/Processes** options, or
2. Right-click on the Project name or any folder and select **Import Processes**.
3. Finally, choose the corresponding file ending with *.xsd*.

To create a Screenflow:

1. Select from **File** menu the **New** option and select **Screenflow**, or
2. Right-click on the Project in the tree node and select **New screenflow**, or
3. From the task you are implementing as a Screenflow, select the **New** option. In this case, you are guided to create the screenflow through a wizard. This option facilitates all instance variables creation, argument creation and argument mapping. For further information, see Generate Processes, Procedures and

Screenflows automatically.

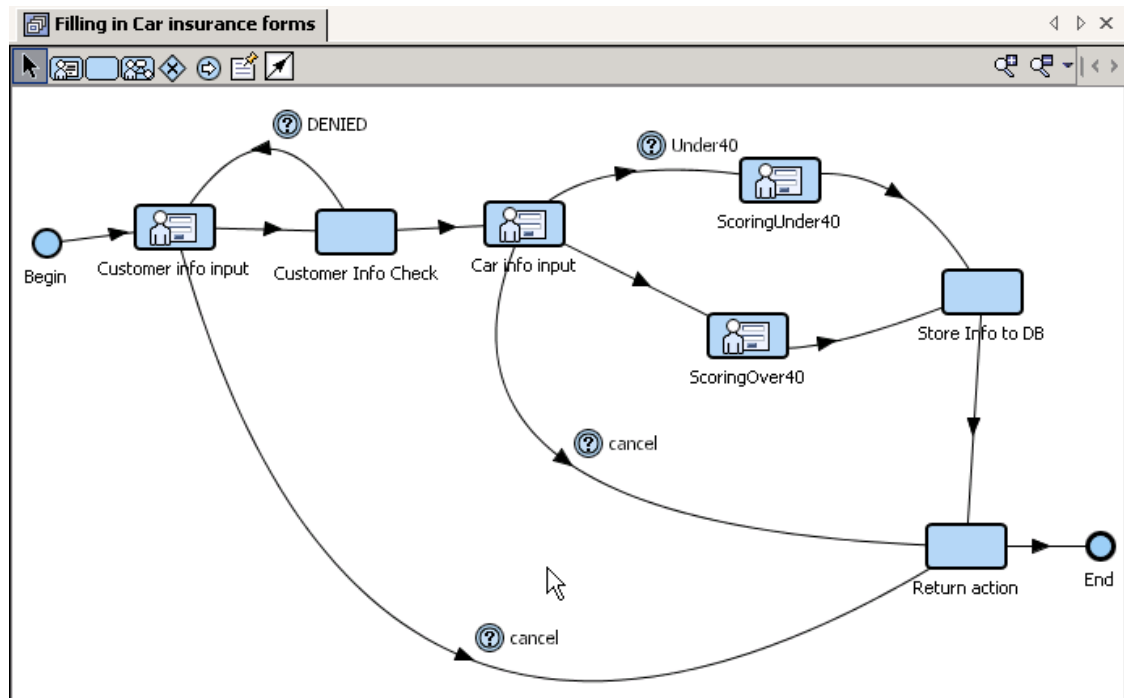


The screenshot shows a 'Screenflow' dialog box with the following fields and buttons:

- Name:** A text field containing 'FillingInCarInsuranceForms' and a 'Localize' button.
- Id:** A label 'Id: FillingInCarInsuranceForms'.
- Description:** A text area containing 'Filling in Car insurance forms' and a 'Localize' button.
- Others:** An 'Author:' label and a text field containing 'Administrator'.
- Buttons:** 'OK', 'Cancel', and 'Help' buttons at the bottom.

1. Enter a screenflow **Name** . This field accepts blank spaces and it is not limited.
2. Click **Localize** to include the screenflow name in other languages. See Localization for further information.
3. Enter a screenflow **Description**.
4. Enter the process **Author**.
5. Click **OK** and the new screenflow appears with the default **Begin** and **End** activities displayed.

6. Add all the required activities within the screenflow.





Activities within a Screenflow

The screenflow can manage the following activities

- Automatic activity
- Call Screenflow activity
- Interactive component call activity

Automatic activity

- Create an Automatic component call activity  /  . Next,

complete the main task by selecting the **Main Task** option from its short cut menu (right-click).

This activity **Implementation types** can use a:

- Method, or
- Component Method that runs on a server.

Note that Automatic activities always run on the FuegoBPM Server.

METHOD

Automatic1

Implementation type

Method

Method

Method Name :

automatic1



Edit New

OK Cancel Help

- Select a Method or create one. This Method will be executed when the task is performed. See Business Method for further information.

Call Screenflow activity

You can call a designed screenflow in the project within the current screenflow. The called screenflow works synchronically, this means that the instance that reaches the Call Screenflow activity will remain there until the called screenflow flow is completed.

- Create a Call Screenflow activity  /  .
- Complete the properties option by selecting as the *Related Process*, the **Screenflow to call**. If the called screenflow expects any arguments you need to map them (all arguments must be mapped). As well select the **Argument set Name**.

Activity: Call Screenflow

Category

- Activity Id
- Related Process
- Image

Related Process

Related Process

Screenflow name: Screenflow2

Argument mapping Edit New

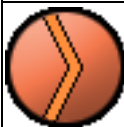



Argument set name



Select the mapping set name to use from the related process

BeginIn

OK Cancel Help

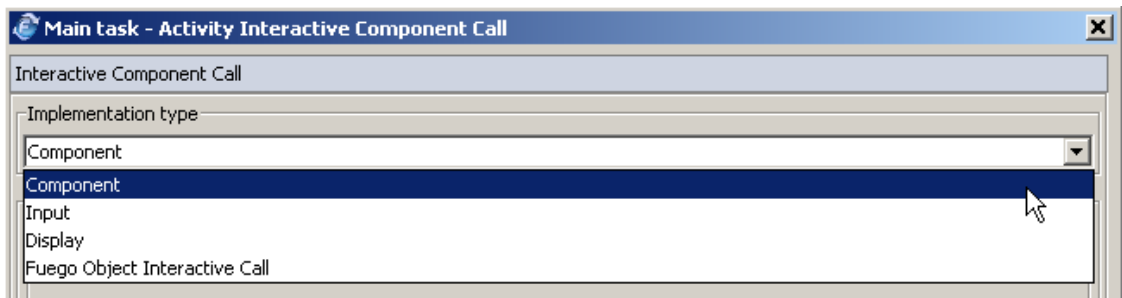
Interactive component call activity

Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			

- Create an Interactive component call activity  /  and then add a task within it by selecting the **Main Task** option from its

short cut menu (right-click).

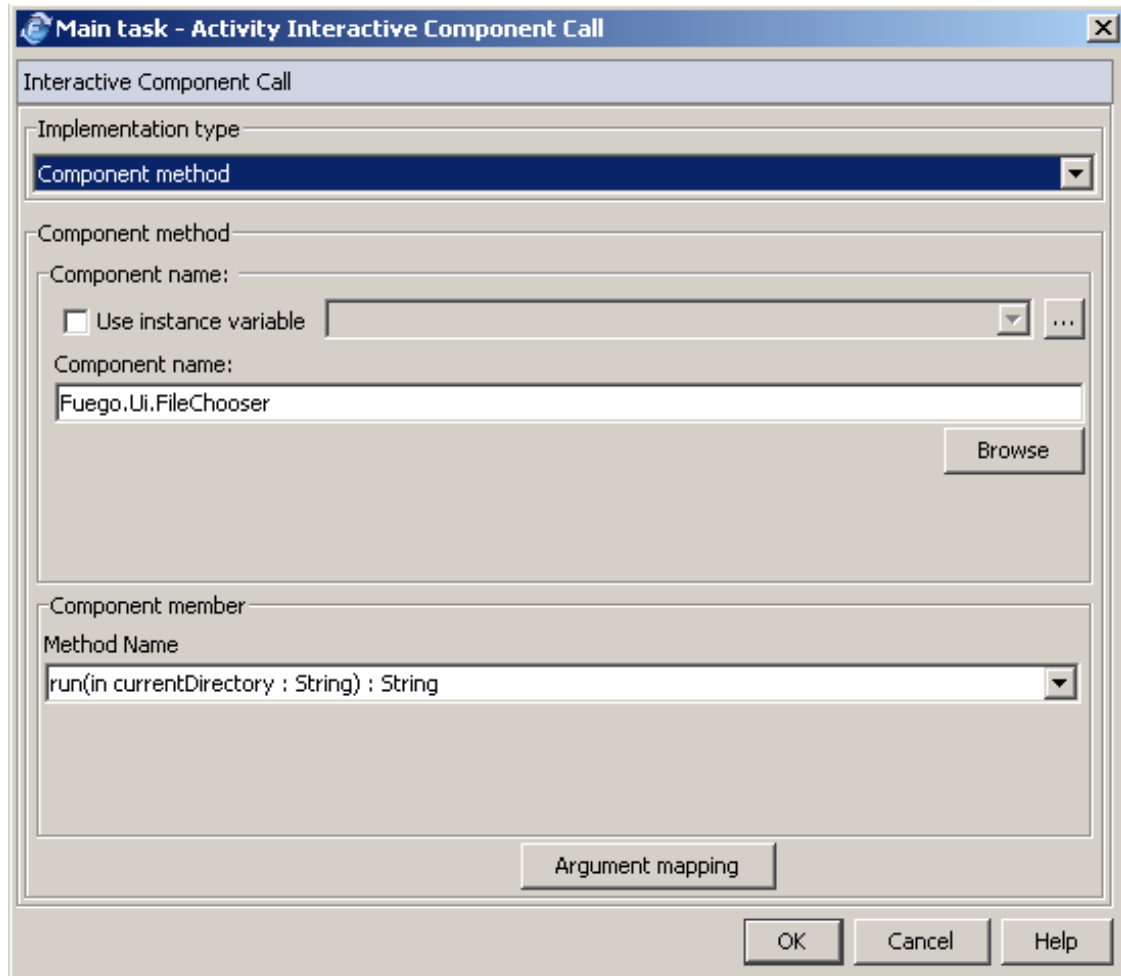
- The Task properties dialog box allows you to define the component to be executed.



This activity runs a **Component Method**, an **Input** or a **Display**.

COMPONENT METHOD

The task runs the defined Component method (See Component.)



Instance variable: Indicates that the invocation of the method will be applied to the selected variable. By selecting a variable, the Component name is implied and taken from the variable type.

Component Name: Represents the type of component to be invoked. Complete the component name or browse to choose the component from the Project Catalog (you can only select those defined to run on the client-side). If an instance variable has been selected, this value will be filled out from the variable type and cannot be edited from here.

Method Name: Indicates the Component method to run.

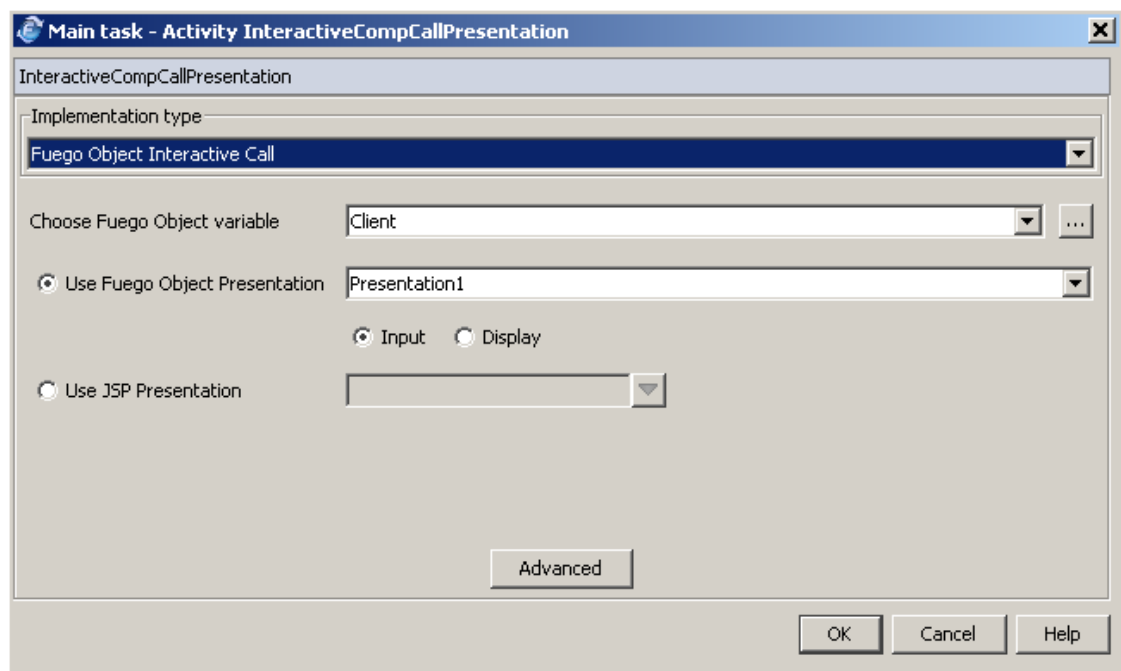
Argument Mapping: If the method requires any in or out arguments, these arguments have to be mapped to instance

variables or predefined variables that contain the value/s to be passed. Moreover, if you are using an instance variable (for example, a Fuego Object instance var) and the component updates any of the attributes within it, you need to explicitly map the Fuego Object instance variable to an argument. See [Argument Mapping](#) for further information.

The Component method must **run on client**.

FUEGO OBJECT INTERACTIVE CALL

The task runs the Fuego Object associated to the defined variable.




Choose the Fuego Object variable: Indicate the instance variable that defines the Fuego Object to use.

Use Fuego Object Presentation, or Use JSP Presentation: select any of the Fuego Object's presentations, or select a JSP one.

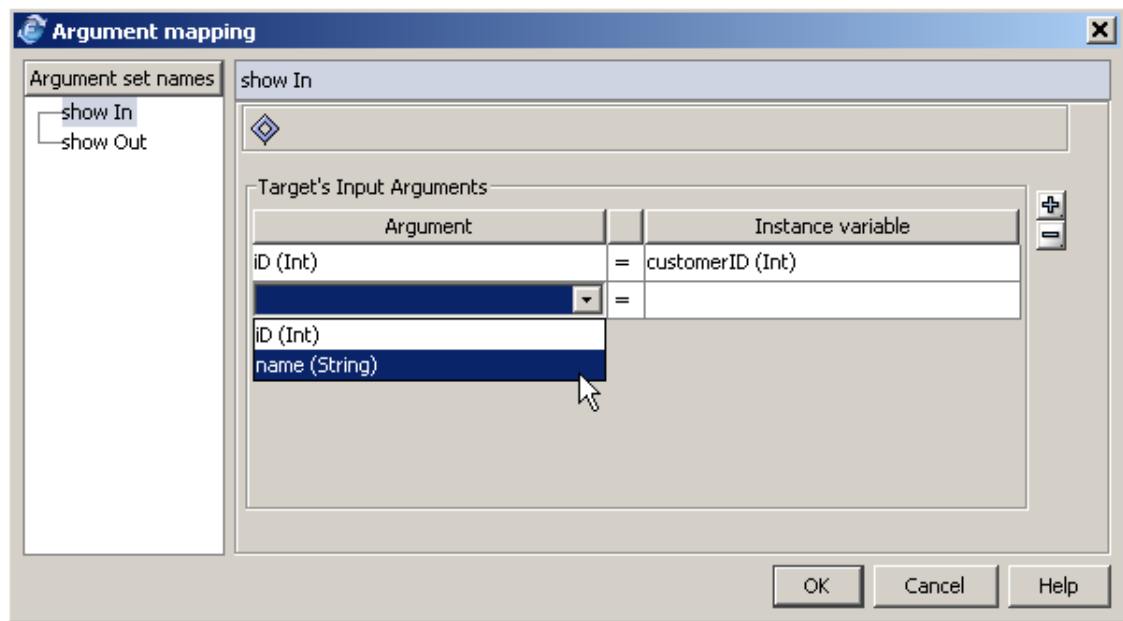
Input /Display: If you select to use a Fuego Object presentation, then select the way the presentation will act. In order to input information or just to display it.

Note

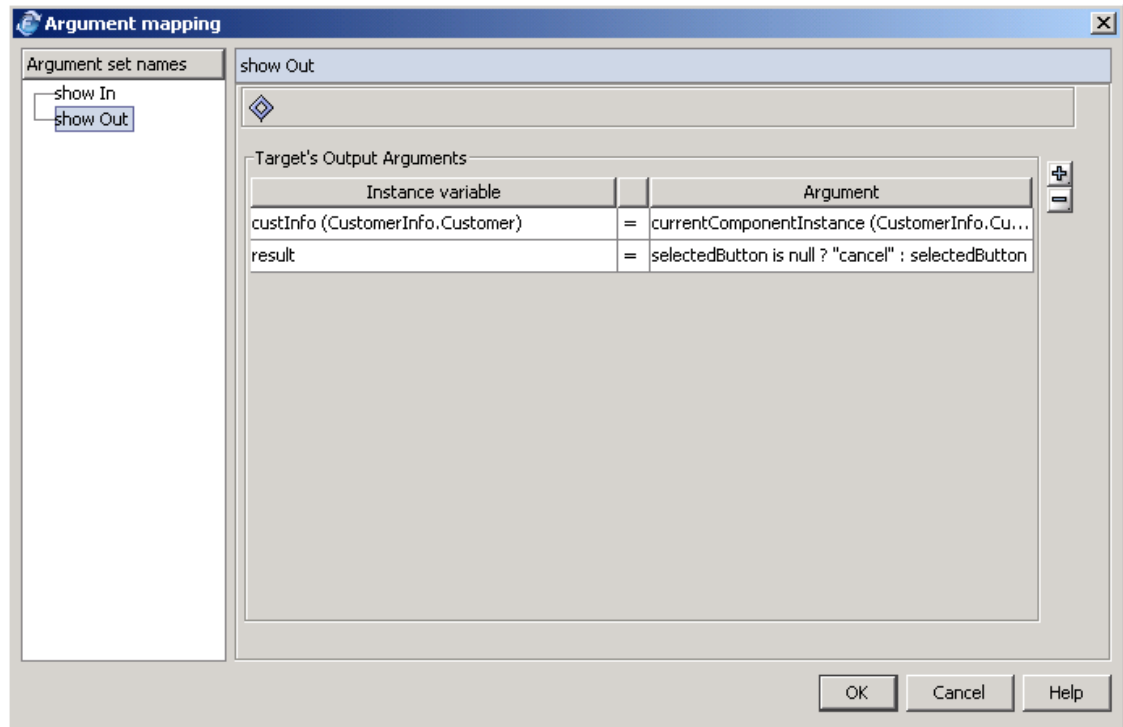
 If you use a JSP, you have to previously import it from the **webRoot** folder in the left panel.

Advanced / Argument Mapping: you can define in / out arguments to and from the Fuego Object you are implementing.

The input arguments are the defined attributes in the Fuego Object.



The output arguments can be the current Fuego Object instance and the selected button as the Fuego Object was executed.



INPUT

This implementation displays an input window to catch information. It has the same capabilities as the input statement.

The input fields are instance variables.

InterCompCallInput

Implementation type: Input

Input dialog title: Customer Info Input

	Label	Instance variable	Type
<input checked="" type="checkbox"/>	Name	name (String)	Text
<input checked="" type="checkbox"/>	Address	address (String)	Text
<input type="checkbox"/>	Age	age (Int)	Number
<input type="checkbox"/>	Gender	gender (String)	Choice

Buttons: OK, Cancel, Ignore

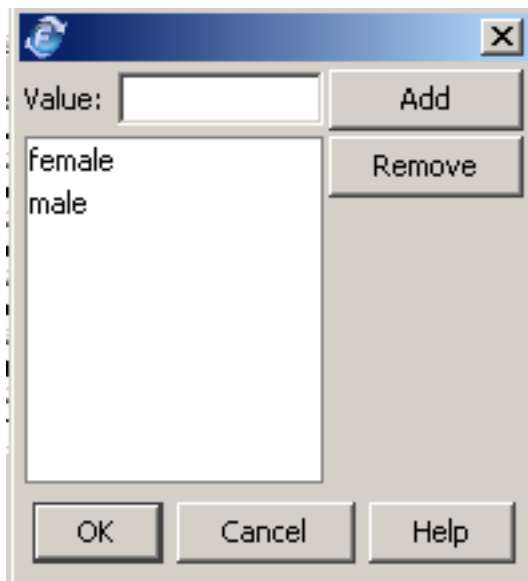
Assign selected button to: result (String)

Cancel button is: Ignore

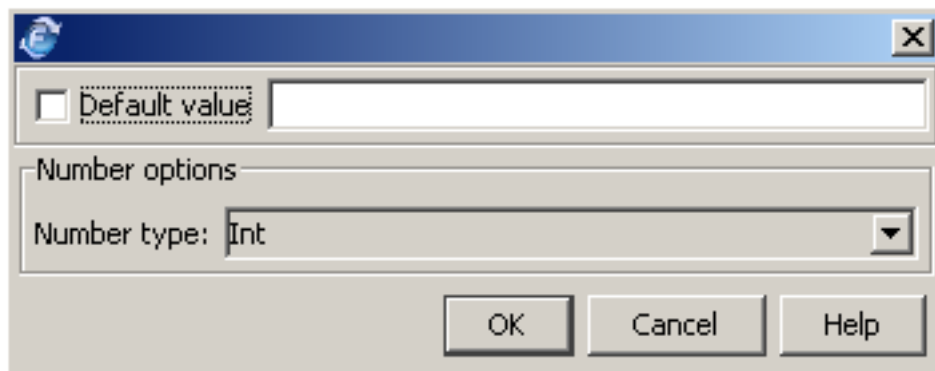
OK, Cancel, Help

Based on the Instance variable type, you can select the input form type. You can add additional information about the type by selecting any of the following: ...

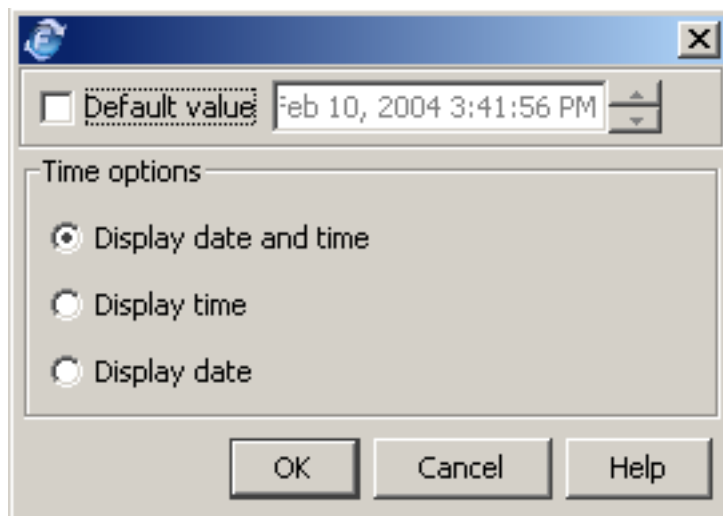
- **String:** can be **Text**, **Long Text**, **Password** (when you input information into this type of field, each character is replaced by an "*") and **Choice**. In the **Choice** option, you need to define the valid values:



- **Number:** you can add a default value.

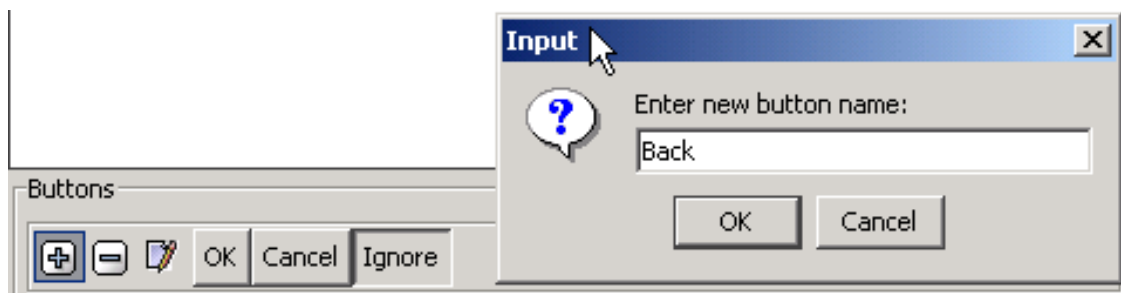


- **Boolean:** you can set the default value to **true** or **false**.
- **Time:** you can define the date/time format:

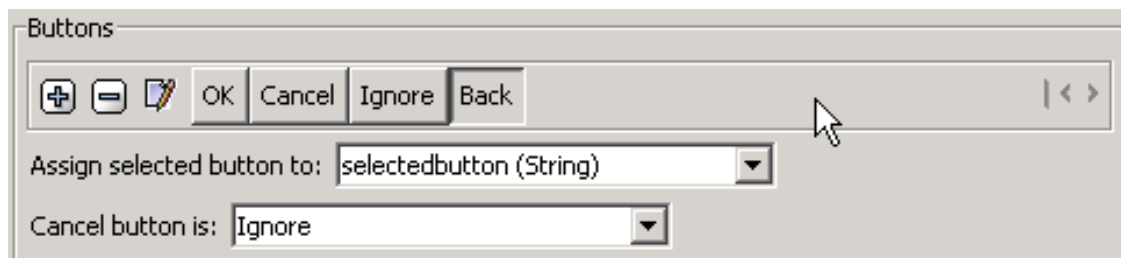


Input form Buttons

You can add buttons to the input form as shown below:



And an additional button to the **OK**, **Cancel** and **Ignore** default buttons displays.

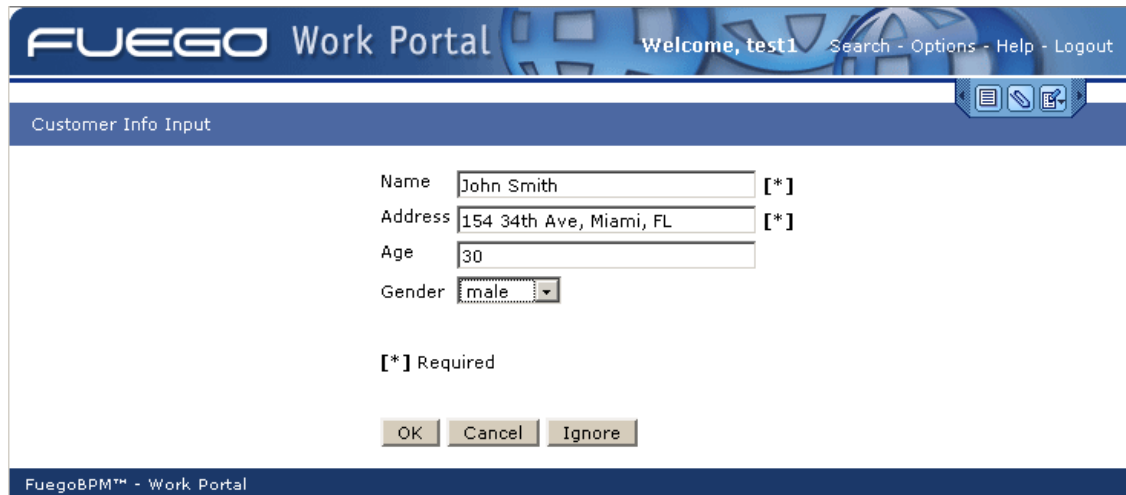


Furthermore, you can capture the *selected button* in an instance variable to use later in the process.

Finally, you can define any of the buttons, that when selected, all

changes are ignored and avoids the input of required fields

The Input form in the Work Portal appears as follows:



The screenshot shows the 'FUEGO Work Portal' interface. The header bar includes the logo, the text 'Work Portal', and a welcome message 'Welcome, test1'. Navigation links for 'Search', 'Options', 'Help', and 'Logout' are present. A toolbar with icons for document, edit, and save is also visible. The main content area is titled 'Customer Info Input' and contains a form with the following fields: 'Name' (text input with value 'John Smith' and a required field asterisk), 'Address' (text input with value '154 34th Ave, Miami, FL' and a required field asterisk), 'Age' (text input with value '30'), and 'Gender' (dropdown menu with 'male' selected). A legend indicates that the asterisk denotes a required field. At the bottom of the form are three buttons: 'OK', 'Cancel', and 'Ignore'. The footer of the page reads 'FuegoBPM™ - Work Portal'.

DISPLAY

This implementation displays information. An **expression** needs to be implemented. It can contain instance variable information as simple text. It has the same capabilities as the display statement.

Main task - Activity IntCompCallDisplay

IntCompCallDisplay

Implementation type
Display

Display dialog title: "Customer Information"

Value "Name: " + name + "\nAddress: " + address + "\nAge: " + age

Buttons

Assign selected button to: selectedbutton (String) Instance Variables

Default button: OK

☐ Use Fuego Object Presentation

OK Cancel Help

Input form Buttons

See **Input** explained above.

The Display form in the Work Portal appears as follows:

FUEGO Work Portal Welcome, test1 Search - Options - Help - Logout

Customer Information

Name: John Smith
Address: 154 34th Ave, Miami, FL
Age: 30

OK

FuegoBPM™ - Work Portal

If you select to **Use Fuego Object Presentation**, the form appears in a different display format more likely to a Fuego Object Presentation rather than the standard format for the display statement.

Using Screenflows

A screenflow can be invoked from an activity's tasks. Define the task as a screenflow type and apply from there. See Tasks for further information.

Returning from a Screenflow

When you return from a screenflow to the calling activity, the last **action** within the screenflow is passed. This **action** becomes the **calling activity's action**.

Searching for a screenflow

See Searching for a process, procedure or screenflow.

Deployment

Screenflows follow the same rules as Fuego Objects concerning publication and deployment.

Rollback Techniques

You must think about how to manage rollback actions, if necessary. It is recommended that you design the screenflow where permanent changes to external resources are made only in the last activity of the screenflow (usually an automatic activity). This way, your screenflow, when called from Work Portal, will properly support the browser's 'Back' button.

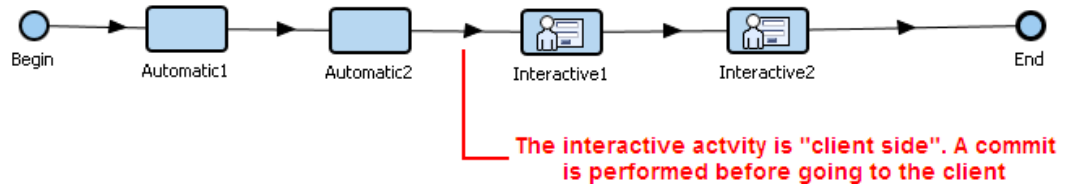
Screenflow and timeout behavior in Enterprise

See Execution timeout for detailed information.

Screenflow and runtime

Transactions handling

The transaction is committed just before an interactive activity, and a new one is created just after an interactive activity. This works this way to avoid keeping resources locked in the engine while the user is doing interactive tasks.



Cancelling an Interactive activity

Screenflows are a series of interactive activities which handle the user interaction with Fuego. Each interactive activity can be a Fuego Object presentation, display statement, input statement or JSP page. Even though each of these elements provide a cancel option (or button), that option does not cancel the complete screenflow, just that interactive activity. Cancelling the last presentation in a screenflow will not automatically revert the changes submitted in previous presentations belonging to the same screenflow.

For example, if a screenflow which edits a Fuego Object, has two presentations, and the user submits the first one, any changes performed in that presentation will be committed even though the user may then cancel the second presentation

Screenflows Examples

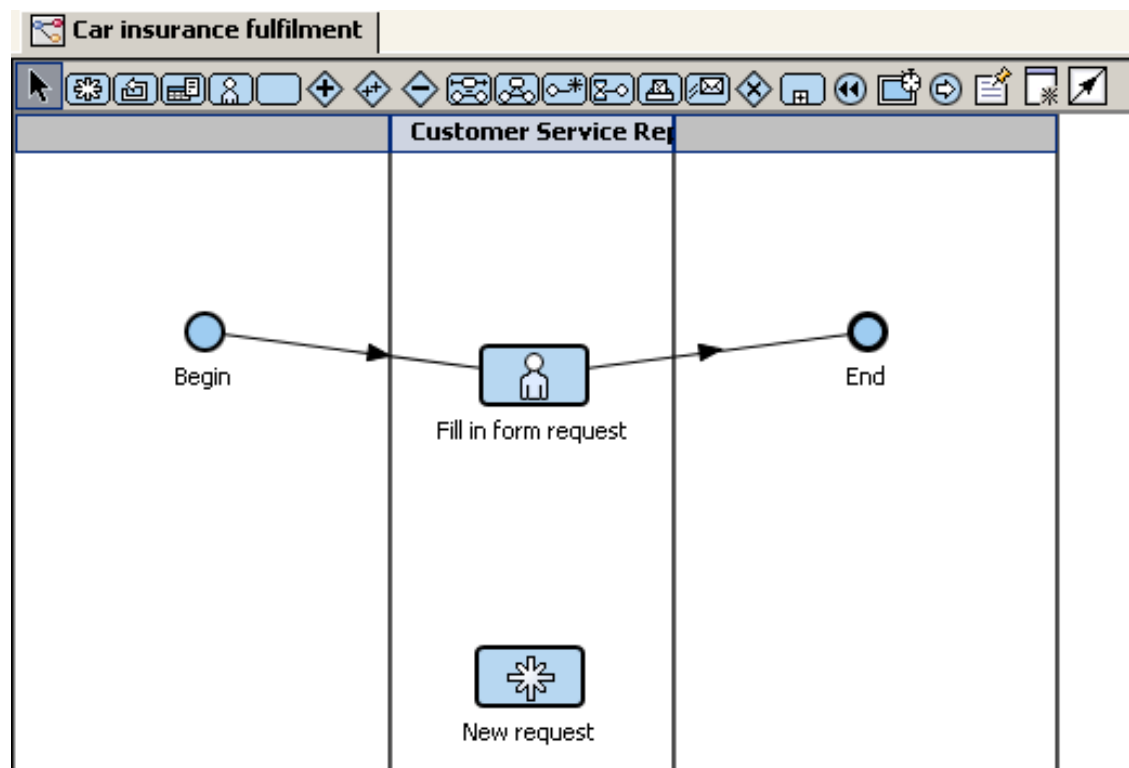
See Screenflows Examples for screenflow samples.

Screenflow examples

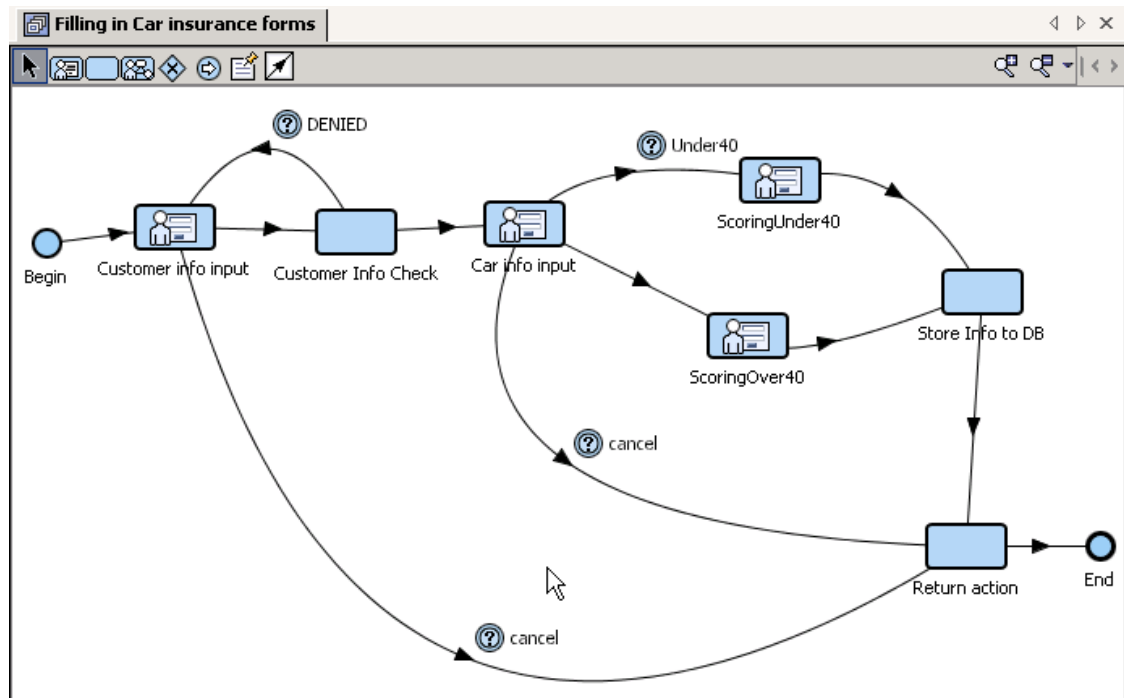
Using Screenflows

Filling in a Car insurance form

A customer care representative needs to complete a car insurance form for a customer. This can be mapped with a screenflow since it will need to invoke several forms.



The **Fill in form request** activity implements a screenflow:



On the first page of the form, personal information has to be entered (**Customer info input** activity.)

This form is implemented using a Fuego Object and one of its presentations.

The following form appears in Work Portal:

The screenshot shows the 'FUEGO Work Portal' interface. The header bar includes the logo, the text 'Welcome, fuegouser', and navigation links: 'Search - Options - Help - Logout'. A toolbar with three icons is on the right. The main content area features a form titled 'Customer Information'. The form contains the following fields and values:

Customer Information	
Custid	10
Name	James Robertson
Birthdate	1/3/1967
Age	38
<input type="button" value="Submit"/> <input type="button" value="Cancel"/>	

The footer bar reads 'FuegoBPM™ - Work Portal'.

Before the customer representative continues with the next form, the input information is checked (**Customer Info Check** activity) and it might need to be modified.

The second form contains other questions about the car. (**Car info input**):

The screenshot shows the 'FUEGO Work Portal' interface. The header bar includes the logo, the text 'Welcome, fuegouser', and navigation links: 'Search - Options - Help - Logout'. A toolbar with three icons is on the right. The main content area features a form titled 'CAR INFORMATION'. The form contains the following fields and values:

CAR INFORMATION	
Plate	BDC101
Model	1999
Color	White
Value	15000
<input type="button" value="submit"/> <input type="button" value="refresh"/> <input type="button" value="cancel"/>	

The footer bar reads 'FuegoBPM™ - Work Portal'.

Finally, other information is entered depending on the age of the applicant. The screenflow models two subsequent activities for different forms depending on the age (**ScoringUnder40** and **ScoringOver40**.)

Customer's input form if he/she is younger than 40:

The screenshot shows the FUEGO Work Portal interface. The header bar includes the FUEGO logo, 'Work Portal', a welcome message 'Welcome, fuegouser', and navigation links 'Search - Options - Help - Logout'. A toolbar with icons for document, edit, and print is visible. The main content area displays a 'Scoring' form for a customer with ID 10, named James Robertson, born 1/3/1967, aged 38. The form includes a checkbox for 'Do you keep your car in a Garage at night?' which is checked. At the bottom are 'Submit' and 'Cancel' buttons. A footer bar reads 'FuegoBPM™ - Work Portal'.

Scoring	
Customer ID	10
Name	James Robertson
Birthdate	1/3/1967
Age	38
Do you keep your car in a Garage at night?	<input checked="" type="checkbox"/>
<div>Submit Cancel</div>	

Customer's input form if he/she is older than 40, as it is probable that he/she has children that may drive the car:

The screenshot shows the FUEGO Work Portal interface. The header bar includes the FUEGO logo, 'Work Portal', a welcome message 'Welcome, fuegouser', and navigation links 'Search - Options - Help - Logout'. A toolbar with icons for document, edit, and print is visible. The main content area displays a 'Scoring' form for a customer with ID 25, named Susan Garret, born 11/4/1960, aged 44. The form includes two checkboxes: 'Do you keep your car in a Garage at night?' (checked) and 'Do you have children under 23 that use the car?' (checked). At the bottom are 'Submit' and 'Cancel' buttons. A footer bar reads 'FuegoBPM™ - Work Portal'.

Scoring	
Custid	25
Name	Susan Garret
Birthdate	11/4/1960
Age	44
Do you keep your car in a Garage at night?	<input checked="" type="checkbox"/>
Do you have children under 23 that use the car?	<input checked="" type="checkbox"/>
<div>Submit Cancel</div>	

This example manipulates four different forms to complete. The

screenflow manages all execution of the forms and the input logic sequence.

If any of these forms is cancelled, then it is assumed that the whole screenflow needs to be performed again.

Find the project **ScreenflowCase01.fpr** in FuegoBPM's installation directory, in the studio/samples directory to study the example further.

The FuegoBPM's participant name is: **fuegouser**.

External Tasks

External tasks are tasks that are completely implemented outside FuegoBPM. The External task implementation allows you to implement the interactive activity (more precisely the GUI presented to the user) outside FuegoBPM.

It is a framework to allow adding an external UI interface when executing an interactive task without using any of the FuegoBPM presentation capabilities (Fuego Object Presentations, Screenflows, etc).

For example, this is useful if you need to integrate the FuegoBPM Server with a .NET client.

These tasks are accessible from PAPI/PAPIWS by calling the methods:

```
prepareExternalActivity( ... )  
commitExternalActivity( ... )
```

For further information see the PAPI's JavaDoc documentation.

The framework works as follows:

1. A page displays all instances (you can get this using PAPI or PAPIWS).
2. You select the instance for execution (this instance is waiting in an interactive activity).
3. To start the execution, the **prepare** method should be executed. This method can retrieve some information from the instance as well as leveraging retrieval of other information using FuegoBPM's integration capabilities. Upon successful execution of the **prepare** method, the instance will remain locked for that participant to prevent other participant from starting the execution for that instance.
4. You can render your own UI (ASP.NET, etc).
5. When this is submitted, the **commit** method should be executed. This method receives information collected from the external UI and most likely synchronize this data with instance variables. You can also perform transactions against your backend systems using FuegoBPM's integration capabilities. Upon successful execution of this method, the lock over the instance will be released and the task will be marked as completed.

When you select the External implementation you have to specify two methods:

- The *prepareExternalActivity()* method can be used to get any values needed by the presentation before actually displaying it.

This method extracts and processes information from the instance variables or Fuego Objects, and makes it accessible through its output arguments.

- The *commitExternalActivity()* method should be invoked once the user finished with the GUI and processing can continue in FuegoBPM.

This method completes the task (and the instance if necessary), and maps its input arguments to instance variables.

Both methods can declare input/output arguments that will be passed along whenever they are invoked. As well both methods need to be invoked through PAPI or through PAPI-WebServices.

For example:

The instance arrives to the activity and waits there until someone sends the *prepareExternalActivity()* method to it. It executes the method and answers back a set of arguments (valid values lists or predefines/default values to be used by the GUI). After the user finishes with the GUI (which could be implemented anyway you want), the *commitExternalActivity()* method is invoked and the values entered by the user are passed along so that FuegoBPM can use them (set them into a Fuego Object for example). Then the instance moves forward in the process (unless you finish the *commitExternalActivity()* method with an Action different than OK or RELEASE).

- *Configuration*: you can optionally define an URL. When the task is executed from the Work Portal, the Work Portal redirects the execution to the URL.

Using Relay to

Relay to invocations are a special kind of invocation designed to optimize interactive task executions.

Note



Consider using Screenflows instead of "relay to" invocations, since

Screenflows have all the benefits of **relay to** invocations with the simplicity of process-like design.

When developing the BP-Method, you must consider the impact the BP-Method task will have on your process. For example, BP-Methods in Interactive activities usually require some kind of response from a human participant.

By nature, humans are unpredictable. A human participant may start their task in Work Portal. Then, something may happen and they forget to complete their task. They return to it later.

Uncompleted tasks lock execution threads in the FuegoBPM Server and other resources (such as database connections and open transactions). As less resources are available for the server, its scalability is reduced. It means that the server cannot accommodate as many users as it would under normal circumstances. To ensure that server resources are released while waiting for end user input, you can use *relay to* invocations in your methods.

When you use *relay to* in an Interactive BP-Method, the server immediately releases all resources allocated for that execution and does not wait for end user input. This frees the server to service another user. When the participant finally enters their input, the server routes the results of the interaction with the participant to the BP-Method designated in the *relay to* statement.

Syntax

Relay to can be used with all interactive statements and with client-side components, you have a relay to syntax for:

- **display**
- Client-side components
- Standard **input**

- Fuego Object **input**

But in general the syntax is of the form:

```
<invocation>
relay to <BP-Method name>
  [using
    <BP-Method argument> = <invocation output argument>
    ...]
```

with some special considerations for **input** statements. Let's go through each part of a **relay to** invocation:

- **invocation**: This is the interactive statement that we want to execute. It can be a **display**, an **input** or a method that belongs to a client-side component.
- **BP-Method name**: Name of the process' method that the server will execute after the component execution is finished.
- **BP-Method argument**: Argument of the process' method that will receive some of the invocation's output arguments.
- **invocation output argument**: Output argument of the invocation. This may also be a local variable.

Using display with relay to

The following example is a standard display statement used in conjunction with **relay to**:

```
display "Do you want to continue?"
  using buttons = [ "Ok", "Cancel" ]
relay to afterQuestion
```



```
using  
    choice = selection
```

The **afterQuestion** method must have an argument named **choice** of type **String**.

Notice that **display** statements are asynchronous if the Method does not use its output arguments (e.g., the **selection** argument). If that is the case, **relay to** optimization is not necessary.

```
//This statement will not block at run-time  
display "Done!" using buttons = [ "Ok" ]
```

Using client-side components with relay to

The following example is a typical use of **relay to** with a client-side component:

```
run FileChooser  
    using currentDirectory = directory  
relay to loadFile  
    using  
        fileName = selectedFile
```

If in an invocation to a client-side component the return value is not required, the call is executed asynchronously until the code attempts to read an attribute of the component.

For example, if you run the following code in an interactive task (not from the debugger):

```
//Run the file chooser
run FileChooser
    using currentDirectory = "."

//This will be displayed in the log before the user
// dismisses the FileChooser
logMessage "After RUN" using severity = SEVERE

//Execution will block here until the user dismisses the
//FileChooser dialog box
display "File chosen: " + FileChooser.selectedFile
```

You will see the file chooser dialog in Work Portal. Before selecting something, check the server's log. You will see that the message is already there. This happens because execution can take place concurrently while the user is entering data. The code will continue to run until a result from that particular component is needed (in our example, when we read the **selectedFile** attribute). When this happens, the method will have to wait for the first invocation to finish before continuing. You may use this fact to perform some task while the user is entering data.

If your code needs the return value of a component, for example when using the **Table** component to display data, then the use of **relay to** does not give you any advantage, because the component will be executed asynchronously.

Using standard input statements with relay to

The following is a typical example of a standard **input** with **relay to**:

```
creditCardNo = ""
acceptedTypes = [ "visa" : "Visa", "master" : "Mastercard",
                  "amex" : "American Express" ]
creditCardType = "visa"
firstName = ""
lastName = ""
expiration = 'now'
comments = ""
```

```
input "First Name:" firstName (required),
      "Last Name:" lastName (required),
      "Credit card type:" creditCardType (readonly)
                                in acceptedTypes,
      "Credit card No.:" creditCardNo,
      "Expiration Date:" expiration,
      "Additional Comments:" comments (textarea)
using
    title = "Enter payment info",
    buttons = [ "Ok", "Cancel" ]
relay to updateCreditRecord
using
    firstName = firstName,
    lastName = lastName,
    creditCardType = creditCardType,
    creditCardNo = creditCardNo,
    expiration = expiration,
    comments = comments,
    button = selection
```

Note that in the argument mapping section you can reference the fields used as input in the *relay to* statement.

If the field used in the input is an array reference the code is as follows:

```
array = [ "One", "Two" ]
input "Enter the value for the first item: " array[0]
relay to processResult
using
    result = array[0]
```

To reference the field you must use exactly the same expression used in the *input* statement. Otherwise, the compiler will interpret it as a variable and will not bind the result of the edition to the argument. However, if you want to pass along the original value, you should first assign it to a local variable. See the modified example for this case:

```
array = [ "One", "Two" ]  
original = array[0]  
input "Enter the value for the first item: " array[0]  
relay to processResult  
    using  
        result  = array[0],  
        reference = original
```

Using Fuego Object input statements with relay to

Fuego Object presentations are a special case. This is because Fuego Objects are not exactly client-side components. They are 'run-anywhere' components. This means that, when you use Fuego Objects in the server, the object is there. And when you display a presentation, the object travels to Work Portal for execution. Because of this, you cannot pass additional arguments to the target method; they are implied.

See the following example:

```
object = MyFuegoObject()  
input object  
    using selectedPresentation= "simple"  
    relay to createRecord
```

The method **createRecord** must have two arguments:

- instance: Must be of a type that is assignable from **MyFuegoObject**
- selectedButton: Must be of a type that is assignable from **String**.

What is important is the order of the arguments, not their names.

After the presentation is submitted or canceled, the *createRecord* method is called by the server, passing the instance of the Fuego Object after it has been modified by the presentation in the first argument, and the button that was selected as the second argument.

Procedure and Screenflows versus IPC

- **Procedures** and **Screenflows** have no roles. They inherit the participant from the activity that invokes them (just like a method from a component would).
- **Procedures** and **Screenflows** are private to the project they were created in.
- **Procedures** and **Screenflows** are not published and/or deployed and are not assigned to an Organizational Unit, as subprocesses are.
- **Procedures** and **Screenflows** can have different versions working with different versions of the process that invokes them. Unlike subprocesses, they work for the process that uses them. Procedures and Screenflows are hooked to the process where they are called from (with Subprocesses, the process always uses the last version of the child process).
- If the **Procedure** or **Screenflow** changes, newly published processes will incorporate the changes. However, a process that is already published (before the change to the procedure or screenflow) will continue to use the old version. In order to upgrade the process to use the new version of a procedure or screenflow, it should be republished.
- **Procedures** are atomic.
- **Screenflows** execute each activity in a separate transaction.

Chapter 6. Argument Mapping & Correlations

Argument Mapping

Argument mapping converts arguments into instance variable to keep information with an instance as it moves through a process. Argument mapping applies to Interprocess Communication (IPC) activities, Procedures and Screenflows. It also applies when calling Components.

IPC Activities

Some arguments are generally passed to and from IPC activities. The received arguments are converted into instance variables so that this information is kept within the instance. In addition, some instance variables are passed as arguments to the called process.

In order to achieve the passing of arguments, you must define the **argument mapping**.

Argument mapping varies from one activity to another. Mainly in IPC activities, there is one activity that **calls** and the other one that is **called**. The **called** activity defines the arguments and the **calling** activity passes these arguments.

The IPC activities are:

1. Begin
2. End
3. Process Creation
4. Termination Wait
5. Subflow

6. Process Notification

7. Notification Wait

In IPC activities, argument variables are mapped by matching the incoming/outgoing arguments with instance variables or predefined variables. In addition, expressions can be used to set variables.

IPC activities work in pairs, one as the calling/notifying activity and the other as the called/notified activity. The pairings are listed in the table below:

Called/Notified Activity (defines the interface)	Calling/Notifying Activity (completes the interface)
Begin	Subflow (create part)
	Process Creation
Notification Wait	Process Notification
End	Subflow (wait part)
	Termination Wait

To map arguments

Right-click on the activity and select Argument Mapping.

Begin

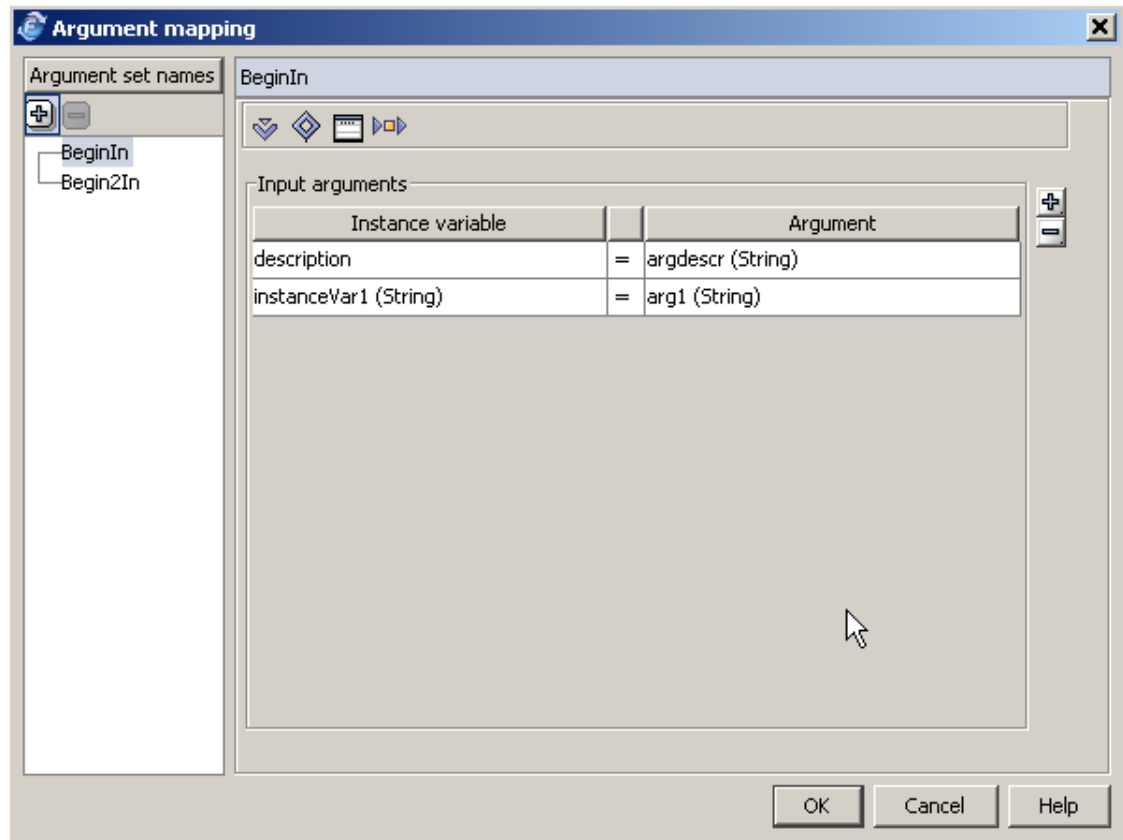
Define all the arguments the Begin activity receives (name and type). The Begin activity defines the interface.



The activities that can call the Begin activity (Subflow and Process Creation) have to complete the expected arguments with instance variables or expressions of their own. This information is passed through the arguments and within the Begin activity instance variable values are set with this data.

Each received argument is preserved in an instance variable, if

desired.

A panel with two columns is shown.




You can add the Arguments and Instance variables that you need at that moment. Select the Arguments  or Instance Variables  icons and complete the information required (See Arguments variables and Instance Variables).

1. Add a new line by clicking on the "+" sign on the top-right of the screen. A new line appears.
2. Click on the first column and select from the pull down menu any of the process instance variables or the writeable predefined variables that are available.
3. Map this variable to an argument. Select it from the pull down


menu in the second column. The selected instance or predefined variable's value is set with the information received in the argument.


Note

 As arguments are not mandatory, this table can be empty.

The Begin activity can define more than one set of arguments mapping with a different number and type of arguments. Thus, each calling activity can use one of these sets of arguments mapping.

To add another set of argument mapping, click on the “+” sign in the left panel and choose a new *Argument set name*. If you have an alternative set of mappings, you will be able to use the **Message based Transitions** (See Message based Transition) as possible outgoing transitions for these activities.

Additionally, you might require validation of the received arguments. You have the option to build a script that runs after the argument mapping is set. Select the **Advanced**  icon. The Method editor opens to add a script.

If you need to define any of these arguments as a Correlation, select the **Correlation**  icon.

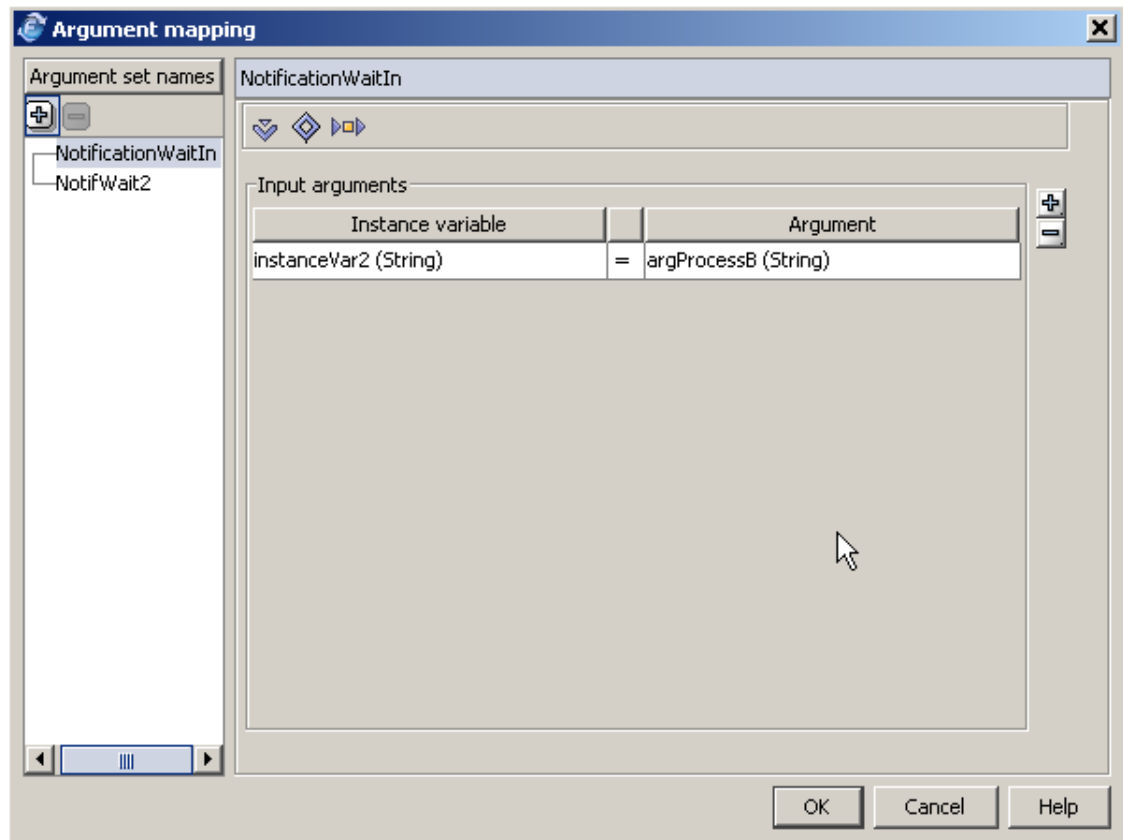
Notification Wait activities

You define all the arguments the Notification Wait activity receives (name and type). The Notification Wait activity defines the interface.

The activity that can call the Notification Wait activity, Process Notification, has to complete the expected arguments with instance or predefined variables or expressions of its own. This information is passed through the arguments and within the Notification activity. Instance variables or writable predefined variables' values are set with this data.

Each received argument is preserved in an instance variable or a predefined variable, if desired.

A panel with two columns is shown.



You can add the Arguments and Instance variables that you need at that moment.

1. Select the Arguments or Instance Variables buttons and complete the information required (See Arguments Variables and Instance Variables).
2. Add a new line by clicking the "+" sign on the top-right of the screen. A new line appears.
3. Click on the first column and select from the pull down menu any of the process instance variables or the writable predefined

variables.

4. Map this variable to an argument. Select it from the pull down menu on the second column. The selected instance or predefined variable's value is set with the information received in the argument.
5. If needed, define any of these arguments as a Correlation.

Note



As arguments are not mandatory, this table can be empty.

The Notification Wait activity can define more than one set of arguments mapping with a different number and type of arguments. Thus, each calling activity can use one of these sets of arguments mapping.

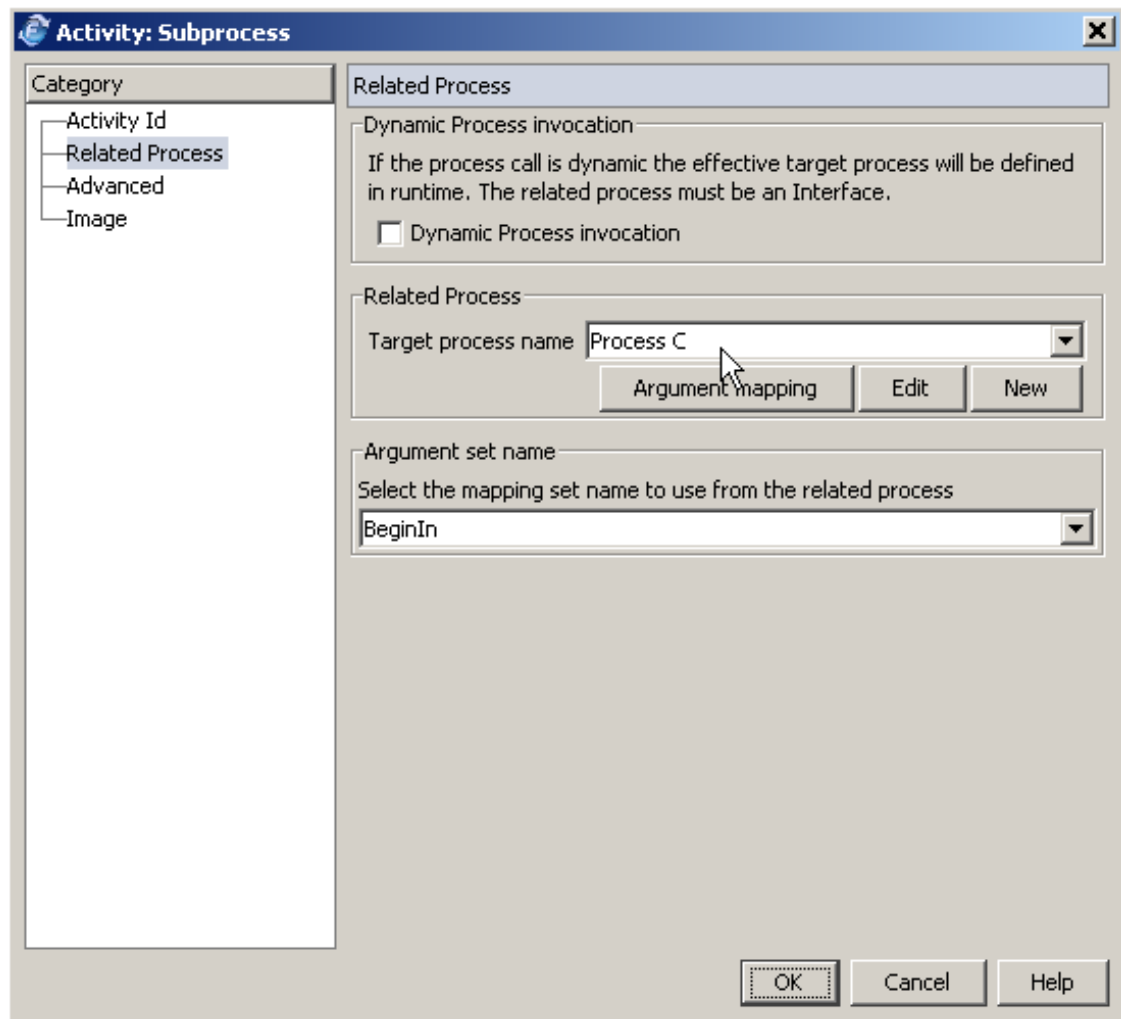
To add another set of argument mapping, click the "+" sign in the left panel and choose a new *Argument set name*.

If you have an alternative set of mappings, you will be able to use the **Message based Transitions** (See Message based Transition) as possible outgoing transitions for these activities.

Subflow (create part), Process Creation

Based on the *related process* that you are calling, you will need to set the values of the required arguments with process instance variables or predefined variables used within the calling process. Alternatively, the arguments can be set by using expressions.

Subflow and Process Creation activities have a related process, the process the activity is calling. This process is defined within the Properties panel:

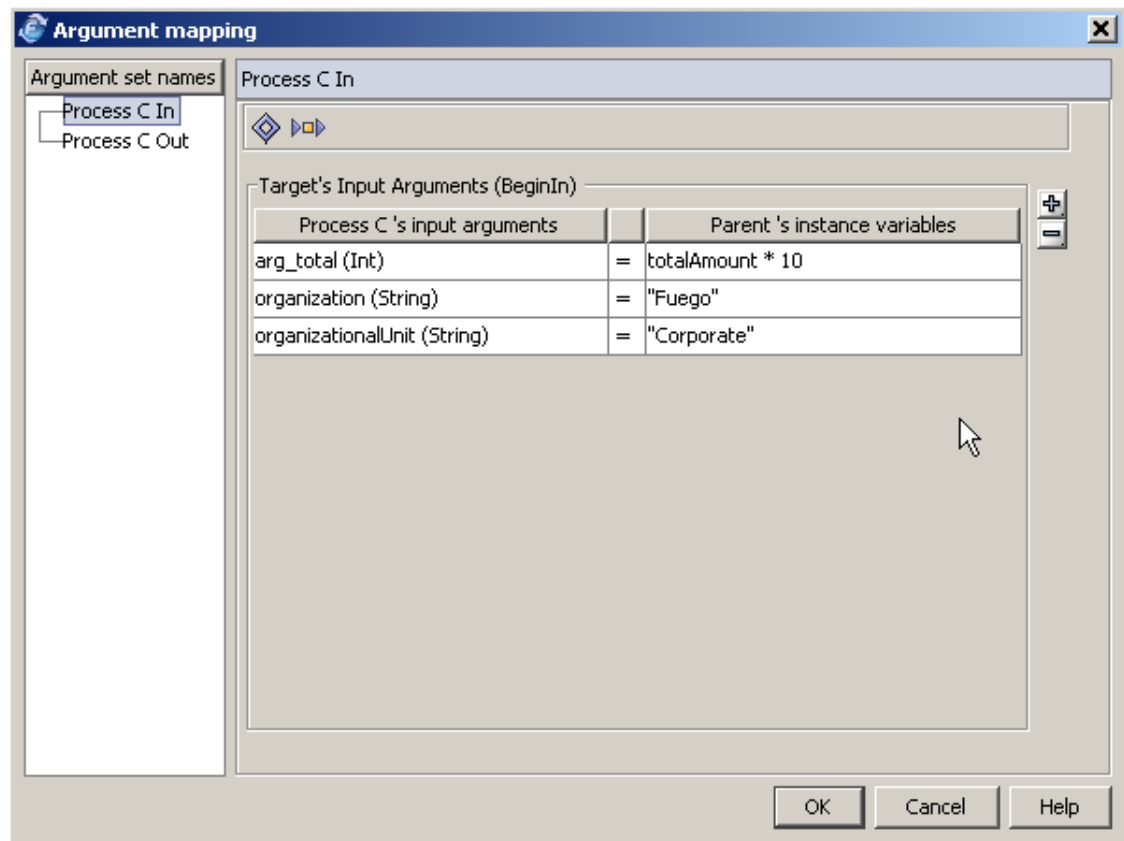


In this case, the Begin activity of the related process defines the arguments to pass to it based on the selected Argument set name.

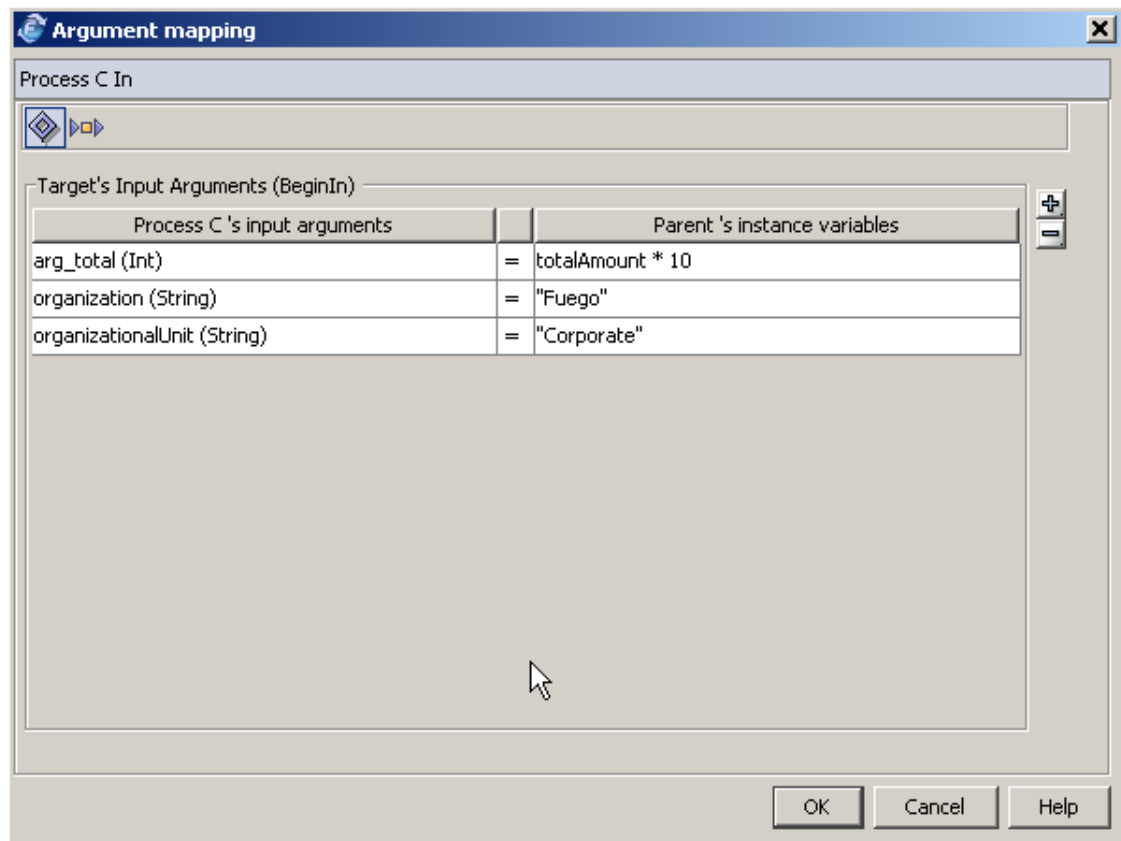
In particular, the Subflow activity has two interfaces: one of them called the **Subprocess create part** and the other called the **wait part**.

The **Process Creation** activity plays the role of the **create part** and the **Termination Wait** activity is the **wait part**.

To define the Argument Mapping in a **Subprocess** activity, you should define the **related process in arguments** to be passed. In the example, the related process is **Process C**.



In a **Process Creation** activity, you should define the same arguments.



You can add the Instance variables that you need at that moment. Select the **Instance Variables** button and complete the information required. (See Instance Variables.)

1. Add a new line by clicking the "+" sign on the top-right of the screen. A new line appears.
2. Click on the first column and select from the pull down menu any of the populated arguments. These arguments are the ones defined by the Begin activity of the related process. The set of arguments to map is also associated in the Related Process Tab within the Properties.
3. Map this argument to a process Instance variable or predefined variable. Select it from the pull down menu in the second column.

The instance or predefined variable value is passed to the argument and will be received later by the Begin activity.

In the second column, you can also define an expression using an instance variable or predefined variable.

If any of the required arguments are not mapped, the called activity receives a null value in that argument. No arguments can be added in these types of activities since they take the arguments from the related activity.

If the called process (in the example *Process C*) is deployed in a specific Organizational Unit, you must define the default *input arguments* **organization** and **organizationalUnit**. These arguments are passed to the predefined variables in the called process.

If you need to define any of these arguments as a Correlation, select the **Correlation** button.

Process Notification

Based on the *Notification* you need to perform, you need to set the values of the required arguments with process instance variables or predefined variables used within the notifying process. Alternatively, you can set the argument values by using expressions.

The Process Notification activity has an associated Process/activity, which it notifies. This process/activity is defined within the **Properties** panel:

Activity: Process Notification

Category

- Activity Id
- Notification
- Advanced
- Image

Notification

Related Process

Target process name: Process B

Argument mapping Edit New

Notifies : Parent process

Select a Wait Notification activity of the related process.

Activity: Notification Wait

Argument set name

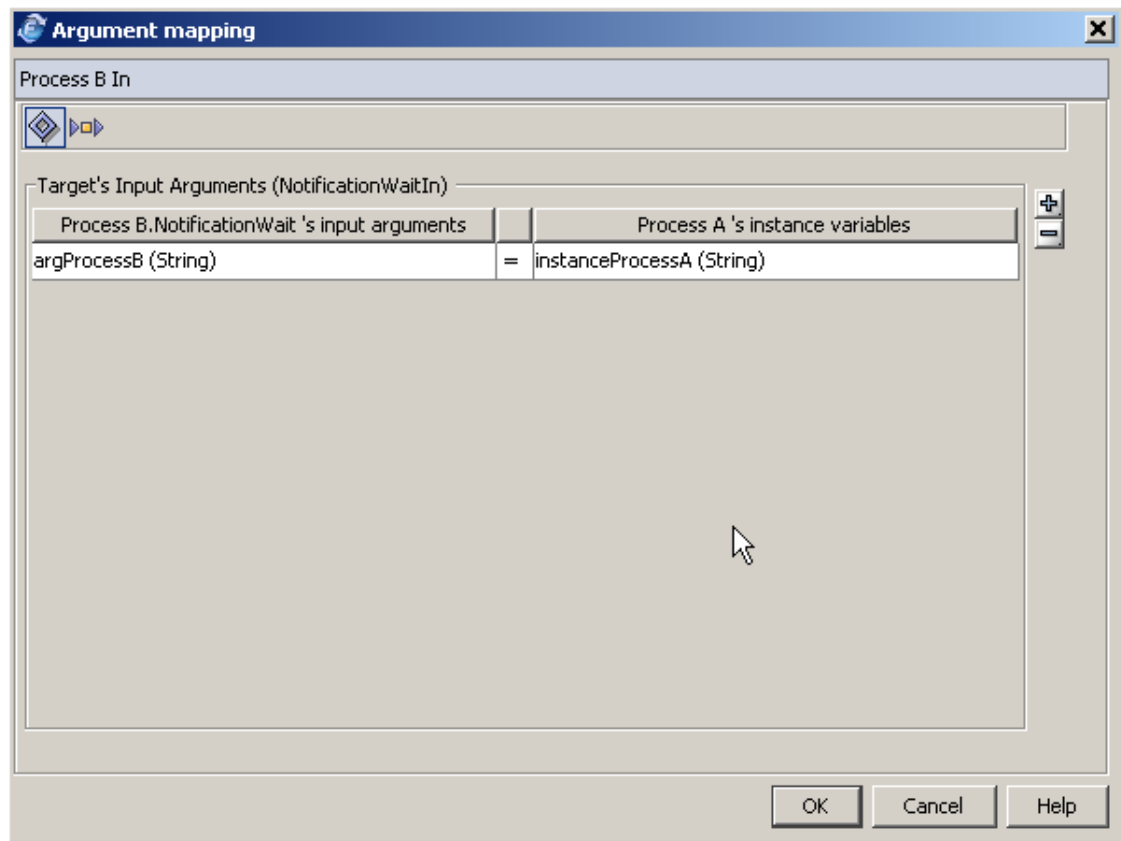
Choose the argument set to use of the selected target activity

NotificationWaitIn

OK Cancel Help

In this case, the Notification Wait activity of the related process defines the arguments to pass to it based on the selected Argument set name.

In a **Process Notification** activity, you need to define the **related process input arguments**. In the example, the Related Process is **Process B**.



You can add the Instance variables that you need at that moment. Select the **Instance Variables** button and complete the information required. (See Instance Variables.)

1. Add a new line by clicking the "+" sign on the top-right of the screen. A new line appears.
2. Click on the first column and select from the pull down menu any of the displayed arguments. These arguments are defined by the Notification Wait activity of the related process. The set of arguments to map is also associated in the Notification Tab within the Properties. The peerInstanceID is used if it is an external notification and no relationship between the notifier and the notified activities was kept. The peerInstanceID has to be completed with the InstanceID that you specifically want to notify.

3. Map this argument to a process Instance variable or predefined variable. Select it from the pull down menu in the second column.

Map this argument to a process Instance variable or predefined variable. Select it from the pull down menu on the second column.

On this second column, you can also define an expression using, or not, an instance variable or predefined variable .

The instance or predefined variable value, or the expression will be passed to the argument and will be later received by the Notification Wait activity.

If any of the required arguments are not mapped, the called activity will receive a null value in that argument.

No arguments can be added in these types of activities since they take the arguments from the related activity.

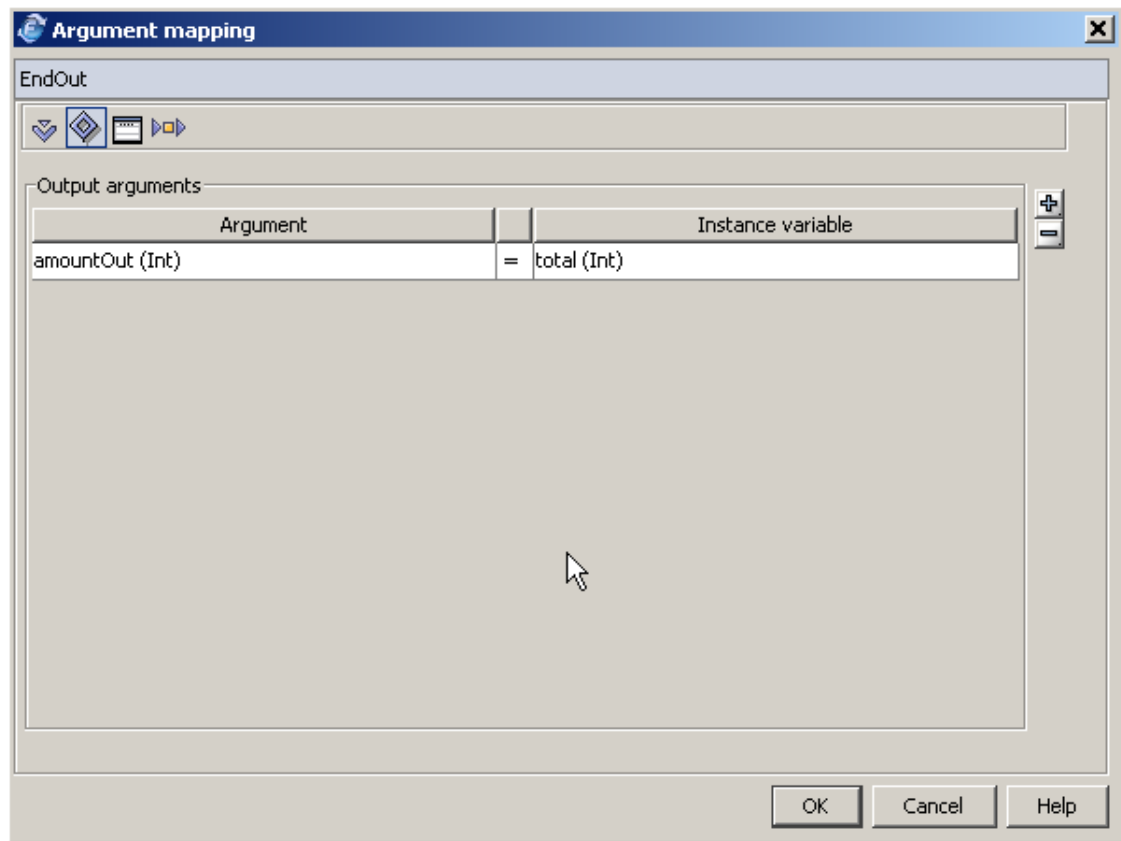
End

You define all the arguments the End activity returns (name and type) to the calling activity.

The End activity defines the interface.

The activities that receive the returned values are Subflow & Termination Wait activities. Therefore, they complete the argument mapping by defining in which instance variables or predefined variables they receive the defined arguments. See (Subflow – wait part or Termination Wait).

A panel with two columns is shown.



You can add the Arguments and Instance variables that you need at that moment.

Select the Arguments or Instance Variables buttons and complete the information required (See Arguments Variables and Instance Variables)

First of all, add a new line by clicking the "+" sign on the top-right of the screen.

A new line populates.

Click on the first column and select from the pull down menu any of the populated output arguments.

Map this argument to a process Instance variable or predefined variable. Select it from the pull down menu on the second column.

On this second column, you can also define an expression using, or

not, an instance variable or predefined variable .

The instance or predefined variable value or expression will be passed to the argument and will be later received by the Subprocess or Termination Wait activities.

Additionally, you might require to do some clean up (free resources as closing a file) before the instance reaches the end of the process. You have the option to build a script that runs before the argument mapping. Select the **Advanced** icon. The Method editor opens to add a script.

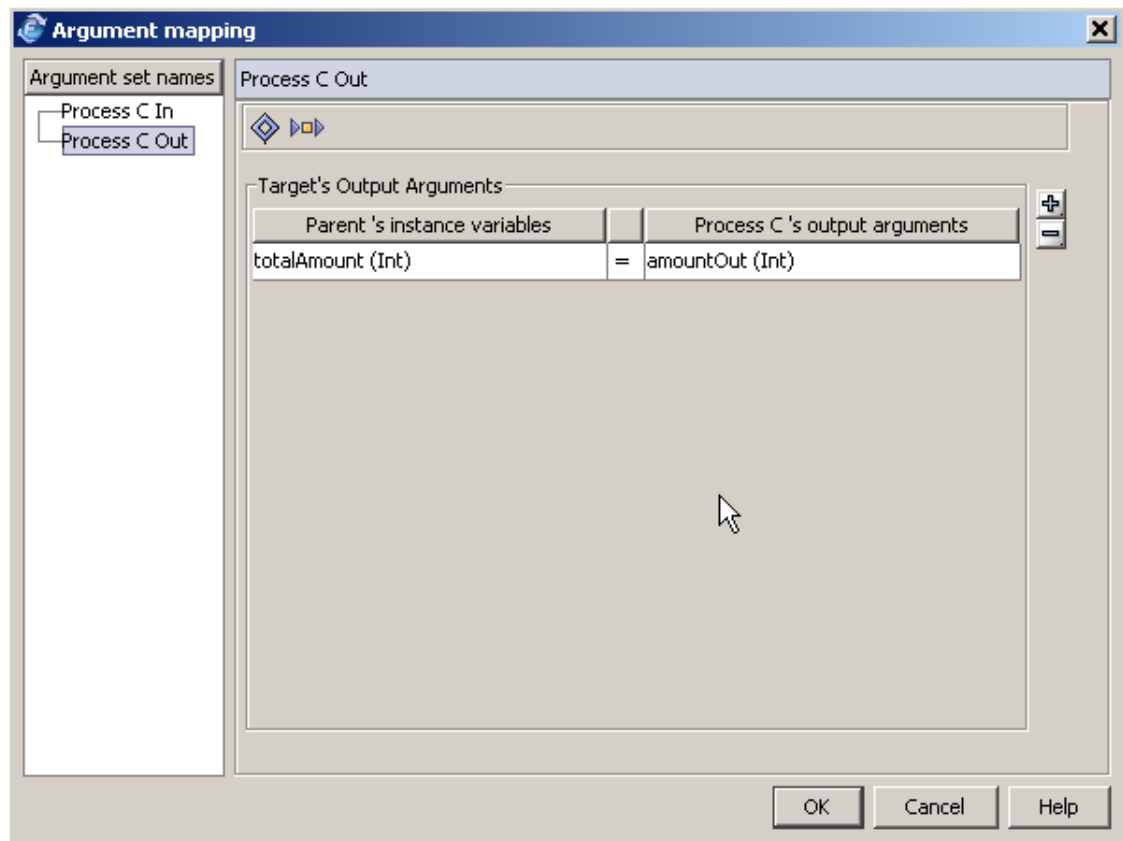
Subflow (wait part), Termination wait

Based on the *Related process* you are calling, you need to define all the arguments you receive from the called process (from the End activity) and match them with Instance variables or predefined variables used within the calling process.

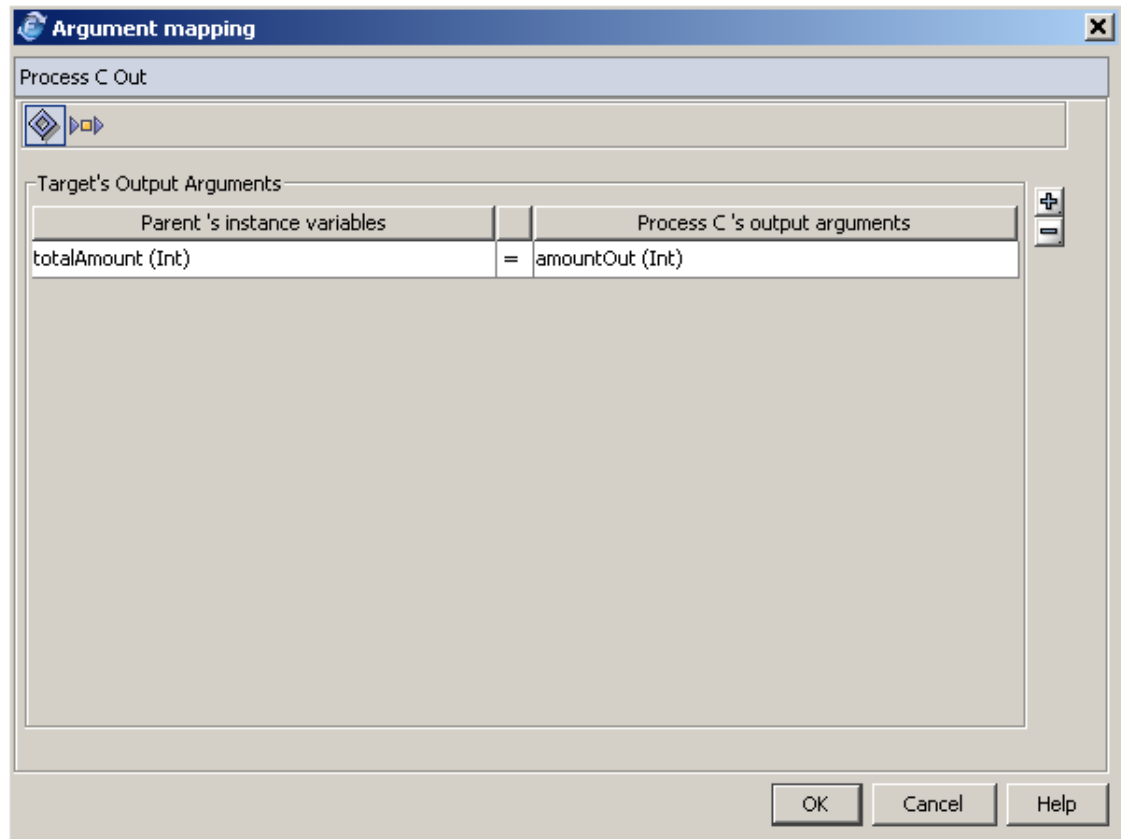
Each received argument will be preserved in an instance variable, if it is so desired.

A panel with two columns is shown.

Subflow



Termination Wait



You can add the Instance variables that you need at that moment.

Select the Instance Variables button and complete the information required (See Instance Variables.)

First of all, add a new line by clicking the "+" sign on the top-right of the screen.

A new line populates.

Click on the first column and select any of the process instance variables from the pull down menu.

Map this variable to an argument. Select it from the pull down menu on the second column. These arguments are the ones defined by the related process that is returning arguments through its End activity.

Since receiving these arguments is not mandatory, this table can be empty.

For these types of activities no arguments can be added or removed since the interface is defined by the End activity of the related process.

Procedures and Screenflows

Procedures and Screenflows Argument Mapping are defined in the Begin and in the End Activities of the procedure or screenflow.

Anyone who invokes any of them should map these arguments.

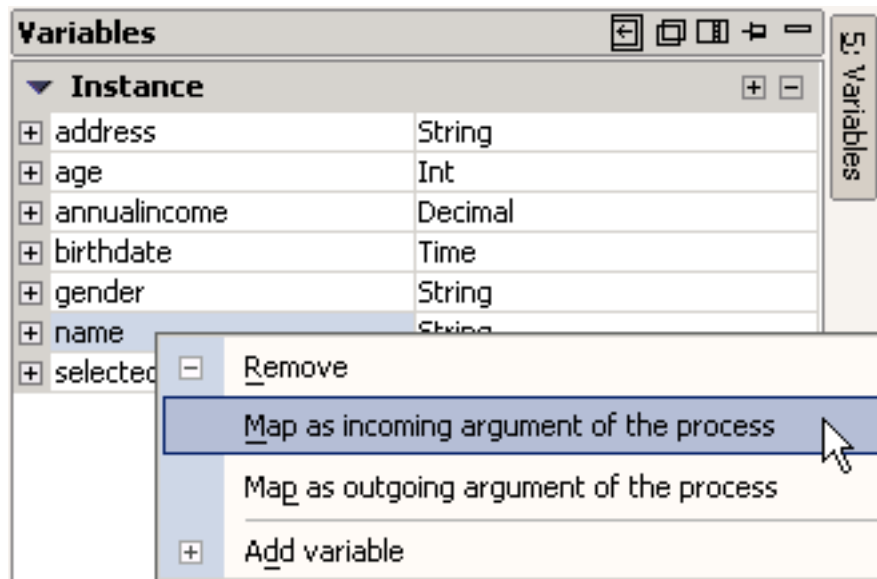
For example, an activity task can be implemented by invoking a procedure. Therefore, it will have to map the instance or predefined variables to the required arguments defined by the procedure to receive (in its Begin activity) as well as those that the procedure sends back (in its End activity.)

See Tasks for further information on how to use a procedure.

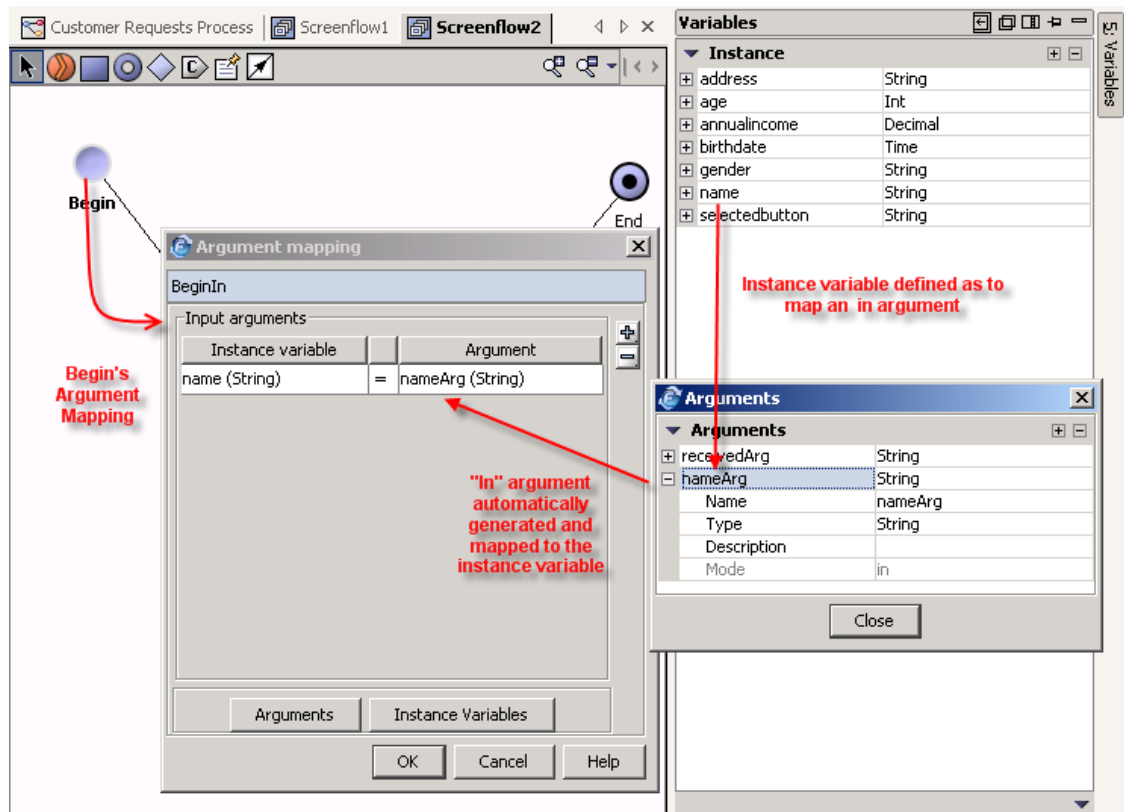
From the task you are implementing as a Procedure or Screenflow, select the **New** option. In this case, you are guided to create the procedure through a wizard. This option facilitates all instance variables creation, **arguments creations and argument mapping**. For further information, see Generate Processes, Procedures and Screenflows automatically.

Defining an Instance Variable as an in/out argument

To simplify the argument mapping, right click on the name of any of your already defined instance variables. You can map this variable as an **incoming** argument or an **outgoing** argument.



In and Out arguments are automatically generated. If it was mapped as an **in** argument, the Begin argument mapping is set. If it was mapped as an **out** argument, the End argument mapping is completed.



Calling Components

Argument mapping is also used in a component method call task defined in an interactive or automatic activity.

After selecting the component and the method of the component to be used, if the method receives or returns any type of argument, these arguments should be filled in or mapped by clicking on the argument mapping button in the task configuration panel.

Main task - Activity Automatic

Automatic

Implementation type
Component

Component method

Properties

Activity task execution timeout: 5m

☐ Use instance variable

Component name:
Fuego.Lib.Participant

Browse

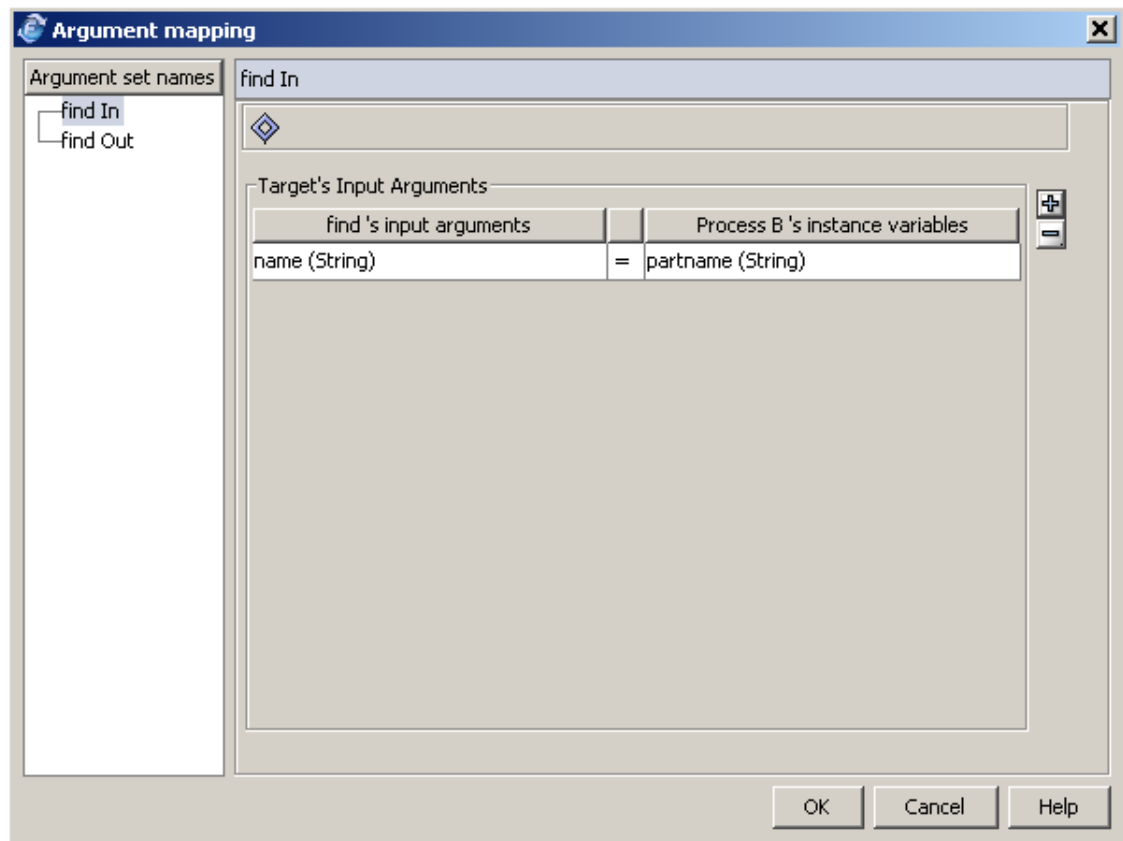
Component member

Method Name
find(in name : String) : Fuego.Lib.Participant

Argument mapping

OK Cancel Help

If the component receives arguments, they can be filled out by mapping process instance variables with them. To retrieve the method's return value or any of the component public attributes, an outgoing mapping can be defined. On the left column of the mapping, a list of all the process instance variables will be shown. They can be mapped to any of the public attributes the component has or to the method return value. This list will be shown on the right column of the mapping panel.



Correlations

Correlation sets are the implementation for a custom mechanism for instance searching.

Consider the usual supply-chain situation where a buyer sends a purchase order to a seller. The seller needs to asynchronously return an acknowledgment for the order, and the acknowledgment must be routed to the correct business process instance at the buyer. To do this, a *business token* is copied into the acknowledgement for correlation, for example, the *purchase order number*.


At designer time, the designer can define the different correlation sets that can be used during the process.

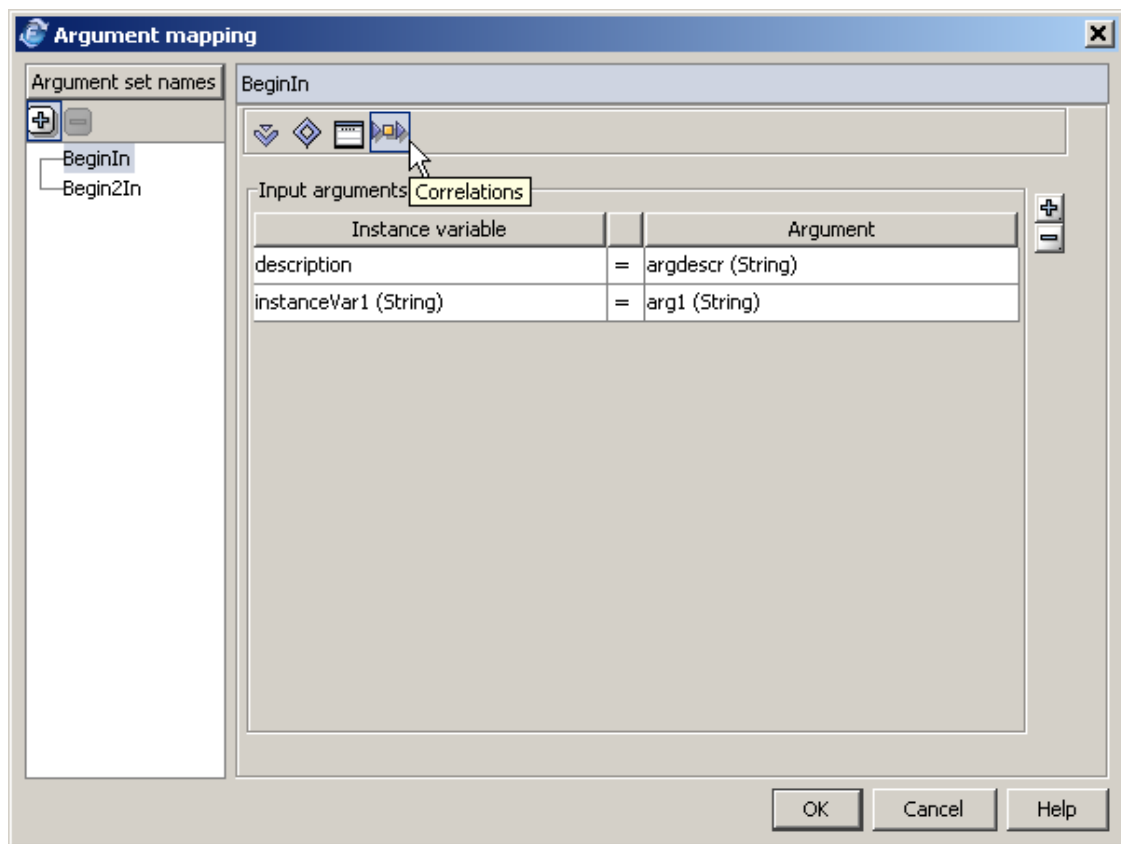
The correlation sets are composed by a name, and the properties of the set with their proper types. The properties are references to the selected arguments of the Argument set.

For example, a correlation set name is a : "Purchase Order", and the properties are "clientId" string, "orderNo" integer.

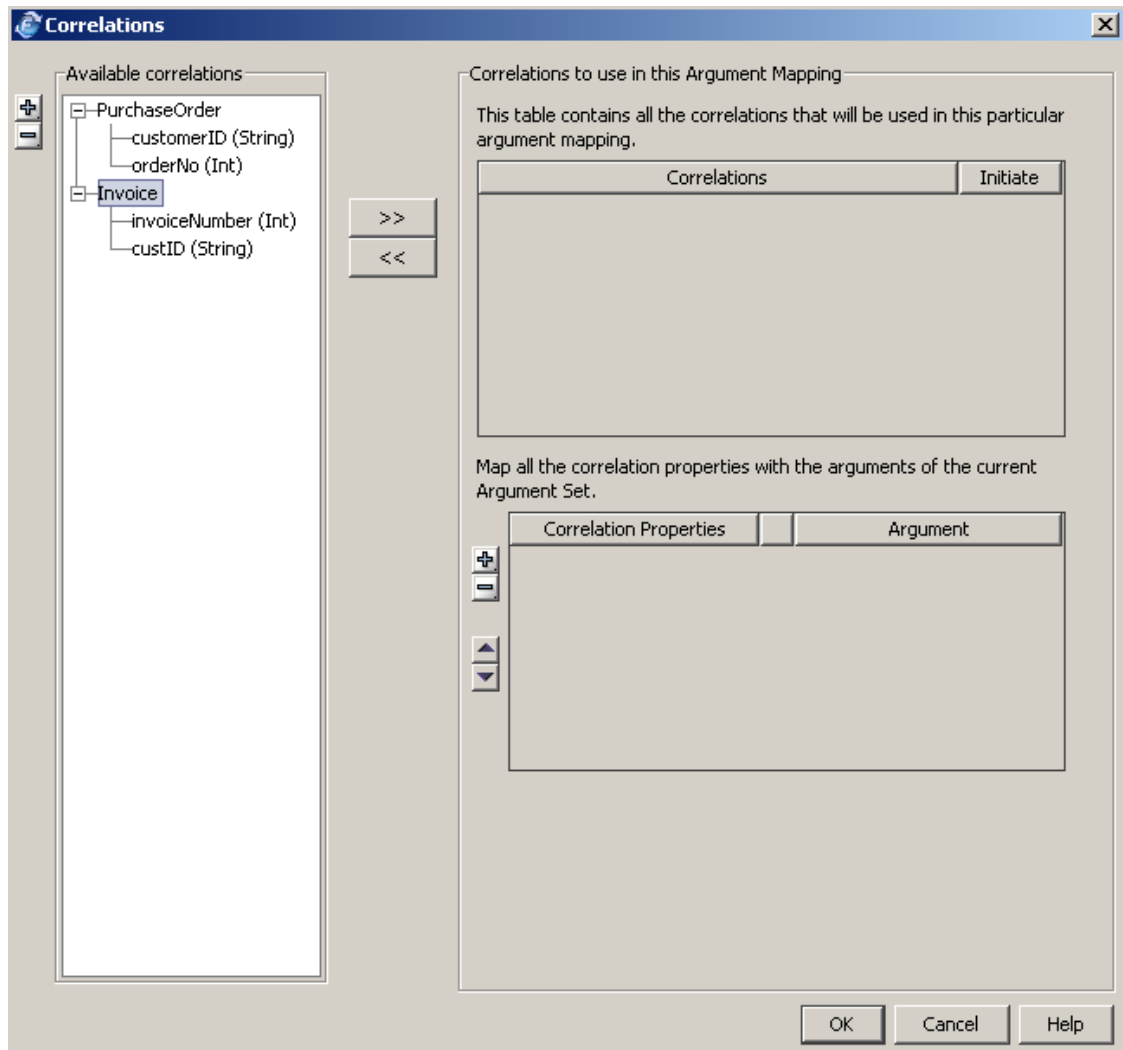
Adding Correlations

Correlations are defined in the Argument Mapping panel for the **Begin**, **Subflow**, **Process Creation**, **Termination Wait**, **Process Notification** and **Notification Wait**.


Select the **Correlations**  icon from the Argument Mapping panel:

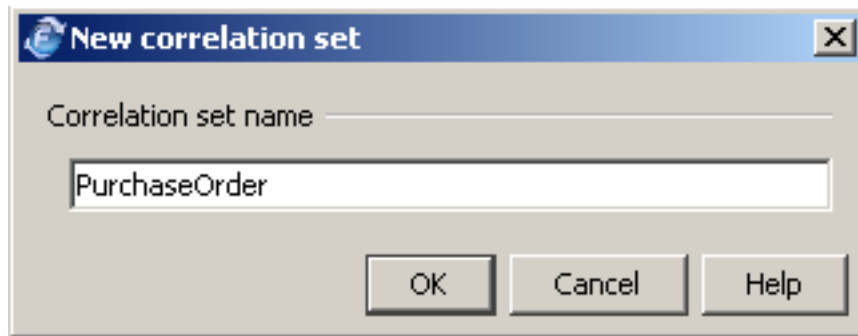


The correlations dialog displays:



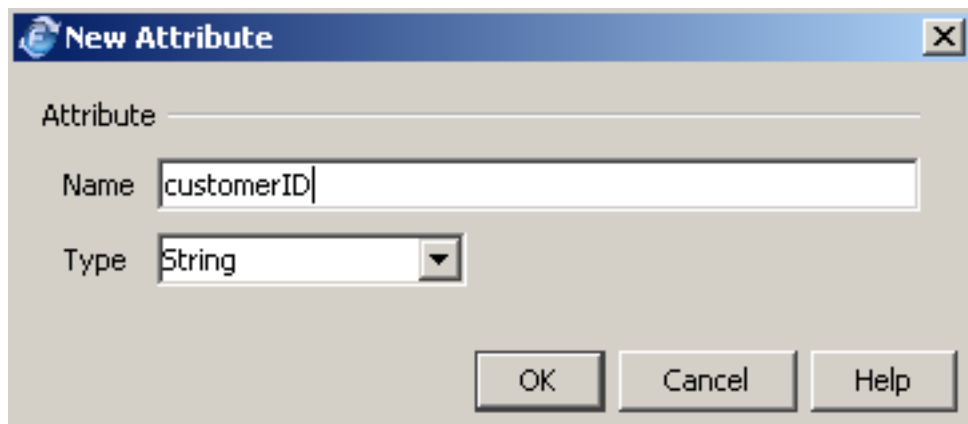
In the left panel, you can add all the correlations that are used in the process. Every correlation defined in the tree will be available to every argument mapping of any IPC activity of the process.

To add a **Correlation** click the  button. Complete the Correlation set name.




Each correlation has a set of properties with name and type (only simple types).


Right click on the new added Correlation and add the properties

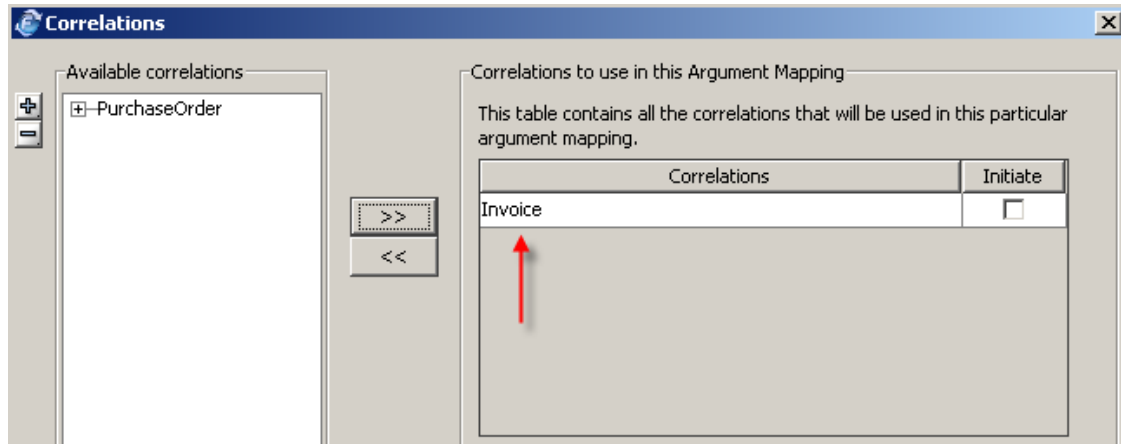


Note

 You must not change the order in which you defined the properties

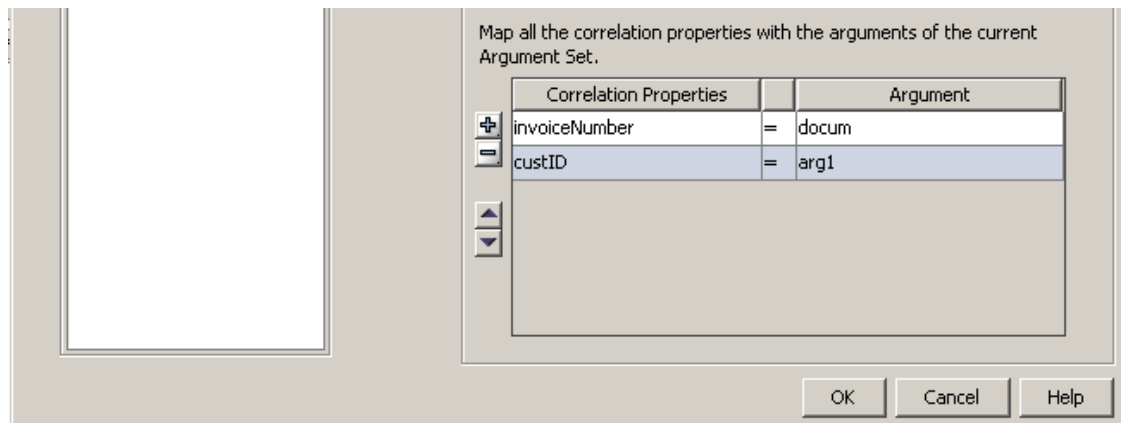
Selecting a Correlation

To use a **Correlation**, select it from the left panel and add it by pressing the  button. You can select as many correlations as needed.




Select the **Initiate** check box if you want the correlation to be persisted or not by the caller/receiver part. You must *initiate* the correlation when you need it to be created. If you do not initiate it, the purpose of the correlation is for *look up*.

In the right bottom panel, all the correlations properties are displayed so you can map them to any of the arguments defined in that Argument set Mapping or just type an expression.



Note

 Be sure that the values that the correlation will have are unique so the instance can be correctly found

An activity can have more than one argument set, and each argument set can have more than one correlation set mapping its

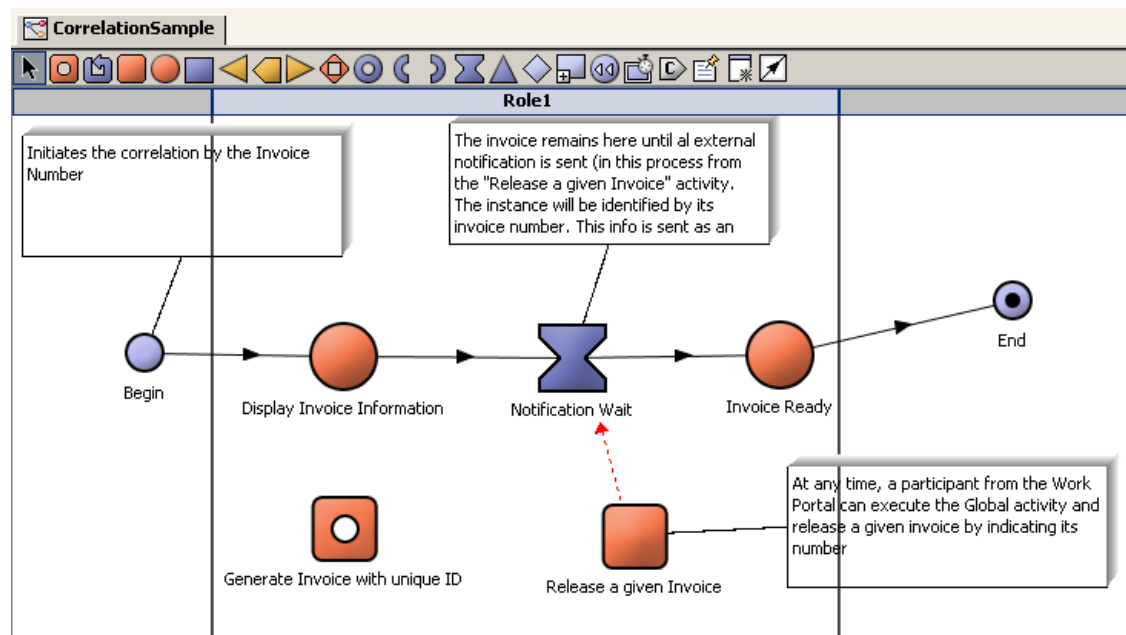
values.

In the case of a Wait Notification, the Argument Set that has a Correlation defined does not necessarily need to have any Argument Mapping (pair instance variable / argument). You can define arguments only to be used in the correlation. They receive the values that are assigned to the Correlation's properties.

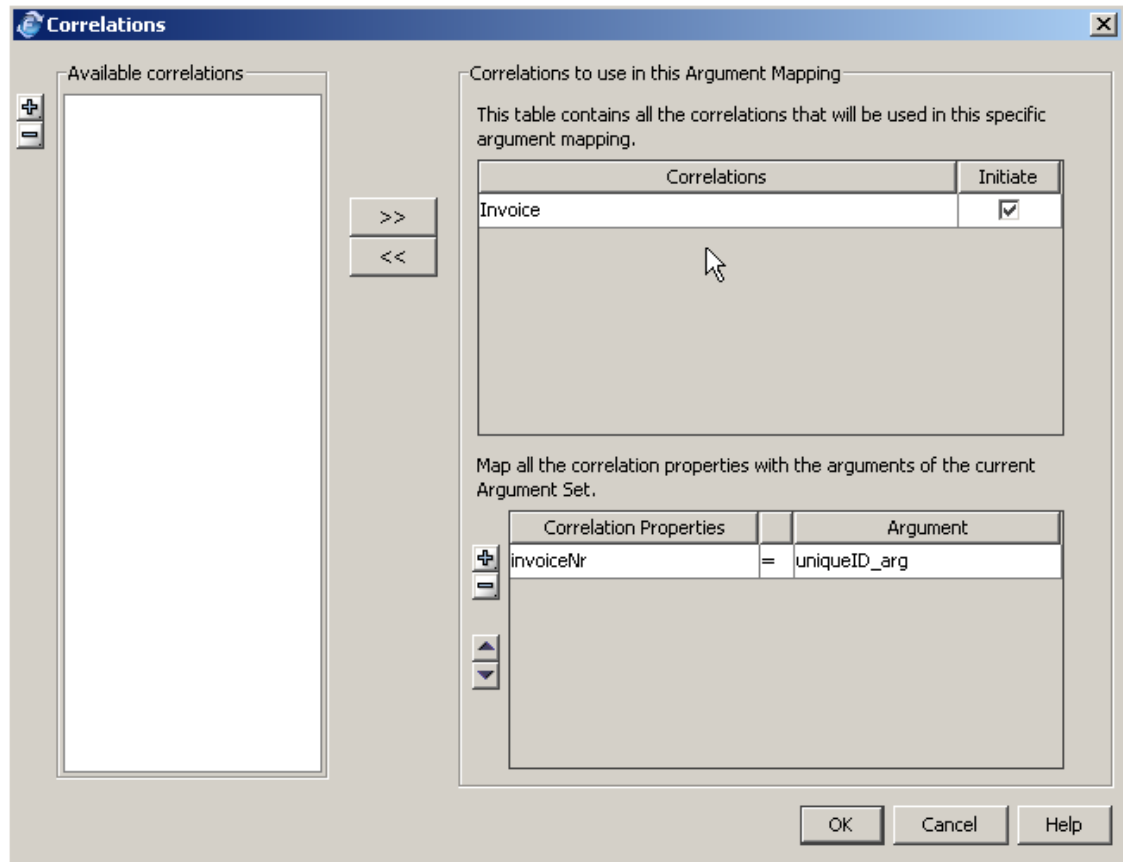
Examples

Case 1

The following is a simple example where the instance represents an Invoice. The invoice number is unique therefore the instance can be identified by the Invoice Number.



The Begin activity defines the argument mapping and as well initiates a Correlation called "Invoice":



Later in the process the instance will be put on hold in the **Notification Wait** activity.

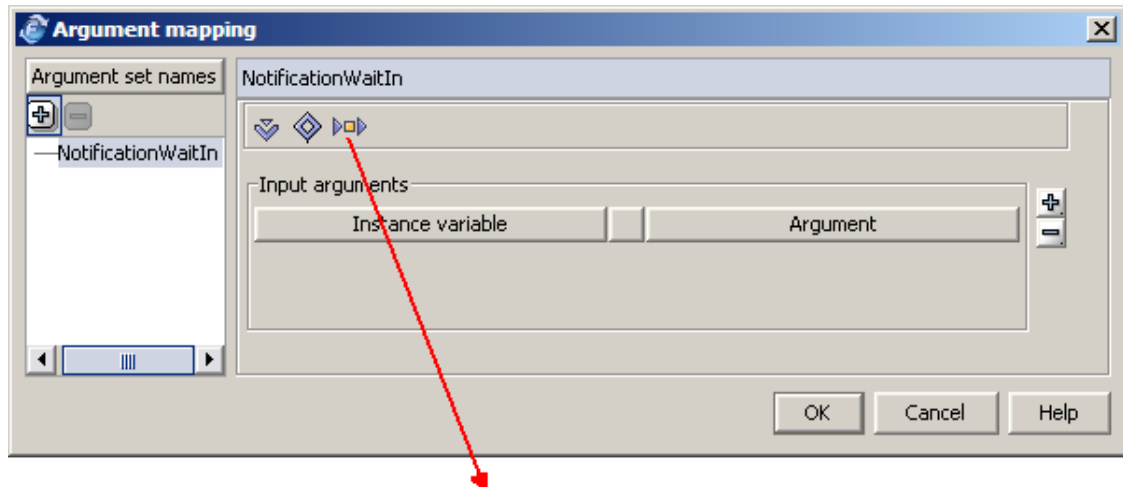
The notification is sent through the "Release a given instance" activity (Global activity) , that uses the **Notification** Fuego's component.

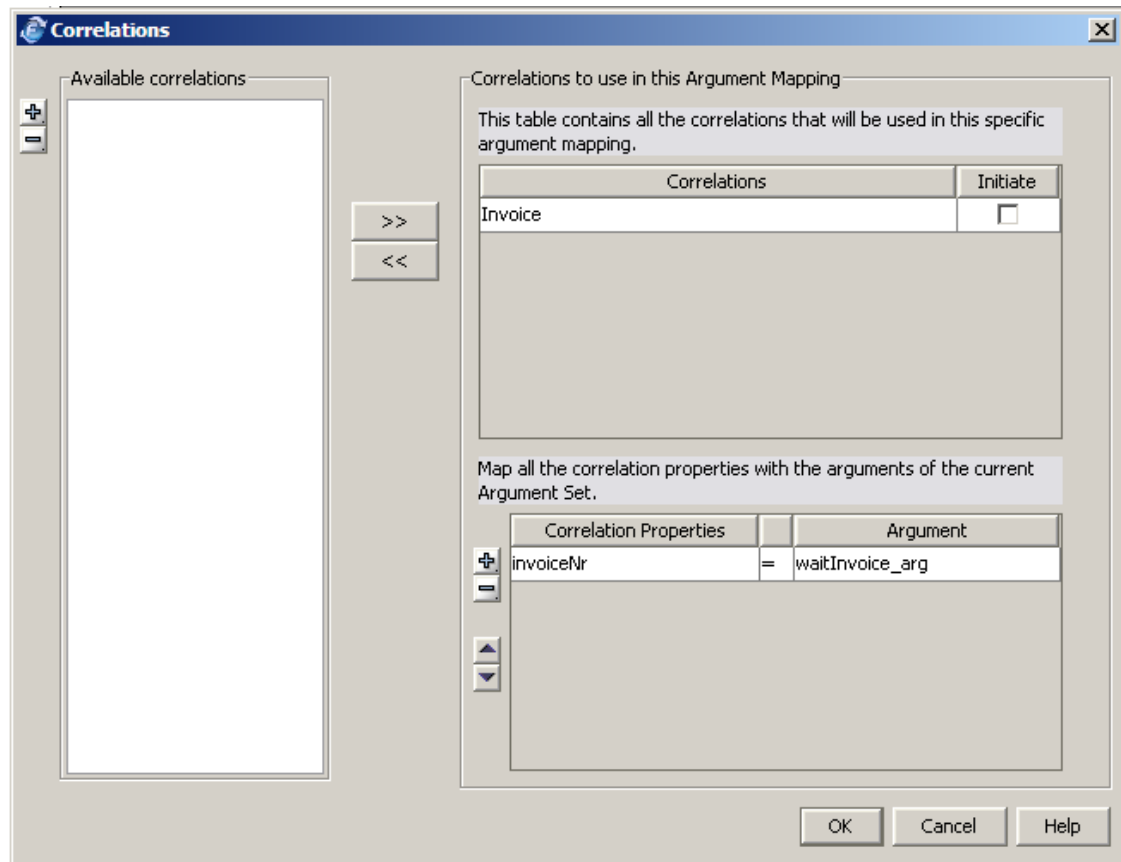
```
input "Input Invoice Numbre
(Instance unique ID you want to notify) : "
: invoiceNum

myargs["waitInvoice_arg"] = invoiceNum
send Notification
    using processId = process.id,
        activityName = "NotifcationWait",
        arguments = myargs,
        argumentSetName = "NotifcationWaitIn"
```

This Notification is received by the **Notification Wait** activity.

The received arguments in the *Notification Wait* activity, that are mapped to the correlation, are used to look up for the correct instance and notify it.





Find the project **CorrelationCase01.fpr** in FuegoBPM's installation directory, in the studio/samples directory to study the example further.

The FuegoBPM's participant name is: **test**.

Case 2

The following shows an example of two processes:

1. Repair Request

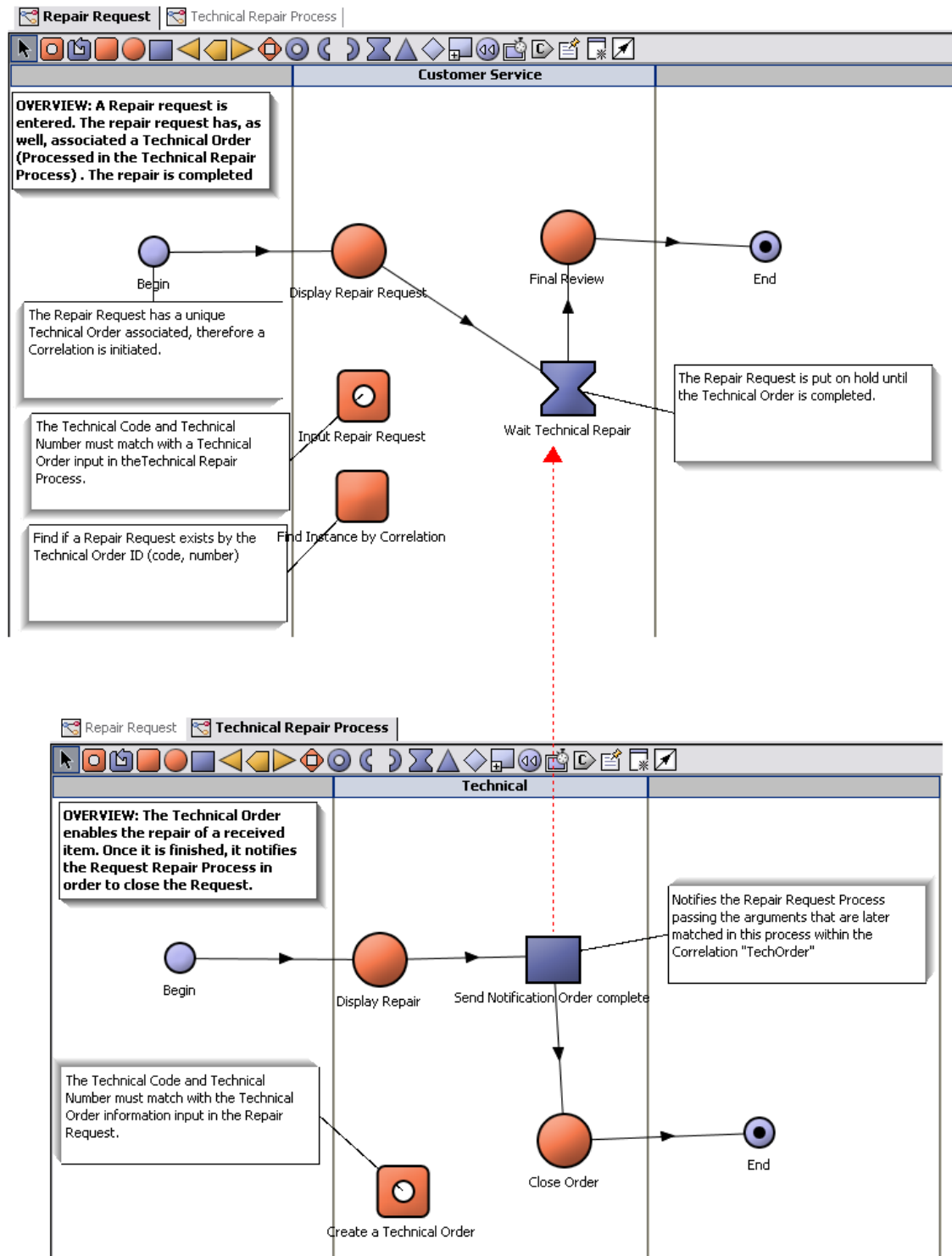
This process manages **Customer's Repair Request** at the Customer Service Department. The **repair request** matches the instance.

Every repair request has a unique **technical order** associated.

2. Technical Repair Process

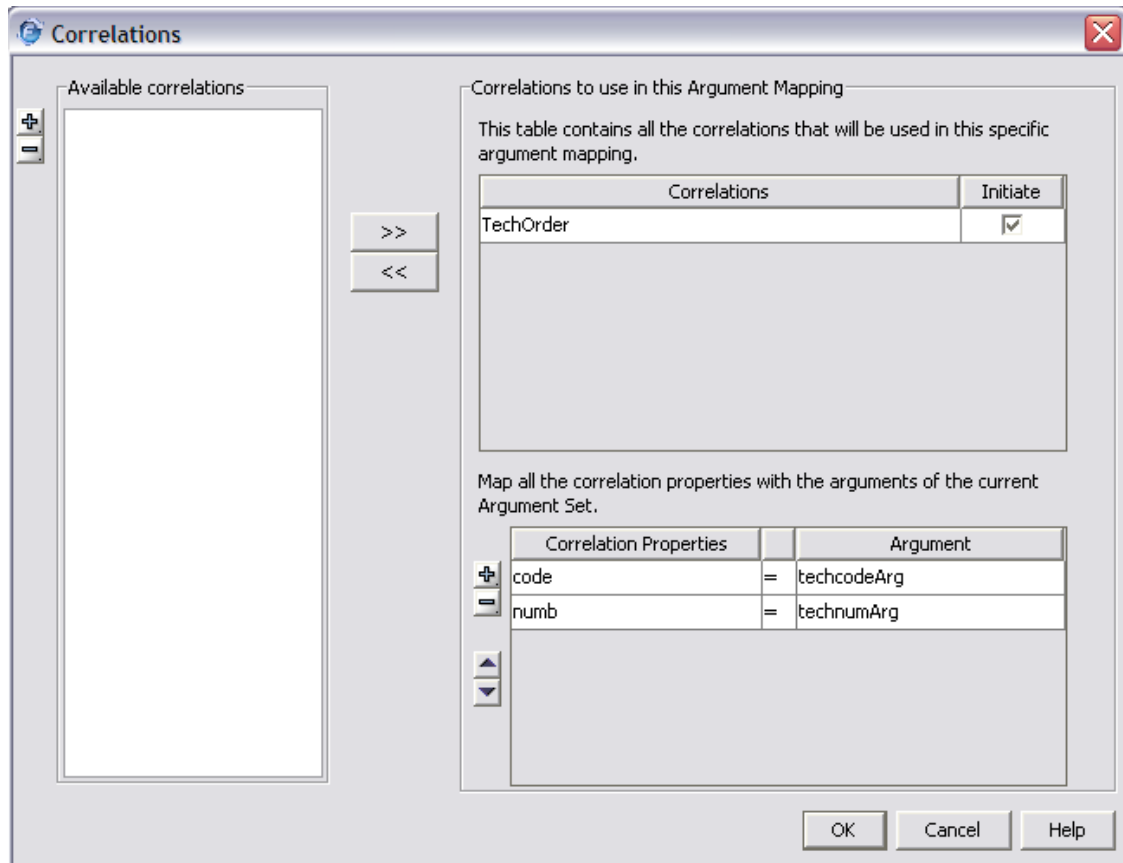
This process manages **Technical repair Orders** at the Technical Department. The **technical order** matches the instance. The technical order is identified by a **technical code** and a **technical number**.

Each technical order corresponds to a **repair request**.



Repair Request Process

The Begin activity defines the argument mapping and as well initiates a Correlation called "TechOrder".



The Repair Request instances are, as well, identified by the technical order ID (code/number).

Later in the process the instance will be put on hold in the **Wait Technical repair** activity. The instance is released when receiving a notification that is generated in the other process, Technical Repair Process.

When the notification is received, the received arguments are mapped to the correlation, and used to look up for the correct instance and notify it.

Technical Repair Process

The **"Send Notification Order complete"** activity (Automatic

activity) sends a notification to the other process and within its arguments; it sends those that are defined in the Correlation **TechOrder**. When received on the other process, they are matched to the instance and therefore, released from the Notification Wait activity.

The automatic activity implements the **Notification** Fuego's component:

```
corrargs as Any[Any]
corrargs["techorder_arg"] = technicalCode
corrargs["technum_arg"] = technum

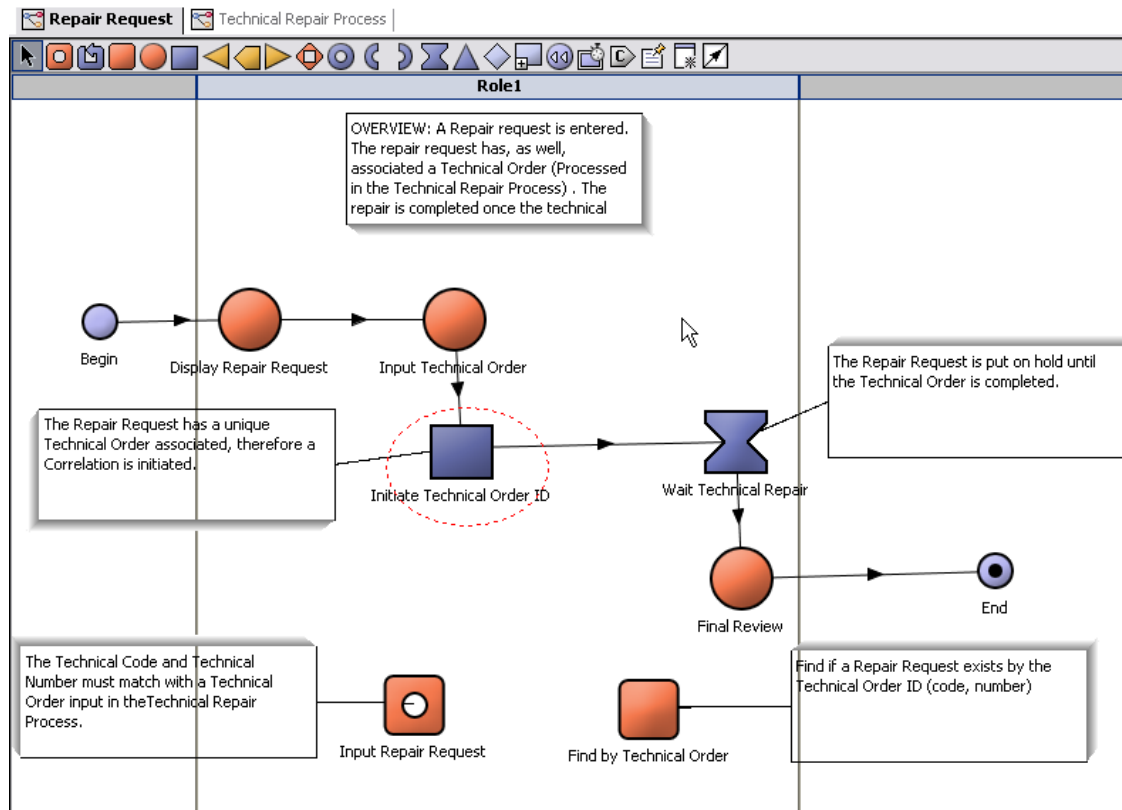
send Notification
  using processId = "/RepairRequest",
    activityName = "WaitTechnicalRepair",
    arguments = corrargs,
    argumentSetName = "WaitTechnicalRepairIn"
```

Find the project **CorrelationCase02.fpr** in FuegoBPM's installation directory, in the studio/samples directory to study the example further.

The FuegoBPM's participant name is: **test1**.

Case 3

The following example is similar to Case 2 but with a small variation.



The **technical order ID** that identifies the **repair request** using a correlation is not initiated at the begin activity when the instance is created. It is initiated further in the process using the Correlation fugeoblock component. The correlation is:

The FBL that *initiates* the correlation is:

```
//Initiate a Correlation (defined as "TechOrder")
//for this instance to be
// identified by "technical code / technical number"

initiate Correlation
    using name = "TechOrder",
    values = [techcode, technum]

// techcode and technum are instance variables in which
//the technical order info
// is stored
```


Find the project **CorrelationCase03.fpr** in FuegoBPM's installation directory, in the studio/samples directory to study the example further.

Case 4

The following shows an example of three processes:

1. Customer Request Order

This process manages a **Customer's Order Request** at any Department. Any requester can need to purchase items and generates a **customer order request** that matches the instance.

Every **request** will have a unique **provider's invoice** associated identified as providerID/invoice number.

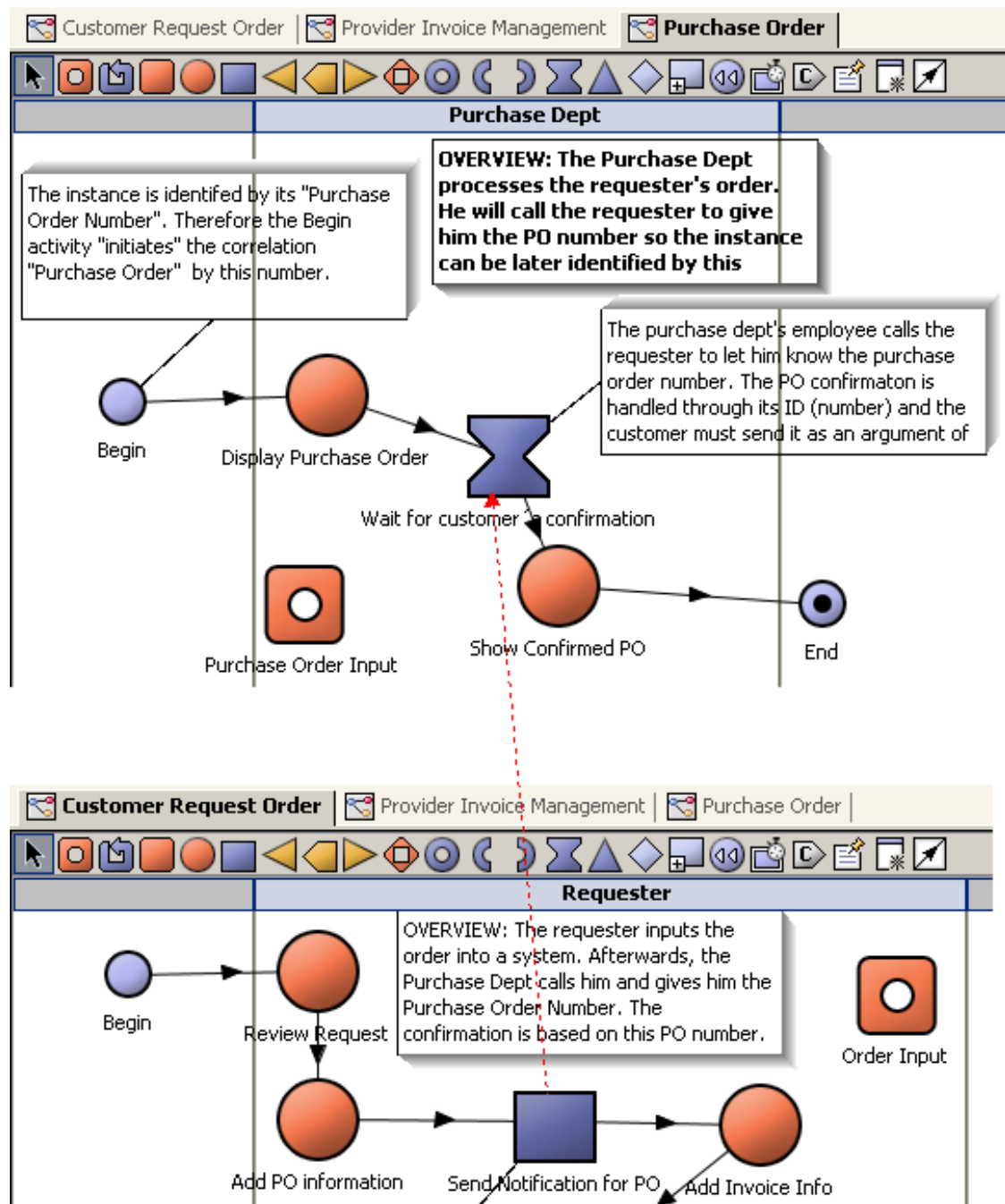
2. Purchase Order

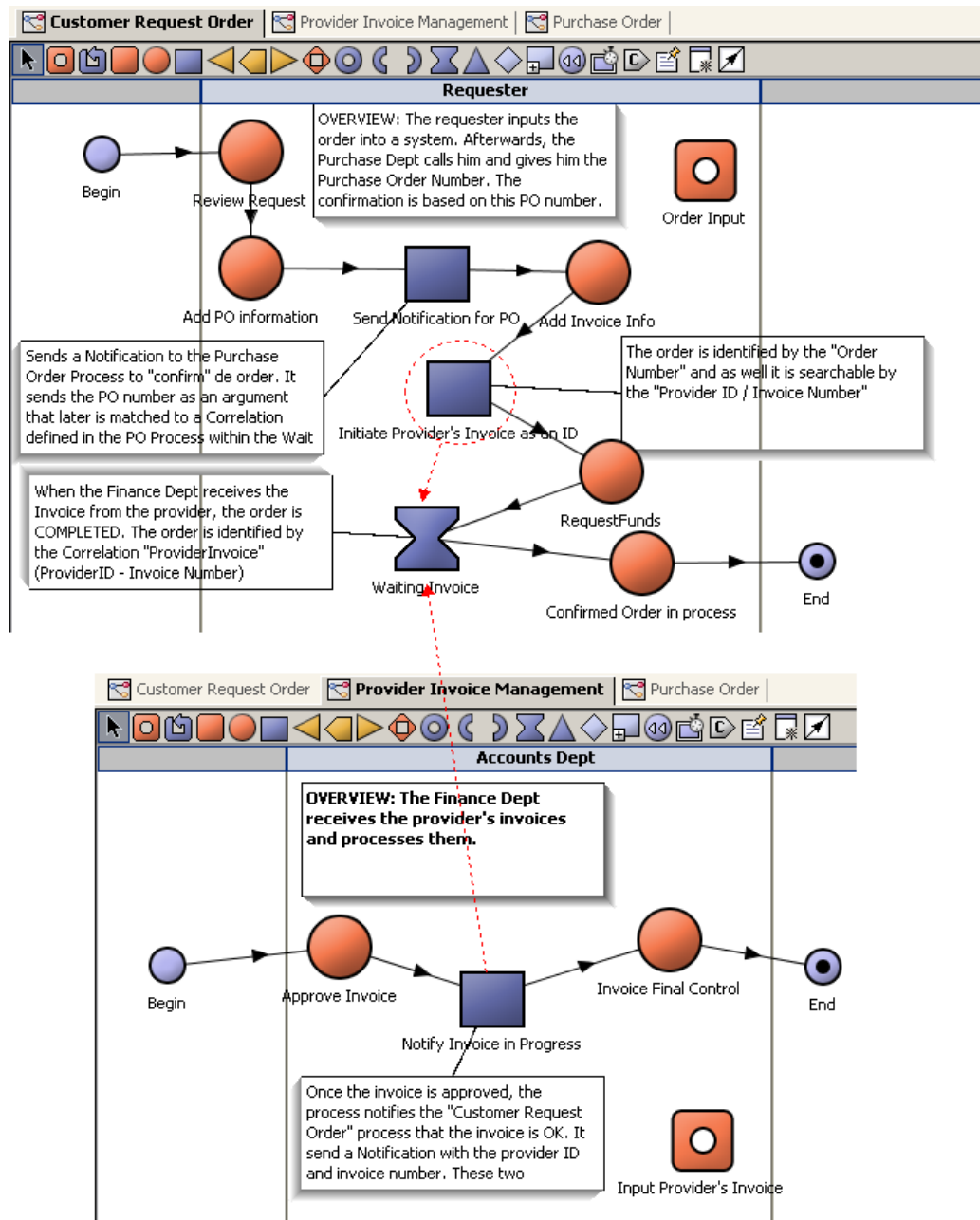
This process manages **Purchase Orders** at the Purchase Department. The **purchase order** is generated in an independent process and matches the instance. The purchase order is identified by its **number** and needs a final confirmation by the requester (Customer Request Order Process).

Each technical order corresponds to a **repair request**.

3. Provider Invoice Management

This process manages all the received **Provider's Invoices** at the Accounts Department. The **invoice** matches the instance. Each invoice corresponds to a **customer order request**.





Remarks:

1. The **Customer request Order** process identifies its instances by **Order Number**, and as well, in the middle of the process, it **initiates** a correlation for the **invoice ID**.

2. The **request order** remains stand by until the provider's invoice is received. The **Provider's Invoice Management** process notifies it.
3. The **purchase order** information associated to the **order request** is given manually, on the phone. So later the **Customer request Order** process can notify the **Purchase Order** process using that information.

Find the project **CorrelationCase04.fpr** in FuegoBPM's installation directory, in the studio/samples directory to study the example further.

Managing Exceptions

When using correlations and sending a Notification, the exception **CannotStoreNotificationException** can be caught if any problem occurs.

For example:

```
args["invoiceNum"]=invoiceNum

do
  send Notification
    using processId = "/ProcessName",
    activityName = "NotificationWait",
    argumentSetName = "NotificationWaitIn",
    arguments = args

on e as CannotStoreNotificationException
  display "Failed to find invoice:  '" + invoiceNum +
    "    used as a correlation value"
end
```

Chapter 7. Groups

Groups



A Group is a compound activity. It is composed of a set of activities that may include other Groups. Sometimes the elements of the Group are called Sub-Activities/Sub-Groups or Inner-Activities/Inner-Groups.

Groups provide exception and compensation handlers as well as time outs within the group.

When should you use Groups ?

The following list provides some guidelines to help you decide when to use groups. Use groups when:

- You need to define a due interval within which a group of activities must be completed.
- You need to manage a specific exception that can occur in any activity within a group. (Instead of handling the exception in each activity, you can define the group and handle the exception from within it.)
- You need to reverse (compensate) a certain situation that involves more than one activity to be rolled back.
- You need to manage a group of activities as a unique transaction (Atomic transaction.)
- You need to clean up the design. Groups help you better visualize the process design since you can group a set of activities and collapse the group in order to see only one activity that represents a group of activities.

A group  /  can be formed by any activity (also called leaf

activity) with the exception of Global, Global Creation, Global Automatic, Begin and End activities. Also keep in mind that a group **cannot** contain **only** a split or a join activity.

Characteristics

The group is displayed as a dotted line rectangle that contains the activities/subgroups inside the group.



Right-click on the rectangle to view the following options:

- Properties
- Collapse/Expand group
- Add due transition
- Add compensate transition
- Add exception transition

Creating a Group

There are several ways to create a Group:

- Drag the group icon from the toolbar to a role lane. Move all the activities or groups that compose this new group into the dashed box that represents it. Be aware that all transitions from or to the activities that are moved will be lost.
- Select all the activities/groups within the design that should be moved to the group. Right-click and select **Create group with selection**. In this case, transitions between the activities will remain and two new ones will be generated: a unique incoming transition to the group and a unique outgoing transition from the group.

- Select the **Group** icon  /  from the toolbar and drag it to select the area to be included in the new group.

No matter which option you choose, the **Activity** dialog box appears.

1. Type a group **Name**. Whatever appears in this field will be displayed on the FuegoBPM Studio workspace. This field accepts blank spaces and is not limited.
2. Click **Localize** to include the group name in German, Spanish or Portuguese. See Localization for further information.
3. Optionally, type a group **Description**.

Activity: Group1

Category

- Activity Id
- General
- Advanced
- Image

Activity Id

Activity Id Group1

Name


Group1 Localize

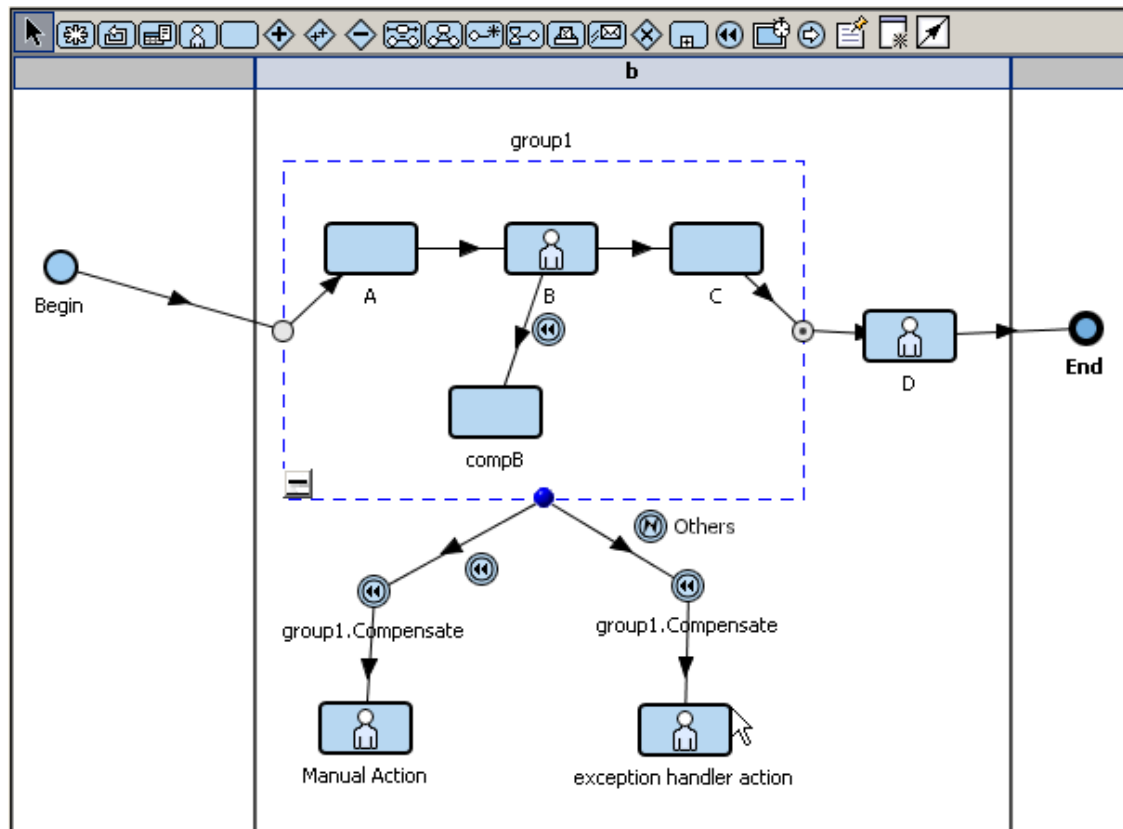
Description

Group1 Localize

OK Cancel Help

Note

 The Begin and End activities within a group are shown by default on the left and right side of the box. You can set them to be on the top and bottom by set the corresponding preference. From the **View** menu, select the **Groups** option. Select the desired **Begin and End Orientation**.



You can **Collapse** or **Expand** the group by double clicking in the group workspace.

Properties

General Category

Atomic Groups

A Group can be flagged as **Atomic** in order to be executed as a single transaction. An atomic group is defined as a group of **automatic activities** all executed in **one transaction**. For a group to become atomic, all its Sub-Groups must be flagged as atomic. An atomic group cannot include Interactive activities or external notifications.

Activities inside an atomic group can belong to different roles.

An atomic group can contain ONLY:

- Automatic activities.
- Split/Join activities.
- SplitN/Join activities.
- Notification/Wait activities (only synchronization between copies.)

If any other activity (such as Interactive) is dropped into a group, you will be warned about it and the action will be automatically undone.

An atomic group can contain other atomic groups. However, it cannot contain non-atomic groups.

If a non-atomic group is dropped into an atomic group, you will be warned about it and the action will be undone.

Atomic groups cannot handle exceptions or compensations within the group. Exceptions that occur inside the atomic group have to be handled by the group exception flow or any outer group.

Advanced Properties

Generate Events: See Generate Events. If the group does not generate events, it will not be listed in the Audit Trail, but the activities within it, if they do generate events, will appear.

Collapse/Expand group

Collapsed mode: The group can be represented within the graphical design as a simple icon, even though it contains several activities/groups.

Expanded mode: The group is fully displayed with all the activities/groups it contains. When expanded, the group has a visual representation of its start and end point (like a Begin and End.) These icons are placed on **the edge** of the rectangle that represents the group. The unique incoming transition to the group reaches the

begin point and the unique outgoing transitions of the group leaves from the end point. Double-click on the group to switch between both modes.

Groups and Transitions

Due Transitions

A due transition can be specified for "leaf" activities as well as for "group" activities.

A group can have a due transition that applies to the instance in any activity within the group. In such case, due time will begin running as soon as the instance arrives at the first activity within the group (entry point).

To define which calendar rule to apply, the server looks for the organizational unit where the process of the instance being executed has been deployed. If no calendar rule is defined for the organizational unit, the server keeps looking in the upper levels of the organizational hierarchy until it finds a calendar rule to apply. If no calendar rule is found, it is assumed that all days are working days. If FuegoBPM Enterprise is installed, FuegoBPM Server will also take into account the calendar rule set for the organizational unit where the process is deployed and the activity role where the instance is running. The calendar rule set at role level is first evaluated by FuegoBPM Server and overrides rules defined for the organizational unit, if any are defined. Refer to Calendar Rules for further information.

Group due transitions override any single activity's (within the group) due transition. If a group has a due transition and it exists inside another group that has a due transition, the instance is ruled by the outermost group's due transition if both transitions expire at the same time. For example, between an activity due transition and the group that contains the activity, due transition, the instance will be routed according to the **group due transition**.

When an activity task is being executed over an instance, the task timeout is set to the shortest due interval of:

- The task timeout interval (defined as an server property),
- BP-Method timeout interval that is currently being executed (i.e., the BP-Method that is being executed within the list of BP-Methods that might be connected with relay-to),
- The activity due transition interval,
- Each nesting group due transition (if there are multiple nested groups, all due transition are evaluated),
- The process deadline.

See **DUE TRANSITIONS** for further information.

Compensate Transitions

Each leaf activity can have its own compensate flow. In addition, the group can also have its own compensate flow. The group's compensate transition can send the instance to a leaf activity or to a group activity that will perform any necessary action to compensate the situation. This group represents the compensate flow.

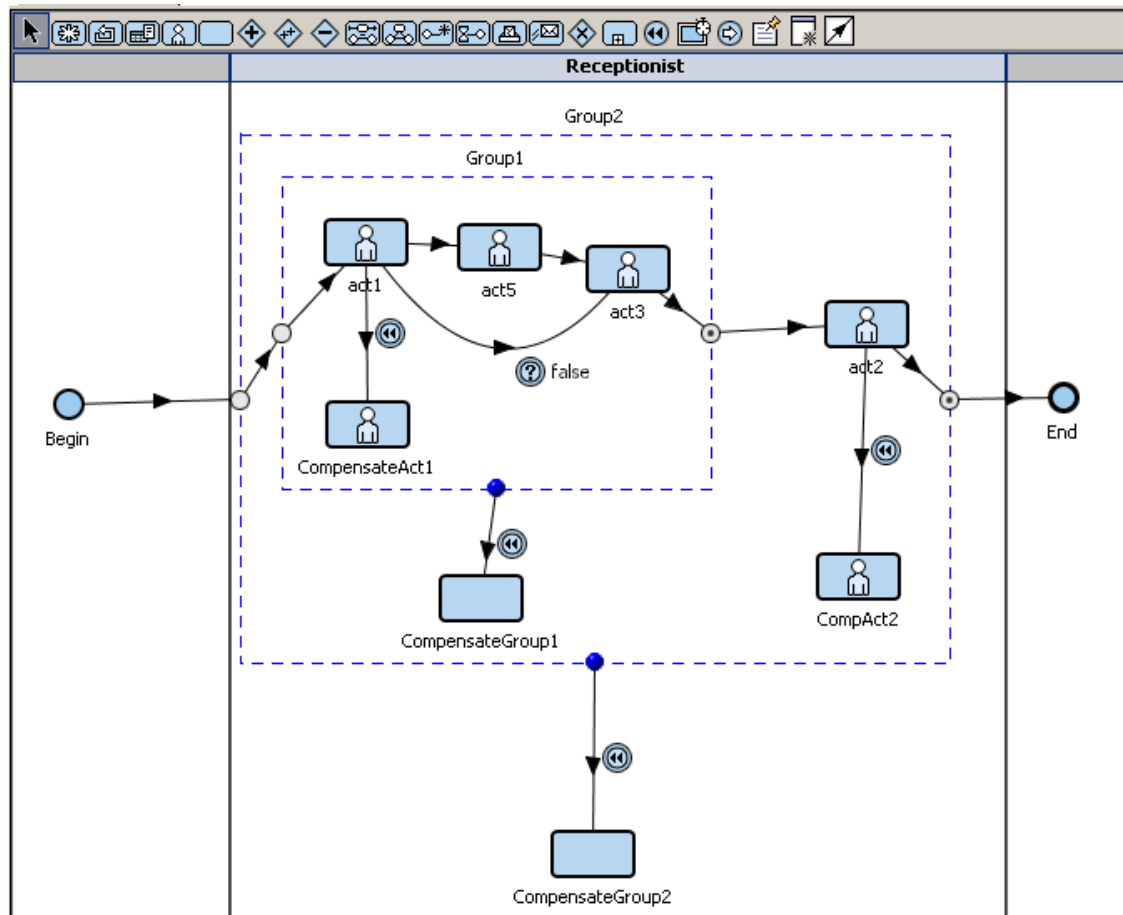
Within the group's compensate flow, you might not reference a specific inner compensation (compensation defined in some inner activity). Therefore, the server will call all the compensations defined in the group in the inverted order in which they are executed. If it has reference to an inner compensation, the server will call only this compensation.

In the following example, *Group 1* is an inner group of *Group2*. Moreover, Group1 has activities *act1*, *act3* and *act5* within it.

- If *act1* needs to be compensated, the *CompensateAct1* activity will be executed.
- If *act3* needs to be compensated, as there is no compensate transition going out of this activity, **no** compensation is

performed.

- If no compensate transition is defined for a Group, then all inner activities compensations are performed. They are performed in reverse order of the process flow. In the example, if there were no compensate transition for *Group2*, inner compensations are executed: *CompAct2* and then *CompensateGroup1* in this order.



A **group** can also be defined by grouping activities inside the **compensate flow** or **exception flow**. In this case, no exit point for the group will exist.

See **COMPENSATE TRANSITIONS** for further information.

Exception Transitions

Groups can call and manage their own exception flow (in the same way as a Process does). See **EXCEPTION TRANSITIONS** for further information.

Rollback Techniques

If you want to rollback the actions in a group, you should call an exception (which can be a localized exception or the process-wide exception) and then use the compensation activity.

FuegoBPM Studio and the Standards

FuegoBPM Studio's standards match the concept of **Scopes** in BPEL.

Groups and Grab activities

See Groups and Grab activities.

Groups Examples

Groups Examples.

Groups and Grab activities

An instance can be grabbed from any activity, regardless of whether the activity is **grouped** or not. While grabbed, the instance will remain as **grabbed** in the grab activity.

In other words, when:

- A Grab activity's task is executed over the instance, or
- The instance is ungrabbed, or
- The instance is sent to an activity different from the source activity, the source activity group's property will rule the operation (i.e., due transition expiration, exception handling, compensation handling).

The source activity's due time is removed when the instance is grabbed. On the other hand, the process instance deadline is kept active. Moreover, the due time for groups nesting the source activity are also kept active (in the event that the activity is within a group that is actually within another group).

When a grabbed instance is ungrabbed, from that moment on, the instance can be processed as usual in the source activity (activity from which it was grabbed).

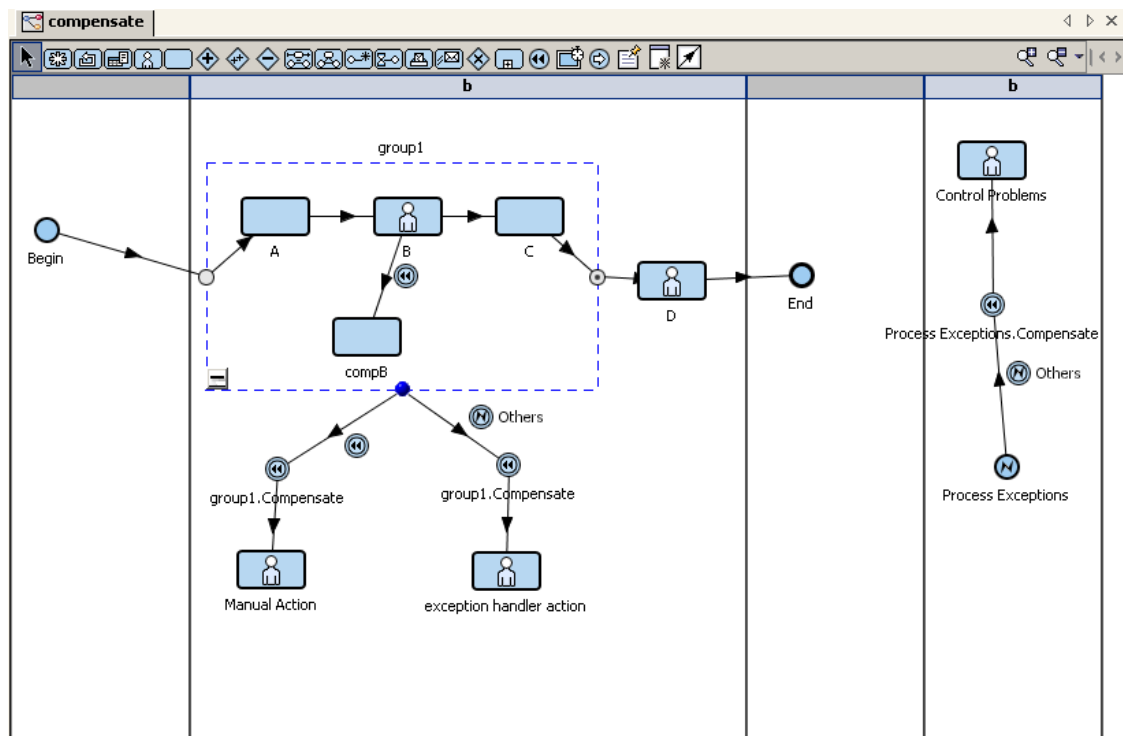
Once an instance has been grabbed, it can be sent to an activity different from the source activity **BUT ONLY WITHIN THE SAME GROUP**.

Groups Examples

Understanding the Behavior of Groups

Case 1: Groups and Compensation

In the following example, **group 1** has a defined **Compensation handling Flow** as well as an **Exception handling Flow**.



When will group1's compensation handling flow be executed?

This will happen if *activity D* fails with an exception.

As *activity D* fails with an exception, the instance is submitted to the **Process Exception Handler Flow**. The first action in this flow is to compensate all its inner groups' **Compensation Handler flows**. Therefore, *group1*'s **compensation handling flow** (defined by its compensation transition) is executed beginning with the **Compensate** activity that will trigger all inner compensation, in this case the *CompB* activity. After that, the *Manual Action* activity will take place.

Case 2: Groups and Exceptions

In the example above, if an exception occurs within *group1*, it is handled by its **Exception handler flow** (guided by the Exception transition). For example, if an exception occurs in *activity C*, the first activity to execute is *group1.Compensate*, which will trigger activity *CompB* as the compensation for *activityB* (inner activity). Next, it will












return to the exception flow and execute *exception handler action* activity. At this point, the instance has reached the end of the exception flow and it is submitted to the next activity after the group. Therefore, *activity D* is executed.

Chapter 8. Transitions

Transitions

A transition is the bridge between two activities. Transitions use directional arrows that display the direction of the flow. An instance flows through a process by following the logic that applies to a transition.

The types of transitions are as follows:

- Unconditional: Instances flow through the transition unconditionally.
- Conditional  / : Instances flow through the transition if certain conditions are met.
- Due  /  /  : Instances flow through the transition according to time conditions.
- Compensate  /  : Instances flow through the transition if compensation processing is required. The actions performed reverse (or undo) any work done in the previous activity in the event that BP-method failure occurs.
- Message Based  /  : Instances flow through the transition if the activity that manages different argument mappings receives a message.
- Exception  /  : Instances flow through the transition if an exception occurs.
- Precedence: Only available in a Split-Join circuit. Copies within a Split-Join circuit can have a synchronization or a precedence. The precedence is represented by a dotted transition.

From any activity (except GlobalCreation, Global and Global Automatic, which have no transitions) you can design any of the previous transitions, as required.

How are Transitions evaluated?

When an activity has multiple outgoing transitions to determine what path is the right transition for the instance to follow, transitions are evaluated as follows:

1. **Message Based transitions:** if the activity received a message and the corresponding message based transition exists (based on the argument mapping name set), this transition determines the instance path. If more than one message to the same message based transition is received, the first one that arrives triggers the instance execution. The rest of the messages are ignored.
2. **Conditional transition:** If there are no message based transitions for an argument mapping name, the conditional transitions are checked.
3. Those messages that do not match #1 or #2 follow the **Unconditional** transition.

Creating a transition


To add a Transition you can either:

1. Right-click on the source activity.
2. Select **the type of Transition that is required** from the shortcut menu.
3. Draw the line to the target activity. The **Transition Properties** dialog box appears.

Or

1. Select the *Transition* icon from the toolbar and drag it to the source activity.
2. Draw the line to the target activity.
3. Select **the type of Transition** that is required. The **Transition Properties** dialog box appears.

Note

 If the **Transition Properties** dialog box does not appear, right-click on the transition line and select **Properties** from the shortcut menu. To change your preferences, select **File** and then **Preferences** from the Process Designer menu options. The **Preferences** dialog box appears. Select the **Transitions** category and select the **Show properties automatically when adding a new object** check box. Next time you add a transition, the **Transition Properties** dialog box appears.

- Complete the **Properties** tab information depending on the type of transition.
- Select the **Description** tab to add the **Name** and **Description**. These two properties can be displayed as the transition label in the Designer Workspace.

Moving a transition line

To **move** a transition you can:

- Right-click on the transition. The **From** and **To** options move the transition line to alternate activities in the process. Change the receiving activity, the source activity or both.

- **From:** Select the new source activity from the shortcut menu.
- **To:** Select the new receiving activity from the shortcut menu.

Or

- Select the transition, drag any of the edges of the transition and drop it on the new source or target activity.

Curving a transition

The **Curve** option changes the trajectory of a transition line. This option makes it easier to fit activities and transitions on the screen.

- **Curve:** Select it and the default curve appears. Left-click on the transition line, hold the left button down and drag (stretch or shrink) the curve to the desired arc. Release the mouse button and the transition is curved.

Removing a transition


To remove a transition, right-click on the transition and select **Cut** from the shortcut menu.

Setting the transition preferences



Select the **Preferences** option from the **File** menu. Within the **Transition** category, you can choose to set preferences on how transitions are shown on the Designer workspace.

- **Show properties automatically when adding a new object:**

If selected, each time you add a transition, the properties dialog will automatically display.



- **Visual properties:** You can determine what label is shown near the transition on the Designer workspace: the transition **Condition**, the **Name**, the **Description**, the **Name and Description**, the **Condition and Description**, or **None**. If you select **Description** or **Name and Description** to be shown, the  icon appears next to the transition. Click on it to display the full description.

Conditional Transition

When your process requires restricted workflow between two activities, you should add a **Conditional transition**  / . A Conditional transition indicates that workflow will only occur if specified conditions are met. The special conditions are added by using a BP-Method in the **Condition** field in the **Transition Properties** dialog box.


For example, in an Order Management process, a conditional transition directs instances from the *Review Order* activity to the *Special Care* activity if the order status is equal to "Expedite" or "Alert". The BP-Method for this condition is as follows:

```
orderStatus in ["Expedite", "Alert"]
```

After you create the transition, a line with a directional arrow connects the two activities on the design workspace. The  /  icon next to the transition indicates that it is a **Conditional transition**.

As well, the transition's **Name**, **Description** or **Condition** may appear next to it depending on the chosen preference.

Note

 If you do not want to see the **Conditional transitions**, disable the **Show Conditional Transition** property from the **View** menu, **Transitions** option.

Rules

The rules for using conditional transitions are as follows:

- Conditional transitions are available for most activities, with the exception of End and Split-N. (Global, Global Creation and Global Automatic do not require transitions.)
- Activities cannot have an unconditional and a conditional transition to the same destination activity at the same time.

Defining the Condition

The **Condition** is defined in the Transition Properties dialog box by typing a BP-Method in the **Condition** field. By default the transition's **name** is used as the expected **result** of the condition.

For example if the transition represents the normal flow then you can **name** it as **OK** and the condition is automatically built as **result == "OK"**. This is the default condition and is valid while the condition is empty.


If you manually define a condition then the default condition is no longer valid.

Instance variables can be used in the condition as well.

More than one conditional transition might be required. Multiple conditional transitions flowing out of an activity are ranked in order


to determine the evaluation precedence. The precedence is assigned in ascending order according to the creation order given when the process is first designed in FuegoBPM Studio. Nevertheless, the conditional transitions' order can be changed by using the **Conditional transitions order** properties window that FuegoBPM Studio displays in the activity shortcut menu (right-click on the activity) when more than one conditional transition has been defined.

Tip

 it is highly recommended to **name** the condition to represent its condition. Furthermore, use the predefined variable **result** to set the result of it.

- The syntax is automatically checked. If everything is correct, a **blue flag** appears on the upper-right corner of the **Conditional** field. If something is incorrect, a **red flag** appears. Drag the mouse over the red line below the red flag and the error displays. In addition, the error is shown at the bottom of the dialog box if you click on the statement that has the problem.
- Click **Ok** to close the Transition Properties dialog box. When the dialog box closes, the last check is performed. If something was not corrected, the error message will display.

Note

 It is not recommended to use a variable type ANY because, in order to compare it, you will have to cast it before. If this is not done, the comparison might fail.


See Fuego Business Language Programming Guide for further information.

Unconditional Transition

When your process requires unrestricted workflow between two activities, you should add an **Unconditional Transition**. This type of transition indicates that no conditions exist to prevent instances from moving to the next activity. Therefore, the transition occurs unconditionally.

After you create the transition, a line with a directional arrow connects the two activities on the design workspace. No icon is displayed next to the transition.

Note

 If you do not want to see the **Unconditional transitions**, disable the **Show Unconditional Transitions** property from the **View** menu, **Transitions** option.




Rules

The following rules apply to unconditional transitions:


- Each activity must have at least one **outgoing unconditional transition**. Exceptions to this rule are the following:
 - if the activity has a **due transition**. The **Split and Split-N activities** are the only ones that **must** have an unconditional transition, as an exception to this rule.
 - Global Creation, Global, Global Automatic and End activities have no outgoing transitions.
- Each activity must have only one outgoing unconditional transition. Exceptions to this rule are:

- Split activities can have more than one outgoing unconditional transition.
 - Interactive activities may have more than one outgoing unconditional transition if the **User selects transition** check box is selected on the Interactive's **Activity Property** dialog box.
-
- Activities cannot have an unconditional and a conditional transition to the same destination activity.



Due Transition


A **due transition**  /  /  is used when a process requires an instance to move to the next activity within a specified time period. Due transitions are used to implement deadline processing.

Note

 Activities can use only one due transition to link to another activity.

After you create the transition, a line with a directional arrow connects the two activities on the design workspace.

The  /  icon next to the transition indicates that it is an **Interval expression or Interval constant due transition**. The due condition is displayed as well.

The  icon next to the transition indicates that it is a **Scheduled based due transition**.

Note

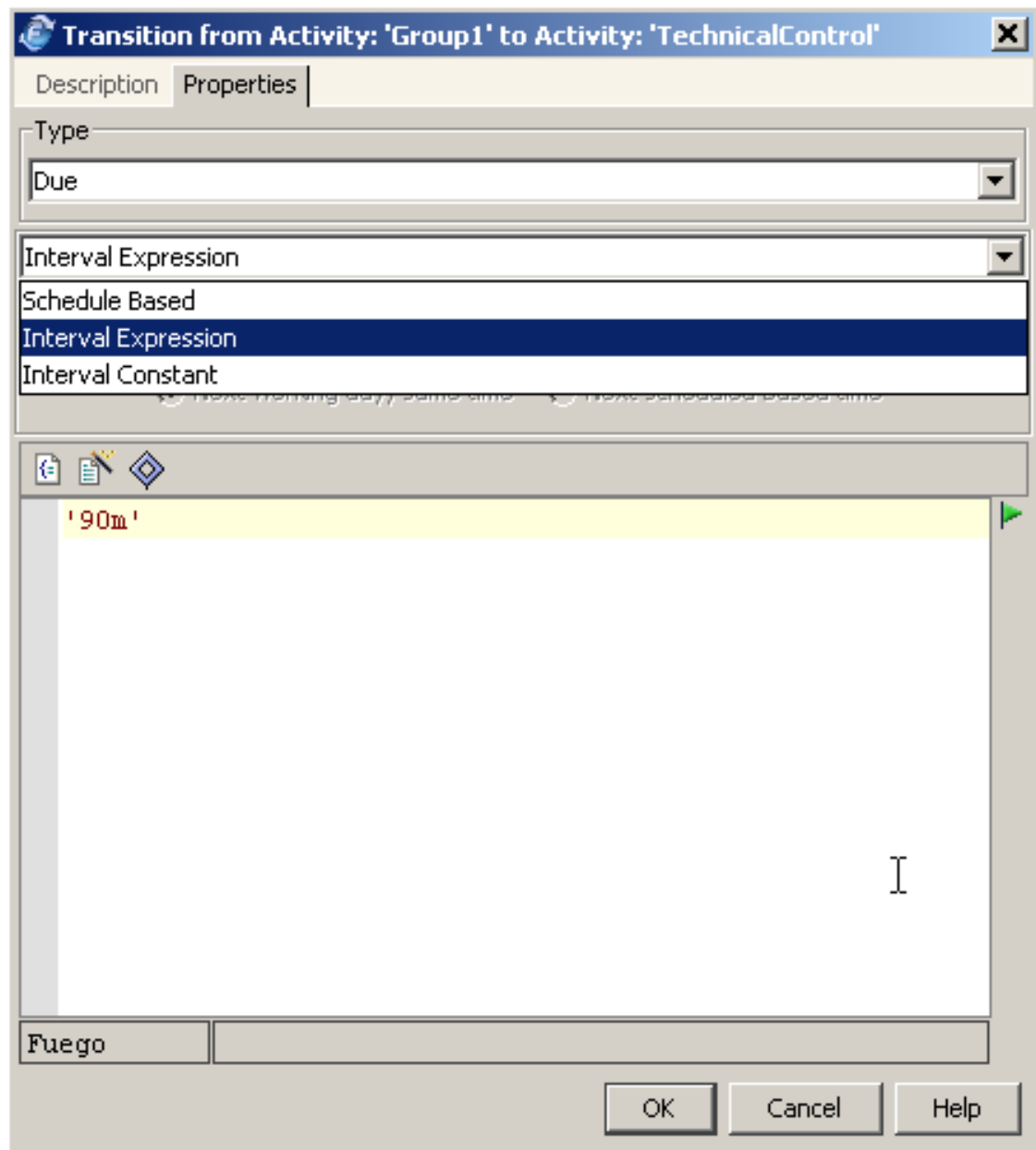
 If you do not want to see the **Due transitions** disable the **Show Due**

Transitions property from the **View** menu, **Transitions** option.

Defining the Due Condition

The **Due Condition** can be defined as **Schedule based**, **Interval expression** or **Interval constant**.

In any of these cases, you can decide to use Calendar Rules. So, if the calculated due date is not a working day, the applicable due date is moved to a valid date.



Schedule based

If the next scheduled due time falls on a non-working day, you have the ability to select whether the due time is passed to the **Next working day, same time** or **Next scheduled based time**.

For example, if you set the due time to happen once a week, every Monday at 11:00 AM and one Monday is a holiday, then if you have

selected:

Next working day, same time the due time is set to Tuesday at 11:00 AM, or

Next scheduled based time the due time is set to next Monday at 11:00 AM.

The Scheduled option indicates that the due transition is set to run on a specific date at a specific time.

The **DAILY** choice allows you to select a time you want the due time to apply each day. For example, every day at 3:00 AM.

Transition from Activity: 'Automatic2' to Activity: 'DueControl'

Description Properties

Type

Due

Schedule Based

☒ Use calendar rules

☒ Next working day, same time ☐ Next scheduled based time

Daily Weekly Monthly

When

3:00 AM

OK Cancel Help

The **WEEKLY** choice allows you to select a "**day of the week**" and a time that the due time applies every week. For example, every *Monday* at " **10:18 A.M.**"

Transition from Activity: 'Automatic3' to Activity: 'DueControl'

Description Properties

Type
Due

Schedule Based

☒ Use calendar rules

☒ Next working day, same time ☐ Next scheduled based time

Daily Weekly Monthly

Day	When
Monday	10:18 AM

OK Cancel Help

The **MONTHLY** choice allows you to select the **month**, the **week of the month** (First-Second/Fourth-Last)/the **day of the week** (Sun-Sat) or **the day of the month and the day** (1-31) and the **time** to which the due time applies. For example, you can choose the third (3rd) Thursday of every month at 3:45 P.M.

Transition from Activity: 'Automatic1' to Activity: 'DueControl'

Description Properties

Type
Due

Schedule Based

☒ Use calendar rules


☒ Next working day, same time ☐ Next scheduled based time

Daily Weekly Monthly

Month	Week	Day	When
All	Third	Thursday	3:45 PM

OK Cancel Help

Note

 Notice that the due is moved based on the day and not on the time as explained on the dialog box

Interval Expression

The Interval Expression option allows the due time to begin the

moment the instance arrives at the activity plus an interval time. The interval time to add to the arriving time can be the result of an expression.

If the calendar rules apply and the calculated due time is on a non-working day, the due time is postponed to the first working time.

- In the **Due Interval** field, type a time interval, making sure that you surround the time interval with a single quote. (See below for time syntax examples.) For example, if you require a time interval of five minutes, you need to type '5m'. This means that the instance has five minutes (5m) to flow through the Due transition from the source activity to the next activity. To implement this example, you can also define the Due transition as an Interval constant (see below for more information).

If you require a more complex condition, add a BP-Method in the **Due Interval** field to force an instance through the process in a specified amount of time under certain conditions. In the following example, the BP-Method indicates that an order greater than \$5000 can sit in the Account Manager's queue for only 2 minutes ('2m'). If the order is less than \$5000, the order can remain in the queue for 5 hours ('5h').

```
(OrderAmount > 5000) ? '2m':'5h'
```

- The syntax is automatically checked. If everything is correct, a **green flag** appears on the top right corner of the **Conditional** field. If something is wrong, a **red flag** appears. The error is shown at the bottom of the dialog box if you click on the statement that contains the problem.

Note

Time will start running immediately after the instance reaches the source activity of the due transition.

Time syntax

Due interval logic to indicate an interval of time is added to the due transition properties. The syntax is:

- *d* for day
- *h* for hour
- *m* for minute
- *s* for seconds

Examples of valid intervals to enter in Due transition logic include:

- '3d ' for three days
- '1h' for one hour
- '4m' for four minutes
- '2m30s' for two minutes and 30 seconds.

See METHODS REFERENCE GUIDE for further information on time and interval syntax.

Interval Constant

The Interval Constant option allows the due time to begin the moment the instance arrives at the activity plus a defined time (number of months/days/hours/minutes/seconds).

If calendar rules apply and the calculated due time is on a non-working day, the due time is postponed to the first working time.

For example, you decide that the due time is always the instance arrival time plus 7 days and 12 hours.

No expressions are available. If so, define the due time as an Interval expression.

Note

Time will start running immediately after the instance reaches the source activity of the due transition.

Priorities for the Due times

A due transition can be specified for leaf activities as well as for group activities. Therefore, an instance can be affected by multiple due times as follows:

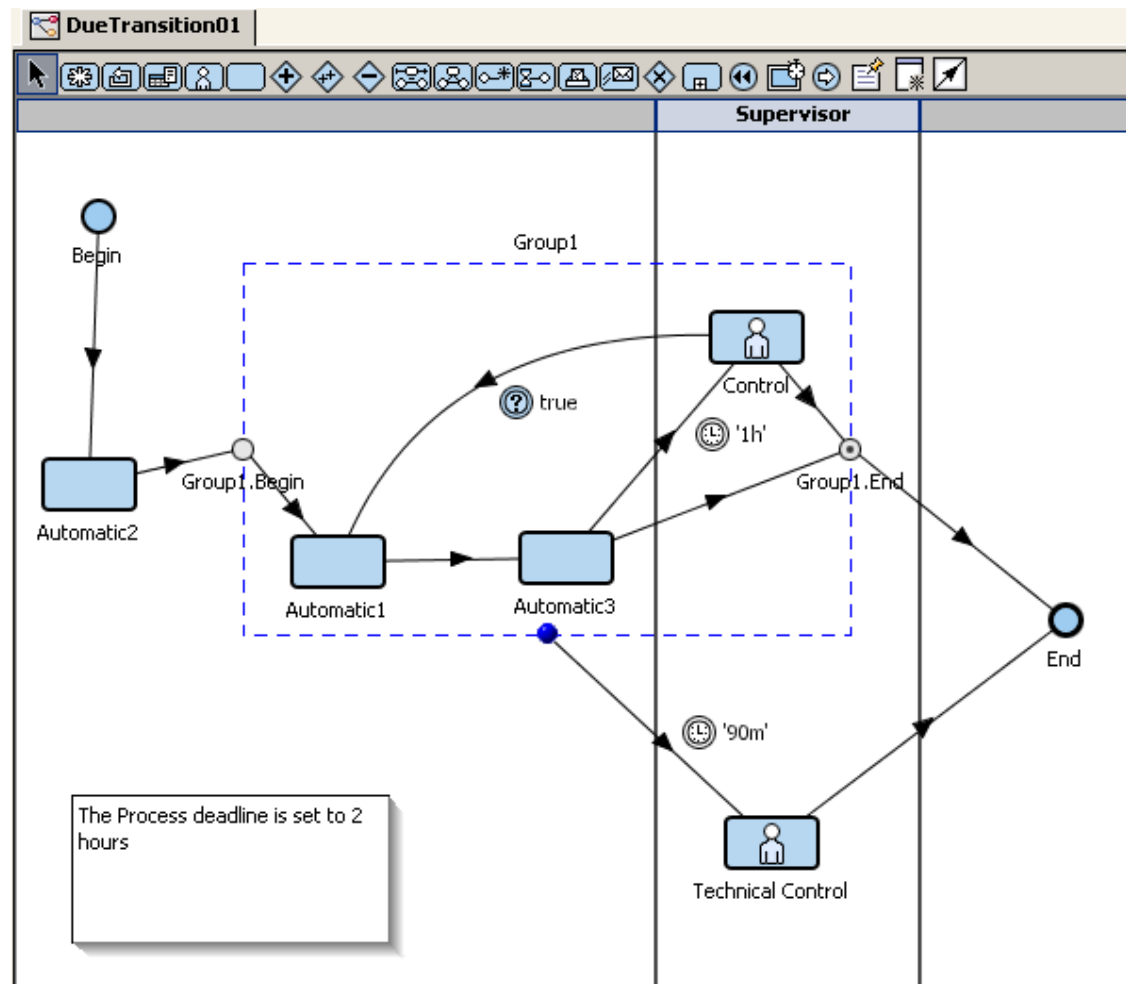
- the activity due transition interval,
- each nesting group due transition interval,
- the process instance deadline.

Thus, when an instance arrives at an activity, these three options are considered in order to determine the shortest due time and which due transition is first priority.

You should consider that a group's due transition time begins when the instance arrives at the first activity within that group.

The due time assigned to an instance, as it arrives at an activity, is the shortest time of all the three possible due time options listed above.

For example, as shown in the image below,



when the instance arrives at the activity **Automatic3**, as this activity has a due transition, the due time is the shortest time between:

- **1 hour:** defined in the *Automatic3* due transition.
- the remaining time of **90 minutes** as defined for the *Group1* due transition. This means that if the instance has remained for **50** minutes in the *Automatic1* activity, the remaining time for the group is **40** minutes. In consequence, the instance will probably flow through the *Group1* due transition to the *Technical Control*

activity instead of flowing to the *Control* activity (**1 hour**).

- the remaining time of **2 hours** defined as the process due deadline. If the instance has been in the *Automatic2* activity for **100** minutes, the remaining time for the process is **20** minutes. Thus, the instance will probably flow to the *End* activity instead of flowing to the the *Technical Control* activity (**90 minutes**) or to the *Control* activity (**1 hour**).

If more than one due transition expires at the exact same time, the instance is sent through the outermost due transition (of those transitions which time interval expired at that moment).

For example, the instance is within an **activity** that has a due transition that expires at noon. Additionally, this activity belongs to a **group** that has also a due transition that expires at noon. At noon the instance will flow through the **group** due transition.

When a due interval is calculated for an activity (leaf or group), the calendar rules are taken into account as regards the role where the instance is or whether the group spreads over many roles.

Task timeout

If a task is running, the time it has to complete depends on the task timeout, defined with the predefined variable **timeout** or 5 minutes by default. But the due time explained above also applies. Therefore, if there is a task running for a specific instance and any of the due time at activity, group or process level expires, although the task might have some remaining time, the task will finish and the instance will flow through the corresponding due transition.

Moreover, if the task timeout was set with an interval greater than specified as the Server property "Maximum task timeout", the task will fail until the process or maximum task timeout is fixed.

Continuing with the example image above, the *Automatic1* activity has a task that has set a timeout to **30 minutes**. And the instance has



been in the *Automatic2* activity for **100** minutes, the remaining time for the process is **20** minutes. So, although the task theoretically has 30 minutes to execute, after **20 minutes** the process due deadline expires, the task is terminated and the instance flows to the End activity.

Due time calculation failures


If the due time calculation fails, the transaction corresponding to the activity from which the instance came from, fails.



For example, if you SEND an instance from an *Interactive* activity to an automatic activity that has a due transition but its calculation fails, the previous *Interactive* activity execution fails.

Exception Transition

An **exception transition**  /  is used to submit the instance to an Exception Handler flow when the activity or group of activities fail or throw an exception.

Note

 Activities or groups can use only one exception transition to link to the exception flow. The exception flow can have as many activities as required to fix the exception condition.

After you create the transition, a line with a directional arrow connects the two activities on the design workspace. The  /  icon next to the transition indicates that it is an **Exception transition**.

The source of the transition is the activity or group where the exception occurred, and the target is the first activity in the exception handler flow. As the exception handler flow is independent from the main process flow, it cannot have transitions back to the main process flow.

Exception holder variable: you can create and define an **instance**

variable as the **Exception holder variable**. If you do not define one, FuegoBPM Studio creates it automatically.

Once the exception occurs it is stored in this variable and it is available within the exception flow that is handling the exception. It can be used by the developer to debug or analyze the exception in depth.

You can create only one variable and re-use it in all the exception transitions or you can associate different instance variables to each exception flow. Normally the variable type matches the exception type.



If there is no variable defined (backward compatibility), a default *exceptionHandler* instance variable (Any type) is created when you check the design.

At runtime if the variable receives an unmatching type, then a warning is logged.


The variable content is available only within the Exception flow to where the instance is routed through the exception transition. Therefore the exception contained in the variable is not propagated to other processes.

For further information, see Exception Handling.



Compensate Transition

A **compensate transition**  /  is used when an activity or group of activities require that the actions performed by the BP-methods should be reversed. Reversal is needed in case of total or partial BP-method failure, which could be caused by any number of things such as a call to an external system that fails, equipment failure, a database call with bad data and so on.

Note

 Activities or groups can use only one compensate transition to link to the compensate flow. The compensate flow can have as many activities as



required. However, no other compensate transition between them applies.

After you create the transition, a line with a directional arrow connects the two activities on the design workspace. The  /  icon next to the transition indicates that it is a **Compensate transition**.

The source of the transition is the activity to be compensated and the target is the first activity in the compensation handler flow. Since the compensation handler flow is independent from the main process flow, it cannot have transitions back into the main process.

For further information, see [Compensate Handling and Compensate Activity](#).

Message Based Transition

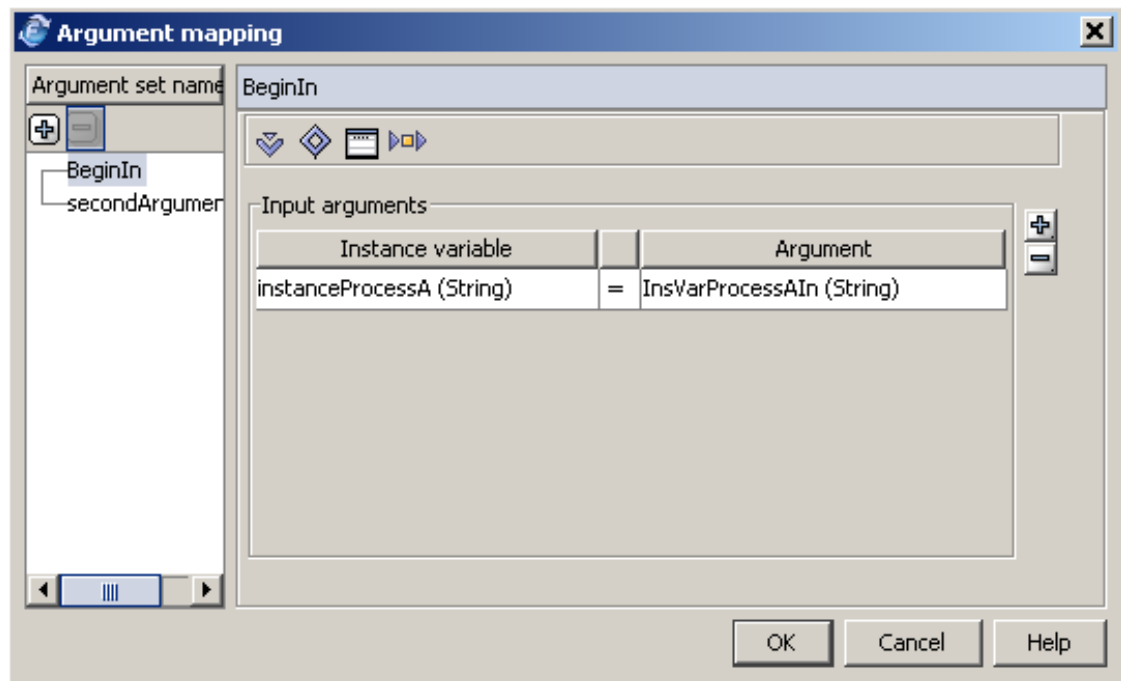
Message Based Transitions are available for the Begin and Notification Wait activity types. A **Message Based Transition**  /  can be added by using the source activity argument mapping sets.

Begin and Notification Wait activities can receive different sets of arguments. Each set has a name defined as the **Argument set name**. Within each set, the arguments can be mapped to instance variables or predefined variables. For any of these mappings, a **new outgoing transition** can be added to the activity. The transition type is called **Message Based Transition**. Basically, you define the transition that the instance will flow through based on the received message.

There cannot be more than one outgoing **message based transition** for the same **Argument set name**.

Example

The following is a process with a Begin activity with two *Argument sets*:



For each argument mapping you can add an outgoing **message based** transition from the Begin activity to another activity.

Right-click on the Begin activity and select **Add Message based Transition**. See Creating a transition for further information.

Transition from Activity: 'Begin' to Activity: 'Automatic1'

Description Properties

Type
Message based

Message selection
Choose one of the argument set names from the source activity of this transition. When this activity receives a message that corresponds to this set, the instance will be routed through this transition.

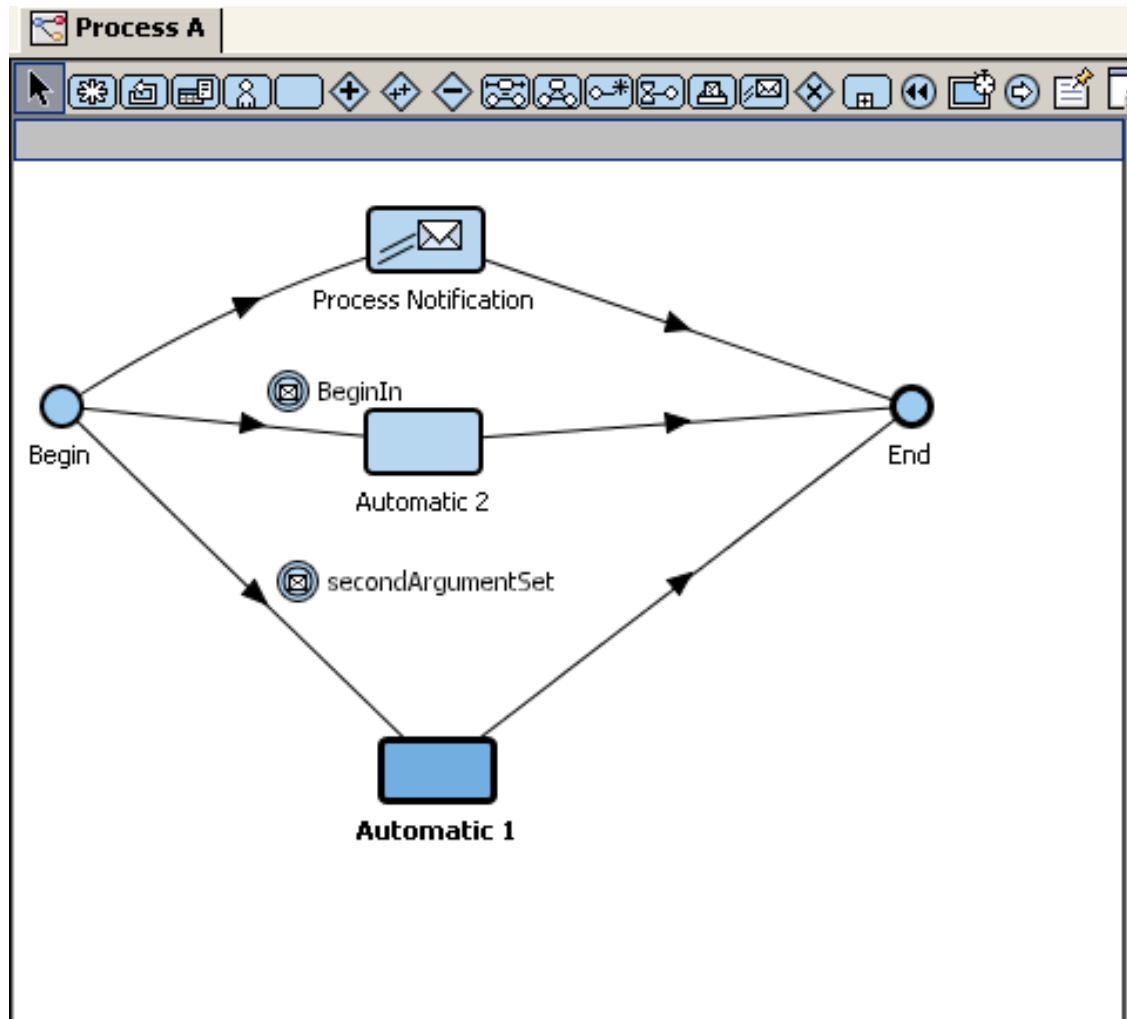
Argument mapping
secondArgumentSet

BeginIn
secondArgumentSet

OK Cancel Help



Select the Argument mappings name that corresponds to this transition. All available argument sets names are displayed, that is to say, all the argument sets that you have defined in the Begin activity and that have not yet been associated to any message based transition.

Now the process appears as follows:



This means that if the Begin activity receives the message called **BeginIn**, it will be routed to the activity alternative1. If the Begin activity receives the message called **secondArgumentSet**, it will be routed to the activity alternative2.

Note

 A line with a directional arrow connects the two activities on the design workspace. The  icon next to the transition indicates that it is a **Message Based transition**. The *Argument Set Name* is shown as well.

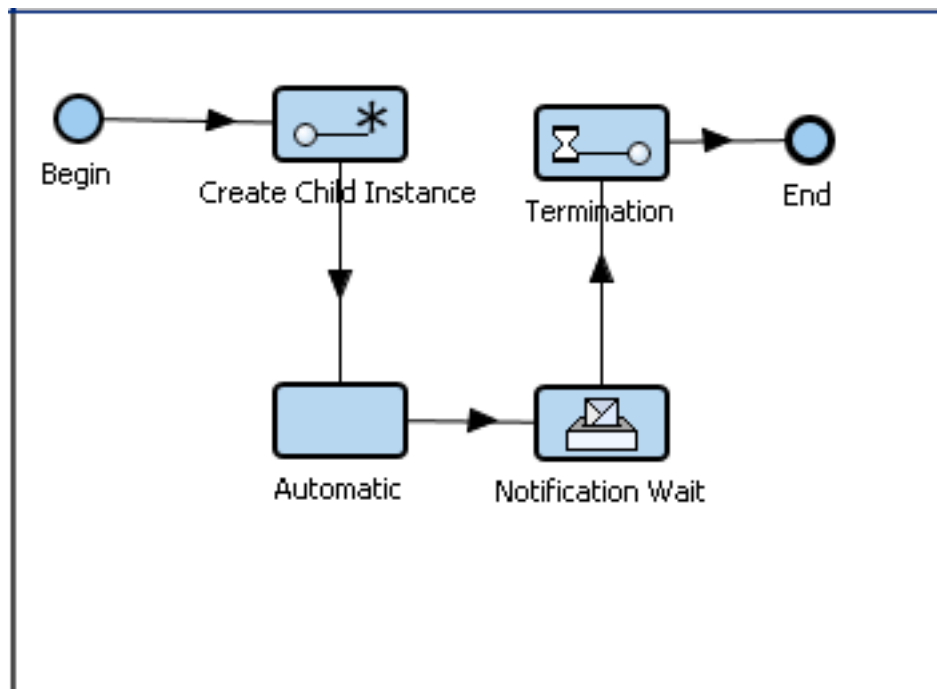
Sending a Message

The activities that create instances (Process Creation, Subflow and

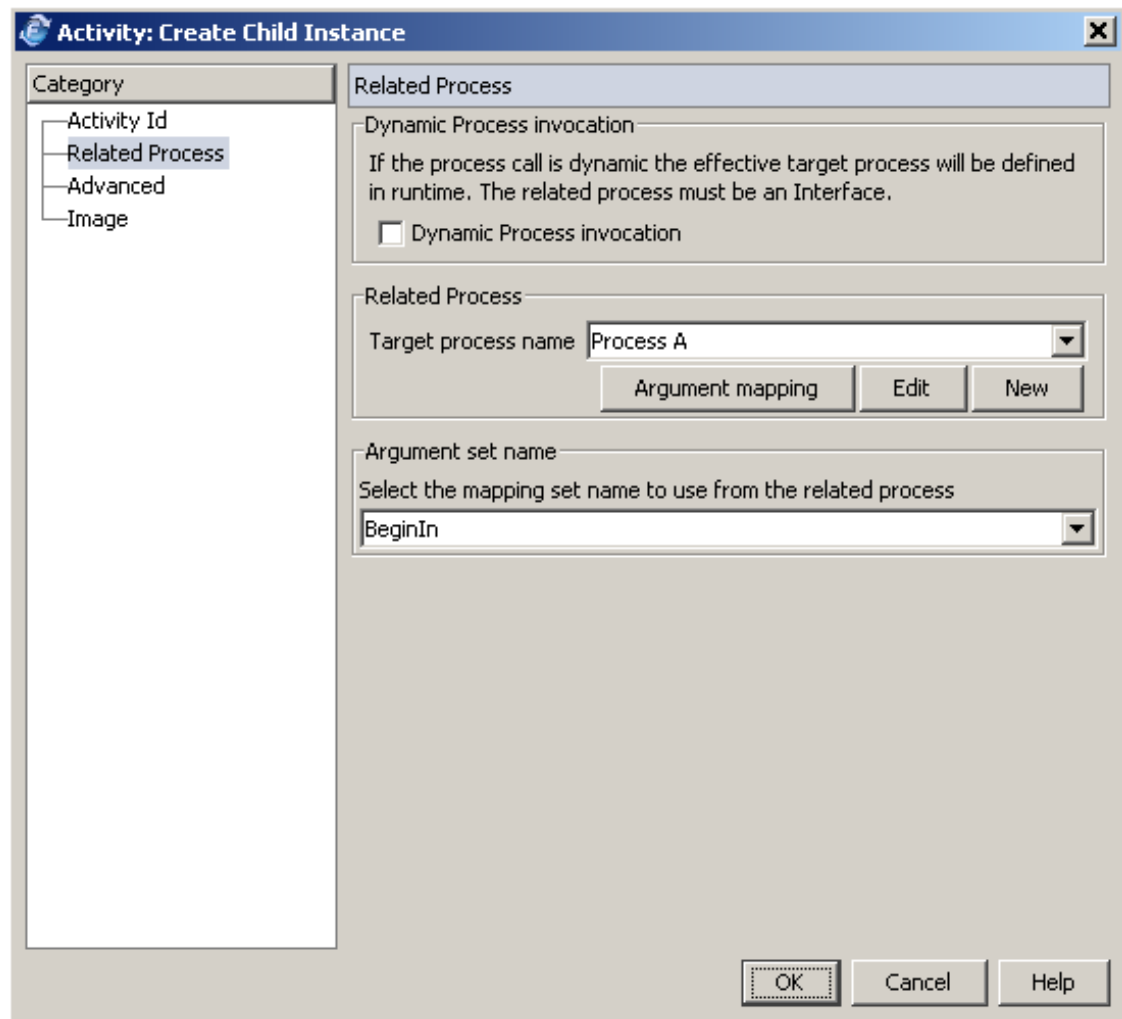
Global Creation) and the one that sends notifications (Process Notification) have a field in the properties dialog that keeps the name of the target **message** (argument set name) that is sent.

According to the selected **message**, these activities will **map** the **set of argument values** with **instance or predefined variables values** and/or **expressions** . (Edit the Instance variable field and add the necessary expressions.)

For example, using the example above, the following process creates instances in the above process:



The Process Creation activity is defined as follows:



The selected Mapping name corresponds to the first set of arguments mapping of the process that is being called (MessageBasedTest).

Chapter 9. Using Variables

Using Variables

Variables are placeholders in memory for values in your process. Variables can be used locally, in the BP-Method of one activity, or universally throughout the process.

Each variable has a name, description, type and value.

There are different types of variables:

1. Instance variables
2. External Variables
3. Business Variables
4. Argument Variables
5. Predefined Variables
6. Local Variables
7. Business Parameters

Which variable should I use?

- **Instance variables** are used throughout the entire process and are passed from one activity to another, and from one task to another.
- **External variables** are process instance variables references. They will be automatically externalized at publishing time.
- **Business variables** are process instance variables references.

They will be automatically externalized at publishing time. Use them when you want to use it as a dimension or measurement to store information into the FuegoBPM Data Store.

- **Argument variables** act as an interface. By using them, you can transform instance variables so that they may be passed to or from external components or between processes.
- **Predefined variables** enable you to set or get information from the process that may affect your instance, as it passes through the process.
- **Local variables** can only be used in the BP-Method task where the variable is created.

Variables can be used in:

1. Task BP-Methods
2. Transition BP-Methods
3. Fuego Object Methods

The type of variable you can use depends on where you are trying to use it.

Adding Variables

Variables can be created from the **Variable tab** that is on the right of FuegoBPM Studio's workspace, set by default.

They can also be created on certain screens where FuegoBPM Studio provides buttons to create them. For example, you can create both Instance and Argument variables while defining an Argument Mapping set.

Depending on the context you are working with, you will be able to

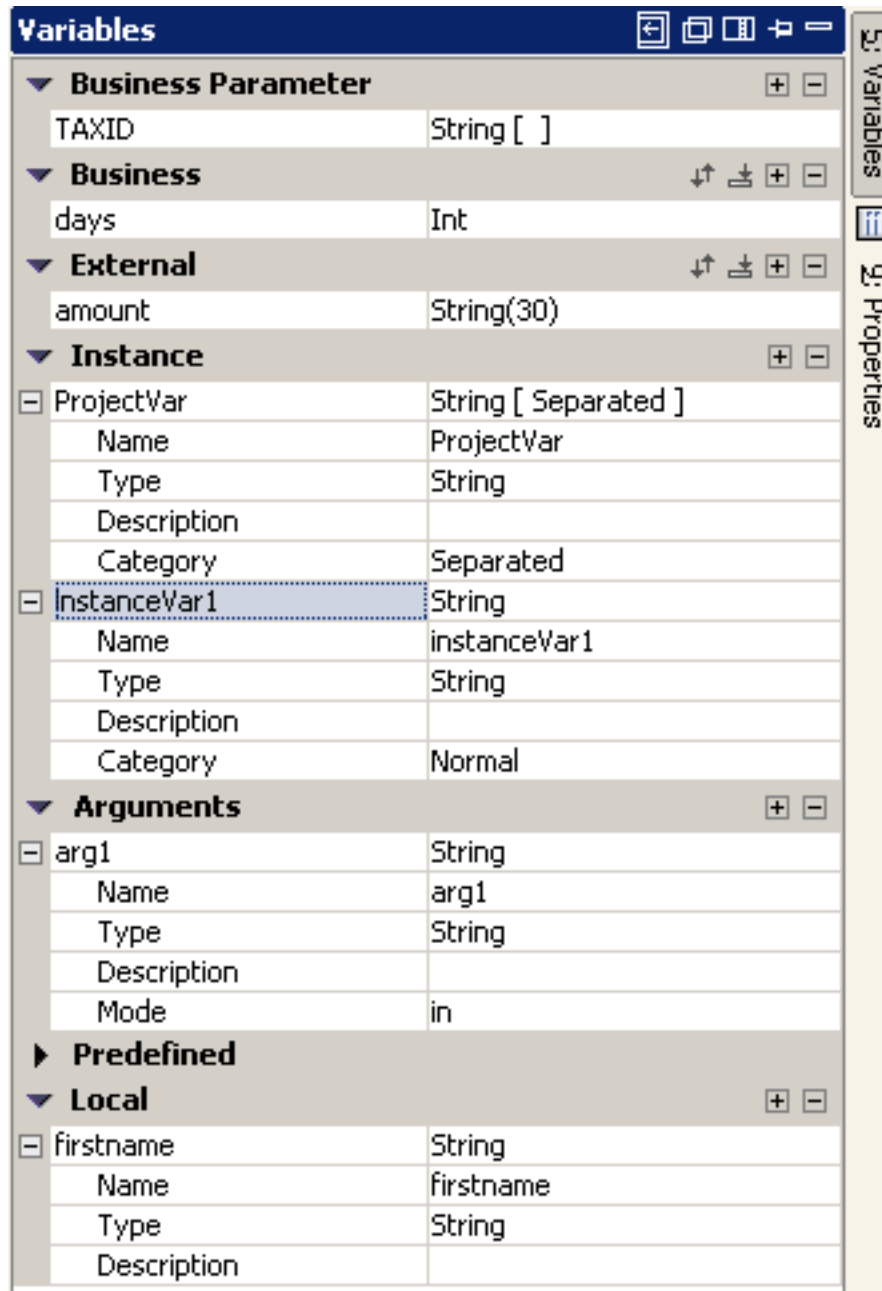
create certain types of variables that will display in the **Variables** frame. The *Business variables* are only displayed when you are designing a process.

To add a variable, first add a new line by clicking the "+" sign on the top-right of the variable's type section.

A new line appears.

For all the types of variables, you have to complete the *Name*, *Type* and *Description*.

1. Type the variable's name in the **Name** field.
2. If you are declaring a simple type variable, select the type of variable from the **Type** drop-down menu. Otherwise, click **Browse** to browse the Component catalog for the type of variable you need. Alternatively, you can also type the variable type in the field.
3. Optionally, type a definition of the variable in the **Description** field.



Project, Instance and Argument variables have some extra properties to be completed. For further information see the variables types documentation.

Removing Variables

From the **Variables** frame, select a variable and click on the "-"

(minus) sign on the right of the variables types section.

Activities and Variables use

Each activity encompasses a specific action. Therefore, some activities accept more kinds of variables than other activities.

Instance Variables

Instance variables are variables attached to an instance. They contain information that refers to the instance. Instance variables can be accessed from the activities as the instance flows through a process from the Begin activity to (and including) the End activity.

All activities, except Global Creation, Global and Global Automatic can access instance variables. Examples of instance variables that may be found in a shipping order management process include: *invoiceNumber*, *customerName*, *customerNumber*, *orderStatus*, *orderAmt* and *shipStatus*.

Instance variables are called in any BP-Method by the variable's name. If there is any possibility for a naming conflict, for example, you have a SQL component and the table has a field with the same name as the instance variable, then you should refer to the instance variable as *this.instancevariablename* within the BP-method.

Instance variables have a special property called **Category**:

1. Normal
2. Separated
3. External

Normal Instance Variables

Most of the Instance variables are defined as *Normal*. Only those that have some special characteristics need to be categorized in a

different way.

Note



All normal instance variables size is limited to 2Mb. The total size of all the normal instance variables cannot exceed 2Mb.

Separated Instance Variables

Instance variables used to store a large amount of data (10 Kb or more) need to be categorized as *Separated*.

FuegoBPM separately stores these variables on the database to avoid performance problems. Large FuegoObjects or serialized Java objects are examples of instance variables defined as *Separated*.

Note



The size of a separated instance variable is limited to 2Mb

Note



In Split-N activities, **separated instance variables** values are not automatically copied to the separated instance variables of the copies. They have to be copied manually in the Split-N BP-Method.

External Instance Variables

Instance variables that contain significant information in a process can be categorized as *External*.

When a variable is defined as **External** , it becomes available as an External Variable across the project.

For example, the instance variable *CustomerName* used in a process within a Project also needs to be used by all other processes across the Project. Therefore, define it as **External** and in each process that needs to use it, declare the same variable (same name and type) and all will be related.

External variables are also available for display in Work Portal, as

they are saved within FuegoBPM Studio's database.

From Work Portal, you can search for instances by certain conditions regarding the external variable.

Variables Initialization

Variables defined as **numeric** (integer, real, decimal, etc) have an initial default value to **0** (zero).

Boolean variables have the initial default value set to **FALSE**.

All other types have the **null** value.

Using Fuego Objects as Instance Variables

See How to use a Fuego Object as an instance variable for more information.

SQL Components as Instance Variables

See SQL Components as Instances variables for more information.

External Variables

External Variables are Instance variables that have been defined as **External** .

Note



These variables are defined to contain information that can be retrieved in the Work Portal. For example, an invoice number, a customer id defined by an external system, and so on.

The external variable is used to save information across processes. For example, when working in the Work Portal, you can search for all instances in any process corresponding to an **invoice number**.

You can add a new **external variable** or define an **instance**


variable as external.

External variables are available for any process within a Project. Each process in a project can define the **same** instance variable and, when defined as *external*, they will all be related.

Once the Instance Variable has been defined as *external*, the **external variable** appears in the *External Variable* section within the **Variable frame**.

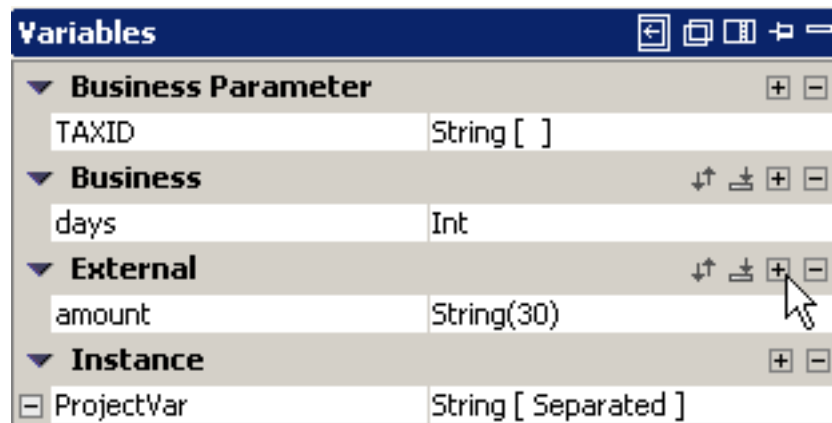
Any change on that variable has to be done on the External variable directly and all instance variables that reference to it are automatically updated.

Note

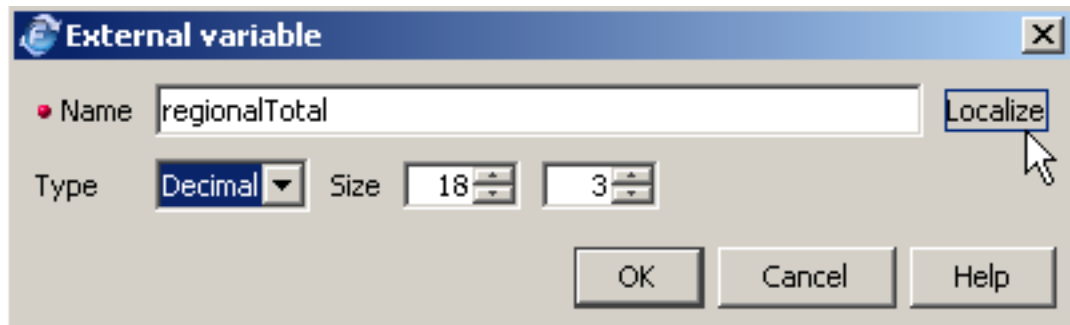
 **IMPORTANT: External Variables** are very special variables and have to be defined thoroughly during the development phase. Once defined and during users testing phase or production phases, the recommendation **is not to change** its type nor use its name if it has previously been used for one purpose and later on you want to save other kind of information. On the other hand the use of external variables should be limited to a short number. Not more than 6 variables is recommended.

Adding and changing an external variable

You can add an external variable from the External section within the Variables tab. Click on the "+" sign:






A dialog box populates:



Define the type and size of the external variable. Optionally, click the Localize button to enter the name in alternate languages. This **label** represents the **name** that is displayed in **Work Portal**. The label can be defined in the different languages when displayed in the Work Portal. See Internationalization for further information on available languages.

Once the variable has been created you can :

- Convert the variable to a Business variable 
- Use the External variable in the process : an instance variable is created in that process defined as external.


To change type or size or localize the external variable, select the variable and click on the type and size. Click on the button  and the properties dialog populates.

Note

 The maximum characters for an external variable name is 16

Business Variables

Note

 **These variables are defined to contain information that can be used to define a measurement or dimension in the Data Store. They represent business indicators considered by the business analyst. They can be retrieved in the Work Portal.**


The business variable is used to save information across processes and use it to generate the Data Store stored information.

See Data Store and Business Activity Monitoring for further information.

You can add a new **business variable** or define an **external/instance variable** as business.

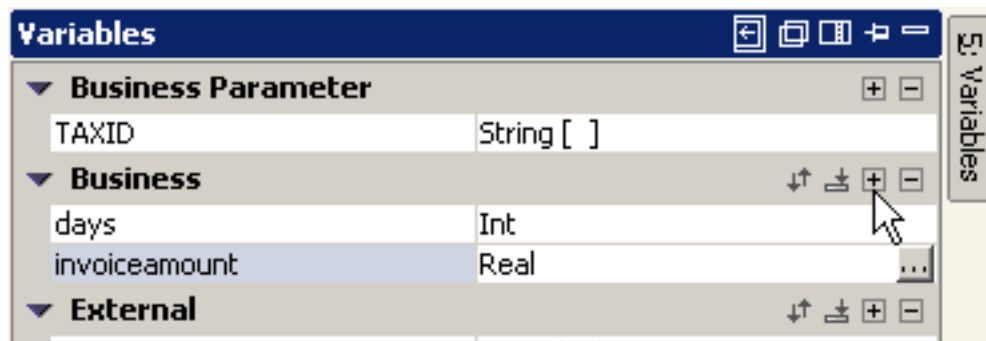
Business variables are available for any process within a Project.

Note

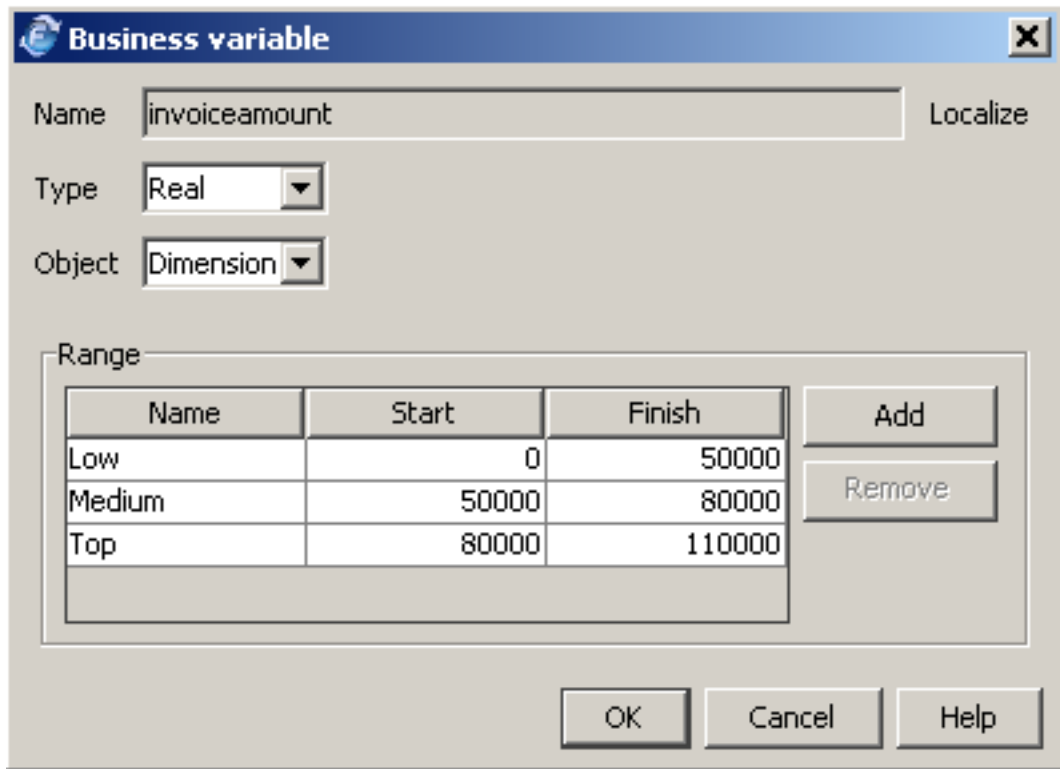
 **IMPORTANT: Business Variables** are very special variables and have to be defined thoroughly during the development phase. Once defined and during users testing phase or production phases, the recommendation **is not to change** its type nor use its name if it has previously been used for one purpose and later on you want to save other kind of information. On the other hand the use of business variables should be limited to a short number. Not more than 6 variables is recommended.

Adding and changing a business variable

You can add a business variable from the Business section within the Variables tab. Click on the "+" sign:



A dialog box opens:



The dialog box is titled "Business variable" and contains the following fields and controls:

- Name:** A text field containing "invoiceamount". To its right is a "Localize" button.
- Type:** A dropdown menu currently showing "Real".
- Object:** A dropdown menu currently showing "Dimension".
- Range:** A section containing a table with three columns: "Name", "Start", and "Finish".

Name	Start	Finish
Low	0	50000
Medium	50000	80000
Top	80000	110000



To the right of the table are two buttons: "Add" and "Remove". At the bottom of the dialog are three buttons: "OK", "Cancel", and "Help".

Define the type and size of the business variable. Optionally, click the Localize button to enter the name in alternate languages. This **label** represents the **name** that is displayed in **Work Portal**. The label can be defined in the different languages when displayed in the Work Portal. See Internationalization for further information on available languages.

If the variable type is *numeric*, that is *Int*, *Decimal* or *Real*, you must define whether the variable is going to be used in the FuegoBPM Data Store as a *dimension* or *measure*. And you must define at least one range of values for it. For example, if you are modeling a business variable that represents the *invoice amount*, you can define ranges like shown above.

If the variable type is *not numeric* (string, time, bool) then it can only be used as a dimension (set as default).


Once the variable has been created you can :

- Use the Business variable in the process : an instance variable is created in that process defined as external.
- If you no longer need the variable as a **business variable** but you need to access it from the Work Portal, then you can convert the variable to an External variable 

Warning



If you have a business variable defined with ranges and convert it to an external variable, the defined ranges are deleted.

To change type or size or localize the business variable, select the variable and click on the type and size. Click on the  button and the properties dialog opens.

Note



The maximum characters for a business variable name is 16

Argument Variables

Argument variables act as an interface. You can transform instance variables with argument variables so that they may be passed to or from external components or between processes. See Argument Mapping for further information on passing information between processes and their calling/called or notifying/notified activities.

Arguments variables can be defined as:

- In
- Out
- In/Out

For each activity, the valid options depend on how arguments can be treated in each case. Therefore, arguments can be in each activity as follows:

- Interactive, Grab, Automatic, Global, Global Creation: *In/Out*
- Subflow: *Out* for the create BP-method, *In* for the wait BP-method.
- Process creation, End, Process Notification: *Out*
- Termination wait, Begin, Join, Notification Wait: *In*
- Split, Split-N, Conditional: N/A

Argument variables can be considered as **External** or as **Internal** based on what they were defined for.

External

The first type of arguments are those defined in the activities (Begin, End and Notification Wait) that can deal with external components of the process. These external components could be an HTML form, a Java program, another process, Web Services, and so on. "External" Argument variables are the interface to a process. They are responsible for passing variables from a process to an external component and receiving variables from external components into the process. Arguments can be grouped in argument sets.

Two FuegoBPM processes that need to communicate with each other use external argument variables to pass the variables between the processes. One process, the parent process, passes variables through a Subflow activity to another process (a subprocess or child process). The child process's Begin activity receives these argument variables and maps them to instance variables. When the child process has finished processing the instance, it passes the instance variables to the End activity, which then maps the instance variables back to argument variables to pass back to the Subflow activity in

the parent process.

Internal

The second type of argument variable is internal to a process. It is available for BP-Methods used within the process (for example, a BP-Methods that receives arguments and can be invoked from another BP-Method assigned to an activity task). In either case, argument variables appear as follows in the Method Editor:

```
arg.myArgument
```


The following sets an argument variable equal to an instance variable as follows:

```
arg.myArgument = myInstance
```

The following sets an instance variable equal to an argument variable as follows:

```
myInstance = arg.myArgument
```

Note

 The usage of the *arg* keyword is optional. However, it is necessary when you need to distinguish a local or instance variable that has the same name as an argument variable.

For example, suppose that there is a method that contains a local variable called *orderNo*. The method also receives an argument called *orderNo*, and the process has an instance variable called *orderNo* as well. In order to distinguish the usage of each variable you will need to qualify it with the corresponding keyword:

To refer to the instance variable you must type:

```
"this.orderNo"
```

To refer to the argument variable you must type:

```
"arg.orderNo"
```

Local variables have no qualification.

"Copy" Argument

There is a predefined argument called *copy* that is only available when writing a **Join** activity BP-Method. *copy* represents the instance-copy that arrived at the *Join* activity.

In the corresponding **Split** of the *Split-Join circuit*, if *copies* are created, they can be processed in the Join activity upon arrival. The way to reference copies is by using the *copy* argument keyword.

copy is used to access the values of the variables of the copies of an instance in the Join activity.

For example,

```
// Set the original instance variable with
// the copy instance variable.
orderTotal = copy.orderTotal
```

See Split-Join activity for further information.

Predefined Variables

Predefined variables are global to all parts of the process and, just as their name implies, they have already been defined. Most of the predefined variables deal with instances as they relate to activities and are used to keep track of an instance and its status as it flows through a process. Some predefined variables are modifiable and others are not. The following table lists the predefined variables included in FuegoBPM Studio.

List of FuegoBPM Studio Predefined Variables

Predefined Variable	Permitted Values	Type	Description
action	Modifiable : OK ; FAIL ; RELEASE ; CANCEL ; REPEAT ; ABORT ; BACK ; SKIP ; NONE	Fuego.Lib.Action	Marks the action to be performed on a process instance as a result of previous BP-Method statements. See below Using the action variable.
activity	Read only	Fuego.Lib.Activity	Returns an object that is the current activity.
activity.deadline	Read only	Time	Set automatically if there is a Due transition going from the current activity (receptionTime + the due transition time interval.)
activity.source	Read only	Fuego.Lib.Activity	The activity from which the current instance came into the current activity. Null in the case of Global and Begin activities.
attachments	Read only	Fuego.Lib.Attachment[]	Array of the instance attachments.
children	Read only	String[ordered Object]	Array of the Ids of the instances that

Predefined Variable	Permitted Values	Type	Description
			are children of this instance, ordered by the list of activities that created them.
creation.participant	Read only.	Participant	Participant that created the process instance.
creation.time	Read only	Time	Creation time of the instance.
currentException	Read only	String	Name of the exception, if the instance is within an exception flow, which was given to it by the throw statement .
deadline	Modifiable.	Time	The instance will expire if it is not completed before the specified time.
description	Modifiable: Any string that does not contain special characters	String	Name of the instance that appears in Web Portal. By default, it is ProcessName+id.number.
id.id	Read only	String	An instance's identifier, which uniquely identifies an instance. It includes the

Predefined Variable	Permitted Values	Type	Description
			deployed process name, organization, organizational unit and the instance Id (including thread id).
id.number	Read only	Int	A number that uniquely identifies an instance within an Server.
id.copy	Read Only	.	The current instance thread number.
notes	Read only	Int	An array of all notes added to an instance.
organization	Read Only	String	Name of organization where the process is running.
organiztionalUnit	Read Only	String	Name of the process' organizational unit where the process is running, such as Marketing or Finance.
parent.id	Read only	String	The Id of the parent instance of the current instance if there is

Predefined Variable	Permitted Values	Type	Description
			one; NULL otherwise. In the case of a Procedure, it contains the id of the calling process
parent.copy	Read only	Int	The parent instance thread number.
parent.number	Read only	Int	A number that uniquely identifies the parent instance within an Server.
participant	Read Only. Any participant	Fuego.Lib.Participant	The human participant stored in directory services that is currently processing the instance.
participant.locale	Read Only	Fuego.Util.Locale	The language, country and variant, if applicable, the participant has set.
participant.next	Modifiable: Any participant	Fuego.Lib.Participant	The participant to which the instance will be sent next. <i>Note: If the instance is grabbed, the</i>

Predefined Variable	Permitted Values	Type	Description
			<i>participant.next</i> is cleared. If the instance is sent BACK or SKIP from an Exception flow or from an Interruption, the only way to reset the <i>participant.next</i> is using the Unselect method from the Participant component. See Participant component documentation for further information.
participant.sticky	Modifiable: Bool	Fuego.Lib.Participant	If set to true, the <i>participant.next</i> is set as the preferred participant. Each time the instance moves to an activity and the <i>participant.next</i> is enabled to work with it (belongs to the role where the activity is), the instance is submitted to the <i>participant.next</i> work queue. In the Split and SplitN activities, all generated

Predefined Variable	Permitted Values	Type	Description
			copies maintain the participant.sticky value from the original instance. In the SplitN method you can change its value. <i>Note: if the participant.next is changed during the process, be sure that participant.sticky is reset unless the new participant.next is considered the preferred participant.</i>
priority	Modifiable: 1 Lowest ; 2 Low ; 3 Normal ; 4 High ; 5 Highest	Int	Priority of the instance.
process	Read Only. Any process	Fuego.Lib.Process	The process the instance belongs to.
process.id	Read Only	String	The Identifier of the deployed process containing the deployed process' name, its organization and organizational

Predefined Variable	Permitted Values	Type	Description
			unit.
process.idNumber	Read Only	Int	A number that identifies the process inside an server
process.name	Read only	String	The name of the process of the instance.
receptionTime	Read only	Time	The time that the current instance was received at the current activity.
result	Modifiable: Any string	String	It contains the result that you set in the BP-Method. For example, you can set a result and in the outgoing conditional transition of the activity ask for that value. See below for more information.
status	Read Only. RUNNING, EXCEPTION, SUSPENDED, GRABBED, COMPLETED, ABORTED, ACTIVITY_	ProcessInstanceState	Current status of the process instance.

Predefined Variable	Permitted Values	Type	Description
	COMPLETED		
timeout	Modifiable: String indicating an interval, enclosed in single quotes: '5m'	Interval	The length of time that a BP-Method or an external component has to complete before the server cancels its execution. By default this variable is set to 5 minutes.
totalCopies	Read only	Int	Number of threads or copies of an instance.

Using the Action Variable

When a BP-Method is executed, multiple types of output can result from the execution. BP-Method execution status is indicated by the value of the predefined variable *action*.

Depending on the value of *action*, the server will respond accordingly and will save or undo (commit or rollback) the changes invoked when the BP-Method is executed.

Valid values for the *action* variable

The valid values for the *action* variable are listed in the following sections.


Action.OK or **Action.NONE**

- Indicates that the BP-Method execution was successful.
- The default value for **action** variable. Therefore, it is not mandatory to set a value to the **action** variable.

Action.FAIL

- The way to indicate that the BP-Method has failed. The instance data is reversed and, if there is a rollback BP-Method, it is executed.
- Can be used in **any kind of BP-Method, in any kind of activity**.
- If a rollback BP-Method is included for Interactive activities, it is executed. If an Automatic activity fails, the server will retry the BP-Method until it succeeds or invokes the maximum number of retry times and it is routed to an exception handling activity. Maximum number of retry times is set in the Server Properties option on the **Run** menu.

Note

 A component exception is treated like *action=FAIL* by the server. NO error is logged in the server log. If you want to log a message, you will have to use the `logMessage` statement inside the Method before the failure.

Action.CANCEL

- The way to cancel the BP-Method. The instance data is reversed but the rollback BP-Method, if it exists, will not be executed.
- Can be used in activities of **Interactive, Grab or Global Creation** type.
- If the activity type is any other (i.e., *Automatic*, etc.), this value is

ignored, as if it had never been set.

- No trace of the BP-Method failure or execution appears in the Audit Trail.

Action.RELEASE

- Ends the BP-Method execution successfully.
- If the activity type is **Interactive, Grab or Automatic** the instance is released to the next activity without processing any of the BP-Methods in the current activity, even if they are marked Mandatory.
- If the activity type is **Join**, the original instance will be released and all the active copies will be aborted.
- If the activity type is any other (i.e., *Global*, etc.), this value is ignored, as if it had never been set.

Action.REPEAT

- If the activity type is **Interactive or Grab** REPEAT indicates that, although the task was successfully executed, it will remain **pending**. Therefore, the participant will be able to execute it again. Note that if the task is Mandatory, the participant will have to execute it repeatedly until the task appears as completed.
- If the activity type is any other (i.e., *Global, Automatic*, etc.), this value is ignored, as if it had never been set.

Action.ABORT

- Aborts the instance. The instance is sent to the End activity and marked as aborted.
- Used in activities of **Interactive, Grab or Automatic** type.
- Used in activities of **Join** type. The original instance will be aborted and therefore, all copies will be aborted.
- If the activity type is any other (i.e., *Global*, etc.), this value is ignored, as if it had never been set.

Warning



Instances that are aborted cannot be recovered.

Warning



The transaction is committed. Therefore, any change performed in the BP-method is persisted.

Action.BACK

- Used in an activity in an exception handling flow to send an instance back to the activity where the exception occurred. Normally, the exception flow is used to fix the exception and, after fixing, sends the instance back.
- Sends the instance back to the activity where the exception occurred.
- Used in activities of **Interactive, Automatic and Grab** type, when they are in an exception handling flow.
- If the activity type is any other (i.e., *Global*, etc.) or the activity is not an exceptions handler, this value is ignored, as if it had never been set.

Warning

The transaction is committed. Therefore, any change performed in the BP-method is persisted.

Action.SKIP

- Used in activities belonging to an exception handling flow to send an instance back to an activity in the main flow of the process. The instance goes to the point of BP-Method failure and is immediately released to the next activity without re-performing the BP-Method that caused the failure.
- Sends the instance to the activity immediately AFTER the one where an exception occurred.
- Used in activities of **Interactive or Automatic** type, when they are exceptions handlers.
- If the activity type is any other (i.e., *Grab*, *Global*, etc.) or the activity was not an exceptions handler, this value will be ignored as if it had never been set.

Warning

The transaction is committed. Therefore, any change performed in the BP-method is persisted.

Example: action variable

The following BP-Method example shows how you can manually set the *action* variable if a Boolean expression evaluates to *true*.

```
if selectedButton == "Yes" then
  action = OK

elseif selectedButton == "Abort" then
  action = ABORT
```

```
else  
action = BACK
```

Using the "result" variable

The **result** variable contains the result that you set in the BP-Method. For example, you can set a result and in the outgoing conditional transition of the activity ask for that value. Further in the process this value will no longer be available. The **result** variable is persistent between calls to the BP-Methods and is reset when the instance has already flown through the corresponding transition to another activity.

As well if the *action* variable is not used in the BP-Method, the predefined variable *result* values are mapped to the variable *action*. For example, if the BP-Method has the line **result = "fail"**, this is equivalent to **action = Action.FAIL**.

If the BP-Method uses the *action* variable, then the *result* can have any value, either predefined or not. For example:

```
result = "release"; action = Action.FAIL';  
//in this case the BP-Method fails  
result = "abort"; action = RELEASE';  
// in this case the BP-Method does not fail
```

The *action* variable is reset at every BP-Method invocation to the default value while the *result* variable is reset after the instance flows into the next activity.

Note



The predefined variable **result** is used to maintain compatibility to previous version.

Using the "timeout" variable

The **timeout** variable defines the time that a BP-Method has to complete its execution. If this time is exceeded, the Server cancels the BP-Method execution and assumes that it has failed.

There is a Maximum Timeout defined for all BP-Methods of all processes within the server. This value is defined in the FuegoBPM Web Console or FuegoBPM Studio Server Preferences.

The timeout set within a BP-Method (timeout variable) cannot exceed the Maximum timeout set in the Web Console. If you set a greater value, a runtime exception is thrown and the BP-Method execution is aborted.

See Timeout for further information.

Examples

Case01: Suspend Action

Find the project **ActionsCase01.fpr** in FuegoBPM's installation directory, in the studio/samples directory to study an example about handling the SUSPEND action and assigning the participant who suspended the instance.

To test this example, use the FuegoBPM's participant: **test**.

Local Variables

You define and use **local variables** inside a single process method. The scope of this type of variable is the method itself and cannot be accessed from outside. Once the method has been executed, the information stored in a local variable is lost.

Creating a Local Variable

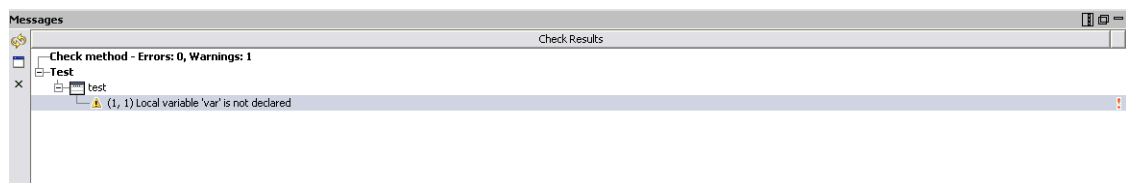
Local variables can be created by performing the following actions:

- Adding a local variable in the Studio Variable flap panel and clicking on + in the Local variables section,
- Declaring it from the Fix option.

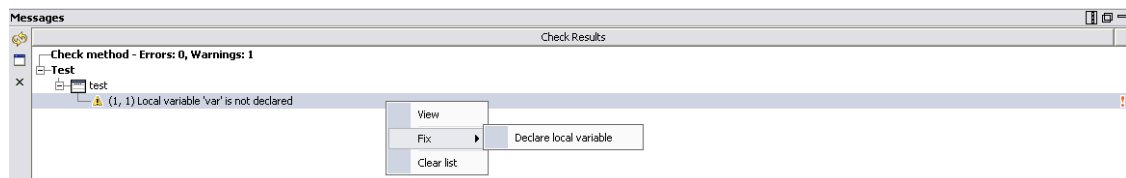
Example:

```
" var = 5 "
```

var is not a variable yet, so the compiler gets an error.



By right-clicking on the error, a **fix** option appears showing a possible solution for the problem.



Business Parameters

The business parameter is used to save information defined at the organization level. These parameters are visible for all processes.

Business parameters are available for any process within a Project.

Adding and changing a business

parameter

You can add a business parameter from the Variables tab. Click on the "+" sign:



A dialog box populates:

Organizational Unit	Values
Chicago	81

Define the type and the value of the parameter applicable for the whole organization.

If the parameter has a different value for any of the organizational units, you have to define it in the table below.

For example if taxes are the same percentage for all the organization

sales but in Chicago there is a different percentage, you have to specify it in the table.

Business parameters characteristics applicable for FuegoBPM Studio

It is strongly recommended **NOT** to change the value of a Business parameter as they should contain constant values.

If you do need to change a Business Parameter you can change it at runtime using the Fuego component **Business Parameter** in the **Lib** category. See the component documentation within the Studio.

Notes:

- If you change a Business parameter from a method you must be aware that the new value is not immediately available for all instances. Even more, if this value is changed from a BP-Method, the result may not always be the expected one and not available at the same time across the all participants.
- If the business parameter is used in a due transition expression of an activity, the business parameter value that applies is the one defined at the time the instance enters the activity. For example, the business parameter **"MAXTIME"** is used in the due transition expression of the activity **"Reply to customer"**. When the instance **"Request Customer 1"** arrives, the due time is calculated using the value that the **MAXTIME** has at that moment. If another instance (in any process) changes the value or you manually change it in the Web Console, the new value does not apply for the due time of the instance **"Request Customer 1"** for the activity **"Reply to customer"**. It will apply for all instances that arrive to that activity after the business parameter was changed.

Warning



if you change the Business Parameter at runtime, and you then stop and restart the Server, all business parameters are restored from the project definition.

Chapter 10. Exception Handling

Exception Handling

An exception is a special event that can occur at any point during an instance's flow through a process. It can be a system exception or a business (user) exception.

An exception is handled by a special flow called **Exception Handler flow**. An **Exception Handler flow** is a set of activities attached either to a *Group* or *Activity*. It responds to exceptions inside the *Group* or *Activity*. This flow is executed when the *Group* or *Activity* fails or throws an *exception*.

Before you continue learning about how Exceptions are handled, learn about different concepts and definitions related to Exceptions.

Exceptions

Exception Types

Exceptions can be classified as:

- *System Exceptions*
- *Business (or User) Exceptions*

System exceptions include error conditions like Communication exceptions, database exceptions, invalid operands, services not available or anything that occurs in an unexpected way (e.g., RuntimeException of Java). These exceptions are usually predefined in the System or they are defined by the programmers of low-level libraries. The **System exceptions** are those in the Java package or those that inherit from java.lang.RuntimeException.

Business exceptions are defined by the individual programmer and they are expected to be thrown in a normal execution. Examples of this kind of exceptions could be *Not Enough Money*, *Account Is Closed* or *Operation Not Authorized*. All other exceptions different from System exceptions are defined as **Business Exceptions** within FuegoBPM Studio or they are catalogued in an external jar.

Possible sources for **Business exceptions** are:

- User code (BP-Method or Fuego Objects) through the use of the 'throw' statement.
- Catalogued components with their own exceptions (i.e., a Web Service with a defined exception).

And possible sources for **System exceptions** are

- User code (BP-Method or Fuego Object) although this is highly **discouraged**.
- Catalogued components with their own system exceptions.
- FuegoBPM Studio Server. There are some special kinds of System exceptions that FuegoBPM Studio's server can throw and the user is not allowed to catch. These exceptions are *Time Out* and *Execution Aborted*. Moreover, it is important to note that they can be thrown during the execution of:
 - **Interactive activities** (no exception handler for System Exceptions is allowed).
 - **Automatic activities**

System exceptions tend to be temporary errors that can usually be solved by running the BP-Method again. However, trying to handle a Business Exception by retrying the code does not make much sense.

Exceptions must be created in the catalog (they are like Fuego Objects) or catalogued from any source (Web Services, Java components, etc.) See Cataloguing Exceptions.

Business (or User) Exceptions

You can manually throw an exception at any time by using the *throw* keyword and an exception name as shown in the following BP-Method line:

```
throw creditException
```

where *creditException* is the name of the exception.

Note



Standard naming convention specifies ending the exception name with *Exception*. This allows for easier readability of the variables in a process.

Handling exceptions

There are two possible ways to handle an exception when it is produced.

- Add an *on* statement at **BP-Method level** and then do the recovery work in the same execution.
- Add an **Exception Handler** at process level (or a smaller scope, for example for a group).

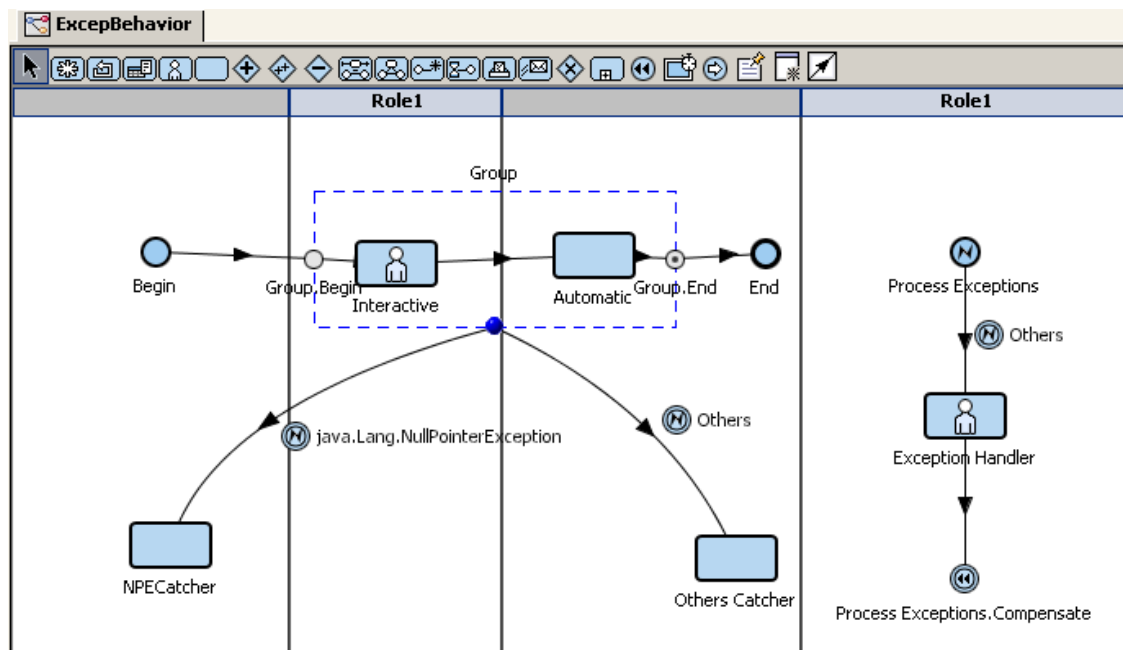
All kinds of exceptions (System or Business) can be handled in an Exception Handler defined by the user. However, in order to handle the exception in an **Exception Handler**, the exception must not be caught in the BP-Method or, if caught in the BP-method, it must be

thrown again.

How to handle exceptions

What happens if an exception occurs within an Interactive or an Automatic activity?

The following example summarizes the Exception behavior in the server. It also indicates when compensation will apply as the transaction is committed.



System Exception	User Defined Exception	Interactive activity	Automatic activity
Component, BP-Method or Server execution throws NullPointerException	-	The transaction is rolled back and the user receives the Exception. As no commit is	The transaction is rolled back, the activity is retried as many times as defined and in the end,

System Exception	User Defined Exception	Interactive activity	Automatic activity
		performed, no compensation is enabled	the instance is routed to the corresponding Exception catcher (NPECatcher). As no commit is performed, no compensation is enabled
-	- Component or BP-Method throws NewException	The transaction is committed and the instance is routed to the corresponding Exception catcher (OthersCatcher). No compensation is enabled.	The transaction is committed and the instance is routed to the corresponding Exception catcher (OthersCatcher). No compensation is enabled

Exceptions

See Exceptions to show a list of some typical Java exceptions.

How to handle Exceptions

BP-Method Level

If you want to handle an exception within the same BP-Method, use the *on* statement at **BP-Method level** and then perform the

recovery work within the same execution.

You have multiple alternatives to handle an exception within the same Method.

Handling Exceptions for a Specific Code.

When you program a set of statements, take into consideration that certain exceptions might occur within it.

Therefore, you can use the following code to contain the problem:

```
do
    // Set of statements
on [exceptionobject as] ExceptionName
    // Set of statements to treat the exception
end
```

ExceptionName is the name of the possible exception that can occur within the *Set of statements* and *exceptionobject* is the exception object name. In the statements to treat the exception, you can reference any attribute or method of the exception object.

Example 1

If the exception *Fuego.Sql.SQLException* occurs within a specific set of statements (in the example, *sql statements*), the exception is caught and treated within the same BP-Method.

```
// other statements
do
    // sql statements
on Fuego.Sql.SQLException
    display "There was an unexpected problem.
           Please, retry the operation."
end
```

If a *Fuego.Sql.SQLException* occurs within *other statements*, this instance of the exception is not handled within the BP-Method as the *Fuego.Sql.SQLException* treatment is only applicable if it occurs within the *do* statement (*sql statements*).

The previous BP-Method handles the exception at BP-Method level and no handler is called.

Example 2

This example is similar to the previous, but the *throw* statement , as well as the use of the *exception object* are introduced.

```
do
  // sql statements
on se as Fuego.Sql.SQLException
  display "There was an unexpected problem.
  Please, retry the operation" + se.message
  throw se
end
```

In this example, the *Fuego.Sql.SQLException*, referenced as well as the *se* object is thrown to be captured and eventually treated by the **Parent's Group Exception flow**. The exception is propagated (see **Exception Handler Flow** below).

Note



In **Interactive activities**, only **user defined** exceptions can be thrown to be captured and treated by the Parent's Group Exception flow. System exceptions, display an error message, but the instance will remain in the Interactive activity.

When you use an object reference as *se* is used in the example above, you can handle the exception attributes and methods. For example the *cause*, *message*, *localizemessage*, *stackTrace*, among others. To see all available attributes and methods press Ctrl + space bar after the typing *se*.

Handling Exceptions for the Entire Method

Another way to code the exception handling is by **stacking** all exceptions at the end of the method. In this case, any exception that occurs within the Method (*Set of statements*), if defined at the end of the method, is caught and treated within the Method.

```
// Set of statements
on [aa as] ExceptionName1
do
    // Set of statements to treat the ExceptionName1
end
on [bb as] ExceptionName2
do
    // Set of statements to treat the ExceptionName2
end
on Exception
do
    // Set of statements to treat all other exceptions
end
```

If the *ExceptionName1* or the *ExceptionName2* exceptions occur in any part of the Method, they are handled within the Method.

Example

In the example below, the *on exit* code is introduced. This statement is very helpful to release any locked resources.

```
// first of many exceptions that might be caught
on IllegalArgumentException
do
    // some code here
    display "The problem is " +
        IllegalArgumentException.message
end
// second of many exceptions
on bb as SQLWarning
do
    display bb.message
end
```

```
// add all other known exceptions to handle
//...
// Other exceptions treatment
on Exception
do
    // some code here
    display "The problem is " + Exception.message
end
// This is the "finally" syntax to close files, etc.
// This code is executed no matter what exception occurred
// (or did not occur)
on exit do
    close theFileThatWasOpened
end
```

- Multiple known exception treatments are handled as *IllegalArgumentException* and *SQLWarning* exceptions.
- The *Exception* exception captures all non-specified exceptions.
- The **on exit do** statement makes sure that all locked resources are released.

Forcing an Exception

Under certain condition you might need to force an **Exception** that is caught and treated in the same Method or in a Parent's group Exception Handling flow.

For example, you need to handle all Customer database interface problems in an Exception. So, if there is a problem with a Customer information search, you want to *throw* an exception, which is caught by an Exception Flow that contains BP-Methods to deal with Interface problems.

```
if (CustomerNotFound = true) then
    throw InterfaceErrorException
```

```
end  
  
on interr as InterfaceErrorException  
do  
    // Set of statements to treat the Customer Interface  
    // problems  
end
```

or

If the **on** statement is not defined in the same Method, then the exception is sent to the Parent's group and if there is any exception transition that catches the *InterfaceErrorException* exception or an *Others* exception transition, the Parent's group Exception Handling flow handles it.

Exception Handler Flow

The other way to handle an exception is to add an **Exception Handler** at process level (or a smaller scope, for example for a group).

Exception Handlers are associated to a specific Exception and they are defined in a scope of action. The scope of the handler can be:

- **an activity**
- **a group of activities**
- **the whole process.**

Any **Business** or **System Exception** can be used to define an exception handler. However, **these exceptions must be added to the Project catalog** of the Exception Handler's process in order to use them.

Defining an Exception Handler Flow

Exception handler flows are a set of activities designated for exception handling. These activities do not reside between the Begin and End activities and are not connected to the main flow by transitions.

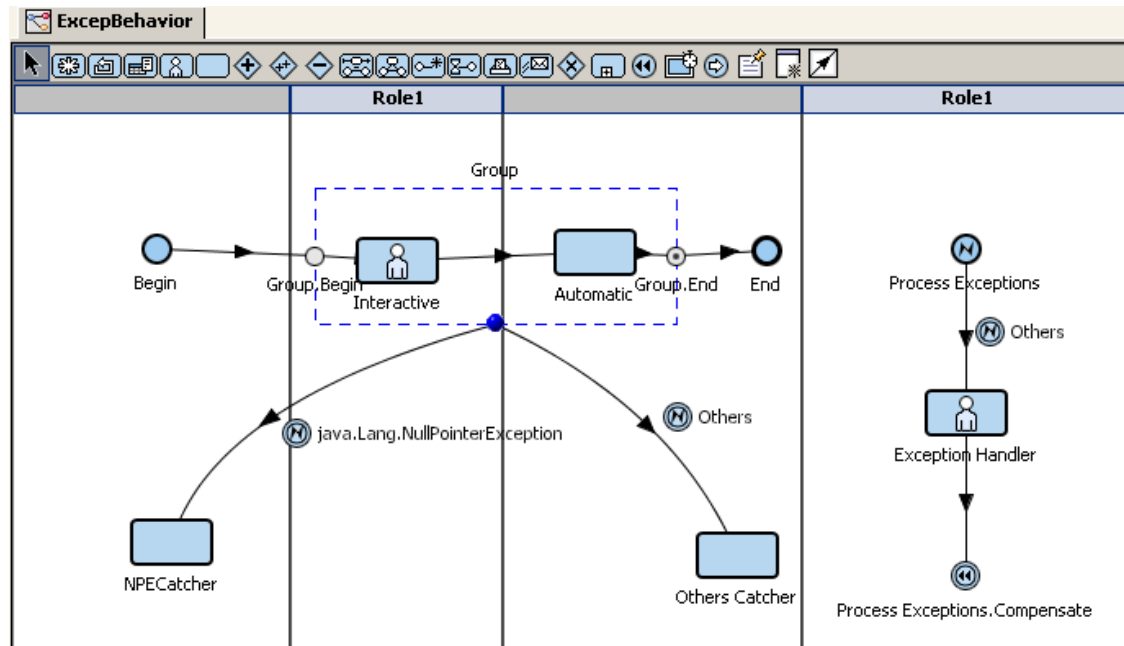
When an exception occurs, the original instance is submitted to the corresponding **Exception Handler flow** based on the exception transition, if defined.

If the activity has **no** Exception Handler flow, the exception is caught by the **Parent's Group Exception handler flow**. If the Parent's Group Exception handler flow is not defined, then the instance is treated by the Process Exception Flow, if defined.

Characteristics

- *Where can Exception Handler Flows be added?:* An exception handler can be added to any activity (except to a Begin or an End). The Begin and End activity can fail under certain conditions but these cases are not very common.
- In the Begin activity, if a System Exception occurs, no instance is generated (for example, if an exception happens while mapping the arguments). And if a User Defined Exception happens the instance is created and it will flow to the **Process Exception Flow** for its handling. This last case can happen if you use a BP-Method in the Begin activity, and actually this is **not** recommended.
- In the End activity, if an exception occurs, the instance will flow to the **Process Exception Flow** to be handled. If the End activity has to send a notification, the BP-Method is retried forever (or until the server is stopped) in case of failure.

- *Handling more than one type of exception:* The business designer could define one or more exception handlers for the same group as long as they handle different exceptions. All these exceptions must be catalogued.

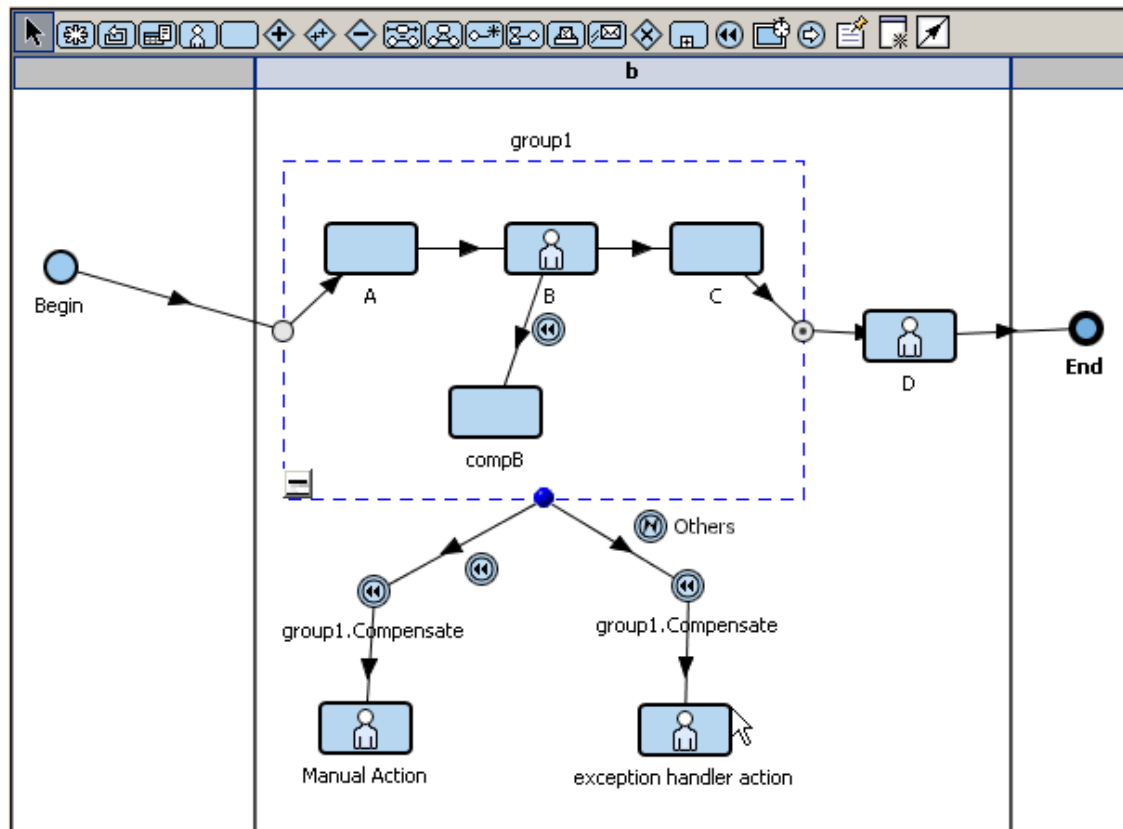


- *Throwing Exceptions:* If you explicitly define an exception handler, you should be careful about throwing the exception again or ignoring it.
 - If the exception is **thrown again**, it will be caught by the *Parent Group* (outer group).
 - If the exception is ignored, the instance will continue its normal execution, however the execution of the *Group* is considered failed. You should fix the condition that produced the exception.
- Example: If the exception is an expiration or some kind of

exception that will leave the instance in an unusable state, if this instance returns to the normal process flow, it will fail in the following activity.

Returning from an Exception handler flow

- If the user sends the instance back using the Work Portal, the instance will be sent to the activity where the exception was thrown or thrown for the second time.
- Once the instance reaches the end of the Exception Handler flow and if no action was defined (*BACK*, *SKIP* or *ABORT*), it will be submitted to the next activity after the group that threw the exception.



In the above example, once **group1's exception handler** flow actions finish (after executing *exception handler action*), the instance is submitted to *activity D*.

Examples:

When writing your BP-Method within the exception flow, you have a few choices on how to handle the exception condition. You can try any of the following:

- **action = BACK** : Sends the instance back to where it encountered the exception. You must fix the problem that caused the exception before you send the instance back.

```
// fix the problem - for example, extend a process deadline
```

```
deadline = 'now' + '2h'  
// send the instance back to the main flow  
action = BACK
```

Note



If you have not fixed the exception condition, the exception will continue to be thrown.

- **action = SKIP** : SKIP returns the instance to the main flow and releases the instance from the activity that caused the exception. The task(s) are not processed even if marked as **Mandatory** . This action may require additional process design logic to complete any required BP-Method you skipped by using action = SKIP.

```
// send the instance back and skip the Method task  
action = SKIP
```

- **action = ABORT** : If everything else fails, abort the instance.

Warning




Aborted instances cannot be retrieved. Use *logMessage* to capture any pertinent instance information before aborting.

Automatic Activity as the last activity in the Exception Flow

When the last exception flow activity is an automatic activity, after the execution is completed and the default action (action = OK) takes place , the instance will be sent to the activity next to the scope where the exception was thrown.

Note

 A *lonely* due transition (no other outgoing transition) outgoing this last automatic activity in the exception flow does not mean that the automatic activity has a next activity. Therefore implementing this use of due transition within an exception flow is useless

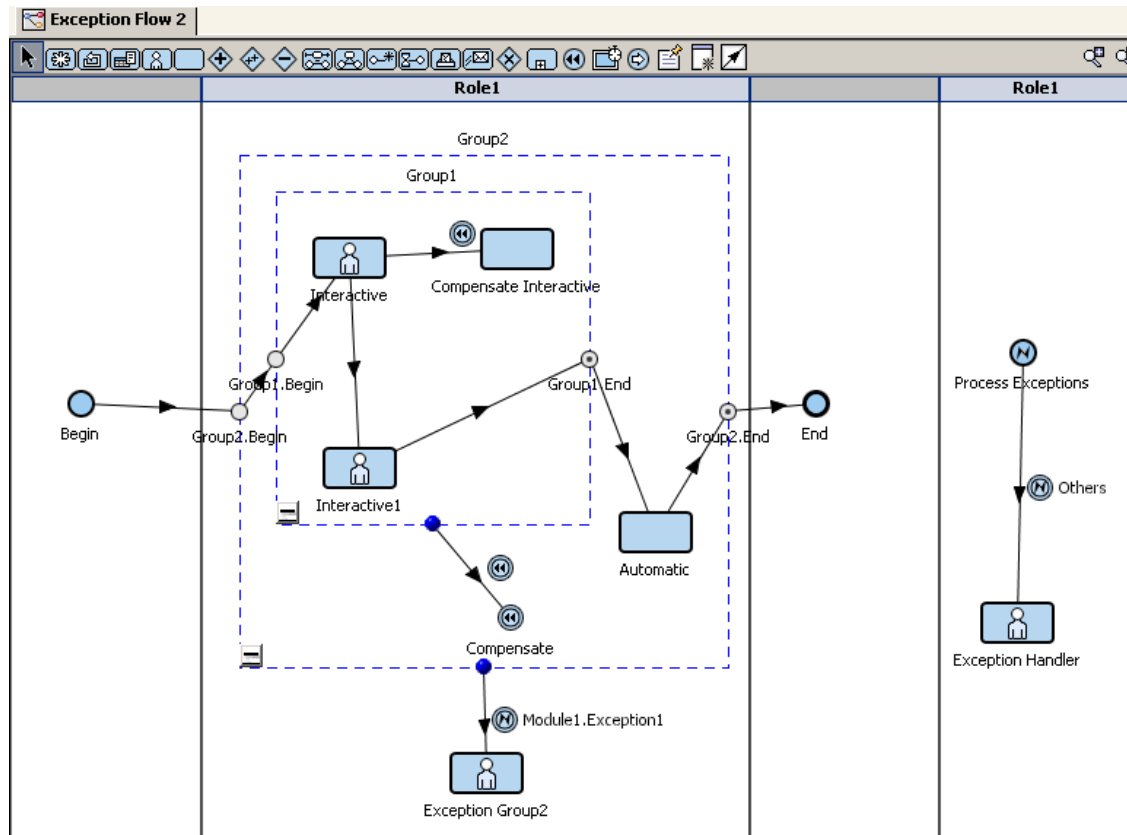
Default Exception Handler Flow

The **Default Exception Handler flow** applies to any group and refers to the Exception Handler flow that takes care of an exception under certain circumstances.

If the *Group* does not have an explicit *Exception Handler Flow*, the **Default** is set to this *Group* **dynamically**. The *Default Exception Handler* is not visible from the designer.

- It will catch all exceptions (**catch all** clause defined in BPEL4WS) inside the *Group* and invoke all compensations handlers inside the *Group* in the reverse order of completion.
- Rethrow the fault to the *Parent Group*.
 - If the *Parent Group* is not contained by any other *Group*, the instance is treated by the Process Exception flow.

For example:



- If an exception called *Exception2* occurs within *Group 1* as no *Exception Handler Flow* was defined, the **Default** set dynamically does the following:
 - catches the exception and invokes the *Compensate Interactive* activity as part of the invocation to all compensations handlers inside *Group1*.
 - rethrows the exception to *Group 2* and the instance flows to the *Exception Handler Flow* for *Group2*.
 - as this flow does **not** treat an exception as *Exception2* (but does an exception called *Exception1*) the exception is re-thrown to the *Process Group*, therefore, to the *Process Exception Flow*. So the instance flows to the *Exception Handler* activity.

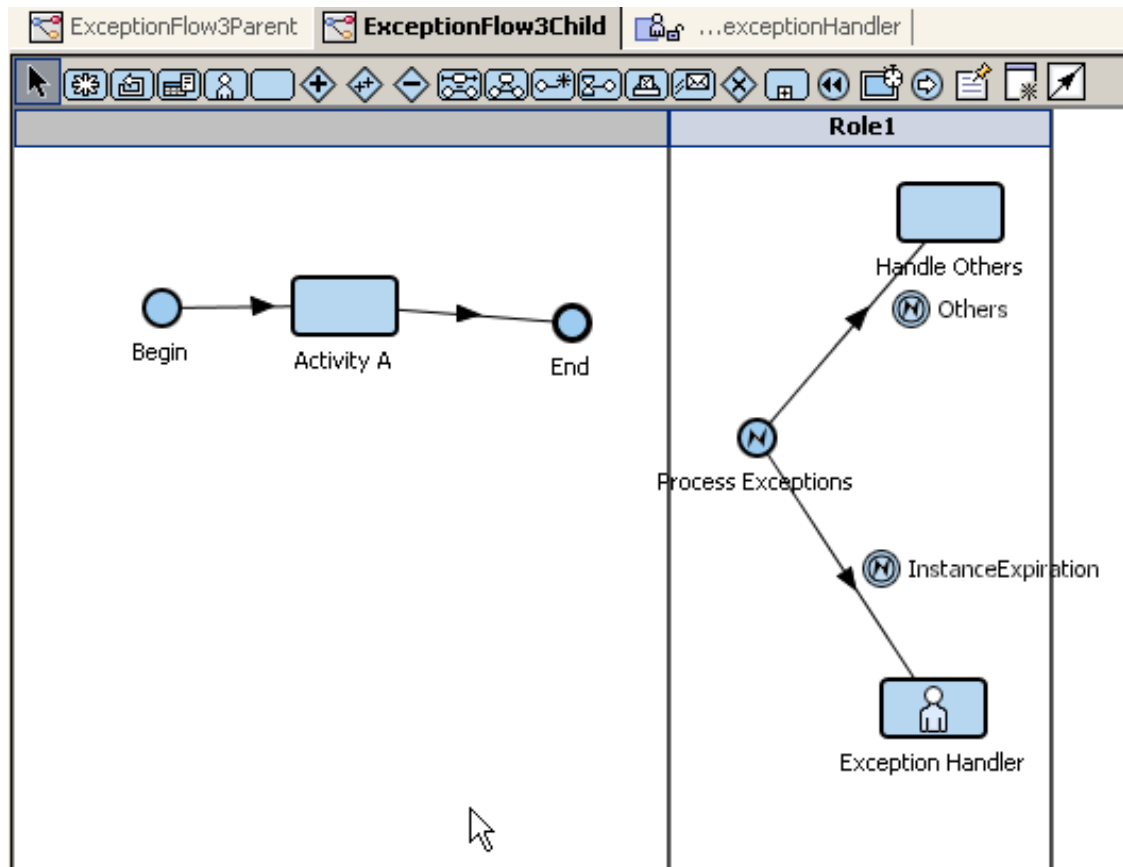
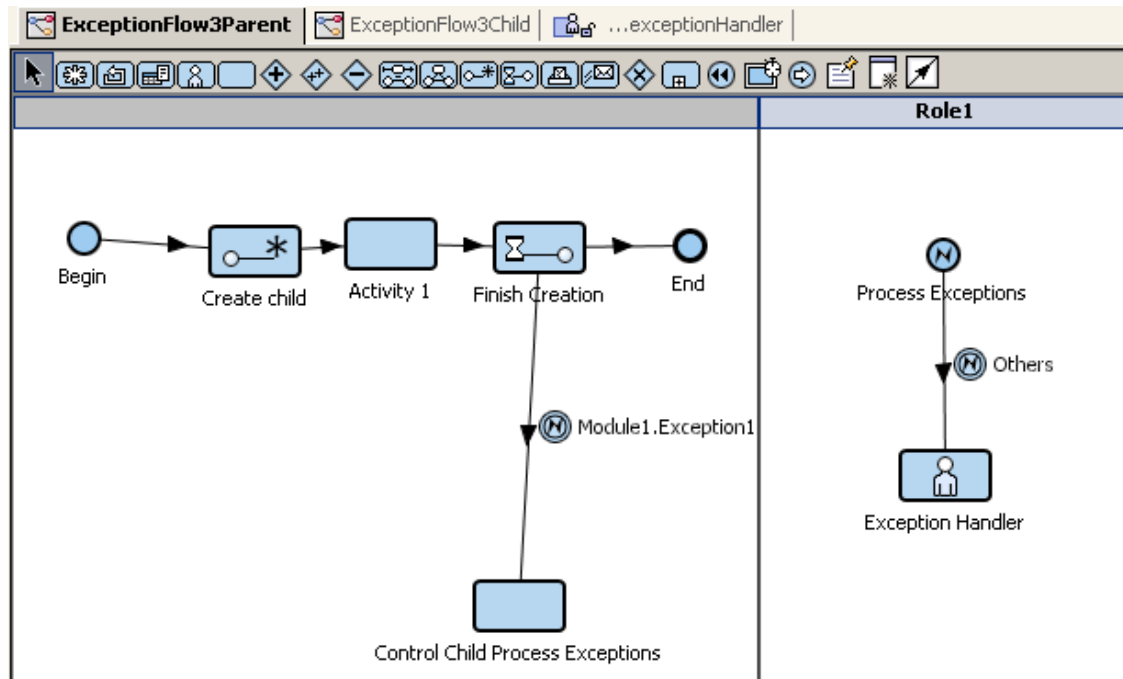
- if the *Process Exception Flow* would not take care of all **Other** exceptions, but for example only for *Instance expiration* exceptions, then *Exception 2* has no possible treatment and the instance flows to the End.

What happens if the instance was created by a process into a Subprocess?

1. If the child process cannot handle the exception, then the exception is re-thrown to the Parent Process. The parent instance will not receive the regular notification for it to continue in the Parent process flow. Instead, it is sent to the corresponding Flow exception with the child's exception. The child instance flows to the End with an ABORTED state.
2. If the child process can handle the exception then it is very important to send the instance BACK o ABORT it so the parent instance is notified to continue its flow. If not, the parent instance will remain in the Subflow or Termination Wait activity.

For example:

In the following project there are 2 processes: **ExceptionFlow3Parent** and **ExceptionFlow3Child**. The activity **Create child** from the process **ExceptionFlow3Parent** creates instances in the process **ExceptionFlow3Child**. The parent instance flows through **Activity 1** and remains in the **Finish Creation** activity until it receives its child's notification.



Case 1: Exception1

If the exception called **Exception1** happens in the activity **Activity A** within the process **ExceptionFlow3Child**, no exception flow in the **child** process can handle it. Therefore, the exception is re-thrown to **ExceptionFlow3Parent**. The parent process receives the exception. As there is an Exception Flow that handles that type of exception, the **parent instance** flows to the **Control Child Process Exception** activity.

The **child instance** flows to the End with an ABORTED state.

Case 2: Instance Expiration

If an instance expiration exception happens within the **ExceptionFlow3Child** (in **ActivityA**), the child process will handle it as the Process Exception Flow has that option. But it is very important to send the instance BACK or ABORT it so the **parent instance**, remaining in the **Finish Creation** activity, is notified and can continue its flow.

```
deadline = 'now' + '2h'  
// send the instance back to the main flow  
action = BACK
```

The **child instance** flows to the **End**.

Propagation of Exceptions

An exception handler defined for an activity could do something and then call the exception handler defined at process level for that exception.

Any exception that occurs within an Exception Handler flow is managed by the Exception Handler flow of its *Parent group*. If there is no explicit Parent Group, it will be managed by the Exception Handler flow of the Process Group or Process Exception Flow

System Exceptions behavior (Non user defined exceptions)

System exceptions have a particular behavior for some scenarios:

- In Interactive activities, if a system exception occurs, an error is displayed in the Work Portal and the instance will remain in the Interactive activity. If the system exception occurs within an Automatic activity, the server will retry until the maximum retries (configured in the Web Console) is reached. After that, the instance is sent to the corresponding Exception Flow.

Examples

Example: User-Defined Exception

Throwing a user-defined exception

In the following example, logic in the BP-Method requests the end user to enter the amount of credit the customer is requesting. This company's business rules require that any amount over the specified credit limit must be approved by a supervisor.

```
input "Credit request amount" credit
  using title = "Requesting Credit",
  buttons = ["OK", "Cancel"]
  returning mySelection = selection

if credit > creditLimit then
  throw creditException
else
  ...
end
```

Catching a User-defined Exception

After a user-defined exception has been declared, you will need to decide how to program the BP-Method to fix the exception conditions. At runtime, after the exception is thrown, if the exception *is not treated in the same BP-Method*, the exception jumps out of the main flow of the process and into the defined **exception flow**.

Exception flows are not connected to the main process flow, which enables unexceptional instances to continue flowing normally through the main process flow without being hindered by the instance causing the exception.

You can have many activities of many different types in an exception flow, but you may want to end the exception flow with an Interactive activity in a supervisory role lane.

Warning



Exceptions must be fixed and sent back to the main flow of the process (**action = BACK**) or aborted (**action = ABORT**). The decision to abort should be made by a human with a thorough understanding of the consequences of aborting an instance. Once aborted, an instance cannot be retrieved.

Another alternative is to use **action = SKIP** to skip the task causing the problem. Note that any mandatory BP-Method for the activity or the task in the activity is not performed when you use **action = SKIP**.

In the following BP-Method, a supervisor makes the decision on whether to abort the instance. Note the use of *logMessage* . Using *logMessage* ensures that appropriate instance information is recorded in the FuegoBPM Server log files. These log files can be viewed from the **FuegoBPM Logviewer**.

```
// exception cannot be fixed programmatically
if exceptionCondition == true then
    display "Do you want to abort the instance?"
    using title = "Abort Instance",
    type = "warning",
```

```
        options = ["Abort", "Cancel"],
        default = "Cancel"
        returning userSelected = selection

    // check to see what user selected
    if userSelected == "Cancel" then
        // call method to try to fix exception
        fixException FuegoExceptions
    end

else

    // log the instance
    logMessage "The instance " + description +
    " was aborted. Customer number: " + custNum +
    " Order Amt: " + orderAmt
        using severity = SEVERE

    // abort the instance
    action = ABORT

end
```

Process to Handle Exceptions

There is another option to handle exceptions: Handle User-defined Exceptions in a **dedicated process**.

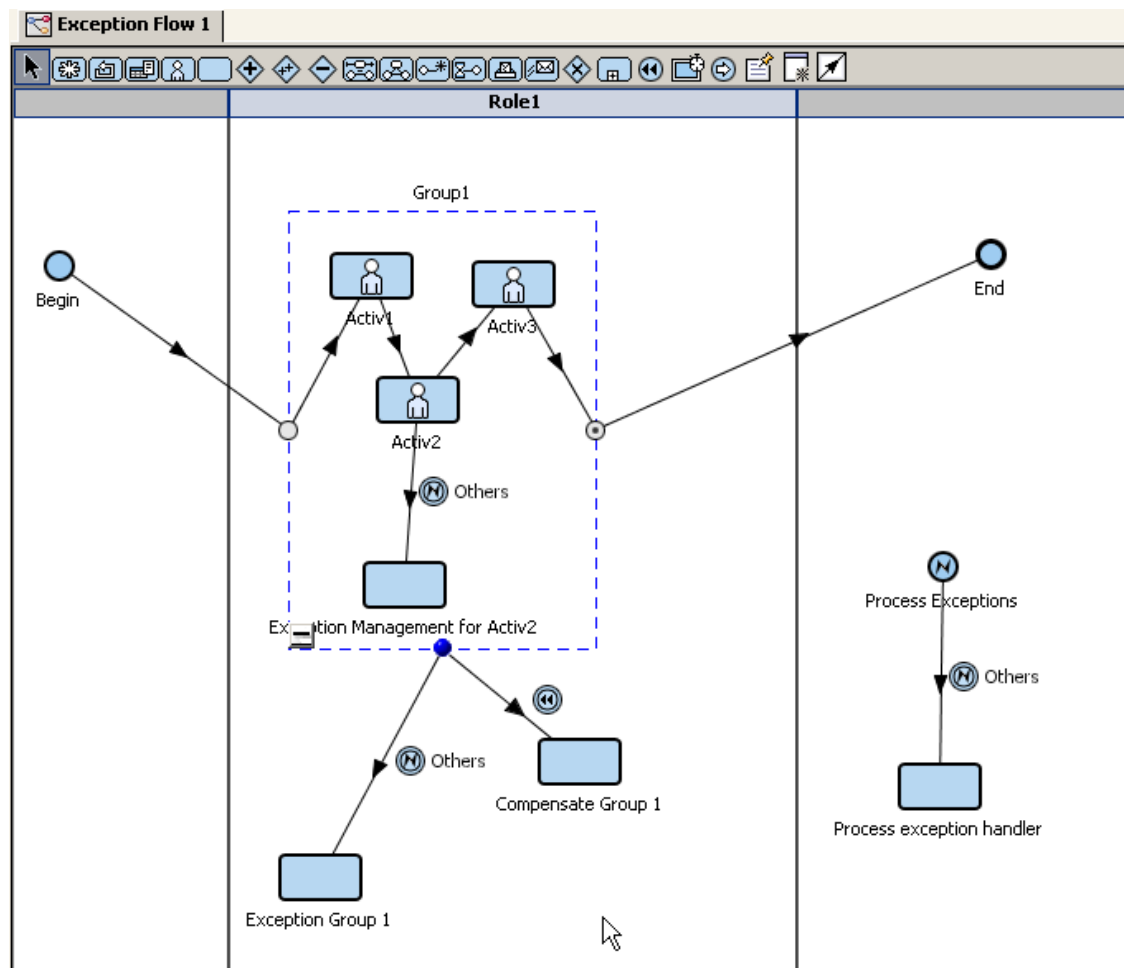
This is very useful if you have a list of known exceptions that can occur in multiple processes within your project or in batch processes that need to continue processing when an exception occurs.

See Exception Examples - Case 1 for the example

Exceptions within an Exception Handler Flow


What happens if an exception occurs within an Exception Handler Flow?

The following example shows you the behavior for this scenario:



- If an exception occurs within *Activ 2*, it will be handled by its own Exception Handler flow. In this case it will be handled by *Exception Management for Activ 2*
- If an exception occurs within *Exception Management for Activ 2*, it will be handled by the Parent's Group Exception Handler Flow. In this case it will be handled by *Exception Group 1*

Note

 If the exception is perfectly handled by the Parent's Group Exception Handler and the instance completes the flow with no further problems, then

the instance will return to the main flow and both exceptions (the one occurred within *Activ2* and *Exception Management for Activ 2*) are resolved. The instance flows to *End*

- If an exception occurs within *Exception Group 1* activity, it will be managed by the outest group, therefore, by the Process Exception Handler Flow. In this case it will be handled by *Process exception Handler* activity.
- If an exception occurs within *Process exception Handler*, the instance flows again into the Process exception Handler flow with the exception that has been thrown. If this same exception occurs or has occurred twice for the same activity and for the same instance, the instance does not flow again into the Exception flow, but remains in the activity where the exception happened until somebody manually grabs it and corrects the problem.

For example, if the **exception A** occurs within the *Process exception Handler* activity, the instance flows again into the **Process exception Flow**. As it arrives at the *Process exception Handler* activity, the same **exception A** happens therefore as it happens for the second time, the instance will **not** flow into the exception flow but remain in the *Process exception Handler* activity.

- If an exception is thrown and the instance arrives at the **Process exception Flow** but within this flow no treatment is expected for that exception, it will be managed by the Main Default Exception Handler, handled by the Server directly. This Exception handler is not shown in the designer but it will execute the compensation for all the process. In this case *Compensate Group 1* will be executed. Afterwards the instance is sent to the End, with an Aborted state.

Global Automatic Activity and Exception

Handling

The Global Automatic does not have any association with instances (there is no instance pending in the activity), so the behavior when dealing with exceptions is a little different from that with other activities. See Global automatic activity and Exception for further information.

Exception Examples

Using Exceptions

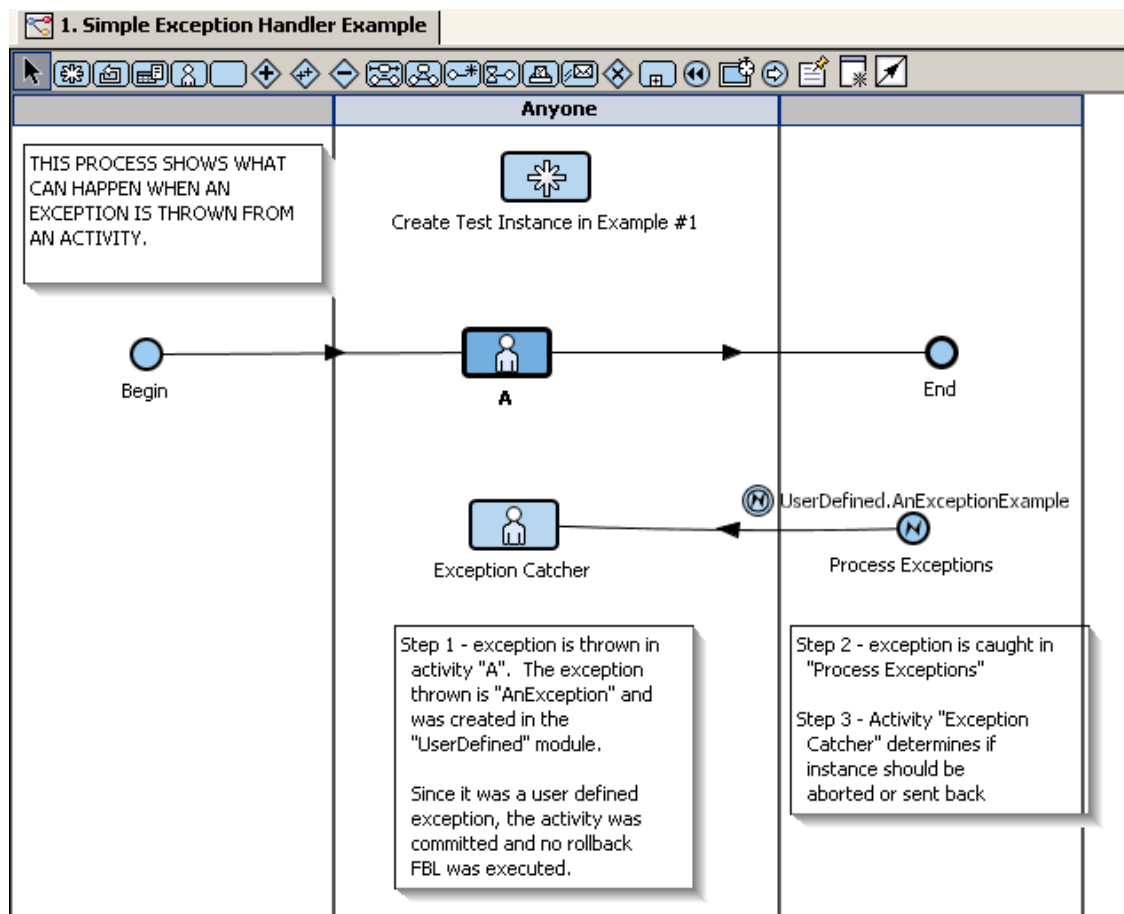
Case 00: Simple Exception Examples

The following examples show different scenarios on how exceptions can be handled:

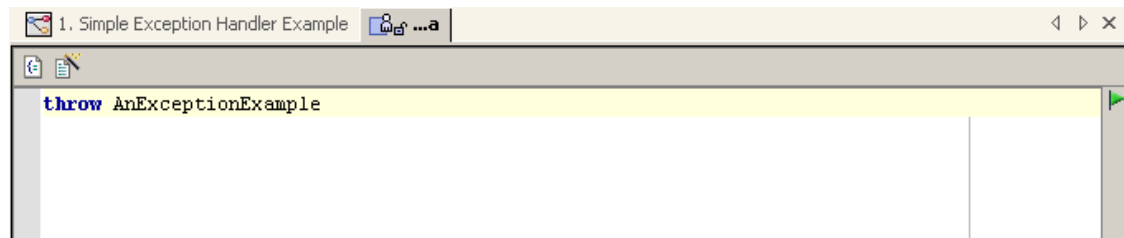
1. Exception thrown by an activity
2. Exception within a group
3. Exception caught by Process Exception
4. Exception propagated from Subprocess example

1. Exception thrown by an activity

This process shows what can happen when an exception is thrown from an activity.

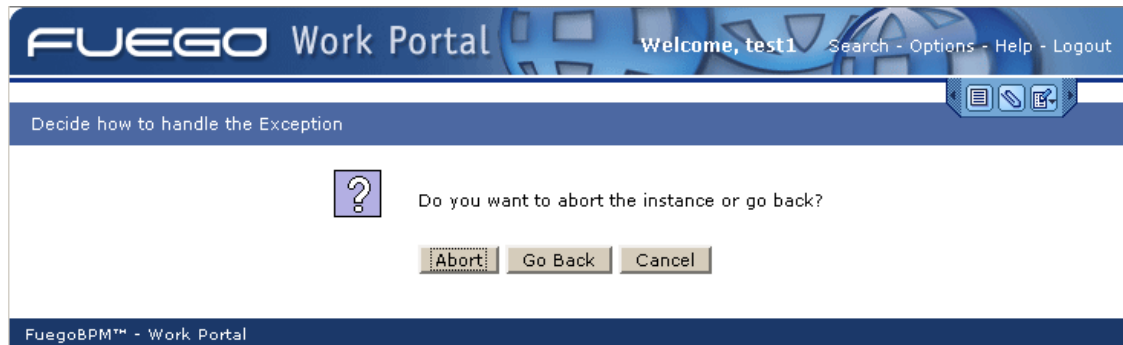


The interactive activity **A** throws the exception **AnExceptionExample**.



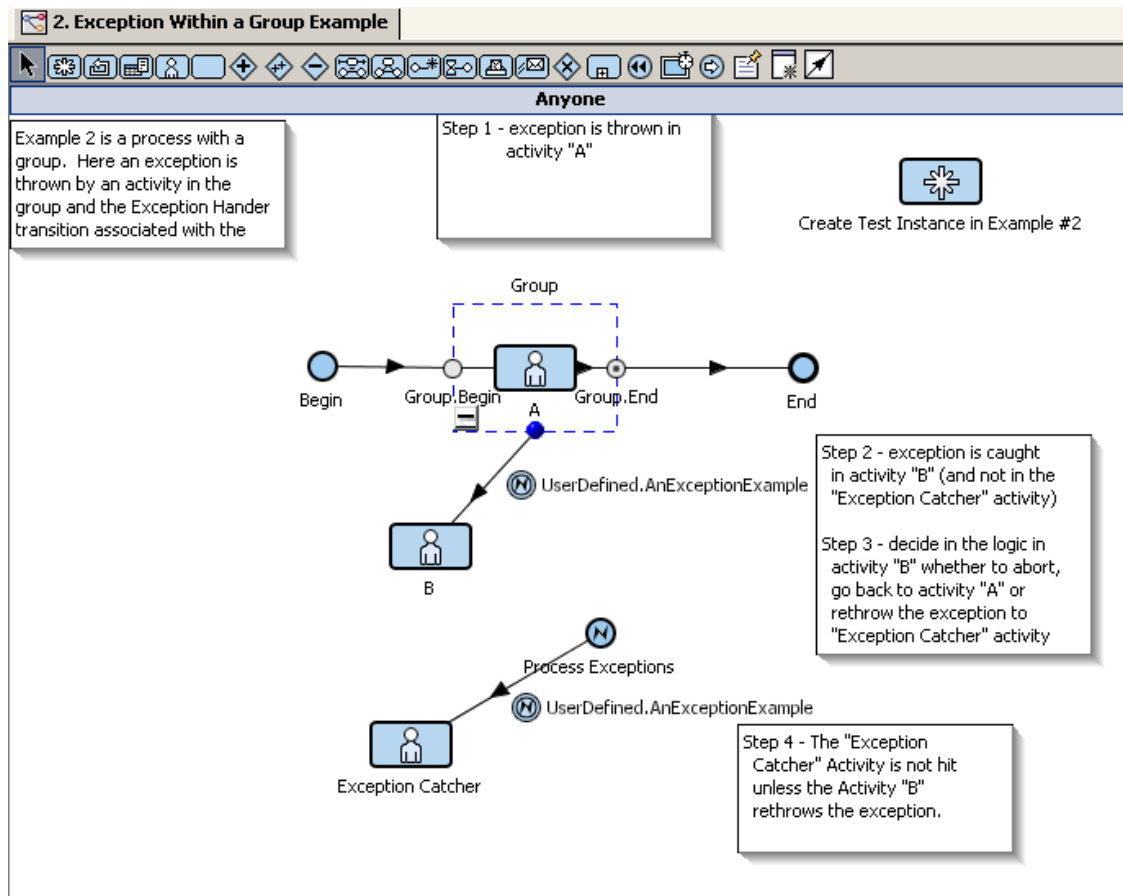
This exception is caught by the **Process Exception Flow Handler** and as it does treat that exception, the **Exception Catcher** activity within the flow is run.

In Work Portal you handle the instance sending it **BACK** or **ABORTING** it.



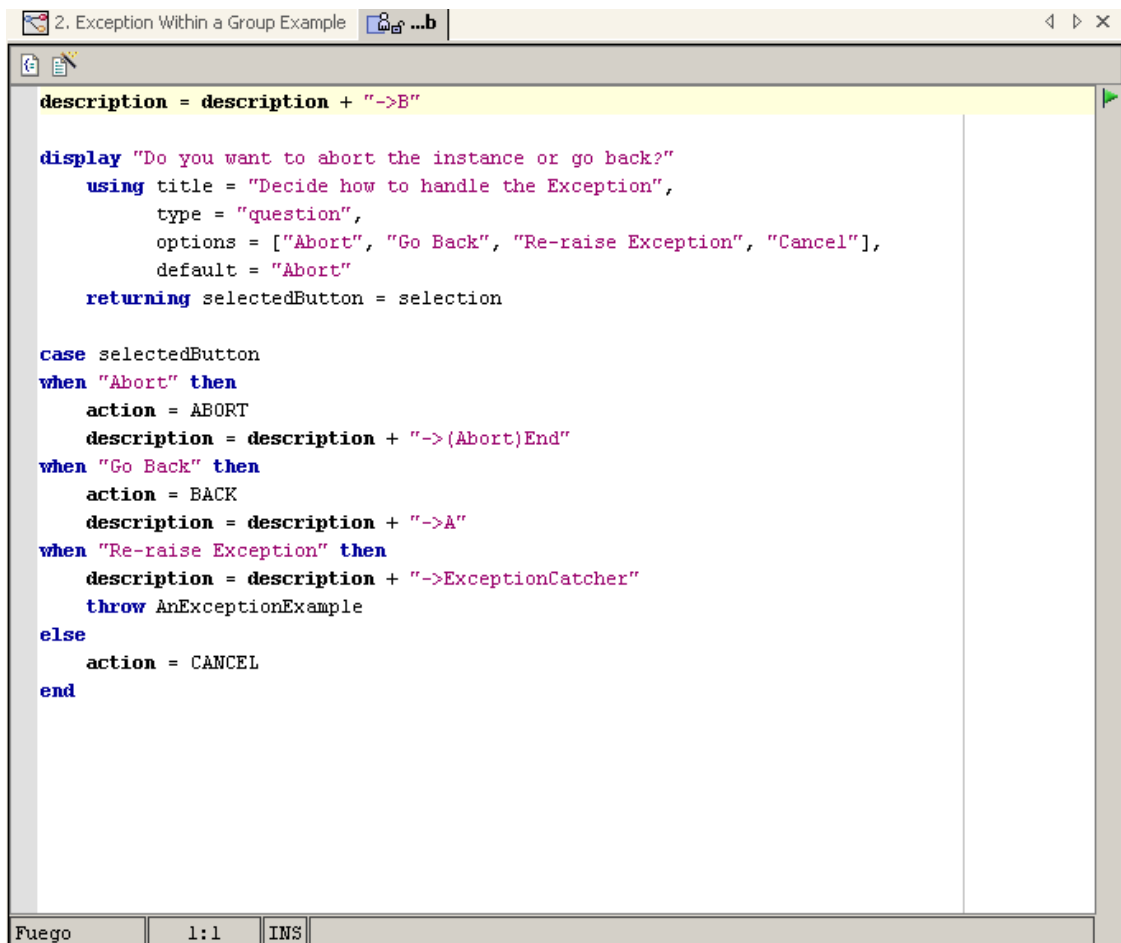
2. Exception within a group

The following process has a group and an exception is thrown by an activity in the group. The Exception Handler transition associated with the Group catches it and handles it. It also shows what happens when an exception is "re-thrown" (re-raised) and caught by the process's Exception Handler.



The interactive activity **A** throws the exception **AnExceptionExample**.

The exception is caught in the interactive activity **B** as this type of exception is handled by the **Group Exception Flow Handler**. The participant has to choose to **Go Back**, **Abort** the instance, or **Re-raise** the exception. In this last case, the exception is re-thrown to the **Process Exception Flow Handler**.



```
description = description + "->B"

display "Do you want to abort the instance or go back?"
  using title = "Decide how to handle the Exception",
    type = "question",
    options = ["Abort", "Go Back", "Re-raise Exception", "Cancel"],
    default = "Abort"
  returning selectedButton = selection

case selectedButton
when "Abort" then
  action = ABORT
  description = description + "->{Abort}End"
when "Go Back" then
  action = BACK
  description = description + "->A"
when "Re-raise Exception" then
  description = description + "->ExceptionCatcher"
  throw AnExceptionExample
else
  action = CANCEL
end
```

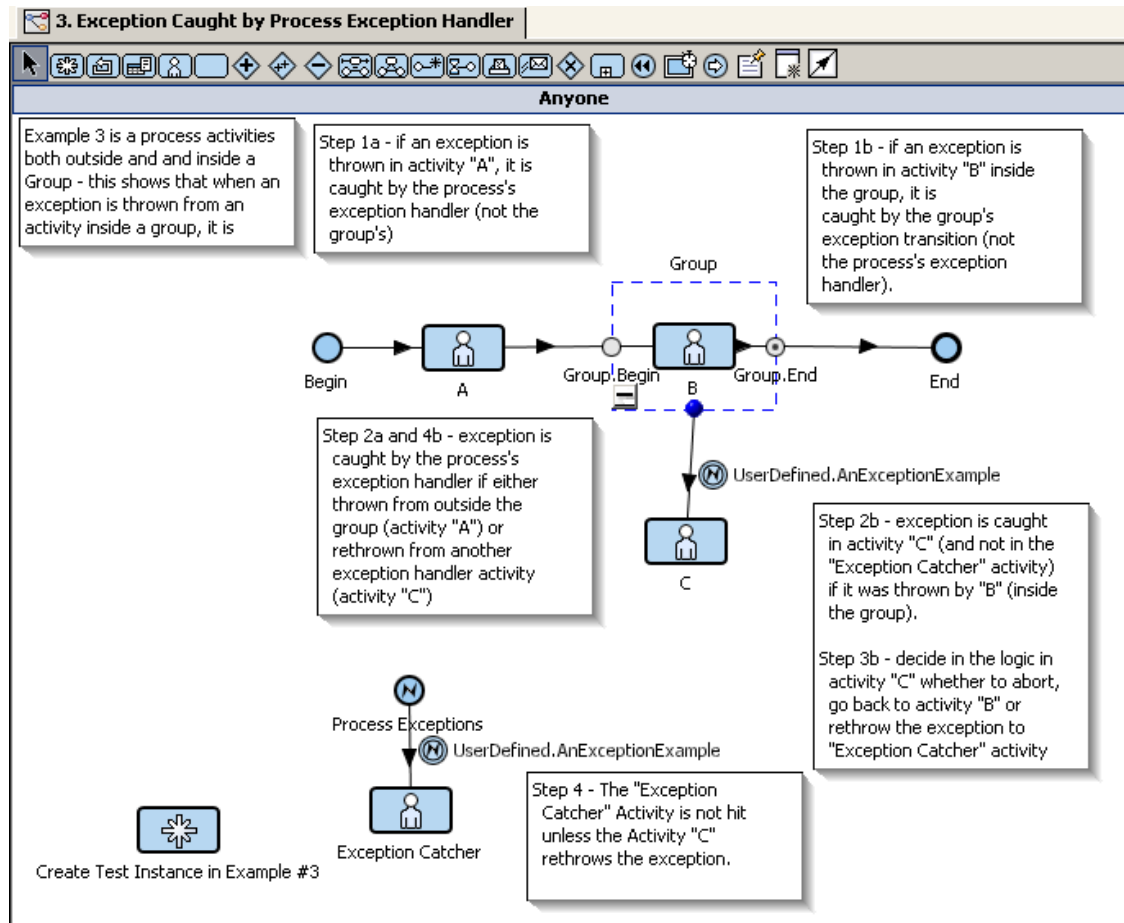
In Work Portal when you re-raise the exception, the instance appears in the **Exception Catcher** activity within the Process Exception flow.

The screenshot shows the FUEGO Work Portal interface. The header includes the logo, user name 'test1', and navigation links. A left sidebar contains menu items like 'Inbox', 'Applications', 'Attachments', 'Bookmarks', 'Consultations', and 'History'. The main content area is titled '2. Exception Within a Group Example > Inbox' and shows a table with one entry.

	Description	Activity	State	Received	Deadline	Participant
A->B->ExceptionCatcher	Exception Catcher	Exception	5:58 PM			

3. Exception caught by Process Exception

The next example is a process that have activities outside and inside a Group. When an exception is thrown from an activity inside a group, it is caught by the group's exception transition (if it exists). When an exception is thrown from an activity outside a group, the Process Exception handler flow catches it.



The interactive activity **A** can throw the exception **AnExceptionExample** or just pass the instance to activity **B** in which the exception is thrown.

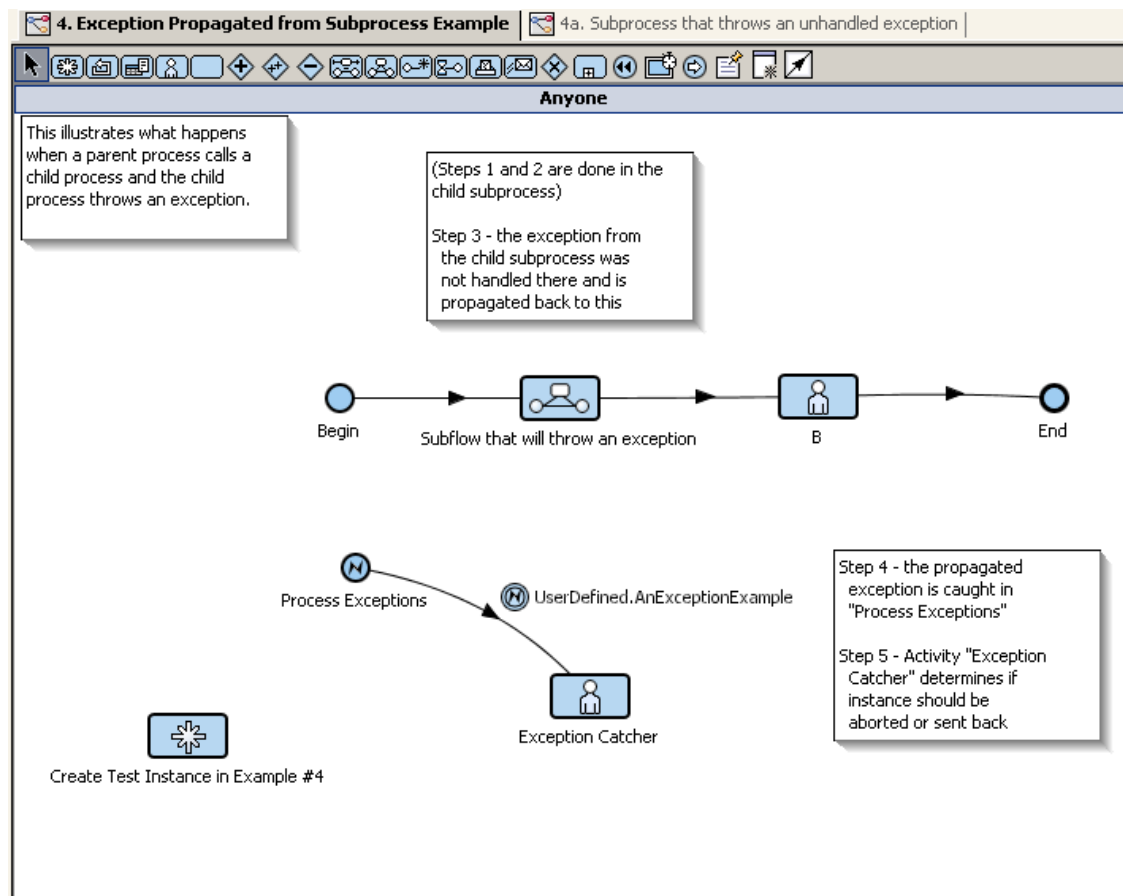


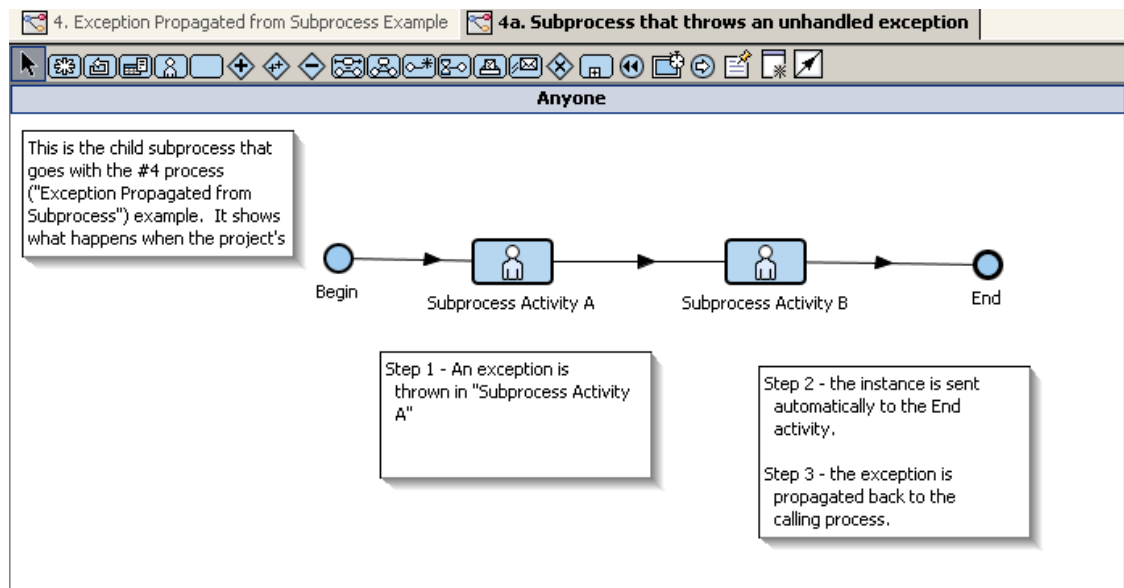
If you choose to throw the exception in the **A** activity, the instance flows with the exception to the Process Exception flow. If it is passed to **B**, the exception is thrown in this last activity and the Group

Exception Handler flow catches it (in activity **C**).

4.Exception propagated from Subprocess example

The following example illustrates a Parent process that calls a Child process and the Child process throws an exception.





A child instance is generated in the Subprocess (4a). Parent instance and child instance have similar descriptions. The difference is that the child description ends in *Sub*.

FUEGO Work Portal Welcome, test1 Search - Options - Help - Logout

Inbox Showing 1-1 of 1

Description	Activity	State	Received	Deadline	Participant
Instance4Sub	Subprocess Activity A	Running	6:01 PM		

FuegoBPM™ - Work Portal

The Subprocess does not handle exceptions, so when one occurs, it is propagated to the parent process (4).

Subprocess activity A throws an exception that is not handled in the Subprocess. Therefore, it is propagated and the parent instance receives the exception.

The screenshot shows the FUEGO Work Portal interface. The top header includes the logo, 'Work Portal', a welcome message 'Welcome, test1', and links for Search, Options, Help, and Logout. A left sidebar contains navigation links: Inbox, Applications, Attachments, Bookmarks, Consultations, and History. The main content area is titled 'Inbox' and shows a table with one entry. Above the table are various action icons. The table has columns for Description, Activity, State, Received, Deadline, and Participant.

Description	Activity	State	Received	Deadline	Participant
Instanced	Exception Catcher	Exception	6:02 PM		

Notice that the parent instance is the one that has the exception and it was sent to the **Exception Catcher** activity.

Find these examples in the **ExceptionCase00.fpr** project in FuegoBPM's installation directory, in studio/samples.

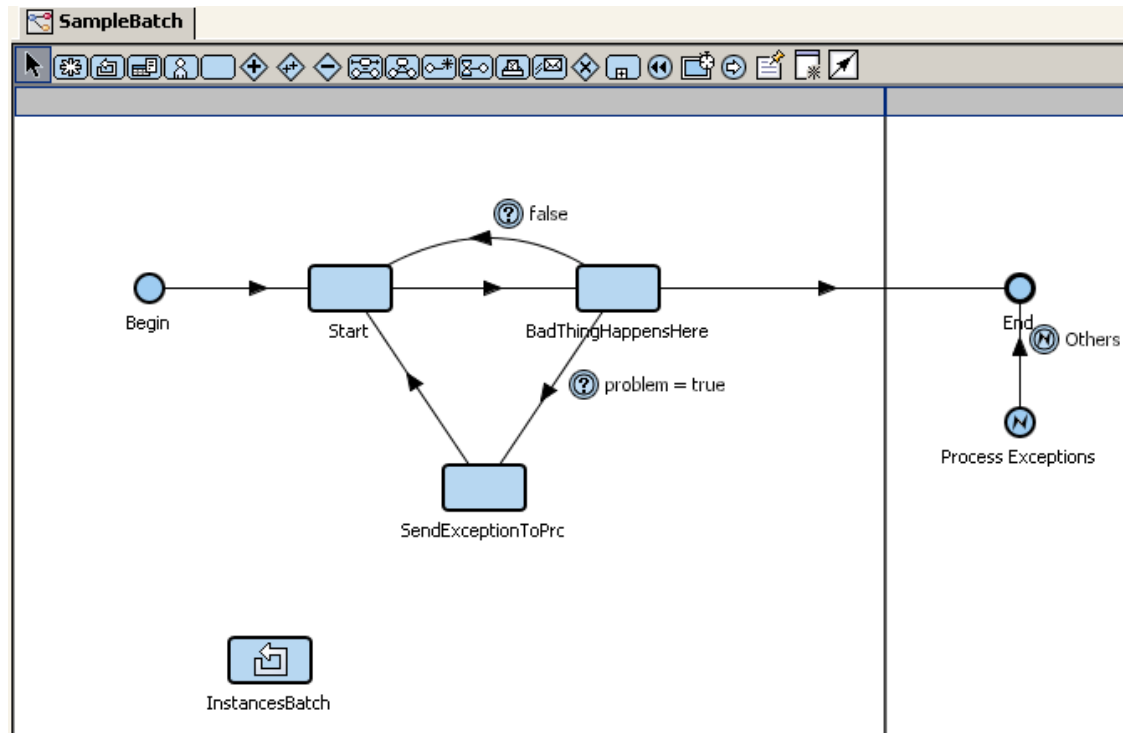
The FuegoBPM's participant name is: **test1**.

Case 01: Process that handles exceptions

In the following example, there are two processes:

- **SampleBatch:** This is a batch process in which certain exceptions might occur, but the instance needs to continue although something goes wrong.
- **ExceptionHandler:** This process manages instances created from other processes (in this example, from *SampleBatch*) and takes care of the errors that occur in the other processes.

SampleBatch Process



In this process, instances are generated from a file. The example is set so that all instances *have problems* and generate a new instance in the **SendExceptionToPrc** activity. The new instance is generated in the **Exception Handler** process to be treated by an *Administrator*.

The instance in the **SampleBatch** process continues its flow.

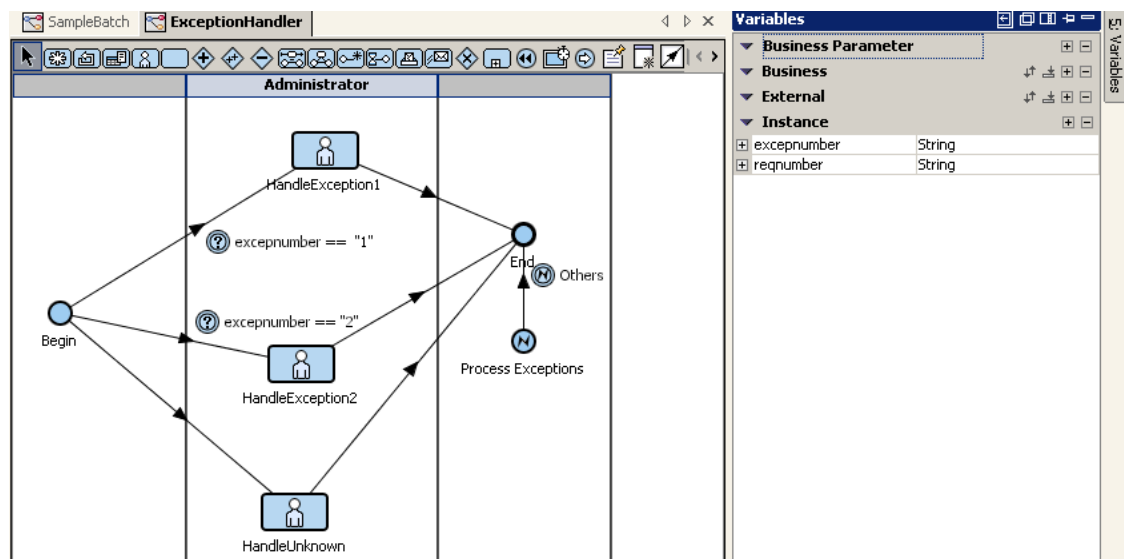
The **SendExceptionToPrc** Method is as follows:

```
args["exceptionNo"] = String(1)
args["requestNo"] = String(111)

exceptionProcessId = organization + "/"
                    + organizationalUnit + "/"
                    + "ExceptionHandler"

create ProcessInstance using processId = exceptionProcessId,
                           parameters = args
```

ExceptionHandler Process



This process handles all kinds of exceptions that are based on the received argument *excepnnumber*.

The 'Argument mapping' dialog box shows the configuration for the 'BeginIn' argument set. It contains a table for input arguments:

Instance variable		Argument
excepnnumber (String)	=	exceptionNo (String)
reqnumber (String)	=	requestNo (String)

At the bottom of the dialog are 'OK', 'Cancel', and 'Help' buttons.

Based on this parameter, the exception is submitted to different

interactive activities to resolve the problem.

Find the project **ExceptionCase01.fpr** in FuegoBPM's installation directory, in studio/samples to follow the example.

The file.txt that requires this project to run has the following format:

integer, string

For example:

1111,John Smith

2222,Karen Owens

3333,Frank Weinz

4444,Alex Rivera

The FuegoBPM's participant name is: **test1**.

Global Automatic Activities and Exceptions

The Global Automatic activity does not have any association to instances (there is no instance pending in the activity). So, when dealing with exceptions, the behavior is a little different from that of other activities. The following two cases explain the processing of exceptions within Global Automatic activities.

Case 1 : When an unexpected exception occurs (like a `NullPointerException`), the GAA will run at the next scheduled time. For example, if the activity is of a polling type with properties defined to run the BP-Method with a two (2) minute gap, after the two minutes have elapsed the BP-Method will run again.

Case 2: When a `UserDefinedException` is thrown, the Global Automatic will not run anymore. In order to make the BP-Method start processing again, you need to restart the FuegoBPM Enterprise Server.

For both cases, the `retryIn` or `retryAt` variables have no effect because the BP-Method has failed. In consequence, the BP-Method is not complete and its commit is not done.

The first case could be assimilated to a `FileNotFoundException` exception or some other external problem that can be externally fixed. For example, you may need to save the file in order to retrieve it next time the Global Automatic BP-Method is run.

In the second case, a user-defined exception is thrown with the full intention to stop the processing of the BP-Method in a Global Automatic activity. A log is added to the server log explaining that the exception was thrown.

Note



If the second case must be used, the developer responsible for designing the process must include a BP-Method in the process to check and make sure that the Global Automatic activity has executed. If a user-defined exception is thrown in a Global Automatic, the FuegoBPM Server must be restarted.

Exceptions

FuegoBPM Studio allows you to use and handle exceptions in your methods and processes.

For further information about handling exceptions:

- in your code
 - Compound Statement
 - Throw Statement
- in your processes

- Process Exception Flow
- Exception Transition
- Exception Handling

The following list includes some of the most common Java exceptions that can appear:

- ClassNotFoundException
- NumberFormatException
- NegativeArraySizeException
- ArithmeticException
- CloneNotSupportedException
- NullPointerException
- ArrayStoreException
- IllegalThreadStateException
- StringIndexOutOfBoundsException
- Exception
- ArrayIndexOutOfBoundsException
- IllegalAccessException
- UnsupportedOperationException
- InterruptedException
- InstantiationException

- `IllegalArgumentException`
- `IndexOutOfBoundsException`
- `RuntimeException`
- `SecurityException`
- `IllegalMonitorStateException`
- `ClassCastException`
- `IllegalStateException`

ClassNotFoundException

The *ClassNotFoundException* is thrown when a required class is not found at run-time. This may happen because a jar file is missing.

NumberFormatException

The *NumberFormatException* is thrown to indicate that the application has attempted to convert a string into one of the numeric types, but the string does not have the appropriate format.

NegativeArraySizeException

The *NegativeArraySizeException* is thrown if an application tries to create an array with negative size. Since arrays can be empty, but not negative, this exception is thrown.

ArithmeticException

The *ArithmeticException* is thrown when an exceptional arithmetic condition has occurred. For example, a division by zero on integer numbers (Real numbers support division by zero and return NaN).

The following example shows the use of a Compound Statement to display the exact arithmetic exception that has occurred:

```
do
  divisor as Int = 0
  dividend as Int = 26
  result as Int
  result = dividend / divisor
on e as Java.ArithmeticException
  display e.message
end
```

CloneNotSupportedException

The *CloneNotSupportedException* is thrown to indicate that the *clone* method has been called to clone an object but the object's class does not implement the *Cloneable* interface. In other words, the object could not be cloned.

NullPointerException

The *NullPointerException* is thrown when an application attempts to use **null** in a situation in which an object is required.

ArrayStoreException

The *ArrayStoreException* is thrown when you attempt to store the wrong type of object in a native array (certain arrays returned by an introspected Java component). For example, this is thrown if you try to store a decimal variable into a String-type array. This is similar to a *ClassCastException* but for array elements.

IllegalThreadStateException

The *IllegalThreadStateException* is thrown to indicate that a thread is not in an appropriate state for the requested operation.

StringIndexOutOfBoundsException

The *StringIndexOutOfBoundsException* is thrown when accessing a character inside a string, and the requested index is out of bounds.

Exception

The *Exception* exception and its subclasses are a kind of *Throwable* and indicate conditions that a reasonable application might want to catch. Basically, *Exception* is an extension of *Throwable*, which extends *RuntimeExceptions*.

ArrayIndexOutOfBoundsException

The *ArrayIndexOutOfBoundsException* is thrown when you try to access an array with an illegal index number, e.g.:

```
names as String[]  
names = ["John", "Peter", "Mary"]  
  
display names[3]
```

IllegalAccessException

The *IllegalAccessException* is thrown when an application tries to access a class or a method, but the currently executing method does not have enough privileges to do so because the class or method is not public or it is in another package.

This exception might happen if you change the access flags of an introspected component and replace the jar without re-introspecting it.

UnsupportedOperationException

The *UnsupportedOperationException* is thrown when you have requested an operation that is not supported.

InterruptedException

The *InterruptedException* is thrown when a thread is waiting, sleeping or otherwise paused for a long time and another thread interrupts it

using the interrupt method in Thread class.

InstantiationException

The *InstantiationException* is thrown when an instance of a certain class cannot be created.

IllegalArgumentException

The *IllegalArgumentException* is thrown to indicate an argument passed to a method is illegal.

That could mean that:

- an argument is out of range
- a certain argument is null but it is required

IndexOutOfBoundsException

The *IndexOutOfBoundsException* is thrown to indicate that an index is out of bounds.

RuntimeException

The *RuntimeException* is the superclass of all exceptions that may happen at runtime because of some unexpected condition.

The following exceptions are subclasses of RuntimeException:

- ArithmeticException
- IndexOutOfBoundsException
- NegativeArraySizeException
- NullPointerException

- `ArrayStoreException`
- `ClassCastException`
- `IllegalArgumentException`
- `SecurityException`
- `IllegalMonitorStateException`
- `IllegalStateException`
- `UnsupportedOperationException`

SecurityException

The *SecurityException* is thrown by the security manager to indicate some kind of security violation. For example if a method attempts to exit the Java Virtual Machine while running at the server, a security exception will be thrown.

IllegalMonitorStateException

The *IllegalMonitorStateException* is thrown to indicate that a thread has attempted to wait on an object's monitor or to notify other threads waiting on an object's monitor without owning the specified monitor.

ClassCastException

The *ClassCastException* is thrown when you try to store a value into a variable and the type of the value cannot be converted to the type of the variable. This may happen in several situations:

- when narrowing a type using a forced conversion.
- when you try to interface the process with another process or with an external application and you pass the wrong type of

variable. For example, if a subprocess expects variables of string type and your process sends variables of decimal type, there is a type conflict.

IllegalStateException

The *IllegalStateException* is thrown when an attempt to call a method at a time when it is illegal to do so.

Chapter 11. Compensation Handling

Compensation Handling

Compensation handling reverses the effect of a **completed** Activity (simple activity or a group of activities). A **Compensation Handler flow** is needed when an activity cannot be automatically reversed through the transaction's automatic rollback.

A Compensation Handling flow is very similar to Exception Handling. An **exception** can trigger a compensation flow.

- To define a compensation handling flow, use the **compensate transition**. The source of the transition is the activity to be compensated and the target is the first activity in the compensation handling flow. As the compensation handling flow is independent from the main process flow, it cannot have transitions back into the main flow.
- Compensations are executed on **completed** activities, whether transactional or not. If the activity is not completed (for example, because an exception occurred before it finished), its compensation handler flow will not be triggered.

What is the difference between a Compensation Flow and a BP-Method rollback ?

A Compensation flow is only invoked after an activity has successfully executed. The BP-Method rollback, on the other hand, is executed only after an activity's task has **FAILED**.

Executing the compensation handler flow

- The **compensation handler flow** can be invoked from the **exception handler flow** or a **compensation handler flow**. Compensations can be invoked programmatically through a **Compensate activity**.

Compensation can apply to a **single** activity or to a **group** of activities. Within a group, if the compensate activity needs to rollback only one of its activities, then you have to **select the activity to compensate (Figure 2)**. If it has to compensate either a single activity or a complete group of activities leave the Compensate dropdown without an activity selected (**Figure 1**) .

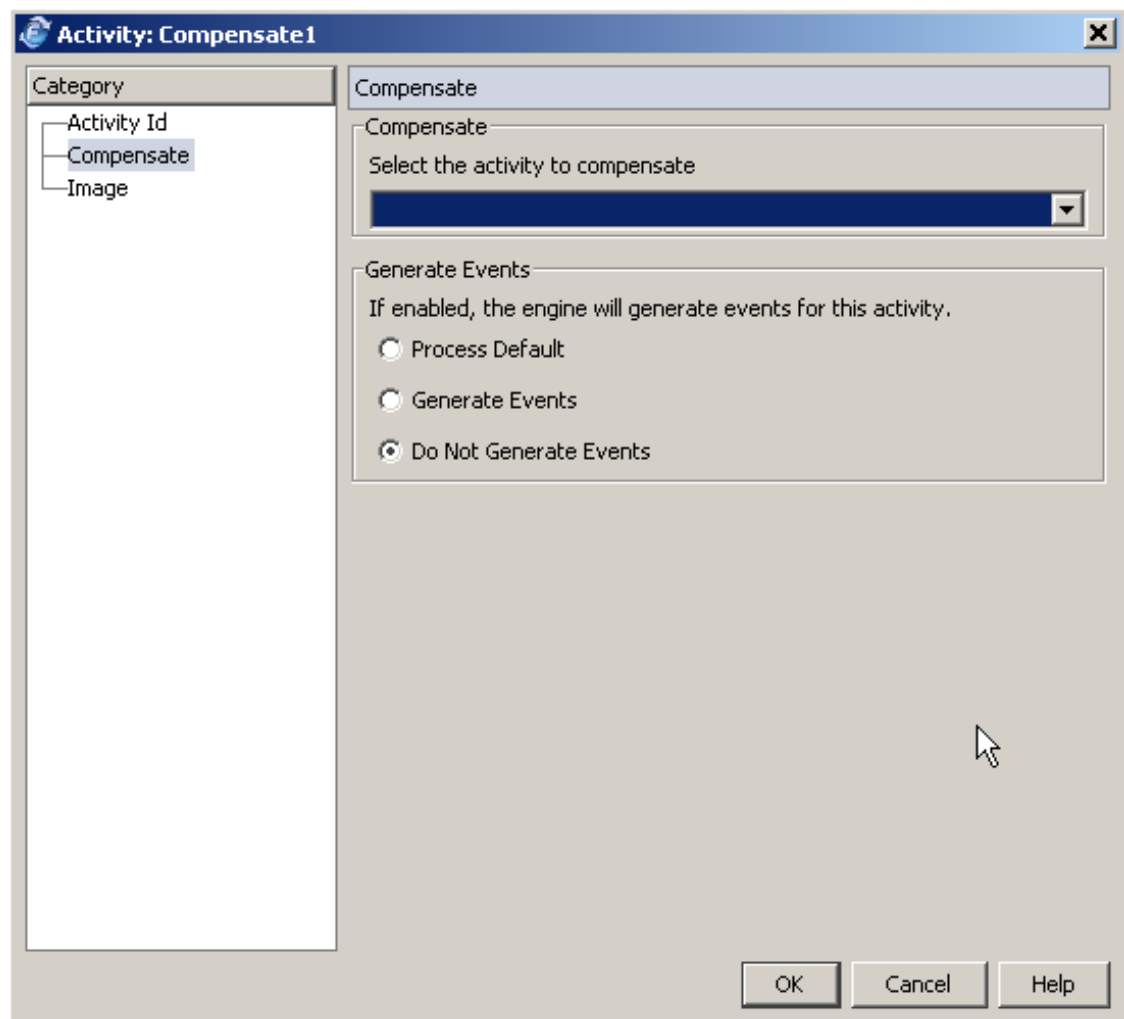


Figure 1--No reference to a specific activity in the Compensate Activity Property.

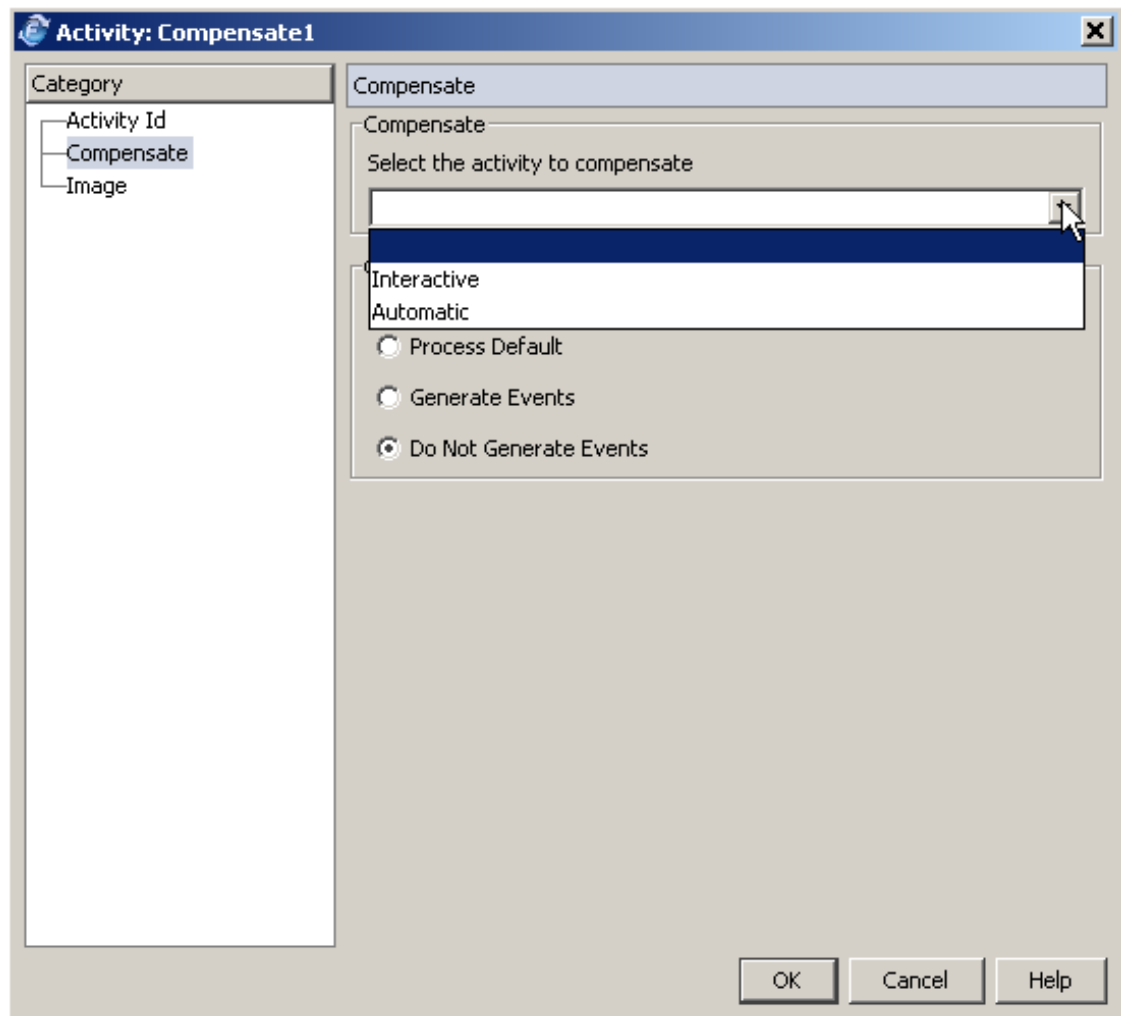


Figure 2--Specific activity in the Compensate Activity Property.

This last case applies to a Group compensation so you must connect the Compensate activity to the Group using a transition.

Default Compensate Handler Flow

- The default compensation handler for a group is to invoke all the

inner compensation handlers in the reverse order in which the original activities have been executed. Therefore, if you define a group but you do not define a compensation handler flow, then the **FuegoBPM Server** assigns the **Default Compensation Handler Flow** (invoke the compensation handler flow(s) of all completed activities inside the group in the reverse order in which they were executed).

- **Example (Figure 3):** If Group3 's compensation is invoked, it will run the compensation of *Automatic 4*, *Group2* and *Group1*. Since the compensation for *Group1* is not explicitly defined, when the *Compensate* activity in *Group3* calls the compensation for *Group1*, the **FuegoBPM Server** will call the compensation for *Interactive2*, *Automatic1* and *Interactive1* in this order.

Some tips

- The instance cannot be sent back by the BP-Method or by the Participant (through the Work Portal) from the **Compensation Handler Flow**.
- A compensation can be added to any activity, except Begin and End activities.
- If the compensation flow for any activity is being called but this activity had failed, the call is ignored.
- When the instance reaches the end of the compensation flow, it will return to the activity that called the compensation flow.

Process using Compensation

- When executing a compensation handler or an exception handler for a given group, you can only invoke compensation handlers of the first level of activities inside this group.
- **Example (Figure 3):** *Compensate* activity in *Group3* exception flow calls compensation for activity *Automatic4*, *Group2* and *Group1*.
- If the property *select the activity to compensate* does not have a specific activity, then it calls the compensate flow of all **completed** activities inside the group in the reverse order in which they were executed.
- **Example (Figure 3):** If activity *Automatic4* fails, an exception is thrown. Therefore, *Group3* exception flow (**Automatic9-->Compensate**) is invoked. In consequence, the *Compensate* activity will call the compensation of *Group2* and *Group1*. In this example, the *Automatic4* compensation flow is not called because compensation is called only in completed activities.

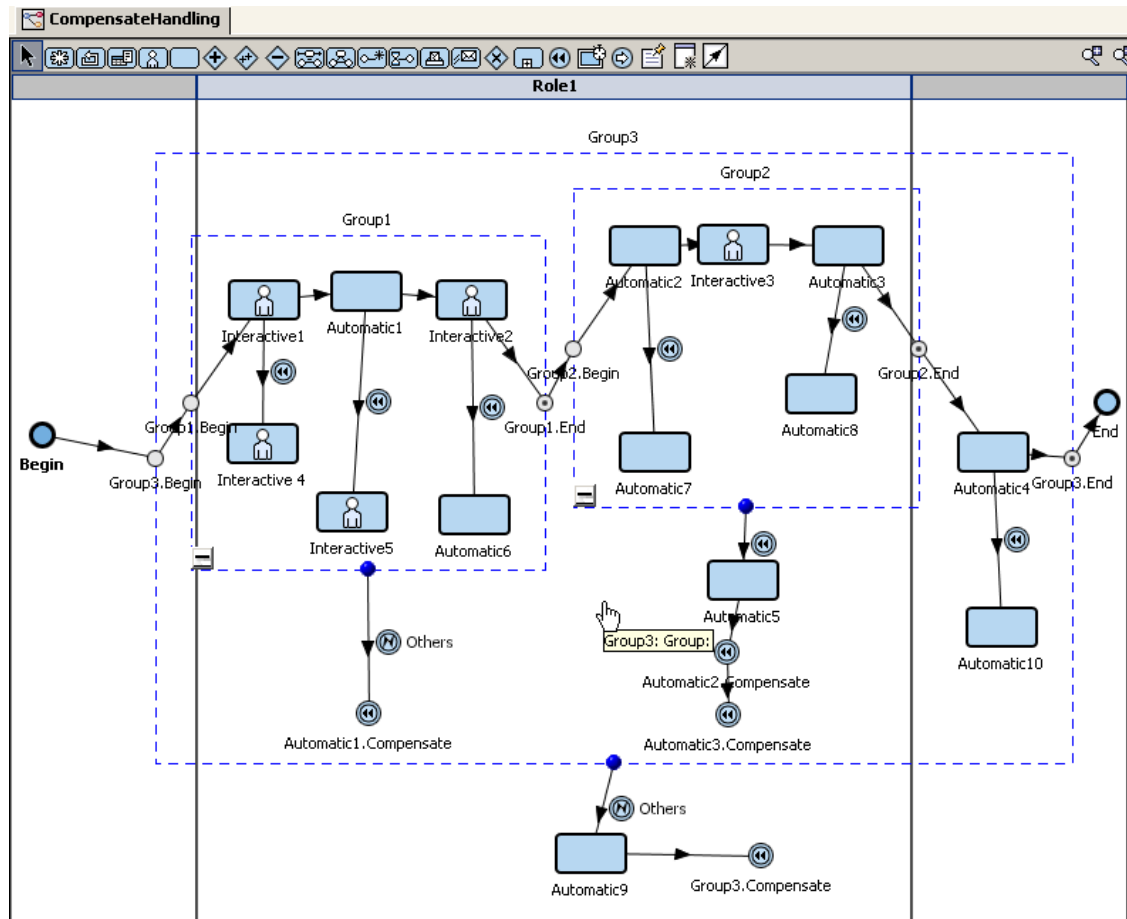


Figure 3 Process using Compensation.

Compensate Activity

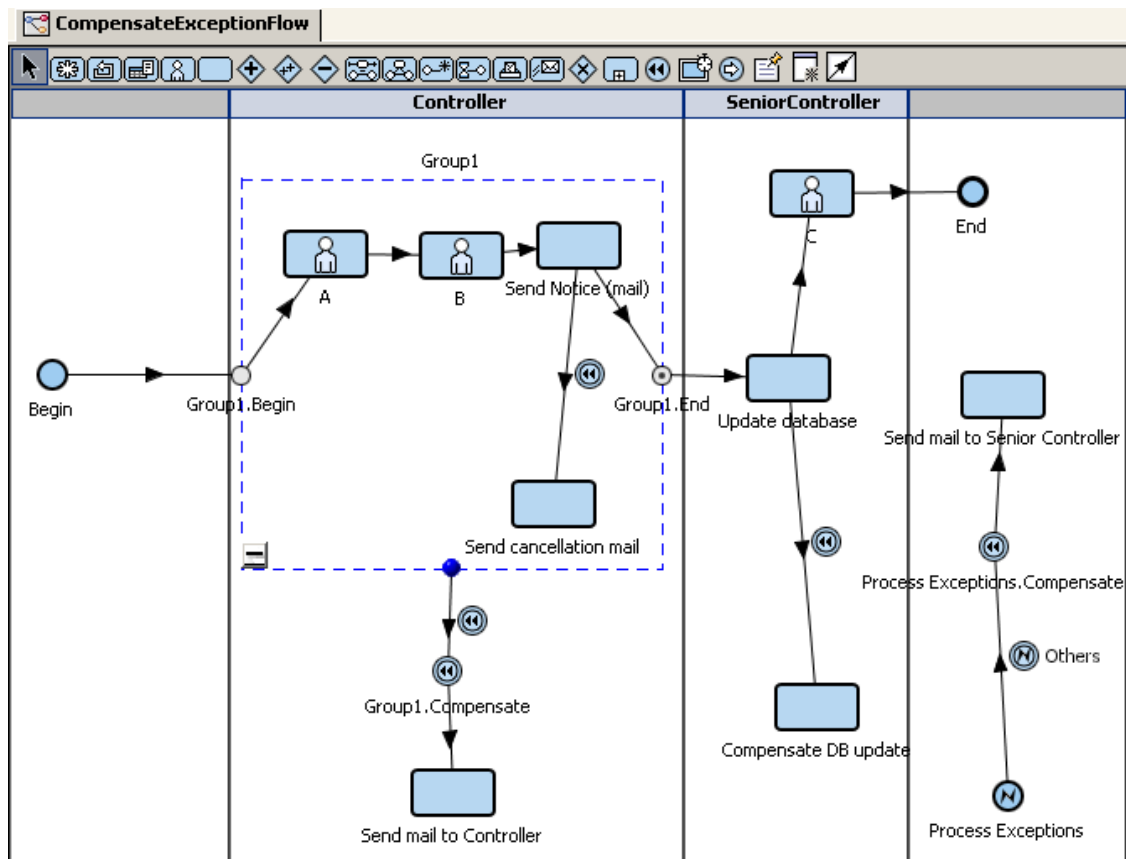
Compensation is the act of reversing the effect of a **completed** Activity (a single activity or a group.)

The **Compensate Activity** triggers the compensation.

The Compensate activity can be part of an Exception Handling Flow or a Compensation Handling Flow.

The *Compensate activity* is always part of a group's Exception Handling Flow or a Compensation Handling Flow. When the instance reaches this activity, the compensate activity will trigger all **inner Compensation flows** within that group.

1. The Compensate activity is executed as any other activity within a flow--when the instance arrives to it.
2. The Compensate activity always triggers an inner compensation flow or compensation activities (those reached through a Compensate transition).







In the example above, the compensate activity *Process Exception.Compensate* within the **Exception Handling Flow for the Process**, when reached, will trigger the corresponding *Compensation Handling flow* for all the activities and groups within the Process (only for those activities that have been successfully completed).

For example, if an exception occurs in *activity C*, the compensation will execute:

- *Compensate DB update*
- **Group1's compensation flow** beginning with
 - *Group1 Compensate* activity that will compensate all inner activities for Group1:
 - *Send cancellation mail* (as "Send notice (mail)" compensation)
 - and continuing with the Group1's compensation flow, activity *Send mail to Controller*

Compensate activity characteristics

Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			

Work Portal and the Compensate activity

Compensate activities do not appear in Work Portal because the Compensate activity does not require any end user intervention. They are automatically executed.

Roles and Automatic activities

Compensate activities can appear either in automatic roles or in user defined role types. However, since Compensation activities have no end-user interface, if it is placed in a user-defined role, it still will not be shown to the end user in Work Portal.

Variables and Automatic activities

No variables are available.

Preconditions

The Compensate activity is activated by an incoming compensate transition.

Post-conditions

When the Compensate activity is executed, all Compensation flows for all inner activities are triggered.

If the Compensate activity is part of an Exception flow or a Compensation flow, after its execution, the next activity in the flow is executed. Once the end of the flow has been reached, the instance will return to the activity that called the exception / compensation flow.

Properties

Activity ID

See Creating an Activity.

The label will automatically populate.

Compensate

The compensation of an activity means that the instance will be routed to its Compensate Handler Flow.

Select the activity to compensate: If the Compensate activity is applying to a group (the compensate activity is part of an Exception flow or Compensation flow), you can select one of the inner activities within the group to which the compensate will revert. If an activity is selected, when the group's compensation is triggered, it will only revert all actions performed by that activity (if the activity was **completed**). No other activities within the group will be

compensated.

If it is not selected, all **completed** activities within the group are compensated in the reverse order in which they were executed during the process flow. For example, if you have three activities that are executed in the following order:

1. Activity 1
2. Activity2
3. at the end Activity3, the Compensation will revert Activity3 first, then Activity2 and finally Activity1

Generate Events: See Generate Events.

Transitions and Compensate activities

One incoming compensate/exception transition is required and one or more outgoing transitions can be added in order to continue the compensation/exception flow to another activity.

Tasks

No Tasks are available.

Business Process Method considerations

No BP-Methods are available.

When do you have to use a Compensate Activity?

- In the event that you need to explicitly revert an activity or group of activities actions. If you define a group and, for some reason something fails, you can manage the compensation of the

completed activities. See Compensate Handling for further information.

Compensate activity examples

See Compensate activity examples

Compensate Activity Examples

Using Compensation

Case 01: Simple Compensation Examples

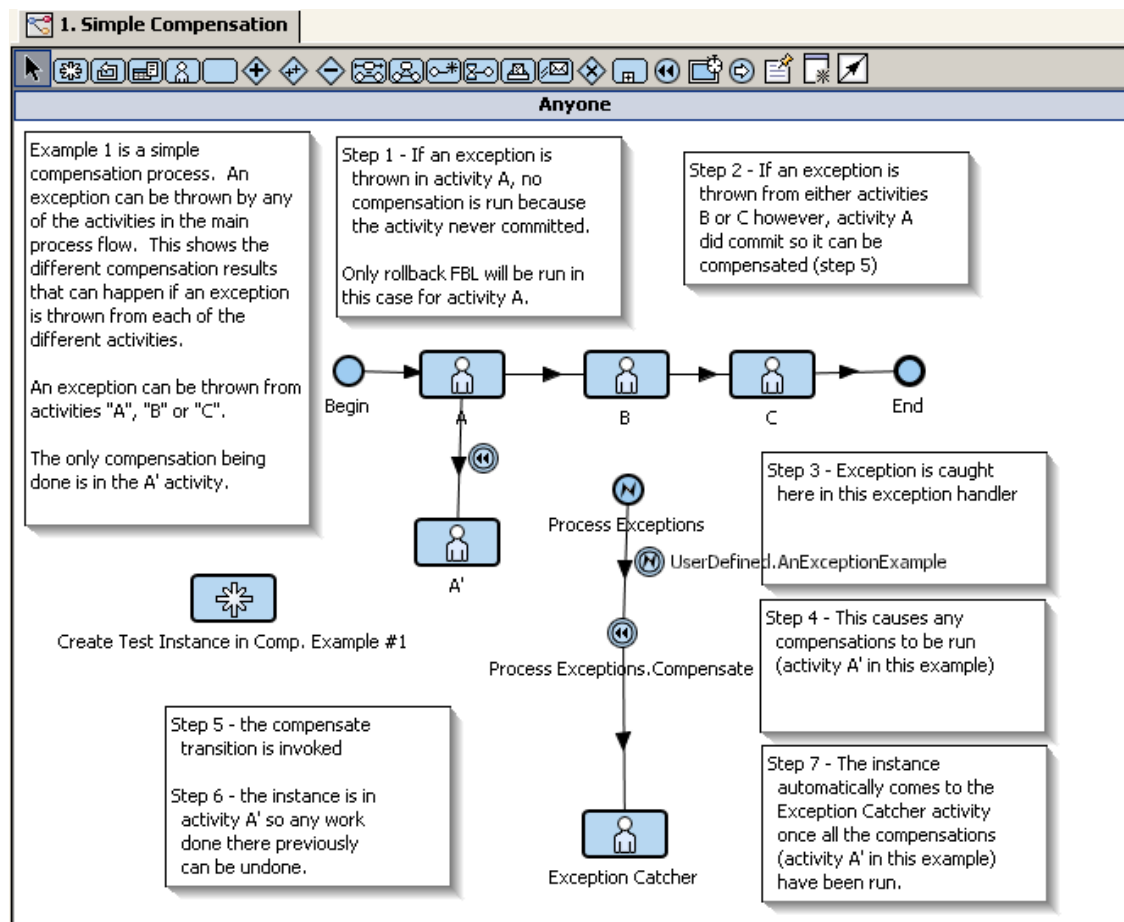
The following examples show different scenarios on how compensation can be handled:

1. Simple Compensation
2. Multiple Compensations
3. Compensation transition with a Group
4. Compensation with a Group that has an exception transition

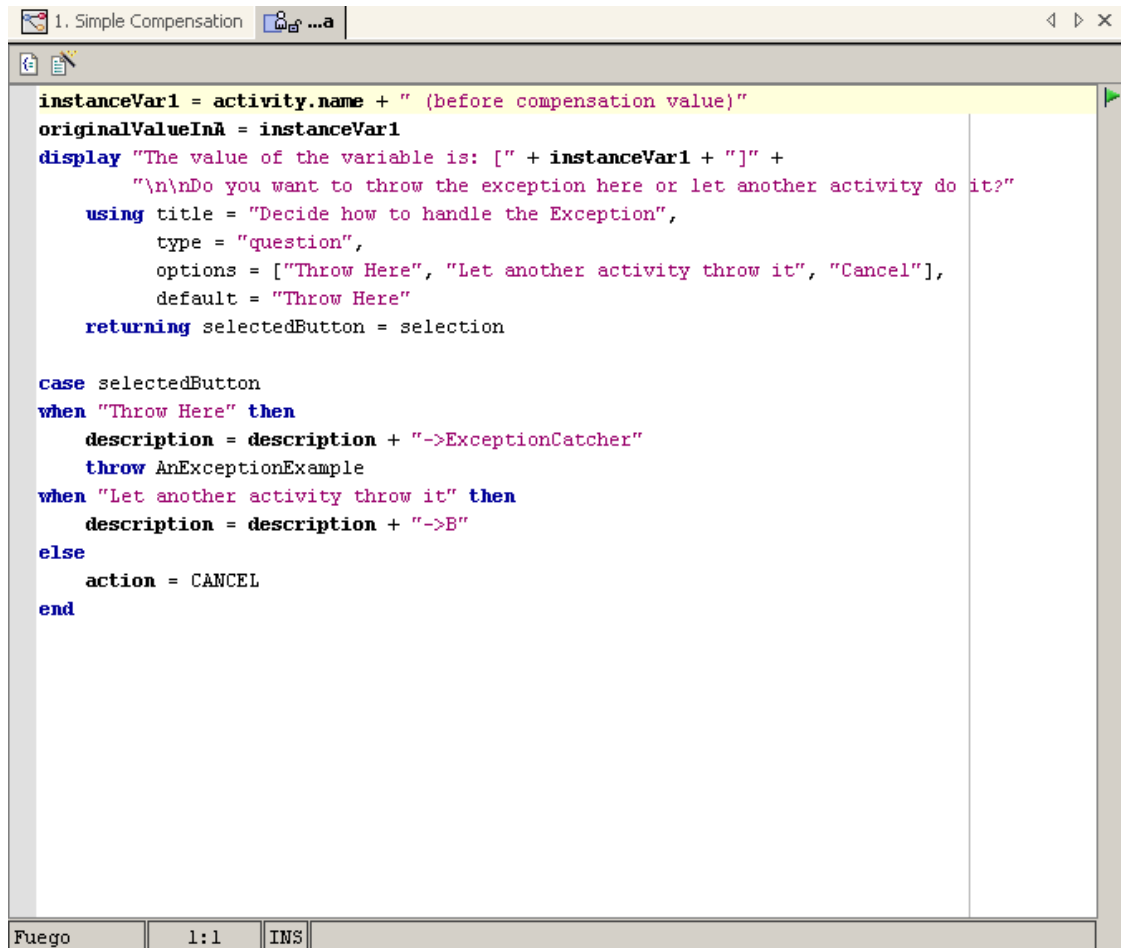
1. Simple Compensation

This process shows what can happen when an exception is thrown from each of the different activities (**A**, **B** or **C**) and the different compensation results.

The only compensation being done is in the **A'** activity.



The **A** interactive activity main task is to choose whether you throw the exception in the **A** activity or let the instance keep on flowing and produce it later in the process.



```

instanceVar1 = activity.name + " (before compensation value)"
originalValueInA = instanceVar1
display "The value of the variable is: [" + instanceVar1 + "]" +
  "\n\nDo you want to throw the exception here or let another activity do it?"
  using title = "Decide how to handle the Exception",
    type = "question",
    options = ["Throw Here", "Let another activity throw it", "Cancel"],
    default = "Throw Here"
  returning selectedButton = selection

case selectedButton
when "Throw Here" then
  description = description + "->ExceptionCatcher"
  throw AnExceptionExample
when "Let another activity throw it" then
  description = description + "->B"
else
  action = CANCEL
end

```

Fuego 1:1 INS

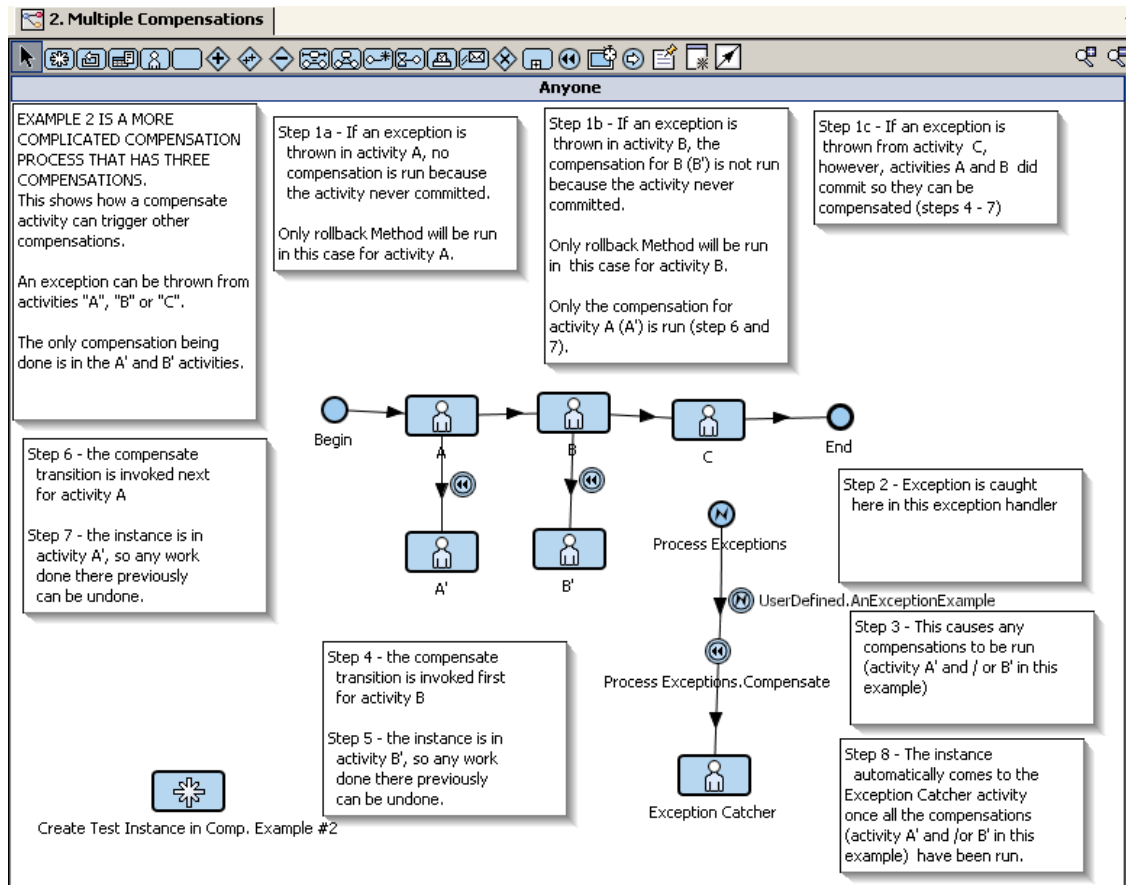
If the exception is thrown in **A**, the instance flows to the Process Exception flow and NO compensation is done as the activity never committed.

If no exception is thrown the instance flows to the **B** activity.

If the exception is thrown in **B**, the instance flows to the Process Exception flow and the **Compensation activity** is run. As activity **A** did commit, **A'** activity is run following the **compensation transition**.

2. Multiple Compensations

The following example shows how a compensate activity can trigger other compensations.



The **Process Exceptions.Compensate** activity can trigger either **A'** and /or **B'** depending on if **A** or **B** activities were committed.

If the exception is thrown in the **C** activity, the instance flows to the Process Exception Flow and the **Compensate** activity is run. As activities **A** and **B** have been committed, first **B'** activity is run and the **A'** activity is run.

In Web Portal, the **instance description** changed and shows the different activities it went through:

FUEGO Work Portal Welcome, test1 Search - Options - Help - Logout

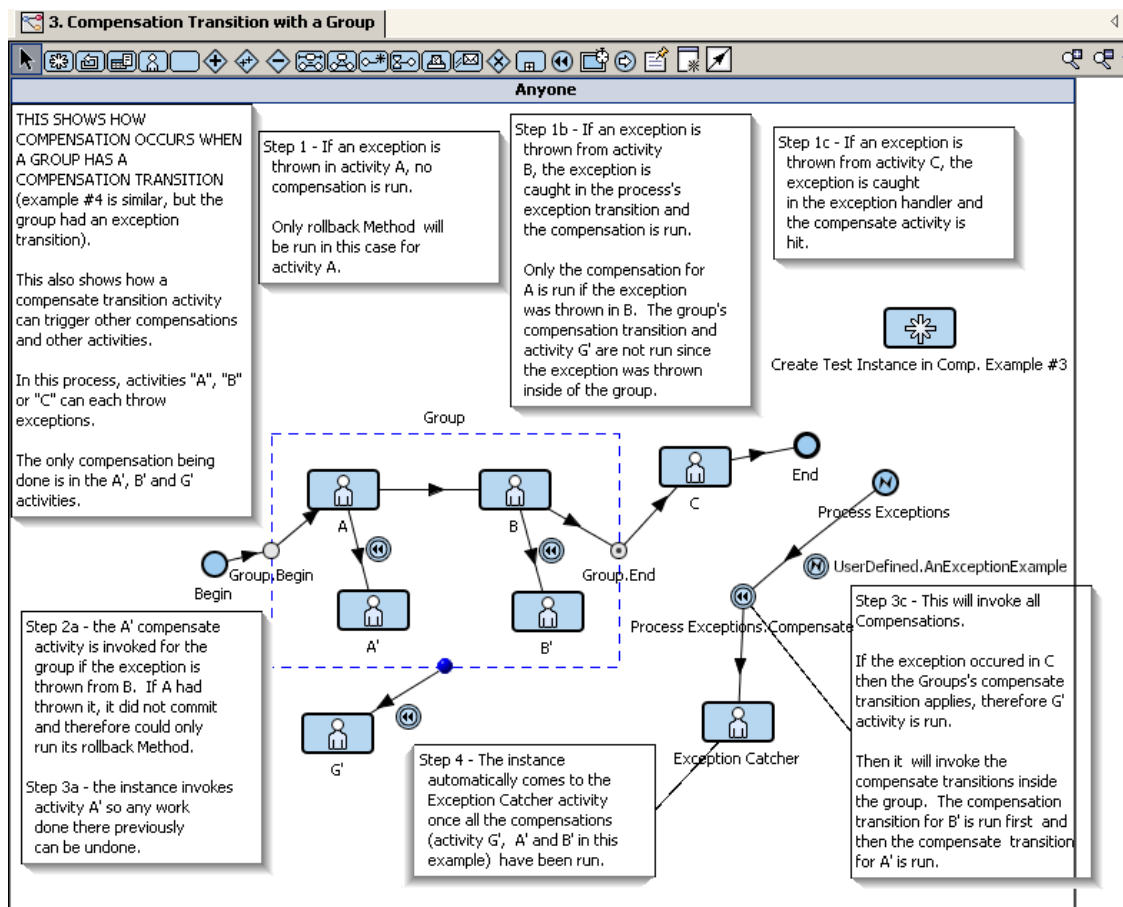
2. Multiple Compensations > Inbox Showing 1-1 of 1

Description	Activity	State	Received	Deadline	Participant
A->B->C->ExceptionCatcher->B'->A'	A'	Exception	6:21 PM		

FuegoBPM™ - Work Portal

3. Compensation transition with a Group

This process shows how compensation occurs when a group has an Compensation transition.



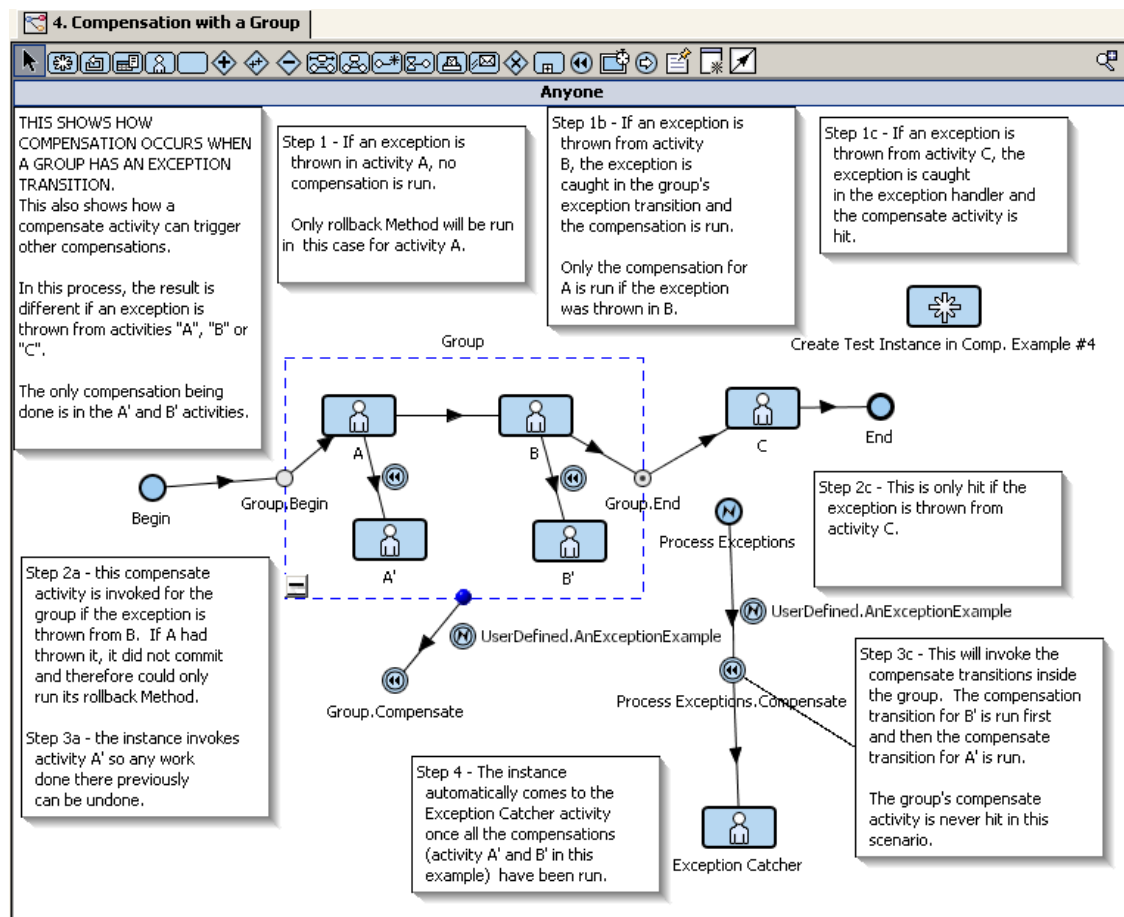
Notice that **G'** activity is run only if the exception is thrown in **C** activity.

The instance flows to the Process Exception Flow and triggers the **Process Exceptions. Compensate** activity.

The first compensation applies to the **Group** and **G'** is invoked following the compensate transition. Then the Group inner compensation are executed starting with **B'** and finally **A'**.

4. Compensation with a Group that has an exception transition

This process shows how compensation occurs when a group has an exception transition.



If **A** activity throws an exception, it is caught by the Group Exception handling and the **Group. Compensate** activity is run. But as no Compensation can be done (**A'** activity is never executed as **A** was not committed), the instance flows to the next activity after the

Group, therefore to **C** activity.

If **B** activity throws an exception, it is caught by the Group Exception handling and the **Group. Compensate** activity is run. Inner compensation are run. Therefore, the instance flows to **A'** activity (**B'** activity is never executed as **B** was not committed).

If **C** activity throws an exception, it is caught by the Process Exception handling and the **Process Exceptions. Compensate** activity is run. Inner compensation are run. The Group has no Compensation transition so all Group's inner compensation are triggered beginning with **B'** and then **A'**.

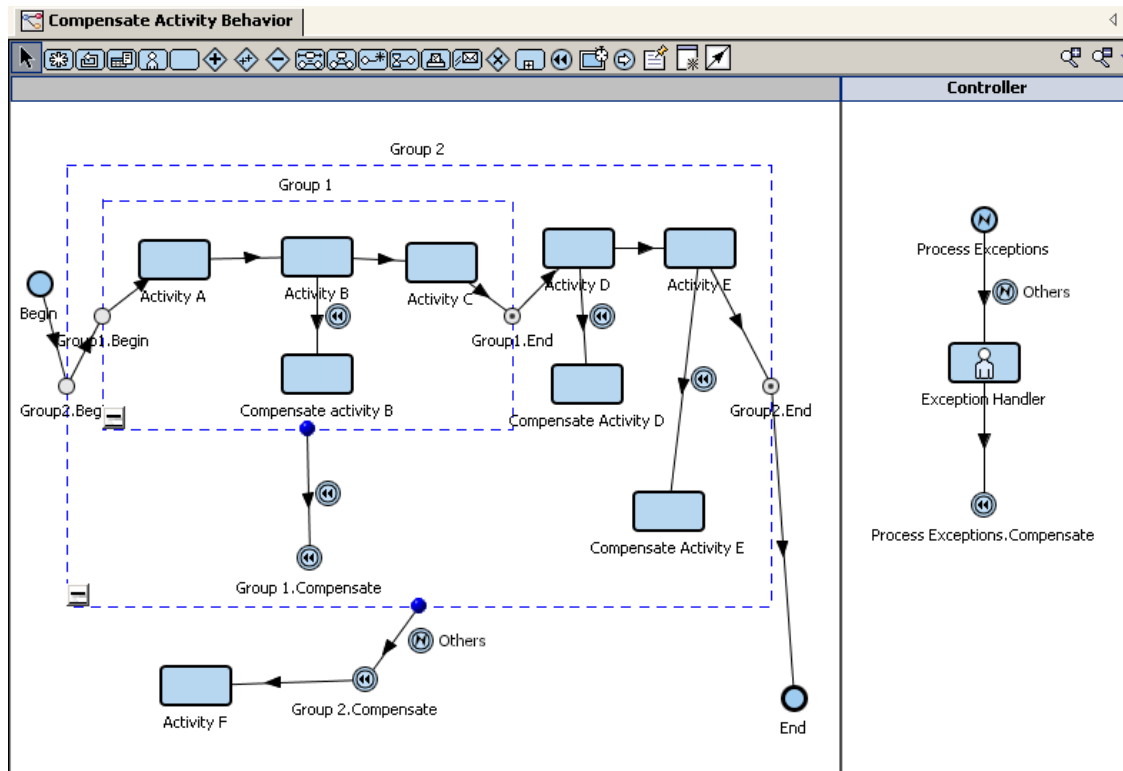
Find these examples in the **CompensationCase01.fpr** project in FuegoBPM Studio's installation directory, in studio/samples.

The FuegoBPM's participant name is **test1**.

Understanding Compensate activities behavior

Compensate activities are always part of a **Compensation Handling Flow** or an **Exception Handling Flow** and they **trigger all inner compensation flows**

In the following example, we can see 2 **Compensate activities** (**Group 2.Compensate** and **Group 1.Compensate**). The first one in an Exception Handling Flow and the other in a Compensation Handling flow. The Compensate activities act when the instance reaches the activity and trigger the inner compensation flows or compensation activities (those reached through a Compensate transition).



In the example above, if an exception occurs within *Activity E*, the instance flows to Group 2's Exception Handling flow.

1. The first activity within this exception flow is the **Group 2.Compensate** activity.
2. This activity triggers all inner compensation flows.
 - a. The first compensation to execute is *Compensate Activity E*, but as *Activity E* finished with an exception and was not successfully completed, then no compensation is executed.
 - b. The second compensation to execute is *Compensate Activity D*.
 - c. The third compensation is to execute the Compensation Flow for *Group 1*. And the first (and only activity) within it is the **Group 1. Compensate** activity that triggers all inner compensations within Group1.

- i. Finally, and as part of Group1's inner compensation, *Compensate Activity B* is executed.
3. After the **Group 2, Compensate** activity is completed and **Activity F** is run.

Chapter 12. HTML Process API

HTML Process API

This feature provides you with the capability of executing actions and tasks on process instances from an HTML page using Work Portal functionality. They are:

- Global activities execution
- Instance actions execution
- Instance tasks or item execution

Required Configuration

You have to define a participant named *guest* .

This participant becomes the anonymous FuegoBPM Studio user to log in to Work Portal. Its definition is not mandatory, and it is only used when executing Global Creation activities. If this user is not defined, a user and password will be required at execution time.

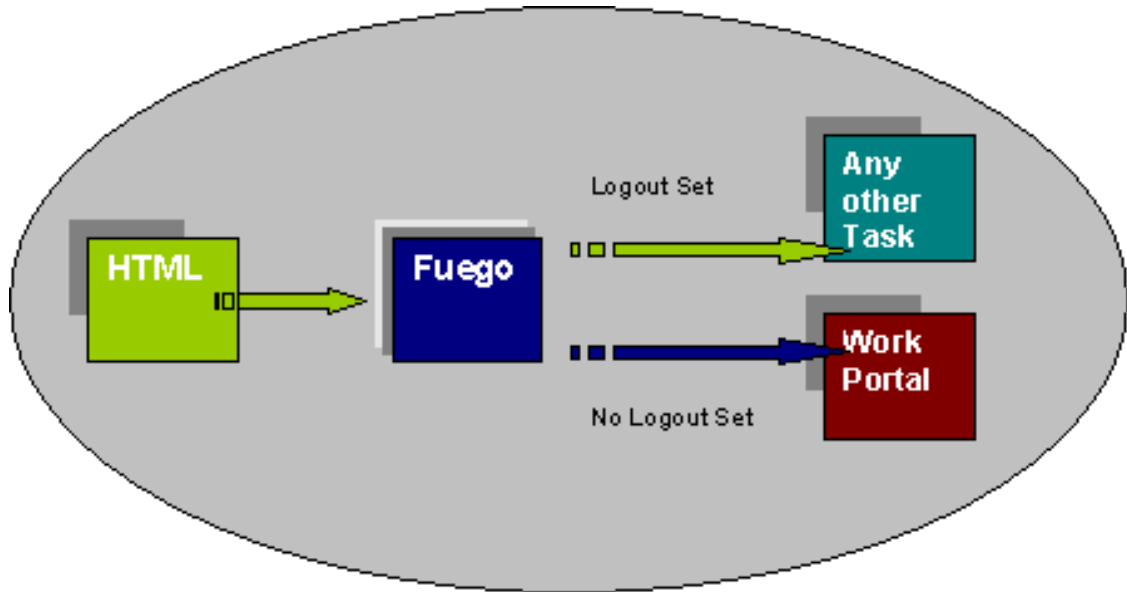
Description

As mentioned previously, this feature provides you with the capability of using Work Portal functionality through a URL and parameters.

Using the non-exclusive sessions (anonymous guest user) or his own (depending on the function), the final user logs in to Work Portal and performs the programmed action, task, activity. Logging out when finishing will depend on the established configuration.

Using this functionality the developer may, from his HTML page, connect and perform actions on a FuegoBPM Studio process through Work Portal and then continue with another task outside FuegoBPM

Studio. Likewise, the developer may let the user stay inside Work Portal to go on working.



Work Portal invocation and execution of process activities, actions or tasks from an HTML page must always be done from an HTML Posting Form block. The structure of this block varies depending on what is being done.

Non-exclusive sessions / Anonymous Participant

The anonymous participant configured as the guest participant in the Organization is used to execute actions that do not require an exclusive session. That is why the non-exclusive session or anonymous participant is used only for the execution of Global activities. Its definition is not mandatory.

If the anonymous user is used to log in and to perform any activity in Work Portal, the Audit Trail will show that this participant was the one that executed the action (only valid for creation).

In order to be able to use the anonymous participant in the execution of any Global activity, it must belong to the role that

matches the abstract role of the process in which the activity is defined.

The combination of two attributes within the form block code in the HTML page presents specific Portal functionality to end users. These attributes are:

- **fuego.portal.userNESession** - set to true, which requires the use of the non-exclusive session through the anonymous participant, if any, that was configured as a guest in the organization.
- **fuego.portal.logoutURL** - set with a URL, which indicates to log the user out and to go to this URL afterward.

Attribute	Data	Behavior
fuego.portal.userNESession	fuego.portal.logoutURL	
Set to TRUE	Set with a URL	The anonymous user guest defined in the organization is searched for. If it is not defined, a user name and password are required through the login dialog. After the user ends processing, the session is closed and sent to the URL specified in the logoutURL attribute. Note: if the user sets the "Remember user" check box, next time,

Attribute	Data	Behavior
		this login dialog is not opened.
Set to FALSE or not present	Not present	If a user name is required through the login dialog, after the completion of the processing, the user will remain logged into the Web Portal and is able to continue working within it.
Set to FALSE or not present	Set with a URL	Instead of looking for the anonymous participant, the user will always be required through the login dialog.
Set to TRUE	Not present	The anonymous participant will remain logged in to the Web Portal. The final user will be able to process any activity that does not require an exclusive session, such as Global activities. When the final user intends to process any action on an instance, a participant for exclusive session will be required before continuing to work.

Working on Enterprise environment

When you move your project using HTML process API, you need to configure the anonymous participant in the *directory.properties* file. The *guest* user is only valid in FuegoBPM Studio environment.

1. Go to the file located at :
\$INSTALLATION_DIRECTORY\\webapps\\portal\\WEB-INF\\directory.properties
2. Set the following properties:

```
directory.default.preset.portal-anonymous.participant=guest
directory.default.preset.portal-anonymous.
    participant_password=guest
```

Be sure that the user you are configuring in this file as anonymous participant to log in into the FuegoBPM Work Portal has the correct roles assigned.

Do not forget to configure the *non-exclusive session* as well in the *portal.properties* file as in the Studio environment.

User Authentication Steps

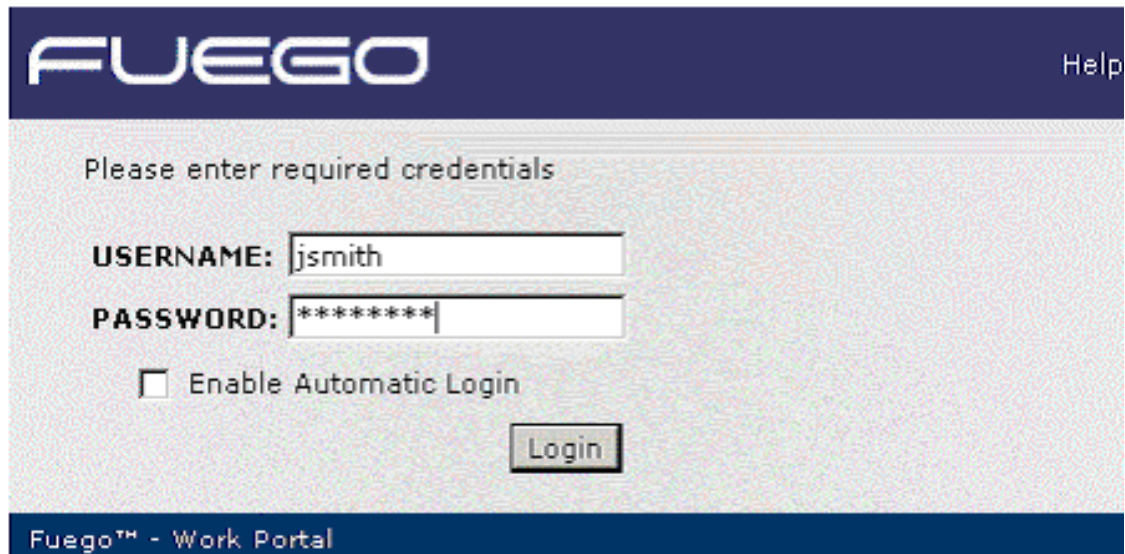
Global Activity execution

If the attribute `fuego.portal.userNESession` is included and set to `true`, the definition of the anonymous user `guest` is searched for in the organization structure (participant and password). If the anonymous user `guest` is defined, this user is the one that is logged into Work Portal.

If no anonymous user has been defined or `userNESession` is set to `false`, then a participant is required to log in through the login dialog.

Action and Item execution

A participant is always required to log in through the login dialog.

The image shows a web-based login interface for 'FUEGO'. At the top, there is a dark blue header bar with the 'FUEGO' logo in white on the left and a 'Help' link in white on the right. Below the header, the main content area has a light gray background. It starts with the text 'Please enter required credentials'. There are two input fields: 'USERNAME:' followed by a text box containing 'jsmith', and 'PASSWORD:' followed by a text box containing '*****'. Below these fields is a checkbox labeled 'Enable Automatic Login'. To the right of the checkbox is a 'Login' button. At the bottom of the form, there is a dark blue footer bar with the text 'Fuego™ - Work Portal' in white.

Global Activities Execution

This option provides the capability of creating process instances or executing Global activities as queries. Global Automatic activities are not available, as this framework is a way of using Work Portal functionality and Work Portal has no way of reaching automatic activities. If a Global Automatic activity is executed, an error will be displayed.

Similar functionality can be simulated with the usage of web services or PAPI. However, the advantages of this feature over the usage of the web services or PAPI are that:

- user interaction is possible; something that cannot be done with web services
- no programming is required; something that has to be done in order to use PAPI

HTML Form block programming

```
<form method="post" ACTION="PortalURL/servlet ">
```

Refer to the common HTML code explanation section 4.6.

The following attributes are generally hidden, as they are the parameters sent to the servlet.

activityId :

```
<INPUT TYPE="hidden" name="activityId"
value="ou/processName/variation-version/activityName">
```

The *activityId* attribute represents the process Global activity that will be executed. The *activityId* value always has the following structure:

- ou: Organizational Unit in which the process has been deployed. If the process is deployed for non-specific ou, leave it blank.
- processName: name of the Process which the Global activity belongs to.
- variation-version, where:
 - variation: variation given when the process was deployed.
 - version: version given when the process was deployed.

Note



Variation and version have to be concatenated by a dash character "-".

- activityName: name of the Global activity to execute.

Example: with Organizational Unit.

```
<INPUT TYPE="hidden" name="activityId"
      value="/BuenosAires/OrderFulfillment#
            Default-1.0/verifyCredit">
```

where

- ou = BuenosAires,
- processName = OrderFulfillment
- variation_version = Default-1.0 (variation = Default, version = 1.0)
- activityName = verifyCredit

Example: without Organizational Unit.

```
<INPUT TYPE="hidden" name="activityId"
      value="/OrderFulfillment#Default-1_0/verifyCredit">
```

where

- ou = ,
- processName = OrderFulfillment
- variation_version = Default-1.0 (variation = Default, version = 1.0)

- activityName = verifyCredit

actionId:

```
<INPUT TYPE="hidden" name="actionId" value="RUN_GLOBAL">
```

The actionId attribute will always have the value "RUN_GLOBAL" when a global activity is executed.

fuego.portal.useNESession:

```
<INPUT TYPE="hidden"  
      name="fuego.portal.useNESession" value="true">
```

The fuego.portal.userNESession attribute set to true enables the intention of using the defined anonymous participant. If it is not defined, then a FuegoBPM participant will be required through the login dialog.

fuego.portal.logoutURL:

```
<INPUT TYPE="hidden"  
      name="fuego.portal.logoutURL" value="logoutOk.html">
```

Refer to the common HTML code explanation section 4.5.

fuego.portal.logoutURL_Error:

```
<INPUT TYPE="hidden"  
      name="fuego.portal.logoutURL_Error"  
      value="logoutError.html">
```

"Refer to the **HTML common code** explanation."

Arguments:

Arguments are included in the attributes list if the Global activity to be executed has defined input arguments. If the activity arguments have default values and they are not sent from the HTML page, the default will be used. The argument attributes in the form block of the HTML page may be hidden or not, according to the application.

There is a rule to name the arguments in the form block.

"arg_" + argumentName

Suppose that the activity receives two arguments, let's name them argument1 and argument2. Then, the name of the attributes in the form block will be: **arg_argument1** and **arg_argument2**.

```
Argument1: <input type="text"
                name="argArgument1">
Argument2: <input type="text"
                name="argArgument2">
```

Finally the known submit button.:

```
<INPUT TYPE="SUBMIT" NAME="Submit"
  VALUE="create a instance">
</form>
```

Note















See the **Examples** section to find a complete example.


Execute an action for an Instance

This option allows the execution of any action that is provided by Work Portal. The final user always has to log in to Work Portal with a participant user name and password.

The list of possible actions is:

Action Name	Work Portal Icon	Description
PROCESS		Processes the activity. If the activity has more than one task, then the instance panel is displayed so that the user can process any or all of its tasks.
COMPLETE		Moves the instance to the next activity in the process. Dimmed until all mandatory tasks for an instance at the current activity are processed.
SEND_TO		Allows the user to assign the instance to a specific user and sends it to the next activity in the process.
BACK		Allows the user to return to the activity where an exception occurred and to continue processing the instance from that point. Note: This button is only available

Action Name	Work Portal Icon	Description
		provided that an exception handler was included in the design of the business process and an exception in the instance invoked the exception handler.
ABORT		Terminates and deletes an instance. All processing ceases and the instance is removed from the business process
SUSPEND		Pauses the instance at the indicated activity.
RESUME		Resumes the instance and allows it to continue running.
GRAB		Sends an instance to a Grab activity.
UNGRAB		Releases an instance from a Grab activity.
SELECT		Assigns the instance to the participant for that activity and blocks other users from processing the instance.
UNSELECT		Unselects the instances that the participant has selected to work.
SELECT_ITEM		The participant selects a task or item within an activity for a specific instance to work on it.

Action Name	Work Portal Icon	Description
UNSELECT_ITEM		The participant frees the task on an instance that had been previously selected.

HTML Form block programming

For the lines below, please refer to the **HTML common code** explanation section.

```
<form method="post" ACTION="servlet/instanceActions">
<INPUT TYPE="hidden"
      name="fuego.portal.logoutURL" value="logoutOk.html">
<INPUT TYPE="hidden"
      name="fuego.portal.logoutURLerror"
      value="logoutError.html">
```

In general, the attributes described below are hidden, but it will depend on the application.

instanceStampId:

```
<INPUT TYPE="hidden" name="instanceStampId"
      value="instanceStampIdValue">
```

The instanceStampId is the name of the instance. The instanceStampIdValue must be made up of:

- processName
- variation_version, (variation and version have to be concatenated by a dash character "-")

- instanceNumber
- thread (number of copy)
- activityName, for example
"instanceActions/Default-1_0/6/0/firstActivity"

A way of composing this value is using a Method, if possible depending on the process and application. Truncating the value of the instanceID instance variable from the "@" character and concatenating the activity name, described as follows:

```
instanceStampId = instanceId.substring(0,  
    instanceId.indexOf("@")) +  
    "/confirmSubscription";
```

```
<INPUT TYPE="hidden" name="actionId" value="actionIdValue">
```

actionId:

The actionId attribute represents Work Portal functionality intended to execute. The actionIdValue must contain any of the possible actions in the table described in the previous section.

- PROCESS
- COMPLETE
- SEND_TO
- BACK
- ABORT

- SUSPEND
- RESUME
- GRAB
- UNGRAB
- SELECT
- UNSELECT
- SELECT_ITEM
- UNSELECT_ITEM

When Action ID is SELECT_ITEM or UNSELECT_ITEM

```
<INPUT TYPE="text" name="itemId" value="0">
```

Represents the number of tasks comprised within the Activity tasks/items list. The list starts in 0.

When Action ID is GRAB or UNGRAB

```
<INPUT TYPE="text" name="grabActivityId"
value="grabActivity">
```

- **Grab:** represents the name of the grab activity in the process where the instance must be sent to. (A process might contain more than one grab activity.)
- **Ungrab:** represents the name of the grab activity by which the

instance has been grabbed.

Finally, the submit button:

```
<INPUT TYPE="SUBMIT" NAME="Submit" VALUE="do action">
</form>
```

Note



See the **Examples** section to find a complete example.

Item/task execution

This option allows the execution of a task for an instance in an Activity. It is only valid if the activity has at least one defined task.

The execution of an item may send arguments as the execution of global activities. Arguments will be included in the attributes list. The argument attributes in the form block of the HTML page may be hidden or not, according to the application.

There is a rule to name the arguments in the form block.

"arg_" + argumentName

Suppose that the activity receives two arguments, let's name them argument1 and argument2. Then, the name of the attributes in the form block will be: **arg_argument1** and **arg_argument2**.

```
Argument1: <input type="text" name="argArgument1">
Argument2: <input type="text" name="argArgument2">
```

HTML Form block programming

For the lines below, please refer to the **HTML common code** explanation section.

```
<form method="post" ACTION="servlet/ExecutionDispatcher">
<INPUT TYPE="hidden"
      name="fuego.portal.logoutURL" value="logoutOk.html">
<INPUT TYPE="hidden"
      name="fuego.portal.logoutURLerror"
      value="logoutError.html">
```

actionId:

```
<INPUT TYPE="hidden" name="actionId" value="RUN_ITEM">
```

The actionId attribute will always have the value "RUN_ITEM" when an activity task is the one to be executed.

instanceStampId:

```
<INPUT TYPE="hidden" name="instanceStampId"
      value="instanceStampIdValue">
```

The instanceStampId represents the exact identification of an instance in a process.

The instanceStampIdValue has to be made up of:

- processName
- variation-version (variation and version have to be concatenated by a dash character "-")
- instanceNumber

- thread (number of copy)
- activityName, for example
"/instanceActions#Default-1.0/6/0/firstActivity"

A way of composing this value is using a Method statement, if possible depending on the process and application. Truncating the value of the *instanceID* instance variable from the "@" character and concatenating the activity name, as follows:

```
instanceStampId = instanceId.substring(0,  
    instanceId.indexOf("@")) +  
    "/confirmSubscription" ;
```

taskDesc:

```
<INPUT TYPE="hidden" name="taskDesc" value="taskName">
```

The attribute taskDesc must contain the name of the activity task to be executed.

itemId:

```
<INPUT TYPE="hidden" name="itemId" value="0">
```

Represents the number of tasks within the Activity tasks/items list. The list starts in 0.

Finally the submit button :

```
<INPUT TYPE="SUBMIT" NAME="Submit" VALUE="do action">
</form>
```

Note



See the **Examples** section to find a complete example.

HTML common code

This section explains the common code of the HTML form block.

See each option HTML code to contextualize it.

post:

It is recommended to use the "post" method.

```
<form method="post" ACTION="PortalURL/servlet ">
```

ACTION:

ACTION must refer to the ExecutionDispatcher servlet, that is, Work Portal servlet in charge of instances execution.

- PortalURL: varies according to Work Portal to which is required to connect. For example: *http://localhost:8080/portal*
- servlet: depends on the action being executed
 - For **PROCESS, COMPLETE, SEND_TO, BACK, ABORT, SUSPEND, RESUME, GRAB, UNGRAB, SELECT, UNSELECT, SELECT_ITEM, UNSELECT_ITEM** the URL should use the *instanceActions*.

- For **RUN_ITEM** y **RUN_GLOBAL** the URL should use the *executionDispatcher*.
- The complete value for the ACTION attribute could be:

```
ACTION=
"http://localhost:9595/htmlProcessAPI/servlet/
instanceActions"
or
ACTION=
"http://localhost:9595/htmlProcessAPI/servlet/
executionDispatcher"
```

fuego.portal.logoutURL:

```
<INPUT TYPE="hidden" name="fuego.portal.logoutURL"
      value="logoutOk.html">
```

The fuego.portal.logoutURL attribute is not mandatory. And it indicates which is the html to be displayed when finished.

fuego.portal.logoutURLError:

```
<INPUT TYPE="hidden" name="fuego.portal.logoutURLError"
      value="logoutError.html">
```

The fuego.portal.logoutURLError attribute is not mandatory and it indicates which is the html to be displayed in the event that an error occurs. If it is not defined, then the one defined in the logoutURL attribute will be used.

For Examples, please refer to HTML Process API Examples.

HTML Process API Examples

The processes examples and the set of html pages are distributed with FuegoBPM Studio installation. You can find them under the *sample* directory of the installation.

Three options

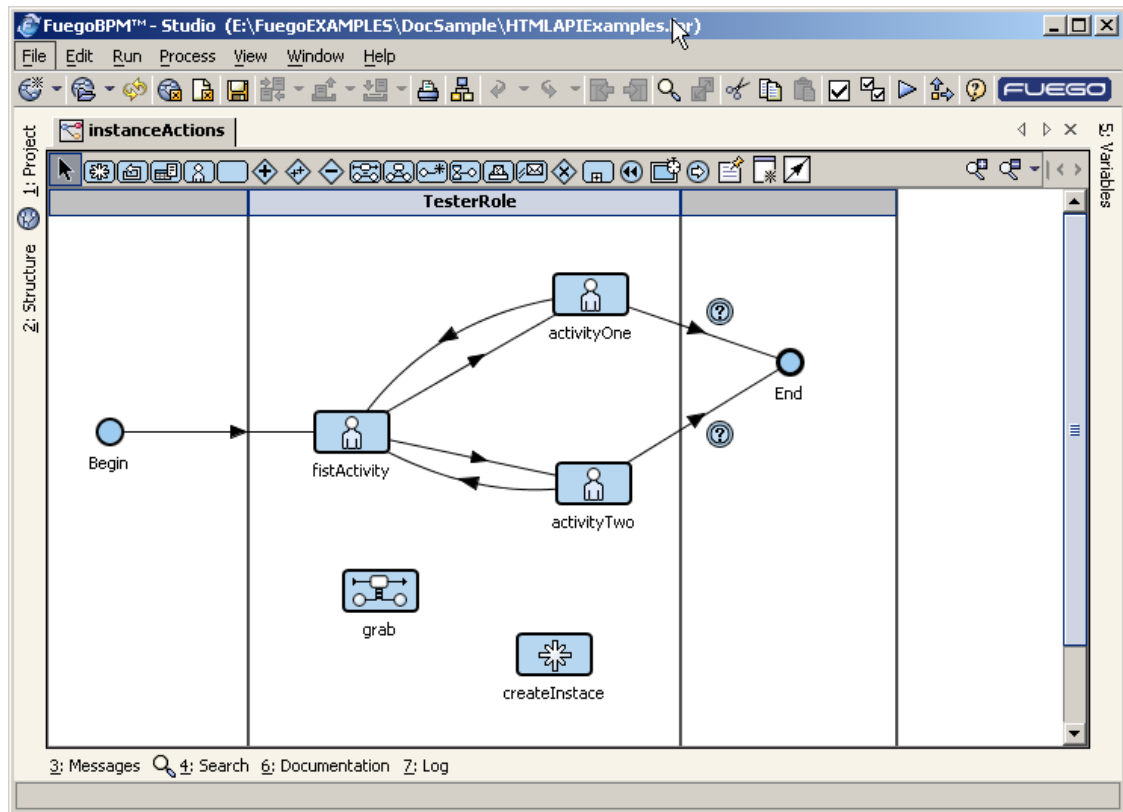
Process

The example process has been designed by modeling the three possible actions to execute from an HTML page using the framework.

The global activities execution has been modeled through a Global Creation activity.

The flow of the process allows for the execution of different actions for an instance, such as complete, process, select, grab, back, and so on.

The item execution example is contained in the activity *firstActivity*, as it has two tasks to perform.

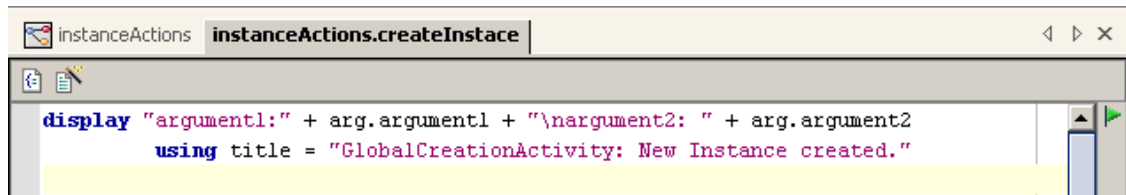


Business Process Methods

Global Activities Execution

As previously mentioned, the example is based on a Global Creation activity. The `createInstance` activity receives two arguments, `argument1` and `argument2`. These arguments have to be sent as parameters from the form post method of the HTML pages. Remember that those arguments from the html should always bear the "arg_" prefix (`arg_argument1.`) (*).

The Method is very simple, since it only displays the values received in the argument variables.



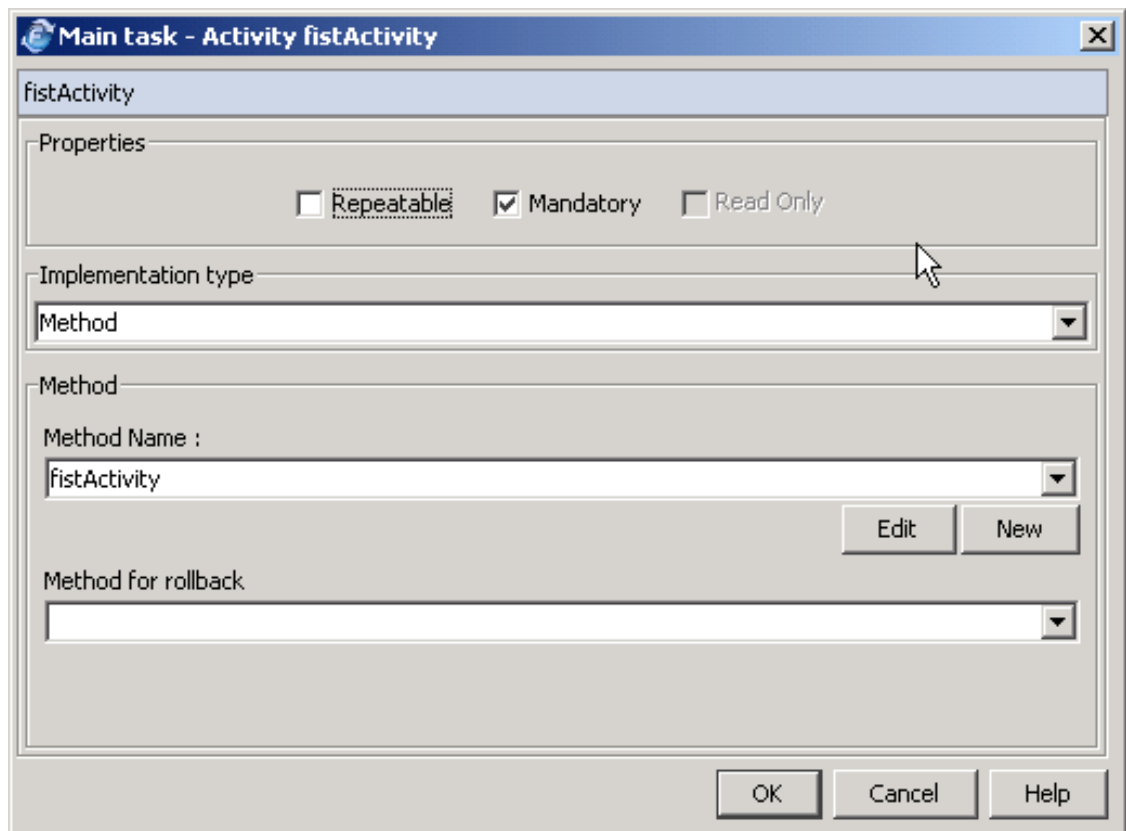
Reference (*) about arguments is related to reference (6) of the HTML pages section.

Instance Action Execution

The simulation is done with an instance located in the activity *firstActivity* of the process. Many actions can be executed on it.

Instance Task Execution

The process simulates this option through the task *firstTask*, in the *firstActivity*.



The task's BP-Method only contains a display sentence to verify that it is being executed.

```
display "First Task"
```

HTML pages

InstanceActions

This HTML page contains three sections. Each section corresponds to each execution option available for use.

```
<HTML>

<head>
</head>
<body style="font-family: Verdana, Arial, Helvetica">
<h1>Custom Logout Url Test</h1>

<!-- Global Creation Activity-->
<hr>
<h3>Test Global creation using anonymous access.</h3>
(1) <form method="post"
      ACTION=
"http://localhost:8585/portal/servlet/ExecutionDispatcher">
(2) <INPUT TYPE="hidden" name="activityId"
      value="/instanceActions#Default-1.1
      /createInstance" />
(3) <INPUT TYPE="hidden" name="actionId"
      value="RUN_GLOBAL" />
(4) <INPUT TYPE="hidden"
      name="fuego.portal.useNESession" value="true"/>
(5) <INPUT TYPE="hidden" name="fuego.portal.logoutURL"
      value="../instanceActionsLogoutOk.html"/>
      <INPUT TYPE="hidden"
      name="fuego.portal.logoutURL_Error"
      value="../instanceActionsLogout_Error.html" >
(6) Argument1:<input type="text" name="arg_argument1"><br>
      Argument2:<input type="text" name="arg_argument2"><br>
      <br>
      <INPUT TYPE="SUBMIT" NAME="Submit"
      VALUE="create an instance" />
</form>
```

```

<hr>

<!-- Instance Actions -->
<h3>Test Instance actions</h3>
(1) <form method="post"
    ACTION=
"http://localhost:8585/portal/servlet/instanceActions">
(5) <INPUT TYPE="hidden" name="fuego.portal.logoutURL"
    value="../../../instanceActionsLogoutOk.html" />
    <INPUT TYPE="hidden"
        name="fuego.portal.logoutURLError"
        value="../../../instanceActionsLogoutError.html" />
    <table>
    <tr>
        <td>instanceStampId:</td>
(7) <td><INPUT TYPE="text" name="instanceStampId"
        value="/instanceActions#Default-1.1
            /11/0/fistActivity"
            size="40"></td>
    </tr>
    <tr>
        <td>actionId:</td>
(8) <td><select name="actionId">
        <option>PROCESS</option>
        <option>COMPLETE</option>
        <option>SEND_TO</option>
        <option>BACK</option>
        <option>ABORT</option>
        <option>SUSPEND</option>
        <option>RESUME</option>
        <option>GRAB</option>
        <option>UNGRAB</option>
        <option>SELECT</option>
        <option>UNSELECT</option>
        <option>SELECT_ITEM</option>
        <option>UNSELECT_ITEM</option>
        </select>
        </td>
    </tr>
    <tr>
        <td>itemId:</td>
(9) <td><INPUT TYPE="text" name="itemId" value="0"/>
    (Note: Only used in SELECT_ITEM and UNSELECT_ITEM)
    </td>
    </tr>
    <tr>
        <td>grabActivityId</td>
(10) <td><INPUT TYPE="text" name="grabActivityId"
        value="grab"/>
        (Note: Only used in GRAB and UNGRAB)
    </td>

```

```

        </tr>

        </table>
        <br>
        <INPUT TYPE="SUBMIT" NAME="Submit" VALUE="do action"/>
        </form>
        <hr>

        <!-- Item Execution -->
        <h3>Run Item</h3>
(1)  <form method="post"
      ACTION=
"http://localhost:8585/portal/servlet/ExecutionDispatcher">
(5)  <INPUT TYPE="hidden" name="fuego.portal.logoutURL"
      value="../instanceActionsLogoutOk.html"/>
      <INPUT TYPE="hidden"
      name="fuego.portal.logoutURLError"
      value="../instanceActionsLogoutError.html"/>
      <table>
(11)<INPUT TYPE="hidden" name="actionId" value="RUN_ITEM"/>
      <tr>
        <td>instanceStampId:</td>
(7)  <td><INPUT TYPE="text" name="instanceStampId"
      value="/instanceActions#Default-1.1
      /1/0/fistActivity"
      size="40"></td>
      </tr>
      <tr>
        <td>taskDesc:</td>
(12) <td><INPUT TYPE="text" name="taskDesc"
      value="firstTask"/>
      </td>
      </tr>
      <tr>
(12) <td>itemId:</td>
        <td><INPUT TYPE="text" name="itemId" value="0"/>
      </td>
      </tr>

      </table>
      <br>
      <INPUT TYPE="SUBMIT" NAME="Submit" VALUE="do action"/>
      </form>
      <hr>

    </html>

```

instanceActionsLogoutOK

This is the page specified in the logout ok URL of the main page. In any of the three processing options in the main html page, instanceActions, the instanceActionsLogoutOk HTML would be displayed when the execution ends without error conditions.

```
<HTML>

<head>
</head>
<body style="font-family: Verdana, Arial, Helvetica">
<h1>Instance Action Test</h1>

Action successfully completed.

<form action="instanceActions.html">
<input type="submit" value="continue">
</form>

</html>
```

instanceActionsLogoutError

This is the page specified in the logout error URL of the main page. In any of the three processing options in the main html page, instanceActions, the instanceActionsLogoutError HTML would be displayed when the execution ends under error condition.

```
<HTML>

<head>
</head>
<body style="font-family: Verdana, Arial, Helvetica">
<h1>Instance Action Test</h1>

Action completed <b>unsuccessfully</b>.

<form action="instanceActions.html">
<input type="submit" value="continue">
</form>

</html>
```

References

INSTANCE CREATION SECTION:

- **(1)**

```
<form method="post"
ACTION="http://localhost:8080/portal/
        servlet/ExecutionDispatcher">
```

The use of the execution dispatcher of portal local host is being indicated.

- **(2)**

```
<INPUT TYPE="hidden" name="activityId"
value="/instanceActions#Default-1.1/createInstace"/>
```

The global activity to execute is the createInstace activity that belongs to the instanceAction process (see the Process section for further details). Change the variation and version, to the corresponding to your deployed process.

- **(3)**

```
<INPUT TYPE="hidden" name="actionId" value="RUN_GLOBAL"/>
```

The actionId should always have the value RUN_GLOBAL to execute a global activity.

- **(4)**

```
<INPUT TYPE="hidden" name="fuego.portal.useNESession"
      value="true"/>
```

The intention to use the anonymous participant, if it is defined, is being indicated. If it is not defined, a FuegoBPM participant will be required through the Work Portal login dialog.

- **(5)**

```
<INPUT TYPE="hidden" name="fuego.portal.logoutURL"
value="../instanceActionsLogoutOk.html"/>
<INPUT TYPE="hidden" name="fuego.portal.logoutURLError"
value="../instanceActionsLogoutError.html"/>
```

If the execution ends without or under error condition, these are the HTML pages to be displayed.

- **(6)**

```
Argument1: <input type="text" name="arg_argument1">
Argument2: <input type="text" name="arg_argument2">
```

This is the argument section. The global creation createInstance activity receives two arguments as parameters. In the HTML page,

the values for this parameters are requested to the end user. This reference is related to the (*) reference in the Method section.

Note



See how the argument name is composed when passing the parameter through the form block (arg_ + argumentName.)

INSTANCE ACTION SECTION:

- **(7)**

```
<INPUT TYPE="text" name="instanceStampId"
      value="/instanceActions#Default-1.1/11/0/fistActivity"
      size="40">
```

The instanceStampId is being set, where:

- process : instanceActions,
- variation : Default
- version : 1_1
- instance number : 11
- instance copy : 0
- activity : fistActivity.

If you try to execute this example, remember to change these values according to the specific moment you are testing. The variation and version may vary depending on how many times you have deployed the project and activity name may vary while instance flows through the process. The instance number may vary if you are testing the

same action process on new created instances.

- **(8)**

```
<select name="actionId">
    <option>PROCESS</option>
    <option>COMPLETE</option>
    <option>SEND_TO</option>
    <option>BACK</option>
    <option>ABORT</option>
    <option>SUSPEND</option>
    <option>RESUME</option>
    <option>GRAB</option>
    <option>UNGRAB</option>
    <option>SELECT</option>
    <option>UNSELECT</option>
    <option>SELECT_ITEM</option>
    <option>UNSELECT_ITEM</option>
</select>
```

The sample HTML page contains the list of possible actions to be selected by the end user. It will depend on the application, but generally this should be a hidden attribute.

- **(9)**

```
<INPUT TYPE="text" name="itemId" value="0"/>
```

In this example, the task to be executed is the first in the list of the activity.

- **(10)**

```
<INPUT TYPE="text" name="grabActivityId" value="grab"/>
```

In this example, the grab activity defined in the process to where the instance will be sent is named "grab".

RUN ITEM SECTION:

- **(11)**

```
<INPUT TYPE="hidden" name="actionId" value="RUN_ITEM"/>
```

When an activity task is being executed, the actionId attribute should always have the RUN_ITEM value.

- **(12)**

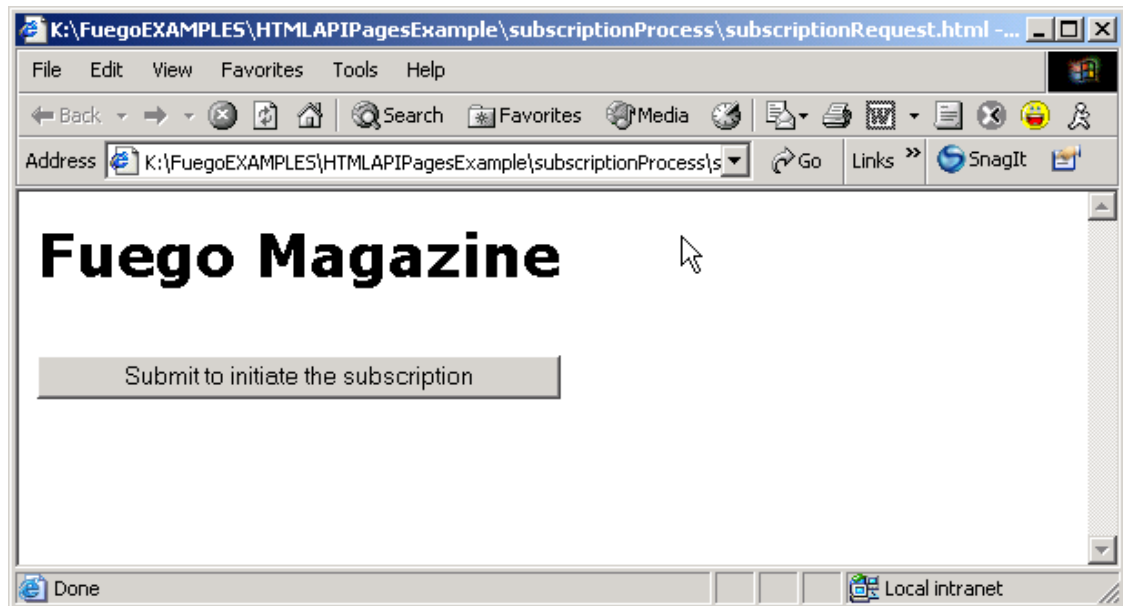
```
<INPUT TYPE="text" name="taskDesc" value="firstTask"/>
```

The task that will be executed in this example is the firstTask task of the fistActivity (see the Process section for further details).

Subscription process example

Process

The example process simulates the subscription to a Magazine. The company has decided to provide a web solution to allow users to subscribe to the magazine.



The input and approval of each subscription is executed and controlled by a FuegoBPM process. To input the subscription the user executes a global creation activity from a web page.

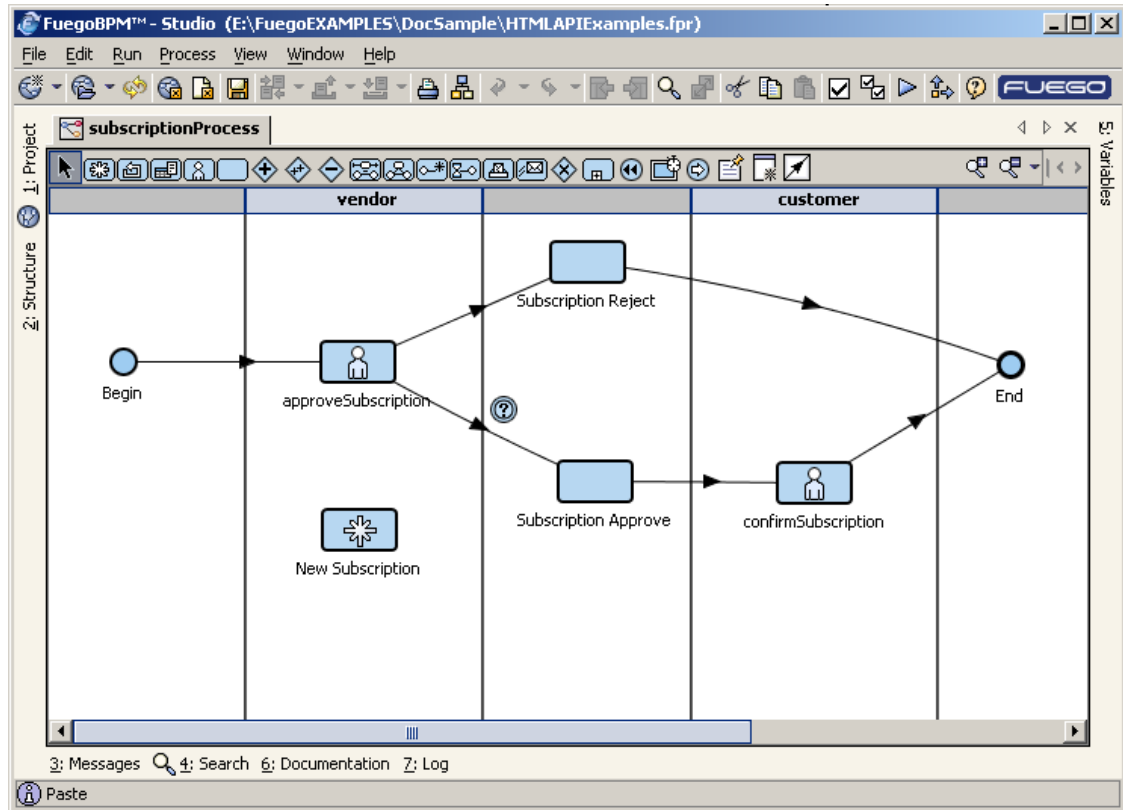
Using the framework interface the user creates an instance in the FuegoBPM process through the *NewSubscription* global creation activity, which invokes and executes the *inputSubscription* screenflow.

The approval is done through the *approveSubscription* activity. According to the decision made by the analyst to approve the subscriber or not, such person receives a mail explaining the decision. This activity executes the *approveSubscripSC*.

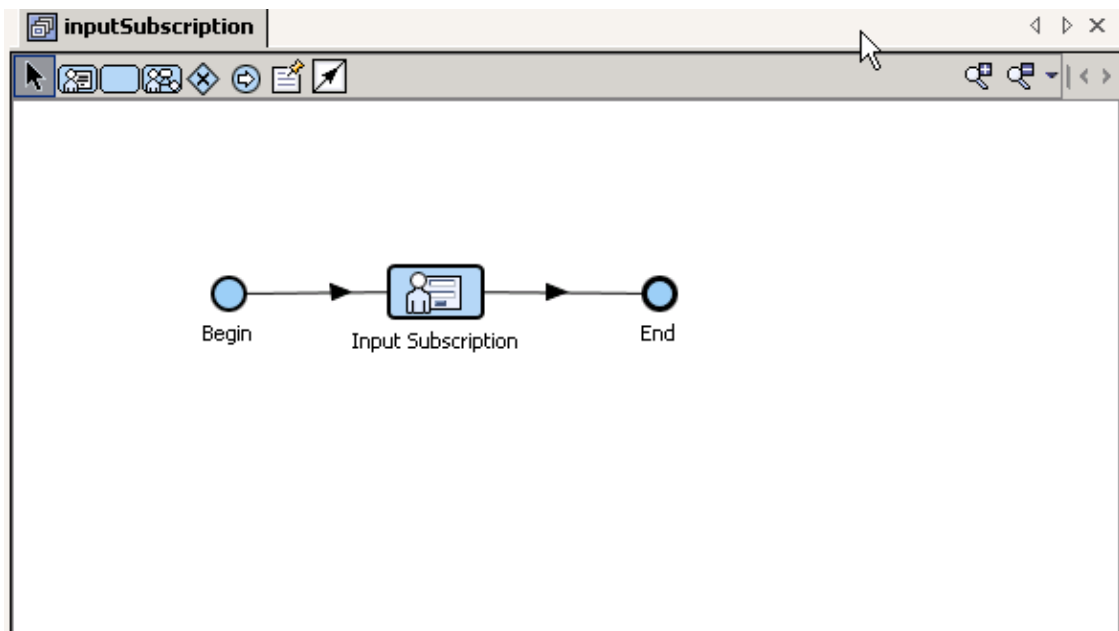
Two automatic activities, *Subscription Approve* and *Subscription Reject*, are the ones that prepare the final interaction with the user. If the subscription has been approved the mail will contain a final html code so that the user can accept or reject the terms of subscription and confirm it.

If it is rejected, the mail will contain a text explaining that the subscription has not been approved.

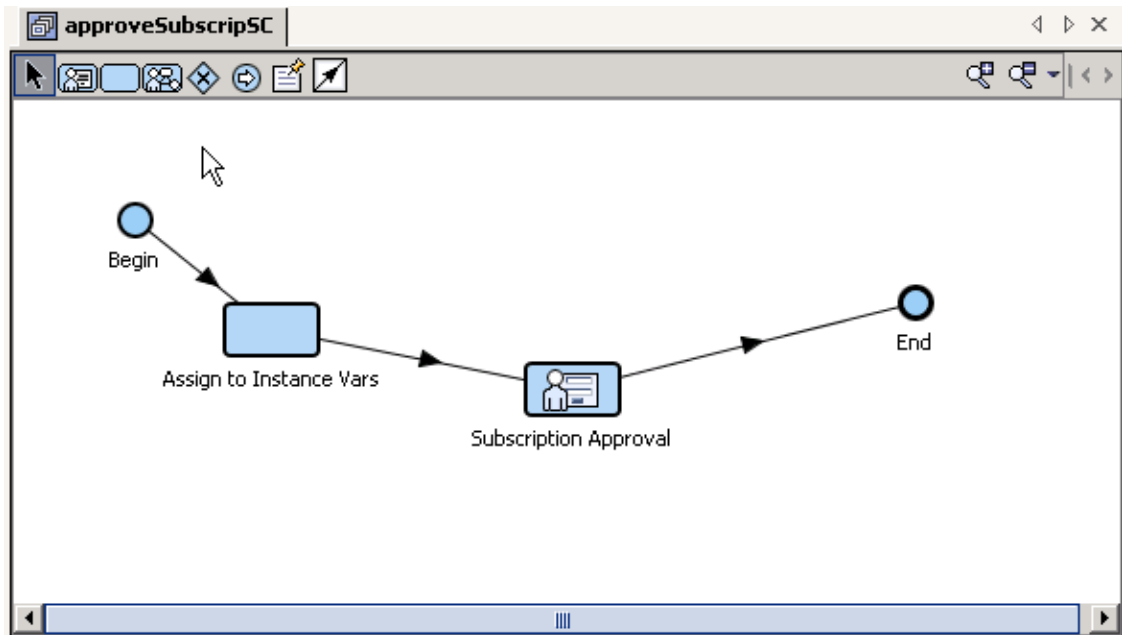
subscriptionProcess



inputSubscription screenflow



approveSubscripSC screenflow



Business Process Methods

NewSubscription Activity:

This activity has no Business Method related. It invokes the *inputSubscription* screenflow.

Through the *Input Subscription* interactive component call of the screenflow the required data is input by using the SubscriptionFO Fuego Object.

Main task - Activity Input Subscription

Input Subscription

Implementation type
Component

Component method

Properties

☒ Use instance variable subscriptionFO ...

Component name:
SubscriptionComponents.SubscriptionFO
Browse

Component member

☒ Use presentation Subscription

☐ Use instance variable ...

Argument mapping

OK Cancel Help

approveSubscription Activity:

The screenflow *approveSubscripSC* receives as an input argument the Fuego Object containing the subscription data, which is loaded into an instance variable (see Argument mapping).

The *Subscription Approval* activity of the screenflow, displays the information and requires an approval or rejection.

Subscription Approval

Implementation type: Input

Input dialog title: Subscription Approval. Instance Variables

	Label	Instance variable	Type
<input checked="" type="checkbox"/>	User ID	participantId (String)	Text
<input checked="" type="checkbox"/>	Name	participantName (String)	Text
<input checked="" type="checkbox"/>	Surname	participantSurname (String)	Text
<input checked="" type="checkbox"/>	Mail	participantMail (String)	Text
<input checked="" type="checkbox"/>	Telephone	participantTelephone (String)	Text
<input checked="" type="checkbox"/>	Fax	participantFax (String)	Text

Buttons: Approve Cancel Reject

Assign selected button to: button (String)

Cancel button is:

OK Cancel Help

According to the decision taken by the analyst, approve or reject, the selected button is send to the caller activity of the main process as the result to send the subscription to automatic activities that send a message to the subscriber informing if his/her request has been approved or not.

Subscription Approve Activity

Prepares the mail to be sent to the subscriber indicating that the subscription has been approved. Asks for the final acceptance or rejection of the terms and conditions and creates the FuegoBPM participant.

```
// Mail body.
mailBody = "<html><head><body style='font-family:
            Verdana, Arial, Helvetica'>"
mailBody = mailBody +
            "Hi," + participantName + "<br><br>"
```

```
mailBody = mailBody +
    "Your subscription was successfully approved.
    If you accept the terms, you must
    click the accept
    button and login one time.<br>"
mailBody = mailBody +
    "<form method='post' action='" +
        formAction + ">"
mailBody = mailBody +
    "<input type='hidden' name='instanceStampId'
        value='" +
        instanceStampId + ">";
mailBody = mailBody +
    "<input type='hidden' name='fuego.portal.logoutURL'
        value='" +
        logoutUrl + ">" ;
mailBody = mailBody +
    "<input type='hidden'
        name='fuego.portal.logoutURLerror' value='" +
        logoutUrlError + ">";
mailBody = mailBody +
    "<input type='hidden' name='actionId'
        value='COMPLETE'>";
mailBody = mailBody +
    "<input type='submit' name='accept'
        value='ACCEPT'>";
mailBody = mailBody + "</body></html>"

do
    session = DirectorySession.currentEngineSession

    ou = DirOrganizationalUnit.fetch(dir : session,
                                    id : "Fuego Inc./test")
    role = DirOrganizationalRole.fetch(session : session,
                                       id : "ExternalUser")

    roleAss[] = RoleAssignment.create(role : role,
                                      permissions : 0)

    dirHumanParticipant = DirHumanParticipant.create(
        session : session, id : participantId,
        firstName : participantName,
        lastName : participantSurname,
        displayName : participantName + " " +
            participantSurname,
        mail : participantMail,
        telephone : participantTelephone,
        fax : participantFax,
        password : participantPassword,
        ou : ou, rolesAssignment : roleAss,
        enabled : false)
```

```
    logMessage "SEVERE"
        using severity = SEVERE

    // Now send a mail to confirm the subscription.
    send Mail
        using from = "gloria@fuegolabs.com",
            subject = "Subscription Approved",
            message = mailBody,
            contentType = "text/html",
            @to = participantMail
    on e as Exception
        logMessage "Problems creating participant: " +
            e.message
            using severity = INFO
    end
```

```
    // Now send a mail to confirm the subscription.
    send Mail using to =participantMail,
        from = "magazine@fuego.com",
        subject = "Subscription Approved",
        message = mailBody,
        contentType = "text/html"

    on Exception
        logMessage "Problems creating participant: " +
            Exception.message using severity = INFO
    end
```

Subscription Reject Activity

Sends the mail to the subscriber indicating that his application has been rejected and sets the action to *ABORT* to terminate the instance in the process.

```
action = ABORT
send Mail using to =participantMail,
    from = "magazine@fuego.com",
    subject = "Subscription Rejected",
    message = "Hi," + participantName +
        "\n\n Your subscription has been rejected. \n Sorry",
```

```
contentType = "text/html"
```

confirmSubscription Activity

It does not have a related Business Process Method. This activity is the one in which the instance resides after the mail indicating the approval of the subscription has been sent to the user. When he/she executes the final action, the instance will flow to the *End* activity.

HTML pages

subscriptionRequest page

This is the page used by the final user to apply for a subscription. It creates an instance in the subscriptionProcess process.

```
<HTML>

<body style="font-family: Verdana, Arial, Helvetica">
<h1>Fuego Magazine</h1>

<FORM METHOD="post"
(1)  ACTION="http://localhost:8080/portal/
      servlet/ExecutionDispatcher">

(2)  <INPUT TYPE="hidden" name="activityId"
value="/subscriptionProcess/Default-1.0/createSubscription"/>

(3)  <INPUT TYPE="hidden" name="actionId"
value="RUN_GLOBAL"/>

(4)  <INPUT TYPE="hidden" name="fuego.portal.useNESession"
value="true"/>

(5)  <INPUT TYPE="hidden" name="fuego.portal.logoutURL"
      value="../subscriptionRequestLogout.html"/>
      <INPUT TYPE="hidden"
      name="fuego.portal.logoutURL_Error"
      value="../subscriptionRequestError.html"/>

<br>
(6) <INPUT TYPE="SUBMIT" NAME="Submit"
      VALUE="Submit to initiate the subscription"/>
```

```
</FORM>

</HTML>
```

subscriptionRequestError page

This page is invoked from the subscriptionRequest page when the subscription was executed ending under an error condition (see reference (5) of the subscriptionRequest page code).

```
<HTML>

<head>
</head>
<body style="font-family: Verdana, Arial, Helvetica">
<h1>Fuego Magazine</h1>
Your Subscription process was completed
    <b>unsuccessfully</b>.
</body>

</html>
```

subscriptionRequestLogout page

This page is invoked from the subscriptionRequest page when the subscription was executed ending without any error condition (see reference (5) of the subscriptionRequest page code).

```
<HTML>

<head>
</head>
<body style="font-family: Verdana, Arial, Helvetica">
<h1>Fuego Magazine</h1>
Your request is in process.
You will receive a mail with a notification.
</body>
```

```
</html>
```

subscriptionRequestFinish page

This page is invoked from the HTML block in the mail sent to the user when the subscription was approved. It allows the user to know that his confirmation to the subscription has successfully ended (see reference (b) in the Method section of this example).

```
<HTML>

<head>
</head>
<body style="font-family: Verdana, Arial, Helvetica">
<h1>Fuego Magazine</h1>
Your Subscription process was successfully completed.
You will receive the first issue in a month.
</body>

</html>
```

References

- (1)

```
<FORM METHOD="post "
ACTION="http://localhost:8080/portal/
servlet/ExecutionDispatcher">
```

The use of the execution dispatcher of portal local host is being indicated.

- (2)

```
<INPUT TYPE="hidden" name="activityId"
value="/subscriptionProcess#Default-1.0/createSubscription"/>
```

The global activity to execute is the *createSubscription* activity that belongs to the subscriptionProcess process. (see the Process section for further details). Through this execution, an instance is created and the approval process begins. Change the variation and version, to the corresponding to your deployed process.

- **(3)**

```
<INPUT TYPE="hidden" name="actionId" value="RUN_GLOBAL"/>
```

The actionId should always have the value RUN_GLOBAL to execute a global activity.

- **(4)**

```
<INPUT TYPE="hidden" name="fuego.portal.useNESession"
value="true"/>
```

The intention to use the anonymous participant, if it is defined, is being indicated. If it is not defined, a FuegoBPM participant will be required through the Work Portal login dialog.

- **(5)**

```
<INPUT TYPE="hidden" name="fuego.portal.logoutURL"
```



```
value="../../../subscriptionRequestLogout.html"/>
<INPUT TYPE="hidden" name="fuego.portal.logoutURLerror"
value="../../../subscriptionRequestError.html"/>
```

If the execution ends without or under error condition, these are the HTML pages to be displayed respectively.

- **(6)**

Arguments input section. Submit button.

Create a participant example

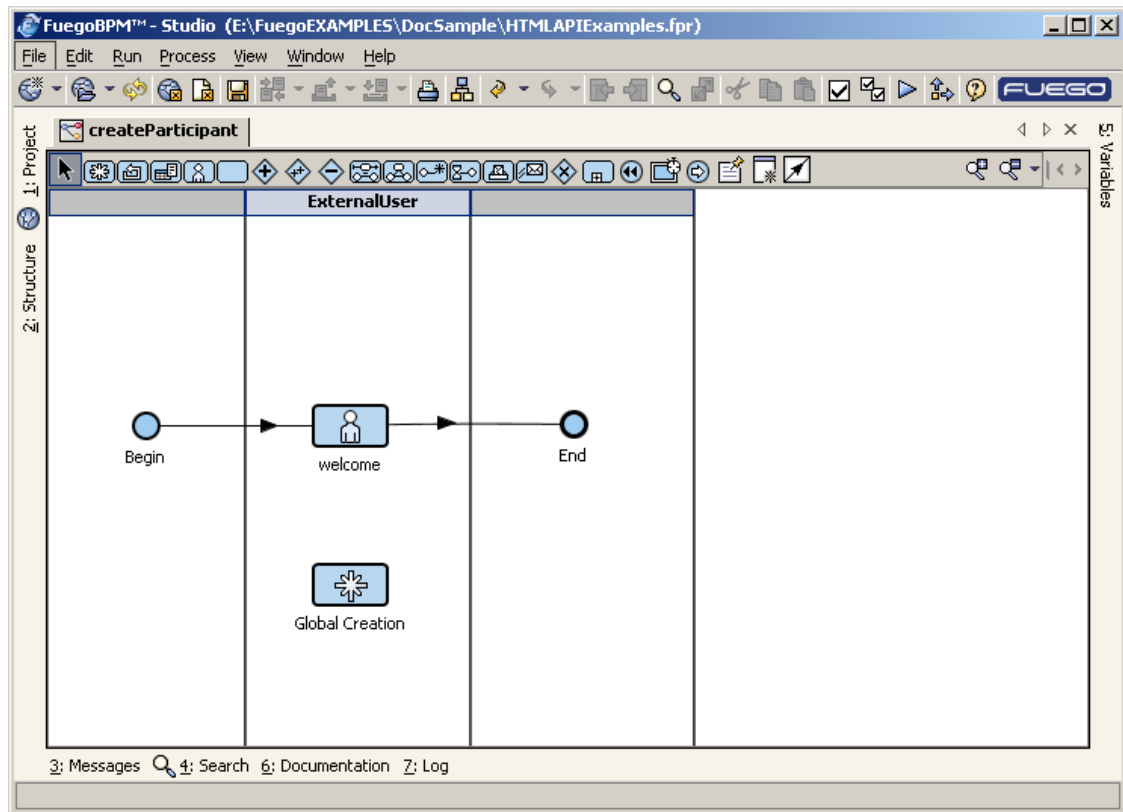
Process

This process shows how to create a FuegoBPM participant. The Global activity create is executed by the end user from an html page. This example shows that user interaction with a FuegoBPM process from an HTML is possible.

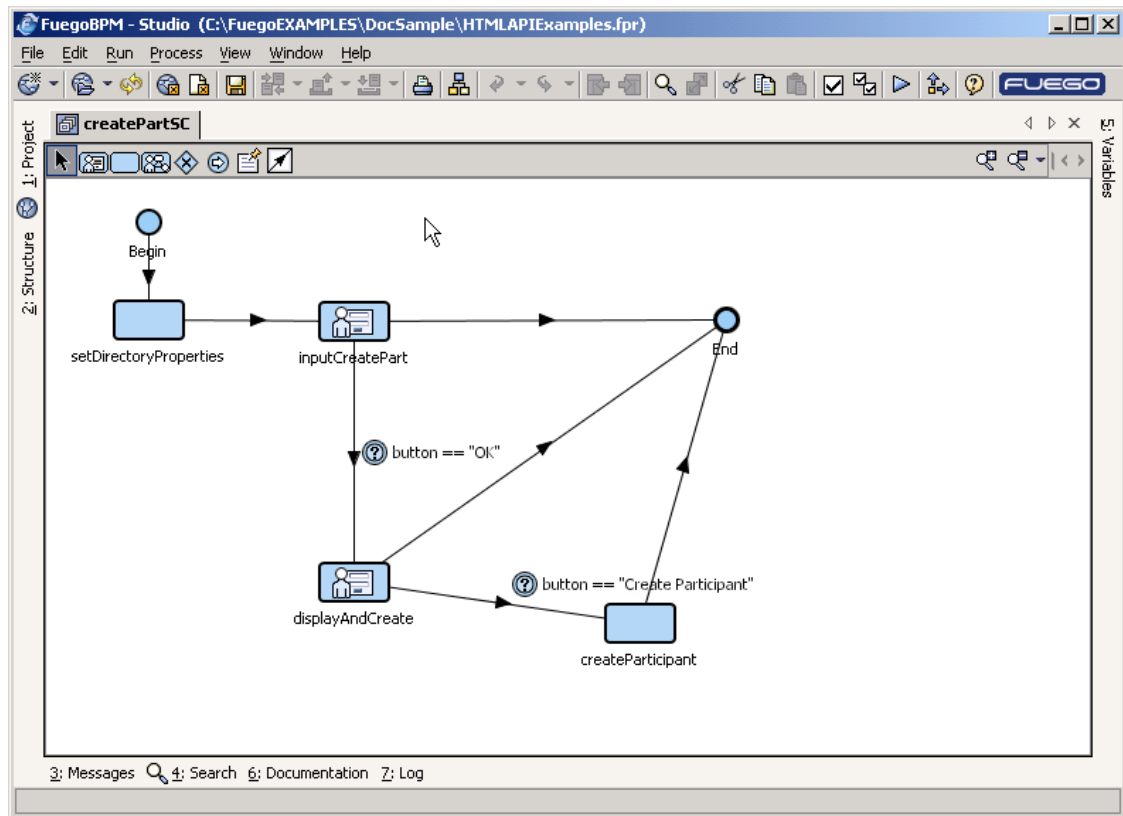
You will note that, from the HTML, the non-exclusive session will be used and as logout URL is set the one corresponding to Work Portal. This is a way of providing the end user with the possibility of creating his own participant and using it immediately after logging in to Work Portal.

The process is very simple. It implements in a Global activity a screenflow that handles the input and confirmation of the participant data. This global activity, *Global Creation*, is invoked from the HTML page.

Process



Screenflow



Business Process Methods and activities definitions

The activity *Global creation* of the main process invokes the screenflow that handles the data input. Its definition is as follows:

The screenshot shows a dialog box titled "Main task - Activity Create Participant". It has a close button in the top right corner. The dialog is divided into several sections. The first section, "Create Participant", is highlighted in blue. Below it is the "Implementation type" section, which contains a dropdown menu currently showing "Screenflow" and a "Test" button. The next section is "Related Process", which includes a "Screenflow name" dropdown menu showing "createPartSC", and "Edit" and "New" buttons. Below this is an "Argument mapping" section, which is currently empty. At the bottom of the dialog are three buttons: "OK", "Cancel", and "Help".

The process requires the user to input the information of the participant that he intends to create in FuegoBPM Studio. To do so, the interactive component call activity *inputCreatePart* is defined as follows:

inputCreatePart

Implementation type
Input

Input dialog title: Participant data input. Fill in the form and press OK

	Label	Instance variable	Type
<input checked="" type="checkbox"/>	User ID	participantId (String)	Text
<input checked="" type="checkbox"/>	Password	participantPassword (String)	Password
<input checked="" type="checkbox"/>	Name	participantName (String)	Text
<input checked="" type="checkbox"/>	Surname	participantSurname (String)	Text
<input type="checkbox"/>	Mail	participantMail (String)	Text
<input type="checkbox"/>	Telephone	participantTelephone (String)	Text
<input type="checkbox"/>	Fax	participantFax (String)	Text

Buttons

Assign selected button to: button (String)

Cancel button is: Cancel

OK Cancel Help

The information just input is displayed so that the user can confirm or modify it if there are any mistakes. To do so, the interactive component activity *displayAndCreate* is defined as follows:

displayAndCreate

Implementation type
Input

Input dialog title: Participant data confirmation. Instance Variables

<input checked="" type="checkbox"/>	<input type="checkbox"/>	Label	Instance variable	Type
<input checked="" type="checkbox"/>	<input type="checkbox"/>	User ID	participantId (String)	Text
<input checked="" type="checkbox"/>	<input type="checkbox"/>	Name	participantName (String)	Text
<input checked="" type="checkbox"/>	<input type="checkbox"/>	Surname	participantSurname (String)	Text
<input checked="" type="checkbox"/>	<input type="checkbox"/>	Mail	participantMail (String)	Text
<input checked="" type="checkbox"/>	<input type="checkbox"/>	Telephone	participantTelephone (String)	Text
<input checked="" type="checkbox"/>	<input type="checkbox"/>	Fax	participantFax (String)	Text

Buttons

+ - Create Participant Cancel

Assign selected button to: button (String)

Cancel button is: Cancel

OK Cancel Help

Finally an automatic activity creates the participant using the data input by the user in the previous activities.

```

session = DirectorySession.currentEngineSession

ou = DirOrganizationalUnit.fetch(dir : session,
    id : "Fuego Inc./test")
role = DirOrganizationalRole.fetch(session : session,
    id : "ExternalUser")

roleAss[] = RoleAssignment.create(role : role,
    permissions : 0)

dirHumanParticipant = DirHumanParticipant.create(
    session : session,
    id : participantId,
    firstName : participantName,
    lastName : participantSurname,
    displayName : participantName + " " +
        participantSurname,

```

```
mail : participantMail,  
telephone : participantTelephone,  
fax : participantFax,  
password : participantPassword,  
ou : ou,  
rolesAssignment : roleAss,  
enabled : false)
```

HTML pages

```
<HTML>  
  
<BODY>  
  
<h1>Create a Participant</h1>  
  
<FORM METHOD="get" ACTION=  
  "http://localhost:8585/portal/servlet/ExecutionDispatcher">  
  
  <INPUT TYPE="hidden" name="processId"  
    value="/createParticipant#Default-1.0"/>  
  
  <INPUT TYPE="hidden" name="activityId"  
    value="/CreateParticipant"/>  
  
  <INPUT TYPE="hidden" name="actionId"  
    value="RUN_GLOBAL"/>  
  
  <INPUT TYPE="hidden" name="fuego.portal.useNESession"  
    value="true"/>  
  
  <INPUT TYPE="hidden" name="fuego.portal.logoutURL"  
    value="http://localhost:8585/portal/">  
  
  <INPUT TYPE="SUBMIT" NAME="Submit"  
    VALUE="Press Button to hit 'Global Creation' Activity"/>  
  
</form>  
  
</BODY>  
  
</HTML>
```

Chapter 13. BPEL Processes

FuegoBPM & BPEL Processes

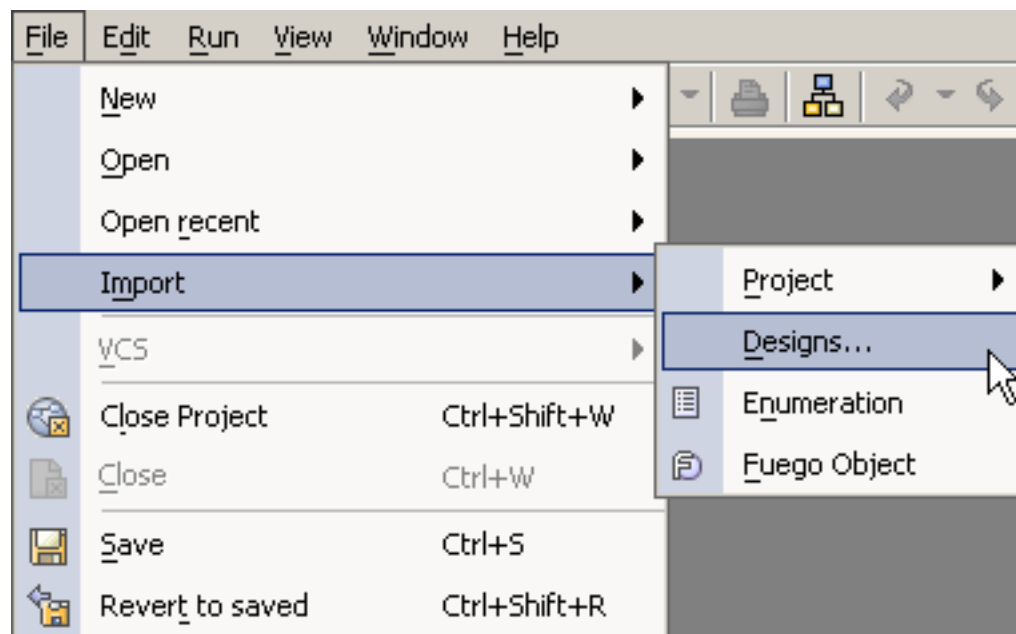
FuegoBPM supports BPEL Processes. You can import a BPEL processes into FuegoBPM Studio as well as to design a BPEL process directly in the product.

Importing a BPEL process

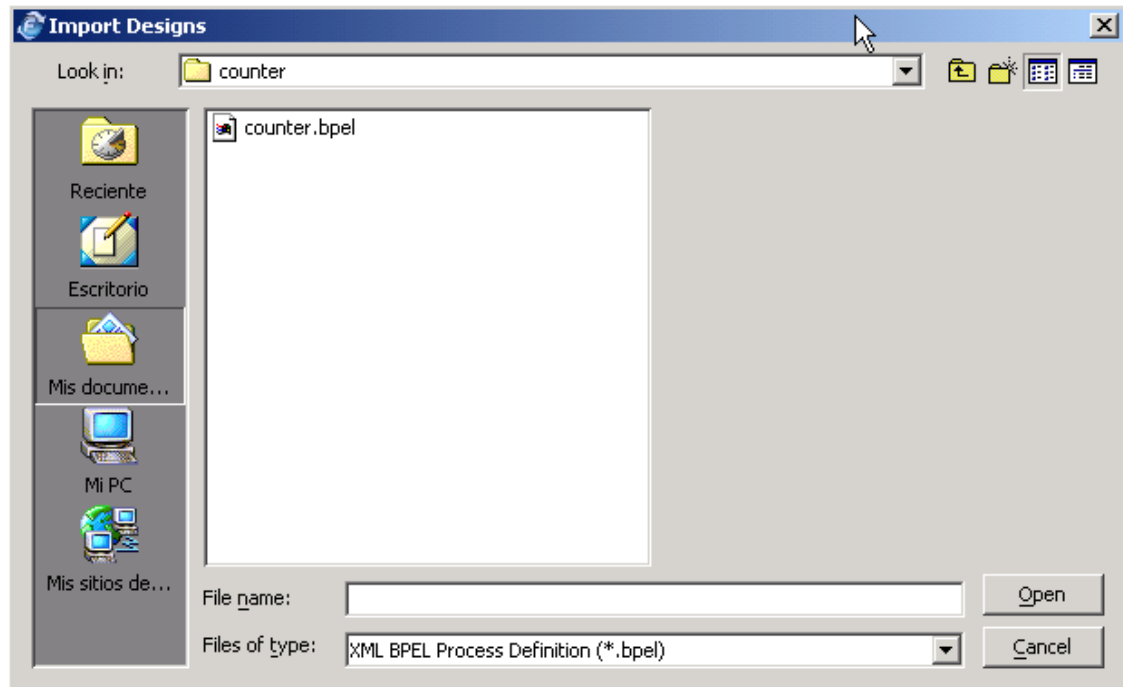
All processes have to be added to a project. Therefore create a project.

If the BPEL process to import has components, catalog them before you import the process.

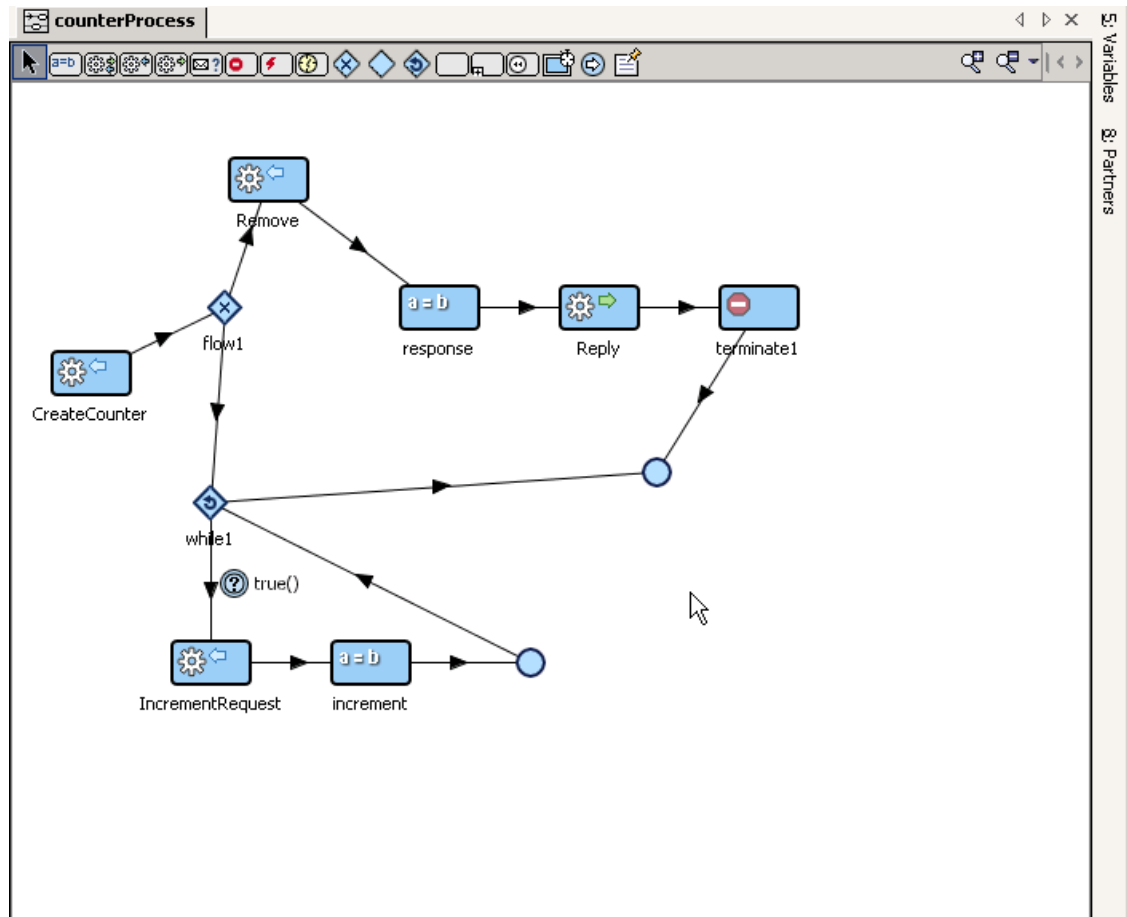
Select from the **File** menu, the **Import** option. Select **Designs**.



Browse to find the BPEL process to import.



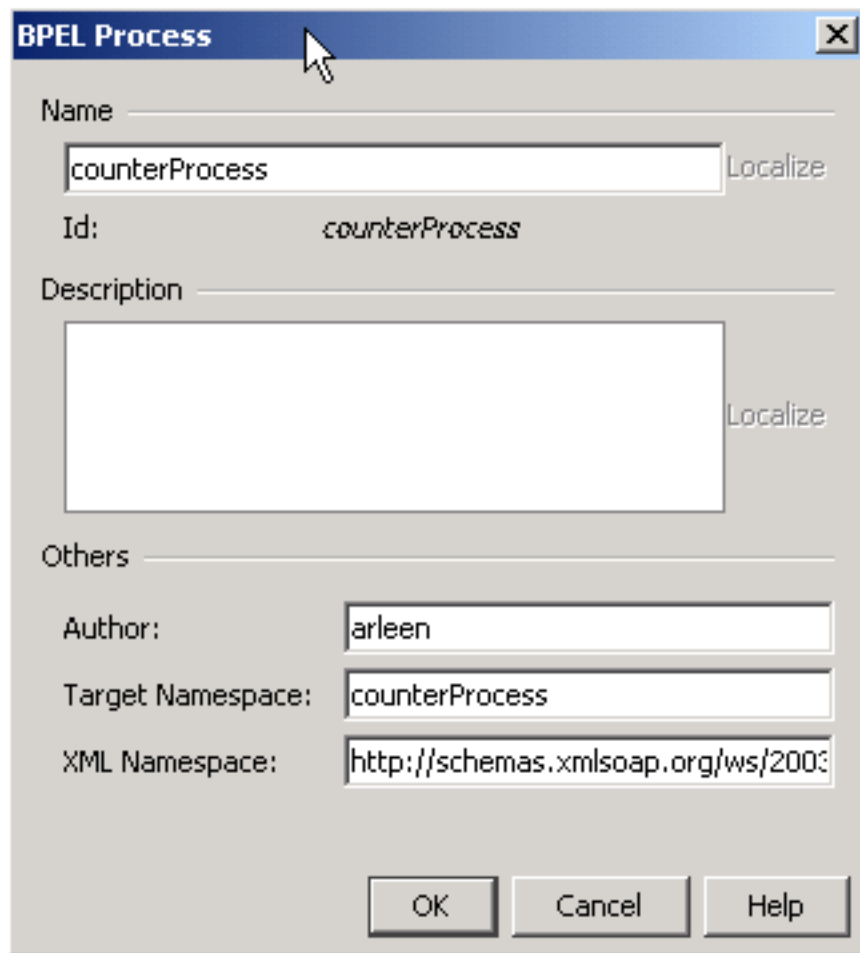
Once you finish the import procedure, the process is automatically created. A new process is created with the same name as the file that was imported.



Creating a BPEL process

First of all, create a project.

Right click on the project tree node and select the **New BPEL process** option. You can also create a new process from the **File -> New -> BPEL Process** menu or from **New -> New BPEL Process** toolbar button.

A screenshot of a 'BPEL Process' dialog box. The title bar is blue with the text 'BPEL Process' and a close button. The dialog has several sections: 'Name' with a text box containing 'counterProcess' and a 'Localize' button; 'Id:' with the text 'counterProcess'; 'Description' with a large empty text area and a 'Localize' button; and 'Others' with three text boxes: 'Author:' containing 'arleen', 'Target Namespace:' containing 'counterProcess', and 'XML Namespace:' containing 'http://schemas.xmlsoap.org/ws/2003/'. At the bottom are 'OK', 'Cancel', and 'Help' buttons.

BPEL Process

Name Localize

Id: *counterProcess*

Description Localize

Others

Author:

Target Namespace:

XML Namespace:

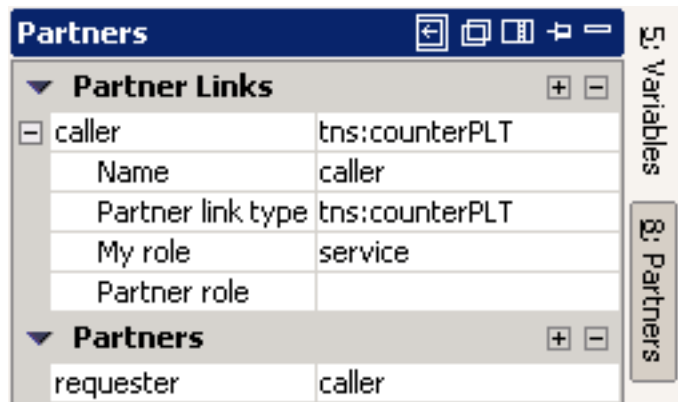
OK Cancel Help

The workspace is set with a BPEL toolbar from where you can select the BPEL activities.

Start designing the BPEL process.

If you forget to catalog the components used within the process, you will get an error at publishing time.

Partners Tab



Define the Partner Link types defined in the used webservices and complete the roles.

As well, define the partners and the partner links that they support.

FuegoBPM BPEL processes

FuegoBPM BPEL processes are in BPEL native format.

As well, FuegoBPM Server is a native BPEL server.

For a brief description of the BPEL activities refer to BPEL Activities.

BPEL documentation

Please refer to the BPEL4WS Specification at these locations:

[<http://dev2dev.bea.com/technologies/webservices/BPEL4WS.jsp>]

[<http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/>]

[<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbiz2k2/html>]

[<http://ifr.sap.com/bpel4ws/>]

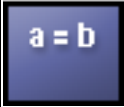


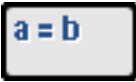
[<http://www.siebel.com/bpel>]

BPEL activities

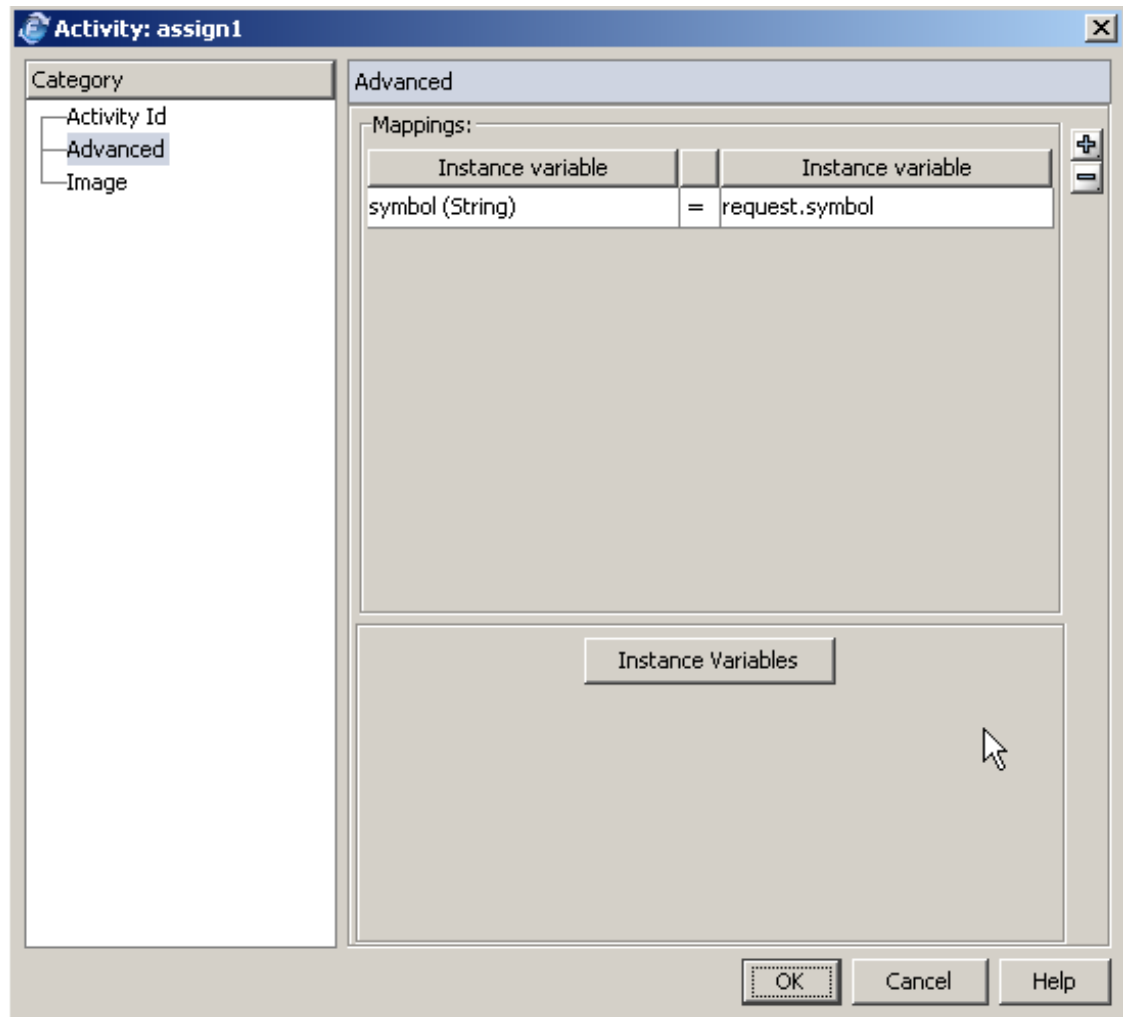
FuegoBPM has a complete BPEL activities toolbar from which you can choose to build the process.

Each activity has properties to be defined, mainly in the **Advanced** tab, according to the characteristics of each activity. For further information please refer to Defining an Activity.

ASSIGN





Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			

Advanced Tab



Map the instance variables. For example the received information into the FuegoBPM's instance variables.

INVOKE

Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			

Advanced Tab

Activity: invoke

Category

- Activity Id
- Advanced
- Image

Advanced

Port Type

Component name:

Webservices.DelayedQuotes.StockQuoteService

Browse

Operation

Method Name

getQuote(in symbol : String, out result : Real(32))

Partner Link

Partner link description

provider

Correlations

Input Variable

Instance variable

a

Output Variable

Instance variable

b

OK Cancel Help

Define the operation to invoke. Define the *Component*, *Method* and *Partner*. As well as the *Input* and *Output* variable.

RECEIVE

Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon

Advanced Tab





The screenshot shows the 'Activity: receive' dialog box with the 'Advanced' tab selected. The 'Category' list on the left includes 'Activity Id', 'Advanced', and 'Image'. The main form contains the following fields and controls:

- Port Type**: A section containing a 'Component name' text box with the value 'Webservices.Simple.StockQuotePTService' and a 'Browse' button.
- Operation**: A section containing a 'Method Name' dropdown menu with the selected value 'gimmeQuote(in symbol : String, out quote : Real(32))'.
- Partner Link**: A section containing a 'Partner link description' text box with the value 'caller'.
- Correlations**: A button located below the Partner Link section.
- Input Variable**: A section containing an 'Instance variable' dropdown menu with the value 'request' and a button with three dots.
- Create Instance**: A section containing a 'Create Instance description' text box and a checked checkbox labeled 'Create Instance'.

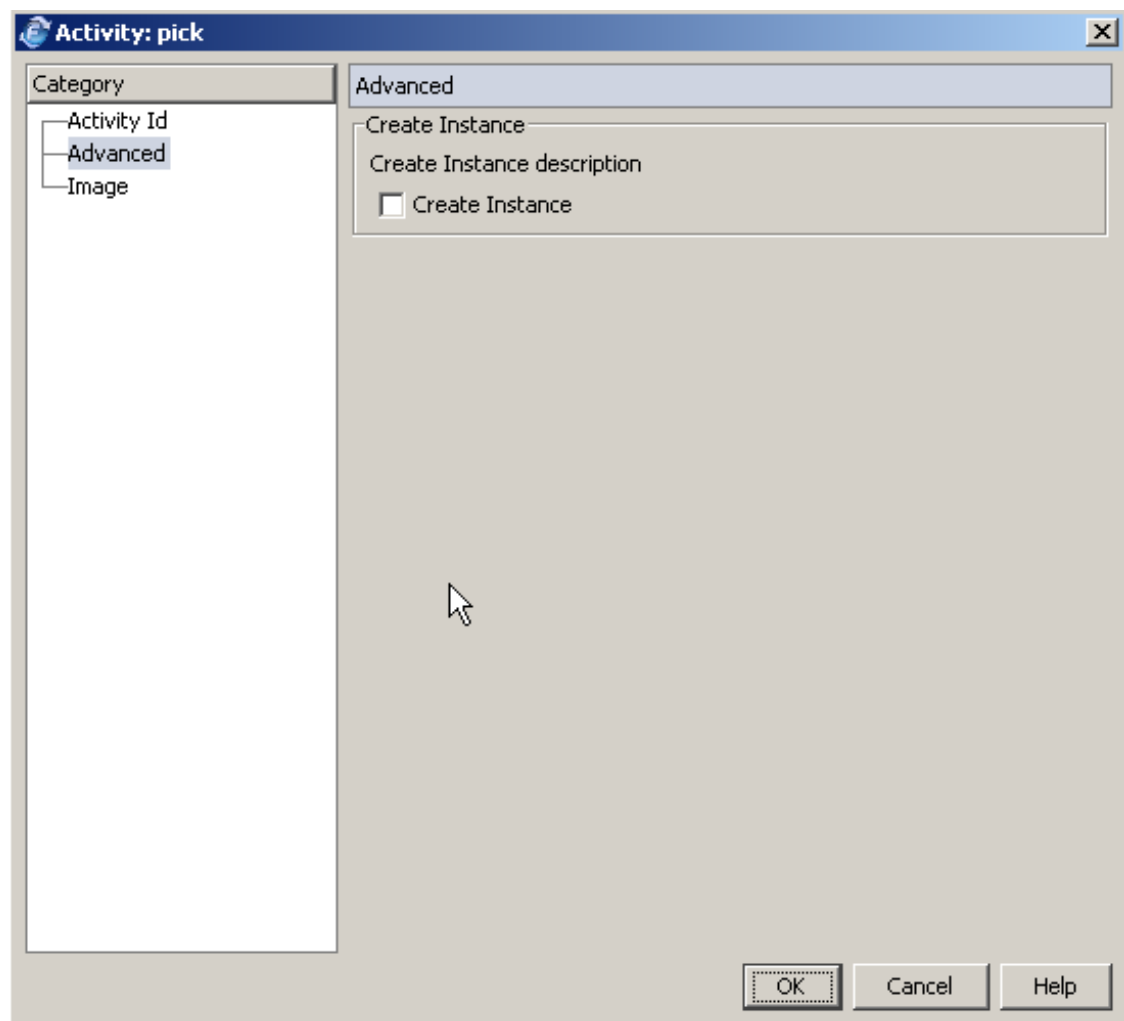
At the bottom right of the dialog are three buttons: 'OK', 'Cancel', and 'Help'.

Define the operation used to receive. Define the *Component*, *Method* and *Partner*. As well as the *Input* variable. Select the *Create Instance* check box if a new instance has to be created to flow through the process.

PICK





Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			

Advanced Tab







Select the *Create Instance* check box if a new instance has to be created to flow through the process.

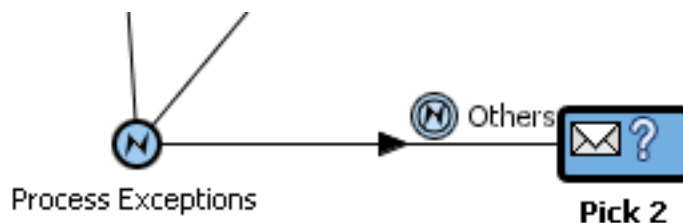
TERMINATE

Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			

THROW EXCEPTION

Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			

FAULT HANDLER: Exception transition



To add a **Fault Handler**, right click on the designer workspace and select **Add exception transition to**. Previously define one of the activities that are part of the exception flow.

Define the exception and catalogue it or use the *Others* exception to catch any exception that might occur

Transition from Activity: 'ProcessGroup' to Activity: 'Pick'

Description Properties

Type
Exception

Exception
Select the exception from the catalogue. If this exception occurs while executing the source activity, the instance will be routed through this transition.

Exception name
Others

Browse

Exception holder variable
Choose the instance variable that will hold the exception once it occurs

excepvar

Browse...

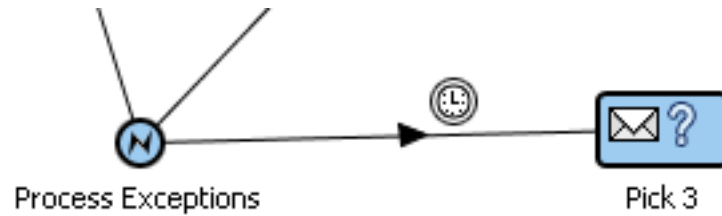
OK Cancel Help

EVENT HANDLERS

To add an **Event Handler**, right click on the designer workspace and select **Add due transition to** , or **Add Message based transition**. Previously define one of the activities that are part of the event handler flow.

Due transition (On Alarm Event)

This is one of the Event Handlers options



Define the due event to handle.

Transition from Activity: 'ProcessGroup' to Activity: 'pick1'

Description Properties

Type

Due

Interval Expression

☒ Use calendar rules

☒ Next working day, same time ☐ Next scheduled based time

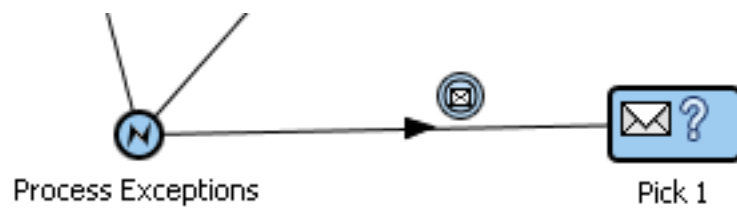
'10s'

Fuego

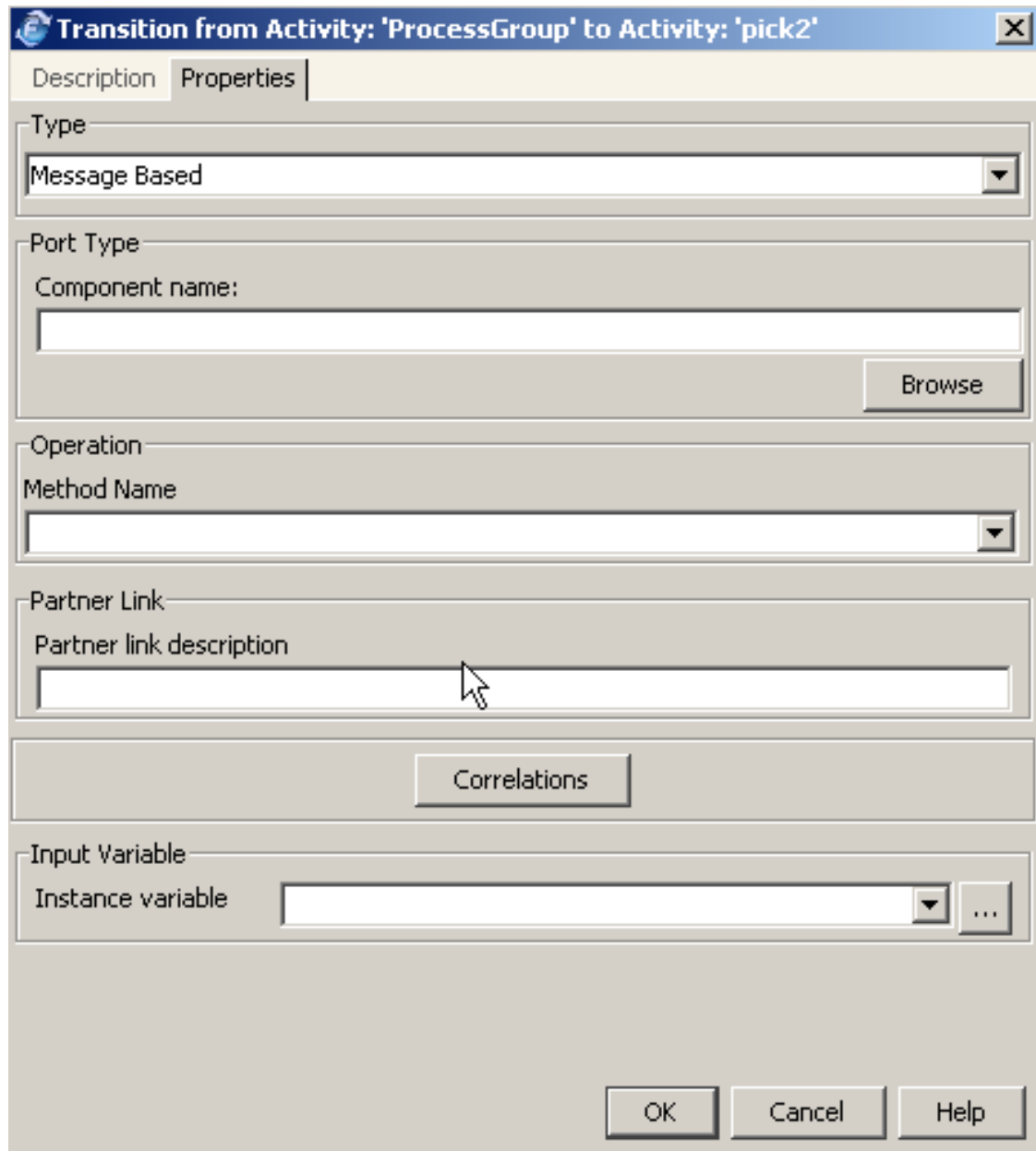
OK Cancel Help

Message based transition (On Message Event)

This is the other one of the Event Handlers options



Define the listener that will wait for the event message. Define the *Component*, *Method* and *Partner*. As well as the *Input* variable.







The image shows a dialog box titled "Transition from Activity: 'ProcessGroup' to Activity: 'pick2'". It has two tabs: "Description" and "Properties", with "Properties" currently selected. The dialog contains several sections for configuring the transition:

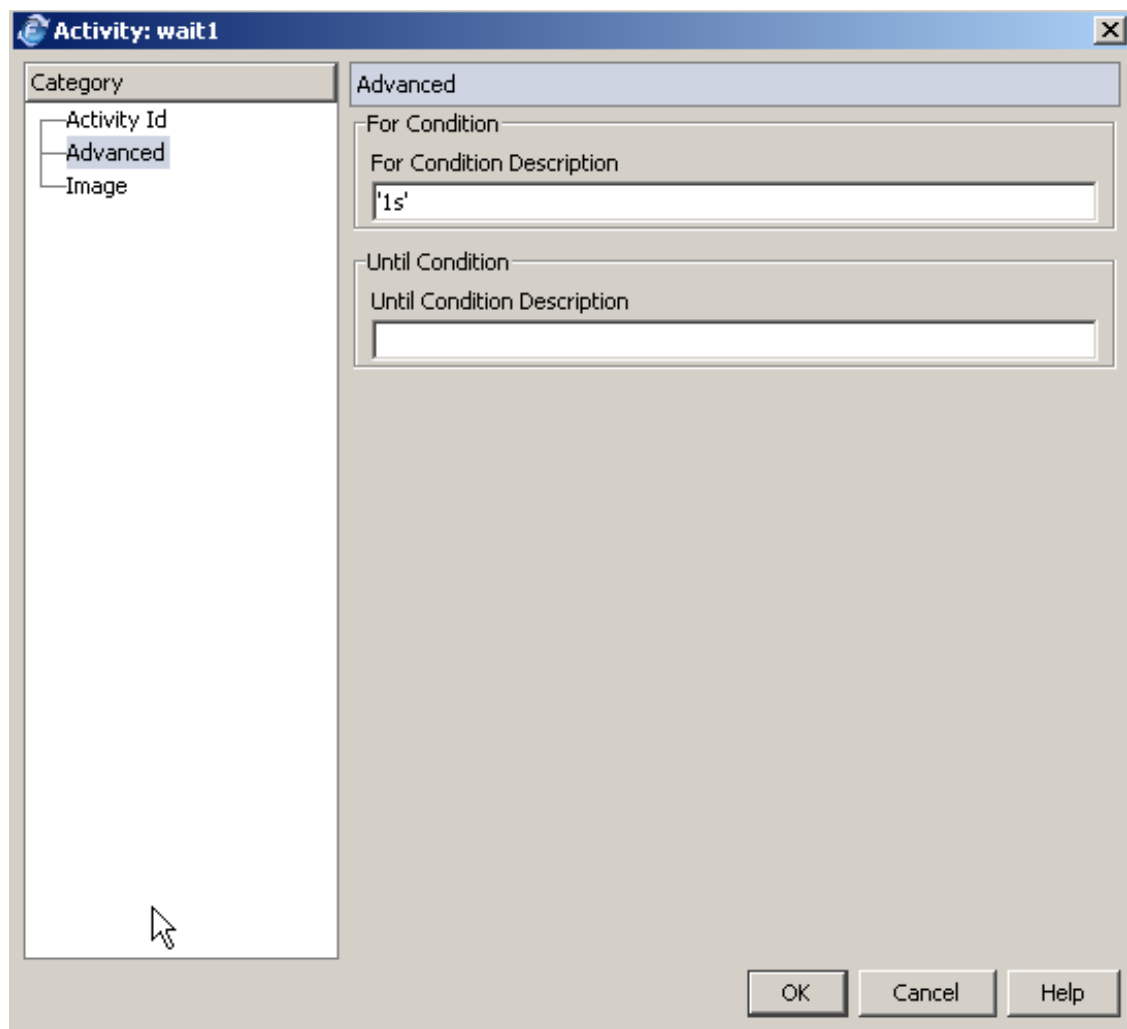
- Type:** A dropdown menu showing "Message Based".
- Port Type:** A section with a "Component name:" label and an empty text box, followed by a "Browse" button.
- Operation:** A section with a "Method Name" label and a dropdown menu.
- Partner Link:** A section with a "Partner link description" label and an empty text box. A mouse cursor is pointing at this text box.
- Correlations:** A button labeled "Correlations" located below the Partner Link section.
- Input Variable:** A section with an "Instance variable" label and a dropdown menu, followed by an ellipsis button "...".

At the bottom right of the dialog are three buttons: "OK", "Cancel", and "Help".

WAIT

Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			

Advanced Tab



Activity: wait1

Category

- Activity Id
- Advanced
- Image

Advanced

For Condition

For Condition Description

'1s'





Until Condition

Until Condition Description





OK Cancel Help

Define the wait condition. You have to define the *for condition* or the *until condition*.

FLOW





Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			

SWITCH





Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			

To represent each *case* of the switch you have to define a conditional transition. The unconditional transition represents the *otherwise*.

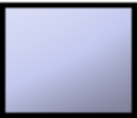



WHILE

Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			

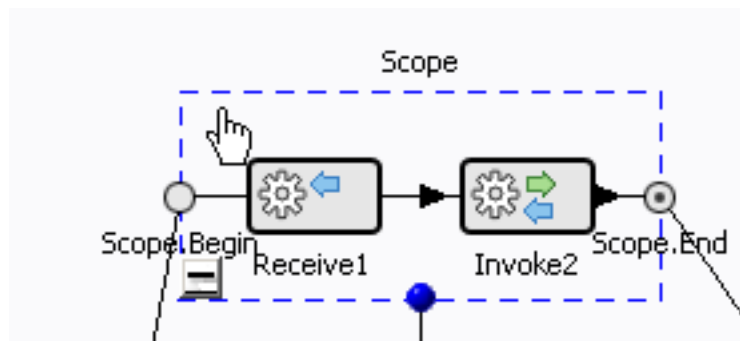
EMPTY

Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			





SCOPE

Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			

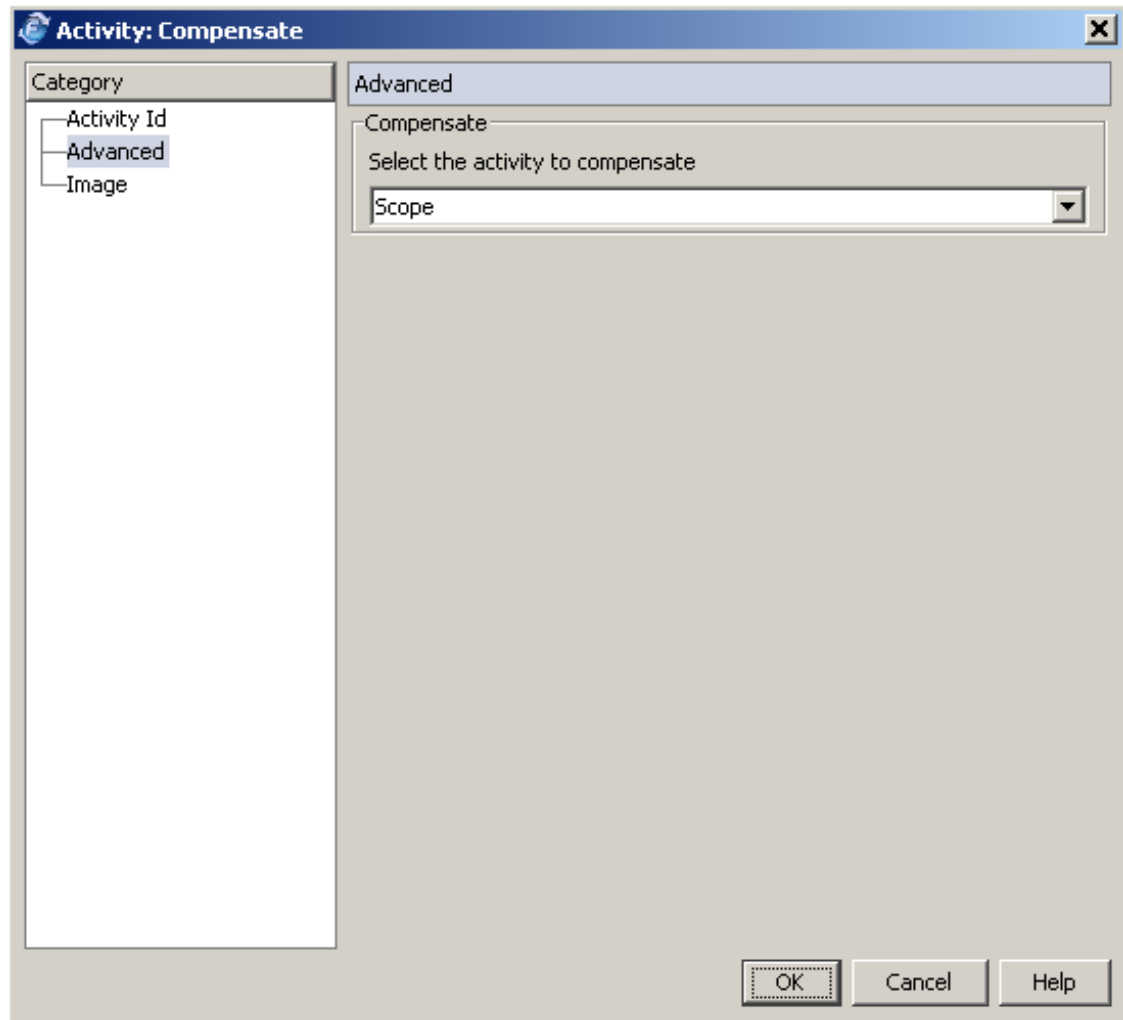
Select a group of activities and define them within a **Scope**



COMPENSATE





Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			

Advanced Tab



Define the activity to compensate.

REPLY

Classic Icon	UML Icon	Business Analyst Icon	BPMN Icon
			

Advanced Tab

The screenshot shows a dialog box titled "Activity: reply" with a close button (X) in the top right corner. On the left is a "Category" tree with three items: "Activity Id", "Advanced" (which is selected and highlighted), and "Image". The main area of the dialog is divided into several sections:

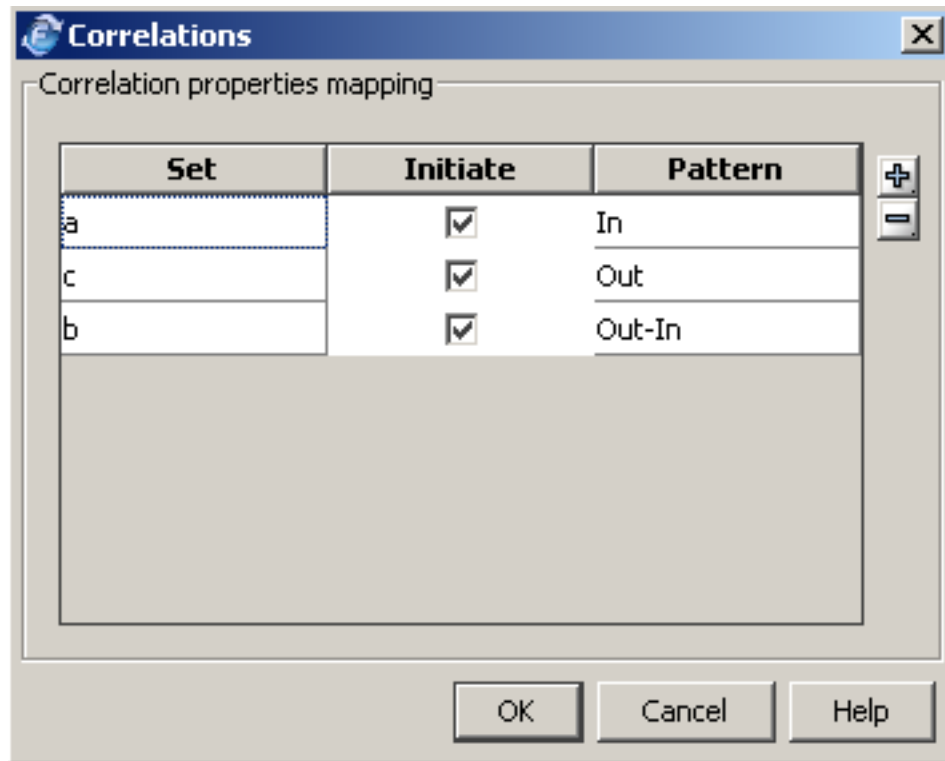
- Advanced**: A section header.
- Port Type**: A section containing a "Component name:" label and a text box with the value "Webservices.Simple.StockQuotePTService". A "Browse" button is located to the right of the text box.
- Operation**: A section containing a "Method Name" label and a dropdown menu showing "gimmeQuote(in symbol : String, out quote : Real(32))".
- Partner Link**: A section containing a "Partner link description" label and a text box with the value "caller".
- Correlations**: A button labeled "Correlations" located below the Partner Link section.
- Output Variable**: A section containing an "Instance variable" label and a dropdown menu showing "response". To the right of the dropdown is a button with three dots "...".

At the bottom right of the dialog are three buttons: "OK", "Cancel", and "Help". A mouse cursor is pointing at the "Cancel" button.

Define the operation used to receive. Define the *Component*, *Method* and *Partner*. As well as the *Output* variable.

Correlations

Correlations can be edited for **Invoke**, **Receive**, **Reply** and **Message Based Transitions**



BPEL4WS documentation

For a more comprehensive description of BPEL activities please refer to the BPEL4WS Specification at these locations:

[<http://dev2dev.bea.com/technologies/webservices/BPEL4WS.jsp>]

[<http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/>]

[<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbiz2k2/html>]

[<http://ifr.sap.com/bpel4ws/>]

[<http://www.siebel.com/bpel>]

Chapter 14. Simulation

Simulation

Once a project has been designed, you can **simulate** its execution. The project can simulate one or more of its processes at the same time.

All the defined Resources will be shared in between processes.

Simulation **does not** effectively execute the activity's tasks. It only **simulates** the process execution by waiting the length of time each task would have taken to execute.

To execute a simulation you have to define **models**. You have:

- Project simulation models, and
- Processes simulation models


A **Project simulation model** is composed by a group of processes simulation models. For each process you can define a **Process simulation model**.

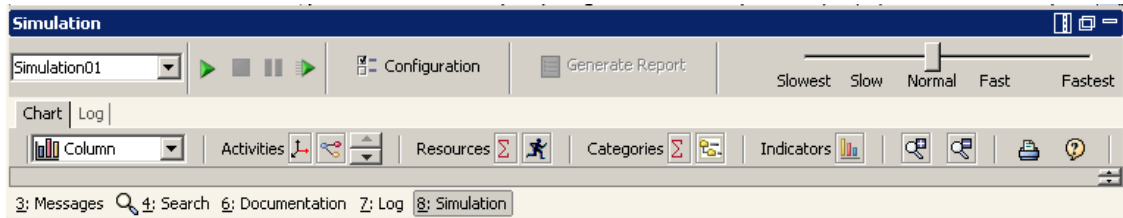
The **project** can have defined multiple simulation models. And each **process** can have multiple simulation models, as well. So you can build different project simulation models combining different processes simulation models.

The project simulation model is defined in the project directory, in *simulation* directory as *name of the model* + *.xpsi*. And the processes simulation models have a name composed of *name of the process* + *name of the model* + *.xsim*.

Each model has its own configuration and simulation definition.


Defining a Project Simulation Model


To define the **Simulation Model**, click on the **Run** icon  on FuegoBPM Studio/Designer's main toolbar. The Simulation panel is displayed at the bottom of your workspace and Flap number 8 is enabled.



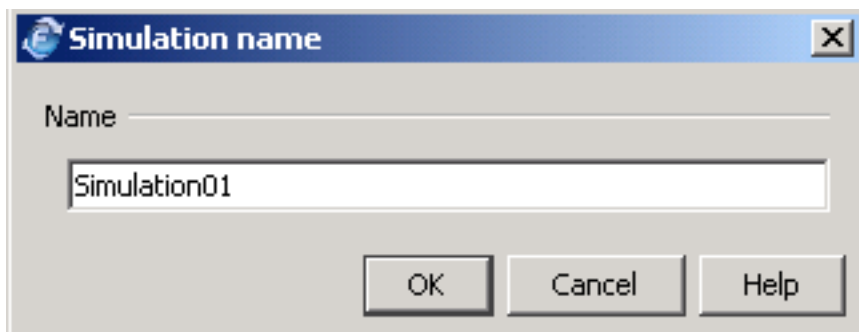
All defined **project simulation models** appear in the combo box.

To create a new model, select **Configuration**.

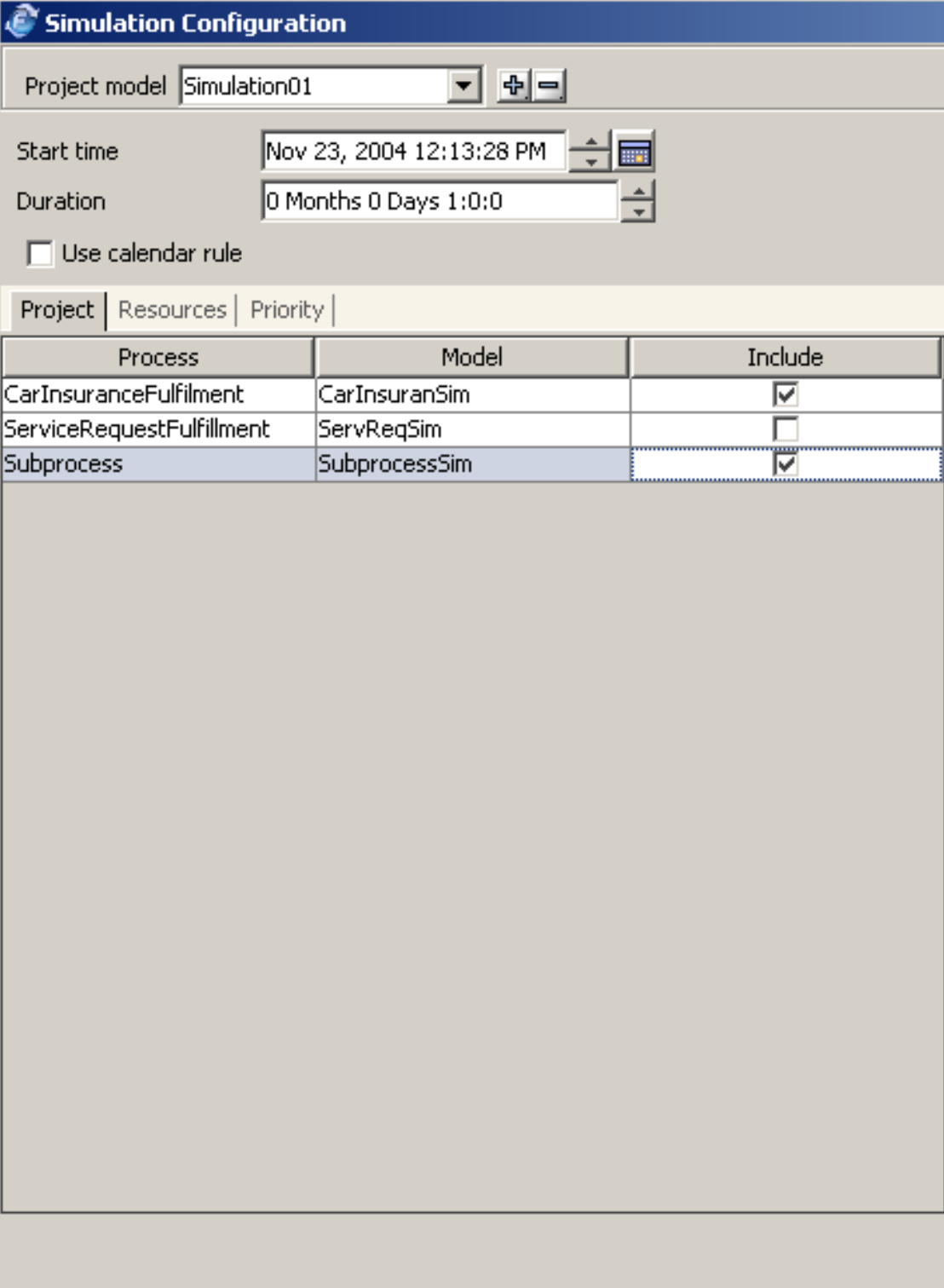
If you already have defined simulation models, you need to add a new one by clicking the  button.

If you have no previous models you can also create the first one by clicking on the **Start** button. 

A dialog box appears requesting the project simulation model's definition.



Next, you have to define the model's properties.



The image shows a 'Simulation Configuration' window. At the top, there's a title bar with a gear icon and the text 'Simulation Configuration'. Below this, there's a 'Project model' dropdown menu set to 'Simulation01', with '+' and '-' buttons. Underneath, 'Start time' is set to 'Nov 23, 2004 12:13:28 PM' with a calendar icon, and 'Duration' is set to '0 Months 0 Days 1:0:0' with a time picker icon. A checkbox labeled 'Use calendar rule' is unchecked. Below these settings are three tabs: 'Project', 'Resources', and 'Priority'. The 'Project' tab is active, showing a table with three columns: 'Process', 'Model', and 'Include'. The table has three rows: 'CarInsuranceFulfilment' with model 'CarInsuranSim' and 'Include' checked; 'ServiceRequestFulfillment' with model 'ServReqSim' and 'Include' unchecked; and 'Subprocess' with model 'SubprocessSim' and 'Include' checked. The bottom half of the window is a large, empty gray area.

Simulation Configuration

Project model: Simulation01

Start time: Nov 23, 2004 12:13:28 PM

Duration: 0 Months 0 Days 1:0:0

☐ Use calendar rule

Project | Resources | Priority

Process	Model	Include
CarInsuranceFulfilment	CarInsuranSim	<input checked="" type="checkbox"/>
ServiceRequestFulfillment	ServReqSim	<input type="checkbox"/>
Subprocess	SubprocessSim	<input checked="" type="checkbox"/>

Project Information panel

Start time: indicates the supposed date that the simulation runs for logging purposes. It is **not** for scheduling purposes.

Duration: indicates the period during which the simulation will run (Months, days, hours, minutes, seconds.) For example, during 1 day and 6 hours.

Use calendar rule: not applicable in this release. It will be available in future releases.

Project Tab

Project Resources Priority				
Process	Model	Include	Process	
CarInsuranceFulfilment	CarInsuranSim	<input checked="" type="checkbox"/>	Process model <input type="text"/> <input type="button" value="v"/> <input type="button" value="+"/> <input type="button" value="x"/>	
ServiceRequestFulfilment		<input type="checkbox"/>		
Subprocess	SubprocessSim	<input checked="" type="checkbox"/>		

The list of all the project's processes are displayed. To make the process part of the project simulation model, you have to first define the **process simulation model** on the right panel and then select the **Run simulation** check box.

If any of these processes act only as a subprocess, this means that instances are generated from a parent process only, then you mustn't check the box. if you do so, within the process, instances will come from the parent process (if it is part of the project simulation model) and as well instances are generated directly in the process.


For all processes that are checked, instances are generated when the simulation is run.

Project Resources Tab

Project Resources Priority						
	Name	Cost	Efficiency	Capacity	Availability	Roles
<input checked="" type="checkbox"/>	Customer Reps	\$1.0	100.0%	3	100%	[Customer Service R...
<input checked="" type="checkbox"/>	Mary Grant	\$0.0	80.0%	1	100%	[Supervisor]
<input checked="" type="checkbox"/>	New Supervisor	\$1.0	100.0%	1	50%	[Supervisor]
<input checked="" type="checkbox"/>	John Daveport	\$0.0	80.0%	1	100%	[Customer Service R...

You define the resources with which you will run the simulation. All the processes share these resources.

Cost is per hour of each resource.

You can **copy** the resources from the organization by clicking on the  button. The simulation default values are taken from the Organization definition. Afterwards you can:

- Add new resources by clicking on the "+" sign.
- Delete a resource by clicking on the "-" sign.
- Modify any of the resources properties as **Cost (per hour)**, **Efficiency or Capacity** (the displayed values are those loaded in the Organization Manager)
- Define the **Availability** that the resource will have in this process for this model.
- Modify, add, or delete roles for the resource.

Priority Tab

Name	Value
Highest	20
High	30
Normal	30
Low	10
Lowest	10

The instances have a priority that determines the way in which they flow within the process. If your simulation model needs to deal with instances with different priorities, you can determine the probability for each priority.

- **Priority:** Highest, High, Normal, Low, Lowest
- **Probability:** the sum of all 5 priorities must be 100


Defining a Process Simulation Model

A process simulation model is defined on the right panel. Select the **Project Tab** from the left panel. Select the process for which you want to create a model (be sure it is grayed and not necessarily its

check box is selected).

All defined **process simulation models** appear in the combo box.

If no model exists, you are able to enter a new simulation model name.

If you already have defined process simulation models, you need to add a new one by clicking the  button.

Process Information panel

Instance generation info:

Within this tab, you can determine the way in which the instances are generated.

- **Enable amount of instances limit:** enable this check box if there is a limit for instance generation. If the check box is enabled, you define the maximum number of instances to generate. The Simulation will run until the duration is completed or the maximum number of instances is reached, whatever happens first.
- **Distribution used for process instance creation:** each instance is created depending on the defined distribution type.
- **Constant:** instances are regularly generated as defined in the **period** property.
- **Uniform:** is similar to the generation of Constant instances but with a **delta** variation. For example, if instances are generated every 30 seconds with a 10% delta, this means that they are generated every 27 to 33 seconds.
- **Normal:** responds to the Gauss Bell distribution.
- **Exponential:** indicates how many instances in an **Average frequency** are generated **every** period.

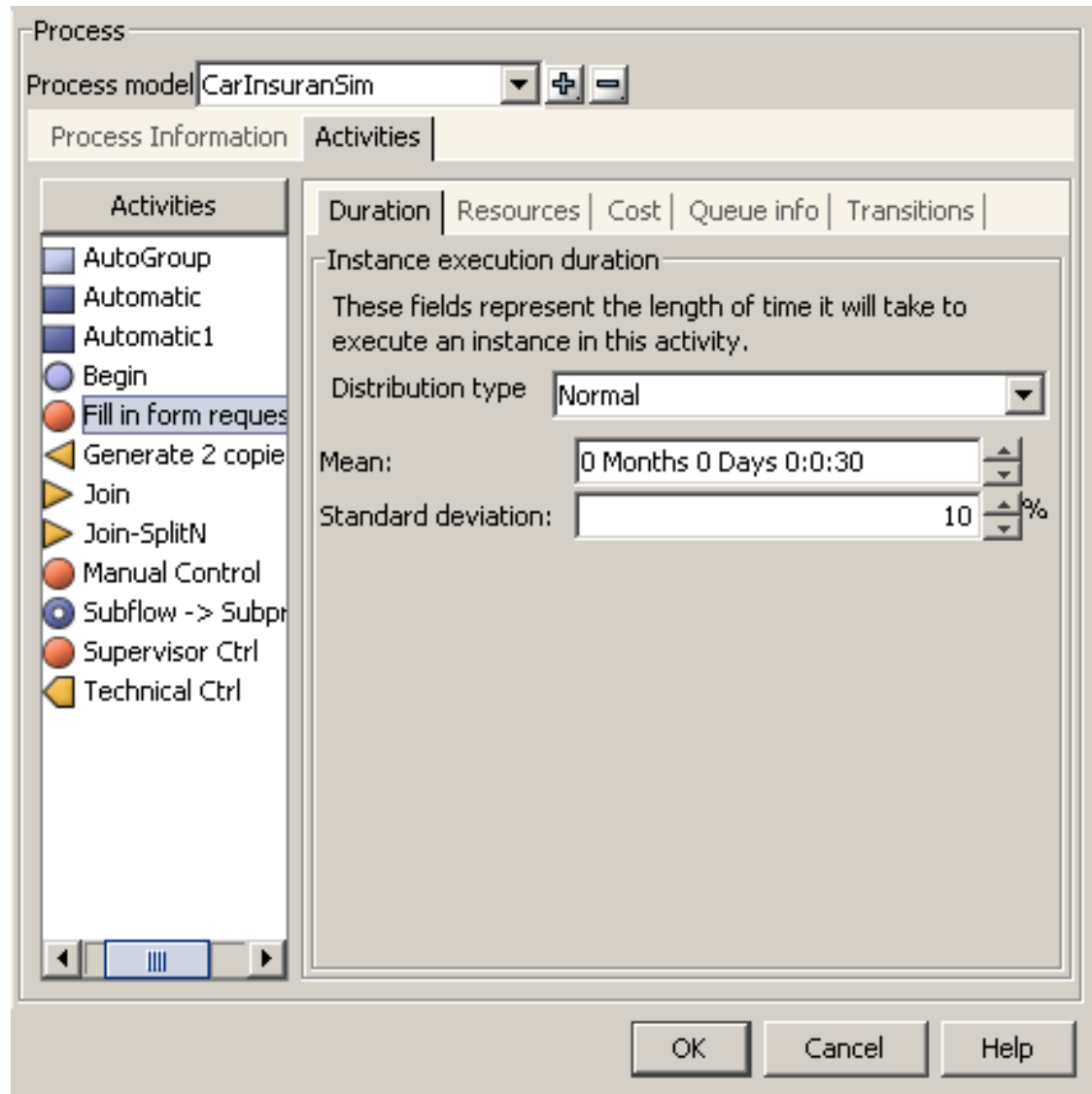
- Exponential distribution considerations: the exponential distribution describes the distribution of time between events in a Poisson process. A Poisson process is characterized by the expected number of events in a time interval of given length. This distribution is commonly used to model random "arrivals", such as:
 - The number of calls to a call center per day
 - Number of times a web server is accessed per minute
 - Number of cars arriving at a toll booth per hour
 - Number of people arriving at a bank teller queue every ten minutes
- The mean and standard deviation of an exponential random variable are given by (time interval)/(expected number of events in that interval).
- Example of use in FuegoBPM Studio/Designer: Let us consider a customer support process in which a phone call by a customer triggers the creation of a process instance. We know, from experience, that our support technicians get about 50 calls per day. We would like to evaluate how different resource allocation scenarios would affect response times. Calls to a call center are a clear example of a Poisson process, so we decide to use an exponential distribution for generating instances in our simulation. Since we know that phone calls occur at a rate of 50 per day, we enter 50 in the instances field and "0 Months 1 Days 0:0:0" in the interval field. The distribution details now read: *Average frequency: 50 instances, every: 0 Months 1 Days 0:0:0.*

Activities tab

For each designed activity you have to complete the properties to determine the simulation behavior.

Activity panel: All the activities are displayed in this panel

Depending on the activity, different properties can be defined.

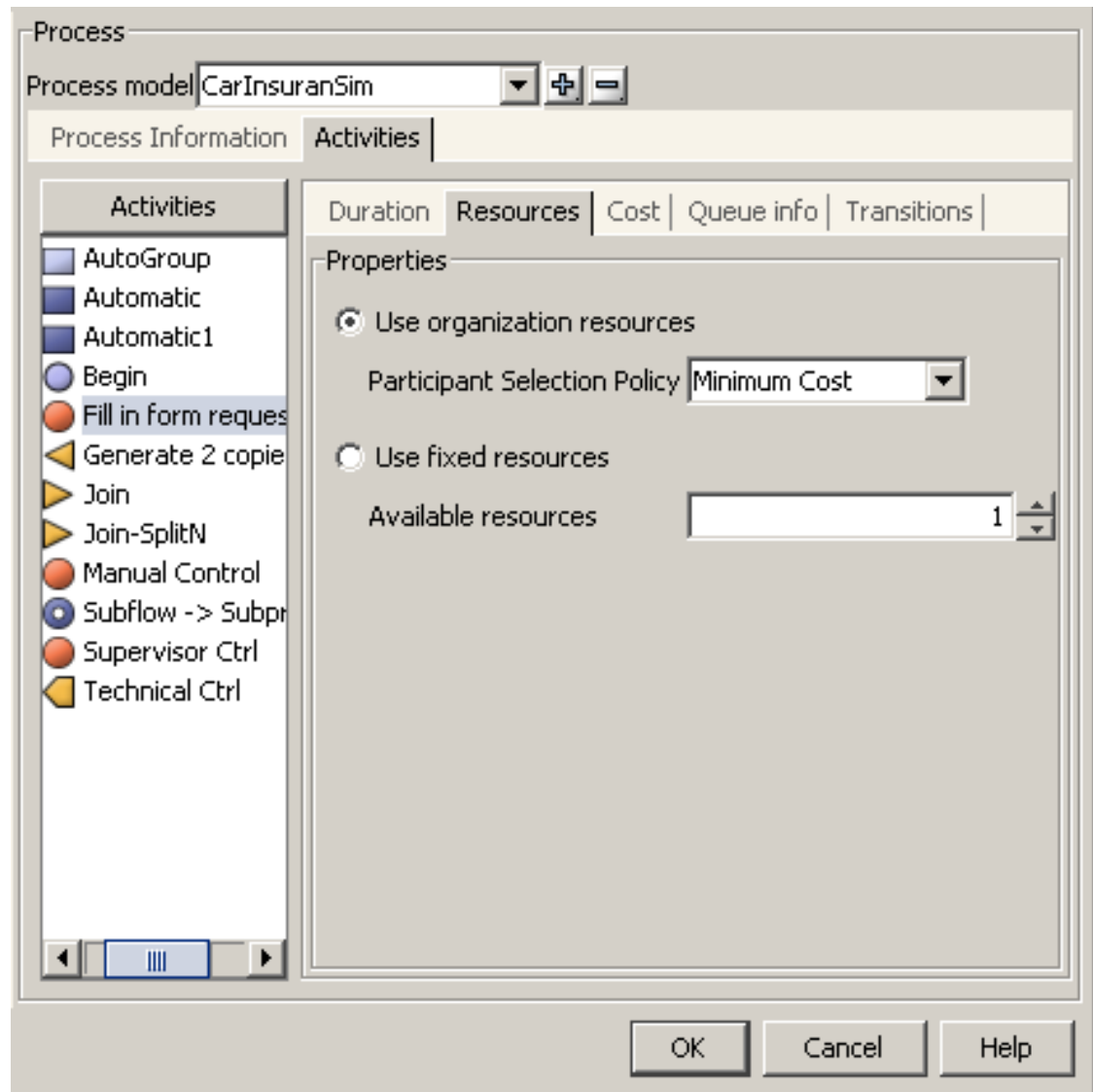


Duration:

The instance remains in the activity depending on the selected **distribution type**. The same happens with instance generation in the process information configuration. Remember that the simulation

DOES NOT execute the tasks, so this should be an estimated execution time of the activity's task.

Resources:

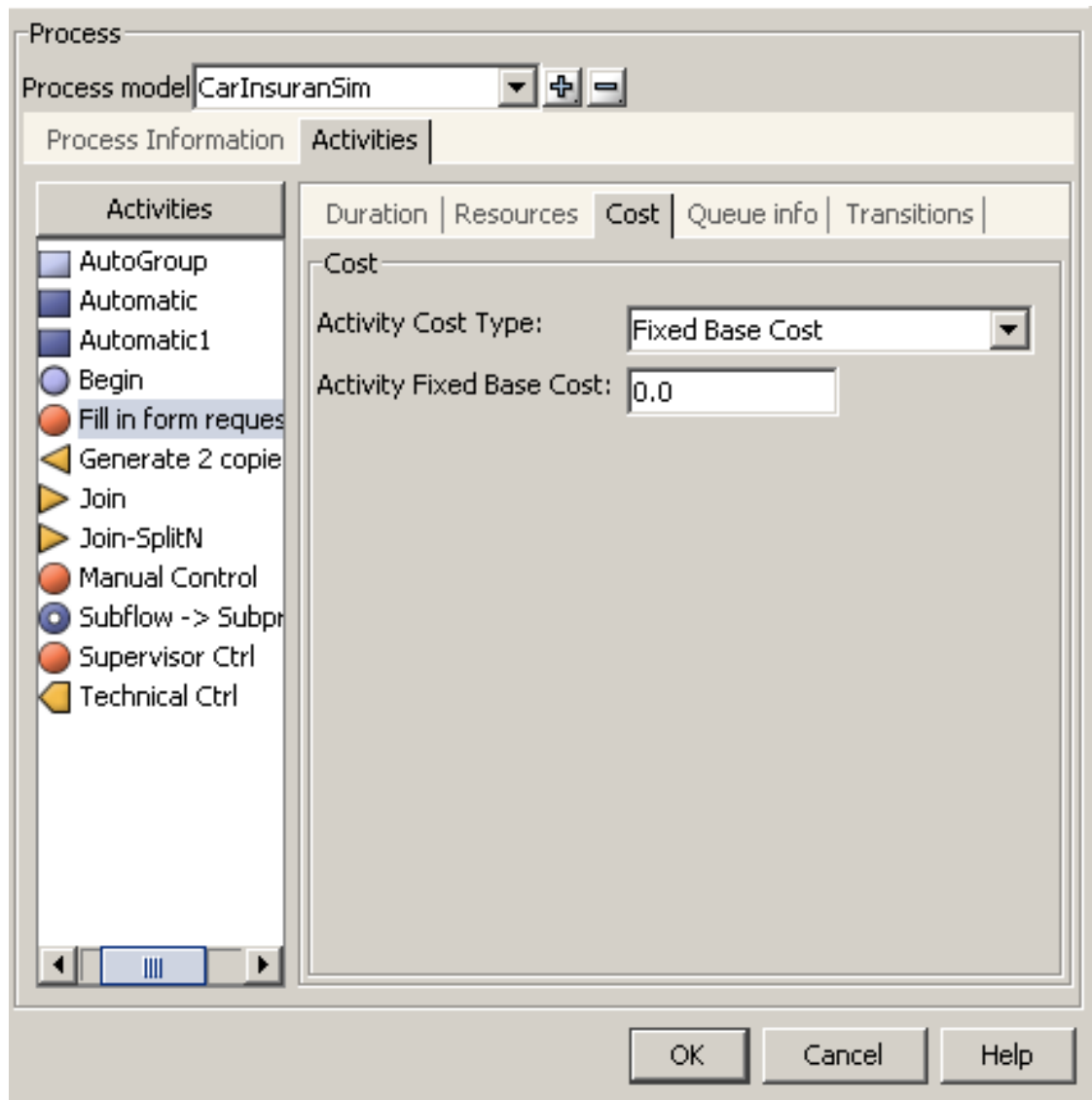


This tab is only available for **Interactive** activities.

- **Use Organization resources:** the simulation model sets all the resources from the project simulation model definition.

- **Participant Selection Policy:** the selection can be based on the Minimum Cost, Maximum efficiency or Randomly. Cost and Efficiency values are those defined in the project simulation model definition for each participant.
- **Use fixed resources:** Indicates the number of participants assigned to the Interactive activity. This option is used when costs and efficiency parameters are not relevant in the evaluation but only the amount of resources is needed. When selecting this option, the resources in the **Project Resources** tab are not taken into account.

Cost:



The cost represents the value of a resource of this activity.

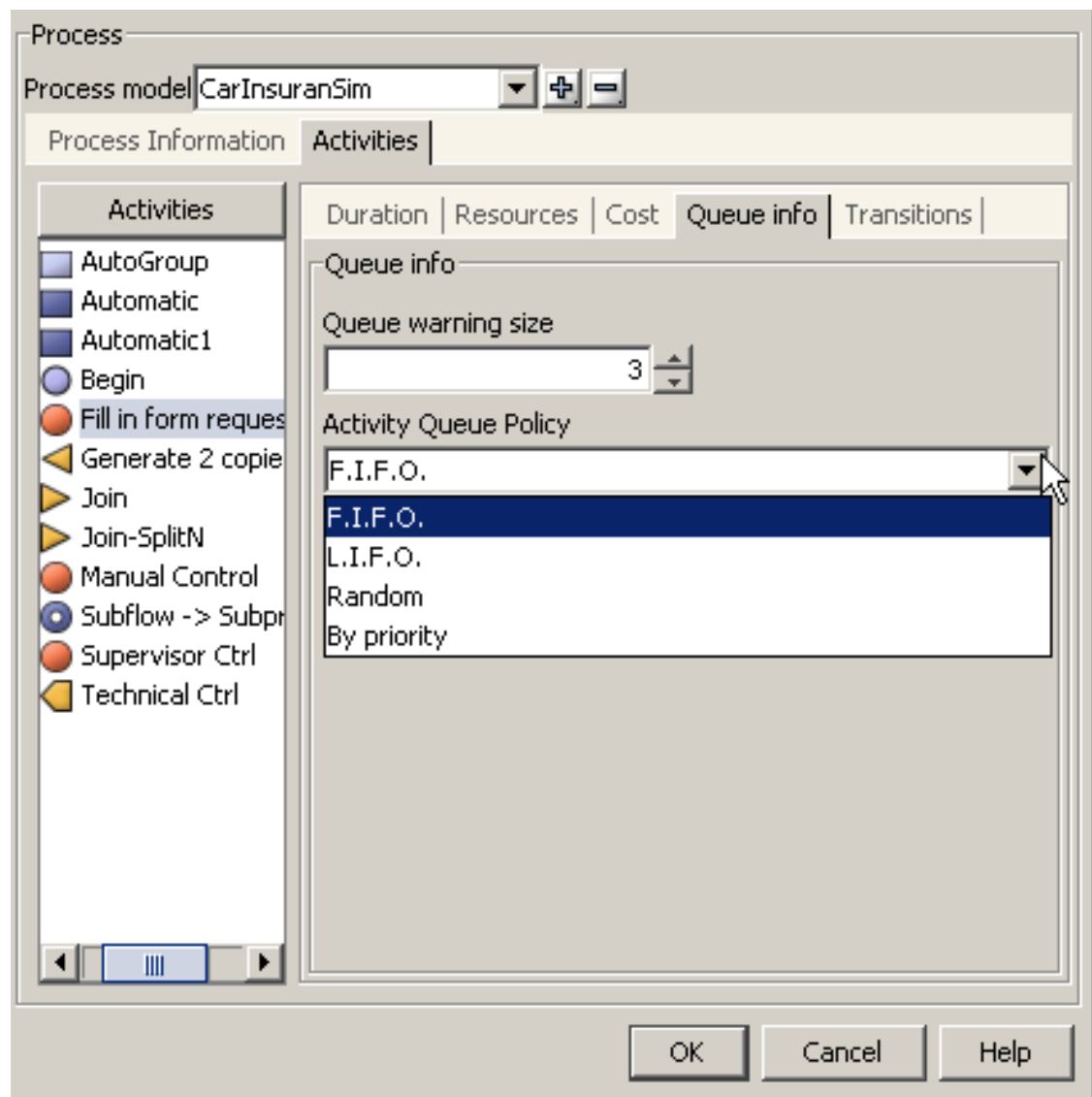
- **Activity Cost type:** it can be defined as a **Fixed Based Cost** or as a **Fixed Based Cost + the Resource Cost**. The resource cost is defined in the Project simulation model. Cost is used for reporting purposes.

The Fixed Based Cost applies each time an instance is processed. The Resource Cost is calculated based on the define cost per hour and the time it takes the resource to execute the instance.

For example: You have defined an interactive activity "Ship the order". The cost to wrap the package is a fixed cost set to \$2 (box, paper to wrap, etc). It takes the clerk 10 minutes to do this work and the cost per hour is \$6.

Therefore the calculated cost for each processed instance (order) would be: \$2 (fixed based cost) + \$1 (resource cost: 10 minutes of the resource defined as \$6 per hour)

Queue info



- **Queue warning size:** when the queue size exceeds the defined one, while the simulation is executing, the queue becomes red.
- **Activity Queue Policy:** Instances are executed F.I.F.O, L.I.F.O, Randomly or by Instance Priority.

Transitions:

Process

Process model: CarInsuranSim

Process Information | Activities

Activities

- AutoGroup
- Automatic
- Automatic1
- Begin
- Fill in form request
- Generate 2 copies
- Join
- Join-SplitN
- Manual Control
- Subflow -> Subprocess
- Supervisor Ctrl
- Technical Ctrl

Transitions

Due interval: 0 Months 0 Days 0:0:0

This table determines the probability of each outgoing transition or exception occurring. The probabilities listed must total 100 percent.

Transition	Probability
Exit because of an exception	1
Aborted instances	1
-> End	5
-> AutoGroup.Begin	93

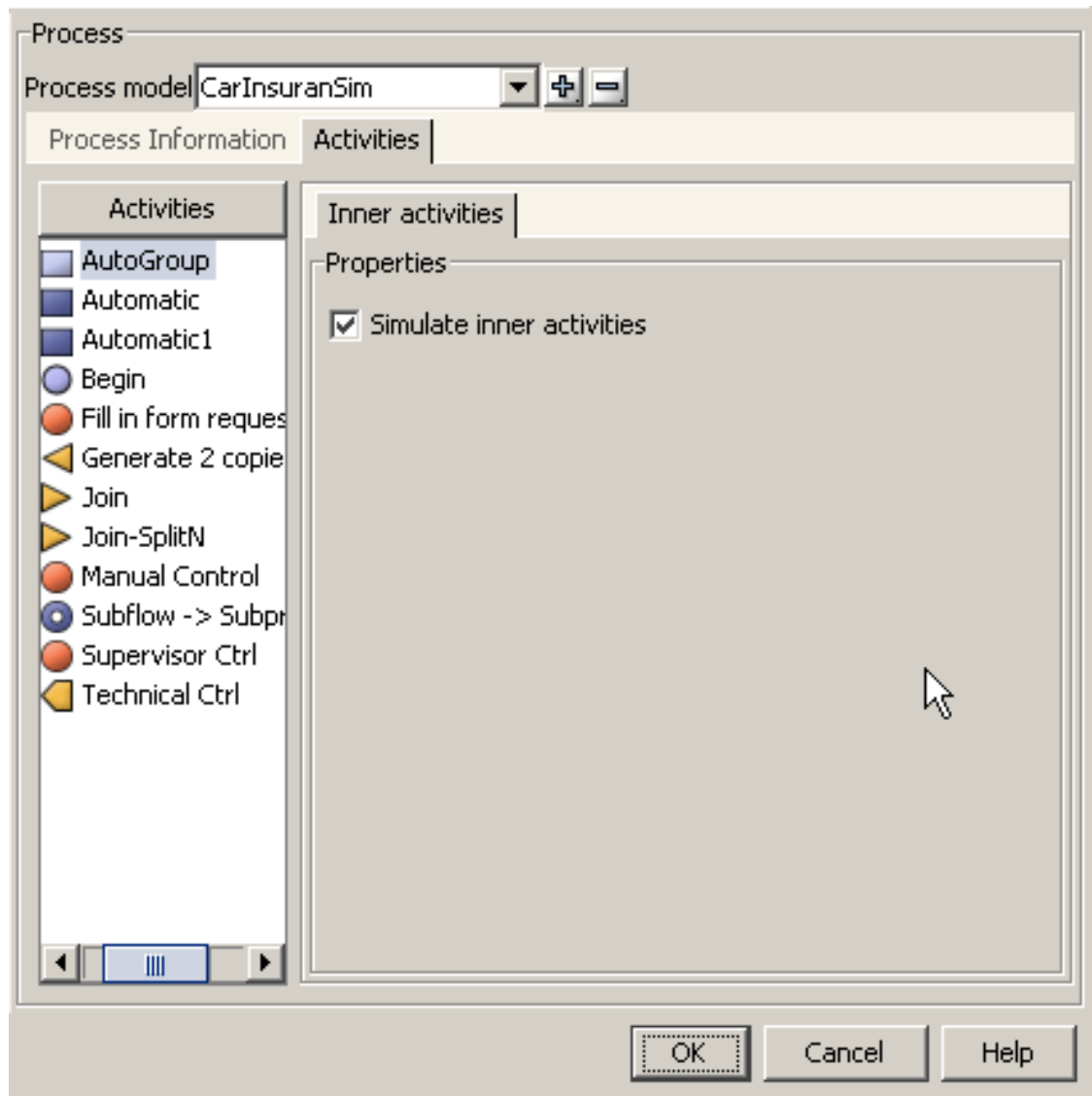
OK Cancel Help

- **Due interval:** if the activity has a due transition and its condition

is not constant, as it depends on a calculation or an instance variable, you need to define the due interval. Remember that during simulation **no methods** are processed.

- **Transition/Probability:** each designed transition populates in the combo box to define the probability that the instance will flow through it. In addition, by default, there two added transitions in the list:
 - **Exit because of an exception:** defines the probability that an instance can produce an exception.
 - **Aborted instances:** defines the probability that an instance can be aborted.

Inner activities:



This tab is only available for **Group** activities.

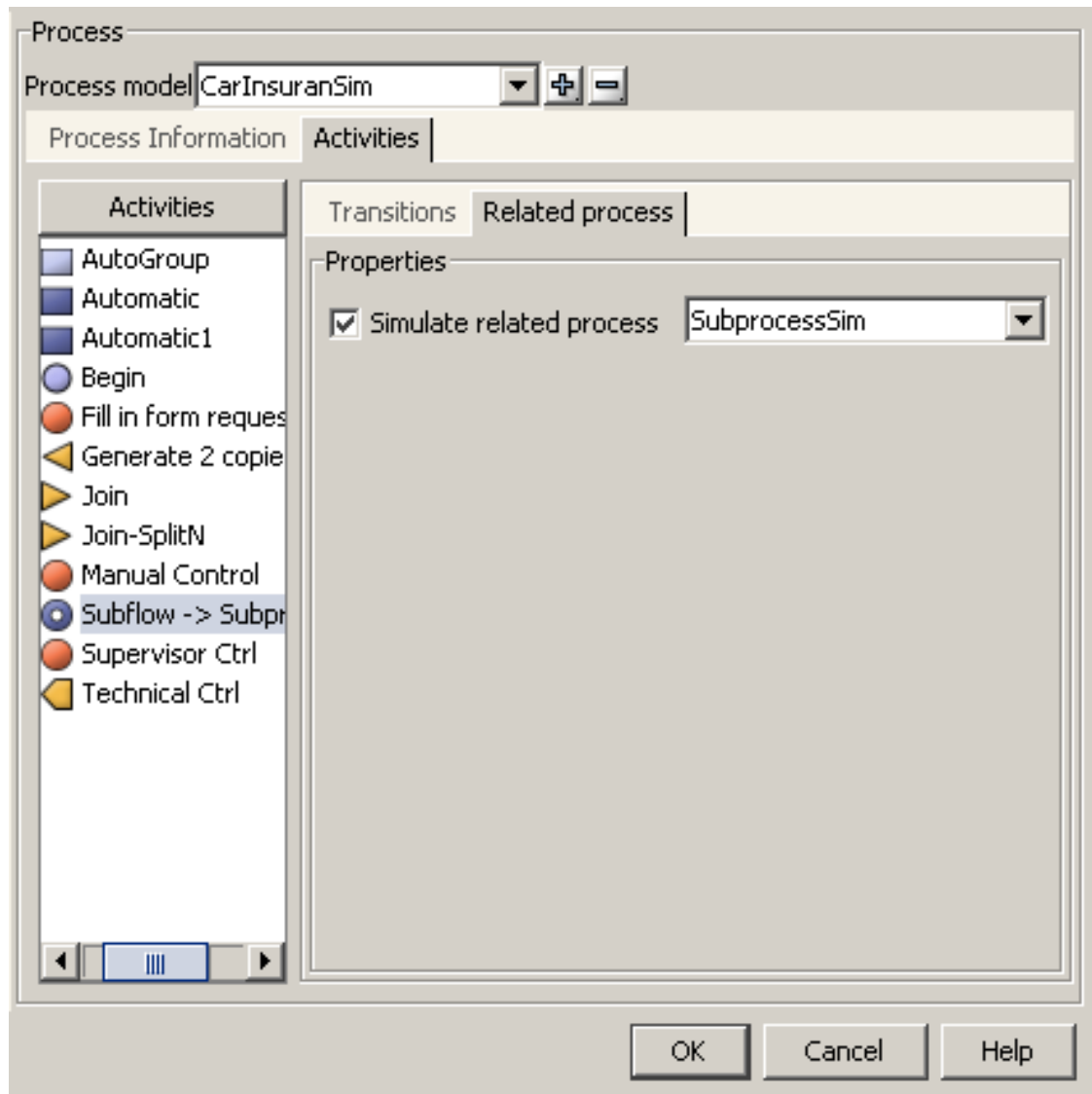
- **Inner activities:** if checked, you have to configure the simulation for all inner activities. Open the group tree and for each displayed activity implement the simulation. If the check box is disabled, you configure the group as if it was a single activity.

Warning



Grab, Global and End activities cannot be simulated.

Related Process:




This tab is only available for **Subflow** activities.

- **Simulate related process:** if checked, you have to define which Simulation model of the related process will run when you execute the Simulation. At runtime, the simulation is executed based on the configuration of the subprocess model as if it were

actually executing the subflow. If the check box is disabled, the Subflow activity will be treated as a simple activity and no subprocess is executed. You have to define all the information for the regular tabs (*Duration*, *Cost*, *Queue info* and *Transitions*).

Note


 If you enable the "Related process" check box, be sure you have defined a simulation model for the related process.

Modifying a Simulation Model


Once you define a Simulation Model, you can change any property by clicking on



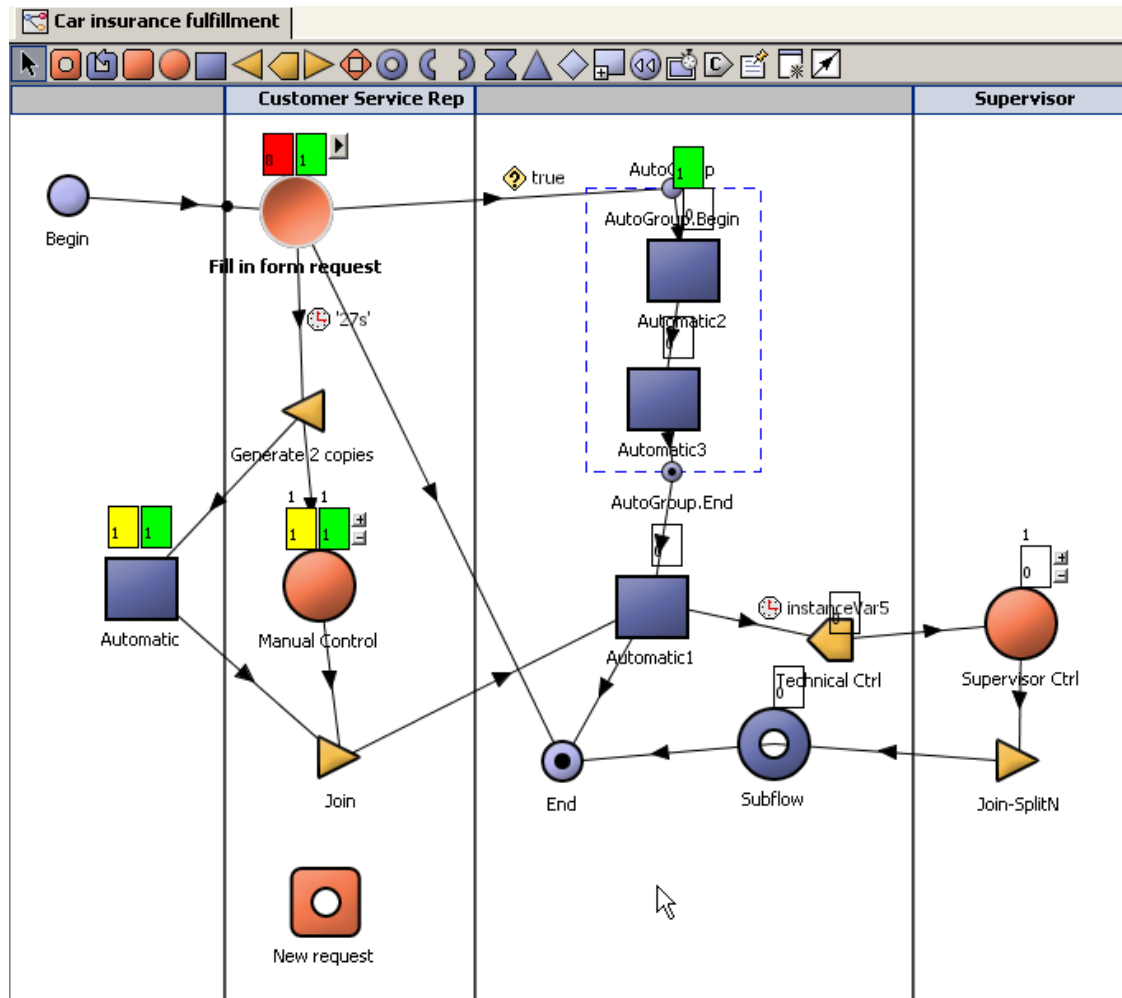
Executing a Simulation Model

To begin executing a defined Simulation model, select the **Project Simulation model** and click on the **Start** button. 


Note

 Be sure that you have selected at least one process as **Run Simulation** in the Project Simulation model.

The simulation is executed and on line behavior is shown on your workspace.



Note

 You can see the properties of the activity during the simulation by right-clicking on the activity and selecting properties.


Flowing instances

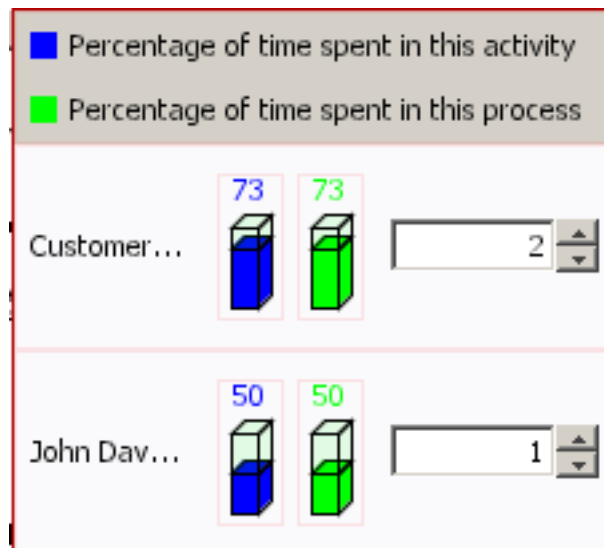
All generated instances are represented as a dot flowing through the process.

Occupancy bar

Each activity has one rectangle or **Occupancy bar** that represents the instances that are being processed (the rectangle on the right).

The occupancy bar is blank if no instance is processed at that time, or green with the number of instances that are executed. The maximum number of resources defined for that activity is on the top of the bar.

- If you defined to use **Fixed Resources**, you can dynamically change the number of resources by clicking on "+" or "-" signs next to the bar.
- If you defined to use the **Organization Resources**, you can modify the resources by clicking on the  icon and a graphic populates.



You can visualize for each resource the **Percentage of time spent in the Activity** and the **Percentage of time spent in the Process**. You can modify the number of resources for each type.

Queue bar

There can also be a second rectangle or **Queue bar** that represents the queue for that activity (the rectangle on the left). The queue bar is red when the Queue size warning is exceeded. When not exceeded, it is yellow. Above this bar you can see the maximum number of instances that the queue reached during the running of

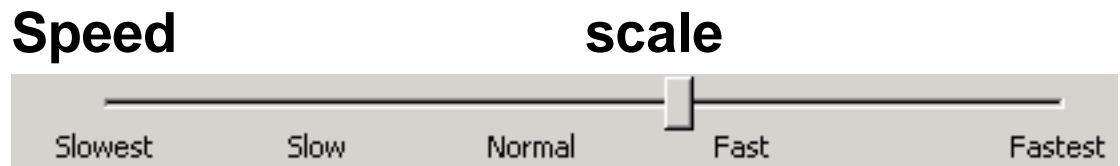
this simulation model.

Stop /Pause (■ / ||)

The simulation execution stops or pauses. Once paused, click the **Run** button in order to continue.

Run to the end (▶)

When you select this button, the simulation runs in the background. No animation is shown. The simulation process is much faster.



You can slow down or accelerate the simulation execution by using the speed bar from Slowest to Fastest.

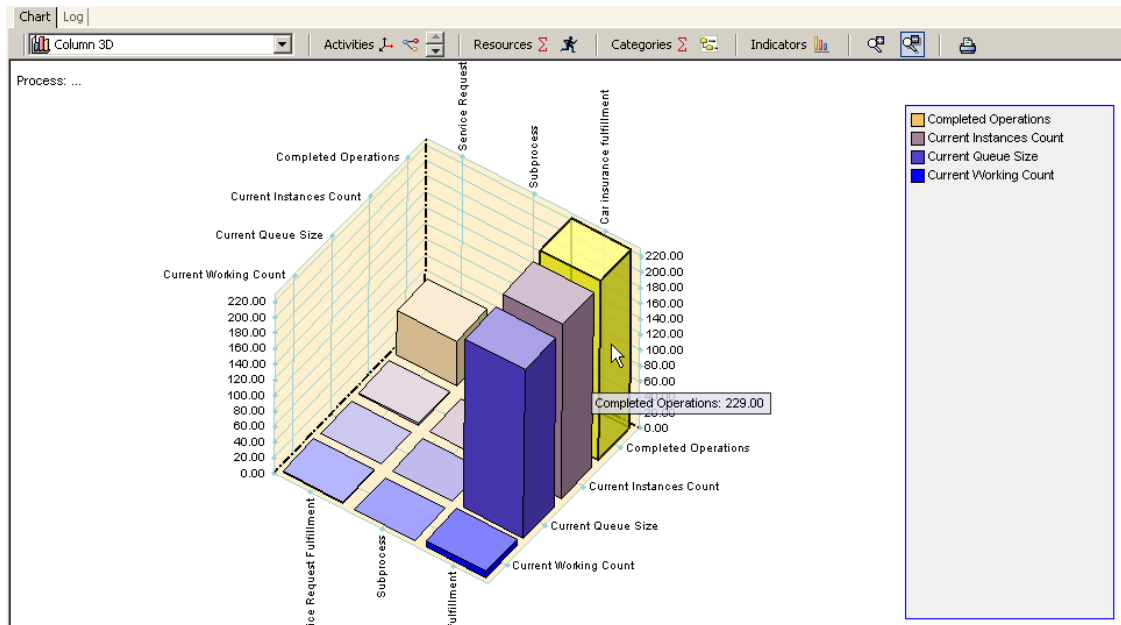
In **Normal** speed, the instances are generated each second. All defined times within the model are scale- based on this.

Changing the simulation model parameters

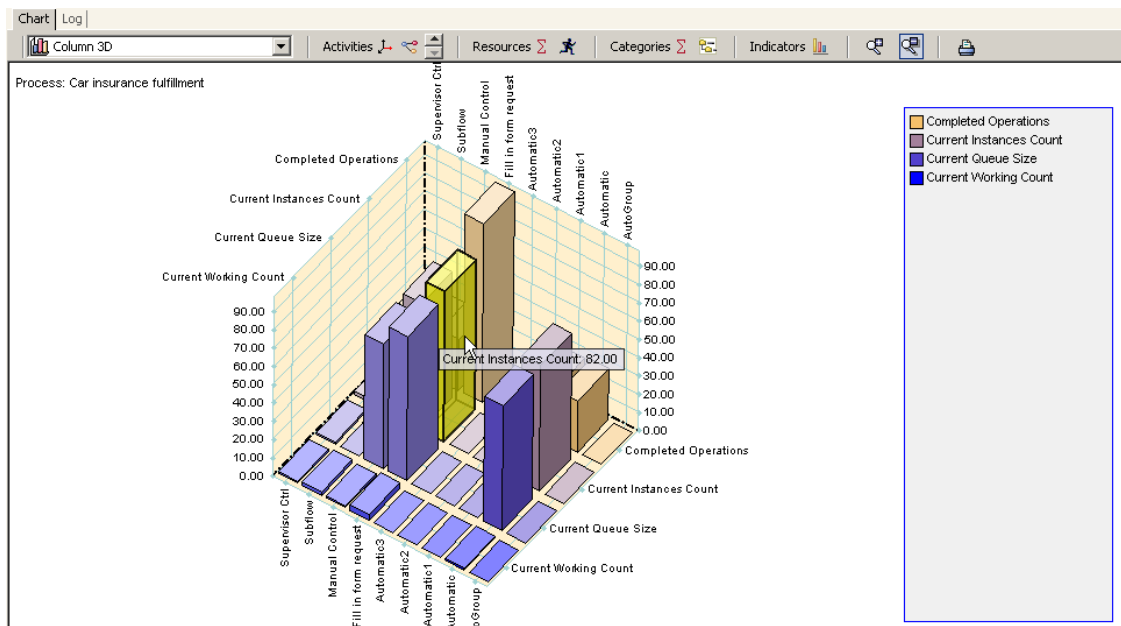
Some of the simulation model's properties can be changed online while the simulation is running. You can dynamically change the number of resources by clicking on "+" or "-" signs next to the bar. Furthermore, you can edit the configuration and change any of the properties and see the new behavior online.

Simulation results

Results can be seen on the lower part of the Simulation Flap.



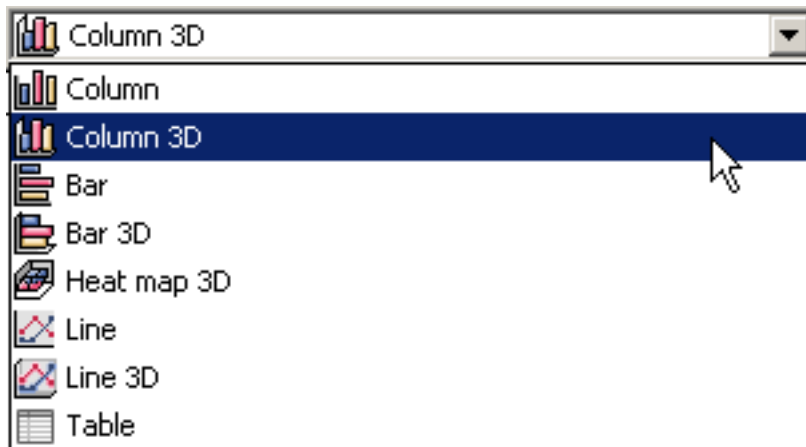
Click on any of the process and the graphic is shown for that particular process and not for all

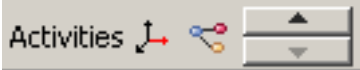


Selecting the representation graphic type

Choose the graphic type in which you want to see the results from

the combo-box on the left top side of the Chart tab. The options are:



The information is shown by process or by activity. To switch between both layers, click the up and down arrows beside the *activity* dimension, . The same can be done if you are

watching the **process level**, left click on any result in the chart and the information is shown by **activity**. To return to the process view, click Ctrl + left click.

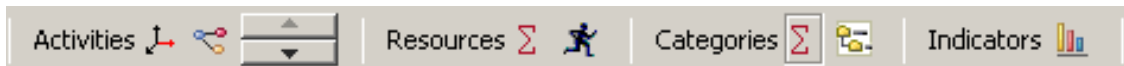
The graphic can be zoomed in and out by clicking the



icons in the chart tool bar.

Setting to view the Simulation Results


You can combine the following dimension for analysis to monitor your business process:





Indicators are always present in the simulation analysis and they are represented in the *Y axis* of the chart.

Combine two out of the three possibilities left, *Activities*, *Resources*, and *Categories*, to fill the "X" and "Z" axis of the chart. Icons to do this

are:

 : It will be represented in the X axis of the graph.

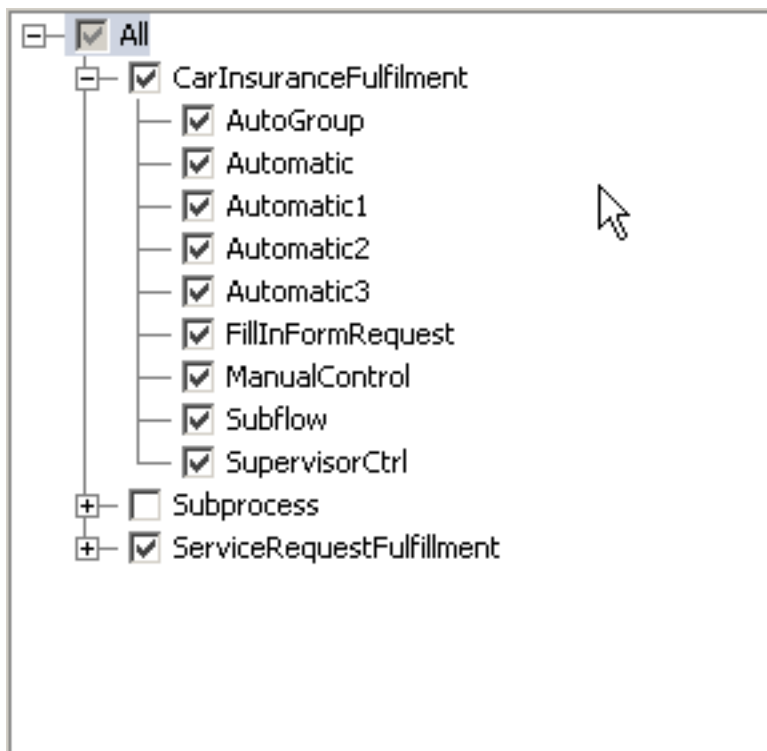
 : It will be represented in the Z axis of the graph.

 : The results will be shown for all the selected values.

Click the following icons next to the available measurements to select the values occurrences to include in the simulation view results:

Activities: 

When you click this icon, a window is displayed showing all the processes/activities.



Resources: 

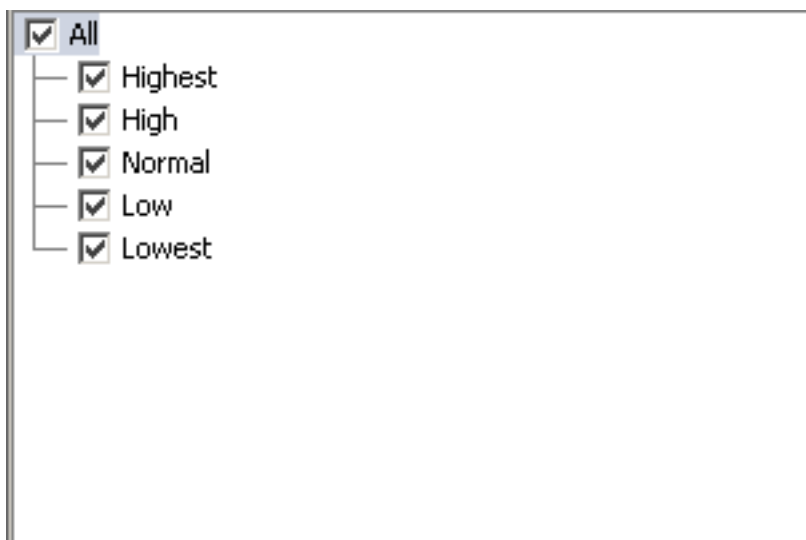
When you click this icon, a window is displayed showing all the

resources defined for this process.




Categories: 

When you click this icon, a window is displayed showing all possible values that the instance priority can have.



Indicators:


When you click this icon, a window is displayed showing all possible indicators you can choose to show in the simulation results.


Available Indicators depend on the Resources  dimension.

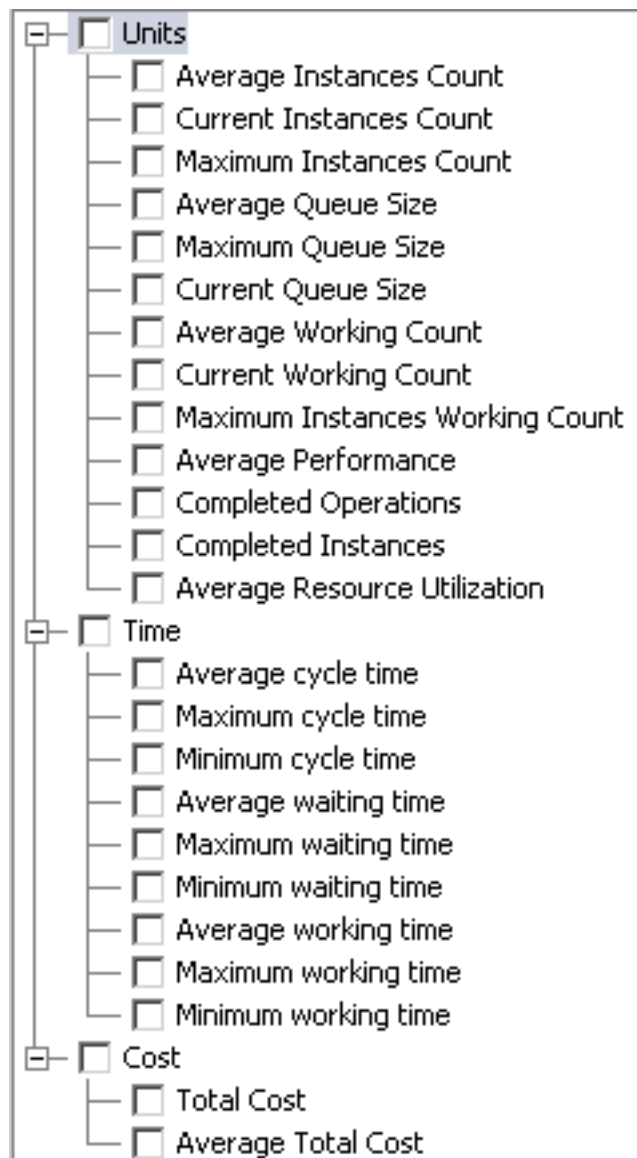
Proceed in the same way for all them:

- click the root check-box, to select/unselect all of them,
- deselect those you want to exclude, or
- select those you want to include.

Indicators when the Resources dimension is not selected

When you click the  icon in the Simulation toolbar, a window is displayed showing the possible measurements you can choose to show in the simulation results.

Indicators vary depending on the dimensions selected. This is the list of measurements shown if the  Resources (participants) dimension has not been checked. The indicators shown below are called Instance / Activity Indicators:



Instance/Activity indicators:

- **Units**

- *Average Instances Count:* The average number of instances processed during the simulation. This is the sum of the Average Working Count and the Average Queue size. When the process view is shown, this amount is the sum of the individual average instances counts for each activity in the process.

When the individual activities of the process are shown, this is the number of instances passing through each individual activity.

- *Current Instances Count*: The number of instances being either waiting to be processed (queued) or are being processed right now. When the process view is shown, this amount is the sum of the individual current instance counts for each activity in the process. When the individual activities of the process are shown, this is the number of instances passing through each individual activity.
- *Maximum Instances Count*: The total maximum number of instances being processed. This is the sum of the Maximum Instances Working Count and the Maximum Queue Size.
- *Average Queue Size*: Average number of instances that waited at an activity.
- *Maximum Queue Size*: Maximum number of instances that waited at an activity.
- *Current Queue Size*: Number of instances that either are (1) waiting at the activity right now (while the simulation is running) or (2) were waiting at the activity when the simulation completed. When the process view is shown, this amount is the sum of the individual queue size counts for each activity in the process. When the individual activities of the process are shown, this is the number of instances waiting to be processed for each individual activity.
- *Average Working Count*: The average number of instances being processed. When the process view is shown, this amount is the sum of the average individual working counts for each activity in the process. When the individual activities of the process are shown, this is the average number of instances working for each individual activity.

- *Current Working Count*: The total number of instances being processed. The amount shown is either the number: (1) being processed at the activity right now (while the simulation is running) or (2) were being processed at the activity when the simulation completed. When the process view is shown, this amount is the sum of the individual current working counts for each activity in the process. When the individual activities of the process are shown, this is the number of instances currently being worked for each individual activity.
- *Maximum Instances Working Count*: The maximum number of instances being processed. When the process view is shown, this amount is the sum of the individual maximum working counts for each activity in the process. When the individual activities of the process are shown, this is the maximum number of instances being worked for each individual activity.
- *Average Performance*: The average number of instances processed per hour.
- *Completed Operations*: Number of instances that have completed an activity during the simulation. When the process view is shown, this amount is the sum of the instances that have passed through (completed) each activity in the process. When the individual activities of the process are shown, this is the number of instances passing through each individual activity.
- *Completed Instances*: Number of instances that have completed an activity or the process during the simulation. The difference with the *Completed Operations* is when the process view is shown, this number is the total number of instances that completed the process. When the individual activities of the process are shown, this is the number of instances passing through each individual activity.
- *Average Resource Utilization*: The average number of resources assigned to each activity.


- **Time**

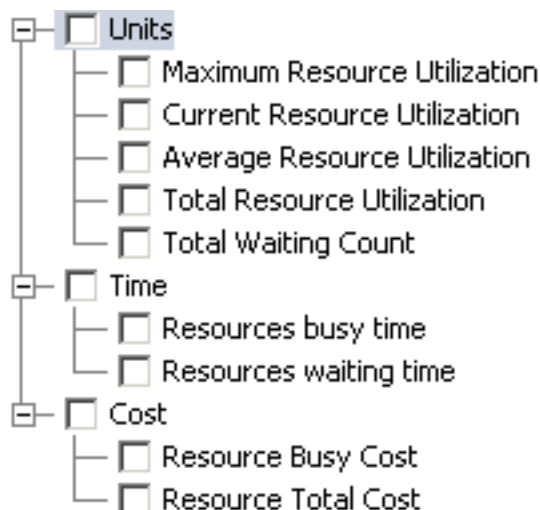
- *Average Cycle Time*: Average time, or simulated 'wall-clock' time during which instances were at the activity. This is the sum of the average working time and the average waiting time.
- *Maximum Cycle Time*: Maximum amount of time, or simulated 'wall-clock' time, that instances were at the activity. This includes the maximum working time and the maximum waiting time.
- *Minimum Cycle Time*: Minimum amount of time, or simulated 'wall-clock' time, that instances were at the activity. Includes work and wait time.
- *Average Waiting Time*: Average time instances spent waiting at the activity.
- *Maximum Waiting Time*: Max amount of time instances spent waiting at the activity.
- *Minimum Waiting Time*: Min amount of time instances spent waiting at the activity.
- *Average Working Time*: Average amount of time that work was actively being performed on each instance going through the activity.
- *Maximum Working Time*: Maximum amount of time that work was actively being performed on any instance at the activity.
- *Minimum Working Time*: Minimum amount of time that work was actively being performed on any instance at the activity.

- **Cost**

- *Total Cost*: Total cost of executing the process for all the instances run through it. When the process view is shown, this amount is the sum of the individual activity cost totals. When the individual activities of the process are shown, this is the total cost for each individual activity.
- *Average Total cost*: Average cost of executing an instance through the process. When the process view is shown, this amount is the average of all the individual activity cost averages. When the individual activities of the process are shown, this is the average cost for each individual activity.

Indicators when there are Resources selected

If you have selected the Resources  dimension for an axis and at least one participant is checked, the indicators will instead be resource related. Shown below is the list of measures called Resource Indicators:



- **Units**

- *Maximum Resource Utilization*: Maximum number of resources assigned
- *Current Resource Utilization*: Number of instances currently being worked on by the participants checked in the Resources.
- *Average Resource Utilization*: Average number of resources working on an instance at any time
- *Total Resource Utilization*: The total number of instances executed by all the participants during the entire execution of this simulation.
- *Total Waiting Count*: number of instances that had to wait for a resource before they were worked on.

- **Time**

- *Resource Busy Time*: amount of time the resource is busy processing the instance.
- *Resource Waiting Time*: amount of time the resource is waiting to process the instance.

- **Cost**

- *Resource Busy Cost*: Resource cost incurred while actually working on instances
- *Resource Total Cost*: Cost of Resources whether they are busy or

idle.

Viewing Chart results

- You can zoom in or zoom out any type of chart by clicking Ctrl + left mouse button + moving the mouse horizontally
- Additionally, in any 3D chart you can rotate the graphic by clicking Shift+ left mouse button + moving the mouse horizontally

Comparing two simulation models

Two flaps are available to compare the results of two simulation models running. The first time you run the simulation from the designer toolbar, **Flap 8** is created to run a first model. If you run another simulation from the designer toolbar, a new flap is created, **number 10**. Use the two flaps to compare results by switching between the flaps. This allows you to perform what-if analysis.

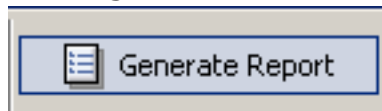
Log Tab

Shows the log of all actions performed during the simulation, step by step.

Reporting

Settings

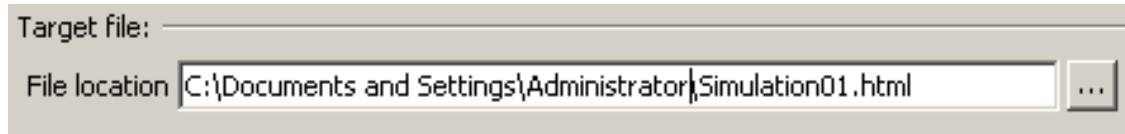
To generate result simulation reports click the



icon. The *Reporting* dialog opens where you

can configure the information to be included in the report.

Browse and select the file location and name for the report you are about to generate in the first area of the dialog.



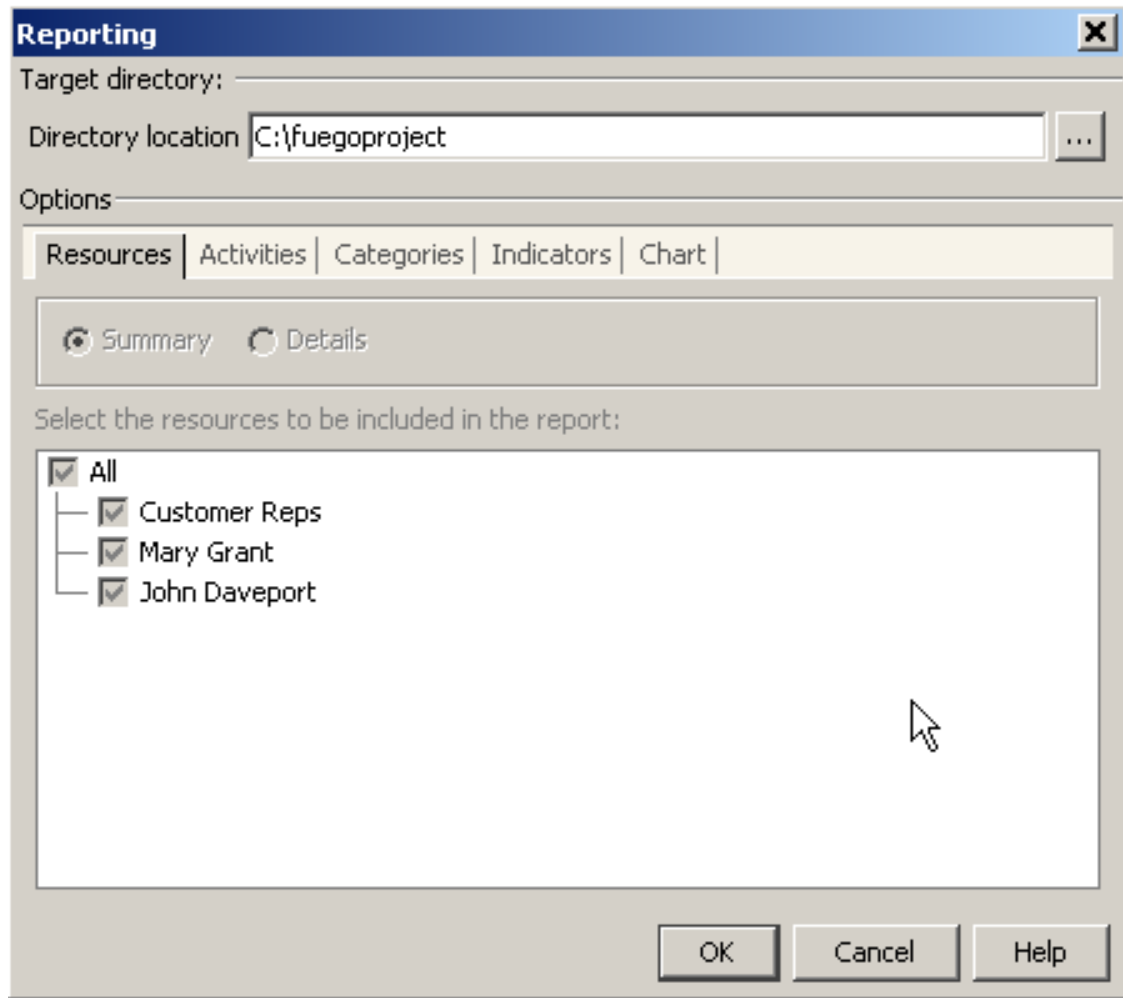
In the *Options* area of the dialog, you are able to configure some of settings to generate the report.

Five tabs are provided to do this, one for each possible dimension to select the graphic type to include in the report: *Activities*, *Resources*, *Categories*, *Indicators*, and *Chart*.

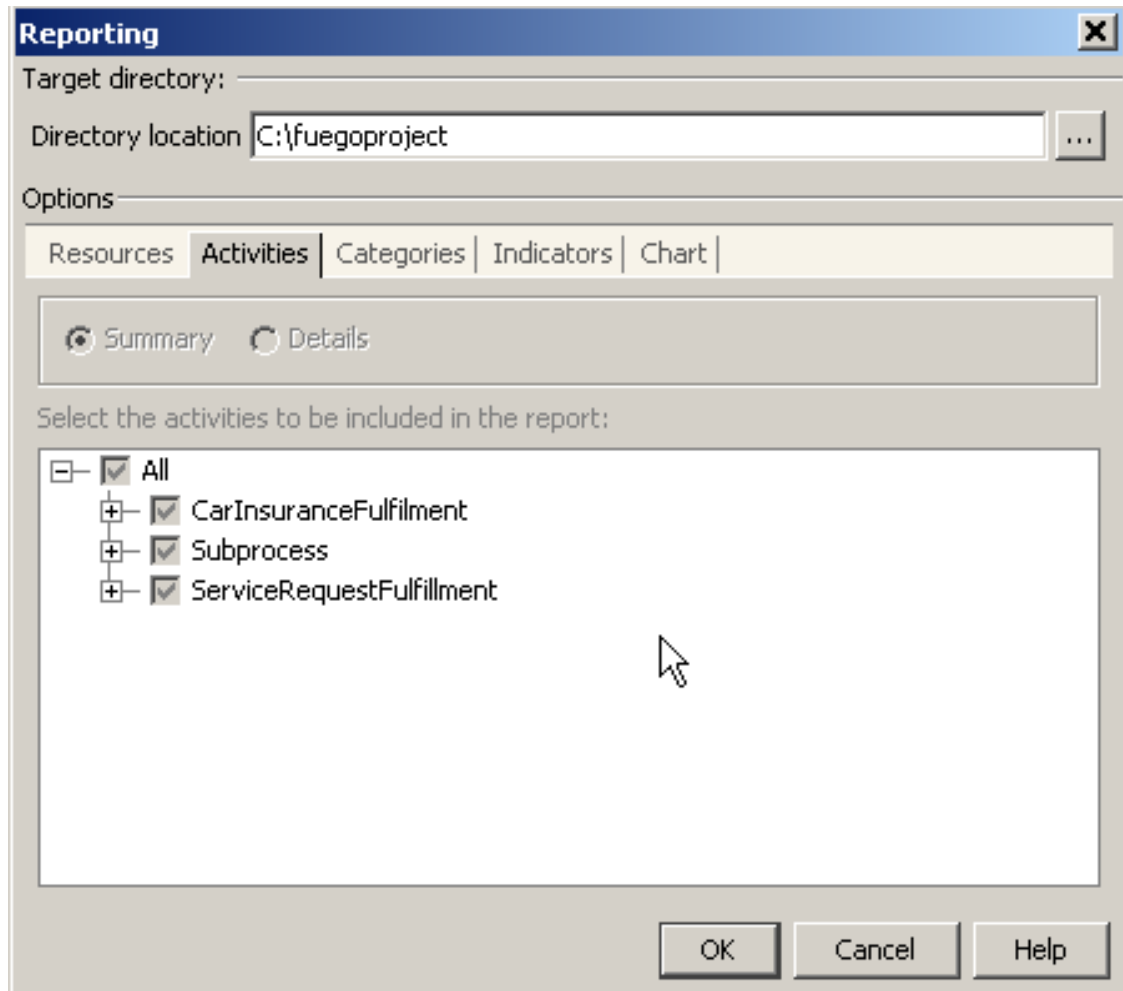
The simulation result settings are used as default values for the report generation. The dimensions selected as fixed for the *X* and *Z* axis cannot be changed at report generation time. They are displayed, but you do not have edit access to them.

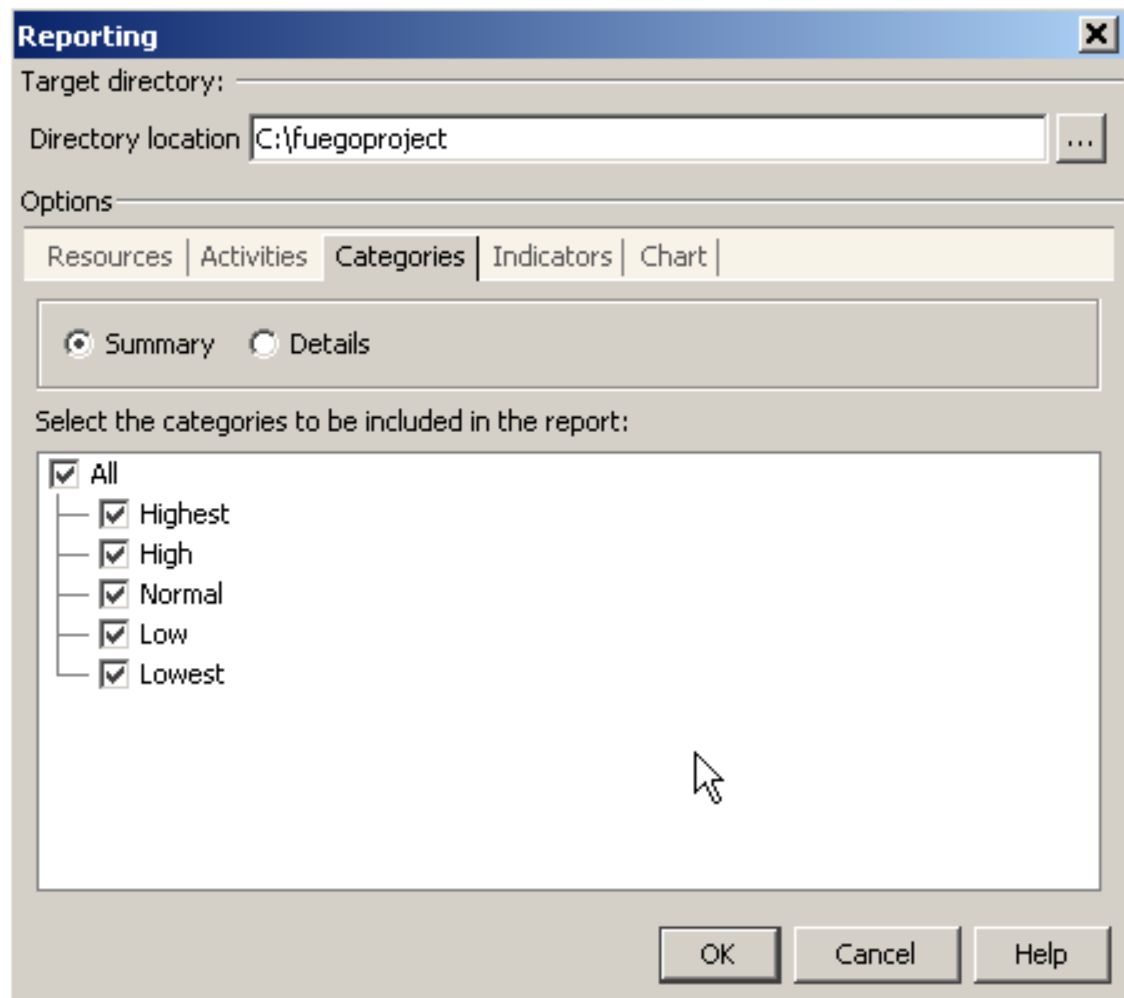
Resources Tab:

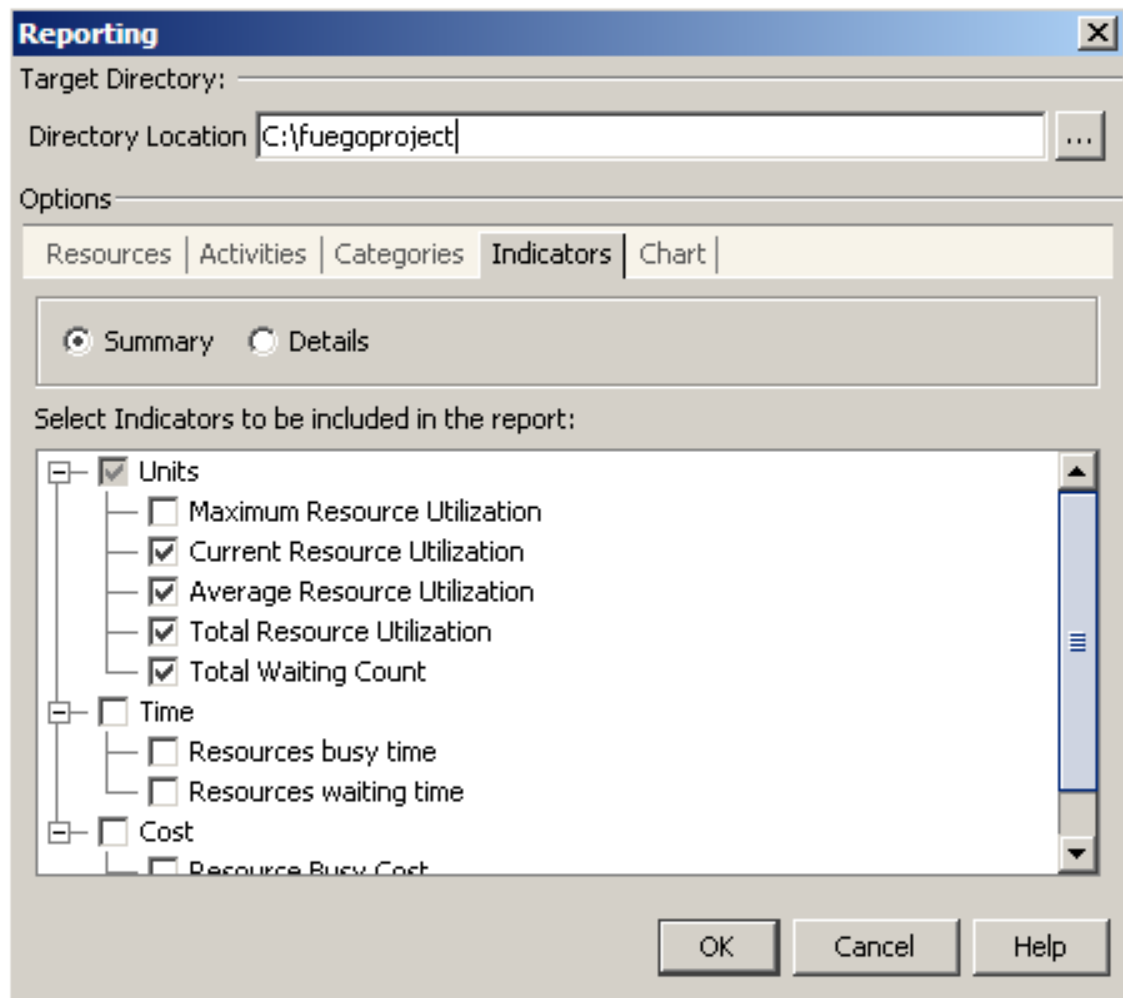
In this example, Resources dimension has been set as *X* axis.

**Activities Tab:**

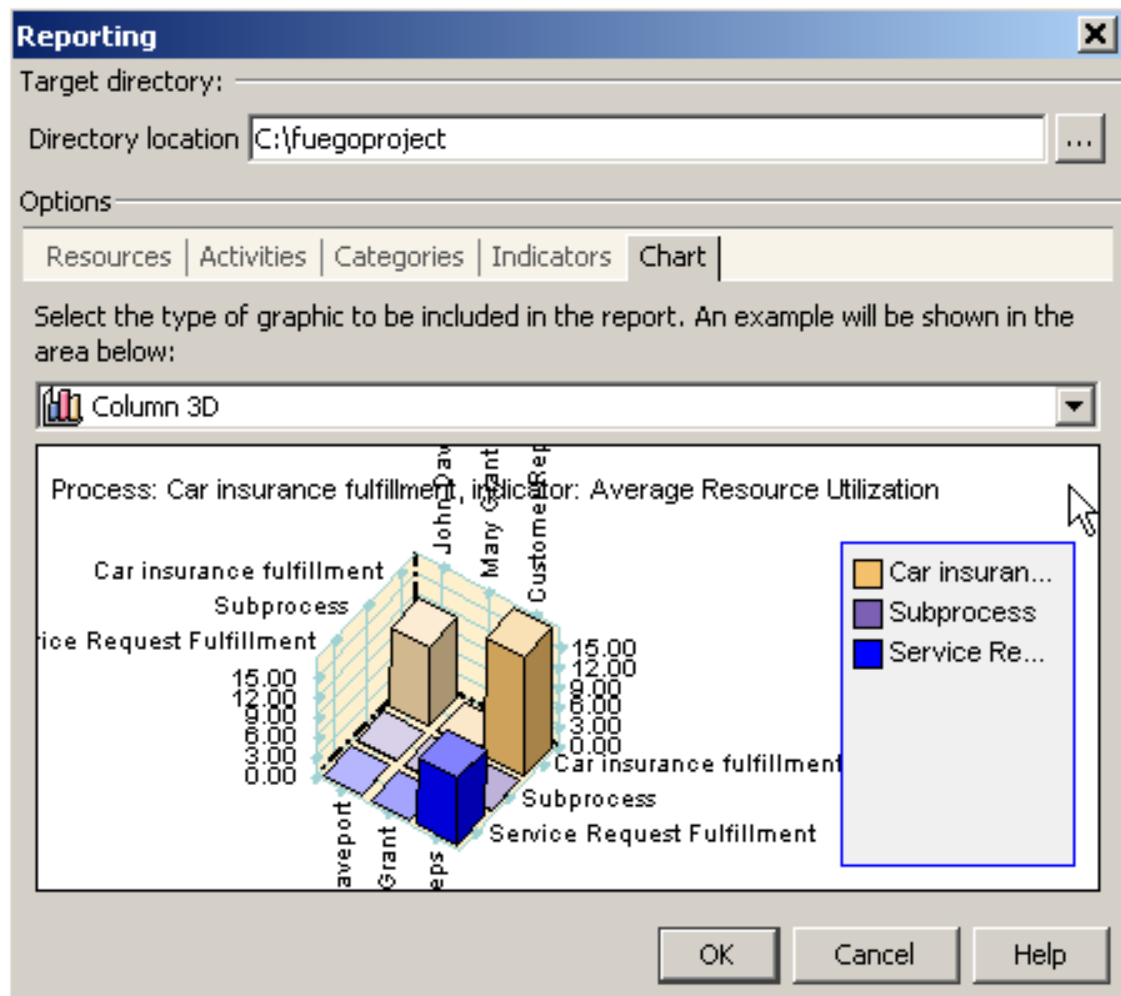
In this example, Activities dimension has been set as Z axis.

**Categories Tab:**

**Indicators Tab:**

**Chart Tab:**

Select the type of graphic to include in the report. An example is shown in the area below the options.



Dimensions that are editable or enabled can be chosen as *Summary* or *Details*. However, only one of them can be included as *Details* in a report. If you select *Summary*, click the possible values for the dimension you want to summarize.

Generated Report

Simulation report generated shows for each process:

- Simulation Data: When you click the link, you will see details of the simulation data regarding instances used to create the report

- Simulation Resource Data: When you click the link, you will see details of the simulation data regarding resources used to create the report.
- Simulation Chart: one or more, depending if the *Details* dimension was selected.

The Simulation Data and Simulation Resource Data are generated in files with extension *.csv* (comma separated values) so you can import them into tools to see this information in different graphics or pies.

As the directory "c:\\fuegoproject" was selected, you will find there files as:

- ModelName_ProcessName.csv (with these columns: Activities,Categories,Indicators,Values)
- ModelName_ProcessName_Resources.csv (with these columns: Activities,Categories,Resources,Indicators,Values)

For example:

- Simulation01_CarInsuranceFulfilment.csv
- Simulation01_CarInsuranceFulfilment_Resources.csv
- Simulation01_Subprocess.csv
- Simulation01_Subprocess_Resources.csv
- Simulation01_ServiceRequestFulfillment.csv
- Simulation01_ServiceRequestFulfillment_Resources.csv

Simulation01_CarInsuranceFulfilment.csv content will look like:

Activities,Categories,Indicators,Values,
Automatic,Highest,Average Instances Count,2.36,
Automatic,Highest,Current Instances Count,7,
Automatic,Highest,Maximum Instances Count,7,
Automatic,Highest,Average Queue Size,2.26,
Automatic,Highest,Maximum Queue Size,7,
Automatic,Highest,Current Queue Size,7,
Automatic,Highest,Average Working Count,0.11,
Automatic,Highest,Current Working Count,0,
Automatic,Highest,Maximum Instances Working Count,1,
Automatic,Highest,Average Performance,12,
Automatic,Highest,Completed Operations,1,
Automatic,Highest,Completed Instances,1,
Automatic,Highest,Average Resource Utilization,0,
Automatic,Highest,Average cycle time,58,Seconds
Automatic,Highest,Maximum cycle time,30,Seconds
Automatic,Highest,Minimum cycle time,30,Seconds
Automatic,Highest,Average waiting time,28,Seconds
Automatic,Highest,Maximum waiting time,28,Seconds
Automatic,Highest,Minimum waiting time,28,Seconds
Automatic,Highest,Average working time,30,Seconds
Automatic,Highest,Maximum working time,30,Seconds

Automatic,Highest,Minimum working time,30,Seconds

Automatic,Highest,Total Cost,0,

Automatic,Highest,Average Total Cost,0,

Simulation01_CarInsuranceFulfilment_Resources.csv content would look like:

Activities,Categories,Resources,Indicators,Values

Fill in form request,Highest,Customer Reps,Maximum Resource Utilization,0.0

Fill in form request,Highest,Customer Reps,Current Resource Utilization,0.0

Fill in form request,Highest,Customer Reps,Average Resource Utilization,28.0

Fill in form request,Highest,Customer Reps,Total Resource Utilization,6.0

Fill in form request,Highest,Customer Reps,Total Waiting Time,5.0

Fill in form request,Highest,Customer Reps,Resources busy time,162,Seconds

Fill in form request,Highest,Customer Reps,Resources waiting time,384,Seconds

Fill in form request,Highest,Customer Reps,Resource Busy Cost,0.36

Fill in form request,Highest,Customer Reps,Resource Total Cost,0.25

Simulation Example

Find this example in the **SimulationCase01.fpr** project in FuegoBPM Studio's installation directory, in studio/samples.

Chapter 15. Defining Business Objects levels

Defining Business Objects levels

Business Object and Business Services

A business object is an object that performs a defined set of operations, such as data validation or business rule generation.

Intelligent business objects (IBOs) abstract the complexities of one or more business services into an object that offers services through an interface. They are intelligent because they “know” how to integrate with the native services from applications or people, thus making the implementation transparent to the process that uses the same. Therefore, the process relies on IBOs to do the right things from the implementation perspective.

Business services are any kind of known activities such as a selling, financial work, or improvements. But when talking about orchestration, the business services definition embraces more concepts. It is anything that an employee or system of an organization performs on behalf of an internal or external customer. To be usable by an Server, a business service must expose an interface that makes it accessible for invocation by a computer program. The services are the instruments that are being orchestrated into a whole.

Cataloging Business Objects in FuegoBPM Studio

The business objects catalog in FuegoBPM Studio is named Project Catalog, as it is composed of all the **components** or Business Objects used by the project to meet the business requirements. By **components** we are not only referring to existing system components but also to any other resource the company manages

and that represents a business object, such as data base tables, JNDI, EJB, etc.

Each component contains the methods, attributes, and properties of a program or system to which it is required to connect. Several standard components are provided to help build and manage the project and its components.

Components and Fuego Objects

Fuego Objects can be used as containers to hold several unrelated components. In this way, you can create your own specialized business service by combining several different types of components into one Fuego Object.

An example of a Fuego Object containing multiple technologies is a Fuego Object working in an Order Entry process. When an order is requested, its information is received through a Fuego Object. After it has been submitted, a method is called, thus adding the order to a database.

Why should you use Fuego Objects as a container for components?

Using Fuego Objects as a container to hold your components provides you with the ability to continually improve and modify your process design by simply modifying the underlying components. The call to the Fuego Object never changes, even though the Fuego Object itself may have been modified.

Using Fuego Objects as containers also simplifies your process design. Activities' tasks only contain one quick call to a Fuego Object instead of multiple calls to different components.

A simple Fuego Object example

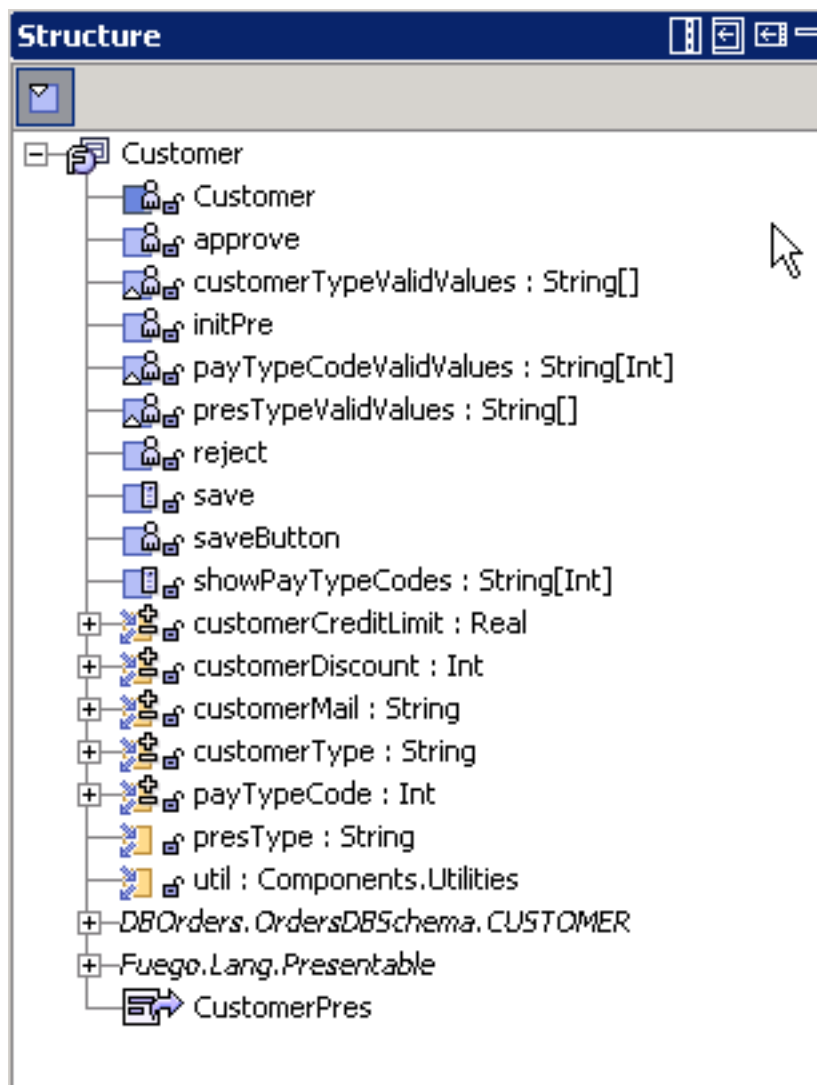
The following is an example of a Fuego Object modeling an Order from an Order Entry system. Since this is the first introduction of Fuego Objects, the example is very simple so that it can be easily

understood.

The order has the following information:

Customer, Order and Order Due Date, a Description, the group of items and quantity required, each item's price, and the total amount of the order. The order has a method name *storeToDB* that stores it into the database of a previously existing Order Entry system. The order has two presentations, one to input the information and another to approve it.

Customer Structure:



Customer Input Presentation:

The screenshot shows a web browser window with the title 'FUEGO Work Portal'. The header bar is blue and contains the text 'Welcome, fuego' followed by links for 'Search', 'Options', 'Help', and 'Logout'. A mouse cursor is pointing at the 'Search' link. The main content area is white and features a form titled 'New Customer'. The form has a light blue background and contains the following fields and controls:

New Customer	
Customer Code	Customer Name <input type="text" value="Joe Smith"/>
Payment Type <input type="text" value="CCard"/>	Customer Discount
Customer Type <input type="text" value="NATIONAL"/>	Credit Limit
Billing Address <input type="text" value="Park Ave.3910. Springfield"/>	
Shipping Address <input type="text" value="Park Ave.3910. Springfield"/>	
Mail <input type="text" value="jsmith@smith.com"/>	
<input type="button" value="Submit"/> <input type="button" value="Cancel"/>	

The footer of the browser window is a blue bar with the text 'FuegoBPM™ - Work Portal'.

Customer Approval Presentation:

FUEGO Work Portal Welcome, Fuego Search - Options - Help - Logout

Customer Commercial Approval

Customer Code 10 Customer Name

Payment Type Customer Discount

Customer Type Credit Limit

Billing Address

Shipping Address

Mail

FuegoBPM™ - Work Portal

Introducing Business Objects into FuegoBPM Project Catalog

Business Objects are implemented in FuegoBPM Studio, by cataloging in the Project Catalog. These components represent the Business Objects with which the project needs to interact. Cataloging components, either from existing or new systems, is the way to orchestrate business services that require performing a system task.

FuegoBPM Projects Catalog

The Project Catalog enables you to catalog the components and organize and group them into modules by some logical criteria. Therefore, a catalog contains the following elements:

- Modules  are containers that you create to logically group

components.

- Components are the representation of any Business Object that becomes available to be used during the process execution through the Process Methods.

Each project has its own catalog of components, so all the processes inside the project check and compile using this catalog. Every time a process is published, the jar file of the Fuego Object catalog is also published to the Directory Service. For further information on this topic, see Implementing Business Objects using Fuego Objects.

The Project Catalog:

- Supports the ability to efficiently store and manage up to 100,000 components.
- Supports unlimited nesting levels of components.
- Achieves the above requirements with minimum memory utilization and maximum performance.
- Provides simple wizards to simplify configuration and introspection of external resources Java, EJB, SQL, CORBA, etc.

The project catalog of components is stored in your local hard drive. The files are located in the *componentCatalog* directory that corresponds to the project.

For example,

C:/MyCompanyProjects/OrderFulfillment/OrderEntry/componentCatalog,

where *"/MyCompanyProjects/OrderFulfillment"* is a user-defined directory in which all projects will be stored. *"OrderEntry"* is the name of the Project.

Under the directory "componentCatalog", you will find the structure of your catalog, a directory tree that corresponds to the catalog modules, and xcdl files for each defined module and component.

When the project is published and deployed, the configurations defined when cataloging the components are deployed in FDI, thus automatically creating a pair. At the same time, the Fuego Object class is attached to the project. For the java components. For further information, see Publish & Deploy.

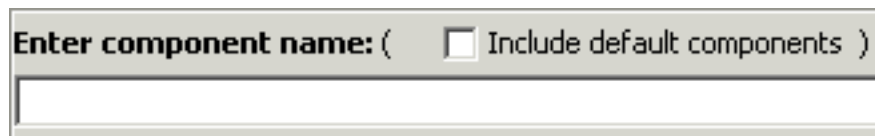
The Project Catalog includes a set of default components designed by Fuego. Go to the **Help** menu and clicking the **Component Index** option you get the list of all the Fuego standard components, the category or module to which it belongs to, and a brief description about it. If you double click on the component, a new tab in the main panel opens for that component. To learn about them please refer to Implementing Business Objects Using Fuego Blocks.

Finding a Component

A catalog can contain a large number of cataloged components. You may have problems quickly finding a component by browsing the catalog tree. An incremental search is provided to easily find a component or a module within the project catalog.

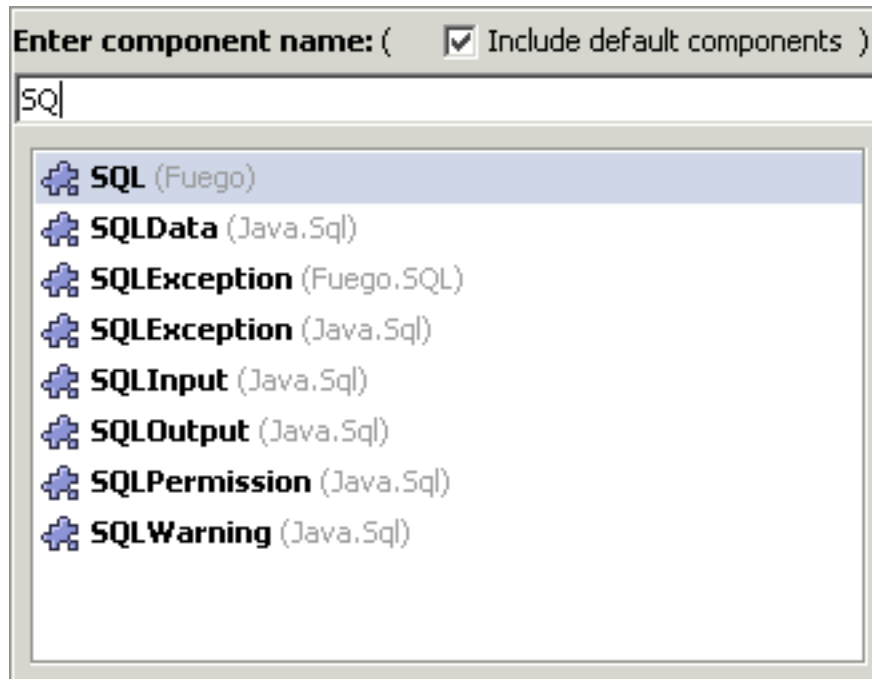
To find a component or module,

- Right-click on any Catalog category, select the option **Find Component**, or
- Use the short cut **Ctrl+Shift+C**. The search dialog opens.

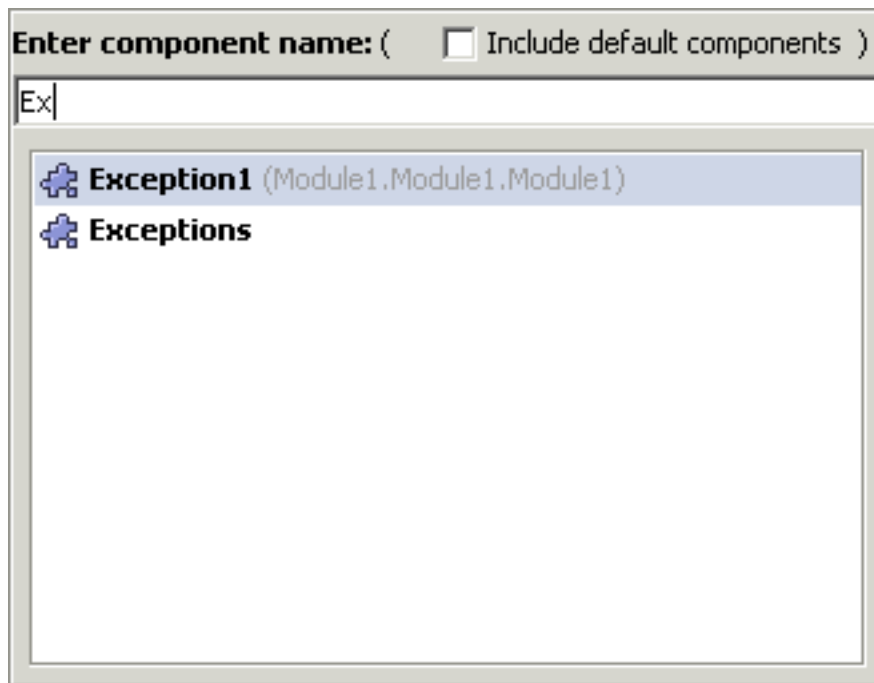


Enter component name: (☐ Include default components)

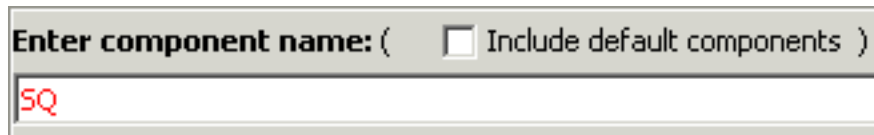
- If you want to include the default components, **java** and **Fuego** modules, click the check box "Include default components."



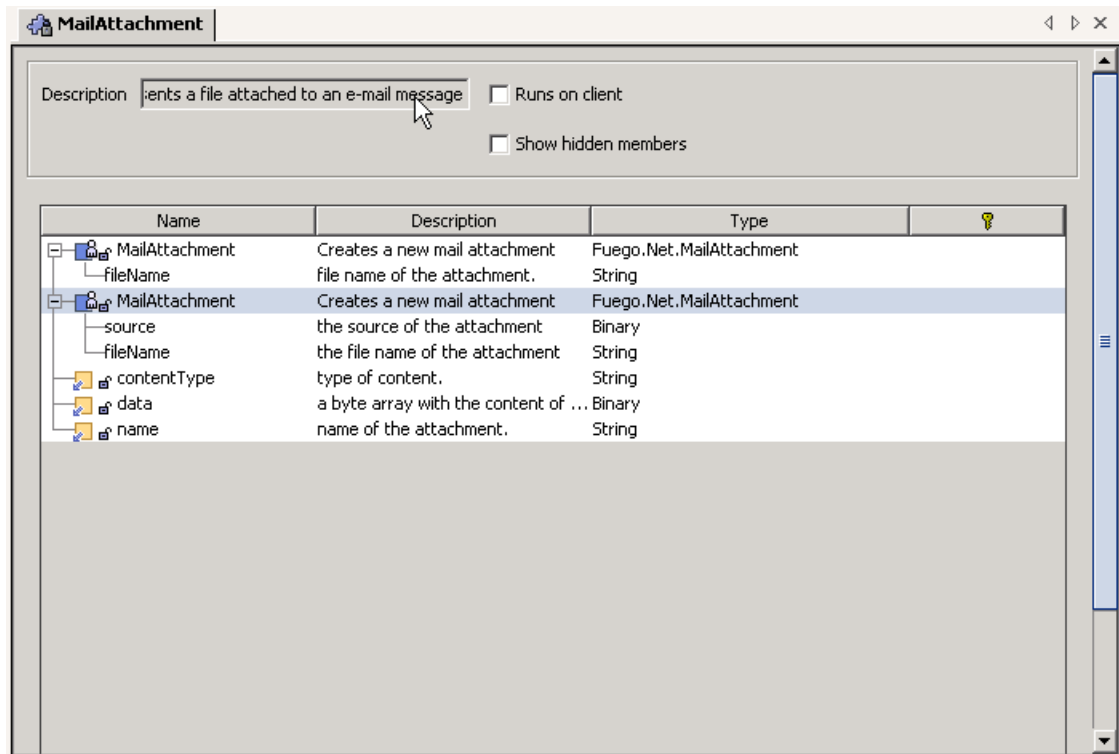
- Type the name of the component you are looking for into the Catalog,
- The search is done incrementally. A list of matching components and modules is displayed in the lower portion of the dialog while you are typing the component name.



- If the component name does not match any existing component in the catalog, the characters are displayed in red.



- Once you have found the module or component, double-click on it. Its properties panel is displayed in the right panel.



Import a Catalog from FuegoBPM 4.5

If you want to migrate a catalog from FuegoBPM 4.5,

1. Select the **Import** option in the main *File* menu and then the option **Catalog**.
2. Browse to the directory where the 4.5 component catalog is located, for example *K:\Documents and Settings\Administrator\.fuego4.5\componentCatalog*, and select the **Open** button. The **import** begins.


Once the **Import** action finishes, you will find in your catalog:

1. One Module per each one existing in the 4.5 catalog you've imported.



2. External resources for each connection required.
3. A generic and empty external resource with the name *JavaLibraries*, where you can catalog all the jars required in your project, or generate JCL type external resources separately.

Modules

Modules Overview

Modules are used to logically group components. Grouping enables you to quickly find a specific component. Modules are denoted by a box shaped icon  in the catalog tree.

Note

 If the  icon is displayed next to modules already created in the project catalog tree, this means that the module is read-only and you cannot modify it.

Modules can be nested in an unlimited quantity of levels provided that:

- Introspectors (Java, SQL, etc) are able to return a tree as the result of the introspection instead of a list of components.
- Groups for Fuego Objects are components defined inside the Fuego Object and they can be used from outside the group.
- Fuego Objects can implement the Inner classes because nested levels are allowed.
- It is possible to organize components from default cataloging sub-modules.

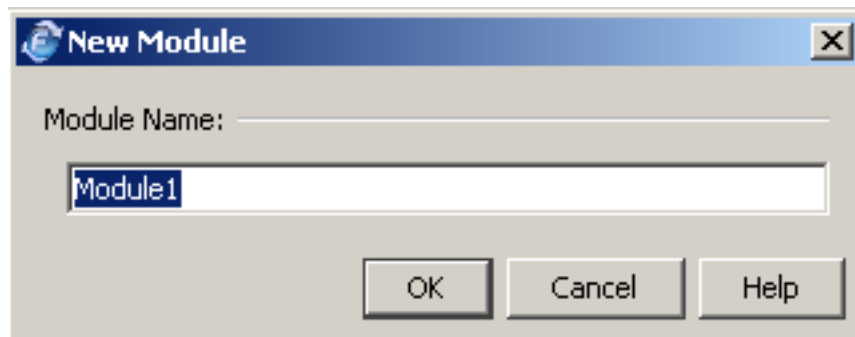
FuegoBPM is installed with two default modules, **java** and **Fuego**. These modules contain standard components, other modules,

components, and Java implementations that run many functions in FuegoBPM. Some of these standard components are also useful when you are designing your business processes. See *Implementing Business Objects using FuegoBlocks* for further information.

Adding a Module

To add a module:

1. Click **New Module** from the menu options displayed by right-clicking on the root category Catalog or on any user defined module. You are prompted to enter a module name.
2. Enter the name in the **Module Name** field, such as **JavaFiles** or **SQL**.

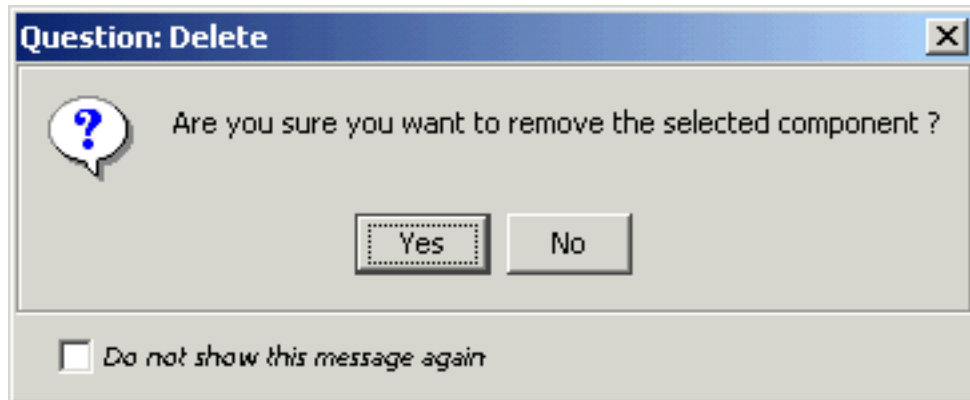


3. Click **Ok**.

Deleting a Module

To delete a module:

1. Right-click on the module name.
2. Select **Delete** from the shortcut menu. You are prompted to confirm the action.



3. Click **Yes**.

When a module is removed, it is deleted from the project catalog. The next time the project is published and deployed, an integration check is done. VCS - Version Control System is the solution in case the deleted module affects the project integrity and you want to restore the old one. If you are not using VCS, you should manually back up the catalog prior to deleting.

Finding a module

An incremental search is provided to easily search for a component within the catalog. For further information see Finding Modules and Components section.

See Also

Implementing Business Objects using FuegoBlocks

Finding Modules and Components section

VCS - Version Control System

Components

Components Overview

Components are the elements that work with each other to create a

system. Generally, any system that exposes its Application Program Interface (API) can be introspected into the Project Catalog, as long as it exposes the API through one of the following technologies:

- Automation
- Java
- XML
- Web Services
- Enterprise JavaBeans (EJB)
- Java Naming Directory Interface (JNDI)
- SQL
- CORBA

Once a component has been added to the catalog, it can be integrated into the project processes, thus enabling you to utilize existing applications already in use in your company structure. For example, you can introspect components of an existing database into the Project Catalog and then use them in the process method of a Global activity to give users the ability to query or modify information contained in the database.

There are two types of components utilized by Fuego products, Fuego blocks or Java components and user cataloged components. Fuego blocks and Java components are already defined and included with FuegoBPM Studio. Components cataloged by the user are stored in user created modules in the Project Catalog.

Fuego Objects can be used as containers to hold several unrelated components. In this way, you can create your own specialized business service by combining several different types of components into one Fuego Object.

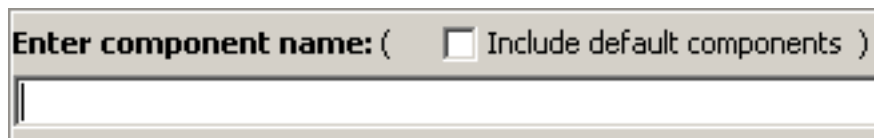
Cataloging Components

A set of wizards is provided to catalog components in an easy, guided way. Each wizard is explained in its corresponding component type section.

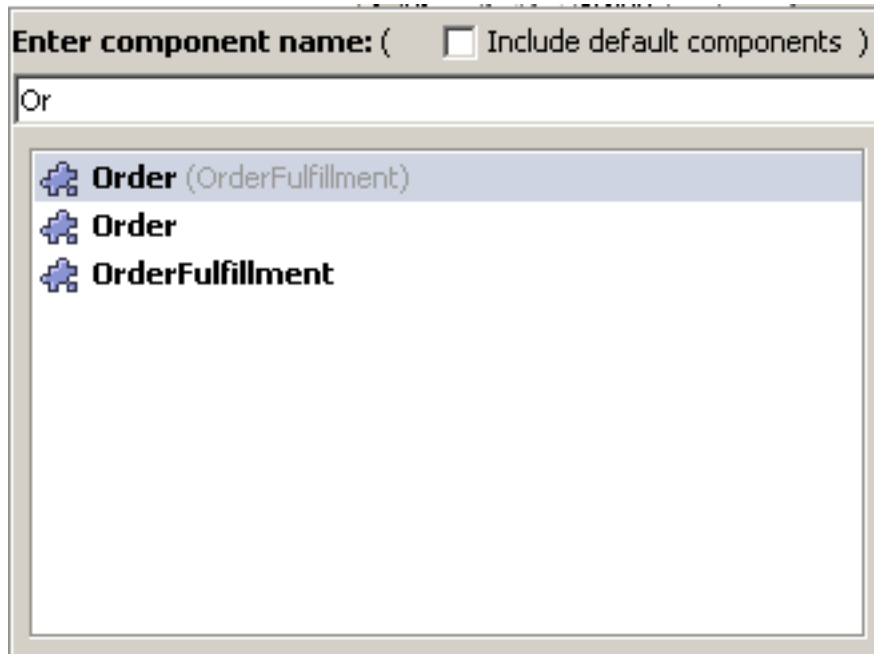
Opening a Component

Choose one of the following methods to open a component:



- Double-click on the name of the component in the catalog section of the Project flap,
- Right-click on the name of the component and select **Open** from the menu,
- From the **File** menu of the FuegoBPM Studio, select **Open** and then **Component**.
- Type the name of the component you want to open. Incrementally, the components that match the name that you are typing are displayed. Click the check box *Include default components* if you want to open one of the components provided by FuegoBPM Studio.



The image shows a dialog box with a title bar. Inside, there is a label "Enter component name: (" followed by a small square checkbox and the text "Include default components)". Below this is a large, empty rectangular text input field.



Note

 If the  icon is displayed next to the component in the project tree, this means that the component is read-only and you cannot modify it.

Viewing or Editing a Component

You can edit the methods and attributes that you selected to add them to a component.

To edit a component:

1. In the left panel of the Component Manager, drill down to the component by clicking on the symbol to the left of the module name then next to the implementation where the component is located.
2. Double-click the component name. The component folder is displayed in the right panel.
3. Select the check box in the **Visible** column next to the methods

you want to include. If you want to remove any methods, disable the check box in the **Visible** column next to the appropriate method.

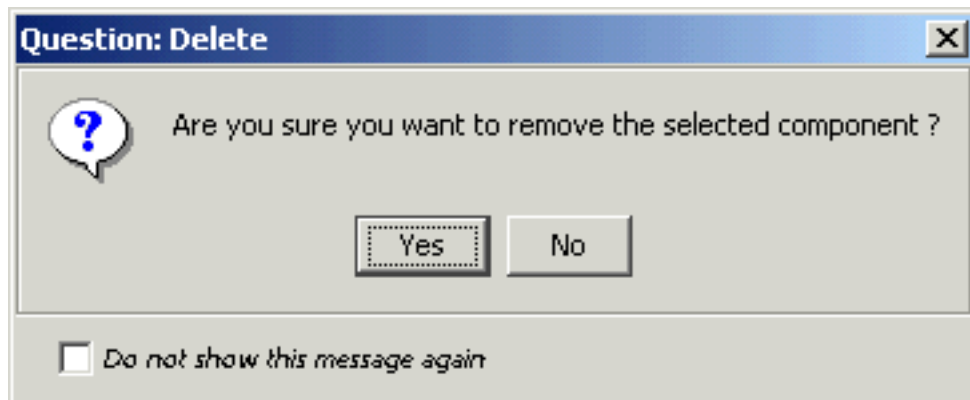
4. To save the changes locally, click the **Save** button on the toolbar.

Deleting a Component

When you do not need a component any longer, you can delete it from the catalog.

To delete a component from the catalog:

1. Right-click the component that you want to delete.
2. Select **Delete** from the shortcut menu.
3. A dialog box appears informing you that the object will be permanently deleted from the catalog.
4. Click **Yes**. This action removes the component from the Project Catalog.



5. If the catalog has been synchronized with the VCS repository, the component is removed from the VCS. You will not be able to

recover the component unless you have a VCS tag that contains it. For further information, see VCS - Version Control System.

Showing/Hiding components in the project catalog tree

Components that have been introspected into the Project catalog can be hidden. This is very useful when introspections have a large quantity of components that are required to be catalogued but are not needed to be seen. For example, the list of complement components when you catalog a COM component.

To show/hide a component right- click on the component and select the available option **Hide** or **Show**. The component remains in the catalog tree but grayed out. To see or not the hidden components, go to the **View** menu and select/unselect the **Show hidden components** preference option.

Documenting the Catalog

FuegoBPM Studio provides the ability to create, modify, and delete documentation for each component. Documentation consists of component comments (defined by the designer at component creation), methods, attributes, etc. (as with JavaDoc.)

For detailed information, please refer to Documenting a Project.

Finding Components

An incremental search is provided to easily search for a component within the catalog. For further information, see Finding Components section.

Implementing Business Objects Using Fuego Objects

A Fuego Object is the representation of custom Business Objects and Services. It can be used as a meta-catalog library, as a structure that makes data easier to manage, or as a combination of both. A Fuego Object is a component that contains attributes, methods, and presentations. A Fuego Object can only contain behavior, but can be used as a way to input information that would be persisted in another type of data container, such as a database or an XML file. Fuego Objects represent an easy way to manage and reuse complex data or data that is too time-consuming to code.

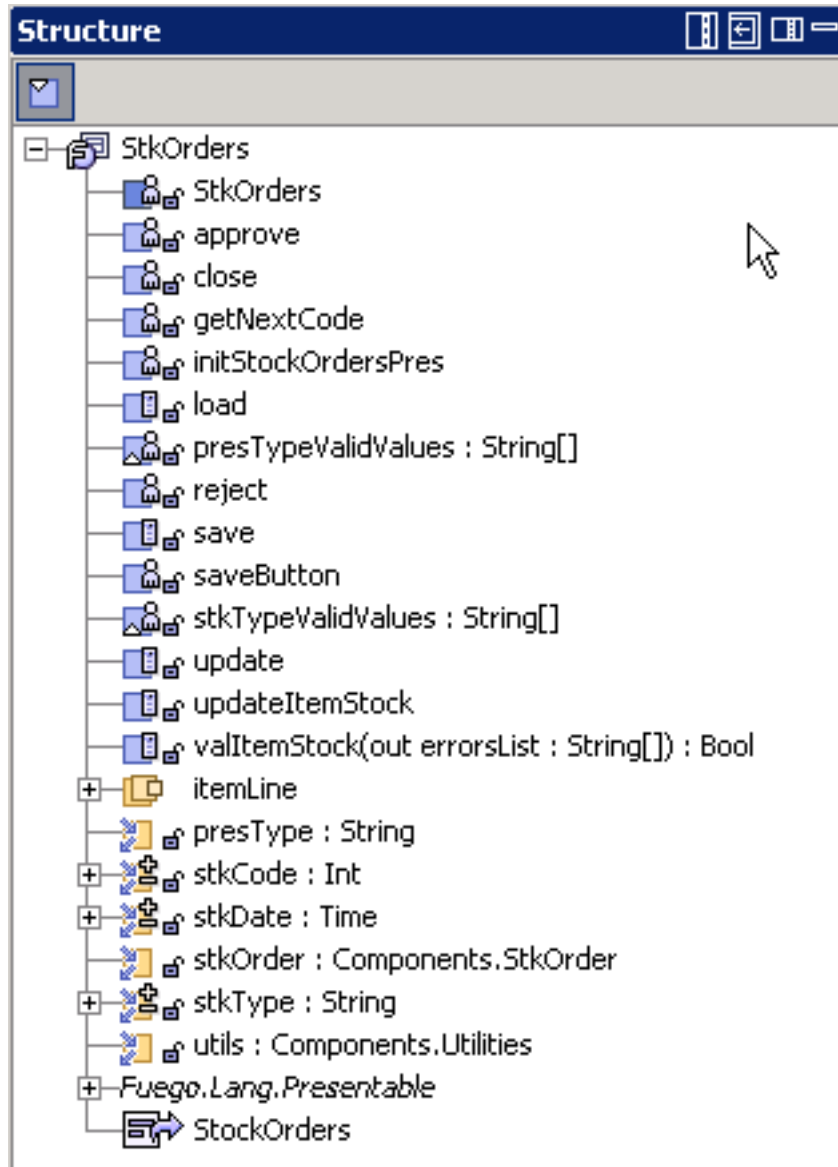
Knowledge Requirements: To understand Fuego Objects, knowledge about Object Oriented Design and programming concepts is required. In order to understand the Fuego Object examples given, you may want to first read the Method Guide.

A Quick Look at a Simple Fuego Object

Fuego Objects consist of the following:

- Attributes : Variables that define the Fuego Object.
- Methods : Pieces of code that can be called to perform some function for the Fuego Object.
- Groups : Several attributes combined together to function as a group.
- Presentations : Attribute information displayed in different ways for different sets of end users.

A simple Fuego Object is shown in the following picture. It depicts a simple Stock Order business object.



Attributes are the data that make up the Fuego Object and that describe it. Their values are set to the information associated to a particular instance. This means that the content of the attributes in the StockOrder Fuego Object shown above will most likely be different than the contents of the same attributes for another instance of the StockOrder Fuego Object. The attribute *stockOrderNumber* of this StockOrder instance might be 120, while another instance's *stockOrderNumber* could be 121.

Methods are used to manipulate the values of the attributes that

make up the Fuego Object. Methods can access, modify and retrieve the values of the attributes in the Fuego Object or they can use those attributes to calculate values based on the attributes.

In the Fuego Object shown, many of the methods were created automatically based on the choices made when creating the different attributes. An example of an automatically generated method is the *stkOrderTypeValidValues* method. This method is created by setting a list of valid values to the *stkOrderType* attribute when defining it. This method returns an array containing all the values allowed for that attribute. This array is used by presentations to display a combo box with the possible values of an attribute.

Every Fuego Object can (but does not have to) contain one or more presentations. Each of the fields on the presentation are tied to one of the attributes. Presentations provide a simple way for end users to view or to input the attributes of the Fuego Object.

When a Fuego Object is created, it is only a data container or non-presentable Fuego Object until a presentation is added to it; then it becomes presentable. See Presentable and Non-Presentable Fuego Objects for further information.

Presentations have special field types associated to each Fuego Object attribute. Each of the fields shown on a Presentation is bound to the attributes contained in the Fuego Object. These attributes have a graphical component associated according to their type. Refer to Fuego Object Presentations to learn the relation between attribute type and the graphical component used to display it in the presentation.

Benefits

Using Fuego Objects provides a number of significant benefits for businesses. Some of the benefits are as follows:

- Fuego Objects are reusable and portable because of they are

stored in XML format. XML (Extensible Markup Language) is an standard language defined by the W3C.

- Fuego Objects are based on the Xform standard and therefore, they can be edited with any form creation software.
- Once built, Fuego Objects can be reused across multiple processes, projects, Servers.
- The time invested in process development and maintenance is drastically reduced because of Fuego Object reusability and the wizards provided by FuegoBPM Studio with their friendly graphical interface.
- Fuego Object methods are coded in a simple but very powerful language (Fuego Language). Fuego Language is easily learned by people who have basic programming knowledge, and at the same time it enables more experienced programmers to perform more complicated tasks.
- Fuego Object method bodies can be completely changed, and as long as their interface is not changed, the calls in your process do not need to be changed. This allows you to change the behavior of a Fuego Object across your processes and projects by simply modifying just a few lines. Not only does this reduce the time cost invested in maintenance consequently reducing the whole project time cycle, but it gives you a greater capacity and flexibility in

requirement change control.

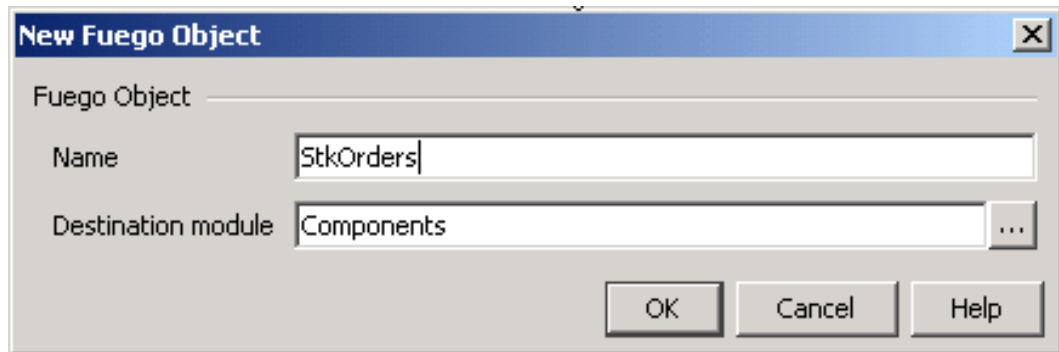
- Components implemented in different technologies can be integrated by wrapping them in a single Fuego Object. The functionally provided by multiple technologies and applications can be combined in order to create a completely new business service, which is not limited to a specific technology. A single Fuego Object could retrieve a list of customer names from a customer relationship management (CRM) application, order information from an enterprise resource planner (ERP) application, and populate details from a corporate database. Therefore, integrating with a small amount of time and effort, all the disparate technologies that are usually present in a business environment.
- Fuego Objects can be used to customize the information display to different users based on their role, authorization, and/or training.
- Coding time and effort can be saved by using a Fuego Object instead of using languages such as Java or C, without losing functionality or even increasing it, and at the same time making your components easy to scale.

Create a Fuego Object

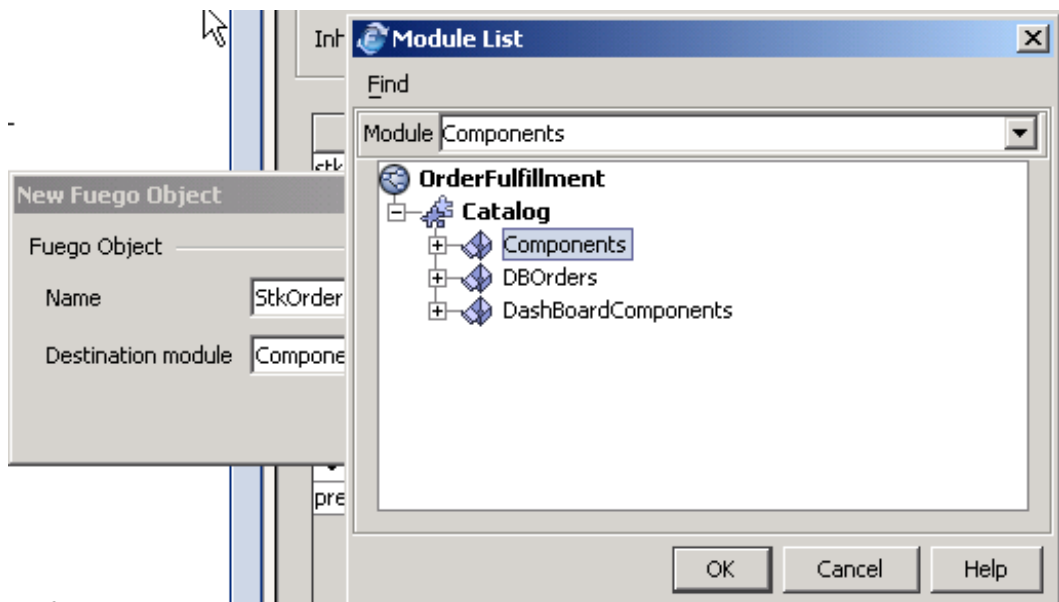
Creating a Fuego Object is the first step in creating a unique business service to integrate with your business process design.

To create a Fuego Object:

1. Right-click on the module in the left panel of the Project Catalog. Select **New Fuego Object** from the shortcut menu.
2. Type the name for the Fuego Object in the dialog box displayed. The *module* field automatically populates with the name of the catalog module from which you began the Fuego Object creation (the module you right-clicked on in step 1).





3. If you want to locate the new Fuego Object in a different catalog module, browse the Project Catalog by clicking the browse button. Select the destination module and click **OK** to proceed.



4. Add as many attributes, groups, methods, and presentations as needed in order to complete your Fuego Object. After doing that, you will be able to call your Fuego Object from a Business

Process Method task in a process design. See Fuego Objects Examples and Designing using Fuego Objects.

Note

 If the  icon is displayed next to a Fuego Object in the project catalog tree, this means that the Fuego Object is read-only and you cannot modify it.

Working with a Fuego Object

The basic structure of a Fuego Object when it is created consists of a constructor method named the same as the Fuego Object. When you create it, the Fuego Object panel opens in the center of the FuegoBPM Studio Workspace. Initially, it is empty since it only shows the Fuego Object attributes you define for it. This panel consists of five columns that show the name and other properties of the attributes and basic information about the Fuego Object Data Structure.

The columns displayed are: *Primary key* (key icon), *Name*, *Type*, *Not null* and *Default value*. This panel allows you to reposition the attributes within the Fuego Object structure by selecting the attribute and then clicking the **Move Up** or **Move down** buttons. It also lets you open an attribute profile by selecting an attribute and then clicking the **Open** button. All attribute properties, are editable from the Fuego Object attributes panel.

Name	Key	Virtual	Type	Not null	Default value
stkOrder		<input type="checkbox"/>	Components.StkOrder	<input checked="" type="checkbox"/>	
utils		<input type="checkbox"/>	Components.Utilities	<input type="checkbox"/>	
stkType	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	String		
stkCode	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Int		
stkDate	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Time	<input checked="" type="checkbox"/>	Jan 28, 2004
itemLine			Components.StkOrders.ItemLine[]		
• currentRow	<input type="checkbox"/>	<input type="checkbox"/>	Int	<input type="checkbox"/>	
• stkOrdIt		<input type="checkbox"/>	Components.StkOrdIt	<input checked="" type="checkbox"/>	
• item	<input type="checkbox"/>	<input type="checkbox"/>	String	<input type="checkbox"/>	
• qty	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Int		
presType	<input type="checkbox"/>	<input type="checkbox"/>	String	<input type="checkbox"/>	

Naming Conventions

Variables and Attributes

Attributes follow the same naming conventions as variables. The following are conventions to be considered when creating variables or Fuego Object attributes.

If names contain more than one word, convention suggests capitalizing the first letter of the second word for readability.

When naming variables, you should use a descriptive name. You do not need to use one-word names. In fact, short phrases can be helpful at times. For example, the following variables indicate the value the variable contains and use more than one word:

- firstName
- paymentType

- shipmentDueDate

If variable names contain more than one word, convention suggests capitalizing the first letter of the second for readability.

Non-alphanumeric characters are not allowed in the name with the exception of the "_" underscore, which may be used anywhere in the middle of the name. For example, in a name that contains two parts, such as "first name", you may want to insert an underscore between the two words to separate them. For example, *first_name*.

The Method reserved words *display*, *while*, *do*, *for*, *for each*, *elseif*, *else*, *input*, *is*, *on*, *throw*, *if*, *exit*, *using* and *end* cannot be used as variable names.

Tip



If you plan to include presentations for your Fuego Object, use an underscore to separate variable parts (if the variable has more than one word). When you generate the presentation, use the template and click the **Preferences** button. Select the *Separate names when an underscore option is found* on the **Label** tab. This quickly creates labels for your presentation that do not have to be edited.

Methods

Method naming conventions dictate the use of a verb at the beginning of the method name to indicate the purpose of the method. Some common method names listed below. Notice that method names also follow variable naming conventions by separating parts with capital letters.

- calculateTotal()
- getItemPrice()
- retrieveRecords()

Additionally, hidden methods are created for each attribute added to the Fuego Object. These methods are called `get{AttributeName}` and `set{AttributeName}`. For example, for the attribute `first_name`, two hidden methods are created: `getfirst_name` and `setfirst_name`. This means that you should not create any methods with these same names. If you do, you will receive a Warning error message.

A restriction to the method name is that **ValidValues** at the end of the method name is not allowed. Automatic methods are created with this name for those attributes that have a definition of valid values.

Caution:

FuegoBPM Studio is not limited by character set. You can use any alphanumeric character in your Fuego Object attribute and method names, even those with accents. However, be aware that if you are using a relational database system, such as Oracle as your directory service, there are some limitations. A character set is installed with the relational database. The database administrator must make sure that he or she selects the appropriate character set for the installation. If you try to commit a catalog with Fuego Object attributes containing characters that are not included in the directory service database, the commit will fail and generate errors. The attributes will have to be deleted and re-added with characters that are supported in the directory service database.

Exceptions

FuegoBPM convention suggests that you name all exceptions with the word "Exception" as the last part of the exception name. For example, the following is a partial list of valid exception names:

- CreditException
- DateException
- OrderAmountException

Fuego Objects Attributes

Attributes are the characteristics that define a Fuego Object. For example, if your Fuego Object is an inanimate object, such as a car, the attributes are thing that define the car. Some car attributes are as follows:

- manufacturer
- model
- color
- engine size
- tire size

A real life example of a Fuego Object is a Customer Invoice. The attributes are the properties that make up a Customer Invoice such as:

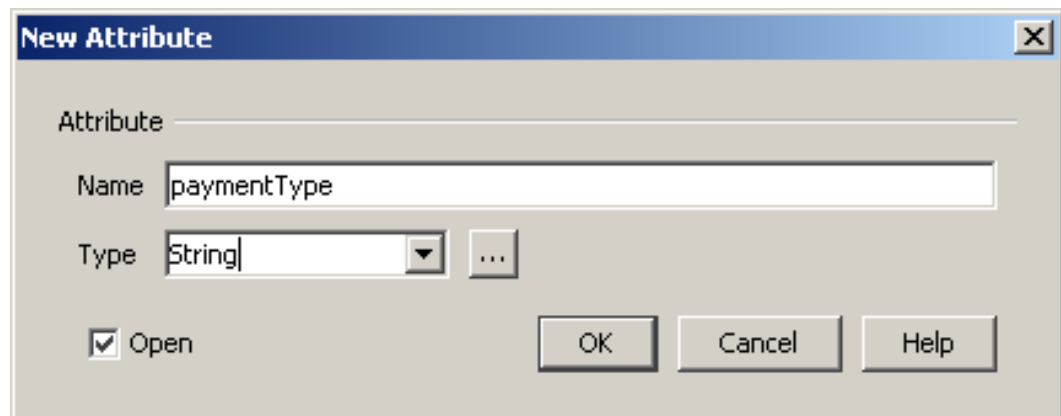
- customer name
- address
- phone number
- item purchased
- quantity
- price
- total

Add an Attribute

Attributes are characteristics that define the states of a Fuego Object.

To add an attribute:

1. Right-click on the Fuego Object and select the **New Attribute** option or use the shortcut **Ctrl+Alt+V**. A blank attribute profile opens in the right panel of the workspace.
2. Enter the *Attribute name* in the dialog that appears. Select the *Type* or browse the catalog of components by clicking the browse button and select the *Show hidden components* to visualize those components set as hidden in the catalog. Click **OK**. The attribute is created under the Fuego Object structure. Its profile is automatically opened when the *Open* check box is selected.




Editing an Attribute

To edit an attribute:


1. Right-click on the attribute you want to edit and select **Open** from the shortcut menu.
2. Double-click on the attribute in the structure panel.
3. From the Fuego Object panel, double-click the attribute you want

to edit, or select the attribute in the Fuego Object panel and click **Open**.

4. When you have finished completing the fields for the attribute, click **Save** on the FuegoBPM Studio main toolbar.

To close an attribute profile, click the  close view button at the top-right corner of the attribute's profile panel.

Note

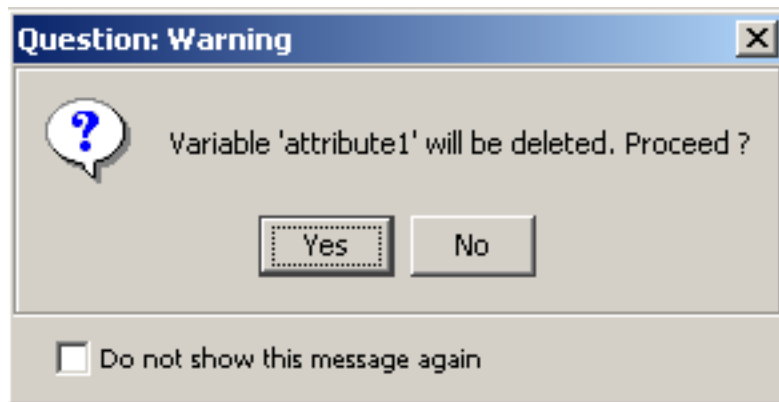
 If you change an attribute type, you will not see the change immediately reflected in the Project and Structure panels. You will see the updated change after saving or opening another Fuego Object element (attribute, method).

To Rename an attribute, select the *Rename* option from the shortcut menu that is opened by right-clicking on the attribute--either on the structure flap or in the Fuego Object editor panel.

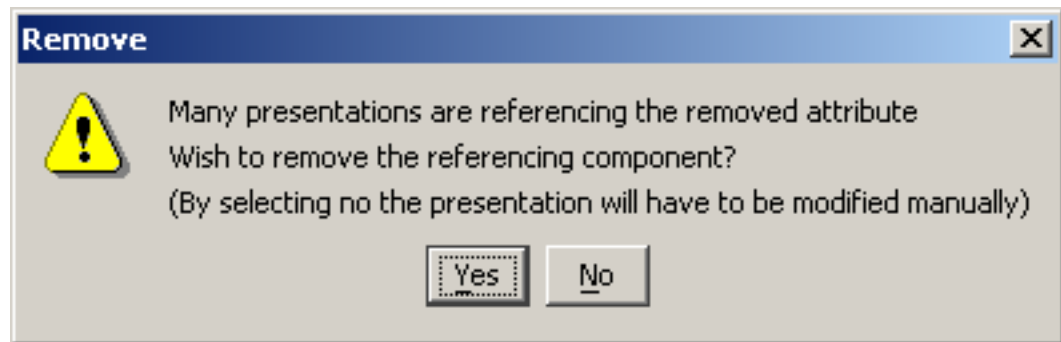
Deleting an Attribute

To delete an attribute:

1. Right-click on the attribute you want to delete and select the **Delete** option from the shortcut menu.



2. If the attribute you are trying to delete is referenced in one or more presentations, a warning message is displayed alerting you about this situation.



3. If you select the **Yes** button, the reference to this attribute in any presentation is deleted. If you select **No**, the presentation will have to be manually updated (remove the reference to the attribute).

Fuego Object Attributes' Panel

Fuego Object attributes' panel provides a simple and intuitive way to define an attribute. In addition, every field has a mouse-over tooltip available.

A screenshot of the 'Fuego Object Attributes' panel. It is a light gray form with several sections. The first section is labeled 'Type' and contains a dropdown menu with 'String' selected, a button with three dots, and a checkbox labeled 'Maximum length' next to a numeric input field with '0'. The second section is labeled 'Storage & Constraints' and contains three checkboxes: 'Virtual' (unchecked), 'Primary Key' (checked), and 'Not null' (unchecked). The third section is labeled 'Default value' and contains a radio button (selected) next to a text input field, and another radio button labeled 'Null'.

Depending on the attribute's type, precision, length, or decimal digits must be defined. The panel automatically displays the required attribute fields according to the type. Attribute fields enable or disable according to the choices you make.

The screenshot shows a configuration panel for an attribute. It is divided into three sections: 'Type', 'Storage & Constraints', and 'Default value'. In the 'Type' section, a dropdown menu is set to 'Bool' with a small '...' button to its right. The 'Storage & Constraints' section contains three checkboxes: 'Virtual' (checked), 'Primary Key' (checked), and 'Not null' (unchecked). The 'Default value' section has a radio button (unchecked) to the left of a dropdown menu set to 'True', and another radio button (checked) to the right of the word 'Null'.

When a field is mandatory and you have not checked it or added a value to it, it displays with a red border and the tooltip displays in red as well.

Defining an Attribute

Fuego Object attributes can be *real* or *virtual*. Defining an attribute as *real*, means that the attribute has a real value associated container. When an attribute is defined as *virtual*, it means that it has no real value associated container, it is only defined by its *read* and *write* access methods.

Read and Write Access

If an attribute has no read access, this means that its value is not accessible. On the contrary, if an attribute has no write access, its value cannot be changed.

All *real* attributes have both read and write access methods set by default. When an attribute is added to a Fuego Object, it is created as *Real* by default. When access methods are added to *real* type

attributes, they override this default. When access methods are removed, they *reset* to the default. These changes never modify the *read/write* access of the attribute.

An attribute can be redefined as *real* or *virtual* by selecting or deselecting the *Virtual* check box available for the attribute in the Fuego Object editor panel.

Virtual attributes do not have access methods. They must be added by the user and it is required to have at least one. If no access method is defined, the Fuego Object compilation fails.

When access methods from virtual type attributes are added, they add the *read* and/or *write* access to the virtual attribute. When the access methods are removed, they remove these accesses from the attribute.

Read access has a return type identical to the attribute's return type. Write access receives an argument called *value*, which is of the same type and has no return type.

Virtual attributes - how to recalculate their values

FuegoBPM Studio debugger and FuegoBPM Work Portal implementations differ, and that affects on how the virtual attribute value is recalculated. The debugger uses swing implementation which is a one tier architecture. On the other hand, the Work Portal, uses HTML which is a three tiers MVC (Model/View/Controller).

Swing implementation calculates virtual attributes automatically, but HTML needs an action (*onChange* or *onAction* methods) to recalculate them.

So, to make the virtual attribute be recalculated an *onChange method* associated to every attribute of you calculate expression.

When to use a Virtual or a Real Attribute

Whether you use virtual or real attributes depends on your needs, but basically, the virtual attribute helps you optimize your Fuego

Object and avoid using a container for a field that does not require one. Virtual attributes can always be calculated and they do not require storage for their values.

Examples

Redefining the read access and write access to a virtual attribute

Suppose that your object has a time field *orderDeadline* that represents the order deadline and you need to change it according to the remaining days that the user enters. To do so, you can define a virtual attribute *remainingDays* that is a *view* of the real *orderDeadline* and overwrite its *read* and *write* access so that you can fulfill your needs.

The overwritten methods are:

read access:

```
return daysSince(orderDeadline, 'now')
```

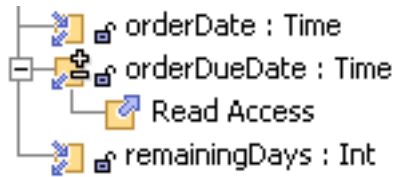
write access:

```
orderDeadline = 'now'.addDays(value)
```

Redefining the read access to a virtual attribute

The example above can be implemented in a different way. Suppose that you have *orderDate* and *orderDueDate* and you want to recalculate the *orderDueDate* by letting the user enter the *remainingDays* of the order. In this case, you can define a virtual attribute overriding its *read access method* that would retrieve the *orderDate* plus the *remainingDays* entered by the user.

The attributes and Fuego Object panel appear as shown in the following images:



Name		Virtual	Type	Not null	Default value
orderDate	<input type="checkbox"/>	<input type="checkbox"/>	Time	<input checked="" type="checkbox"/>	2003-12-04 00:00...
remainingDays	<input type="checkbox"/>	<input type="checkbox"/>	Int	<input checked="" type="checkbox"/>	0
orderDueDate	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Time	<input type="checkbox"/>	2003-12-04 00:00...

and the overridden *read access method* of *remainingDays* is:

```
return orderDate.now().addDays(remainingDays)
```

You will be able to reset the *orderDueDate* attribute by assigning it the *remainingDays* attribute. As its *read access* is overridden, it adds its value to the *orderDate* attribute. That would be the final value assigned to *orderDueDate*.

Redefining the write access to a real attribute

You may override the write access to a real attribute as shown in the following case. The Fuego Object has a String attribute *name*. You want to eliminate any blank spaces from it and give it a default name in the case that it is empty.

The overridden *write access method* of the *name* field is:

```
if value is null or value = "" then
    name = "DefaultName"
```

```
else
  name = trim(value)
end
```

Attributes Types

Available types to define an attribute are:

- Bool: True or false.
- Int: Integer number.
- Decimal: Decimal number value with defined precision.
- Real: Real number value.
- String: Alphanumeric characters.
- Time: A unit of time.
- Interval: An interval of time.
- Binary: Binary contained, for example, an image, a file, etc.
- Any: Represents any kind of type.

When defining the attribute type by clicking the browse button next to the *Type* field, you can select any cataloged component in the Project Catalog as the defined type for the attribute you are declaring. This gives you the ability to typify an attribute within a Fuego Object with any component defined in the Project Catalog, even another Fuego Object. If it the attribute is typed after a different Fuego Object, it is referred to as an Inner Fuego Object. Find an example in the *Inner Fuego Object* section below.

What do the fields mean?

The following table shows the required fields for defining an attribute.

Field	Description	Attribute type
Type	Domain type of data that the attribute contains.	Real & Virtual attributes.
Not null	The attribute may be assigned, or not, a null value.	Attribute must have write access.
Primary Key	Is the attribute part of the primary key of the Fuego Object?	Attribute must have read access.
Time Precision	Only for Time attributes, you can choose the following: Timestamp : The entire date and time, Date only : The date, Time only : The time.	Date attributes.
Absolute	The value of a time attribute which is set as absolute is stored in GMT-0.	
Maximum length	For String. The <i>Maximum length</i> field defines the maximum <i>number</i> of characters allowed for the attribute.	Real & Virtual attributes.
Default Value	For Int, Decimal, Real, String, Time and Interval types. Initial value that the attribute contains. A default	Attribute must have write access.

Field	Description	Attribute type
	value for a Bool type attribute is defined by selecting the <i>True</i> or <i>False</i> radio button.	
Require Expression	The <i>Require Expression</i> field allows you to enter a Boolean expression to be checked as a precondition to the assignment of a value to the attribute. It can be built either by a simple, one line expression that evaluates an expression or by calling a method. This method must return a boolean type.	Attribute must have write access.
Check Expression	<i>Check Expression</i> defines integrity validation of the attribute within the Fuego Object instance. It can be built either by a simple, one line expression that evaluates an expression or by calling a method. This method must return a boolean type.	Any attribute.
Valid Values	List of valid values that an attribute can be assigned. It can be built either by a hardcoded list of valid values that the attribute can	Any attribute

Field	Description	Attribute type
	contain or by a call to a method that returns a list (array) of valid values. This method must return an array of the same domain type as the attribute's type. The implementation allows you to show the actual value or its description. For further details, see section <i>Valid Values</i> on this page.	

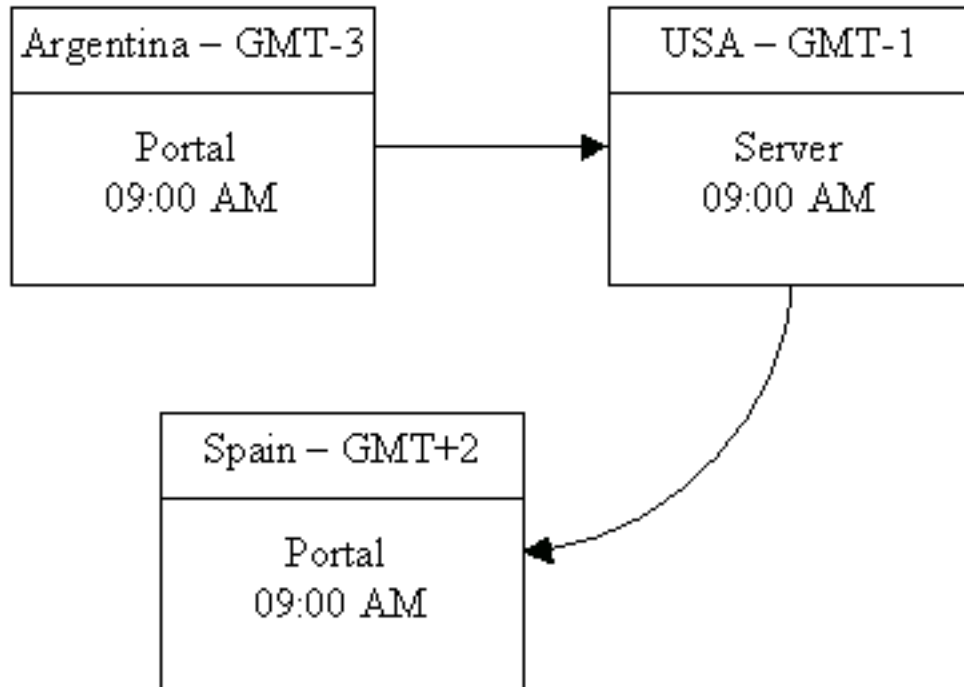
Absolute

If this property in a time attribute is set to true, the date is stored in timezone GMT-0. For example,

1. You input a time in a Work Portal with the timezone set to Argentina, GMT-3. The time input is "09:00 AM".
2. As the attribute has been defined as *absolute*, the time is store in the server, located in USA with GMT-1, without timezone or let's say GMT0.
3. If the same value is loaded in a user Work Portal with the timezone in Spain-Europe/Madrid, GMT+2, the value is still shown as "09:00 AM".



Absolute = true

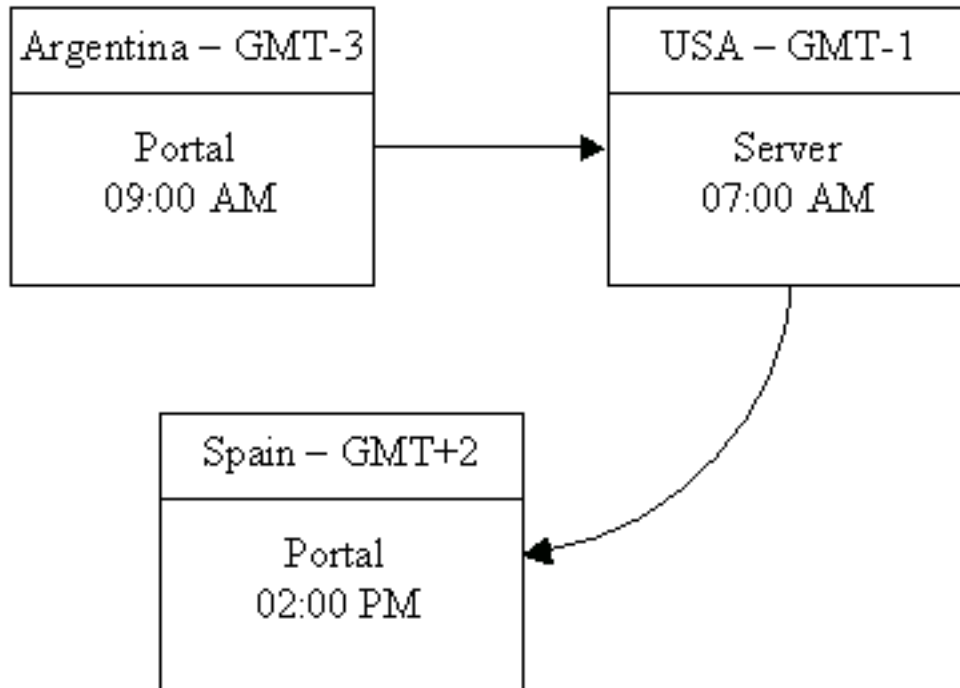


If this property in a time attribute is set to false, the date is stored with the server timezone.

1. you input a time in a Work Portal with the timezone set to Argentina, GMT-3. The time input is "09:00 AM".
2. As the attribute has been defined as *not absolute*, the time is stored using the server timezone, USA GMT-1, 07:00 AM
3. If the same value is loaded in a user Work Portal with the timezone in Spain-Europe/Madrid, GMT+2, the value is shown as "02:00 PM".



Absolute = false



Not Null

This property indicates whether the attribute can be assigned the null value. If the attribute is set as *not null* when it is assigned the null value, it will fail at runtime.

In addition, an attribute defined as *Not null* must always have a *default value*.

String attributes are an exception among attribute types because they can be filled with an empty value "" as default value when you set it as *Not Null* (empty value is a valid value of String attributes). Every other type cannot be set with an empty value.

Read & Write access special considerations

Not null flag is used to validate that the value trying to be set is not null. Thus, the attribute must have write access to be set as not null.

Fuego Object Primary Key

A Primary key can be defined to the Fuego Object to be able to be persistent. The primary key can be formed by one or more attributes of the Fuego Object.

Read & Write access special considerations

Since a primary key is used to store the object, the attribute must be read accessible.

What does the *primary key* provide?

Automatic conversion to String

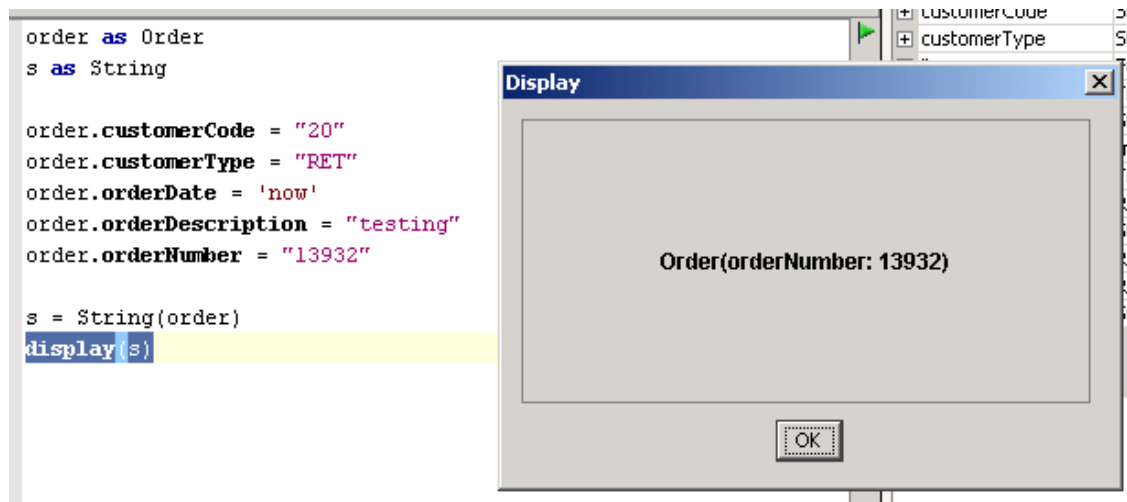
If you have a Fuego Object defined with a primary key, you can automatically convert it to String. In the following example, where the primary key of the *Order* Fuego Object is the attribute *orderNum*, a method is shown converting to String:

```
order as Order
s as String

order.orderNumber = "13932"
order.customerCode = "20"
order.customerType = "RET"
order.orderDate = 'now'
order.orderDescription = "testing"

s = String(order)
display(s)
```

The conversion to String is built with the Fuego Object name and the primary key composition, attribute name and value (Fuego Object Name(attr1:value, attr2:value, etc.)



Comparison by equal

A Fuego Object can be comparable by equal if it has a *primary key*.

If a Fuego Object component has a **primary key** defined and it is **not comparable** then, it will be only comparable by equal. For example, suppose that there is a Method using the Fuego Object *Order*:

```

//variables definition
Order1 as Order
Order2 as Order

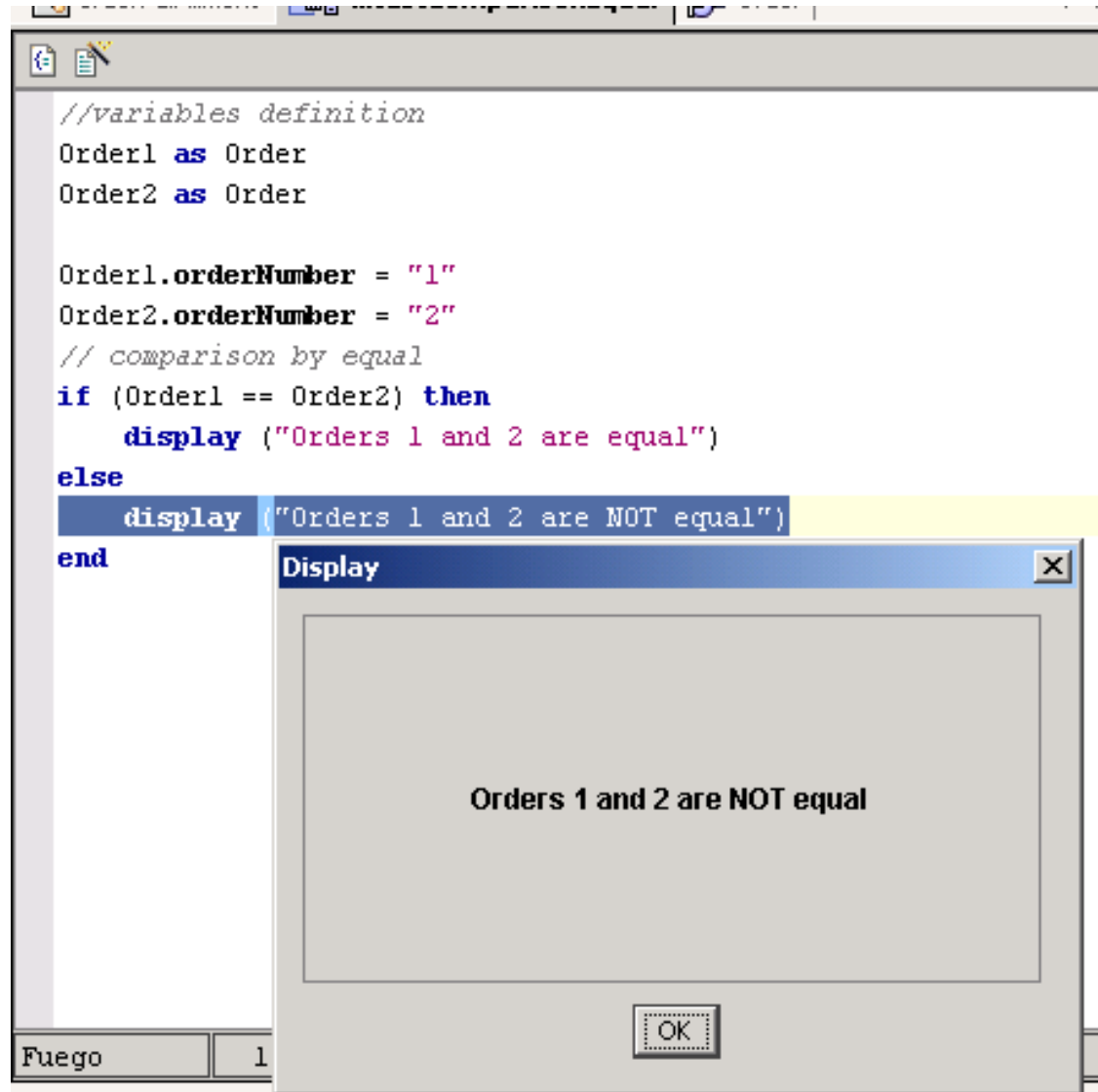
Order1.orderNumber = "1"
Order2.orderNumber = "2"

// comparison by equal
if (Order1 == Order2) then
    display ("Orders 1 and 2 are equal")
else
    display ("Orders 1 and 2 are NOT equal")
end

```

The display statement in the "then" section never appears but the *else* display appears, as the comparison is false, whether the Fuego

Object has a primary key defined or not. If the Fuego Object is not assigned any value, the primary key attributes have the *null* value and the "if" sentence checks true.



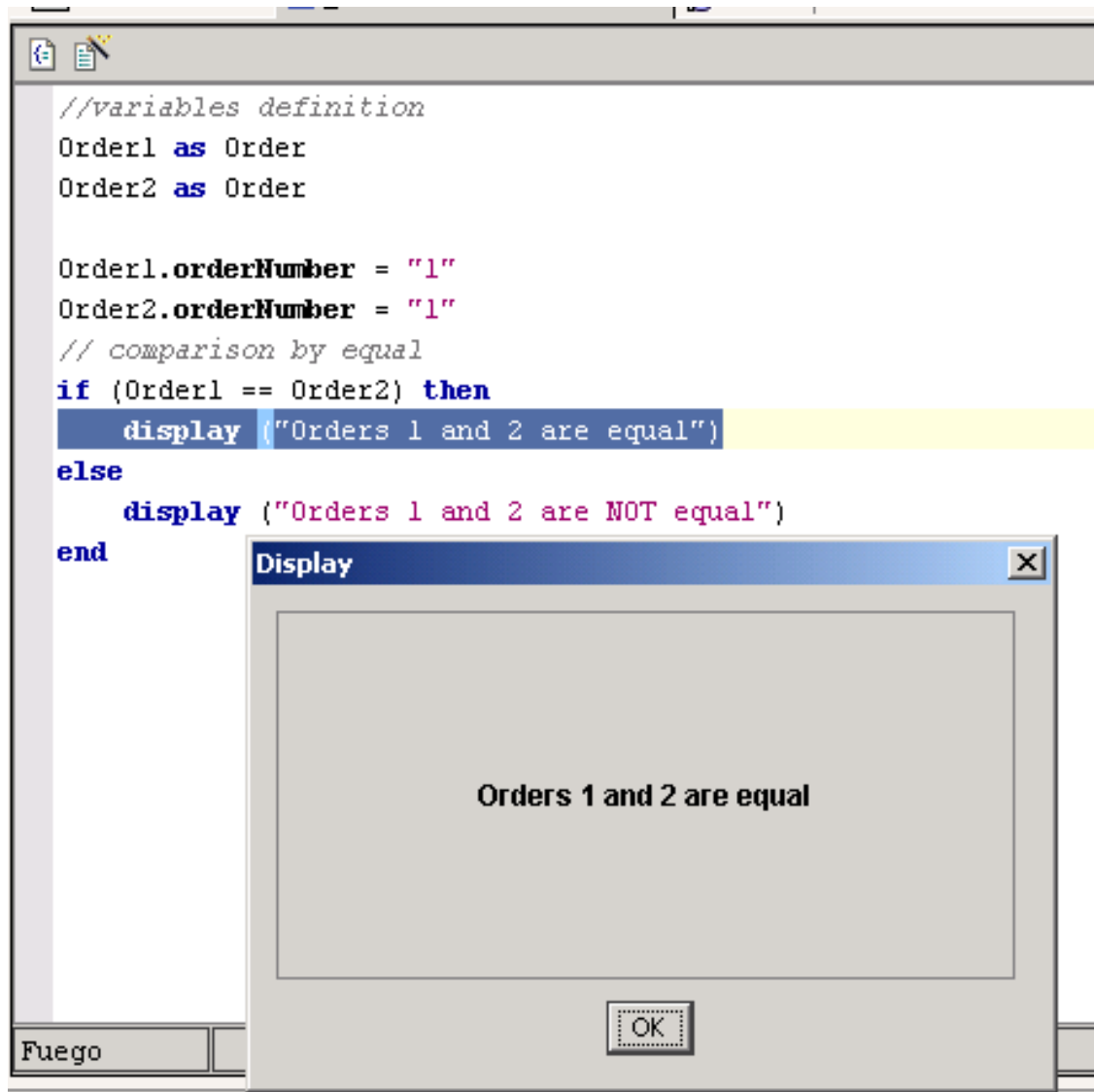
If the method is redefined as follows:

```
//variables definition
Order1 as Order
Order2 as Order

Order1.orderNumber = "1"
```

```
Order2.orderNumber = "1"  
// comparison by equal  
if (Order1 == Order2) then  
    display ("Orders 1 and 2 are equal")  
else  
    display ("Orders 1 and 2 are NOT equal")  
end
```

the *if* is true only when the Fuego Object has the primary key defined.



Primary key and a Comparable Fuego Object

Adding a primary key to a *comparable* Fuego Object provides comparison by *greater than* and *less than* in the example above is changed to:

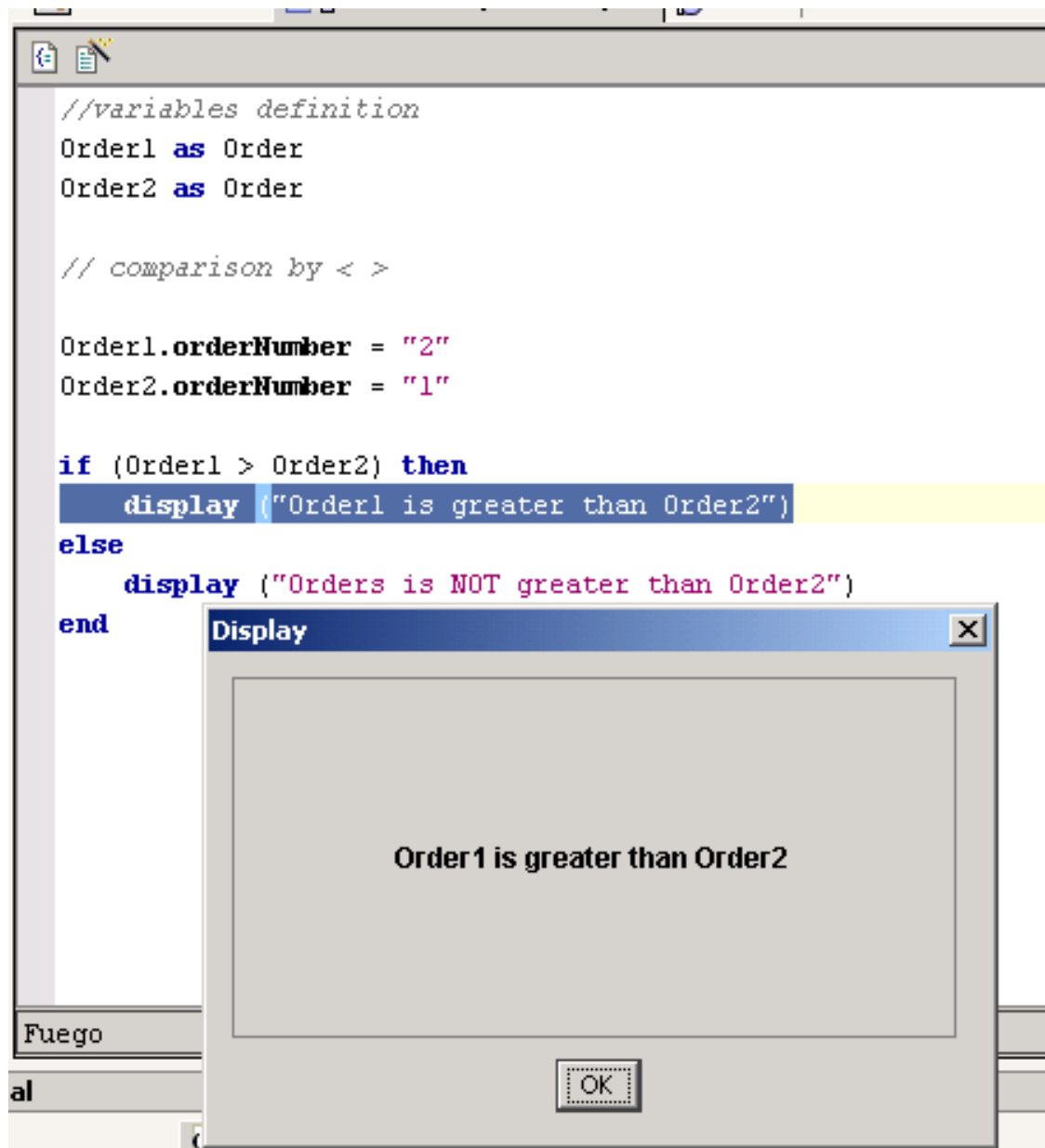
```
//variables definition
Order1 as Order
Order2 as Order

// comparison by < >

Order1.orderNumber = "2"
Order2.orderNumber = "1"

if (Order1 > Order2) then
    display ("Order1 is greater than Order2")
else
    display ("Orders is NOT greater than Order2")
end
```

If the Fuego Object **is not comparable**, then the *if* clause will not compile. If the primary key is built by more than one attribute, the comparison is done by position, first attributes, second attributes, and so on.



The *sort* function can be applied to the Fuego Object.

```
...
array[] = Order1
array[] = Order2

sort array
```

```
...
```

This method is valid only when the Fuego Object has primary key and is comparable. Any element added to the array after the *sort* will not be sorted within the array but will be appended.

If the definition of the array is created as follows:

```
...  
array as String [ordered Order]  
...
```

The array is always ordered; every element is added to it in the correct position by order.

Relationship with other attribute properties.

Primary Key and Type

Not all attribute types can be part of the primary key. The attribute must be of an atomic type. That is the reason why you will not be able to add to the primary key an attribute set as:

- *Any* type,
- *Binary* type,
- *Array* type, *Int*[], *Decimal*[], *String*[], etc.,
- any other component of the Project catalog, for example, an inner Fuego Object.

Primary Key and Not null

There are no restrictions about defining an attribute as primary key if it is null.

Require Expressions

The *Require Expression* field enables you to enter a Boolean expression to be checked as a precondition to the assignment of a value to the attribute. Require Expression has sense if the attribute has write access.

The Require expression is evaluated every time a value is intended to be assigned to the attribute, as it is a preconditional check.

The Require expression intention is to execute validations specifically related to the attribute, for example: validate value ranges.

For example, if the attribute *sex* has as required expression

```
sex = "M" or sex = "F"
```

when an invalid value is assigned to the attribute in a method, for example:

```
person as PersonObj  
person.sex = "V"
```

In debug mode the "ValidationException" is thrown.

From a presentation the following message is displayed when the expression does not check:

Violated constraint validation

sex : Required expression fail**Read & Write access special considerations**

As *Require expressions* are used to validate the arguments passed to a write access method before the access is called, they require the attribute to have write access. For example, if you have an *orderDate* attribute, you can validate the *orderDate* to be today by the following expression:

```
orderDate == 'now'
```

When the attribute defines both a *Require Expression* and a *Default Value*, the consistency check between the value and the expression is not done at design time but at runtime.

To add a validation expression:

1. Open the attribute you want to modify.
2. Enter a Boolean expression or invoke a bool returning method in the *Require Expression* tab.
3. Click the **Check** button on the toolbar.

Check Expressions

The *Check Expression* field allows you to enter a Boolean expression to check the integrity of the attribute in the object context.

This expression is only checked when a presentation is submitted.

Read & Write access special considerations

Read & write access methods are used to access the attributes.

Hence, only attributes with read access can have check expressions. For example, if you have *shipDate* and *orderDate* attributes, you can validate that the *shipDate* is not later than the *orderDate* by the following expression:

```
this.shipDate > this.orderDate
```

To add a check expression:

1. Open the attribute you want to modify.
2. Enter a Boolean expression or invoke a bool returning method in the *Check Expression* tab.
3. Click the **Check** button on the toolbar.

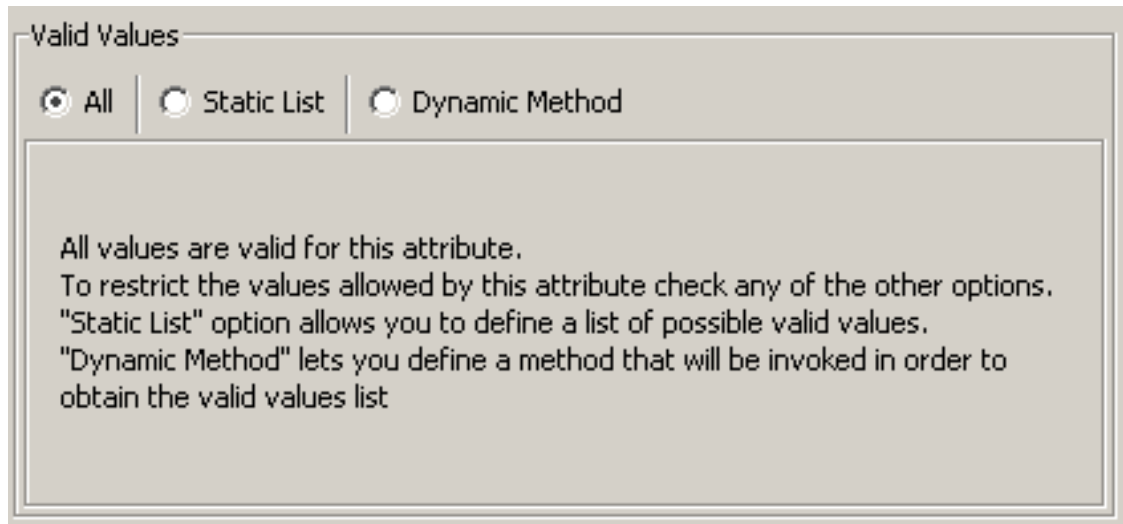
Groups and Check Expression

When the attributes that compose a group have check expression defined, you must define at group definition level that the group has to be included in the Check action. To do so, go to the group attribute edition panel and select the check box "Is it included in check?". By default this property is set to false.

Valid Values

User defined Fuego Object attributes can have a list of *valid values* defined. These valid values are displayed to the process user through presentations in combo boxes or radio buttons.

The definition is done in the lower portion of the attributes properties dialog.



There are three possible ways to define **valid values**:

- *All*: attributes that do not have valid values; the attribute is not shown in a combo or radio group.
- *Static List*: the valid values are input manually at this time, they are static, predefined or hardcoded by the Fuego Object designer at this point.
- *Dynamic Method*: the valid values come from the invocation of a method; they are not hardcoded, they are dynamic.

Static List

The process developer chooses a static list definition for the valid values of an attribute in the following two ways:

- **Static list description: selected** the list of valid values are not the actual values but the descriptions of each of the values.

The 'Valid Values' dialog box shows three radio buttons at the top: 'None', 'Static List' (selected), and 'Dynamic Method'. Below these are four icons (Add, Remove, Move Up, Move Down) and a checked checkbox labeled 'Static list descrip...'. The main area contains a table with two columns: 'Value' and 'Description'.

Value	Description
IN	Stock IN
OUT	Out

- **Static list description: not selected** the combo or radio group shown at presentation time are the actual valid values input by the designer.

The 'Valid Values' dialog box shows three radio buttons at the top: 'None', 'Static List' (selected), and 'Dynamic Method'. Below these are four icons (Add, Remove, Move Up, Move Down) and an unchecked checkbox labeled 'Static list description'. The main area contains a single column labeled 'Value'.

Value
IN
OUT

Editing the list of static valid values

This dialog provides four buttons at the top of the list of the static valid values area in order to *Add*, *Remove*, *Move up* or *Move down* a value row.



Add: To add a new valid value, click the *add* button (plus icon). This adds a new row to the end of the list of valid values or a new empty row immediately after the row that is selected.

Remove: To remove a valid value row, select it or use the TAB key to move to the row you want to delete and click the *remove* button (minus icon).

Move up and Move down: The list of valid values can be sorted by selecting each row and moving it up or down.

To edit the value contained in a row cell of the valid value, select it by clicking on it or move to it with the TAB key. Press the F2 function key. This enables you to edit the value.

Dynamic Method

A dynamic method allows the selection of valid values provided through a method with no arguments and that returns an array of the same type as the attribute.

The invoked method may return only the array of valid codes to the attribute, for example, *String[]*, *Int[]*, *Decimal[]* and so on, depending on the attribute's type. To do so, leave the *Description* check box unselected, which means that the method will not return descriptions but valid codes.

The 'Valid Values' dialog box has three radio buttons at the top: 'None', 'Static List', and 'Dynamic Method'. The 'Dynamic Method' radio button is selected. Below these is a checkbox labeled 'Description' which is currently unchecked. Underneath the checkbox is a section titled 'Method to load valid values without description' containing a text input field and a 'New' button. The text input field contains the value 'String[]'.

In addition, it may be possible that the invoked method returns an array of the descriptions for each valid value of the attribute. The returned type would be *String[Attribute Type]*, for example:

- *String[String]*, if the attribute's type is *String*,
- *String[Int]*, if the attribute's type is *Int*,
- *String[Decimal]*, if the attribute's type is *Decimal*,
- and so on, depending on the attribute's type.

To do this, select the *Description* check box. The method will return the descriptions of the valid codes.

The 'Valid Values' dialog box is shown with the same 'Dynamic Method' radio button selected. However, the 'Description' checkbox is now checked. The section title below it is 'Method to load valid values with description'. The text input field now contains the value 'String[String]'.

As shown in the images above, the returned type of the expected method is shown as a tooltip of the combo box, depending on the *Description* property.

Clicking the **New** button automatically generates a method with the properties already set according to the valid value definition you have made. Of course, you will have to complete the method's code in order to retrieve the valid values and descriptions. You are prompted to name the method.

A typical application of the above is to retrieve a list of valid values from a database. The following examples are simple, but will show you how the retrieval works.

String type attribute

- Suppose the *Customer* Fuego Object has a user-managed attribute *customerType*, which may have the valid values:
 - RET: Retail Customer
 - GLOB: Global Customer
 - NAT: National Customer

Let's construct both a list of valid values showing only the valid codes and a list of valid values showing the description of each possible value.

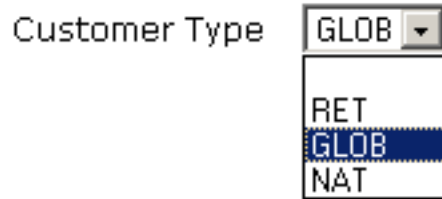
Defining a method like:

- *findCustType* that returns type *String[]* and its code is:




```
return ["RET", "GLOB", "NAT"]
```

we will obtain, at presentation runtime, the list of valid *codes* the attribute can be set to, as follows:



Defining a method like:

- With another *findCustTypeDescr* method that returns a `String[String]` (because the attribute type is *String* as well) with the following code:

```
custTypeVV["RET"]="Retail Customer"  
custTypeVV["GLOB"]="Global Customer"  
custTypeVV["NAT"]="National Customer"  
  
return custTypeVV
```

where *custTypeVV* is a local variable defined as `String[String]`, the list of valid values at presentation runtime will be the description set for each valid value, as shown below:



Decimal type attribute

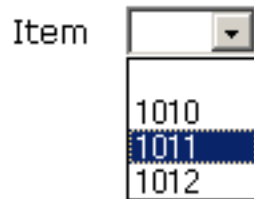
- Suppose, for this example, that the list of possible items to order is known and hardcoded. Then, let's define a Fuego Object to add the pricing list name *Pricing* and an attribute of this name *Item* that may have the following possible codes:
 - 1010 : Color TV "XX" 19
 - 1011 : Color TV "XX" 25
 - 1012 : Color TV "XX" 32

Defining a method like:

- *findItems* that returns type *Int[]* and its code is:

```
return [1010, 1011, 1012]
```

we will obtain, at presentation runtime, the list of valid *codes* the attribute can be set to, like the following:



Defining a method like:

- With another *findItemsDescr* method, which returns a `String[Int]` (because the attribute type is *Int* as well) with the following code:

```
itemVV[1010]="Color TV XX 19"
itemVV[1011]="Color TV XX 25"
itemVV[1012]="Color TV XX 32"

return itemVV
```

where *itemVV* is a local variable defined as `String[Int]`, the list of valid values at presentation runtime will be the description set for each valid value, as shown below:



All the examples for valid values include an empty value that represents the *null* value, as the attributes are defined as possibly null (*Not null* property is not checked). This is automatically done.

Read & Write access special considerations

Valid values are only used by Fuego Object presentations or by user access to the *attributeValidValues* method.

If you need to validate that the value to be set is within the valid values of an attribute, you need to specify it within your code. For example:

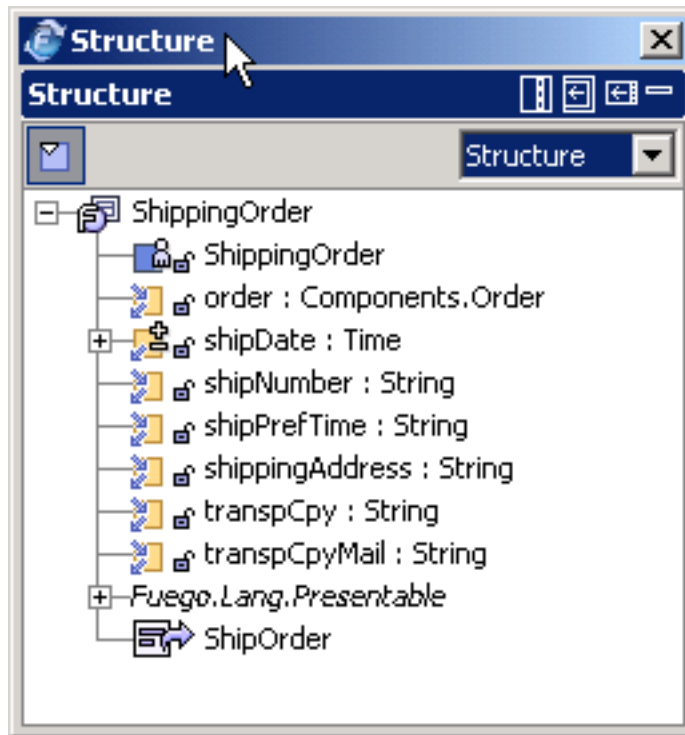
```
if test in customer_typeValidValues() then
    display this.customer_type + " is not a valid value"
else
    display this.customer_type + " is a valid value"
end
```

For further information, refer to **Fuego Object Examples-Project Catalog**.

Inner Fuego Object

You define an Inner Fuego Object when the type of a Fuego Object attribute is another Fuego Object. This is done by selecting the **Browse** button next to the attribute type field and selecting the Fuego Object from the Project Catalog.

In the following example, the *Shipping Order* is a Fuego Object itself with specific data about shipping but it also contains an *Order*, the one being shipped.



Initialize the inner Fuego Object

You must always initialize the inner Fuego Object that you use as an attribute.

Suppose that the *order* attribute is an inner Fuego Object (Order). You must initialize it in the group constructor method using the following statement:

```
order = OrderFulfillment.Order()
```

If you do not do this, the runtime exception *InvalidXOAttributeException* is thrown.

fuego.xobject.runtime.InvalidXOAttributeException: The attribute **XX** is not initialized. It is an attribute of Fuego Object type. That is why it must be initialized before displaying the presentation.

Refer to **Fuego Object Examples** for further information on how to implement it.

Fuego Objects Groups

A group is a simple way to keep track of a list of related attributes. A group's common use is to display multiple rows of information on a presentation. You can add any number of attributes to a group.

An example of a group is the list of items ordered on an invoice. For example, if you order three different books from a book store, the three books are grouped together on one invoice. Each book is a separate line or order item on the order. Each line item also contains attributes such as the book title, the ISBN number of the book and the book price. These attributes are held in a group in a Fuego Object.

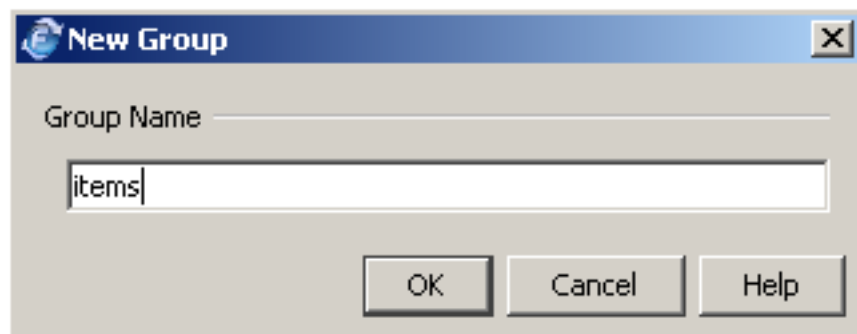
Handling Groups in a Fuego Object

Adding a Group

A *group* is created as an array of objects. The name given to the group at creation time is used to name both the attribute and the object that it contains. For example, if the attribute name is *items* then the group component will be named 'Items.'

To add a group to a Fuego Object:

1. Right-click on the Fuego Object to which you want to add the group.
2. Enter the name of the group you want to create.



Deleting a Group

To delete a group from a Fuego Object, perform one of the following:

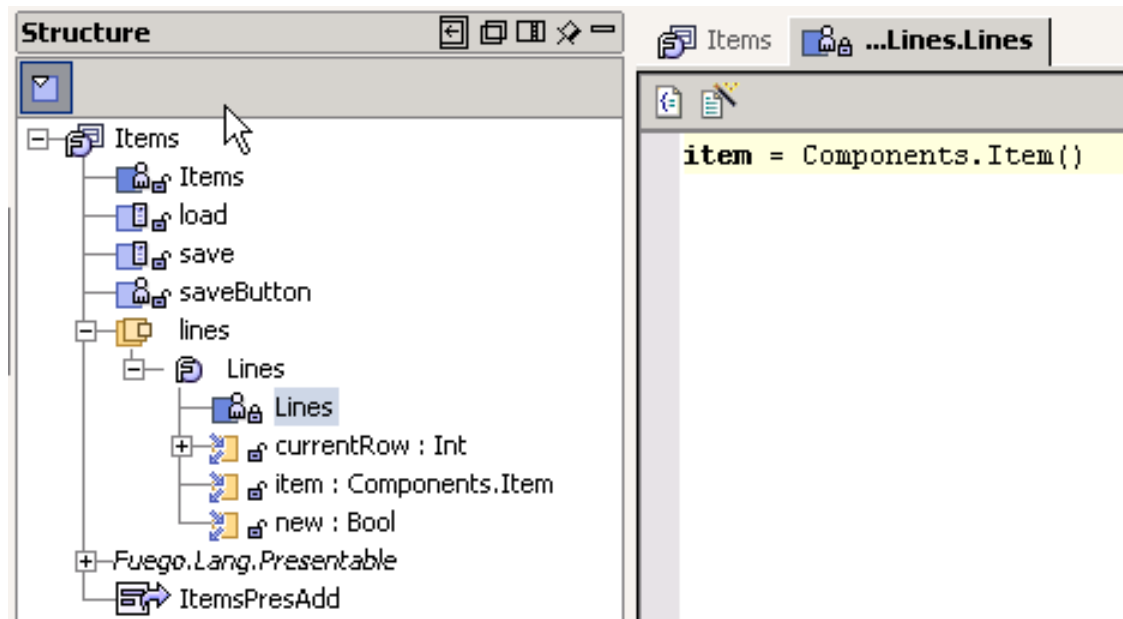
- From the structure panel, right-click on the group attribute and select *delete* from the menu.
- From the Fuego Object editor panel, right-click on the group attribute and select *delete* from the menu.

Group Structure

The attribute added to the Fuego Object is an array of objects. Any array method can be applied to a group. See Array Functions. If a group named *item* is added to the *Order* Fuego Object, its definition would be:



The inner object has to be created within the group constructor.



The attributes contained in a group can be defined as:

- basic types (Int, String, Time, etc.)
- Arrays
- Iterators
- Associative Arrays
- or any other component defined within the catalog

The group structure is shown both in the Structure flap and in the Fuego Object panel.

Working with a Fuego Object group of a previous version

If you are working on a Fuego Object group attribute on an imported or cataloged Fuego Object from a previous version, the structure of a group attribute differs. A previous version Fuego Object created into FuegoBPM Studio (by importing it or by cataloging it, see Handling

Fuego Objects) has an attribute *currentRow* created by default, with the following characteristics:

- the *currentRow* attribute is created and visible in the Fuego Object group structure,
- it has a *readAccess* method with the following code (e.g.):

```
return LaborTracking.Groups.lines.indexOf(this)
```

- *currentRow* is not a reserved word,
- the *currentRow* attribute can be removed.

Fuego Object Group Attributes Practices

Groups and Check Expression

When the attributes that compose a group have check expression defined, you must define at group definition level that the group has to be included in the Check action. To do so, go to the group attribute edition panel and select the check box "Is it included in check?". By default this property is set to false.

Referring to Attributes within the Group

Attributes within a group are not different from simple attributes of the Fuego Object. For more information, refer to Fuego Object Attributes.

Note



You may occasionally need to refer to a special row of the group. To do so, use the array's method *indexOf*.

Get the Row in the Group that was Selected

```
// in a method inside the Group's Fuego Object  
lineSelected = groupName.indexOf(this)
```

Find a value in a group

```
someValue in attributeValidValues
```

Initializing Groups

- To empty out all the values in a group Fuego Object attribute, use the following statement:

```
clear orderItems
```

- To initialize an empty group with values

```
// first empty the Group  
clear this.groupName  
  
// use the 'extend' to add new rows to the group  
for i in 0...numberOfRowsInGroup do  
  extend this.groupName using id = someVariable1,  
  lob = someVariable2  
end
```

Calculating a Group's Attribute

If a group's attribute is the result of a calculation of other group's attributes, it is recommended that the first is defined as *virtual* and its *Read Access* method code is the calculation involving the other group's attribute.

Suppose that the attribute *itemTotal* of the group is the result of the multiplication of the other two groups attributes *itemPrice* and *itemQuantity*. Then, *itemTotal* is defined as *Virtual* and its *Read Access* method code is:

```
return itemPrice * itemQuantity
```

Calculating a Fuego Object's Attribute using Attributes in a Group

If you have a Fuego Object attribute which value is the result of some operation applied on a group attribute on all the lines, define the simple attribute as *Virtual* and overwrite its *Read Access*.

Suppose that "the sum of all the order line items total price is the order's total," then, the order's total attribute has to be defined as *Virtual* and its *Read Access* method code is:

```
return items[ ].itemTotal.sum( )
```

Extending the group array

The recommended way to extend a group array is using the *extend* statement, as shown below:

```
extend this.items
  using itemQuantity = Items.qty,
      itemPrice = Items.price,
      itemTotal = Items.qty * Items.price
```

Get Number of Rows in a Group

```
// "length" - number of rows in group (including null rows)
numberItemsInGroup = length(this.groupName)
// "count" - number of rows in group (excluding null rows)
numberNonNullItemsInGroup = count(this.groupName)
```

Fuego Objects Methods

Object's methods usually contain the code that understands and manipulates an object's state. Sometimes, objects have many tasks that cannot be represented as simple values to be read or modified, but that require computation.

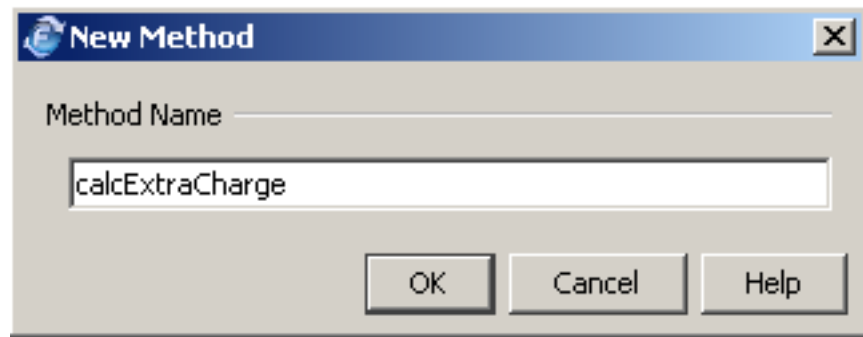
Each method takes a specific number of parameters. Each parameter has a specified type, a primitive type or a reference type (from the catalog). Methods also have a return type.

Adding a Method to a Fuego Object

Methods are reusable self-contained blocks of code used to perform operations and return values associated to your Fuego Object.

To add a method to a Fuego Object:

1. Right-click on the Fuego Object and select **New method** from the shortcut menu.
2. A dialog box is displayed.



3. Enter a name in the **Method Name** field. See **Naming conventions** for method naming conventions. Click the **OK** button. A blank Method scripting panel opens in the center panel of FuegoBPM Studio.

Fuego Objects provide the necessary elements to create a method: the code editor panel to write the method code and additional frames used to complete its definition, the method "properties" frame to indicate for example if the method "runs on server", and the method "variables" frame to define its local variables and input/output arguments.

Defining Method Properties

The screenshot shows a 'Properties' window with a blue title bar. It contains two expandable sections: 'Component Properties' and 'Method Properties'. The 'Method Properties' section is expanded, showing a table with the following properties:

Name	saveButton
Static	False
Cached	False
Abstract	False
Return Type	Void
Server side method	False

Below the table is a scrollable area with a downward arrow on the right side.

- *Name*: Method's name.
- *Static*: Defines whether the method is static or not. A static method is invoked on behalf of an entire class, not on a specific object instantiated from that class. Static methods are also known as class methods. A static method might perform a general task for all objects of the class - such as returning the next available serial number.
- *Cached*: When a method is defined as *cached*, it means that the first time it is executed the result of its execution is stored in a cache. The next time the method is invoked with the same arguments, instead of executing it again, it returns the values stored in the cache.
- *Return Type*: Type of value the method returns.
- *Server Side method*: You define whether the method runs on server or on client. Methods that use SQL require that you select the *Server side method* check box.

Cached methods

As mentioned before, when a method is defined as *cached* it means that the first time it is executed the result of its execution is stored in a cache. The next time the method is invoked with the same arguments, instead of executing it again, it returns the values stored in the cache.

When to use caching

It is convenient to use a cached method when it performs a time-consuming operation. For example, when using an invocation to a very slow external WebService or looking for a value in a slow database.

How does caching work?

When a *cached* method is executed, the cache is searched to determine if a result exists for the method and the input arguments. If a result does not exist, the method is executed. The output arguments (or result) associated to the input arguments are stored in the cache.

For example, having a *cached* method that returns its description, depending on a product code:

```
loadDescription(in code as Int, out description as String)
```

it is still used in the same way:

```
loadDescriptionFor Product  
  using code = 12  
  returning description = description
```

However, if the above method is executed twice, the first time it queries the database to look for the values, but the second time it

returns the stored result without executing the database query. If the method is invoked again but this time with different parameters, the method accesses the database and the new result is stored in the cache associated to the new parameters.

Results are cached until the size of the cache is reached. From then on, older results are discarded and replaced by more recent results.

Cached Methods Properties

- The cache is limited to 50 elements per method. This is not configurable in the present version.
- The input and output arguments can only be typified with predefined types, as String, Int, Real, etc.
- When the cache is full, the older or less recently used results are discarded.
- The cache is only cleaned when the program is restarted.
- A method can only be *cached* if it is defined as *static*.

Defining the Methods Variables

Instance	
Name	customerType
Type	String
Description	
Mode	in

Arguments	
Name	
Type	
Description	
Mode	

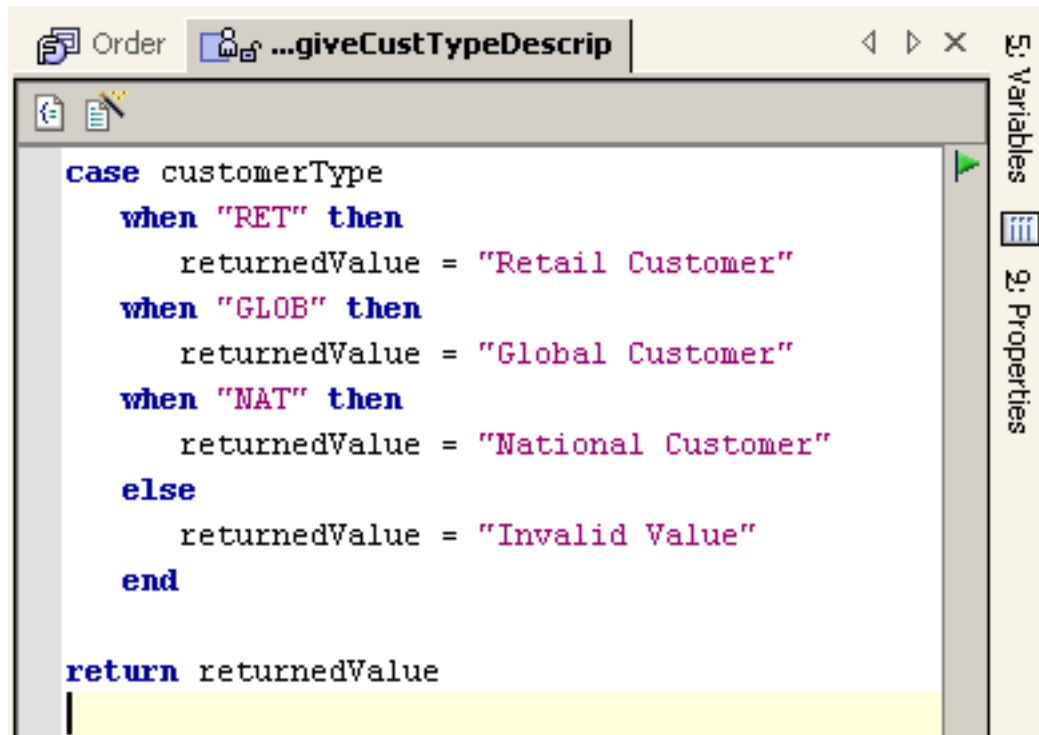
Local	
Name	i
Type	Int
Description	

You can define the local vars and arguments the method works with.

- Local variables scope is limited to the method.
- Argument variables can be defined as in, out or in/out.

Defining the Method Code

Enter the Method code for the method. Use variables and Fuego Object attributes from the Variables frame and components from the Project Catalog as needed. See *Sample methods* or the *Methods Guide* for additional examples.



The screenshot shows a code editor window with a title bar containing 'Order' and '...giveCustTypeDescrip'. The editor displays a Ruby-like case statement for determining customer types. The code is as follows:


```
case customerType
  when "RET" then
    returnedValue = "Retail Customer"
  when "GLOB" then
    returnedValue = "Global Customer"
  when "NAT" then
    returnedValue = "National Customer"
  else
    returnedValue = "Invalid Value"
  end
return returnedValue
```

On the right side of the editor, there is a vertical toolbar with icons for 'Variables' and 'Properties'.

Fuego Object Standard Methods

Fuego Object standard methods automatically appear in every Fuego Object component in the left panel of the Component Manager when a Fuego Object is created. Each of these methods and their use are explained below.

Warning

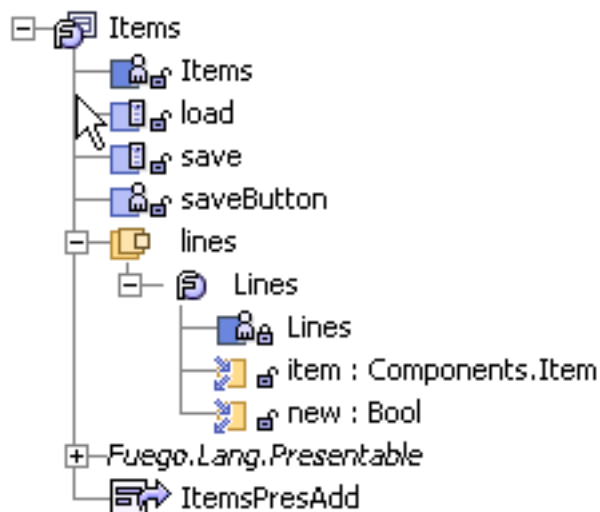
 **Note:** Additionally, hidden methods are created for each attribute added to the Fuego Object. These methods are called `get{AttributeName}` and `set{AttributeName}`. For example, for the attribute `first_name` two hidden methods are created `getfirst_name` and `setfirst_name`. This means that you should not create any methods with these same names. If you do, you will receive a Warning error message.

Constructor Methods

Constructor methods give the object an initial state. Fields can be initialized with a value when they are declared, which is sometimes sufficient to ensure a correct initial state. But often you need more

than simple data initialization to create the initial state: the creating code may need to supply initial data or perform operations that cannot be expressed as simple assignments.

Constructors have the same name as the class they initialize. Constructors, as well as methods, can have parameters (in, out or in/out) and the type of returned data is the Fuego Object they belong to.



When a Fuego Object is created the constructor method is created by default. Sometimes when a Fuego Object is instantiated (created), some default values have to be automatically assigned to object attributes. For example, if you have a Fuego Object called *USACountry*, the Fuego Object may have an attribute *states* (string array). Inside the {constructor} method code, you can initialize the array with the *50 United States*.

Warning



When coding a Fuego Object constructor, never call a client side component or any component that needs user interaction. For example, do not add input or display statements in constructor method code. All code within the constructor method should require no user intervention.

Clone Method

<i>Name</i>	clone
<i>Description</i>	The clone method creates a new object exactly like the object you choose to clone.
<i>Arguments</i>	None.
<i>Return Value</i>	Object.

The clone method creates a new object exactly like the object you choose to clone. For example, if you have an instance variable that refers to a Fuego Object of Person type (personObj is the instance variable that refers to the Fuego Object in the example below) and you want to create a new Person object instance variable (person2Obj in the example below) based on the information contained in personObj, you use the clone method.

```
// Correct way to create a copy of an Fuego Object
personObj.firstName = "John"
personObj.lastName = "Smith"

// Clone personObj into person2Obj
person2Obj = Person(personObj.clone())
```

This cloning method is based on a Java concept. If you enter the following code:

```
// DO NOT TRY TO CLONE THIS WAY
person2Obj = personObj
```

the code copies only the reference of the Fuego Object. Both Fuego Object variables are pointing at the same object. In other words, if you change:

```
personObj.firstName = "Dan"
```

```
personObj.firstName = "Dan"
```

then `person2Obj.firstName` is automatically changed to "Dan" as well. The clone method gives you two distinct objects when you use it to clone an object.

RefreshValidValues Method

<i>Name</i>	refreshValidValues
<i>Description</i>	The <code>refreshValidValues</code> method refreshes the valid values of an attribute that contains valid values.
<i>Arguments</i>	String <code>dataId</code> .
<i>Return Value</i>	None.

The `refreshValidValues` method refreshes the valid values of an attribute that contains valid values. This method receives a string (the name of the attribute in double quotes that has valid values to refresh) and it does not return anything.

The following example Method refreshes valid values in an attribute called *custName*.

```
refreshValidValues("custName")
```

Standard Methods in Presentable Fuego Objects

Methods to set and to get presentation properties at runtime for simple components

The following table shows the list of methods that allow you to

modify presentation-properties at runtime. They only exist in presentable Fuego Objects (those with presentations).

Description	Name	Arguments	Return value	Component type
To get the current background color of one component	<i>getBackgroundColor</i>	String componentId	String color	Button, Cell, Check, Combo, Datetimepicker, Interval, Multilinetext, Radio, Text
To get the current foreground color of one component	<i>getForegroundColor</i>	String componentId	String color	Button, Cell, Check, Combo, Datetimepicker, Interval, Multilinetext, Radio, Text
To dynamically get the component text if has any.	<i>getText</i>	String componentId	String	component type
To set the background color of one component	<i>setBackgroundColor</i>	String componentId, String color	void	Button, Cell, Check, Combo, Datetimepicker, Interval, Multilinetext, Radio, Text
To dynamically set the	<i>setEditable</i>	String componentId, boolean editable	void	component type

Description	Name	Arguments	Return value	Component type
editable property.				
To set the foreground color of one component	<i>setForegroundColor</i>	String componentId, String color	void	Button, Cell, Check, Combo, Datetimepicker, Interval, Multilinetext, Radio, Text
To dynamically set a text to a component.	<i>setText</i>	String componentId, String text	void	Any component that accepts the <i>display</i> property.
To set visible or not a component	<i>setVisible</i>	String componentId, boolean visible, boolean collapsed	void	component type
To ask whether a component is visible or not a component	<i>isVisible</i>	String componentId, boolean visible	void	component type
To dynamically ask if a component has been configured as editable or not.	<i>isEditable</i>	String componentId, boolean editable	void	component type
To dynamically set a group's headers. The	<i>setGroupHeaderText</i>	String groupComponentId, String[] headers	void	component type

Description	Name	Arguments	Return value	Component type
array containing the new headers for the group should have the size equal to the number of columns the group has.				
To dynamically get the groups header.	<i>getGroupHeaderText</i>	String groupComponentId	String[]	component type
To dynamically set a fixed URL to a component.	<i>setLink</i>	String componentId, String url	void	Label, Images, Button
To dynamically get the URL to a component.	<i>getLink</i>	String componentId	String	Label, Images, Button
To get the current CSS file applied to the presentation	<i>getCSSClassName</i>	String componentId	String	Button, Cell, Check, Combo, Datetimepicker, Interval, Multilinetext, Radio, Text
To	<i>setCSSClassName</i>	String componentId,	void	Button, Cell,

Description	Name	Arguments	Return value	Component type
dinamically set a CSS class to a component		String cssClass		Check, Combo, Datetimepicker, Interval, Multilinetext, Radio, Text
To dinamically know if the component is set as <i>required</i> .	<i>isRequired</i>	String componentId	Bool	Button, Cell, Check, Combo, Datetimepicker, Interval, Multilinetext, Radio, Text
To dinamically set a component as <i>required</i> .	<i>setRequired</i>	String componentId , Bool required	void	Button, Cell, Check, Combo, Datetimepicker, Interval, Multilinetext, Radio, Text
To open a given URL.	<i>openURL</i>	String url, Bool newWindow, Bool standardBrowser	void	Button, Cell, Check, Combo, Datetimepicker, Interval, Multilinetext, Radio, Text
To open a presentation in a show mode. (see section below)	<i>show</i>	Instance foInstance, String presentationName	void	Button, Cell, Check, Combo, Datetimepicker, Interval, Multilinetext, Radio, Text

Methods to set and to get presentation properties at runtime for group components

Standard methods are defined to provide cell renderer functionality in array components, as to display certain rows highlighting it in another color or change the font to italics.

Description	Name	Arguments	Return value
To set the background color	<i>setBackgroundColors</i>	<i>String</i> <i>groupComponentId</i> , <i>String</i> <i>columnComponentId</i> , <i>int index</i> , <i>String color</i>	void
To get the background color	<i>getBackgroundColor</i>	<i>String</i> <i>groupComponentId</i> , <i>String</i> <i>columnComponentId</i> , <i>int index</i>	String
To set the foreground color	<i>setForegroundColor</i>	<i>String</i> <i>groupComponentId</i> , <i>String</i> <i>columnComponentId</i> , <i>int index</i> , <i>String color</i>	void
To get the foreground color	<i>getForegroundColor</i>	<i>String</i> <i>groupComponentId</i> , <i>String</i> <i>columnComponentId</i> , <i>int index</i>)	String
To set as editable an element of the group	<i>setEditable</i>	<i>String</i> <i>groupComponentId</i> , <i>String</i> <i>columnComponentId</i> , <i>int index</i> , <i>boolean</i> <i>editable</i>	void
To know if an element of the	<i>isEditable</i>	<i>String</i> <i>groupComponentId</i> ,	boolean

Description	Name	Arguments	Return value
group is editable		<i>String</i> <i>columnComponentId</i> , <i>int index</i>	
To set as visible an element of the group	<i>setVisible</i>	<i>String</i> <i>groupComponentId</i> , <i>String</i> <i>columnComponentId</i> , <i>int index</i> , <i>boolean</i> <i>visible</i>	void
To know if an element of the group is visible	<i>isVisible</i>	<i>String</i> <i>groupComponentId</i> , <i>String</i> <i>columnComponentId</i> , <i>int index</i>	boolean
To set a text	<i>setText</i>	<i>String</i> <i>groupComponentId</i> , <i>String</i> <i>columnComponentId</i> , <i>int index</i> , <i>String text</i>	void
To get a text	<i>getText</i>	<i>String</i> <i>groupComponentId</i> , <i>String</i> <i>columnComponentId</i> , <i>int index</i>	String
To set a link within a group or repeatable section	<i>setLink</i>	<i>String</i> <i>groupComponentId</i> , <i>String</i> <i>columnComponentId</i> , <i>int index</i>	, String
To get a link in a group or repeatable section	<i>setLink</i>	<i>String</i> <i>groupComponentId</i> , <i>String</i> <i>columnComponentId</i> , <i>int index</i> , <i>String link</i>	void

where:

- *componentId*: component name,
- *color*: RGB value of the color you want to set,
- *visible*: true or false.

setVisible method considerations

Last parameter *collapsed*, which works only in HTML means:

- When its value is **true**: the display of the element is turned off; all child elements also have their display turned off. The document is rendered as if the element did not exist in the document tree.
- When its value is **false**: the content of an element box is rendered, including the borders and backgrounds. It still affects document layout as if it was visible.

setting links

The value set in the *link* property overwrites the *onClick* when both are present.

setting color

To set a color, you can either set the Hexadecimal value, if you know it, or use the *Color* enumeration provided by FuegoBPM within its default catalog, located in *Fuego.ui.Color*.

You can set colors to the Fuego Object's components by using the *rgb()* method, which returns the hexadecimal representation of the indicated color. For example:

```
setBackgroundColor(Fuego.Ui.Color.BLACK.rgb())
```

The available colors are:

	Color	Hex Equivalent
#FFFFFF	White	#FFFFFF
#FF0000	Red	#FF0000
#FFA500	Orange	#FFA500
#FFFF00	Yellow	#FFFF00
#008000	Green	#008000
#0000FF	Blue	#0000FF
#800080	Purple	#800080
#000000	Black	#000000
#F0F8FF	Alice Blue	#F0F8FF
#FAEBD7	AntiqueWhite	#FAEBD7
#00FFFF	Aqua	#00FFFF
#7FFFD4	Aquamarine	#7FFFD4
#F0FFFF	Azure	#F0FFFF
#F5F5DC	Beige	#F5F5DC
#FFE4C4	Bisque	#FFE4C4
#FFEBCD	Blanched Almond	#FFEBCD
#8A2BE2	Blue Violet	#8A2BE2
#A52A2A	Brown	#A52A2A
#DEB887	Burly Wood	#DEB887
#5F9EA0	Cadet Blue	#5F9EA0
#7FFF00	Chartreuse	#7FFF00
#D2691E	Chocolate	#D2691E
#FF7F50	Coral	#FF7F50
#6495ED	Corn Flower Blue	#6495ED
#FFF8DC	Corn Silk	#FFF8DC

	Color	Hex Equivalent
#DC143C	Crimson	#DC143C
#00FFFF	Cyan	#00FFFF
#00008B	Dark Blue	#00008B
#008B8B	Dark Cyan	#008B8B
#B8860B	Dark Goldenrod	#B8860B
#A9A9A9	Dark Gray	#A9A9A9
#006400	Dark Green	#006400
#BDB76B	Dark Khaki	#BDB76B
#BD008B	Dark Magenta	#BD008B
#556B2F	Dark Olive Green	#556B2F
#FF8C00	Dark Orange	#FF8C00
#9932CC	Dark Orchid	#9932CC
#8B0000	Dark Red	#8B0000
#E9967A	Dark Salmon	#E9967A
#8FBC8F	Dark Sea Green	#8FBC8F
#483D8B	Dark Slate Blue	#483D8B
#2F4F4F	Dark Slate Gray	#2F4F4F
#00CED1	Dark Turquoise	#00CED1
#9400D3	Dark Violet	#9400D3
#FF1493	Deep Pink	#FF1493
#00BFFF	Deep Sky Blue	#00BFFF
#696969	Dim Gray	#696969
#1E90FF	Dodger Blue	#1E90FF
#B22222	Fire Brick	#B22222
#FFFAF0	Floral White	#FFFAF0
#228B22	Forest Green	#228B22
#FF00FF	Fuchsia	#FF00FF
#DCDCDC	Gainsboro	#DCDCDC
#F8F8FF	Ghost White	#F8F8FF
#FFD700	Gold	#FFD700

	Color	Hex Equivalent
#DAA520	Goldenrod	#DAA520
#808080	Gray	#808080
#ADFF2F	Green Yellow	#ADFF2F
#F0FFF0	Honeydew	#F0FFF0
#FF69B4	Hot Pink	#FF69B4
#CD5C5C	Indian Red	#CD5C5C
#4B0082	Indigo	#4B0082
#FFFFFF0	Ivory	#FFFFFF0
#F0E68C	Khaki	#F0E68C
#E6E6FA	Lavender	#E6E6FA
#FFF0F5	Lavender Blush	#FFF0F5
#FFFACD	LemonChiffon	#FFFACD
#ADD8E6	Light Blue	#ADD8E6
#F08080	Light Coral	#F08080
#E0FFFF	Light Cyan	#E0FFFF
#FAFAD2	Light Goldenrod Yellow	#FAFAD2
#90EE90	Light Green	#90EE90
#D3D3D3	Light Grey	#D3D3D3
#FFB6C1	Light Pink	#FFB6C1
#FFA07A	Light Salmon	#FFA07A
#20B2AA	Light Sea Green	#20B2AA
#87CEFA	Light Sky Blue	#87CEFA
#778899	Light Slate Gray	#778899
#B0C4DE	Light Steel Blue	#B0C4DE
#FFFFE0	Light Yellow	#FFFFE0
#00FF00	Lime	#00FF00
#32CD32	Lime Green	#32CD32
#FAF0E6	Linen	#FAF0E6
#FF00FF	Magenta	#FF00FF
#800000	Maroon	#800000

	Color	Hex Equivalent
#66CDAA	Medium Aquamarine	#66CDAA
#0000CD	Medium Blue	#0000CD
#BA55D3	Medium Orchid	#BA55D3
#9370DB	Medium Purple	#9370DB
#3CB371	Medium Sea Green	#3CB371
#7B68EE	Medium Slate Blue	#7B68EE
#00FA9A	Medium Spring Green	#00FA9A
#48D1CC	Medium Turquoise	#48D1CC
#C71585	Medium Violet Red	#C71585
#191970	Midnight Blue	#191970
#F5FFFA	Mint Cream	#F5FFFA
#FFE4E1	Misty Rose	#FFE4E1
#FFDEAD	Navajo White	#FFDEAD
#000080	Navy	#000080
#FDF5E6	Old Lace	#FDF5E6
#808000	Olive	#808000
#6B8E23	OliveDrab	#6B8E23
#FF4500	Orange Red	#FF4500
#DA70D6	Orchid	#DA70D6
#EEE8AA	Pale Goldenrod	#EEE8AA
#98FB98	Pale Green	#98FB98
#AFEEEE	Pale Turquoise	#AFEEEE
#DB7093	Pale Violet Red	#DB7093
#FFEFD5	Papaya Whip	#FFEFD5
#FFDAB9	Peach Puff	#FFDAB9
#CD853F	Peru	#CD853F
#FFC0CB	Pink	#FFC0CB
#DDA0DD	Plum	#DDA0DD
#B0E0E6	Powder Blue	#B0E0E6
#BC8F8F	Rosy Brown	#BC8F8F

	Color	Hex Equivalent
#4169E1	Royal Blue	#4169E1
#8B4513	Saddle Brown	#8B4513
#2E8B57	Sea Green	#2E8B57
#FFF5EE	Sea Shell	#FFF5EE
#A0522D	Sienna	#A0522D
#C0C0C0	Silver	#C0C0C0
#87CEEB	Sky Blue	#87CEEB
#6A5ACD	Slate Blue	#6A5ACD
#708090	Slate Gray	#708090
#FFFAFA	Snow	#FFFAFA
#00FF7F	Spring Green	#00FF7F
#4682B4	Steel Blue	#4682B4
#D2B486	Tan	#D2B486
#008080	Teal	#008080
#D8BFD8	Thistle	#D8BFD8
#FF6347	Tomato	#FF6347
#40E0D0	Turquoise	#40E0D0
#EE82EE	Violet	#EE82EE
#F5DEB3	Wheat	#F5DEB3
#F5F5F5	White Smoke	#F5F5F5
#9ACD32	Yellow Green	#9ACD32

Note



Remember that the *presentation* is not a component, so any of these methods can be used to dynamically set or get values to it.

show method

The *show* method opens a Fuego Object presentation.

Its execution is an asynchronous execution and can be performed from a running presentation by a Fuego Object method.

The show mode for each component is:

editors	They are not editable
Graphics	Their actions are enabled
Buttons	They are enabled
Lables	They have their actions or URLs enabled
Images	They have their actions or URLs enabled

When a presentation is invoked by the *show* method from another presentation and it is submitted, the *caller* presentation is shown again.

Every time a show is started or finished a new page is registered in the browser history, independently if it is a remote scripting execution or not.

See Dashboard Standard Methods for more information about its use related to building dashboards.

openURL method

This method opens a given URL in a browser window. Its arguments are:

String url Url to open

boolean newWindow, when set to TRUE, opens the given urls in a new windows. When set to FALSE, search for the opener window and opens the given url in it.

boolean standardBrowser, when set to TRUE, opens the given url in a standard browser. When set to FALSE, opens the url in a browser without toolbars, urls, etc.

See Dashboard Standard Methods for more information about its use related to building dashboards.

Standard Methods for FuegoBPM Dashboard components

For information about these methods, please refer to the specific section FuegoBPM Dashboard Standard Methods.

Presentation Buttons methods

The following is the list of methods that can be associated to a button in a presentation to perform an action.

<i>Name</i>	submit
<i>Description</i>	The check and required attributes' expressions are run. If the validations expressions are correct, the presentation is closed. Any changes made to the Fuego Object attributes are retained. The method returns submit to the calling process Method.
<i>Arguments</i>	String text.
<i>Return Value</i>	None.

The string text passed as argument to the *submit* method, is the value returned to the calling process Method instead of the default text returned *submit*. So, if you invoke this method, for example, with the text *approved* as argument, the value returned by the *submit* method will be *approved*. See Fuego Objects Examples for a detailed example of this usage.

<i>Name</i>	cancel
<i>Description</i>	The presentation is closed. The method returns null to the calling process Method. The returned value for the input execution of the presentation is null. Any changes made to the Fuego Object attributes

<i>Name</i>	cancel
	are cancelled. The attributes' values are the same ones they have set before executing the input of the presentation.
<i>Arguments</i>	None.
<i>Return Value</i>	None.

<i>Name</i>	reset
<i>Description</i>	The presentation remains open. The presentation fields revert back to the values they had when the input was executed.
<i>Arguments</i>	None.
<i>Return Value</i>	None.

<i>Name</i>	refresh
<i>Description</i>	The presentation remains open. Each of the presentation's fields are refreshed to their actual values. This method is useful when you have database or dynamic values referenced by attributes.
<i>Arguments</i>	None.
<i>Return Value</i>	None.

<i>Name</i>	printPresentation
<i>Description</i>	The printPresentation method is called when you want to print a Fuego Object presentation. It uses the print function of the browser that

<i>Name</i>	printPresentation
	is viewing the presentation.
<i>Arguments</i>	None.
<i>Return Value</i>	None.

<i>Name</i>	save
<i>Description</i>	The save method is similar to submit in that it closes the presentation, but it does not execute the validation logic.
<i>Arguments</i>	None.
<i>Return Value</i>	None.

The *save* method is usually used when a presentation has fields with constraints such as being required or complying with a certain format, and it must be allowed to the user to *submit* the presentation or to *save the changes for later*.

Suppose an *Expense Report* process which has an *Expense* Fuego Object that allows the user to complete the expense report and submit it to the supervisor for its approval. Within the Fuego Object presentation, the users have the chance to *submit the report to their supervisor* or *to save the report and continue editing it at a later time*. Some fields in the presentation are marked as required (e.g, the description, project, and category for each expense item, etc.) and they are validated upon submission. Using the *save* method is the solution to implement the second option provided to the user, the way to save the changes to the Fuego Object without performing the validations.

<i>Name</i>	action
<i>Description</i>	This selection enables the Method invoked property and allows you to select any method in the Fuego Object from the Method invoked

<i>Name</i>	action
	drop-down menu. The action button property allows you to create your own custom behavior for a button.
<i>Arguments</i>	None.
<i>Return Value</i>	None.

Refer to the Fuego Objects Examples for examples containing these button methods.

Methods to change the keyword focus

These methods transfers the keyboard focus to the component of the given id.

<i>Name</i>	<i>Arguments</i>	<i>Return Value</i>
focus	String componentId	None.
focus	String groupComponentId, String columnComponentId, int index	None.

Methods to handle Errors in presentations

These methods show user error or warning messages.

<i>Name</i>	<i>Arguments</i>	<i>Return Value</i>
showWarning	String warning	None.
showError	String error	None.

Sample Methods

The following are some sample methods that may help you develop

your own methods.

Database Methods

Use SQL keywords to call a database (remember that the database has to be cataloged in the Project Catalog before you can make a call to it from Methods). The following Method statements call the database *MYPRODUCTS* and look at the *prod_name* and *itemNbr* columns. If the *itemNbr* variable entered by the user matches the number in the *itemNbr* column, the *prod_name* description is returned to the user.

```
for each row in
  SELECT prod_name
  FROM MYPRODUCTS
  WHERE prod_id = arg.itemNbr
do
  p = row.prod_name
end

return p
```

The next example looks at a part number entered by a user on a form and compares it to data in a table. Then the Method returns the price of the item that corresponds to the part number entered by the user.

```
for each row in
  SELECT price
  FROM ITEMS
  WHERE item = arg.item
do
  p = row.price
end

return p
```

The last database example comes from a Human Resources process. It takes information from a form filled out by a Human Resources

employee and adds a new employee to an employee database.

```
INSERT INTO EMPLOYEES VALUES
(lastName, firstName, initial, address,
city, state, zip, dateOfBirth, SSN)
```

All database tables should be first added to the Project Catalog before you can call them in Method statements. See SQL Components for detailed information.

Calculations

Use methods to make difficult calculations simple and reusable. The following Method code calculates and returns the total owed by a customer on an invoice form. The local variable *i* is used to keep track of the rows on the invoice, making sure to add the price in each row to the total.

```
for each item in items
do
  orderTotal = orderTotal + item.itemTotal
end

return orderTotal
```

Lengthy Method

Use methods to store lengthy code. For example, you may use an input statement that has a drop-down box allowing the user to choose from the 50 states of the United States. Instead of typing this input statement over and over in different code or even copying and pasting, you can add it to your catalog as a method.

Regular Method call from an activity:


```
input "Customer Name" custName,
"Address" address,
    "City" city,
"State" state in ["AL", "AK", "AZ", "AR", "CA",
"CO", "CT", "DE", "FL", "GA", "HI", "ID", "IL",
"IN", "KY", "LA", "ME", "MD", "MA", "MI", "MN",
"MS", "MO", "MT", "NE", "NV", "NH", "NJ", "NM",
"NY", "NC", "OH", "OK", "OR", "PA", "RI", "SC",
"SD", "TN", "TX", "UT", "VT", "VA", "WA", "DC",
"WV", "WI"]
using title = "Customer Information",
    buttons = ["Ok", "Cancel" ]
    returning mySelection = selection
```

Method call in code (method name is getStateList; component name is myComponent):

```
getStateList myComponent
```

As you can see by the example, methods make entering lengthy code much easier. You only enter the code once in a method and then call the method whenever you need to use it.

For further information on how methods are used from presentations, please refer to Fuego Object Presentations.

Tip



When editing a method in the Method Editor, by clicking *Ctrl + Space* after the name of a module, submodule, component you can obtain its complete list of available attributes and methods (For further information please refer to Methods.)

Fuego Object Methods and Behavior Inheritance Implementation

Please refer to Implementing Behavior Inheritance to learn how to

deal with redefinition of inherited methods.

Testing your Methods in Debug Mode

When you are testing your methods with the Editor Debug Mode, transactions contained in your methods may be defined as to be committed or to be rolled back.

Go to the Studio Preferences and take a look at the *Allow the debugger to commit transactions* property.

If it is checked, then the transactions embedded in your method will be committed. If not, all transactions will be rolled back when the method stops running.

Presentables and Non-Presentables Fuego Objects



A Fuego Object is initially created as non-presentable and only contains attributes and methods with a very simple interface. These components do not have either presentations or methods related to presentations (*getBackgroundColor*, *getForegroundColor*, *getText*, *setBackgroundColor*, *setEditable*, *setForegroundColor*, *setText*, *setVisible*, *cancel*, *printPresentation*, *refresh*, *refreshValidValues*, *reset* and *submit*). See Methods for further information about these methods.

A Fuego Object is changed to presentable when a presentation is added to it. When you add a presentation to the Fuego Object, it only becomes presentable once you have accepted all the steps of the presentation creation. As a consequence, its icon in both structure and project trees is changed.

When a presentation is removed, if it is the last one, the Fuego Object is set to non-presentable again and it will only be a data container. As a consequence, the icons are changed.

Representation

The Fuego Object representation icons according to its type are:

Icon	Fuego Object Type
	Non-Presentable FuegoObject
	Presentable FuegoObject

Fuego Objects Presentations

Overview

Creating a presentation for a Fuego Object provides you with the ability to show how attributes function together in a visual form that can be displayed in Work Portal. A presentation is a quick and easy way to display information contained in a Fuego Object.

You can create a presentation that provides the user with information or a presentation that requests information from the user, or even a combination of both. Presentations are a very good way to manipulate the data collected in the attributes of your Fuego Object. For example, one activity in a process may ask the user to order office supplies for a department. However, if the order exceeds a certain amount, it must be approved by a supervisor.

The presentation to the user who is responsible for ordering appears as a regular order form with a *Submit* button at the bottom.

Once submitted, a method can be called to check the total amount ordered and return the amount as a variable. In the process design, a statement is built to check the number returned by the method. If the number is larger than the amount the user has authority to order, logic in the Method causes the instance to flow to a supervisor's queue for approval. A new presentation is displayed to the supervisor (using the same Fuego Object attributes) and includes all information submitted by the employee who ordered the supplies.

Instead of a *Submit* button, the supervisor sees two new buttons — *Approve* and *Reject*.

Presentation structure and its elements

The elements of the presentation structure can be grouped in:

- Containers, any element that can hold something. A presentation container can contain any type of element including the container itself. Presentations are built based on HTML concepts. The container elements of a presentation are:

Container Name	Description
Presentation	The presentation itself.
Table	A Table is the basic structure element a presentation must include. Fuego Object presentations use tables to present the information in a clear and organized way. It can include rows, cells or other tables.
Row	A row in a table.
Cell	The basic element to create a row.
Array	A table with column titles. It can only include rows and cells.
Repeatable Section	Array that allows references to any type of attribute.

- Components,

Component Name	Description
Anchor	The Link component allows users to create internal or external references in the current page.
Button	Buttons are used to perform some action in the presentation.
Check	Check boxes are used to enable or disable one or more features or options from a set, usually within a dialog box. Typically, a check box contains a small box with adjoining text. When an option is selected, a check mark appears in the box.
Combo	A combo box provides a list box with options to select. Selecting an item in the list box displays the selected text in the field.
DateTimePicker	Allows the designer to select and display date and/or time info.
Image	A graphical object.
Interval	Interval component may contain any time dimension from a number of months to a number of microseconds. It stores time offsets, including time-of-day offsets. It does not do time zone correction since the properties of interval variables do not include the date.
Label	Labels allow adding a static value (String) to form components. This value is not editable at presentation runtime.
Multilinetext	Multilinetext components are used to fill in or to display text with multiple lines.

Component Name	Description
Radio	List of possible options of which only one can be selected.
Text	Text fields can be used to request input or display text or numbers. .
Inner Frame	Inner Frames allow embedding already created HTML within a Fuego Object Presentation frame.

The table below shows the relation between a Fuego Object attribute's type and the graphical component used to display it in the presentation.

Attribute Type	Graphical Component
String	Text field, Multi-line textfield, Combo, Radio Button, Inner Frame (Type "referenced")
Int	Text field, Combo, Radio Button
Boolean	Check box, Text field, Combo, Radio Button
Real	Text field, Combo, Radio Button
Decimal	Text field, Combo, Radio Button
Time	Date Time picker
Interval	Interval, Text fields, Combo, Radio Button
Binary	Image
Any	-----
Group	Array, Repeatable section

Each element has properties of different types, *structure*, *format*, *action* and *data*. Each component has its own set of properties. This provides the developers with the ability to design a presentation with a variety of possibilities according to their needs.

Presentation elements properties types

First, let's see the different properties types and which ones can be applied to the presentation elements.

Structure properties always define any presentation element that can be contained, as an image, a button, row, table, etc.

- Components

- *anchor*
- *button*
- *check*
- *combo*
- *datetimepicker*
- *interval*
- *label*
- *multilinetext*
- *radio*
- *text*
- *inner frame*

- Containers

- *table*
- *cell*
- *row*

- *array*
- *repeatable section*

Format properties give style, organize or arrange the presentation elements according to chosen patterns, like font, color, etc.

GENERAL

Property Name	Property type	Description	Valid Values
halignment	String	Horizontal alignment	> 0
valignment	String	Vertical alignment	> 0
text	String	Text content	---
imagelayout	String	Graphical object.	---
display	String	Shows a text.	---
transparent	Boolean	Transparent or not.	TRUE/FALSE
height	String	Height	---
width	String	Width	---
columwidth	String	Table Column Width	---
colqty	int	The size of a Text component. The calculation is based according to the font type size selected for the component. If you set it to ten, then 10 characters will fit in average.	> 0

Property Name	Property type	Description	Valid Values
lineqty	int	Lines quantity	--
cols	int	Quantity of columns in which a radio button is going to be displayed.	> 0
rows	int	Quantity of rows in which a radio button is going to be displayed.	>= 0
cellpadding	int	Cell padding, space between the cell and its internal element.	--
cellspacing	int	Cell spacing, space between the cells of the presentation.	--
hexpand	int	Horizontal Expand.	--
vexpand	int	Vertical Expand.	--
headers	String[]	Array Headers.	--
headerswidth	int[]	Array Headers width.	--
headerfgcolor	String	Array Header Background Color.	
headerbgcolor	String	Array Header Foreground Color.	
headerborderstyle	String	Array Header Border Style.	
headerbordercolor	String	Array Header Border Color.	

Property Name	Property type	Description	Valid Values
headerborderwidth	int	Array Header border width.	
headerfonttype	String	Array Header Font Type.	
headerfontsize	int	Array Header Font Size.	
headerfontface	String	Array Header Font face.	
indexbgcolor	String	Array Index Background Color.	
indexbordercolor	String	Array Index Border Color.	
indexborderstyle	String	Array Index border style.	
indexfgcolor	String	Array Index foreground Color.	
indexfontface	String	Array Index font face.	
indexfonttype	String	Array Index font type.	
indexborderwidth	int	Array Index border width.	
indexfontsize	int	Array Index font size.	
pagingbgcolor	String	Array paging background color.	
pagingbordercolor	String	Array paging border color.	
pagingborderstyle	String	Array paging border style.	
pagingfgcolor	String	Array paging foreground color.	

Property Name	Property type	Description	Valid Values
pagingfontface	String	Array paging Font face.	
pagingfonttype	String	Array paging Font type.	
pagingborderwidth	int	Array paging border width.	
pagingfontsize	int	Array paging Font size.	
border	boolean	Has border?	TRUE/FALSE
cssclass	String	CSS class name	
iscssenabled	boolean	Is CSS customization enabled?	TRUE/FALSE
cssfilename	String	CSS file name	
ismixedcss	boolean	Is CSS customization enabled as mixed?	TRUE/FALSE
pagingcssclass	String	CSS class name for group paging	
indexcssclass	String	CSS class name for group index	
headercssclass	String	CSS class name for group header	

Format Type : *BORDER*

Property Name	Property type	Description	Valid Values
bordercolor	String	Color to apply to a border.	--
borderstyle	String	Style to apply to a border.	--

Property Name	Property type	Description	Valid Values
borderwidth	int	Border width.	--
bottombordercolor	String	Color of the border bottom.	--
bottomborderstyle	String	Color of the border bottom.	--
bottomborderwidth	int	Width of the border bottom.	--
leftbordercolor	String	Color of the left side of the border.	--
leftborderstyle	String	Style of the left side of the border.	--
leftborderwidth	int	Width of the left side of the border.	--
rightbordercolor	String	Color of the right side of the border.	--
rightborderstyle	String	Style of the right side of the border.	--
rightborderwidth	int	Width of the right side of the border.	--
topbordercolor	String	Color of the border top.	--
topborderstyle	String	Style of the border top.	--
topborderwidth	int	Width of the border top.	--

border-width

This is a shorthand property that allows an author to specify 'border-top-width', 'border-right-width', 'border-bottom-width' and 'border-left-width' properties. If one or more of the values are not present, the value for a missing side is taken from the opposite side that is present. If only one value is listed, it applies to all sides.

border-style: This is a shorthand property that allows an author to specify 'border-top-style', 'border-right-style', 'border-bottom-style' and 'border-left-style' properties. If one or more of the values are not present, the value for a missing side is taken from the opposite side that is present. If only one value is listed, it applies to all sides.

- *none*: No border is rendered. This overrides any value of 'border-width', if present.
- *dotted*: The border is rendered as a series of dots.
- *dashed*: The border is rendered as a series of short lines.
- *solid*: Renders a solid line.
- *groove*: Creates the effect of the border being grooved or carved in the rendering surface (A 3-D groove - the opposite of 'ridge'). The groove bevel color is rendered based upon the value of the 'color' property.
- *ridge*: Creates the effect of the border being raised from the rendering surface (A 3-D ridge - the opposite of 'groove'). The ridge bevel color is rendered based upon the value of the 'color' property.
- *inset*: Creates the effect of the border being embedded in the rendering surface (A 3-D inset.) The inset bevel color is rendered based upon the value of the 'color' property. A distinction exists between this value and 'groove'.
- *outset*: Creates the effect of the border coming out of the rendering surface (A 3-D outset - the opposite of 'inset'). The outset bevel

color is rendered based upon the value of the 'color' property. A distinction exists between this value and 'ridge'.

- *double*: A double line drawn on top of the background of the element. The two lines with the space between add up to the value of the 'border-width' property.

border-color

This is a shorthand property that allows an author to specify 'border-top-color', 'border-right-color', 'border-bottom-color' and 'border-left-color' properties. If one or more of the values are not present, the value for a missing side is taken from the opposite side that is present. If only one value is listed, it applies to all sides. If no border-color is specified for an element's border, the value of the 'color' property is used instead.

Note



Border properties are always applied to the frame of the table (the outside border). But each property is only applied to inner cell borders if they do not have a user predefined property.

Format Type : *COLOR*

Property Name	Property type	Description	Valid Values
bgcolor	String	Background color	--
fgcolor	String	Foreground color	--

Format Type : *FONT*

Property Name	Property type	Description	Valid Values
fontface	String	Font face	Platform Independent
fontsize	int	Font size	Platform

Property Name	Property type	Description	Valid Values
			Independent
fonttype	String	Font type	NORMAL, BOLD, UNDERLINE, BOLD_UNDERLINE

Format Type : *ALIGNMENT*

Property Name	Property type	Description	Valid Values
alignment	String	Content alignment	--
valiagment	String	Vertical alignment	--

When you are designing a presentation, FuegoBPM Studio allows you to use every installed font type of your platform.

But the installed font types are different by platform (also by program). This is why some users will not be able to see the presentation correctly. If this is the case, a warning message is shown, so that you can change the font to a more common type. However, the project containing an unknown font can still be published because the presentation takes the font definitions from the server execution platform. In addition, if there is not a font definition when the presentation is shown, the browser replaces it with a default font.

- **Action** properties let you give behavior to a presentation element.

Property Name	Property type	Description	Valid Values
action	String	button's action types	ACTION, CANCEL, REFRESH, RESET, SUBMIT. You use

Property Name	Property type	Description	Valid Values
			ACTION when you want to customize a behavior in a button. The others are the standard actions provided by Fuego for Fuego Objects presentations buttons. See examples in Fuego Objects Examples - Project Example Catalog - Fuego Object Methods - Action methods.
editable	boolean	If the component is editable.	TRUE/FALSE
onchange	String	A method invoked when the content of the component was changed.	A method without arguments and without return value. If the component is inside the array the method invoked needs an int argument (current row value.)
onclick	String	A method invoked when the content is clicked off.	A method without arguments and without return value.

Property Name	Property type	Description	Valid Values
link	String	URL to open	The <i>link</i> property overwrites the <i>onclick</i> property if defined.
editable	boolean	If the component is editable.	TRUE/FALSE
initializationmethod	String	A method invoked to initialize the presentation. This method can be used to manipulate some data before the presentation is opened.	A method without arguments and without return value.
tab index	Int	The position of the input element in the tabbing order. Its default value is 0.	The valid values for this property will be $n \geq 0$. # Elements with <code>tabIndex=0</code> are ordered based on the source # Any element with <code>tabIndex > 0</code> appears before all elements with <code>tabIndex=0</code> # Any Elements with the same <code>tabIndex</code> are ordered based on the default order. The group's elements inherits

Property Name	Property type	Description	Valid Values
			this property from the group itself and it is readonly.

Load once:

This property applies only for multivaluated components. These components are *combo boxes* and *radio buttons*. The *Load once* property becomes available only when the multivaluated component is within a *group* or a *repeatable section* component and its purpose is for optimizing the valid values refresh.

This property must only be set to *true* in those components which valid values are not dependent on other components' valid values. This is because the list of valid values of a component with the *load once* property set to true, is only loaded the first time the presentation is executed.

If you need the component valid values list to be dependant of another presentation component, you would implement it with an *onChange* method and would refresh the valid values list for the second component. If you would set the *load once* property to true in this case, it would not work, as the refresh valid values would not have affect and the list for the second component would remain related to the first selection.

- **Data** properties are used to set values to an element, as the value of an error message, or the name of an attribute to reference from the presentation.

Property Name	Property type	Description	Valid Values
id	String	User defined identification.	--
methodinvocation	String	A method invoked	A method without

Property Name	Property type	Description	Valid Values
		when an ACTION button is press.	arguments and without return value.
required	boolean	If the component is required.	TRUE/FALSE
validationmessage	String	A custom message to use if the attribute validation fault.	--
gifencoded	String	Encoded Image.	--
reference	String	Referenced attribute.	An attribute name.

validationmessage / Validation Error Message

This message is only displayed when the *Check Expression* of an attribute fails. Have in mind, that *check expressions* are only evaluated when the *submit* button of the presentation is executed. So, the error message will be displayed only when the submit button is pressed and the check expression of any attributes fails.

The messages are displayed at the left bottom of the screen, with the following format:

Attribute Name: Message Error (Check expression fail).

Setting Properties to Presentation Elements

Setting presentation properties at definition time

Let's now go through each element and see which properties are defined by default or can be used to give them different aspects and behavior.

<i>Presentation Element</i>	<i>Property Type</i>	<i>Properties Internal name</i>	<i>Property Studio Name</i>
Array Repeatabile Section	&ACTION	editable	Is editable?
		Enabled/Disabled index tab browsing	Default: Disabled
		Event Listener, method invoked	Assign a method to listen group toolbar events
	<i>DATA</i>	id	Name
		reference	Referenced data
	<i>FORMAT</i>	bgcolor	Background Color
		borderwidth	Border Width
		bordercolor	Border Color
		borderstyle	Border Style
		fontsize	Under Font Type
		fontface	Under Font Type
		fonttype	Under Font Type
		cellpadding	Cell Padding
		cellspacing	Cell Spacing
		rows	Row qty.
		headers	Table header
		headerswidth	Table header width
		headerfgcolor	Background Color (Header Properties)
		headerbgcolor	Foreground Color (Header Properties)
		headerborderstyle	Border Style

<i>Presentation Element</i>	<i>Property Type</i>	<i>Properties Internal name</i>	<i>Property Name</i>	<i>Studio</i>
			(Header Properties)	
		headerbordercolor	Border Color (Header Properties)	
		headerborderwidth	Border width (Header Properties)	
		headerfonttype	Name (Header Properties)	
		headerfontsize	Size (Header Properties)	
		headerfontface	Bold - Italic (Header Properties)	
		indexbgcolor	Background Color (Index Properties)	
		indexbordercolor	Border Color (Index Properties)	
		indexborderstyle	Border Style (Index Properties)	
		indexborderwidth	Border width (Index Properties)	
		indexfgcolor	Foreground Color (Index Properties)	
		indexfontface	Bold & Italic (Index Properties)	
		indexfonttype	Name (Index Properties)	
		indexfontsize	Size (Index Properties)	
		pagingbgcolor	Background Color	

<i>Presentation Element</i>	<i>Property Type</i>	<i>Properties Internal name</i>	<i>Property Name</i>	<i>Studio</i>
			(Paging Properties)	
		pagingbordercolor	Border Color (Paging Properties)	
		pagingborderstyle	Border Style (Paging Properties)	
		pagingborderwidth	Border width (Paging Properties)	
		pagingfgcolor	Foreground Color (Paging Properties)	
		pagingfontface	Bold & Italic (Paging Properties)	
		pagingfonttype	Name (Paging Properties)	
		pagingfontsize	Size (Paging Properties)	
		cssclass	CSS class name	
		ismixedcss	Is CSS customization enabled as mixed?	
		pagingcssclass	CSS class name for group paging	
		indexcssclass	CSS class name for group index	
		headercssclass	CSS class name for group header	

<i>Presentation Element</i>	<i>Property Type</i>	<i>Properties Internal name</i>	<i>Property Studio Name</i>
		rowEvenBackground	The background color of even rows.
		rowOddBackground	The background color of odd rows.
		rowEvenCssClass	The css class name of even rows.
		rowOddCssClass	The css class name of even rows.
		header enabled	To enable/disable the component headers
		index enabled	To enable/disable the component indexes
	<i>STRUCTURE</i>	addbutton	--
		removebutton	--
		row	--
Cell	<i>FORMAT</i>	hexpand	Horizontal Expand
		vexpand	Vertical Expand
		cssclass	CSS class name
		ismixedcss	Is CSS customization enabled as mixed?
	<i>FORMAT ALIGNMENT</i>	alignment	Horizontal alignment
		valignment	Vertical Alignment
	<i>FORMAT BORDER</i>	bordercolor	Default value, inherited from

<i>Presentation Element</i>	<i>Property Type</i>	<i>Properties Internal name</i>	<i>Property Name</i>	<i>Studio</i>
			table	
		borderstyle	Default value, inherited from table	
		borderwidth	Default value, inherited from table	
		bottombordercolor	Bottom Border Color	
		bottomborderstyle	Bottom Border Style	
		bottomborderwidth	Bottom Border Width	
		leftbordercolor	Left Border Color	
		leftborderstyle	Left Border Style	
		leftborderwidth	Left Border Width	
		rightbordercolor	Right Border Color	
		rightborderstyle	Right Border Style	
		rightborderwidth	Right Border Width	
		topbordercolor	Top Border Color	
		topborderstyle	Top Border Style	
		topborderwidth	Top Border Width	
	<i>FORMAT COLOR</i>	bgcolor	Background Color	
		fgcolor	Foreground Color	
	<i>FORMAT FONT</i>	fontface	Bold & Italic	
		fontsize	Size	
		fonttype	Name	
	<i>STRUCTURE</i>	array	--	
		button		

<i>Presentation Element</i>	<i>Property Type</i>	<i>Properties Internal name</i>	<i>Property Studio Name</i>
		check	
		combo	
		datetimepicker	
		image	
		interval	
		label	
		multilinetext	
		radio	
		table	
		text	
Row	<i>STRUCTURE</i>	cell	
Table	<i>FORMAT</i>	cellpadding	Cell Padding.
		cellspacing	Cell Spacing.
		width	Width
		columnwidth	Column Width.
		cssclass	CSS class name
		ismixedcss	Is CSS customization enabled as mixed?
	<i>FORMAT ALIGNMENT</i>	alignment	Horizontal Alignment.
	<i>FORMAT BORDER</i>	bordercolor	Default Value.
		borderstyle	Default value.
		borderwidth	Default value.
		bottombordercolor	Bottom Border Color.
		bottomborderstyle	Bottom Border Style.
		bottomborderwidth	Bottom Border Width.

<i>Presentation Element</i>	<i>Property Type</i>	<i>Properties Internal name</i>	<i>Property Studio Name</i>
		leftbordercolor	Left Border Color.
		leftborderstyle	Left Border Style.
		leftborderwidth	Left Border Width.
		rightbordercolor	Right Border Color.
		rightborderstyle	Right Border Style.
		rightborderwidth	Right Border Width.
		topbordercolor	Top Border Color.
		topborderstyle	Top Border Style.
		topborderwidth	Top Border Width.
	<i>FORMAT COLOR</i>	fgcolor	Inherited from Presentation.
	<i>STRUCTURE</i>	row	
Presentation	<i>ACTION</i>	initializationmethod	Initialization Method
	<i>DATA</i>	id	Name
		reference	Reference data
	<i>FORMAT</i>	height	
		width	
		iscssenabled	Is CSS customization enabled?
		cssfilename	CSS file name
		cssclass	CSS class name
		ismixedcss	Is CSS customization enabled as mixed?
	<i>FORMAT COLOR</i>	bgcolor	Background Color

<i>Presentation Element</i>	<i>Property Type</i>	<i>Properties Internal name</i>	<i>Property Studio Name</i>
	<i>STRUCTURE</i>	background	--
		table	--
Background	<i>FORMAT</i>	imagelayout	Image layout (as part of the Presentation Properties.)
		imageposition	Image position (as part of the Presentation Properties.)
		xaxis	X-Axis Position(as part of the Presentation Properties.)
		yaxis	Y-Axis Position (as part of the Presentation Properties.)
	<i>STRUCTURE</i>	image	Encode image (as part of the Presentation Properties.)
Button	<i>ACTION</i>	action	Action
	<i>DATA</i>	id	Name
		methodinvocation	Method invoked.
	<i>FORMAT</i>	display	Display
		transparent	Is transparent
		cssclass	CSS class name
		ismixedcss	Is CSS customization enabled as mixed?

<i>Presentation Element</i>	<i>Property Type</i>	<i>Properties Internal name</i>	<i>Property Studio Name</i>
	<i>FORMAT BORDER</i>	bordercolor	Border Color.
		borderstyle	Border Style.
		borderwidth	Border Width.
	<i>FORMAT COLOR</i>	bgcolor	Background Color.
		fgcolor	Foreground Color.
	<i>FORMAT FONT</i>	fontface	Bold & Italic.
		fontsize	Size
		fonttype	Name
Check	<i>ACTION</i>	editable	Editable?
		onchange	On change invoke
		required	Required?
	<i>DATA</i>	id	Name
		reference	Referenced Data
		validationmessage	Validation Error Message
	<i>FORMAT</i>	cssclass	CSS class name
		ismixedcss	Is CSS customization enabled as mixed?
	<i>FORMAT BORDER</i>	bordercolor	Border Color
		borderstyle	Border Style
		borderwidth	Border Width
	<i>FORMAT COLOR</i>	bgcolor	Background Color
		fgcolor	Foreground Color
	<i>FORMAT FONT</i>	fontface	Bold & Italic
		fontsize	Size
		fonttype	Name
	<i>STRUCTURE</i>	value	Display
Combo	<i>ACTION</i>	editable	Editable

<i>Presentation Element</i>	<i>Property Type</i>	<i>Properties Internal name</i>	<i>Property Studio Name</i>
		onchange	On change invoke.
		required	Required?
	<i>DATA</i>	id	Name
		reference	Referenced Data.
		validationmessage	Validation Error Message.
	<i>FORMAT</i>	transparent	Transparent?
		cssclass	CSS class name
		ismixedcss	Is CSS customization enabled as mixed?
	<i>FORMAT ALIGNMENT</i>	alignment	Horizontal alignment.
	<i>FORMAT BORDER</i>	bordercolor	Default value.
	<i>FORMAT COLOR</i>	bgcolor	Default value.
	<i>FORMAT FONT</i>	fontface	Bold & Italic.
		fontsize	Size
		fonttype	Name
Date Time Picker	<i>ACTION</i>	editable	Editable?
		onchange	On change invoke.
		required	Required?
		DHTMLcalendar	True: Opens a DHTML calendar, False: Opens an HTML calendar
		Enabled/Disabled calendar tab	Default value: Enabled

<i>Presentation Element</i>	<i>Property Type</i>	<i>Properties Internal name</i>	<i>Property Studio Name</i>
		browsing	
	<i>DATA</i>	id	Name
		reference	Referenced Data.
		validationmessage	Validation Error Message.
	<i>FORMAT</i>	cssclass	CSS class name
		ismixedcss	Is CSS customization enabled as mixed?
	<i>FORMAT BORDER</i>	bordercolor	Border Color.
		borderstyle	Border Style.
		borderwidth	Border Width
	<i>FORMAT COLOR</i>	bgcolor	Background Color.
		fgcolor	Foreground Color.
	<i>FORMAT FONT</i>	fontface	Bold & Italic.
		fontsize	Size
		fonttype	Name
	<i>STRUCTURE</i>	value	--
Image	<i>ACTION</i>	onclick	On Click
		link	Link
	<i>DATA</i>	gifencoded	Encoded image.
		id	Name
		reference	Referenced data.
	<i>FORMAT</i>	height	Height
		width	Width
		cssclass	CSS class name
		ismixedcss	Is CSS customization enabled as mixed?

<i>Presentation Element</i>	<i>Property Type</i>	<i>Properties Internal name</i>	<i>Property Studio Name</i>
	<i>FORMAT BORDER</i>	borderstyle	Border Style.
		borderwidth	Border Width.
		bordercolor	Border Color.
Interval	<i>ACTION</i>	editable	Editable?
		onchange	On change invoke
		required	Required?
	<i>DATA</i>	id	Name
		reference	Referenced Data.
		validationmessage	Validation Error Message.
	<i>FORMAT</i>	cssclass	CSS class name
		ismixedcss	Is CSS customization enabled as mixed?
	<i>FORMAT BORDER</i>	bordercolor	Border Color.
		borderstyle	Border Style.
		borderwidth	Border Width.
	<i>FORMAT COLOR</i>	bgcolor	Background Color.
		fgcolor	Foreground Color.
	<i>FORMAT FONT</i>	fontface	Bold & Italic.
		fontsize	Size
		fonttype	Name
	<i>STRUCTURE</i>	value	
Label	<i>ACTION</i>	onclick	On Click
		link	Link
	<i>FORMAT</i>	cssclass	CSS class name
		ismixedcss	Is CSS customization enabled as mixed?

<i>Presentation Element</i>	<i>Property Type</i>	<i>Properties Internal name</i>	<i>Property Studio Name</i>
	<i>DATA</i>	id	Name
	<i>STRUCTURE</i>	value	Display
Multilinetext	<i>ACTION</i>	editable	Editable?
		onchange	On change invoke
		required	Required?
	<i>DATA</i>	id	Name
		reference	Referenced Data.
		validationmessage	Validation Error Message.
	<i>FORMAT</i>	colqty	Columns
		lineqty	Lines.
		cssclass	CSS class name
		ismixedcss	Is CSS customization enabled as mixed?
	<i>FORMAT BORDER</i>	bordercolor	Border Color.
		borderstyle	Border Style.
		borderwidth	Border Width.
	<i>FORMAT COLOR</i>	bgcolor	Background Color.
		fgcolor	Foreground Color.
	<i>FORMAT FONT</i>	fontface	Bold & Italic.
		fontsize	Size
		fonttype	Name
	<i>STRUCTURE</i>	value	
Radio	<i>ACTION</i>	editable	Editable?
		onchange	On change invoke
		required	Required?
	<i>DATA</i>	id	Name
		reference	Referenced Data.

<i>Presentation Element</i>	<i>Property Type</i>	<i>Properties Internal name</i>	<i>Property Studio Name</i>
		validationmessage	Validation Error Message.
	<i>FORMAT</i>	cols	Columns
		display	Display
		rows	Rows
		cssclass	CSS class name
		ismixedcss	Is CSS customization enabled as mixed?
	<i>FORMAT BORDER</i>	bordercolor	Border Color.
		borderstyle	Border Style.
		borderwidth	Border Width.
	<i>FORMAT COLOR</i>	bgcolor	Background Color.
		fgcolor	Foreground Color
	<i>FORMAT FONT</i>	fontface	Bold & Italic.
		fontsize	Size
		fonttype	Name
	<i>STRUCTURE</i>	value	
Text	<i>ACTION</i>	editable	Editable?
		onchange	On change invoke.
		required	Required?
	<i>DATA</i>	id	Name
		reference	Referenced Data.
		validationmessage	Validation Error Message.
	<i>FORMAT</i>	colqty	Columns
		aslabel	Display as Label.
		inputpattern	InputPattern.

<i>Presentation Element</i>	<i>Property Type</i>	<i>Properties Internal name</i>	<i>Property Studio Name</i>
		iperrorcolor	Fail color (Input Pattern.)
		iperrormessage	Fail message (Input Pattern.)
		mask	Mask
		mkerrorcolor	Mask fail color (Mask.)
		mkerrormessage	Mask fail message (Mask.)
		passwordfield	Oassword?
		cssclass	CSS class name
		ismixedcss	Is CSS customization enabled as mixed?
	<i>FORMAT ALIGNMENT</i>	alignment	Horizontal alignment.
	<i>FORMAT BORDER</i>	bordercolor	Border Color.
		borderstyle	Border Style.
		borderwidth	Border Width.
	<i>FORMAT COLOR</i>	bgcolor	Background Color.
		fgcolor	Foreground Color.
	<i>FORMAT FONT</i>	fontface	Bold & Italic.
		fontsize	Size
		fonttype	Name
	<i>STRUCTURE</i>	value	
Inner Frame	<i>DATA</i>	id	Name
		src	HTML source to include
		text type: "Reference" /	Referenced Data (attribute

<i>Presentation Element</i>	<i>Property Type</i>	<i>Properties Internal name</i>	<i>Property Studio Name</i>
		reference	containing the URL).
		text type: "Fixed"/ text	URL.
	<i>FORMAT</i>	width	Width
		height	Height
		scrolling	Scrolling type: Yes, No, Auto
		cssclass	CSS class name
		ismixedcss	Is CSS customization enabled as mixed?
	<i>FORMAT BORDER</i>	bordercolor	Border Color.
		borderstyle	Border Style.
		borderwidth	Border Width.

Some general considerations

Referenced attribute null or not null and Valid Values:

Combo

Depending on the profile of the referenced attribute, the list of valid values will be as follows:

- If the attribute has been defined as not null only the valid values are shown.
- If the attribute has been defined as null the valid values plus an empty value that represents the NULL value are shown.

Radio Button

Depending on the profile of the referenced attribute, the list of valid values will be as follows:

- If the attribute has been defined as not null only the valid values are shown, all buttons in the group are unselected.
- If the attribute has been defined as null only the valid values are shown, but all buttons in the group can be unselected. To unselect a valid value means the NULL value is selected.

Array and Repeatable section

The Array and repeatable section widgets provide a number of actions to be performed over them (add rows, delete rows, etc). You can intercept these events after they are processed by Fuego and perform some post processing. For example, you could be interested in the ADD event and define an even listener method named "processEvent(GroupEvent)" to the widget. Then, every time an event occurs (the user adds or deletes a row) in the widget, the method will be invoked. In this method you could take any desired action (log the event for example or send an email). Note that the argument received is always of the predefined type `Fuego.Util.GroupEvent` and it will contain the necessary information to identify what event took place. It contains:

- `id`: represents the type of event (eg: `GroupEvent.ADD`).
- `indexes`: An array that represents the selected indexes (for ADD and REMOVE_LAST events its value is null).

The listener method is called after the desired action is performed, therefore the datamodel is updated when the method is executed.

Check `Fuego.GroupEvent` component Documentation and Use Case

example within the FuegoBPM Studio to learn more about it. The project example *CompGroupEvent* under the FuegoBPM Studio sample directory contains an example of usage.

Methods to set and get presentation properties at runtime

You could set and get presentation properties at runtime by using the standard methods provided for presentable Fuego Objects. For further information about the standard methods provided, please refer to the **Standard Methods in Presentable Fuego Objects** section in the Fuego Objects Methods help page.

Properties referencing methods when the Fuego Object is implementing behavior inheritance

If an inherited method is redefined in the Fuego Object, only one method will be shown in the method selection combo of those presentation component properties that reference Fuego Object methods.

If the Fuego Object has overwritten the method, only the one from the Fuego Object will be shown, if the Fuego Object has not overwritten the method from the delegate, then the inherited method will be shown.

Only one method will be shown and it will invoke the closest one to the object in the inheritance chain.

Continue learning about Fuego Objects presentations in the Adding Presentations to Fuego Objects and Designing a Form topics.

Adding Presentations to Fuego Objects

Fuego Object presentations appear as HTML forms to the user through Work Portal. Several steps can be taken to customize the appearance of the presentation.

Fuego Objects HTML Presentation run using JavaScript remote scripting. Remote scripting is available for the Internet Explorer, Mozilla and Netscape browsers. The usage, or not, of these

technologies changes the way in which the HTML presentation is refreshed and how the *back* button behaves. Without using remote scripting, as in any HTML page, the back button goes to the previous page. When the implementation uses remote scripting, there are no new pages in every action. This means that *back* always works on the same page. The back button returns to the page from where the task was executed. Users are able to turn off this technology from their Work Portal Options. Others browser, like Konqueror, Opera, etc., always work using reposting.

Note

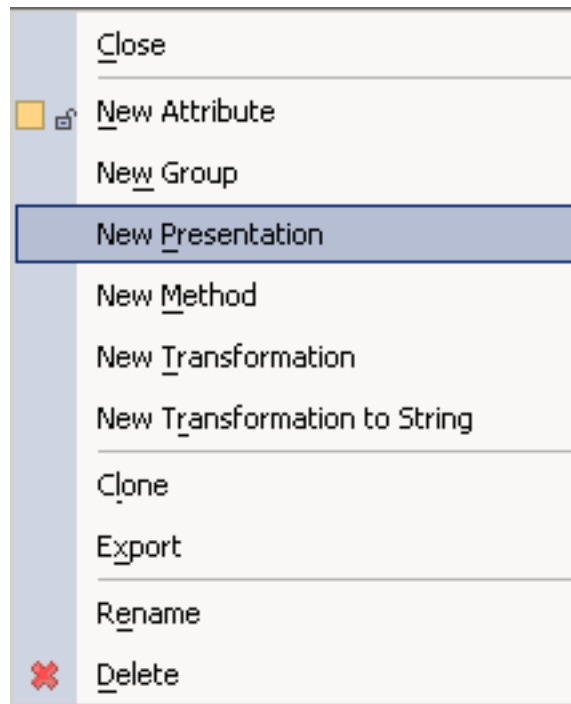


Every time a Fuego Object Presentation is used in HTML, it must be designed in compliance with constraints imposed by the Web.

First Steps to Add a Presentation

To add a presentation:

1. Right-click on the Fuego Object in the left panel of the Project Catalog and select **New Presentation** from the shortcut menu.



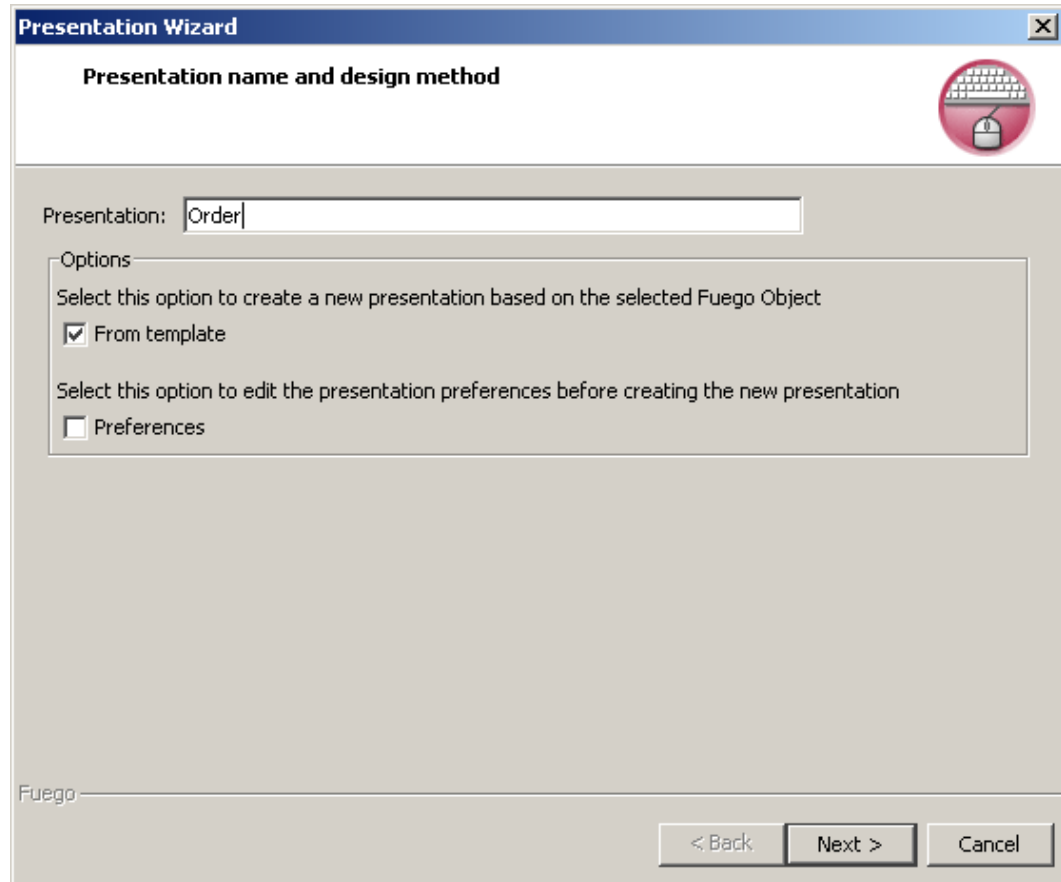
2. The first step on presentation creation wizard opens. Enter a name for the presentation in the **Presentation name** field.
3. Continue with one of the following options:
 - a. Create a presentation using the template.
 - b. Create a presentation using template preferences.
 - c. Creating a presentation without a template.

Create a Presentation Using the Template

A template is available to help you quickly add a presentation to a Fuego Object.

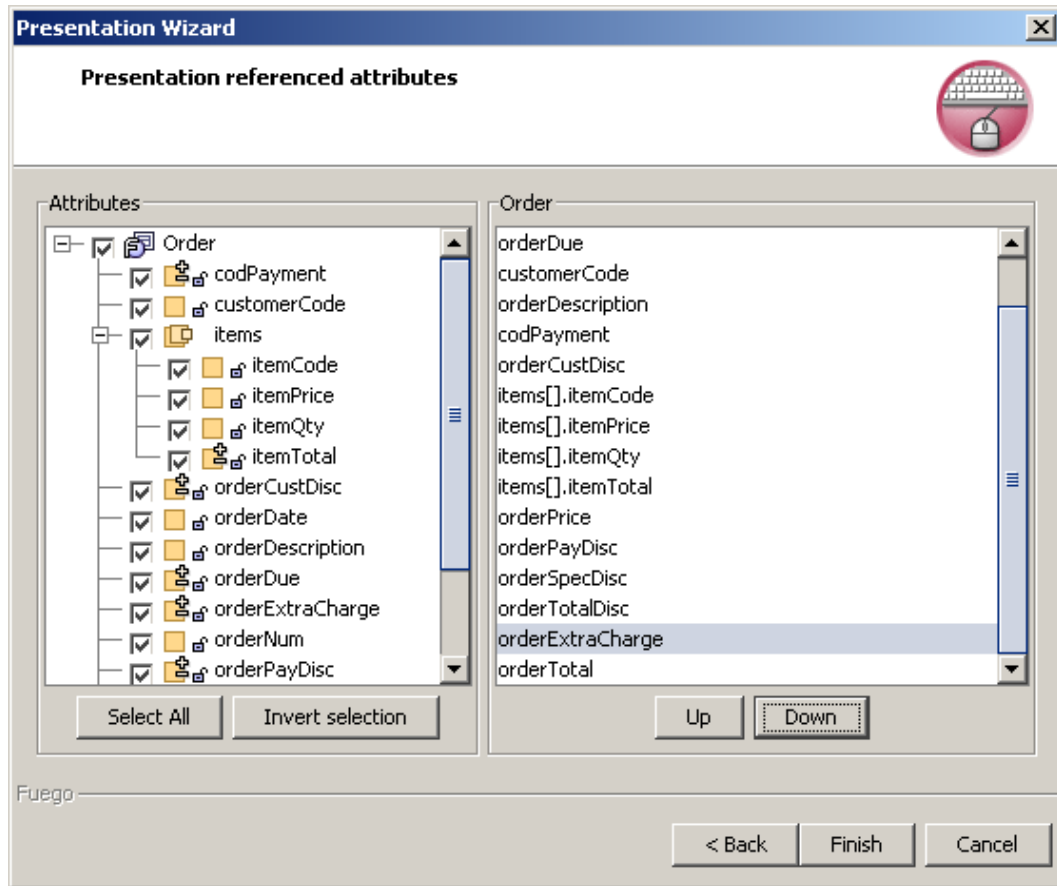
To create a presentation using the template:

1. After the first steps of creation, select the **From template** check box.



The screenshot shows a 'Presentation Wizard' dialog box with a blue title bar. The main area is titled 'Presentation name and design method' and features a red circular icon with a keyboard and mouse. Below the title, there is a text field labeled 'Presentation:' containing the word 'Order'. Underneath, a section titled 'Options' contains two instructions: 'Select this option to create a new presentation based on the selected Fuego Object' and 'Select this option to edit the presentation preferences before creating the new presentation'. The first instruction is followed by a checked checkbox labeled 'From template', and the second is followed by an unchecked checkbox labeled 'Preferences'. At the bottom left, the word 'Fuego' is displayed next to a horizontal line. At the bottom right, there are three buttons: '< Back', 'Next >', and 'Cancel'.

2. Click **OK**. You are prompted to select the Fuego Object attributes that the presentation will reference.



3. You can expand the Fuego Object and see its attributes. Select all of them by clicking on the Fuego Object check box or by clicking the **Select All** button. Deselect those attributes you do not want to include. Invert the selection if it is quicker by clicking the **Invert selection** button. Reorder the attributes in the presentation by clicking on the **Up** and **Down** buttons. When you move the Fuego Object attributes up and down, keep in mind that the presentation wizard will not allow you to separately move a group of inner object attributes. Attributes that belong to these objects within a Fuego Object are always kept together. The presentation creation wizard has a special behavior when a fuego object has nested references (eg: Fuego Object A has an attribute with type Fuego Object B and Fuego Object B has an attribute with type Fuego Object A). The attribute selection tree, on the left side of the wizard, displays

the first depth level attributes. When you click the *Select all* button, only those top-level attributes are selected. If you want to include attributes of another level, browse the tree using the "+" sign at the left side of the complex attributes (groups or Fuego Objects).

4. Click **Next** to finish.

When the presentation is created using the template, components are defined as *required* when they refer to Fuego Object attributes set as *primary key*.

The attribute type default length is used to set the field length in the presentation.

You can now edit the presentation. See **Editing a presentation** for further information.

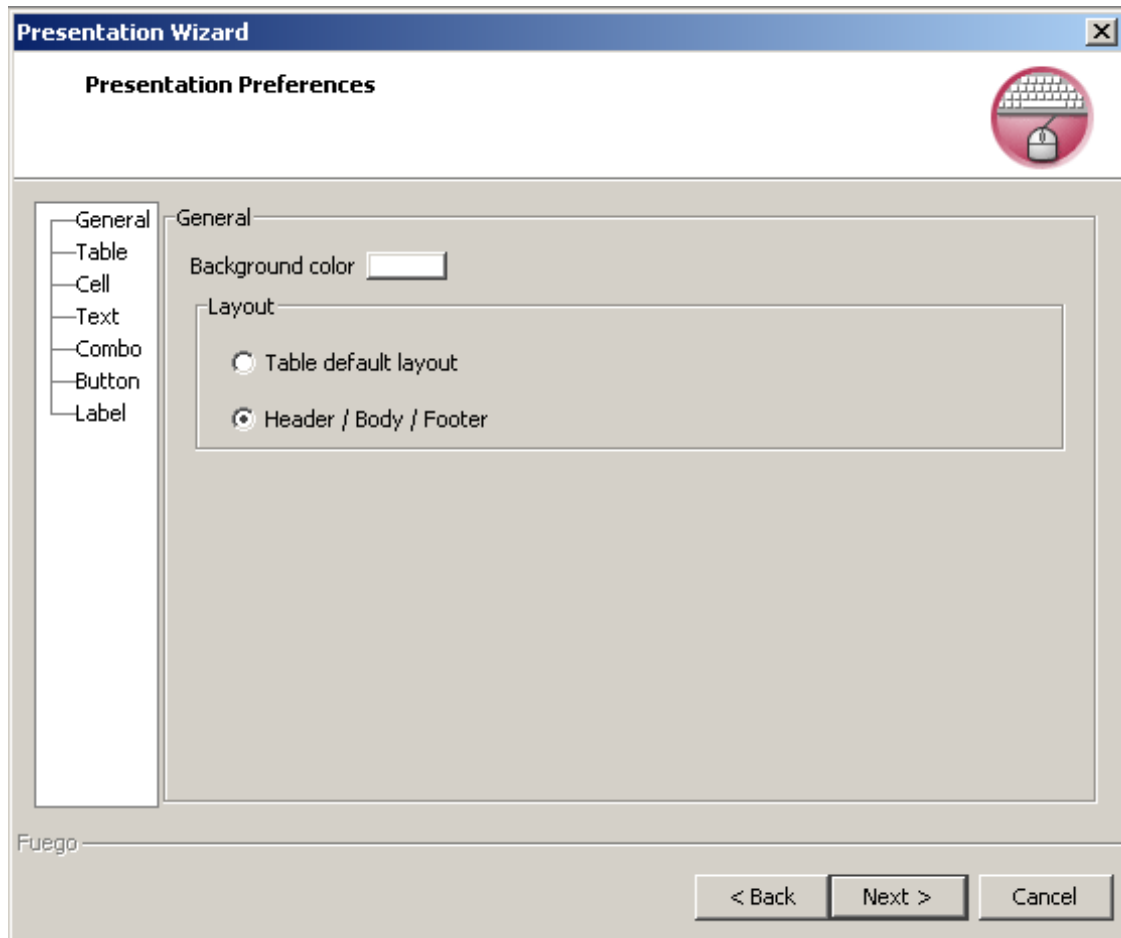
Create a Presentation Using Template Preferences

Using template preferences is the most time-saving choice when creating a presentation. Before the presentation is generated, you can predefine all of your preferences.

To create a presentation using template preferences:

After the first steps of creation, select both the **From template** and the **Preferences** check boxes. The **Preferences** option is not available if you are not generating the presentation using the template.

Click **Next**. The *Presentation Preferences* step of the wizard is displayed.



The Presentation Preferences screen is divided into two parts. On the left, is the list of the presentation elements to which you can select preferences.

1. **General** enables you to:
 - a. Configure the background color of the presentation. The default color is white. Change the color by clicking on the rectangle button next to **Presentation background**. The Choose color dialog box appears. Click the appropriate color and then click the **OK** button.
 - b. Select the Layout used to create the presentation. You may chose to create it as a basic *Table* in HTML or divide it into

three sections: *Header/Body/Footer*. The *footer* section will contain the default buttons created: *submit*, *refresh* and *cancel*. The *body* section will contain the attributes you have chosen to reference from the presentation.

2. **Table** enables you to change the padding and spacing of the table cells. The default padding is 7 and the default spacing is 0, which are generally sufficient for most presentations.
3. **Cell** enables you to change the way in which information is displayed in a cell.
 - a. *Font type* is the style in which any text appears. The choices are **Bold**, *Italic*, **BoldItalic** and Regular. Regular is the default option.
 - b. *Font face* is the font in which any text appears on the presentation. The drop-down lists all fonts available in your machine. Note that your user may not have any special fonts loaded on their machine. In this case, any unavailable fonts are defaulted to the user's default font.
 - c. *Font size* is the size of the text on the presentation. Adjust accordingly. If you leave the size 0, the text appears in the users' default size.
 - d. *Alignment* is where any text appears in the cell. The choices are:
 - i. center, which aligns text in the center of the cell. Center is the default choice.
 - ii. left, which aligns text to the left side of the cell.
 - iii. right, which aligns text to the right side of the cell.

- e. *Vertical alignment* is where any text appears in the cell. The choices are:
 - i. center, which aligns text in the center of the cell. Center is the default choice.
 - ii. top, which aligns text to the top of the cell.
 - iii. bottom, which aligns text to the bottom of the cell.
- 4. **text** enables you to change the way in which the text appears on the presentation.
 - a. Font Type is the style in which any text appears. The choices are Bold, Italic, Bold_Italic and Regular. The default is Regular.
 - b. Font size is the size of the font on the presentation. The default size is 10.
 - c. *Horizontal Alignment* is where the text appears in the cell or form field. *Left* by default. The choices are:
 - i. center, which aligns text in the center of the cell. The default option is Center.
 - ii. left, which aligns the text to the left side of the cell.
 - iii. right, which aligns text to the right side of the cell.
 - d. *Horizontal Alignment (numeric attributes)* is where the text appears in the cell or form field. *Right* by default. The choices are:
 - i. center, which aligns text in the center of the cell. Center

is the default option.

- ii. left, which aligns the text to the left side of the cell.
- iii. right, which aligns text to the right side of the cell.

e. *Alignment within cell* aligns text in form fields. The choices are:

- i. center, which aligns the text in the center of the form field.
- ii. left, which aligns the text to the left side of the form field. Left is the default alignment.
- iii. right, which aligns the text to the right side of the form field.

f. *Maximum Column count* is the size of any text field. This number defaults to 20. It should be increased depending upon what the user is entering. If the "Maximum Column count" is greater than the length of the attribute, then the length of the attribute is used.

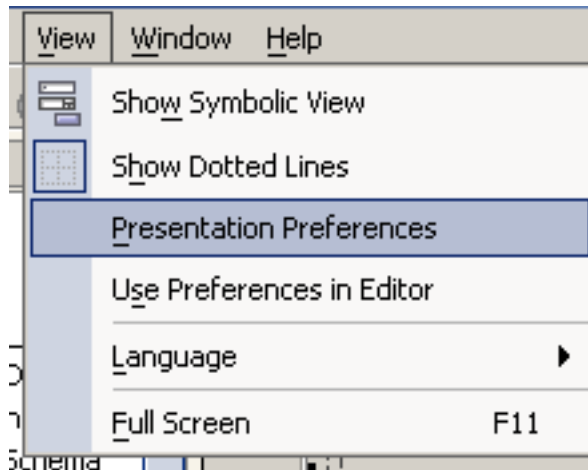
g. *Background color* is the background color of any form field in the presentation. The default color is white. Change the color by clicking on the rectangle next to Background color. The Choose color dialog box is displayed. Choose the correct color and click the Ok button.

h. *Foreground color* is the color in which any text entered by the user appears in the fields. The default color is black. Change the color by clicking on the rectangle next to Foreground color. The Choose color dialog box is displayed. Choose the correct color and click the Ok button.

5. **Combo** aligns text in a combo drop-down menu. The choices are:
 - a. *Center* aligns the text in the center of the drop-down.
 - b. *Left* aligns the text to the left side of the drop-down. Left is the default alignment.
 - c. *Right* aligns the text to the right side of the drop-down.
6. **Button** tab enables you to add buttons to the presentation.
 - a. Selecting the *Transparent* check box makes the color of the button transparent. It will display the same color as the presentation background. If you disable this check box, the button color defaults to gray.
 - b. *Include buttons* allows you to choose which buttons to display on your presentation. Enabling the check box next to the button name adds the button to the presentation. You can change the name displayed on the button by entering the new name in the box next to the button name as shown below. The list of texts corresponding to every button you choose to add to the presentation is at the bottom of the Button panel. You can change the order in which the buttons are displayed by selecting the button to be changed and clicking on the up and down arrows on the right side.
7. **Label** allows you to configure the way the labels display on your presentation.
 - a. *Alignment* within cell determines the alignment of the button within the cell. The choices are:
 - i. center, which aligns the label in the center of the cell.

- ii. left, which aligns the label to the left side of the cell. Left is the default choice.
 - iii. right, which aligns the label to the right side of the cell.
-
- b. *Capitalize first letter?* capitalizes the first letter of every label name on the presentation. Label names are derived from attribute names. Checked by default. For example, if the attribute name is phone_number, it appears as Phone_number on the presentation or Phone number if you select the *Separate names when underscore is found* check box.
 - c. *Separate names when underscore is found?* If you created an attribute name using an underscore between the different parts of the name, by enabling this check box the parts are separated into two distinct names. It is checked by default. For example, if the attribute name is first_name, it appears as first name on the presentation label.
 - d. *Capitalize each word?*, capitalizes the first letter of each word in an attribute name. You must have created your attribute name using an underscore to separate words and also select the *Separate names when underscore is found* check box. Checked by default. For example, if the attribute name is last_name, it appears as Last Name on the presentation label.

You can define the presentation preferences through the main menu *View, Presentation Designer, Presentation Preferences* option.



Creating a Presentation Without a Template

Creating a presentation without using the template is the most time-consuming way to create a presentation. However, sometimes you may want to highly customize a presentation and this choice is the way to accomplish such customizations.

Each presentation is composed of a table with rows and columns. Each cell in a row/column can contain only one object. For example, if you need a field and a label for the field to display on the presentation, you need one row and two columns as shown below. The label appears in the first column and the field appears in the second column. Columns and rows are entered in steps 5-6 below.

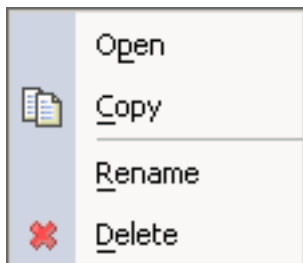
To create a presentation without a template:

1. Right-click on the Fuego Object in the left panel of the Catalog and select **New Presentation** from the shortcut menu.
2. Enter a name in the Presentation name field in the first step screen of the wizard to create the presentation.
3. Disable the **From template** check box (no check mark should be displayed.)
4. Click the **OK** button.

5. Enter the correct number of rows and columns for the table in the **Rows** and **Columns** fields. You can add more rows and columns as you are developing the presentation.
6. The default color for the presentation background is gray. To change the color, click the rectangle in the **Color** field. The Insert table color chooser dialog box is displayed.
7. To change the background color of the presentation, double-click the color block in the Color field. Choose the appropriate color or create one of your own using the HSB and RSB tabs. Click the **OK** button to close the color dialog box.
8. Click the **OK** button. The presentation is displayed.

Handling Fuego Object's Presentations

A created presentation can be renamed, copied, deleted and opened for edition from the Fuego Object structure flap by right-clicking on its name. Choose the action you want to execute from the shortcut menu that opens.



Open a Presentation

To open a presentation:

- Right-click on the presentation name in the Fuego Object structure panel and select **Open** from the shortcut menu, or
- Double-click on the presentation name in the Fuego Object

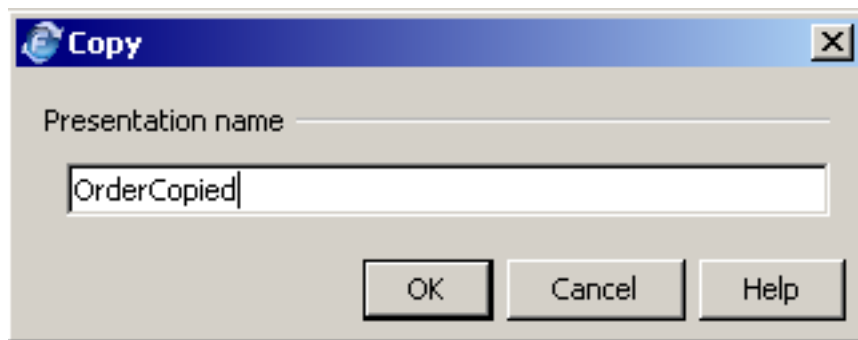
structure panel.

The design panel is opened in the central panel of the FuegoBPM Studio containing the selected presentation.

Copy a presentation

To copy a presentation, right-click on the presentation name in the Fuego Object structure panel and select the **Copy** option from the popup menu.

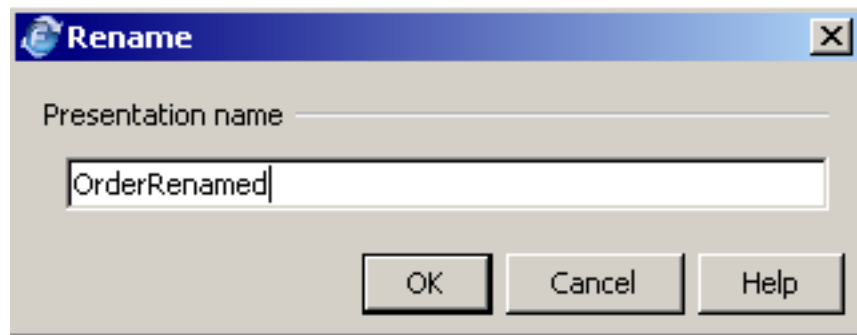
A dialog is displayed so that you can input the name of the new presentation.



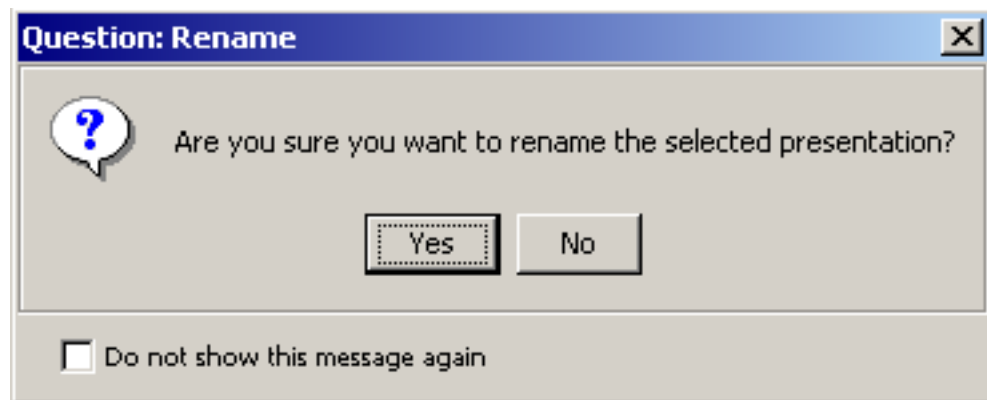
Rename a Presentation

To rename a presentation:

1. Right-click on the presentation name in the Fuego Object structure panel and select **Rename** from the popup menu.
2. A dialog is displayed so that you can input the new name for the presentation. Click **OK** to continue.



3. A dialog is opened asking for a confirmation for the action.



Delete a Presentation

To delete a presentation:

- Right-click on the presentation name in the Fuego Object structure panel and select **Delete** from the popup menu, or
- Select the presentation and press the **delete** key in your keyboard.

See Designing a Form for information on adding the appropriate fields to your presentation.

Designing a Form

Creating a presentation for a Fuego Object provides you with the ability to show how attributes function together in a visual form that can be displayed in Work Portal. A presentation is a quick and easy way to display information contained in a Fuego Object.

Designing a presentation consist basically in defining the structure of it, tables, rows, cells and then adding the components to represent the Fuego Object data structure. A WYSIWYG Designer Panel is provided to do it in an easy and guided way.

Using the Presentation Designer Panel

Defining the Presentation Structure

Adding Components to the Presentation

To take into account

What Makes a Page Refresh?

The actions that cause a refresh of the page are:

- refresh
- reset
- on change invoke
- on click
- action defined in a button
- actions on an array field like add or remove

Customizing Presentations Using Cascading Style Sheets

It is possible to customize presentation appearances by using CSS style sheets as FuegoBPM is REC-CSS1-19990111 (<http://www.w3.org/TR/REC-CSS1.html>) compliant.

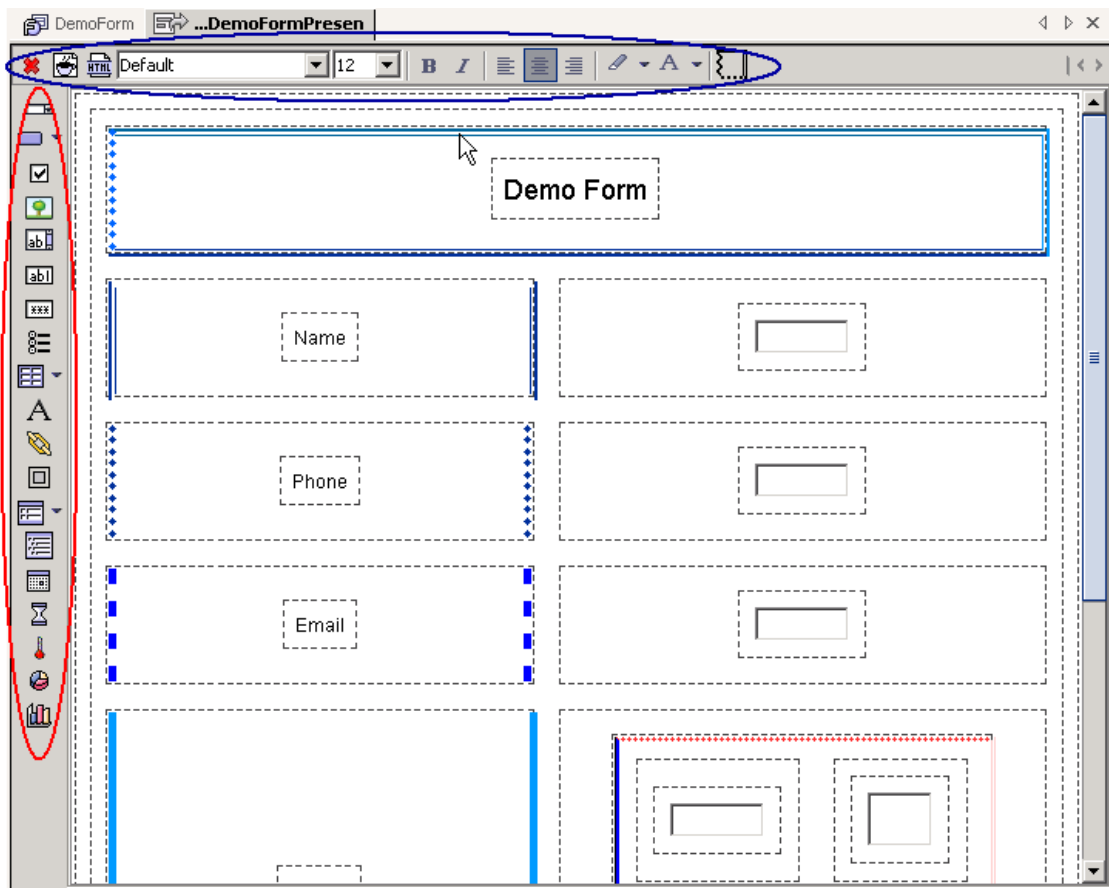
Properties are available for each container and component presentation type when the CSS customization is enable. Please refer to the topic Customizing Presentations with CSS for the details on how to implement it.

Using the Presentation Designer Panel

















Using the Presentation Designer Panel

Toolbars and Layout Panel

Using the Presentation Designer Panel makes it easier to build a WYSIWYG presentation. The image below shows the *Layout panel* where the presentation can be seen while designing it and two toolbars, the *Components bar* on the left and the *Editing bar* at the top of the panel.











- *Components Bar*, located on the left of the panel, has an icon for each component that can be added to a cell.

Component	Icon
Combo	
Anchor	
Button	
Check Box	
Image	
Multiline Text	
Text	
Password	
Radio Button	
Table	
Label	
Array	
Repeatable Section	
Date Time Picker	
Interval	
Inner Frame	

To add a component to a cell, select the empty cell in the presentation layout and click on the component icon you want to add. For detailed information about components, please refer to the section *Adding Components to the Presentation*.

- *Editing Bar*, located at the top of the panel, has an icon for editing

action that can be performed on a selected object in the layout. Editing options will be enabled according to the properties that can be set to the layout object you have selected. For detailed information about properties, components and structure elements please refer to the sections: *Defining the Structure* and *Adding Components to the Presentation*.

Action	Icon
Delete	
Java Preview	
HTML Preview	
Font Type	Combo box with available Font types
Font Size	Combo box with Font sizes
Bold	B
Italic	<i>I</i>
Left Align	
Center Align	
Right Align	
Background	
Foreground	




- *Layout Panel*, you can see the presentation design graphically. Table, rows, cells and components are displayed in the layout

panel. You can edit it by changing the fields order, adding properties to the components or to the structure containers.

- Shortcut Menus from the *Layout Panel*, different menu options are displayed when right-clicking on the layout. It depends on which object of the presentation you have selected at that moment. Menu options displayed for an object in the layout are the same as those displayed when you are working in the presentation structure frame. For example, if you select:
 - a *component*, the menu will display action options for the *Table*, *Row* and *Cell* where the component is contained. You will be able to change its *Id* or *Clear* it.
 - a *cell*, the menu will display action options for the *Table*, *Row* where the cell is contained. You will be able to *move* the cell to the right or left, *replace* the component held by the cell for another one, *split* the cell or *Clear* it.
 - For further and detailed actions for each type of presentation element, please refer to sections *Defining the Structure* and *Adding Components to the Presentation*.
- To change the field order, move the cell to the right or left if the new position is in the same row, or just drag and drop the component you want to move from its position to the empty cell it will contain.

Dash Board Components

Fuego Objects and their presentation have the capability to design a monitoring Dash Board. To do so, the three components needed are provided:

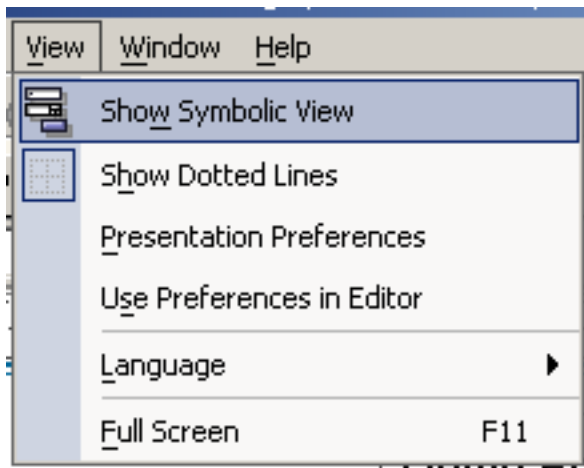
- *Gauge* 
- *Pie* 
- *Chart* 

See more about creating a dashboard using Fuego Objects and their presentations in the **FuegoBPM Dash Board** section.

Changing the Presentation Designer Display

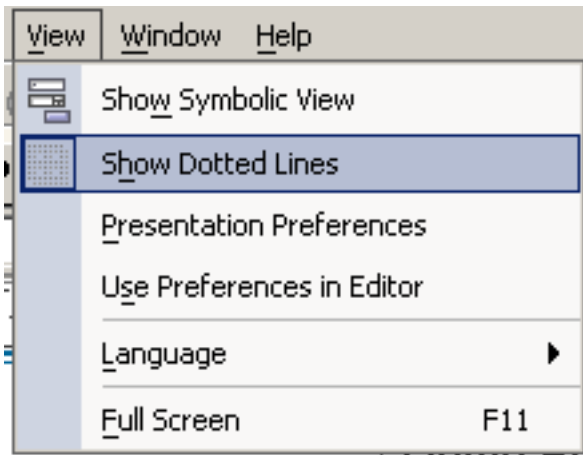
You may change the way the layout is displayed by setting the options in the *View* main menu, *Presentation Designer*:

- Show symbolic view, if selected, it displays the components within the presentation with the icon that represents it instead of its real aspect.



- Show dotted lines, if selected, they delimit each element of the presentation by dotted lines. It is very useful during design to have this option selected as it helps to visualize the elements

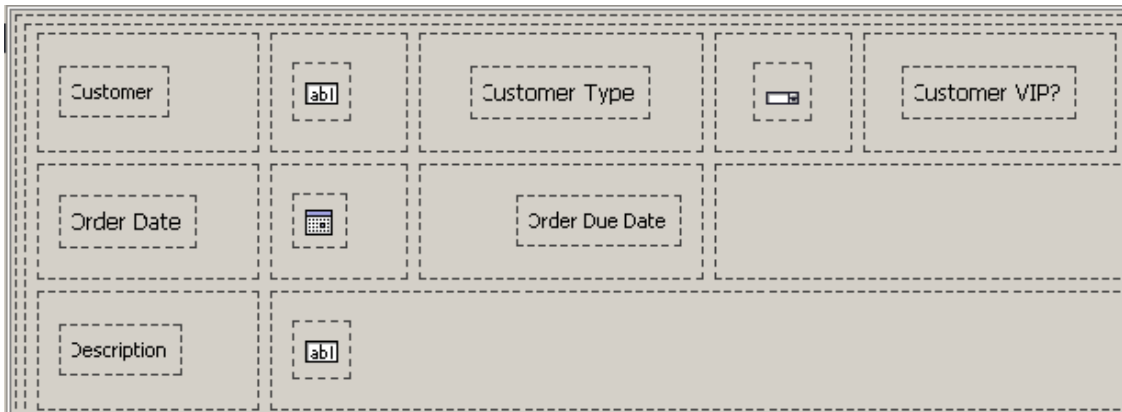
location in the presentation.



Any combination of the above is allowed.

Examples:

- Symbolic View & Dotted Lines



- No Symbolic View & No Dotted Lines.

Customer

Customer Type

Order Date

Order Due Date

Description

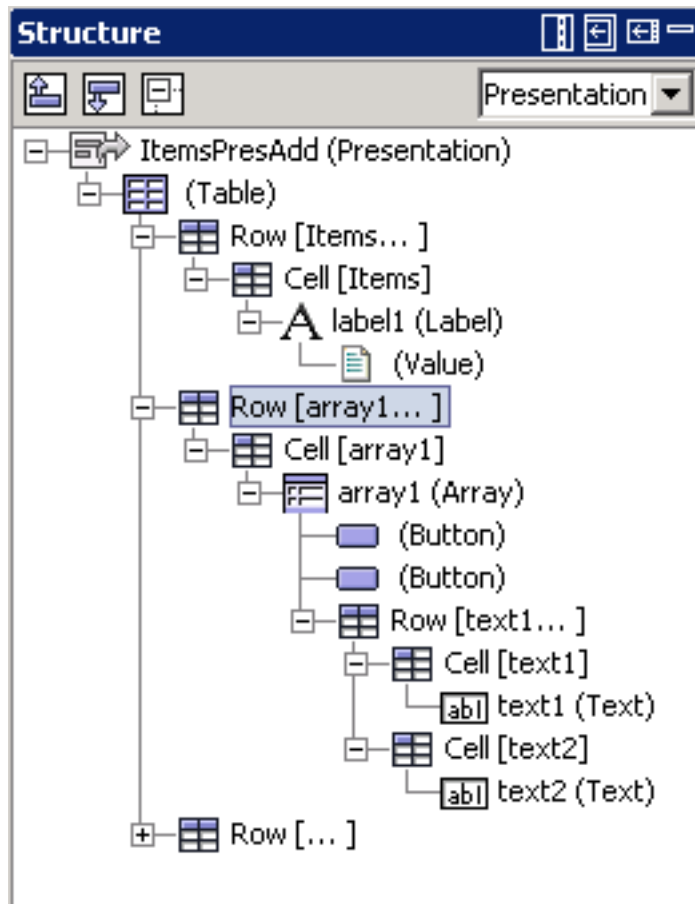
Defining the Presentation Structure

The group of presentation elements used to define their structure are the *containers*.

Presentation

When you change the category in the Structure frame to Presentation, the presentation structure is displayed in the popup menu on the right top corner. Notice that the root element of this structure tree is the presentation. The name you gave it at creation time is shown as a label for the entry. See *Presentation structure and its elements* for further details.

Under the presentation root entry in the structure tree, you will see all the elements that are part of the presentation you are editing.



What can I do at the Presentation Level?

There are no available actions at presentation level. You can only set one of their properties, *Initialization method* and the ones related to its background.

See the section **Handling Fuego Object's presentations** in the Adding presentations topic help to learn what general actions can be done on a presentation, like *rename*, *copy*, etc.

Presentation Properties

- Name: name given at presentation time,
- Referenced Data: name of the Fuego Object to which the presentation belongs,

- Initialization Method: method used to initialize the presentation. This is the only property that can be modified to a presentation,
- CSS support: indicates if the presentation is customized using CSS. For further information take a look to Customizing Presentation with Cascading Style Sheets (CSS)

Presentation properties include the definition for the *Background* and *Error* settings.

Background

The *background* may be considered as an adornment of the presentation and it represents the background layout. The presentation has only one. It can be present or not. It is constructed by default at presentation creation time and it is composed by an *image*.

Background Properties

1. Background Color
2. Encoded Image
3. Image Layout, options:
 - a. Repeat
 - b. Repeat in X
 - c. Repeat in Y
 - d. No-Repeat
4. Image Position, options:
 - a. Scroll

b. Fixed

5. X-Axis Position, options:

a. Left

b. Center

c. Right

6. Y-Axis Position, options:

a. Top

b. Center

c. Bottom

Image Position

This value indicates whether the background image will move when the browser window is scrolled to view text outside the current display area.

- scroll: Background image moves when the browser window is scrolled to view text outside the current display area.
- fixed: Background image does not move when the browser window is scrolled to view text outside the current display area.

Note



If you are using the Internet Explorer Browser this will not take effect.

It renders the image only in the visible container.

Encoded Image

This value indicates the URL source for the graphic. If this property is present, *Image layout*, *Image Position* and *X/Y-Axis positions* properties can also be specified. Transparent portions of the encoded image will assume the color value of any background-color specified.

Image Layout

The position of the image depends on the combination of the *Image Layout* and *X/Y-Axis position* properties.

- Repeat: it repeats the image in every row/column of the presentation.
- Repeat in X: it takes into account the setting of the property Y-Axis position. It repeats the image in every column at the bottom, top or center.
- Repeat in Y: it takes into account the setting of the property X-Axis position. It repeats the image in every row at the left, center or right.

Background Examples

- Image Layout = repeat

Demo Form

Name

Phone

Email

Address

- Image Layout = repeat-y
- X-Axis position = right

Demo Form

Name

Phone

Email

Address

- Image Layout = repeat-x
- Y-Axis position = center

Demo Form

Name

Phone

EGOFUEGOFUE

Email

Address

- Image Layout = no-repeat
- Y-Axis position = center
- X-Axis position = center

Demo Form

Name

Phone

FUEGO

Email

Address

submit

submit

- Image Layout = no-repeat
- Y-Axis position = top
- X-Axis position = left

FUEGO

Demo Form

Name

Phone

Email

Address

submit

submit

Error

At presentation level the error messages can be formatted. Properties to configure it are:

- Background color,
- Foreground color,
- Font size,
- Font type.
- Pop-up Client Errors, this property is used to show the client error in a popup window or in the error area

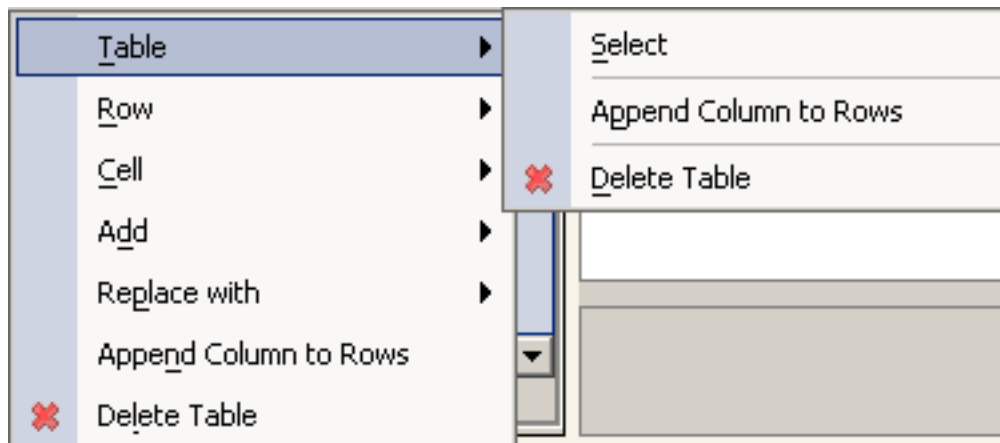
- True: The client errors are shown in a popup window,
- False: The client errors are shown in the error Area

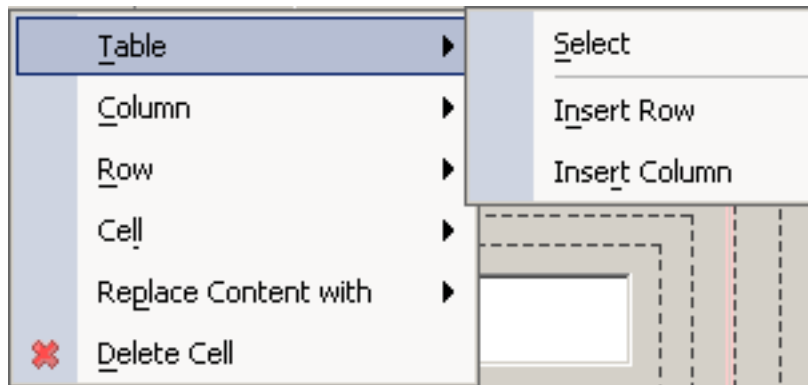
Table

The basic element in the presentation structure is a *table*. This table will contain the rest of the elements that formed the presentation structure; other tables are included.

What can I do to a Table?

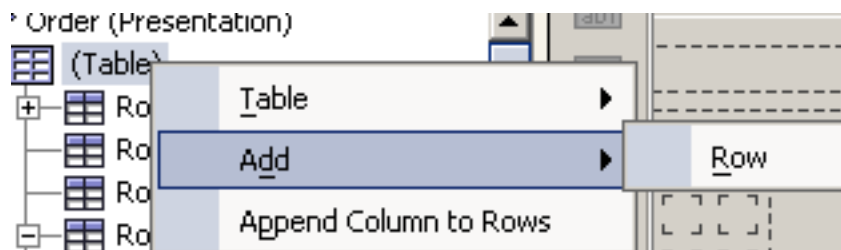
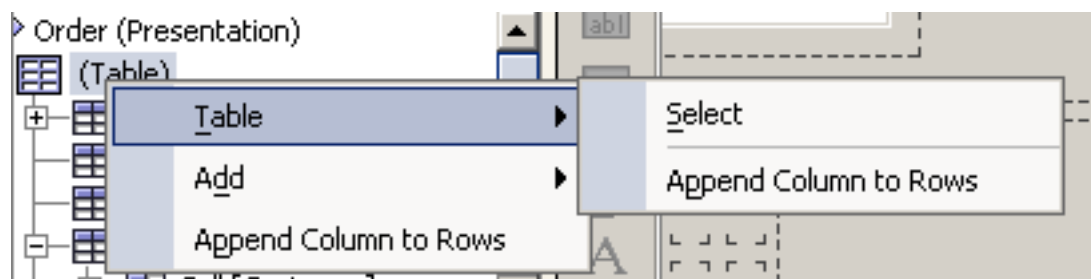
By right-clicking from the table entry in the structure tree or from the design panel, a menu is opened with the following actions available:





1. *Select*: it marks the table as selected in the presentation layout.
2. *Append Column to Rows*: appends cells, one by one, to each table row on the right.
3. *Add Row*: adds a row to the table at the bottom.
4. *Delete Table*: removes the table from the presentation.

If the selected table is the main table of the presentation, then the options menu is restricted to:



A row is added or inserted with the maximum quantity of expanded cells. For example, if you have:

1st row: cell expanded to 1 + cell expanded to 3

2nd row: cell expanded to 2 + cell expanded to 1

the new row will contain **4** cells.

Adding Tables to the Presentation

Tables can be added only to cells that are not part of an array container. See the **Cell** section below for further and detailed information.

Table Properties





- Width,
- Column width, according with the HTML table model the width must be set to the columns. This property is only supported by HTML presentations. SWING presentations will not show any difference. This property accepts as many width values as columns the table has. *columns = cell * colspan (default value = 1) + cell * colspan + ... + n cells of a row.*
- Horizontal Alignment,
- Cell Padding, space between the cell and its internal element,
- Cell Spacing, space between the cells of the presentation,
- Border properties, Style/Width/Color for:
 - Top Border,
 - Bottom Border,
 - Left Border,

- Right Border.

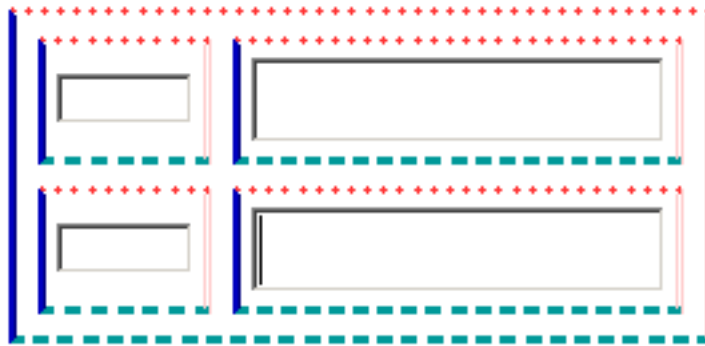
Example

Setting borders and Cell Spacing & Padding.

With the following properties:

Properties		
▼ Properties		
Width		
Column Width	//	
▼ Layout		
Cell Padding		4
Cell Spacing		8
▼ Border		
[-] Top Border	Dotted,3	
Border Style	Dotted	
Border Width		3
Border Color	 [r=255, q=51, b=51]	
[-] Bottom Border	Dashed,3	
Border Style	Dashed	
Border Width		3
Border Color	 [r=0, q=153, b=153]	
[-] Left Border	Ridge,3	
Border Style	Ridge	
Border Width		3
Border Color	 [r=0, q=0, b=255]	
[-] Right Border	Double,3	
Border Style	Double	
Border Width		3
Border Color	 [r=255, q=204, b=204]	

the table will look like:

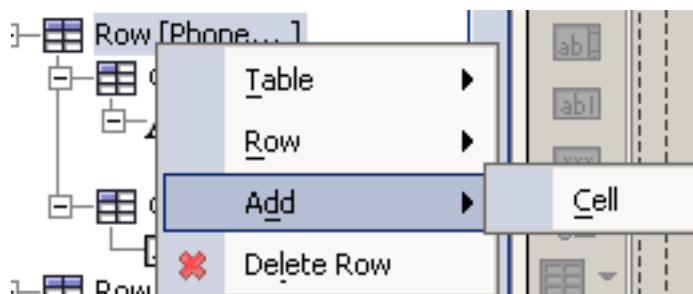
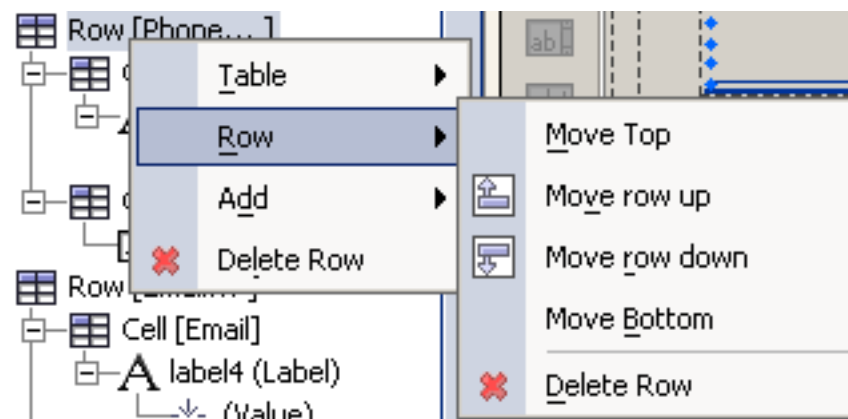


Row

Rows are each line into which the presentation is divided. Each row is split into N columns and it contains cells.

What can I do to a Row?

The following is displayed when right-clicking on a row in the structure tree or the design panel.



1. Move Top: moves the row to the first place in the presentation.
2. Move row down: moves the rows down in the presentation layout.
3. Move row up: moves the rows up in the presentation layout.
4. Move Bottom: moves the row to the last place in the presentation.
5. Add Cell: adds a cell to the row.
6. Delete Row: removes the row from the presentation layout.

A row can be deleted by selecting all the cells within it in the presentation layout by right-clicking and selecting the option that is opened (*Delete selected items.*)

Row Properties

Rows do not have any related property.

Adding Rows to the Presentation

A row can be added by right-clicking on any element of the presentation:

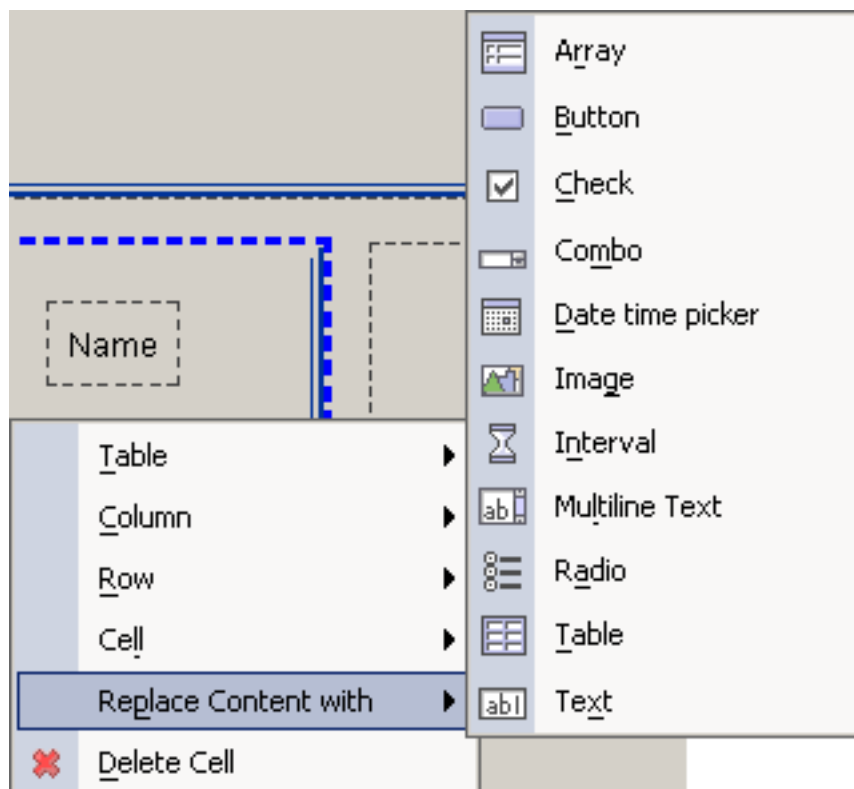
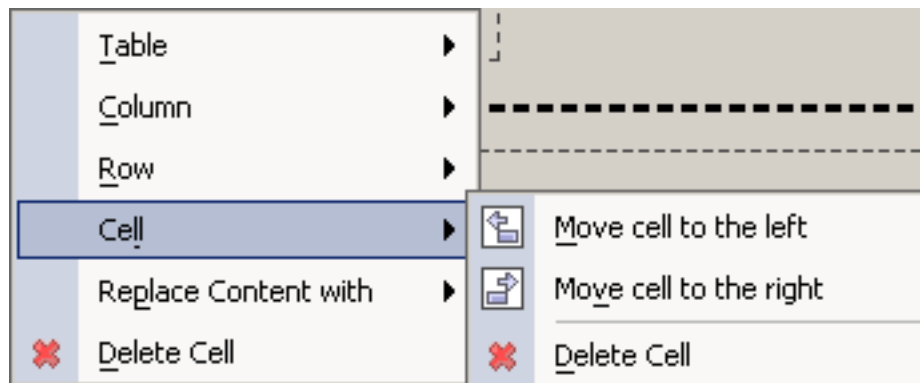
- On a cell: go to the Table menu option and select *Insert Row*. It is inserted above the row that contains the cell you are on.
- On a table: go to the Table menu option and select *Add* and then *Row*. It is appended at the bottom of the table structure.
- On a component inside a cell: go to the Table menu option and select *Insert Row*. It is inserted above the row that contains the cell you are on.

Cell

Cells contain components referenced to Fuego Object attributes or other containers (table or array.) Cells are contained within rows.

What can I do to a Cell?

Right-clicking on a cell in the structure tree of the presentation or having it selected on the presentation layout, a shortcut menu is opened with the following options:



- Move to the right,
- Move to the left,
- Replace with: this option replaces the contained component by the cell with a new component that allows you to select from a submenu. You can choose to copy the properties or not.
- Delete cell: if you delete all the cells in a row, the rows will be deleted as well.

Adding Cells to the Presentation

You can select the **Insert Column** option from the *Table* menu to add cells to the presentation.

This option is enabled when the selected component is a cell or a cell content and appends a column by the selected one. If this position is occupied in another column, the cell is expanded.

Deleting Cells from the Presentation

You can select the **Delete Column** option from the *Column* menu to remove cells to the presentation.

This option is enabled when the selected component is a cell or a cell content and removes the column by the selected one. If this position is occupied in another column, the cell horizontal expand is reset to one minus.

Cell Properties

- Background Color
- Horizontal Alignment
 - center

- left
- right
- Vertical Alignment
 - top
 - center
 - bottom
- Horizontal Span
- Vertical Span
- Border properties, Style/Width/Color for, if border properties are not defined for a cell, it inherits the border settings from the table that contains the cell.
 - Top Border
 - Bottom Border
 - Left Border
 - Right Border

Example

Demo Form

Name

Phone

Email

Adress

In this example, the different cell borders are set to:

- *Demo Form* cell:
 - Top border: Double, 5
 - Bottom border: Double, 5
 - Left border: Dotted, 5
 - Right border: Double, 5

- *Name* cell:
 - Top border: none
 - Bottom border: none
 - Left border: Double, 5
 - Right border: Double, 5
- *Phone* cell:
 - Top border: none
 - Bottom border: none
 - Left border: Dotted, 5
 - Right border: Dotted, 5
- *Mail* cell:
 - Top border: none
 - Bottom border: none
 - Left border: Dashed, 5
 - Right border: Dashed, 5

- *Address* cell:
 - Top border: none
 - Bottom border: none
 - Left border: Solid, 5
 - Right border: Solid, 5

- *Submit* button cell:
 - Top border: Inset, 15
 - Bottom border: Outset, 15
 - Left border: Inset, 15
 - Right border: Outset, 15

- *Cancel* button cell:
 - Top border: none
 - Bottom border: none
 - Left border: none
 - Right border: none

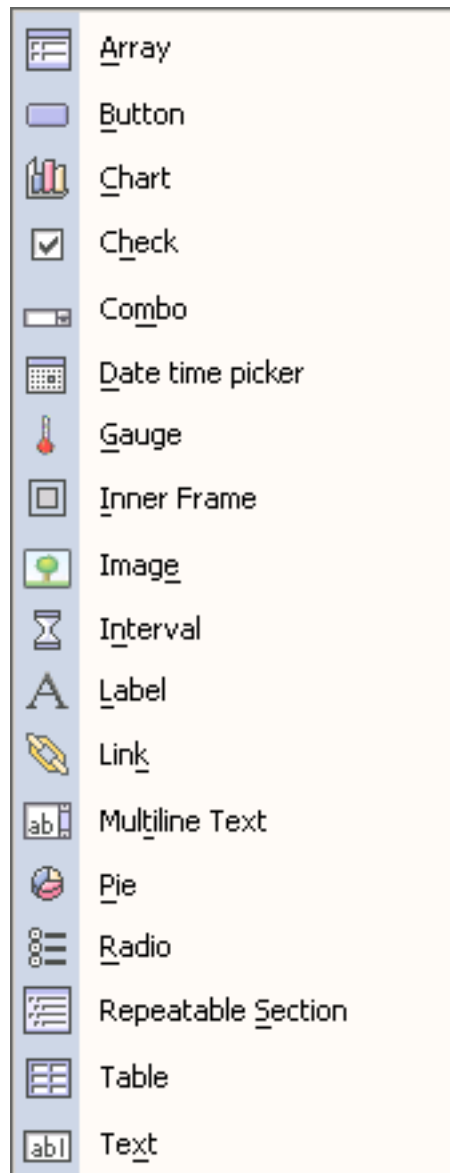
Adding Components to the Presentation

Add a Component

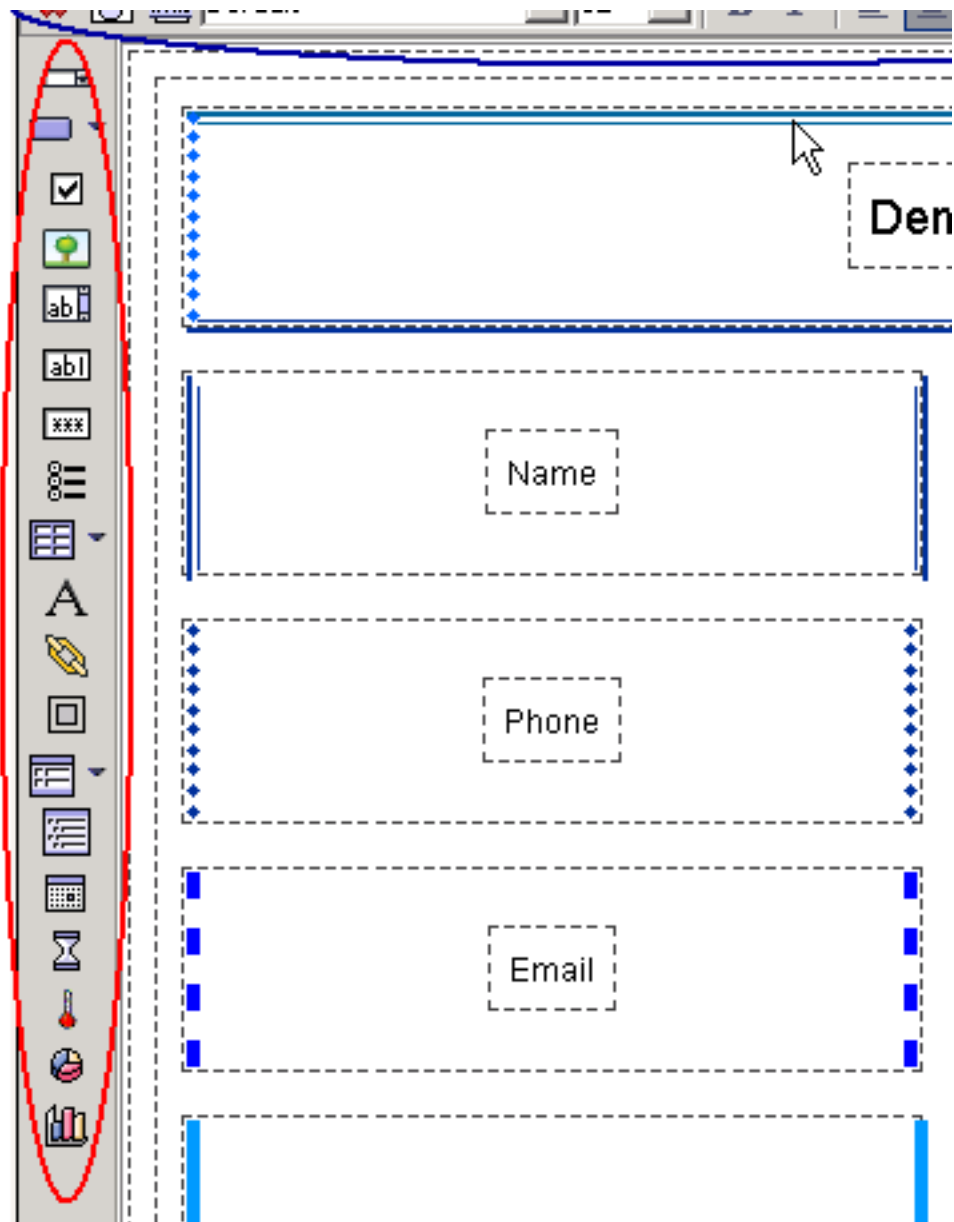
Components are always contained within a cell.

To add a component,

1. Select a cell. You can only add a component to the cell, if the cell is empty.
2. You can now go on doing:
 - a. Right-click and a shortcut menu is displayed. Select the **Add** option. A new submenu opens and it contains all the components you can add to this cell, or



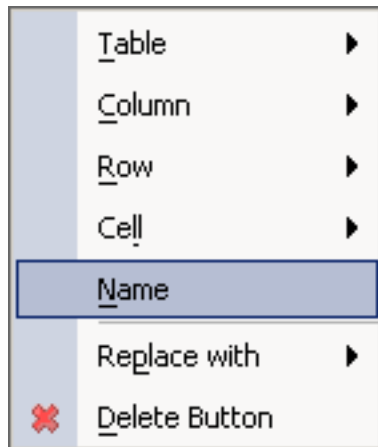
- b. Click on the component icon in the toolbar on the left side of the layout panel.



Additional Actions for Components

1. Select component either from the presentation structure or in its layout.

2. Right-click to display the shortcut options menu.



- a. Name: a dialog showing the actual name of the component is displayed (you can change it here if you want.)



- b. Replace with: a menu with the list of all possible components is opened. Select the one you want to use to replace the existing one in the cell.
- c. Delete: deletes the component from the cell.

For further details about properties, please refer to each component section below.


Anchor

Icon:

Anchor Properties

- Name
- Font Type
- Background Color
- Foreground Color
- Link type: can be one of the following
 - **External:** a link outside the presentation. **URL:** the url of the link. Eg: <http://www.fuego.com>
 - **Anchor:** a link to mark a certain part of the presentation. **Anchor name:** name to identify the anchor.
 - **Local:** a link to some part of the presentation which is marked with a link of type anchor. **Target anchor:** the name of the anchor.
 - Reference: when the url of the link is stored in the attribute of a Fuego Object. **Reference:** the attribute of the Fuego Object which contains the url. It can be selected by clicking over the dots and browsing the tree it displays.
- Text type: if the text to be displayed in the link is stored within an attribute of the Fuego Object, "reference" should be selected. If the text to be displayed is always the option "fixed" should be selected.
- Text: the String to be displayed if fixed option is selected, or the attribute of the Fuego Object if reference option is selected.

Array

Icon: 

Array Properties

- General Properties
 - Name, id
 - Referenced Data: you can only reference Fuego Object Group attributes to an array.
 - Editable
 - Background Color
 - Enabled/Disabled index tab browsing. Default value: Disabled. Mouse click actions are enabled.
 - Enable: It participates in the tab browsing circuit.
 - Disable: It does not participate in the tab browsing circuit.
 - TabIndex, which is the position of the input element in the tabbing order. Its default value will be 0.
- Layout Properties
 - Cell Padding
 - Cell Spacing
- Row Properties
 - Rows, if it has a limit of rows then type the number. The default -1 means no limit.

- Even row background color
- Odd row background color
- Table Properties
 - Header: Header label for the array, type in the name of each column.
 - Header Enabled: the header can be disabled.
 - Header width, Width for header column.
 - Header Alignment: how the label in each column is aligned.
 - Header Background Color
 - Header Foreground Color
 - Header Font type
 - Header Border
- Index Properties
 - Enabled: the row index can be disabled.
 - Background Color
 - Foreground Color
 - Font type
 - Border
- Paging Properties

- Background Color
- Foreground Color
- Font type
- Border
- Event Listener
 - Method invoked, of type *methodName(Fuego.Util.GroupEvent theEvent)*.
- Sorting
 - Enabled: enables or disables the array sort by column.
 - Background Color, the column header will be displayed in this color when the sort by this column is enabled by click the header.
 - Foreground Color, the column header label will be displayed in this color when the sort by this column is enabled by click the header.

Arrays' buttons

The array component provides some buttons to edit its information.





: Adds an empty row at the bottom.




: Removes the last row.

These two buttons actions are at array level. They are available when no row is selected.

 : Inserts an empty row above the selected row.

 : Inserts an empty row below the selected row.

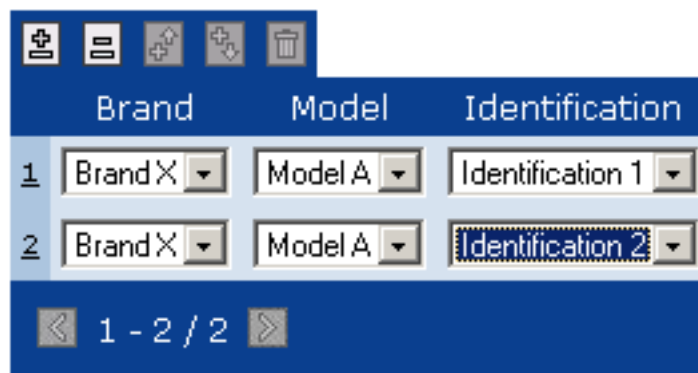
 : Deletes the contained values of the whole selected row.

These three buttons actions are at row level. They are available when a row is selected. To select an array row, click the row number on the left of each of them.

Array Row Counter

The row counter for an array presentation displays the number of elements in the array. For example having an array presentation with the *rows* property set to 5, if 6 rows were added to the array, the initial display of the array in the presentation shows 1-5 /6. If it is selected to view the 6th row, the display shows row six, and the counter shows 6-6 /6. See screen sequence below.

1. Two elements:



2. Five elements:

	Brand	Model	Identification
1	Brand X	Model A	Identification 1
2	Brand Y	Model A	Identification 2
3	Brand X	Model B	Identification 3
4	Brand Z	Model B	Identification 1
5	Brand Z	Model C	Identification 3

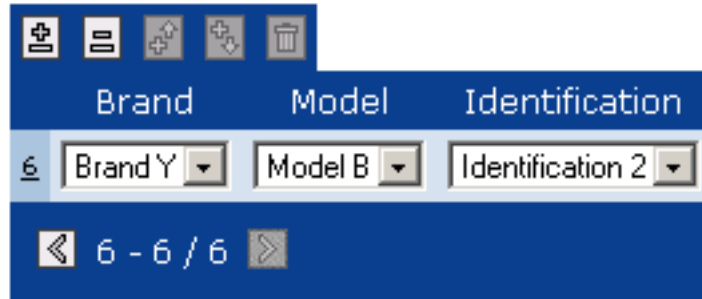
< 1 - 5 / 5 >

3. Six elements, still viewing the first five:

	Brand	Model	Identification
1	Brand X	Model A	Identification 1
2	Brand Y	Model A	Identification 2
3	Brand X	Model B	Identification 3
4	Brand Z	Model B	Identification 1
5	Brand Z	Model C	Identification 3

< 1 - 5 / 6 >

4. Viewing the sixth element:



Actions

You can add a column to an array by doing the following:

- In the Structure panel: right click on the array you want to add the column to and select the **Insert Column to Array** option.
- In the Presentation Designer Panel: select the array component and by right-clicking, select the **Insert Column to Array** option.
- In the Presentation Designer Panel: select a cell within the array component and by right-clicking select the **Insert Column to Array** option from the **Group** menu.


Sorting Array Columns

- It has no multiple column order.
- It sorts the dataModel.
- The sort order is toggled from ascending to descending in every click.

Some examples of sort:

"Sorting by column **Item Code** descending"


Items



	<u>ItemCode</u> ▲	<u>ItemPrice</u>	<u>ItemQty</u>
1	1030	50	10
2	1010	10	200
3	1001	500	300

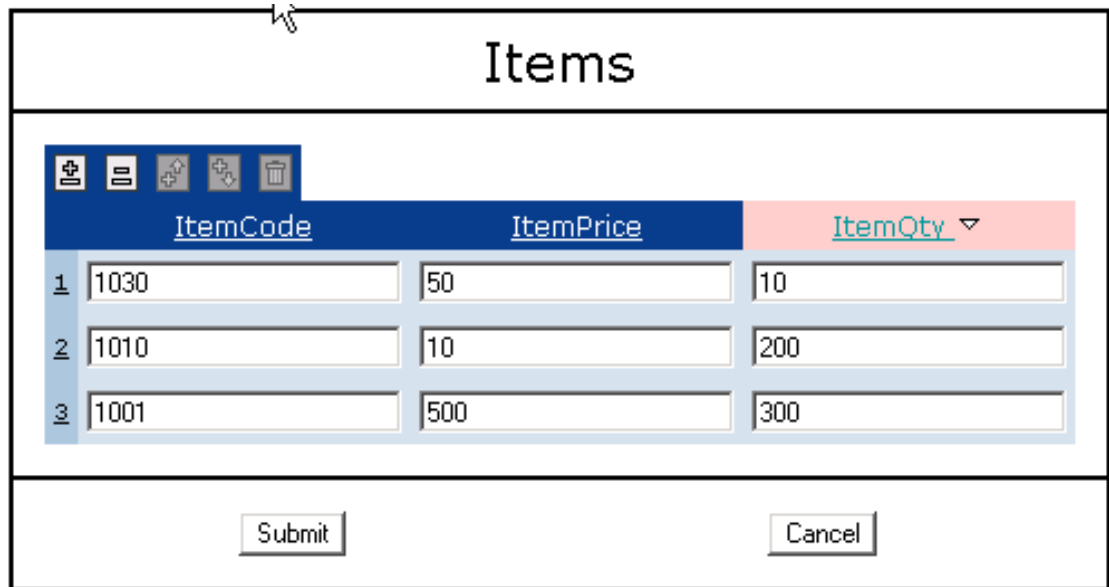
"Sorting by column **Item Quantity** ascending"

Items



	<u>ItemCode</u>	<u>ItemPrice</u> ▲	<u>ItemQty</u>
1	1001	500	300
2	1030	50	10
3	1010	10	200

"Sorting by column **Item Price** descending"



	ItemCode	ItemPrice	ItemQty ▾
1	1030	50	10
2	1010	10	200
3	1001	500	300

Submit Cancel

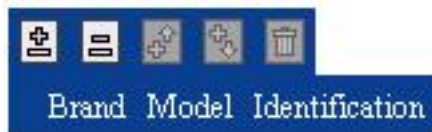
Previewing a Presentation with Arrays

If the presentation has a group, in preview mode a dummy row is added to the group only when it is empty. You add new rows from the *init* presentation method. As workaround, you can clear the group before adding new rows in the *init* method. If you run the presentation from debugger or runtime framework, this will not happen.

Examples

Defaults settings:

Default Header



Brand Model Identification

Default Body

	Brand	Model	Identification
1	<input type="text"/>	<input type="text"/>	<input type="text"/>
2	<input type="text"/>	<input type="text"/>	<input type="text"/>
3	<input type="text"/>	<input type="text"/>	<input type="text"/>

Default Index & Footer

	Brand	Model	Identification
1	Brand X	Model A	Identification 1
2	Brand X	Model A	Identification 2

< 1 - 2 / 2 >

Customizing the Header

- bgColor:193, 28, 13 fgColor: 232, 237, 140 font: Verdana 16 bold

	Brand	Model	Identification
--	-------	-------	----------------

Customizing the Body

- bgColor: red

	Brand	Model	Identification
1	<input type="text"/>	<input type="text"/>	<input type="text"/>
2	<input type="text"/>	<input type="text"/>	<input type="text"/>
3	<input type="text"/>	<input type="text"/>	<input type="text"/>


Alternating color rows

	Brand	Model	Identification
1	Brand X	Model B	Identification 3
2	Brand Y	Model C	Identification 1
3	Brand Z	Model A	Identification 2
<input type="button" value="Previous"/> 1 - 3 / 3 <input type="button" value="Next"/>			

Column Header and row index disabled

Brand X	Model B	Identification 3
Brand Y	Model C	Identification 1
Brand Z	Model A	Identification 2
<input type="button" value="Previous"/> 1 - 3 / 3 <input type="button" value="Next"/>		

Button

Icon: 

Presentation Buttons Methods

Standard methods that can be associated to presentation buttons are provided in presentable Fuego Objects.

Please refer to the **Standard Methods in Presentable Fuego Objects** section in the Fuego Object Methods help page for further and detailed information.

Button Properties

- Name, id
- Display: text that will be displayed within the button.
- Action: action the button performs. It may be one of the default actions, *submit*, *cancel*, *refresh* or *reset*, or custom one in which case you will have to select the *action* value for the button.
- Method Invoked: method invoked when the button is selected.
- TabIndex, which is the position of the input element in the tabbing order. Its default value will be 0.
- Transparent
- Font Type
- Background Color
- Foreground Color
- Border, style, width, and color




Examples

Refer to Fuego Object Examples for an example related to methods and how to handle the action button selected.

Example 1:



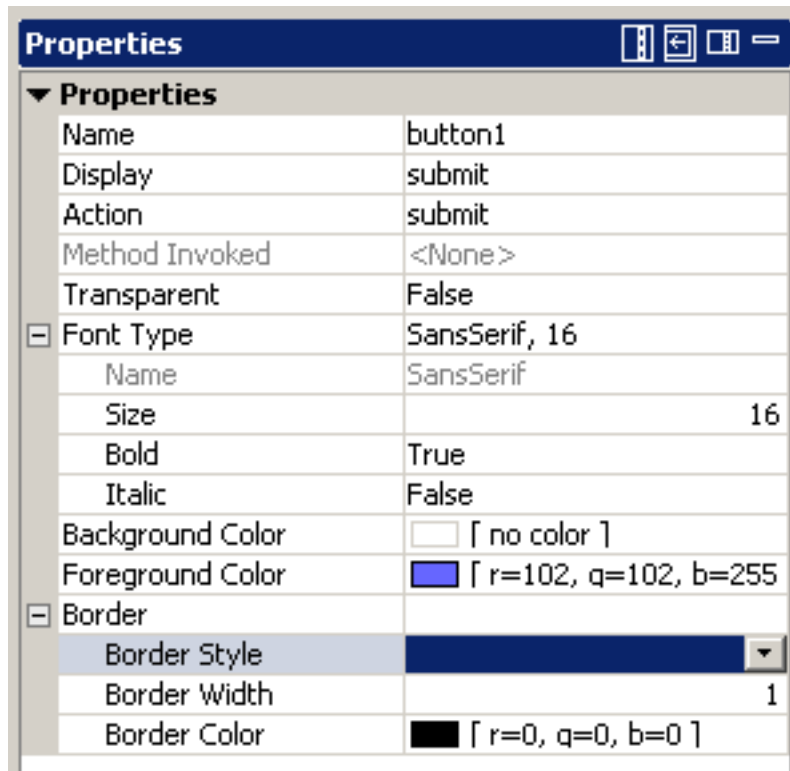
which set properties are:

Properties	
▼ Properties	
Name	button2
Display	cancel
Action	cancel
Method Invoked	<None>
Transparent	True
Font Type	Georgia, 16
Name	Georgia
Size	16
Bold	True
Italic	False
Background Color	 [r=255, q=204, b=204]
Foreground Color	 [r=255, q=204, b=204]
Border	
Border Style	
Border Width	1
Border Color	 [r=0, q=0, b=0]

Example 2:



which set properties are:



Although the border properties are not set in this button, they inherit the values set to the cell which contains them. In this example, the cell has the border properties set to:

- *Submit* button cell:
 - Top border: Inset, 15
 - Bottom border: Outset, 15
 - Left border: Inset, 15
 - Right border: Outset, 15

Check

Icon: ☒

Check properties

- Name, id
- Reference Data, can only reference to Fuego object attributes of Bool type.
- Display: text that will be displayed within the button.
- On change invoke: method that will be invoked when the check is selected.
- Validation Error Message
- TabIndex, which is the position of the input element in the tabbing order. Its default value will be 0.
- Required
- Editable
- Font Type
- Background Color
- Foreground Color
- Border, style, width, and color

"Check boxes" can reference fuego objects attributes defined as an enumeration type.

Example



which set properties are:

Properties	
Name	check0
Referenced Data	checking
Display	Verified
On Change Invoke	<None>
Validation Error Message	
Required	True
Editable	True
Font Type	Default, 18
Name	Default
Size	18
Bold	True
Italic	True
Foreground Color	[r=102, q=51, b=255]
Background Color	[r=153, q=204, b=255]
Border	Double, 4
Border Style	Double
Border Width	4
Border Color	[r=51, q=153, b=0]

Combo

Icon:

Combo Properties

- Name, id
- Reference Data, can only reference to Fuego object attributes of type: String, Int, Real, Decimal, Bool, Interval and Time.

- On change invoke: method that will be invoked when the check is selected.
- Validation Error Message
- Required
- Editable
- TabIndex, which is the position of the input element in the tabbing order. Its default value will be 0.
- Transparent
- Horizontal Alignment
- Font Type
- Background Color
- Foreground Color

Special Considerations

- If the presentation is run in HTML, combo boxes do not support the *Horizontal Alignment* property.
- Depending on the profile of the referenced attribute, the list of valid values will be as follows:
 - If the attribute has been defined as *not null*: Only the valid values are shown.
 - If the attribute has been defined as *null*: The valid values plus an empty value that represent the NULL value are shown.
- If the *combo box* is within a group or repeatable section, the

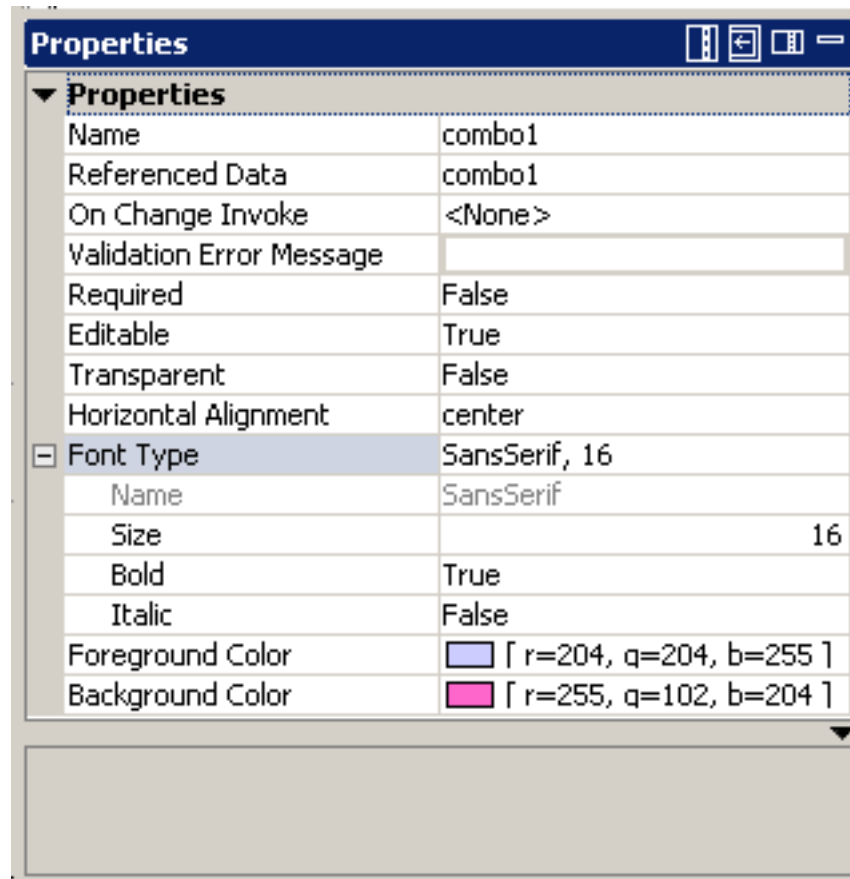
property *Load once* is available.

Examples




In this case the *null* value is present, as the attribute has not been defined as not null.

This combo set properties are:



Date Time Picker

Icon: 

Date Time Picker Properties

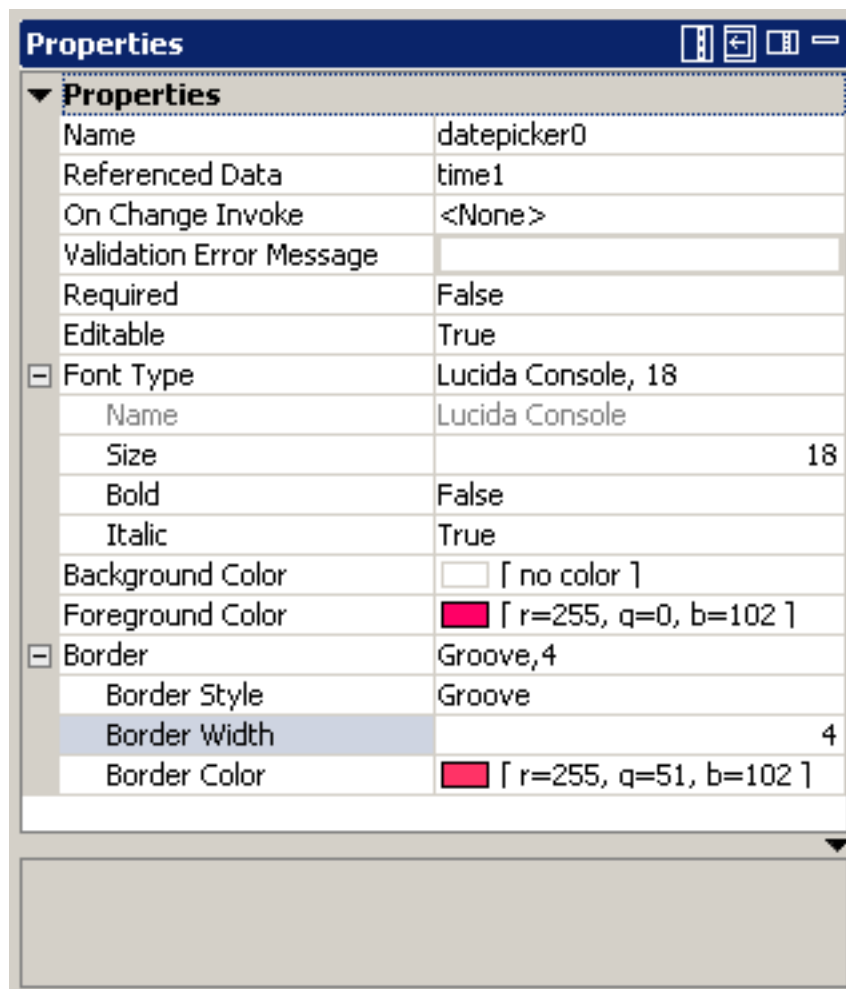
- Name, id
- Reference Data: only allows to reference those Fuego Object attributes defined of type *Time*. Remember that depending on how the *absolute* property of the attribute is defined, the value will include or not the time zone.
- On change invoke: method that will be invoked when the check is selected.
- Validation Error Message
- Required
- Editable
- TabIndex, which is the position of the input element in the tabbing order. Its default value will be 0.
- Enabled/Disabled calendar tab browsing. Default value: Enabled. Mouse click actions are enabled.
 - Enable: It participates in the tab browsing circuit.
 - Disable: It does not participate in the tab browsing circuit.
- DHTML Calendar
 - True: Opens a DHTML Calendar.
 - False: Opens a HTML Calendar.
- Font Type

- Background Color
- Foreground Color
- Border, style, width, and color

Examples



Set properties are:



Image

Icon:

Image Properties

- Name
- Reference Data, can only reference to Fuego object attributes of Binary type.
- Encoded image
- On click, method that will be invoked when a click is performed on the image.
- Is Required?
- On change invoke: method that will be invoked when the check is selected.
- Link
- Standard Browser
- Editable
- Background Color
- Border, style, width, and color
- Width
- Height
- *Editor Properties*
 - Display, label that appears below the image.
 - Font Type, font applied to the display text.

- Horizontal Alignment, alignment applied to the display text.
- Background Color

Examples

In the following examples:

Example Image 1



Example Image 2

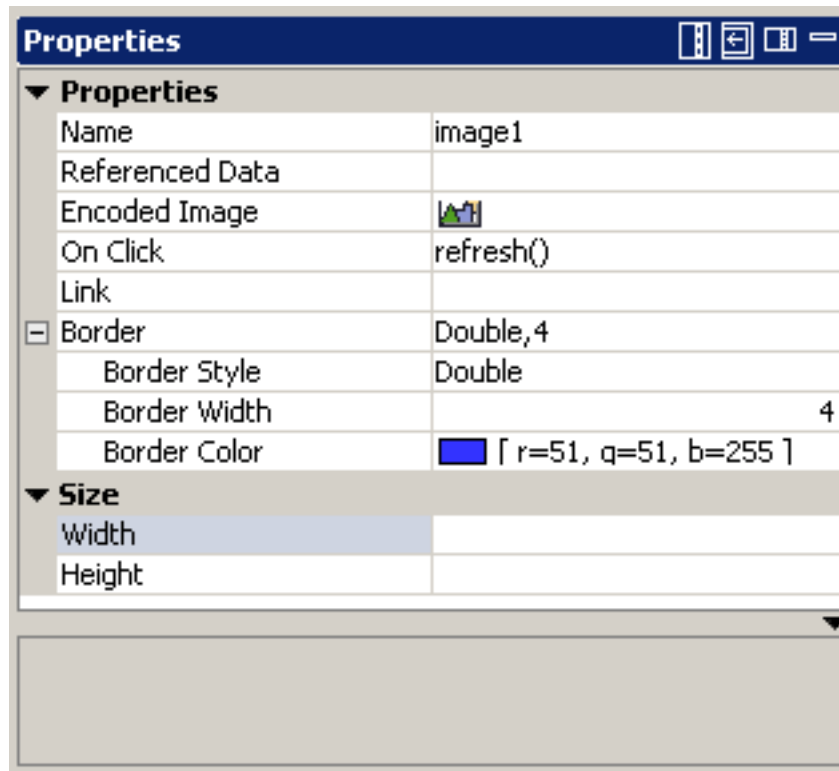


Submit

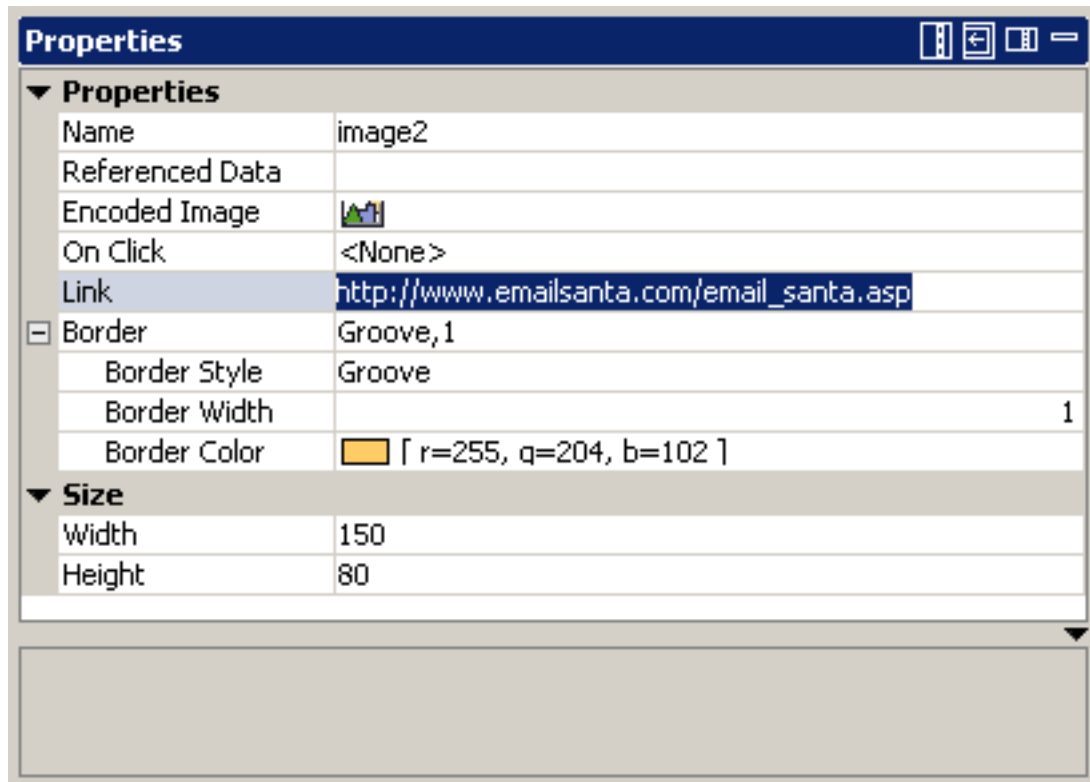
Cancel

Where the properties have been set as:

Example Image 1:



Example Image 2:



In this case, if the *on click* property had been set too, the *link* would overwrite it and take a look at the effect of the *width* and *length* properties.

Inner Frame

Icon:

Inner Frame Properties

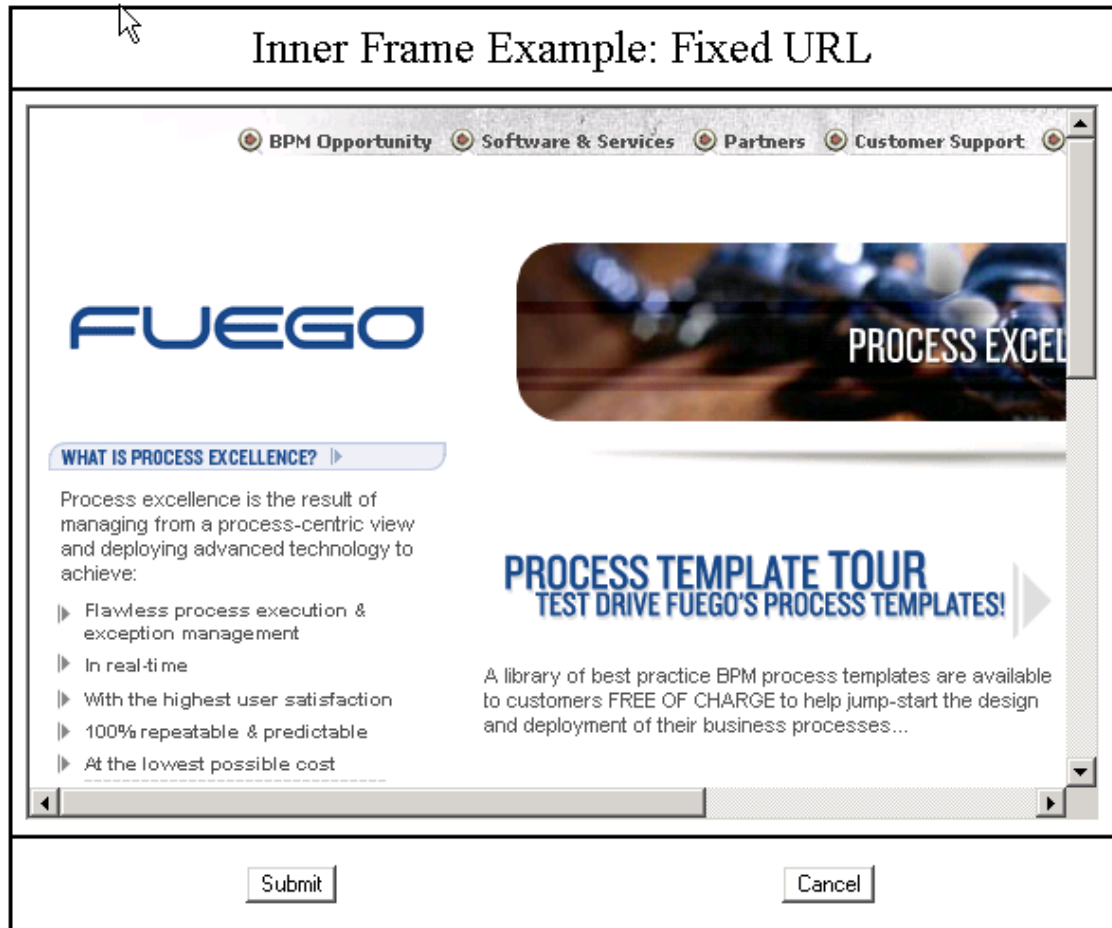
- Name, id
- Width, can be set in percentage, centimeters, pixels, etc,
- Height, can be set in percentage, centimeters, pixels, etc,
- Scrolling,
 - Yes, always with scrool bars, although the HTML fits in the

preset display area,

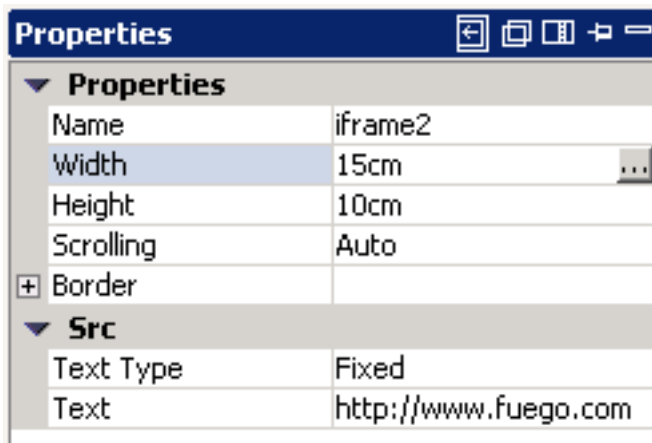
- No, with no scroll bars, although the HTML exceeds the preset size of its display area,
- Auto, scroll bars are present if needed.
- Border, style, width, and color
- Src
 - Text Type: Reference or Fixed,
 - Reference: Attribute where the URL is held,
 - Text : URL

Examples

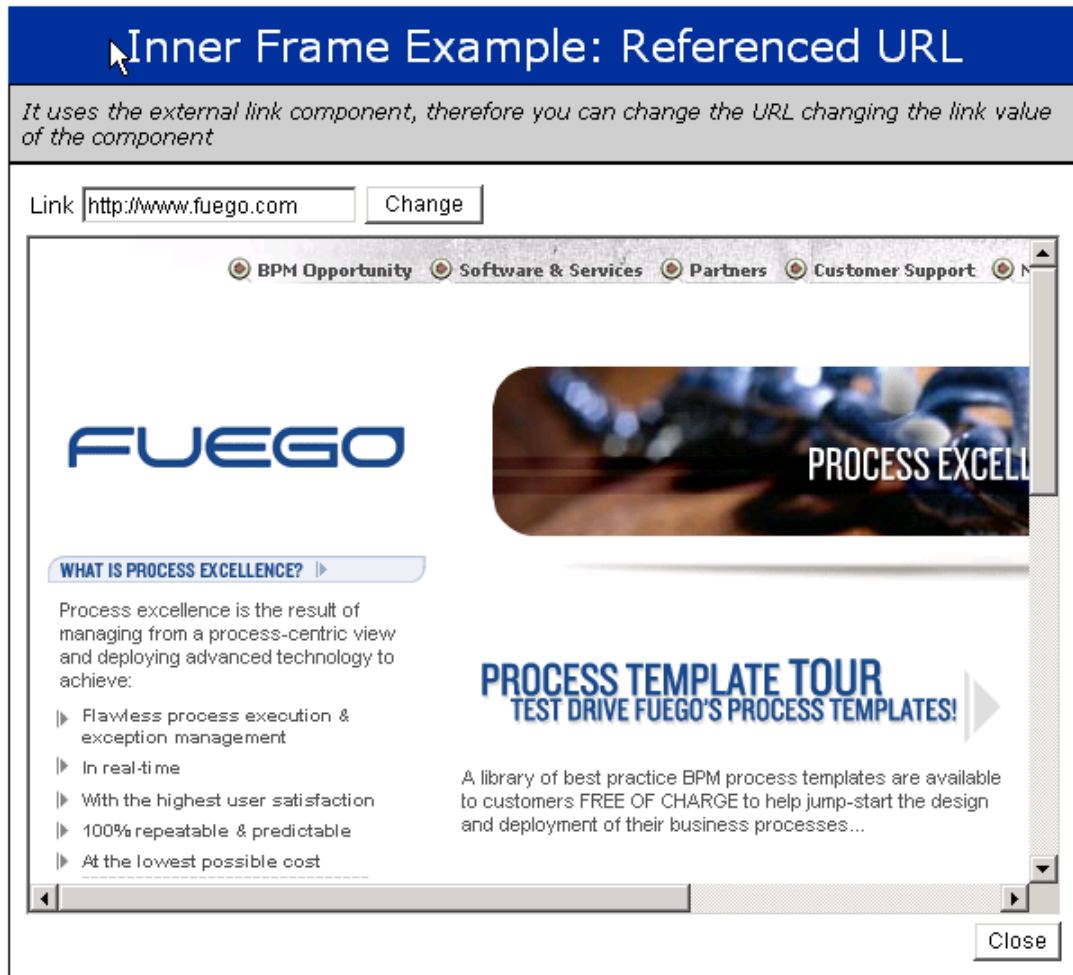
Fixed URL



Where the properties have been set as:



Referenced



Where the properties have been set as:

Properties	
▼ Properties	
Name	iframe
Width	100%
Height	10cm
Scrolling	Auto
+ Border	None, 1, [r=0, g=0, l
▼ Src	
Text Type	Reference
Reference	iframeSrc

In this case, the URL is taken from the "iframeSrc" attribute. This attribute is defined as:

The screenshot shows the configuration window for the 'IFrameReferencedURL' attribute. The window has a title bar with 'IFrameReferencedURL' and a sub-tab labeled '...iframeSrc'. The main area is divided into sections: 'Type' with a dropdown set to 'String' and a 'Maximum length' field set to '0'; 'Storage & Constraints' with checkboxes for 'Virtual', 'Primary Key', and 'Not null', all of which are unchecked; and 'Default value' with a radio button selected for a text input field containing 'http://www.fuego.com' and another radio button for 'Null'. At the bottom, there are two tabs: 'Require expression' (selected) and 'Check expression'.


In this example, the method associated to the presentation button "Change" sets the URL in the Inner Frame dynamically, with the following code:

```
setText this
    using componentId = "iframe",
        text = iframeSrc
```

This could be implemented as well, with a method that sets dynamically the URL related to the attribute in the *on change* property.

See the project *InnerFrameTest* under the sample directory under the FuegoBPM Studio installation directory.

Interval

Icon: 

Interval Properties




- Name, id
- Referenced Data, can only reference to Fuego object attributes of Interval type.
- On change invoke: method that will be invoked when the check is selected.
- Validation Error Message
- Required
- Editable
- TabIndex, which is the position of the input element in the tabbing order. Its default value will be 0.
- Font Type
- Background Color
- Foreground Color
- Border, style, width, and color

Examples

Interval



Where the properties have been set as:

Properties	
Name	interval1
Referenced Data	interval1
On Change Invoke	<None>
Validation Error Message	
Required	False
Editable	True
Font Type	Invalid font type
Name	
Size	16
Bold	False
Italic	True
Background Color	 [r=255, q=255, b=204]
Foreground Color	 [no color]
Border	Inset, 4
Border Style	Inset
Border Width	4
Border Color	 [r=0, q=204, b=102]

Label

Icon: A

Label Properties

- Name, id
- On click: method that will be invoked when a click is performed on the image.
- Link* Standard Browser
- Display: text that will be displayed within the button.

- TabIndex, which is the position of the input element in the tabbing order. Its default value will be 0.
- Font type
- Background Color
- Foreground Color

The label is shown depending the presentation mode:

- If it is in **INPUT MODE**: for both *On click* and *Link* it is displayed as an Anchor.
- If it is in **DISPLAY MODE**: then,
 - *On click* property set: it is shown as a simple text.
 - *Link* property set: it is shown as an Anchor.

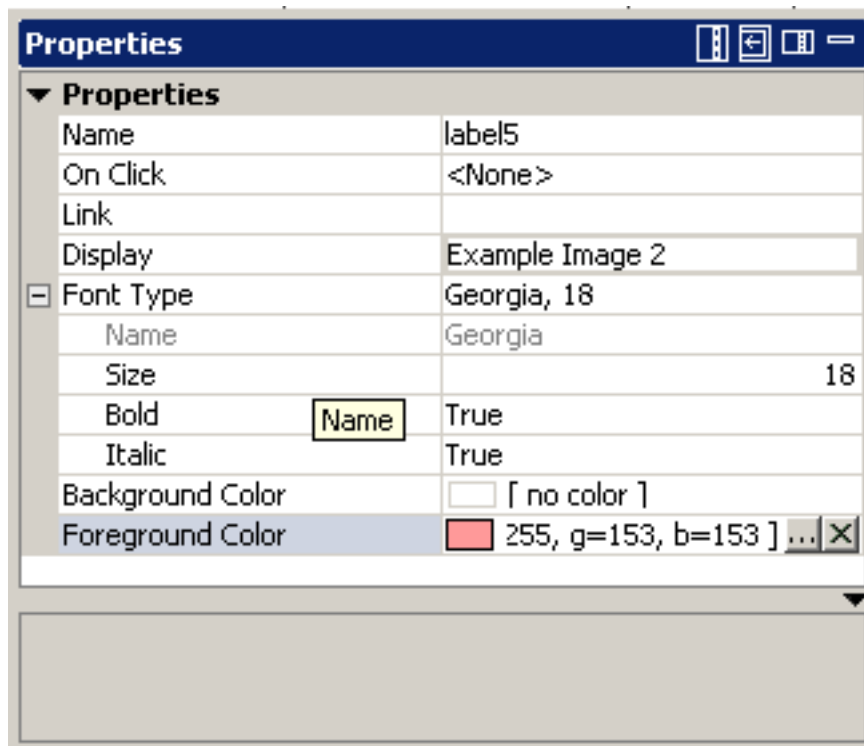
Examples

Redefining the label properties to the label component of the image example:


Example Image 2



In this case, the link that had been set to the image before could be set to the label.



Multiline Text

Icon: 

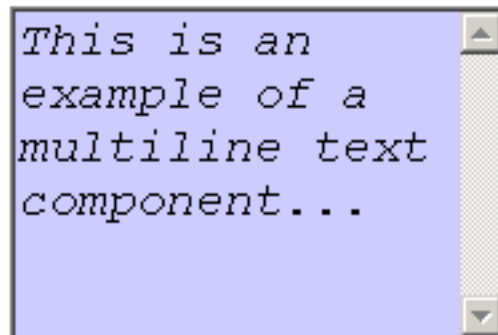
Multiline Text Properties

- Name, id
- Reference Data: can only reference to Fuego object attributes of String type.
- On change invoke: method that will be invoked when the check is selected.
- Validation Error Message
- Required
- Editable




- TabIndex, which is the position of the input element in the tabbing order. Its default value will be 0.
- Columns
- Lines
- Font Type
- Background Color
- Foreground Color
- Border, style, width, and color

Examples


Multiline Text



Which properties have been set to:

Properties	
▼ Properties	
Name	multinetext1
Referenced Data	
On Change Invoke	<None>
Validation Error Message	
Required	False
Editable	True
Columns	15
Lines	6
<input checked="" type="checkbox"/> Font Type	Monospaced, 18
Name	Monospaced
Size	18
Bold	False
Italic	True
Background Color	 204, g=204, b=255] ... X
Foreground Color	 [no color]
<input checked="" type="checkbox"/> Border	
Border Style	
Border Width	1
Border Color	 [r=0, q=0, b=0]

Radio Button

Icon: 

Radio Button Properties

- Name, id
- Reference Data: can only reference to Fuego object attributes of type: String, Int, Real, Decimal, bool, Interval and Time.
- Display: text that will displayed within the button.

- On change invoke: method that will be invoked when the check is selected.
- Validation Error Message
- Required
- Editable
- TabIndex, which is the position of the input element in the tabbing order. Its default value will be 0.
- Font Type
- Background Color
- Foreground Color
- Border, style, width and color
- Columns and Rows: in how many columns and rows the radio button will be built.

Special Considerations

Depending on the profile of the referenced attribute, the list of valid values will be as follows:

- If the attribute has been defined as *not null*: Only the valid values are shown, all buttons in the group are unselected.
- If the attribute has been defined as *null*: Only the valid values are shown, but all buttons in the group can be unselected. Unselecting a valid value means that the NULL value is selected.
- If the *radio button* is within a group or repeatable section, the property *Load once* is available.

Radio Buttons can reference fuego objects attributes defined as an enumeration type.

Examples

Take a look at the following examples:

Example Radio 1

Chose

<input type="radio"/> Orange	<input type="radio"/> Green	<input type="radio"/> Grey
<input type="radio"/> White	<input type="radio"/> Red	<input type="radio"/> Blue
<input type="radio"/> Yellow	<input type="radio"/> Brown	<input type="radio"/> Black



Example Radio 2

<input checked="" type="radio"/> <i>White</i>	<input type="radio"/> <i>Brown</i>
<input type="radio"/> <i>Yellow</i>	<input type="radio"/> <i>Green</i>
<input type="radio"/> <i>Orange</i>	<input type="radio"/> <i>Black</i>
<input type="radio"/> <i>Red</i>	<input type="radio"/> <i>Grey</i>



<input type="button" value="Submit"/>	<input type="button" value="Cancel"/>
---------------------------------------	---------------------------------------

Where the properties have been set as:

Example Radio 1:


Properties	
▼ Properties	
Name	radio1
Referenced Data	radio1
Display	Chose
On Change Invoke	<None>
Validation Error Message	
Required	False
Editable	True
Font Type	SansSerif, 16
Name	SansSerif
Size	16
Bold	True
Italic	False
Background Color	 [r=204, q=204, b=204]
Foreground Color	 [r=0, q=153, b=153]
+ Border	
▼ Layout	
Columns	0
Rows	3

Example Radio 2:

Properties	
▼ Properties	
Name	radio2
Referenced Data	radio2
Display	
On Change Invoke	<None>
Validation Error Message	
Required	False
Editable	True
<input checked="" type="checkbox"/> Font Type	Georgia, 14
Name	Georgia
Size	14
Bold	True
Italic	False
Background Color	 [r=0, q=153, b=153]
Foreground Color	 [no color]
<input checked="" type="checkbox"/> Border	
▼ Layout	
Columns	5
Rows	4

The attribute *radio2* related to this component, has been defined as *not null*.

Repeatable Section

Icon: 

Repeatable Section Properties

- Name, id
- Referenced Data: you can only reference Fuego Object Group attributes to an array.

- Editable
- TabIndex, which is the position of the input element in the tabbing order. Its default value will be 0.
- Rows: if it has a limit of rows then type the number. The default -1 is no limit.
- Background Color
- Cell Padding
- Cell Spacing
- Table Header: header label for the array, type in the name of each column.
- Table Header width: width for header column.
- Table Header Alignment: how the label in each column is aligned.
- Table Header Background Color
- Table Header Foreground Color
- Table Header Font type
- Table Header Border
- Table Index Background Color
- Table Index Foreground Color
- Table Index Font type
- Table Index Border
- Table Paging Background Color
- Table Paging Foreground Color
- Table Paging Font type

- Table Paging Border
- Event Listener
 - Method invoked, of type *methodName(Fuego.Util.GroupEvent theEvent)*.

Repeatable Section Buttons

The array component provides some buttons to edit its information.



: Adds an empty row at the bottom.



: Removes the last row.

These two buttons actions are at array level. They are available when no row is selected.



: Inserts an empty row above the selected row.



: Inserts an empty row below the selected row.



: Deletes the contained values of the whole selected row.

These three buttons actions are at row level. They are available when a row is selected. To select a repeatable section row, click the row number on the left of each of them.

Actions

You can add a column to a repeatable section by doing the following:

- In the Structure panel: right-click on the repeatable section component you want to add the column to and select the **Insert**

Column to Repeating Section option.




- In the Presentation Designer Panel: select the array component and by right-clicking, select the **Insert Column to Repeating Section** option.
- In the Presentation Designer Panel: select a cell within the array component and by right-clicking select the **Insert Column to Repeating Section** option from the **Group** menu.

Examples

Defaults settings: Are the same that those for the Array.

What does a Repeating section look like:

Presentation2

	TEXT	TABLE	IMAGE
1	<input type="text"/>	<div style="border: 1px solid #ccc; padding: 2px;"> <input type="text"/> </div> <div style="border: 1px solid #ccc; padding: 2px;"> <input type="text"/> </div>	 Change
2	<input type="text"/>	<div style="border: 1px solid #ccc; padding: 2px;"> <input type="text"/> </div> <div style="border: 1px solid #ccc; padding: 2px;"> <input type="text"/> </div>	 Change
3	<input type="text"/>	<div style="border: 1px solid #ccc; padding: 2px;"> <input type="text"/> </div> <div style="border: 1px solid #ccc; padding: 2px;"> <input type="text"/> </div>	 Change

Text & Password Field

Text Icon: 

Password Field Icon: 

Text & Password Field Properties

- Name, id
- Reference Data
- Font Type
- Background Color
- Foreground Color
- Column Quantity
- Editable
- TabIndex, which is the position of the input element in the tabbing order. Its default value will be 0.
- Password
- On change invoke: method that will be invoked when the check is selected.
- Validation Error Message
- Alignment, left, center, right
- Required
- Border, style, width and color

The only difference between a text and a password field is that the latter has the *Password* property set to true. If you change it to false,

it immediately becomes a plane text field.

If it is a text field, then it has:

- Input Patterns and
- Masks

For further and detailed information about this properties, please, refer to the sections that explain them below.

Examples

Take a look at the following examples:

The image shows a form layout within a rectangular frame. On the left side, there are two labels: 'Text' and 'Password'. To the right of the 'Text' label is a white rectangular input field containing the text 'Text Example' in red. To the right of the 'Password' label is a light blue rectangular input field containing eight black asterisks. Below these fields, separated by a horizontal line, are two buttons: 'Submit' on the left and 'Cancel' on the right.

Where the properties have been set as:

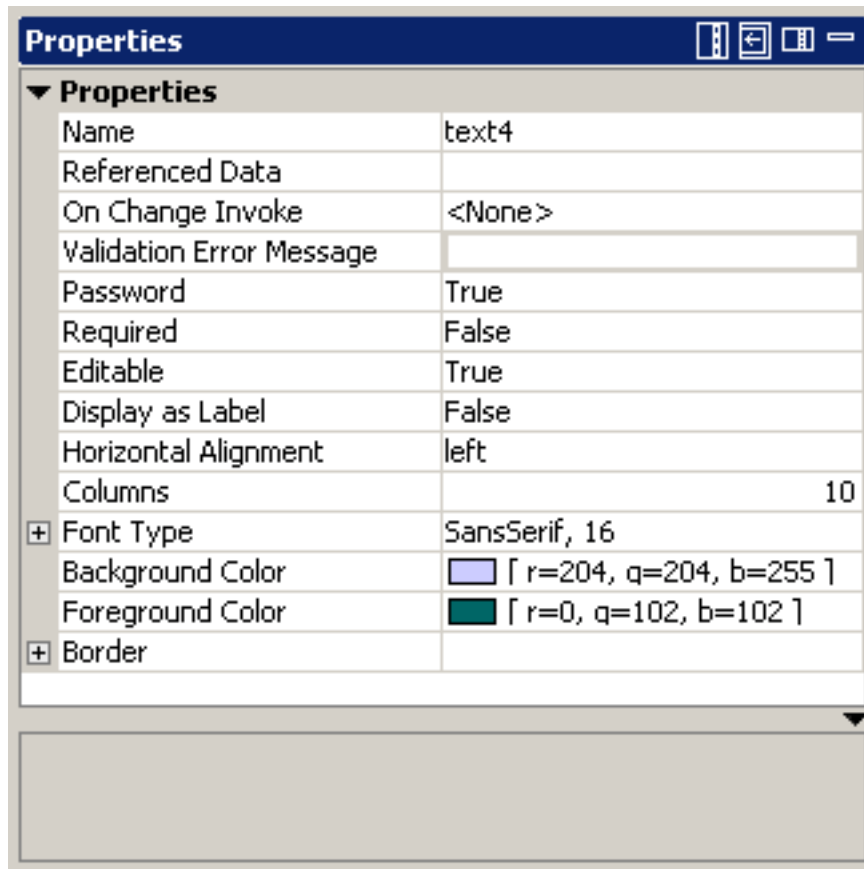
Example Text:

Properties

▼ Properties

Name	text1
Referenced Data	
On Change Invoke	<None>
Validation Error Message	
Password	False
Required	False
Editable	True
Display as Label	False
Horizontal Alignment	left
Columns	20
+ Font Type	Garamond, 16
Background Color	<div></div> [no color]
Foreground Color	<div></div> 255, g=153, b=153] ... X
+ Border	

Example Password:



The attribute *radio2* related to this component has been defined as *not null*.

Defining Common Properties to Presentation Elements

You can select more than one element of the presentation, a structure element and components and set the common properties to the all of them with the same values.

You select more than one element by Click + Ctrl on each of them. The Properties option displays only the common properties to the selected elements.

For example, by doing this, you can set the *background color* property of all the cells to the same color.

Referencing Methods from the Presentation When the Fuego Object is Implementing Behavior Inheritance

If a inherited method is redefined in the Fuego Object, only one method will be shown in the method selection combo of those presentation components properties that reference the Fuego Object methods.

If the Fuego Object has overwritten the method, only the one from the Fuego Object will be shown, if the Fuego Object has not overwritten the method from the delegate, then the inherited method will be shown.

Only one method will be shown and it will invoke the closest one to the object in the inheritance chain.

Data Input - Masks & Input Patterns

Input Patterns

Input patterns objective is to check with a pattern (written as a regular expression) the user input in a Fuego Object presentation's textfield component.

The pattern is an optional property of presentation's textfield components (only when they reference String or Int attributes.) It allows us to perform inputs in a same attribute of different data subsets.

For example, a Fuego Object has a String attribute called *attr1*, which default input pattern for is both a and numeric characters.

We need to present this attribute in two presentations. One of them allowing only numeric input values and the other only with alpha input values.

As explained before, the input pattern is individual by presentation component; that is why it is very easy to solve this situation.

In the first presentation the *input pattern* for the component with a reference to *attr1* will be "only numeric" and in the second one "only alpha," where:

only numeric: `^[0-9]*$`,

only alpha: `^[a-zA-Z]*$`

Let's see another example, a Fuego Object has a String attribute called *email*. The default input pattern for this attribute would be any character, alpha and numeric.

We want to present this attribute in two presentations. One allowing only e-mails from fuego.com input values and the other only with from fuegolabs.com input values.

In the first presentation the input pattern for the component with a reference to attr1 will be "ends with fuego.com" and in the second one "ends with fuegolabs.com," where:

ends with fuego.com: `^[a-zA-Z]*fuego\.com)$`

and **"ends with fuegolabs.com":**
`^[a-zA-Z]*fuegolabs\.com)$`

The check of the pattern input of a textfield will be done every time its content changes. The only exception is a cancel action. Cancel ignores all validation and the presentation is closed.

Input Pattern Definition

To define an input pattern,

- Type the regular expression that defines it in the *input pattern* field,
- Type the message to display when the input pattern fails in the *Fail message* field.
- Set a color to the fail message.

Input Masks

By using masks the data is masked to be shown and unmasked to be stored. Let's take a telephone number example; the mask would be `(####)####-####`, the user would see **(5411)4861-2927** but the data

would be stored as *541148612927* in any numeric or string typed fuego object attribute.

Mask valid characters

There are different characters to define a mask,

Character	Meaning
'	Escape character, used to escape a formatted character.
#	Any valid number
U	Any character. All lowercase letters are mapped to uppercase.
L	Any character. All upercase letters are mapped to lowercase.
?	Any character.
A	Any character or valid number.
*	Anything.

All characters are required.

Mask Definition

To define a mask:

1. Type the regular expression that defines it in the *Mask field*,
2. Type the message to display when the input pattern fails in the *Mask Fail message* field.
3. Set a color to the mask fail message.

Output Masks

Output Masks are applied to non editable textfields of type String, Integer, Decimal or Real. There are two modes: *Generic* and *Numeric*.

Numeric Masks

A numeric mask must be applied to numeric values. It can be applied to a String but it must be a number or it must match with the mask. For example:

- **Pattern \$#,###.00**
- **Valid Values:** 1 or 10000 or \$1 or 10000.00. But %333 is not a valid value.

Non valid values are shown without mask.

Mask characters

Character	Meaning
#	optional number
0	fix number (fill with zero if not exists)
,	Thousand separator (Uses the user locale)
.	Decimal separator (Uses the user locale)
;	Positive and Negative mask separator Eg: #;(#). Negative values are show between ()
'	Escape character

Examples

Mask	Value	Output
#,###.##	2	2
#,###.##	1000000	1,000,000
#,###.##	1.369	1.37
#,###.00	2	2.00

Mask	Value	Output
#,###.00	1000000	: 1,000,000.00
#,###.00	1.369	: 1.37
\$ #,##0.00	2	\$ 2.00
\$ #,##0.00	1000000	\$ 1,000,000.00
\$ #,##0.00	1.369	\$ 1.37
\$ #.00;(\$#.00)	2	\$ 2.00
\$ #.00;\$ (#.00)	-2	\$ (2.00)

Generic Masks

Generic masks can be applied to Number and String values.

- Non valid values are shown without mask.
- The mask evaluation is from right to left.
- If the value is shorter than the mask, the value is completed with: 0 for # and spaces for the rest.
- If the value is longer than the mask, the mask is applied and the characters that are not included in the mask are shown without any changes.

Mask characters

Character	Meaning
#	Number
?	Letter
L	Letter to lowercase
U	Letter to uppercase
A	Any character
'	Escape character

Examples

Mask	Value	Output
"UUUU"	hola	"HOLA"
"UUUU"	holas	"hOLAS"
"UUUU"	ho "	HO"
"UUUU"	hol1 Mundo	"hol1" (Invalid)
"####-####-####"	123456789	"0001-2345-6789"
"####-####-####"	12345678910234	"123456-7891-0234"

Defining Behavior to a presentation

Some behavior is inherited from the attributes' definition. For example, calculate, require, and check expressions that can be specified when the Fuego Object Data Model are defined. Presentations can have their own behavior definition by invoking methods in action properties like *on change* or *on click*. Methods invoked in array fields are those defined within the referenced group and not at the Fuego Object level.

Remember that methods defined as *runs on server* invoked from a presentation cannot invoke a *client side* method, such as a **refresh()**, **reset()**, **submit()**, as in that case you are asking to go back to the client side. Once you are processing on the server side, you cannot go back to the client side from the same method. To do so, you need to define another method that invokes the first one. In addition, it is not possible to invoke a method *runs on client* from a method defined as *runs on server*. An example of this situation is an action method that performs processing on a database as a store, that has to be invoked from the **submit** button. In this case, you cannot invoke the *runs on server* method and define in it as statement the *submit* invocation. To do so, you need to define a *supermethod* that invokes the **store** and then the **submit**.

Example

A `saveAll()` method in the Stock Order Fuego Object defined as

shown below, cannot be invoked from the submit button:

```
store stkOrder

for each element in itemLine do
  element.stkOrdit.stkcode = stkOrder.stkcode
  element.stkOrdit.stktype = stkOrder.stktype
  store element.stkOrdit
end

submit("submit")
```

The correct way is to define an action button, suppose its name is *store* and the methods:

saveAll() code:

```
store stkOrder

for each element in itemLine do
  element.stkOrdit.stkcode = stkOrder.stkcode
  element.stkOrdit.stktype = stkOrder.stktype
  store element.stkOrdit
end
```

saveButton() code:

```
saveAll()
submit("submit")
```

The new button defined *store* invokes the method *saveButton*.

For more examples of implementation, please refer to Fuego Object Examples.

On Change

Suppose you have a field on a presentation that is dependant on what the user selects in a different field. For example, in a shipping process' Fuego Object, the presentation may have a field called Customer that is a drop-down menu listing all customers. When the customer is selected, the Address field automatically populates with the address on file for the customer. This scenario is easy to implement by invoking a method when the Customer field changes. If the component is inside the array, the invoked method needs an int argument that represents the current row value.

OnChange default focus behaviour:

- Working in RemoteScripting mode: After an on change method the focus goes to last focused element.
 - By TAB browsing: the focus goes to next tabbing element.
 - By clicking in other element: the focus goes to the clicked element.
 - By clicking outside but in no element: The focus remains in the element.
- Working in Request mode: After an on change method the focus remains in the element. But with the *focus* method the developer can decide where the focus will be after the onchange. See Fuego Object Methods for details.

On Click

A method invoked when the content is clicked off, a method without arguments and without return value. If the component is inside the array the method invoked needs an int argument that represents the current row value.

Require Expressions

This is a preconditional validation to avoid assigning a wrong value to a Fuego Object attribute. That is why they are evaluated *every time* the presentation tries to update the Fuego Object data model, in any action that makes the page refresh (see What makes a page refresh section.)

For more information see Fuego Object Attributes.

Check Expressions

Check expressions are only evaluated when the *submit* action is executed in presentation. This is like this because the expression of integrity for the attribute may depend on other attributes that may be input later in order.

For more information see Fuego Object Attributes.

Refresh Valid Values

The *refreshValidValues* method is most often called from the code of an Fuego Object method. It is useful when displaying a Fuego Object presentation. For example, you may have two attributes, *customerCode* and *paymentType* with lists of valid values. The list of valid values in the *paymentType* attribute is dependent on the value the user selected in the *customerCode* attribute. To enable this configuration, the *customerCode* attribute's presentation field has a method defined in the On Change Invoke property. This method refreshes the valid values of the *paymentType* attribute field.

Note



Methods called from a presentation cannot include interactive statements like *input*, *display*, etc. If so, the *java.io.NotSerializableException* will be thrown. If you need to display, use a presentation field to do so.

Handling Error & Warning Messages

Properties to set and configure errors

Presentation components Validation Error Message property

This message is only displayed when the **Check Expression** of an attribute fails. Have in mind, that Check Expressions are only evaluated when the submit button of the presentation is executed. So, the error message will be displayed only when the submit button is pressed and the check expression of any attributes fails.

The messages are displayed, by default, at the left bottom of the screen, in the error area, with the following format:

Attribute Name: Message Error (Check expression fail).

If you want to make the error/warning messages be displayed at the top of the screen, go to the *portal.properties* file and change the global portal property to *false*:

```
# Show presentation error messages top or bottom
fuego.portal.XObjects.showMsgInBottom=true
```

Style and Format properties for Error messages

This properties have to be configured at presentation level.

- Background color,
- Foreground color,
- Font size,
- Font type.
- Pop-up Client Errors, this property is used to show the client error in a popup window or in the error area. Pop-up client errors are available only for "Date TimePicker" and "Textfield" components.
 - True: The client errors are shown in a popup window,

- False: The client errors are shown in the error Area


User Error and Warning Messages

Custom messages can be displayed by using the standard methods **showError** and **showWarning**.

Example using the showError method

```
showError this  
  using error = "Error: E-mail address format not valid."
```

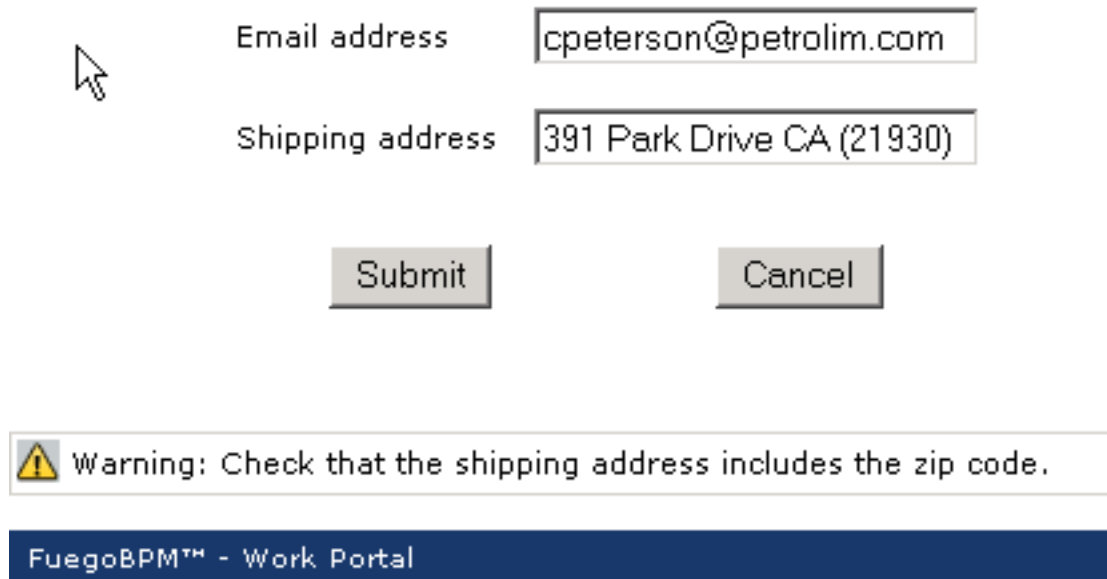
Email address	<input type="text" value="cpeterson@petrolin"/>
Shipping address	<input type="text"/>
<div><input type="button" value="Submit"/> <input type="button" value="Cancel"/></div>	

 Error: E-mail address format not valid.

FuegoBPM™ - Work Portal

Example using the showWarning method


```
showWarning this  
  using warning = "Warning: Check that the shipping  
    address includes the zip code."
```

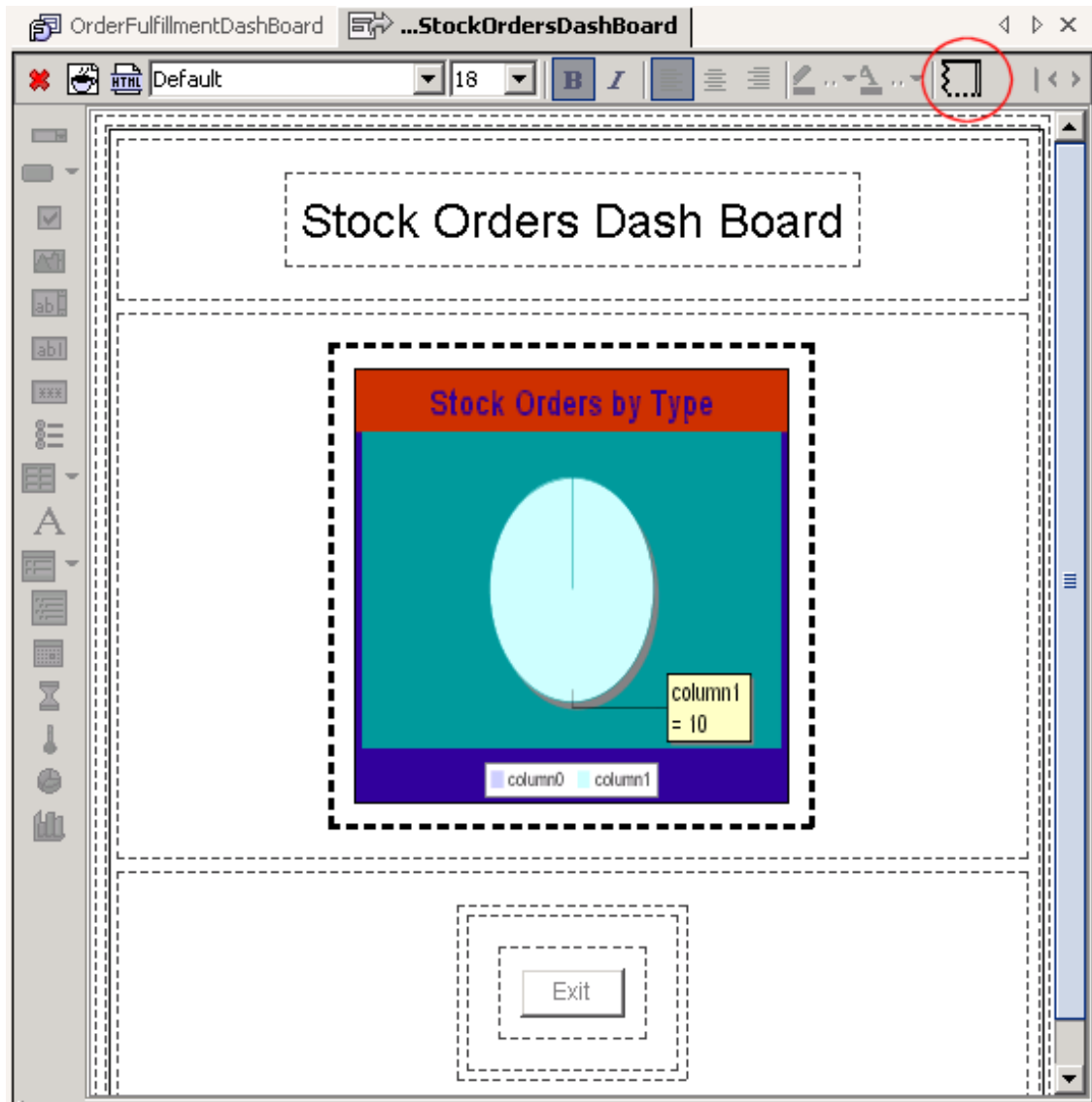


A screenshot of a web form. On the left, a mouse cursor points towards the form fields. The form has two text input fields: 'Email address' containing 'cpeterson@petrolim.com' and 'Shipping address' containing '391 Park Drive CA (21930)'. Below these fields are two buttons: 'Submit' and 'Cancel'. At the bottom of the form, there is a yellow warning icon followed by the text 'Warning: Check that the shipping address includes the zip code.' Below the warning message is a dark blue footer bar with the text 'FuegoBPM™ - Work Portal'.

Presentation Components Border Editor

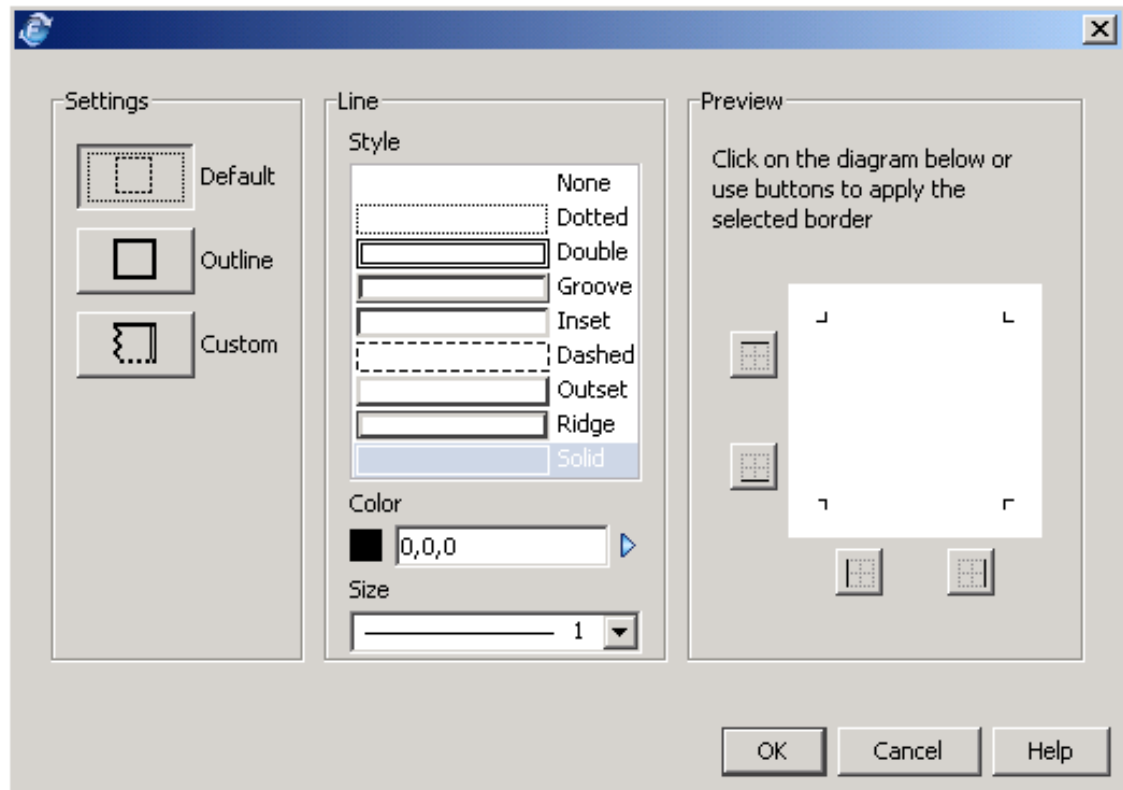
The *border editor* allows to define properties to presentation components and containers in a quick and easy way.

The *border editor* icon is enabled in the presentation layout edition toolbar when one or more presentation elements, components and/or containers are selected. It is enabled only when the selected element has border properties. The icon that identifies it is , in the layout edition bar.



Border Editor Panel

The *border editor panel* is divided in three sections. *Settings* configuration, *Line* properties and borders *Preview*.



Settings Section

The *Settings* section contains three buttons to define the type of configuration.

- The **Default** button sets the element borders with:
 - Border color: black, RGB [0, 0, 0]
 - Border type: None, and
 - Border width: 1.
- The **Outline** button lets define for the four element borders the same settings.
- The **Custom** button lets define different settings for each element border, top, bottom, left and right. This option is only available

for those components and containers that have the four borders properties to be set individually. If the element border property is one setting shared by all the borders, the **Custom** button remains disabled.

Line Section

In this section you can select line *style*, *color* and *width*.


Style

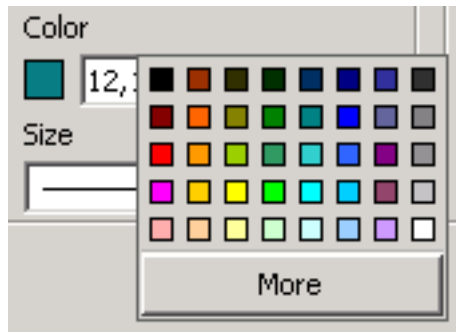
The available styles are:

- None,
- Dotted,
- Double,
- Groove,
- Inset,
- Dashed,
- Outset,
- Ridge, and
- Solid.

Refer to the section Designing a Form for example on how each one looks.

Color

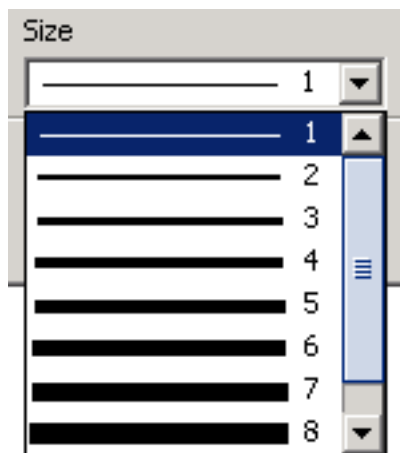
To set a color, type the RGB values by editing them, or click the  icon on the right and select one of the pallet.



Or select a custom color by clicking **More**.

Size

Select the line size from the combo box options.



Preview section

The preview section of the panel shows the settings you are choosing.

It also lets you directly click on the diagram or buttons to choose the border to which apply the selection. This only is available when the presentation element allows a **Custom** configuration.

Presentations and the "input" statement

Once the Fuego Object and its presentation are developed, you will want to execute and input data through the presentation in a

BP_method. To do so, use the *input* interactive statement.

The basic invocation is:

```
input presentableFuegoObject
    using selectedPresentation = "Presentation_Name"
```

If you want to get the selected button in a presentation while the presentation is running, use complete invocation. To do this, use the *selection* parameter in the returning clause.

```
input presentableFuegoObject
    using selectedPresentation = "Presentation_Name"
    returning selectedButton = selection
```

where:

presentableFuegoObject: presentable Fuego Object you are executing

selectedPresentation: the name of the presentation

selectedButton: the variable where to store the **selection**. It has to be declared as *String*. The value that this variable takes corresponds to the *returned value* of the method invoked in the selected button. For example, if the button is defined as a *submit* the value the variable will hold is *submit*.

If you define a button as an *action button*, it has a method that invokes the submit method with another string as returned value. for example, an *approve* button that returns "submit(approved)". In this case, you have to check the returned value (hold in the **selectedButton** variable) to the string approved. For further information see related sections:

- Fuego Object Presentations
- Fuego Object Methods - *submit*
- Fuego Object Methods - *action*
- Fuego Objects Examples - Project Example Catalog - Fuego Object Methods - Action methods

Customizing Presentation with Cascading Style Sheets CSS

Format customization on presentations using CSS is possible as FuegoBPM is REC-CSS1-19990111 (<http://www.w3.org/TR/REC-CSS1.html>) compliant.

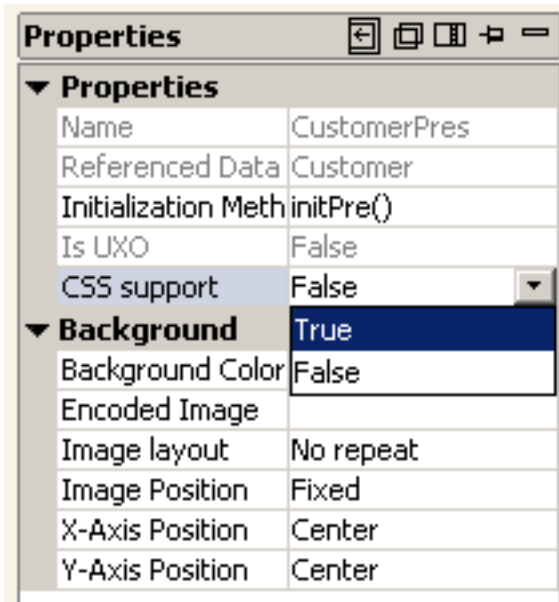
If you want to change the configuration to appear in a style that reflects your business' look and feel, you can modify or add a CSS for you presentations. CSS allows better control over the appearance and positioning of elements on a web page.

Customizing presentations using cascading style sheets gives you advantage to design your Fuego Object presentations and unify the format of those that require formatting in very simple and quick way.

It is only required to define in a style sheet the format characteristics in a tag for each possible presentation container or component and set the presentation as CSS enabled.

Enabling CSS Support in a Presentation

Components properties are enabled only when the property *CSS Support*, at presentation level is set to **TRUE**.



When this property is set to TRUE, new CSS properties are enabled for all of the presentation elements. Including the presentation itself.

CSS Properties:

When the *CSS support* property is enabled at presentation level, new properties have to be set in presentation containers and components, according to the CSS customization you want to configure.

CSS property	Enabled for
<i>CSS filename</i>	Presentation
<i>CSS class</i>	All presentation components and containers
<i>Mixed CSS</i>	All presentation components and containers
<i>CSS header class</i>	Group & Repeatable Section components
<i>CSS paging class</i>	Group & Repeatable Section components
<i>CSS index class</i>	Group & Repeatable Section

CSS property	Enabled for
	components

CSS Class Property

Each presentation element has a related CSS class, which will be included in the CSS file defining the customized properties. The names given by default are the following:

CSS class	Presentation element
<i>fo_presentation</i>	Presentation
<i>fo_presentation_error</i>	Presentation Error properties
<i>fo_table</i>	Table
<i>fo_cell</i>	Cell
<i>fo_combo</i>	Combo
<i>fo_button</i>	Button
<i>fo_check</i>	Check
<i>fo_image</i>	Image
<i>fo_multiline</i>	Multiline Text
<i>fo_text</i>	Text & Password
<i>fo_radio</i>	Radio
<i>fo_label</i>	Label
<i>fo_date</i>	Date time picker
<i>fo_interval</i>	Interval
<i>fo_group</i>	Group & Repeatable Section
<i>fo_group_header</i>	Group & Repeatable Section Header
<i>fo_group_index</i>	Group & Repeatable Section Index
<i>fo_group_paging</i>	Group & Repeatable Section Paging
<i>fo_group_even_tr</i>	Group & Repeatable Section Even Row
<i>fo_group_odd_tr</i>	Group & Repeatable Section Odd Row

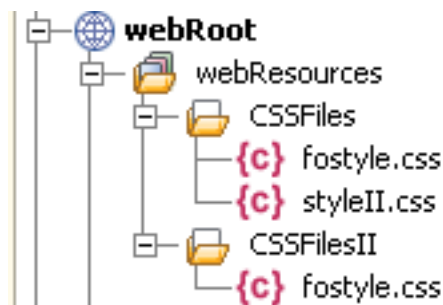
CSS class	Presentation element
<i>fo_link</i>	Anchor
<i>fo_iframe</i>	Inner Frame

You can redefine this class name for those components you need. For example, suppose that all labels in your presentations have to be set with the *font-size* **large** but the label that corresponds to the presentation title has to be set as **xx-large**. You can define a new css class named, for example *fo_label_title*, that will be related only in the label component that corresponds to the presentation title. This new CSS class, must be defined in the CSS file. This gives the developer the flexibility to define a base look and feel for properties shared among most of occurrences of a component and define particular cases with a different CSS class. You could even change all the default names to new names that are more suitable for you. It is not required to keep the default names. See the **Examples** section to learn how this is implemented.

CSS Filename Property

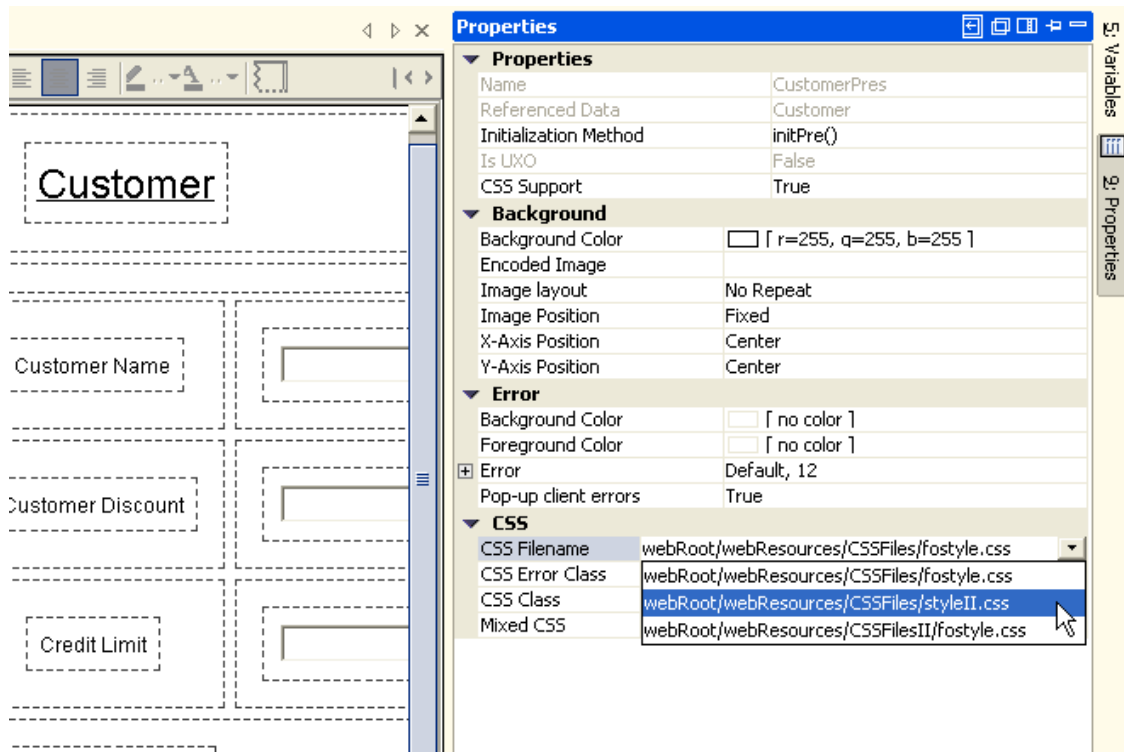
This property holds the name of the CSS file that is applied to customize the presentations.

The css file has to be cataloged in the Web Root entry of the Project structure.



One of the css files catalogued have to be indicated in the CSS Filename property. Choose one of them from the popuplist of the

property.



File Structure

The file has to contain a style definition to each CSS class you are applying in your presentations. For example, suppose that you keep the default CSS class names and that you are defining a set of properties for each component and container of the presentation. Have an entry for each CSS class defined in the default given list, as follows:

```
...

/* Style of button component*/
.fo_button {
color: navy;
background-color: ivory;
font-style: italic;
font-size: small;
font-family: Arial, "Arial";
}
```



```
/* Style of cell component*/
.fo_cell {
border-width: thin;
border-style: solid solid solid solid;
border-color: indigo;
background-color: lavender;
}
/* Style of check component*/
.fo_check {
color: indigo;
background-color: ivory;
}

...
```

Mixed CSS Property

Mixed CSS property is enabled for presentation components and containers. This property allows to design the presentation look and feel combining the settings in the presentation and those in the CSS file indicated to apply.

It is another way to define special styles to a component in a particular presentation.

If a component *Mixed CSS* property is set to true, the settings applied to it are the ones defined in the presentation, overwriting the settings configured in the corresponding CSS class in the CSS file.

See an example of *Mixed CSS* property in the next section.

Examples

Having the following CSS File:

```
/* Style of button component*/
.fo_button {
color: navy;
background-color: ivory;
font-style: italic;
```

```
font-size: small;
font-family: Arial, "Arial";
}
/* Style of cell component*/
.fo_cell {
border-width: thin;
border-style: solid solid solid solid;
border-color: indigo;
background-color: lavender;
}
/* Style of check component*/
.fo_check {
color: indigo;
background-color: ivory;
}
/* Style of combo component*/
.fo_combo {
color: indigo;
background-color: ivory;
font-size: small;
font-family: Arial, "Arial";
}
/* Style of date component*/
.fo_date {
color: indigo;
background-color: ivory;
font-size: small;
font-family: Arial, "Arial";
}
/* Style of image component*/
.fo_image {
}
/* Style of interval component*/
.fo_interval {
color: indigo;
font-size: small;
font-family: Arial, "Arial";
}
/* Style of group component*/
.fo_group {
background-color: indigo;
}
/* Style of group's header component*/
.fo_group_header {
font-family: Arial, "Arial";
background-color: indigo;
color: lavender;
}
/* Style of group's index component*/
.fo_group_index {
font-style: oblique;
background-color: indigo;
```

```
color: lavender;
}
/* Style of group's paging component*/
.fo_group_paging {
}
/* Style of label component*/
.fo_label_title {
color: indigo;
font-style: italic;
font-size: large;
font-family: Arial, "Arial";
}
/* Style of label component*/
.fo_label {
color: indigo;
font-size: small;
font-family: Arial, "Arial";
}
/* Style of multiline component*/
.fo_multiline {
background-color: ivory;
color: indigo;
font-size: small;
font-family: Arial, "Arial";
}
/* Style of radio component*/
.fo_radio {
background-color: ivory;
}
/* Style of table component*/
.fo_table {
}
/* Style of text component*/
.fo_text {
background-color: ivory;
color: indigo;
font-family: Arial, "Arial";
font-size: small;
}
```

Fuego Object Presentation applying the example CSS file

Address C:\Documents And Settings\FGC\Local Settings\Temp\index_tmp40016.html Go Links >>

Pricing List

Pricing List valid from	<input type="text"/>	<input type="text"/>	<input type="button" value="Get All Items"/>								
Pricing list description	<input type="text"/>		<input type="button" value="Get Items Last List"/>								
<div> </div> <table border="1"> <thead> <tr> <th></th> <th>Item</th> <th>Description</th> <th>Price</th> </tr> </thead> <tbody> <tr> <td>1</td> <td><input type="text"/></td> <td><input type="text"/></td> <td><input type="text"/></td> </tr> </tbody> </table>					Item	Description	Price	1	<input type="text"/>	<input type="text"/>	<input type="text"/>
	Item	Description	Price								
1	<input type="text"/>	<input type="text"/>	<input type="text"/>								
<input type="button" value="Store"/>		<input type="button" value="Exit"/>	<input type="button" value="Reset"/>								

The same presentation with CSS Support not set

Address C:\Documents And Settings\FGC\Local Settings\Temp\index_tmp40021.html Go Links >>

Pricing List

Pricing List valid from	<input type="text"/>	<input type="text"/>	<input type="button" value="Get All Items"/>								
Pricing list description	<input type="text"/>		<input type="button" value="Get Items Last List"/>								
<div> </div> <table border="1"> <thead> <tr> <th></th> <th>Item</th> <th>Description</th> <th>Price</th> </tr> </thead> <tbody> <tr> <td>1</td> <td><input type="text"/></td> <td><input type="text"/></td> <td><input type="text"/></td> </tr> </tbody> </table>					Item	Description	Price	1	<input type="text"/>	<input type="text"/>	<input type="text"/>
	Item	Description	Price								
1	<input type="text"/>	<input type="text"/>	<input type="text"/>								
<input type="button" value="Store"/>		<input type="button" value="Exit"/>	<input type="button" value="Reset"/>								

How to set an image as a logo

You can archive it using the *background* class name. Set it to something like:

```
.fo_presentation {  
background-image: url("../images/FUEGO.gif");  
background-repeat: no-repeat;  
background-position: top;  
color: white;  
padding-top: 100px;  
}
```

Where the *padding-top* is the approximate size of the image. It has to be set to avoid the image being overlap with the rest of presentation.

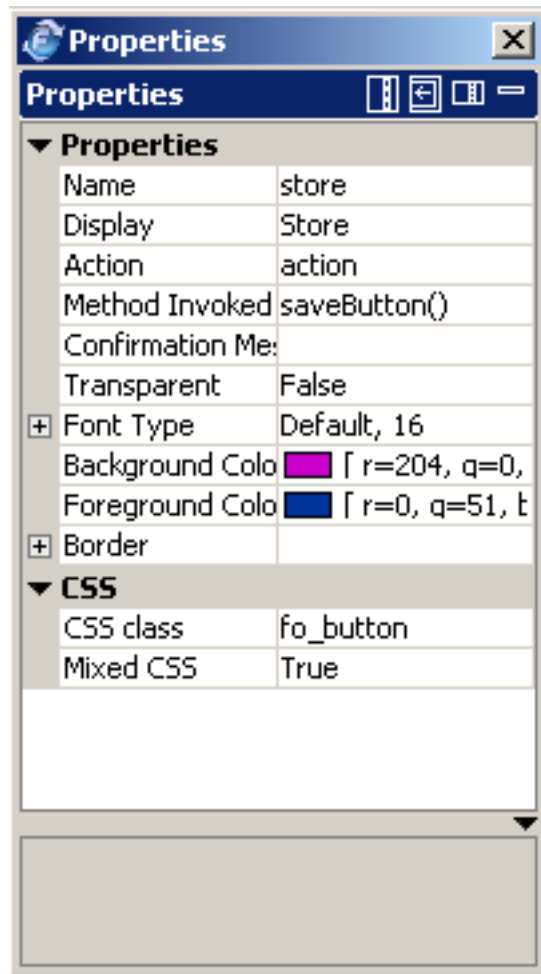
The example makes the image *FUEGO.gif* to appear centered in the top of the presentation only once.

Mixed CSS property example

Having the presentation used before, suppose that you want the *Store* button, to be different from the others to make it more remarkable.

In this case you still want the CSS file to be applied in the whole presentation but this button has to be different. To do so, you could define an special CSS class as explained before, or, use the *Mixed CSS* property, by setting it to true and setting special configurations to the button. When the presentation is displayed, the CSS file definitions are applied to the whole presentation components and containers except to the *store* button.

The presentation properties are still the same as in the first example, the *Store* button properties are set in this particular example like:



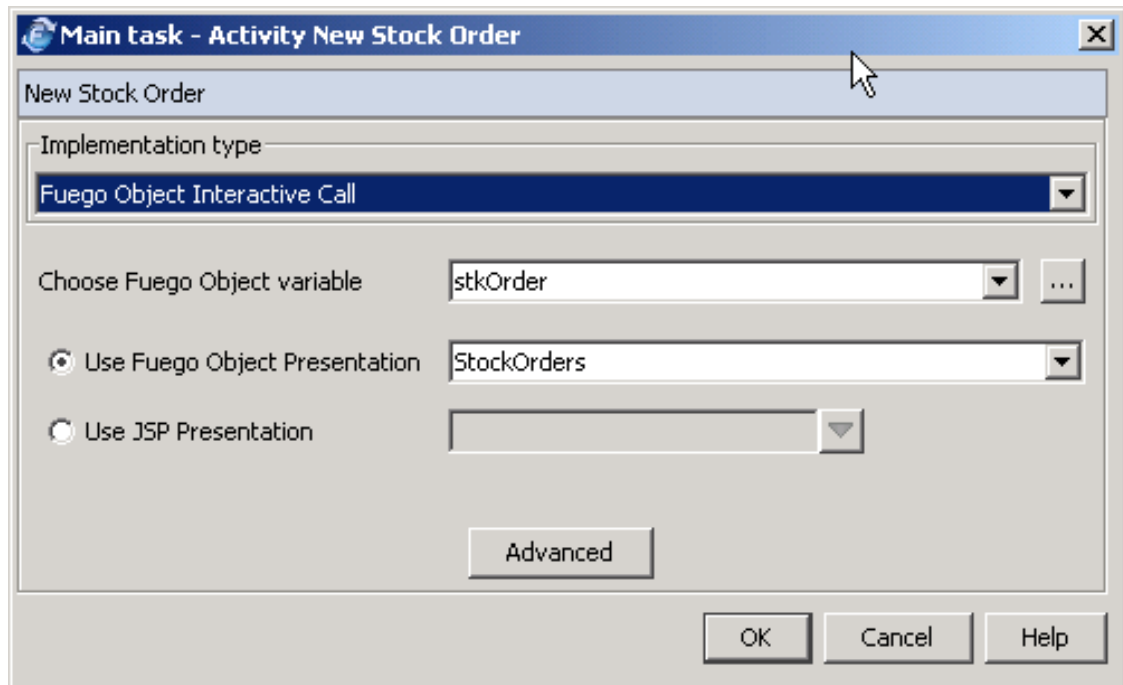
The presentation looks like:

Pricing List								
Pricing List valid from	<input type="text"/>	<input type="button" value="Get All Items"/>						
Pricing list description	<input type="text"/>	<input type="button" value="Get Items Last List"/>						
<table border="1"><thead><tr><th>Item</th><th>Description</th><th>Price</th></tr></thead><tbody><tr><td><input type="text"/></td><td><input type="text"/></td><td><input type="text"/></td></tr></tbody></table>			Item	Description	Price	<input type="text"/>	<input type="text"/>	<input type="text"/>
Item	Description	Price						
<input type="text"/>	<input type="text"/>	<input type="text"/>						
<input type="button" value="Store"/>	<input type="button" value="Exit"/>	<input type="button" value="Reset"/>						

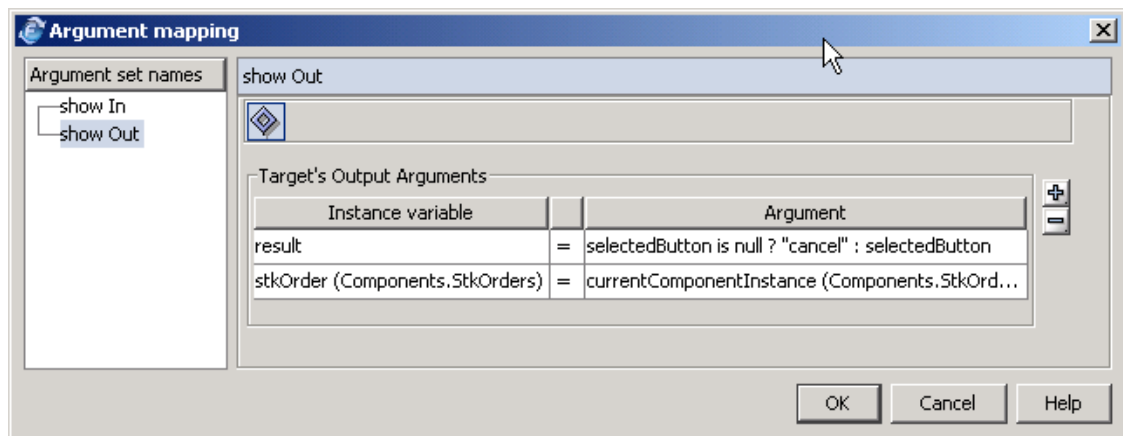
Designing Using Fuego Objects

Using Fuego Objects in Screenflows

An interactive component call in a screenflow that invokes a Fuego Object has to be defined as of type *Fuego Object Interactive Call*. Using a screenflow instance variable of the Fuego Object type, you define which of its presentations will be used in this activity as shown in the picture below.



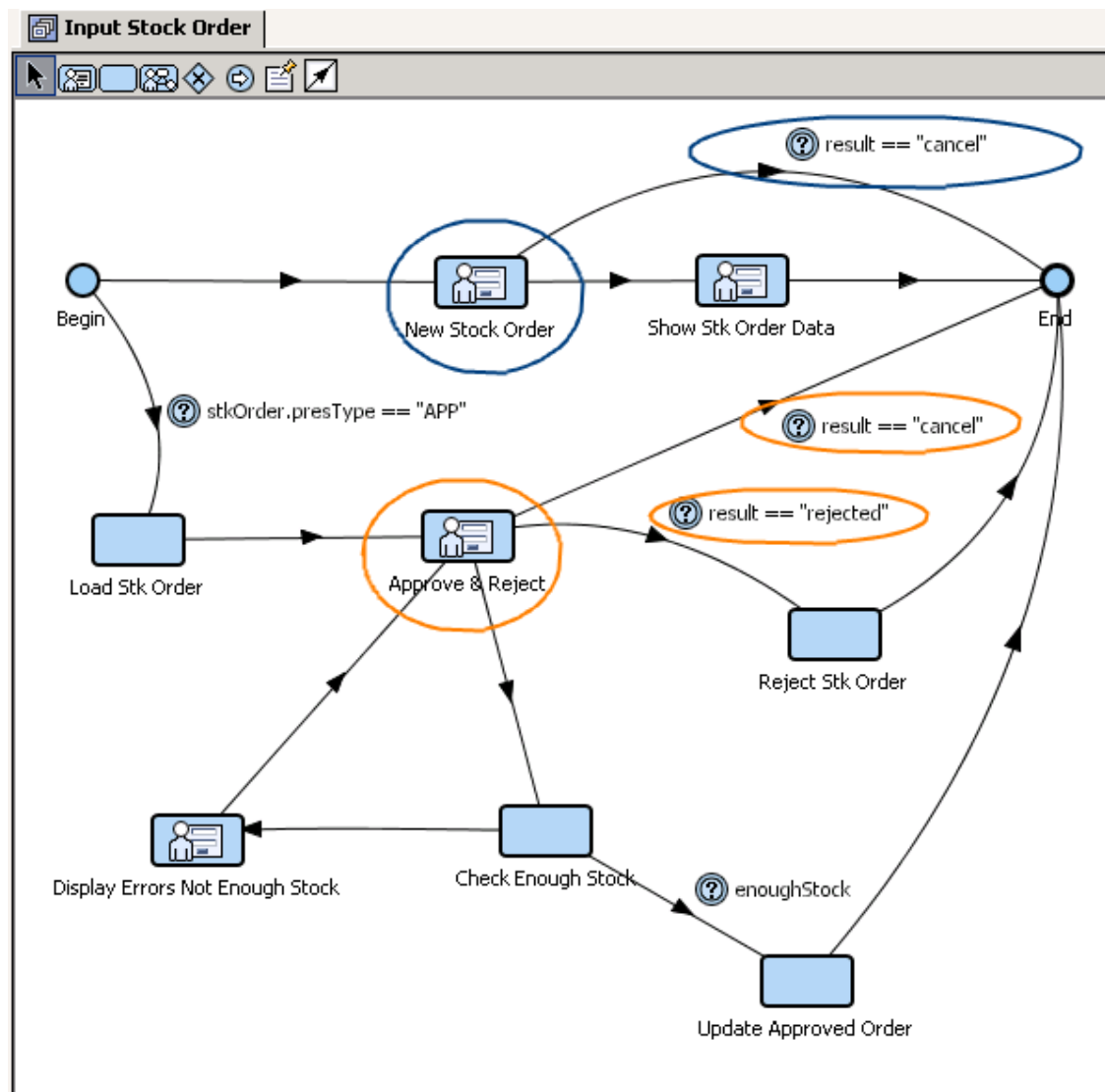
When a Fuego Object presentation is executed, the only way to know what the user selected through the selected button. To know what button was selected by the user and design the screenflow according to it, the *result* predefined variable is automatically filled in the output arguments of the interactive call activity with the value of the returned button. To look at its definition or modify it, open the *Advanced* window in the interactive call main task dialog. And then, click the *show Out* arguments set, as shown in the following picture.



As you already know, the returned value when the *cancel* button is

selected is **null**. In the *result* variable setting, the null value return when the cancel button is selected is replaced by the string **"cancel"**.

Then you have to define the outgoing transitions based on the *result* variable value. An example is shown in the picture below.



Please, see the Screenflows topic help for more details or refer to Project Example - Processes to see more screenflow implementations using Fuego Objects.

Using Fuego Objects in Procedures

Please, see the Procedures topic help for details.

Invoking a Fuego Object in a Component Task

Please, See the Tasks topic help for details. See Fuego Objects Examples for examples related to Fuego Objects.

Creating a Fuego Object from Methods

- To create a Fuego Object in a method, add the invocation to the constructor by dragging and dropping the Fuego Object constructor you want to create to the Method Editor panel.

Handling Fuego Objects

Exporting a Fuego Object

You can export a Fuego Object to be used by other developers. The exported Fuego Object is saved in XML file format or in ZIP files (when the Fuego Object contains group attributes) that can be sent by email or saved on a network connected server. The Fuego Object can then be imported into another Project Catalog.

To export a Fuego Object:

1. Right-click on the Fuego Object to export and select **Export Fuego Object** from the shortcut menu. # The **Select Fuego Object file...** dialog box is displayed.
2. Browse and select the location where you want to save the exported file. Enter a file name in the File name field.
3. Click the **Open** button.

Cataloging a Fuego Object

To catalog a Fuego Object, right-click on the module and select **Catalogue Component** and **Fuego Object** from the menu options.

Importing a Fuego Object

Fuego Objects can be shared between component catalogs. They can be exported from one catalog and imported into another.

To import a Fuego Object


1. Select the option **Import** in the main **File** menu and then **Fuego Object**.
2. The **Select Object file...** dialog box is displayed. Browse the location of the Fuego Object and select it.
3. Click the **Open** button. The Fuego Object is listed under the module.

When importing a Fuego Object into the catalog, it may not be imported into the selected module. It depends on the source version of the component you are importing.

If you are importing a component from a prior version the *zip*, *xcdl* or *xml* file does not include the module information, that is why it is added to the selected module.

Components files of this same version, can be *xcdl* files without module information or *zip* files with module information, that's why the first ones are added to the selected module and the second ones are added to the original module (if the module does not exist it is created).

Note

 If you are importing or cataloging a Fuego Object of previous versions, if it has a group attribute some special actions will take place to make the

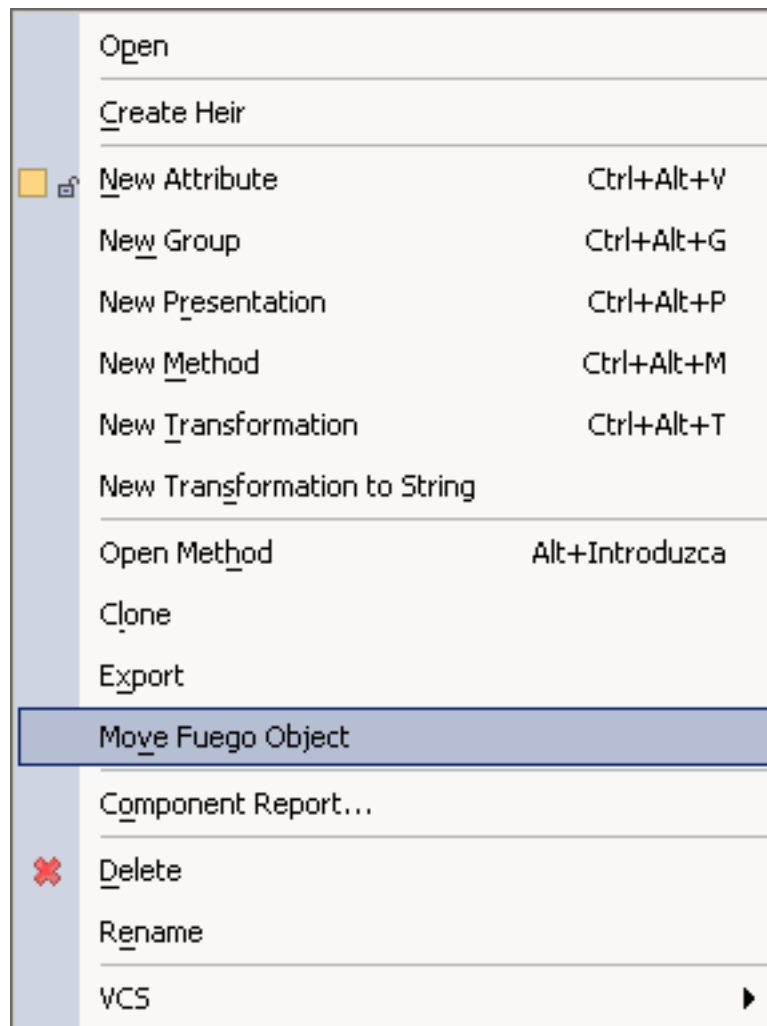
Fuego Object compatible with the present version. Please refer to Fuego Object Group Attributes

Moving a Fuego Object

Fuego Objects can be moved from one module to another.

To move a Fuego Object to a new module:

1. Right-click on the Fuego Object in the components tree and select the **Move Fuego Object** option.



2. The **Module List** dialog is displayed. Select the module to which you want to move the object.



3. Enter the name you want to give to the Fuego Object in its new location. Click **OK**.

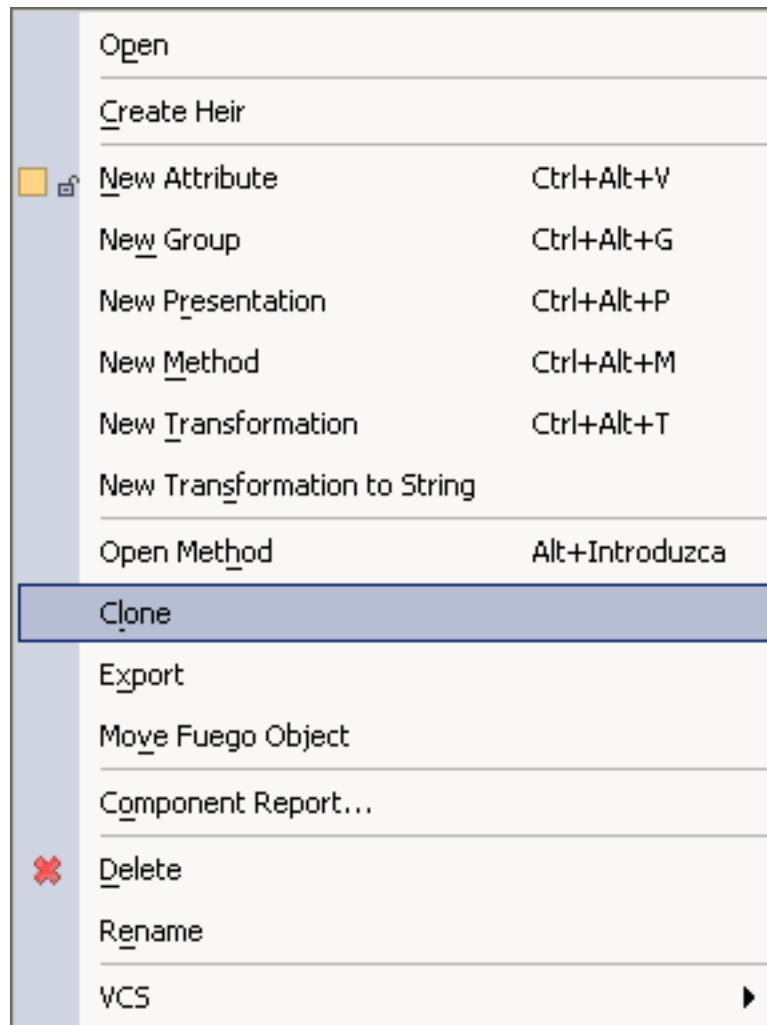
The Fuego Object is moved to the new module and is deleted from the module where it was originally located.

Cloning a Fuego Object

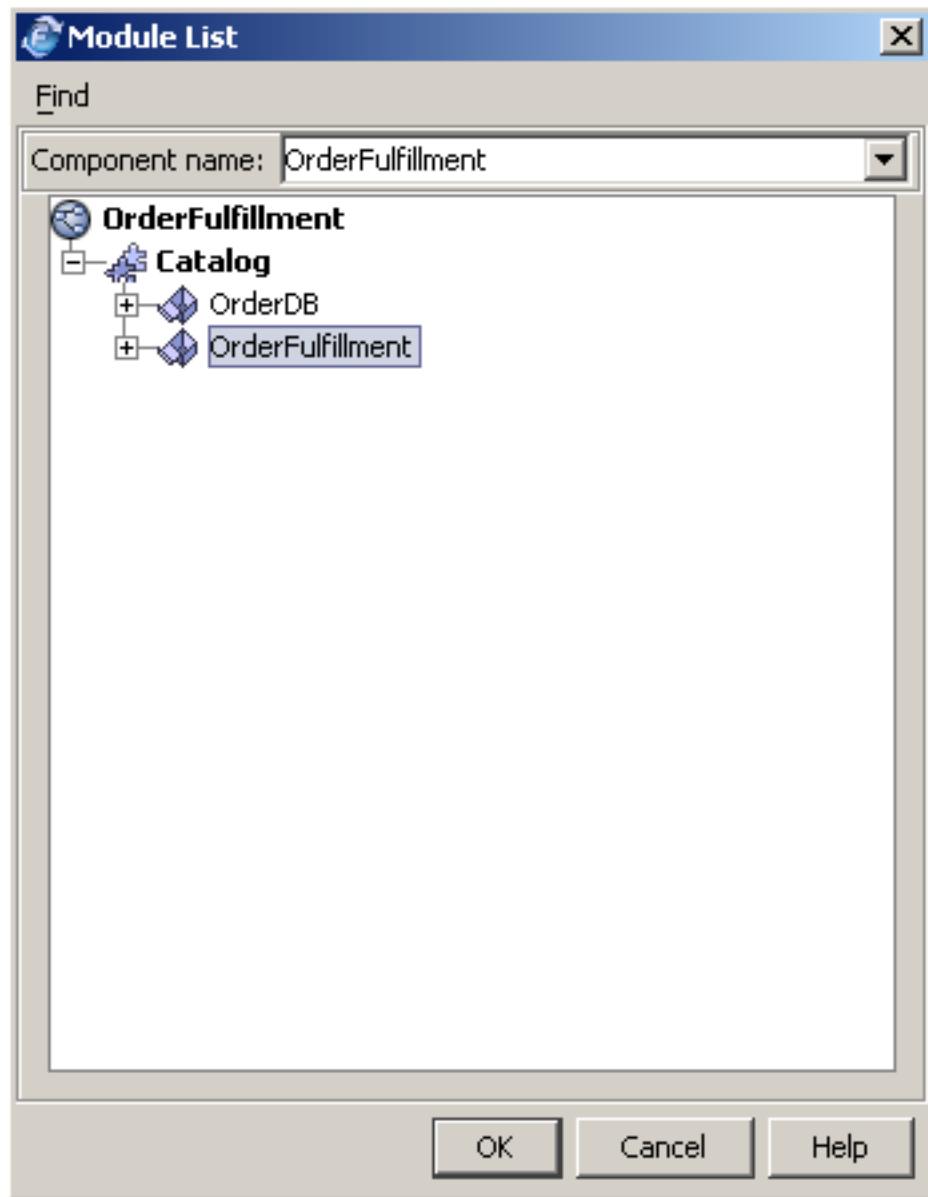
Fuego Objects can be cloned and stored in the same or different modules.

To clone a Fuego Object:

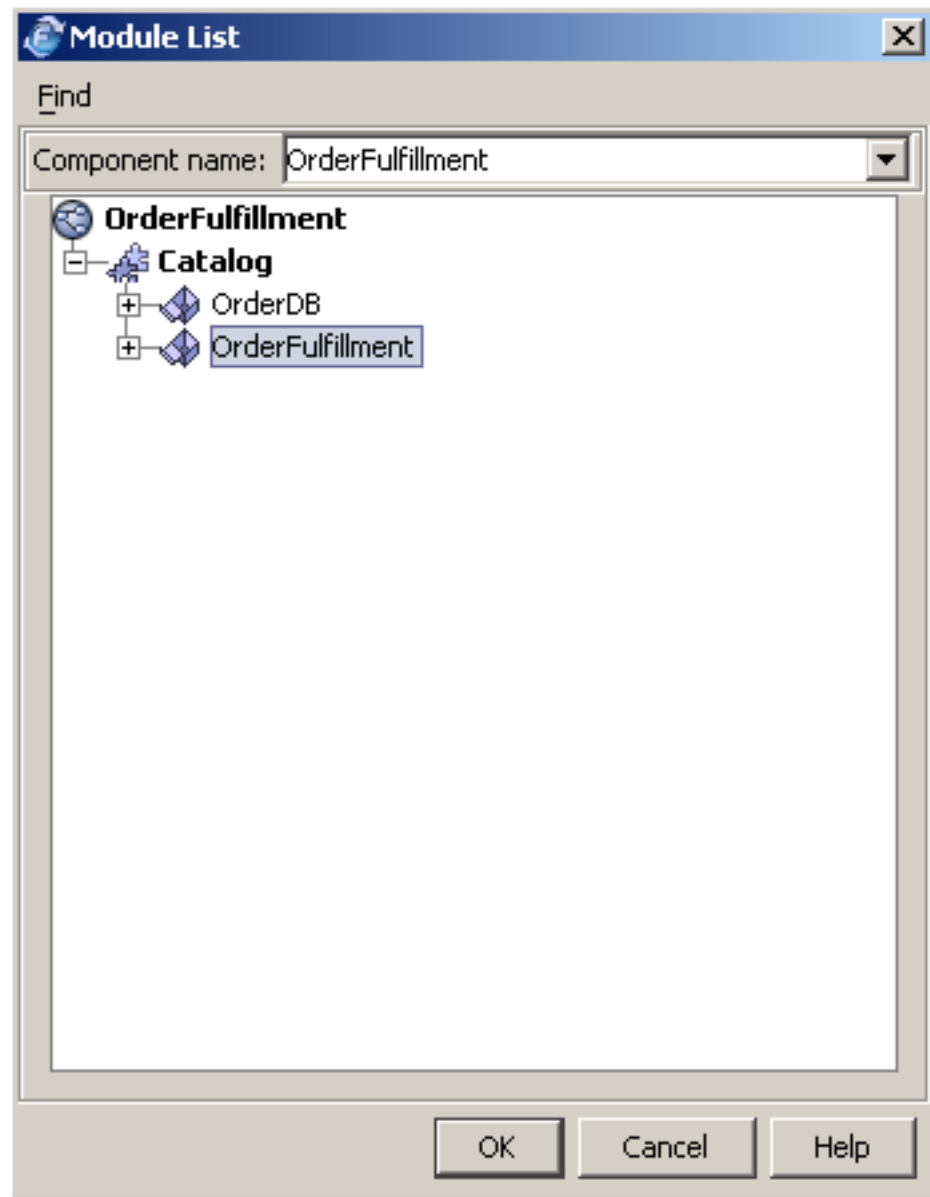
1. Right-click on the Fuego Object in the components tree and select the **Clone** option.



2. The **Module List** dialog appears. Select the module where you want to store the clone.



3. Type the name you want to give to the Fuego Object clone and click **OK**.



Versioning Fuego Objects

FuegoBPM Studio supports a Version Control System (VCS) to provide developers with the ability of versioning not only Projects and processes but also the Project Catalog to which Fuego Objects belong.

VCS is very useful to keep FuegoBPM projects administrated. This

includes processes, components and special files. VCS is a tool every developer needs for code versioning.

The Version Control System implements a client to perform version control operations like add, remove, commit, update into a common repository. FuegoBPM Studio is designed to operate specifically with the CVS version control system in this version and will support Source Safe and others in further versions. The providers are available by installing the corresponding plug-in.

See VCS - Version Control System for further detailed information on how to use VCS for Fuego Objects.

Fuego Objects as Instance Variables

If you call a Fuego Object presentation directly in Business Process Method, the values of the attributes will not automatically persist (be kept) once this Method task is complete. You must assign instance variables to hold the information. Assigning multiple instance variables can be time-consuming and cumbersome when you have a Fuego Object that contains several attributes.

In order to use the attributes of a Fuego Object throughout the life of an instance in a process, you must declare it as an instance variable. The Fuego Object will be referred to in the Business Process Method using the instance variable name instead of the Fuego Object name.

Note



Be careful when using this practice, as the Fuego Object will persist as part of the instance. Bear in mind the size of the Fuego Object that will be stored in the instance variable. You may be persisting information in the process instance instead of in the correct data container.

Use a Fuego Object as an instance variable

- When Fuego Object contents must be shared between different


processes, you should be careful with their size.

Don't use a Fuego Object as an instance variable

- When Fuego Object contents could be retrieved from a data container (e.g., a database).
- When Fuego Object contents could be stored in a data container (e.g., a database), you should create a Fuego Object method to do so.
- When the Fuego Object is only used to display process information. Create a Fuego Object method instead.

If the Fuego Object that has to be stored in an instance variable is 10KB or more, it should be categorized as a Separate Instance Variable. Separated Instance Variables are limited to 2 Mb. For more information, see the Instance Variables section.

To declare an instance variable to represent a Fuego Object:

1. Open the Method Editor and select the **Variables** option.
2. Select the Instance variable category and click the add button . Enter the variable name with the keyboard. For example, *orderVar*.
3. Click the **Browse** button and select the Fuego Object in the Types browser dialog box. Click **OK**. The type is now the Fuego Object and its module. In the example below, *OrderFulfillment* is the module name and *Order* is the name of the Fuego Object.

You are now ready to use the Fuego Object instance variable in your Business Process Method.

Using JSP and Fuego Objects

In order to use the Fuego Tag Library feature you must have previous knowledge about JSP, Java Tag Library and JSTL (JavaServer Pages Standard Tag Library).

Java Servers Pages can be used as an interface between the end user and the Fuego Object. This is really convenient if you are already using JSPs in your company's site. Because you will have your old interface running just by replacing the invocations to bean methods by invocation to Fuego Objects methods and attributes.

It is advisable to encapsulate as much behavior as possible in the Fuego Object and leave the JSP as a media to display or request information.

The main tasks involved in designing a process which uses Fuego Objects with JSPs are:

- Creating the Fuego Object
- Creating the screenflow
- Invoking the screenflow from the main process
- Creating the JSP
- Importing the JSPs
- Importing web resources

Creating the Fuego Object

For detailed information, please refer to Implementing Business Objects Using Fuego Objects.

Creating the screenflow

1. Right click into the process node, and choose "New Screenflow".
2. Add an interactive component call.
3. Right click over the interactive component call and select "Main task".
4. Choose "Fuego Object interactive call" as the implementation type.
5. Where it says "Choose the Fuego Object variable" select the Fuego Object you want to use, this should be an instance variable of the screenflow.
6. Select "Use JSP Presentation".
7. Enter the name of the imported JSP to be used or click over the arrow and select it from the dialog that will be displayed.
8. Make sure you have defined the right argument mappings.

Invoking the screenflow from the main process

To learn more about screenflows, please refer to Screenflows.

Creating the JSP

The JSP should be created in any external editor outside the Studio. Once it has been created it should be imported into the project including its resources. The JSP is a regular one containing the following special features which allows you to use Fuego's tag libraries in order to access the Fuego Object's attributes and methods:

Make you have included the following line at the beginning of the JSP:

```
<%@ page session="true"%>
```

At the very beginning of the JSP file you should import the fuego tag library and the java standard tag library (advisable but not required) as follows:

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<%@ taglib uri="http://fuego.com/jsp/ftl" prefix="f" %>
```

Note that c and f could be any prefix you choose. We have chosen c for jstl and f for Fuego's tags libraries and this convention will be respected in all the examples.

Using the resources: To make reference to your resources you should use the tag **webResources** see Available Tags below.

Available Tags:

postResults

Description: This tag is used to post the resulting Fuego Object back to the screenflow, or to pass them into the following JSP by using an optional attribute to indicate an URL where to forward the results to.

Usage:

- **post back to the screenflow**

```
<form method="post" name="form" action="<f:postResults />">
```

- **post to another URL**

```
<form method="post" name="form" action=
    "<f:postResults forwardUrl="nextUrl"/>">
```

where *nextUrl* is the URL where the Fuego Object resulting of the execution of this JSP is going to be forwarded to.

fieldName

Description: This tag generates a String corresponding to the name of a Fuego Object's attribute. It should be used in the name attribute of an HTML component, in order to use the Fuego Object attribute with the component. This automatically loads the values entered in this component into the Fuego Object. It has an optional attribute *onlyName*, which if set to false does not concatenate the word name to the generated String. This is useful for components that use the name but do not need the word name before it.

Usage:

If you want to access the name of the attribute *telephoneNumber* you would use this tag in the following way:

```
<f:fieldName att="personFo.telephoneNumber"/>
```

Where *personFo* is an instance (the instance variable declared in the screenflow) of a Fuego Object which has an attribute named *telephoneNumber*. The String generated by the use of this tag can be used in any HTML component that requires the name attribute.

Eg.:

```
<textarea <f:fieldName att="personFo.telephoneNumber"/>>
<textarea/>
```

Generated HTML:

```
<textarea name="nameGeneratedByFuego"></textarea>
```

EL(Expression Language) can be used to access the Fuego Object variable in exactly the same way you do in Java or FBL.

fieldValue

Description: This tag generates a String corresponding to the value of a Fuego Object's attribute. It should be used in the value attribute of an HTML component, in order to use the Fuego Object attribute with the component. It has an optional attribute `onlyValue`, which if set to `false` does not concatenate the word `value` to the generated String. This is useful for components that use the value but do not need the word `value` before it, as shown in the text area example below.

Usage:

```
<f:fieldValue att="personFo.telephoneNumber" onlyValue=true/>
```

Where `personFo` is an instance (the instance variable declared in the screenflow) of a Fuego Object which has an attribute named `telephoneNumber`. The String generated by the use of this tag can be used in any HTML component that requires the value attribute.

```
<textarea <f:fieldName att="personFo.telephoneNumber"/>>
  <f:fieldValue att="personFo.telephoneNumber" only=true/>
</textarea>
```

```
<textarea/>
```

Generated HTML:

```
<textarea  
  name="nameGeneratedByFuego">  
  77788888  
</textarea>
```

Where "77788888" is the value of the field `telephoneNumber`.

EL(Expression Language) can be used to access the Fuego Object variable in exactly the same way you do in Java or FBL.

field

Description: This tag generates a String corresponding to the name and value of a Fuego Object's attribute. It should be used in HTML components that have the attributes `name` and `value`, like the input component. This automatically loads the entered values in the JSP into the Fuego Object. This tag combines the two tags described before making the code easier to read.

Usage:

```
<f:field att="personFo.telephoneNumber" />
```

where *personFo* is the name of the Fuego Object variable declared in the Fuego Object and *telephoneNumber* is the name of the Fuego Object's attribute. Eg:

```
<input type="text" id="telephoneNumber"  
  <f:field att="personFo.telephoneNumber" />/>
```


this example shows an input field, whose value will be stored automatically into the attribute *telephoneNumber* of the instance variable *personFo* declared in the screenflow.

The same result could have been achieved by combining the tags *fieldName* and *fieldValue* in the following way:

```
<input type="text" id="telephoneNumber"
      <f:fieldName att="personFo.telephoneNumber"/>
      <f:fieldValue att="personFo. telephoneNumber"/>/>
```

Generated HTML

```
<input type="text" id="telephoneNumber" name="nameGeneratedByFuego" value="77788888"/>
```

Where "77788888" is the value of the field *telephoneNumber*.

EL(Expression Language) can be used to access the Fuego Object variable in exactly the same way you do in Java or FBL.

invoke

Description: This tag is used to invoke a method over the Fuego Object used in the JSP. This method may return a value, which can be stored into any variable declared in the JSP. By default the scope of this attribute will be page, however it can be changed through an optional attribute. The method may receive one or more arguments which can be basic FBL types or complex types (another Fuego Object or Java class).

Usage:

```
<f:invoke var="${fo}" methodName="myMethod"/>
```

Where *fo* is the name of the instance variable Fuego Object declared in the Screenflow, and *myMethod* is the name of the method to be invoked over the Fuego Object. EL(Expression Language) is used to access the Fuego Object variable. You can access Fuego Objects contained in another Fuego Object, exactly in the same way you do in Java or FBL, like:

```
<f:invoke var=${user.history} methodName="myMethod"/>
```

where *history* is an attribute of the Fuego Object user, which is itself another FuegoObject.

- **Method with return value:**

```
<f:invoke var="${fo}" methodName="myMethod"  
        retAttName="myAttribute" retAttScope="Session" />
```

Where *myAttribute* is the name of the variable in the JSP where the returning value will be stored. The optional attribute *retAttScope* can have one of the following values: *Page*, *Request*, *Session* or *Application*.

- **Method with FBL basic type arguments:** Used when the method receives arguments of any of the FBL basic types (String, int, boolean, double, Decimal, Time, Interval, Binary, Any)

```
<f:invoke var="${fo}" methodName="myMethod">
```

```
<f:arg value="28" type="int"/>
<f:arg value="foo" type="String"/>
</f:invoke>
```

The invoked method receives two arguments, the first of type int, and the second of type String.

The nested tag arg is used to pass this method the required arguments, value is the value of the passed parameter and type is a String containing the FBL basic type of the argument

- **Method with Java arguments:**

```
<f:invoke var="{fo}" methodName="myMethod">
  <f:jarg value="<%new Date()%>" className="java.util.Date"/>
</f:invoke>
```

As shown by the tag jarg, the invoked method receives an argument of type java.util.Date, the value of the argument is assigned through the scriptlet <%new Date()%> and the class of the argument is indicated by the property className.

invokel

Description: Has the same functionality of the invoke tag. The only difference is that the arguments are passed in a String which contains their values separated by commas. This tag should be used in simple cases where the Fuego Object does not have an overloaded method with the same number of arguments. This is because it tries to match the method by the number of arguments and if it finds two methods with the same quantity of arguments, it does not deambiguate them. Regarding the returning value accepts the same attributes *retAttName* and *retAttScope* described before for the invoke tag.

Usage:

```
<f:invoke var="{fo}" methodName="myMethod" args="28, foo" >
```

where the String "28, foo" represents the value of the two parameters received by the method.

webResources

Description: Used to access web resources. The value of the `relativePath` attribute is the path to the resource relative to `WebResources` folder.

Eg. If the imported resource is `logo.gif`, which is located in `webRoot/webResources/logo.gif`, the value of `relativePath` should be `"logo.gif"`.

```
<f:webResources relativePath='logo.gif' />
```

Eg. If the imported resource is `css/mycss.css`, which is located in `webRoot/webResources/css/mycss.css`, the value of `relativePath` should be `"css/mycss.css"`.

```
<f:webResources relativePath='css/mycss.css' />
```

Eg. If the imported resource is `js/myjs.js`, which is located in `webRoot/webResources/js/myjs.js`, the value of `relativePath` should be `"js/myjs.js"`.

```
include <f:webResources relativePath='myjs.js' />
```

Usage:

```
<LINK href="
    <f:webResources relativePath='css/exampleStyle.css' />"
    rel="stylesheet" type="text/css">
```

where `css/exampleStyle.css` is the resource imported in `webRoot/webResources/`

```

```

where `img/logo.gif` is the resource imported in `webRoot/webResources/`

Useful tips and common problems:

- You can combine the use of Fuego's tag library together with the use of JSTL (Java Standard Tag Library)
- If you get an error saying that the Fuego Object was not found in the request, make sure that the instance variable declared in the screenflow which invokes the JSP, has the same name you are using in the JSP to access the FuegoObject.
- Where possible try to use the tag field instead of using a combination of the fieldName and fieldValue tags.

Importing the JSP

1. Expand the node `webRoot` in the project tree.

2. Right click over the node *customJSP* in the project tree.
3. Choose *Import JSP*. A dialog where you can browse and select the JSPs to be imported, will appear on your screen. You can import a single file or a complete folder containing the JSPs or both things (i.e. a JSP and one or several folders). After choosing the JSP files you wish to import, press **OK**. The selected files will be copied to a directory named *customJSP* which is located inside the directory *webRoot* in your project file.

Importing web resources

1. Expand the node *webRoot* in the project tree.
2. Right click over the node *webResources* in the project tree.
3. Choose *Import resources*. A dialog where you can browse and select the resources to be imported, will appear on your screen. You can import a single file or a complete folder containing the your resources files or both things (i.e. a file and one or several folders). After choosing the files you wish to import, press **OK**. The selected files will be copied to a directory named *webRoot* which is located inside the directory *webRoot* in your project file.

Iterating Fuego Object Group attribute in a JSP

Using the standard library of tags you can implement the iteration. See example below:

Table example, Fuego Object group name : *telephone*, with two attributes: *type* and *number*.

```
<table>
```

```
<c:forEach var="telephone" begin="0"
items="\${person.telephones}" varStatus="status">
<tr>
<td><input type="text" <f:field att="person.telephones
[\${status.index}].type"/>/></td>
<td><input type="text" <f:field att="person.telephones
[\${status.index}].number"/>/></td>
</tr>
</c:forEach>
</table>
```

Working with attachments

In order to work with attachments the JSP page must be defined as multipart.

```
<form id="fileForm" method="post"
  enctype="multipart/form-data"
  name="fileForm"
  action="<%=URLForAction.postJSPResults(request)%>">
.....
```

There are two possible ways of implementing the usage of attachments with versionable JSP:

1. receiving the attachment as a parameter from the JSP in the screenflow interactive call activity, or
2. assigning the attachment to the Fuego Object in the JSP file definition.

Case I: Receiving the attachment as a parameter

This implementation requires:

Coding the JSP

Define the "attachment" tag of file type. eg.:

```
<input id="attachment" type="file"
      name="attachment" value="">
```

Defining the Screenflow Interactive Call activity

1. Define the activity as "Fuego Object Interactive Call", for your Fuego Object using a JSP Presentation.
2. The argument **attachments** (associative array) becomes available in the show out set of arguments. Assigning it to an instance variable to make it visible in the whole screenflow.

Attachment Information available

The associative array received as argument, holds:

1. *filename*, name of the attached file,
2. *contents*, content of the attached file, and
3. *mediatype*, attached file type.

To make reference to these attributes, the name of the file defined in the JSP page must be used as key index for the array. If the JSP used has the attachment tag defined as in the following example code:

```
<input id="attachment" type="file"
      name="__attachment__" value="">
```


the key indexes for the array would be:

- *attachment_filename*
- *attachment_contents*
- *attachment_mediatype*

In the following FBL code example, the *attachs* instance variable was assigned with the *attachments* out argument in the interactive activity of the invoking screenflow.

```
filename as String = String(attachs["attachment_filename"])
filebytes as Binary = Binary(attachs["attachment_contents"])
```

Case II: Assigning the attachment to the Fuego Object

This implementation requires:

Coding the JSP

1. Define the "attachment" tag using a Fuego Object and an attribute of type binary. In the example below, the Fuego Object name is *fileHolder* and it has a binary attribute named *file*.

```
<input id="attachment" type="file"
      <f:fieldName att="fileHolder.file"/> value="">
```

Defining the Screenflow Interactive Call activity

1. Define the activity as "Fuego Object Interactive Call", for your Fuego Object using a JSP Presentation.
2. No argument matching is required

Defining the Fuego Object

1. The Fuego Object referenced from the JSP must have an attribute defined of type Binary. This attribute will be assigned with the content of the attached file.
2. FuegoBPM looks for attributes in the Fuego Object defined with the names:
 - a. *filename*, will be referenced like : *fileHolder.file_filename*
 - b. *filemediatype*, will be referenced like : *fileHolder.file_filemediatype*

Examples

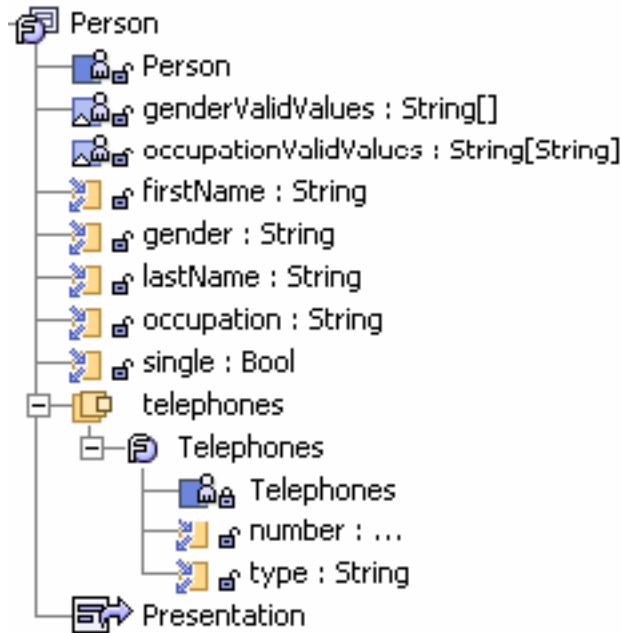
Go to JSP Examples to find usage examples.

Examples using JSP and Fuego Objects

Basic Examples

There is a complete example, which shows the use of JSPs combined with Fuego Objects in the *sample* directory of your FuegoBPM Studio's installation. The project name is *FoodDelivery.fpr*.

In this section you will find useful examples about using different HTML form components. In all the examples we will use as the JSP FuegoObject a person of type Person.



Input Element

```
<input type="text" <f:field att="person.firstName"/>/>
```

or

```
<input type="text" <f:fieldName att="person.firstName"/>
  <f:fieldValue att="person.firstName"/>/>
```

or

```
<input type="text" <f:fieldName att="person.firstName"/>
  value="<f:fieldValue att="person.firstName"
  onlyValue="true"/>" />
```

or

```
<input type="text" <f:fieldName att="person.firstName"/>
value="<c:out value="\${person.firstName }" />" />
```

Checkbox Element

```
<script language="Javascript">
function setInputElementValue(elementId, value){
var theElement = document.getElementById(elementId);
if(theElement != null)
{
theElement.value = value;
}
}
</script>
<c:choose>
<c:when test="\${person.single}">
<input type="checkbox" checked
onclick="setInputElementValue('single', this.checked)">
<input type="hidden" id="single"
value="true" <f:fieldName att="person.single"/>>
</c:when>
<c:otherwise>
<input type="checkbox"
onclick="setInputElementValue('single', this.checked)">
<input type="hidden"
value="false" <f:fieldName att="person.single"/>>
</c:otherwise>
</c:choose>
```

Select element

With Valid Values with no description

```
<f:invoke var="\${person}"
methodName="genderValidValues"
retAttName="genders" retAttScope="Page" />
<select <f:fieldName att="person.gender"/>>
```

```
<c:forEach var="gender" begin="0"
  items="\${genders}" varStatus="status">
<c:choose>
<c:when test="\${person.gender == gender}">
<option value="<c:out value="\${gender}"/>"
  selected="true"><c:out value="\${gender}"/></option>
</c:when>
<c:otherwise>
<option value="<c:out value="\${gender}"/>">
  <c:out value="\${gender}"/></option>
</c:otherwise>
</c:choose>
</c:forEach>
</select>
```

With Valid Values with description

```
<f:invoke var="\${person}"
  methodName="occupationValidValues"
  retAttName="genders" retAttScope="Page" />
<select <f:fieldName att="person.occupation"/>>
<c:forEach var=" occupation " begin="0"
  items="\${ occupations}" varStatus="status">
<c:choose>
<c:when test="\${person. occupation == occupation.key}">
<option value="<c:out value="\${occupation.key}"/>"
  selected="true"><c:out
value="\${occupation.value}"/></option>
</c:when>
<c:otherwise>
<option value="<c:out value="\${occupation.key}"/>">
  <c:out value="\${occupation.value}"/></option>
</c:otherwise>
</c:choose>
</c:forEach>
</select>
```

Group

```
<table>
<c:forEach var="telephone" begin="0"
  items="\${person.telephones}" varStatus="status">
```

```
<tr>
<td><input type="text" <f:field
    att="person.telephones[${status.index}].type"/>/></td>
<td><input type="text" <f:field
    att="person.telephones[${status.index}].number"/>/></td>
</tr>
</c:forEach>
</table>
```

Full JSP

Firstname	<input type="text" value="Nicolas"/>
Lastname	<input type="text" value="Damonte"/>
Single	<input type="checkbox"/>
Gender	<input type="text" value="M"/>
Occupation	<input type="text" value="Consulting"/>
Telephones	<input type="text" value="Home"/> <input type="text" value="4444-2323"/>
<input type="button" value="Submit"/> <input type="button" value="Cancel"/>	

```
<%@ page session="true"%>
<!-- INCLUDING TAC LIBS-->
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<%@ taglib uri="http://fuego.com/jsp/ftl" prefix="f" %>
<html>
<head>
<title>Tutorial</title>
</head>
<body>
<!-- JAVASCRIPT -->
<SCRIPT type="text/javascript">
function submitForm(buttonId){
if(buttonId != null)
{
var isFormOk = true;
if(buttonId == "OK"){
isFormOk = checkForm();
```

```

}
if(isFormOk){
setInputElementValue('selectedButton', buttonId);
document.theForm.submit();
}
}

function setInputElementValue(elementId, value){
var theElement = document.getElementById(elementId);
if(theElement != null)
{
theElement.value = value;
}
}

function checkForm(){
return true;
}
</SCRIPT>
<!-- FORM -->
<form method="post" name="theForm"
      action="<f:postResults />">
<table border="1">
<tr>
<td>Firstname</td>
<td><input type="text" <f:field
      att="person.firstName"/>/></td>
</tr>
<tr>
<td>Lastname</td>
<td><input type="text" <f:fieldName
      att="person.lastName"/> <f:fieldValue
att="person.lastName"/>/></td>
</tr>
<tr>
<td>Single</td>
<td>
<c:choose>
<c:when test="${person.single}">
<input type="checkbox"
      checked onclick=
        "setInputElementValue('single', this.checked)">
<input type="hidden" id="single" value="true"
      <f:fieldName att="person.single"/>>
</c:when>
<c:otherwise>
<input type="checkbox"
      onclick="setInputElementValue('single',
        this.checked)">
<input type="hidden" id="single"
      value="false" <f:fieldName att="person.single"/>>
</c:otherwise>
</c:choose>

```

```

</td>
</tr>
<tr>
<td>Gender</td>
<td>
<f:invoke var="{person}" methodName="genderValidValues"
    retAttName="genders" retAttScope="Page"
/>
<select <f:fieldName att="person.gender"/>>
<c:forEach var="gender" begin="0" items="{genders}"
    varStatus="status">
<c:choose>
<c:when test="{person.gender == gender}">
<option value="<c:out value="{gender}"/>"
    selected="true"><c:out
value="{gender}"/></option>
</c:when>
<c:otherwise>
<option value="<c:out value="{gender}"/>">
    <c:out value="{gender}"/></option>
</c:otherwise>
</c:choose>
</c:forEach>
</select>
</td>
</tr>
<tr>
<td>Occupation</td>
<td>
<f:invoke var="{person}" methodName="occupationValidValues"
    retAttName="occupations" retAttScope="Page" />
<select <f:fieldName att="person.occupation"/>>
<c:forEach var="occupation" begin="0" items="{occupations}"
    varStatus="status">
<c:choose>
<c:when test="{person.occupation == occupation.key}">
<option value="<c:out value="{occupation.key}"/>"
    selected="true"><c:out
value="{occupation.value}"/></option>
</c:when>
<c:otherwise>
<option value="<c:out value="{occupation.key}"/>"><c:out
value="{occupation.value}"/></option>
</c:otherwise>
</c:choose>
</c:forEach>
</select>
</td>
</tr>
<tr>
<td>Telephones</td>
<td>

```



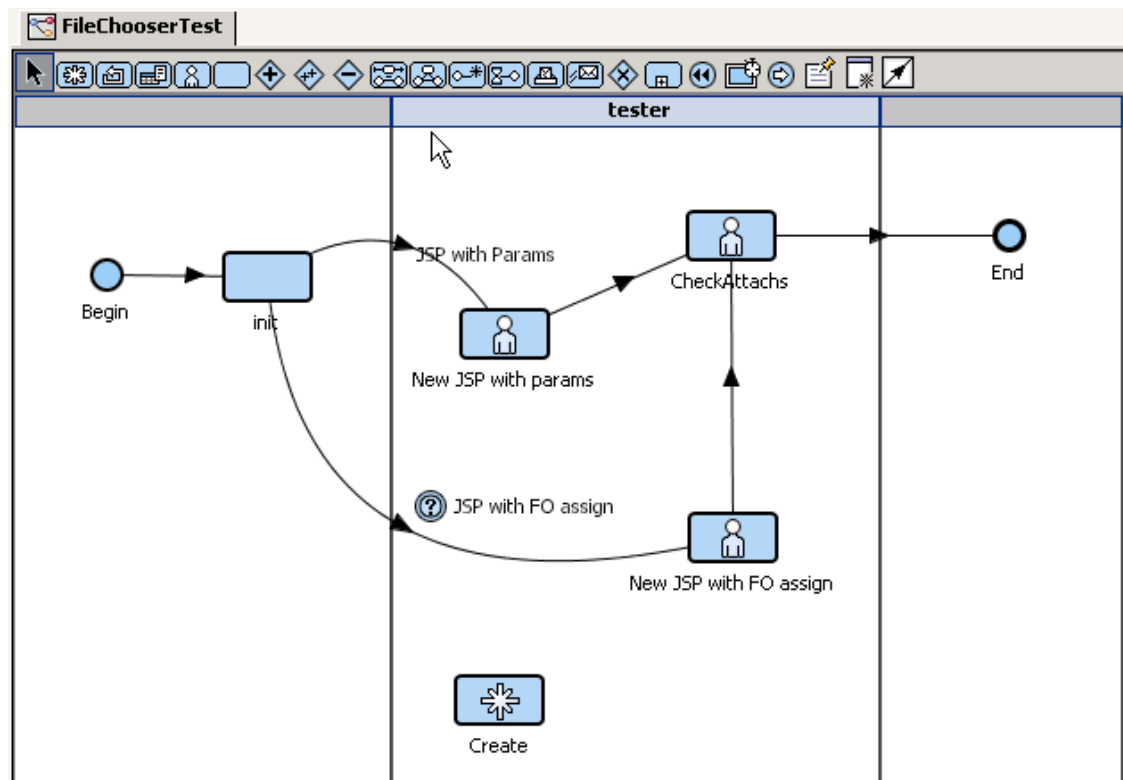
```
<table>
<c:forEach var="telephone" begin="0"
  items="${person.telephones}" varStatus="status">
<tr>
<td><input type="text" <f:field att=
  "person.telephones[${status.index}].type"/>/></td>
<td><input type="text" <f:field att=
  "person.telephones[${status.index}].number"/>/></td>
</tr>
</c:forEach>
</table>
</td>
</tr>
<!-- BUTTONS -->
<tr>
<td>
<input type="hidden" id="selectedButton"
  name="selectedButton" value="OK"/>
<input type="button" id="OK"
  value="Submit" onclick="submitForm(this.id)"/>
<input type="button" id="CANCEL"
  value="Cancel" onclick="submitForm(this.id)"/>
</td>
</tr>
</table>
</form>
</body>
</html>
```

There is a complete example, which shows the use of JSPs combined with Fuego Objects in the *sample* directory of your FuegoBPM Studio's installation. The project name is *FoodDelivery.fpr*.

Examples Using Attachments

The complete example explained in this section can be found in the *sample* directory of your FuegoBPM Studio installation. The project name is *FileChooserTest*.

Main Process - *FileChooserTest*

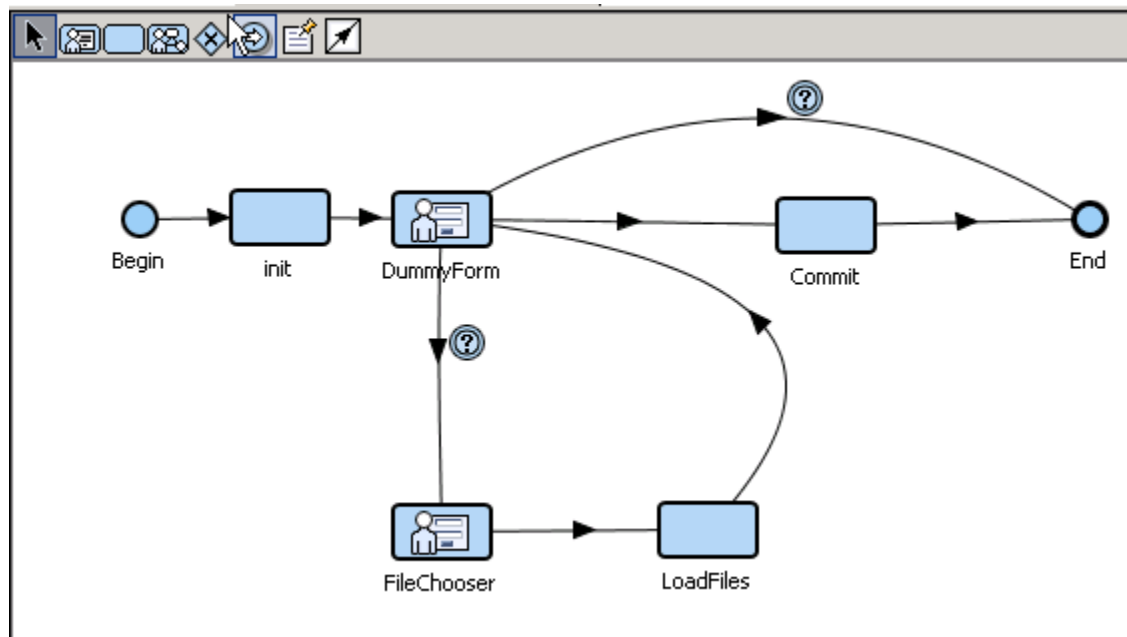


Base on the selection of the type of test prompt in the creation activity, the instance flows to the activity "New JSP with params" or to "JSP with FO assign". Both activities invoke a screenflow, *FileChooserJSPwithParams* and *FileChooserJSPFOAssign* respectively, where the attached files are loaded.

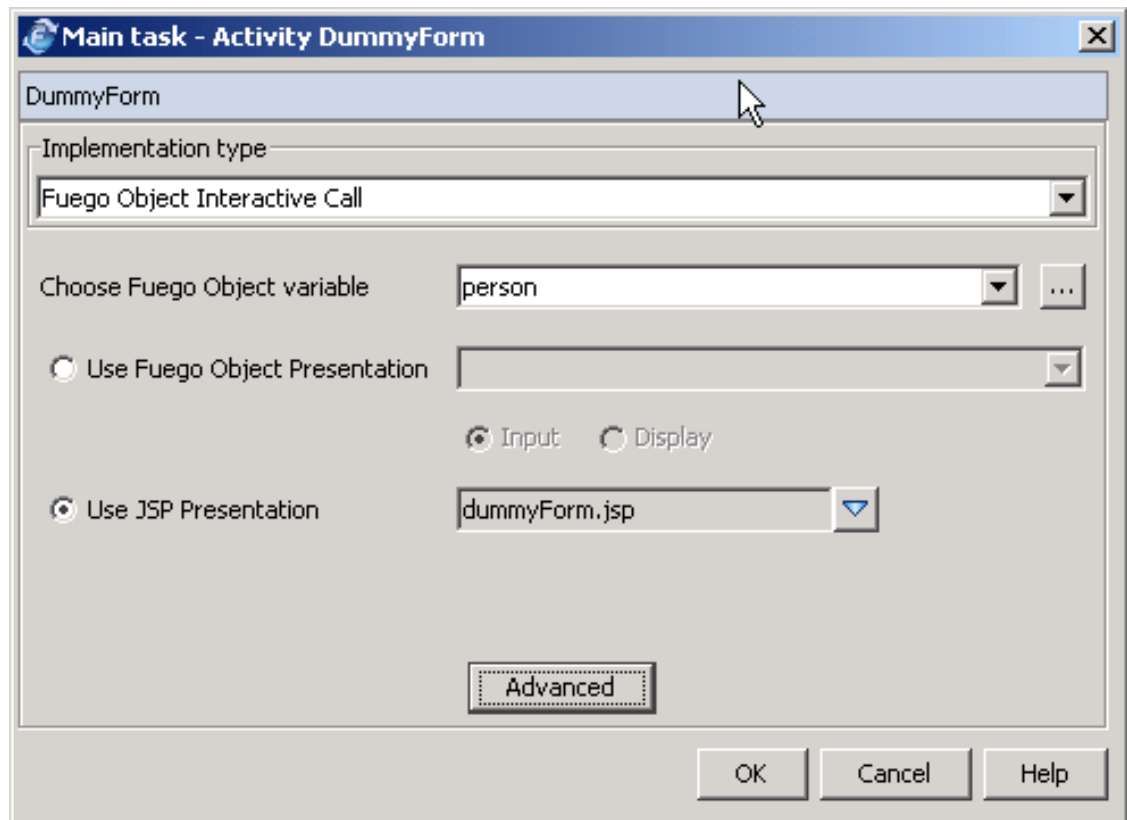
Finally the attachments can be checked in the last activity "CheckAttachs". The following picture shows the execution.

Screenflows

Both screenflows "FileChooserJSPwithParams" and "FileChooserJSPFOAssign" have the same flow.

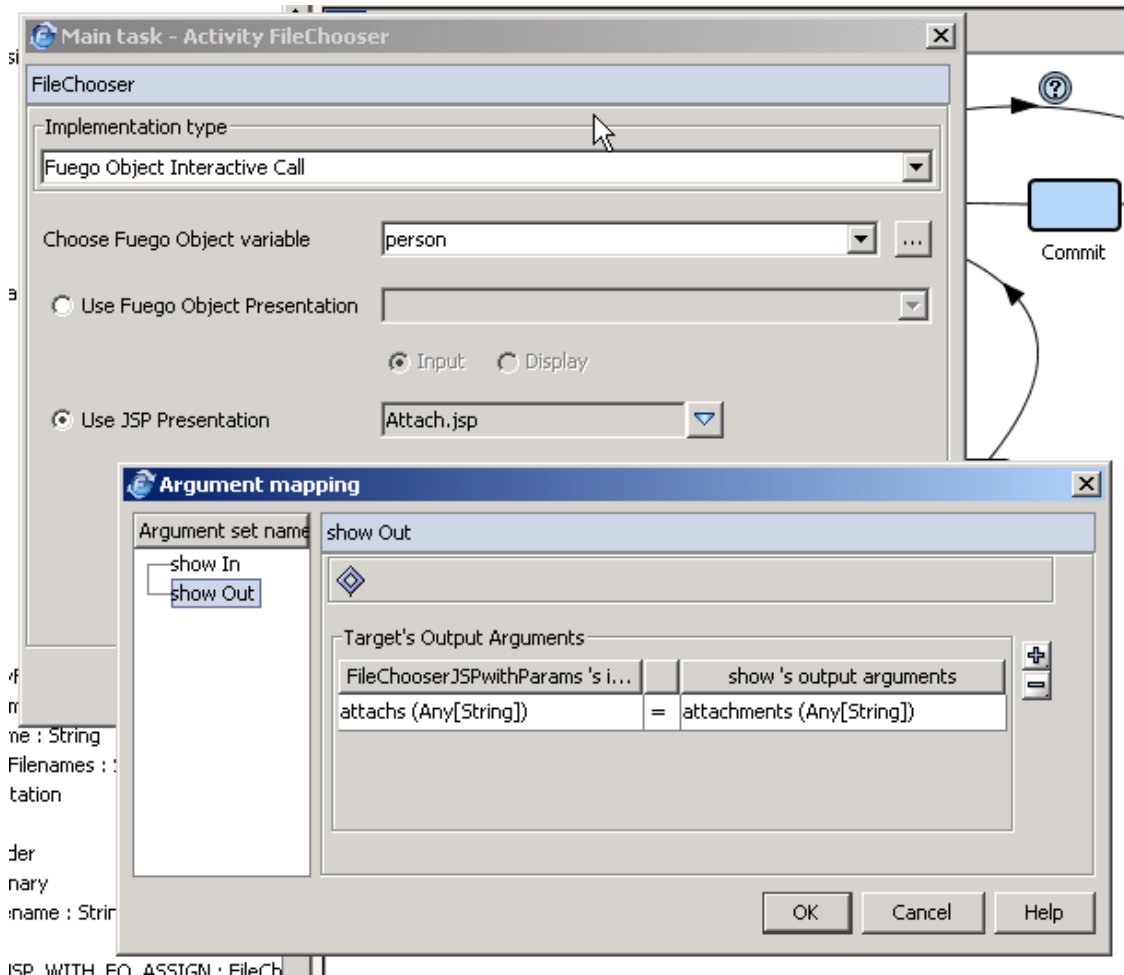


The *DummyForm* activity invokes the jsp *dummyForm* in both screenflows.



FileChooserJSPwithParams

The *FileChooser* activity uses the *person* instance variable of type *DummyForm* Fuego Object. It invokes the *Attach.jsp* and receives the *attachment* argument and stores it in the *attach* instance variable.



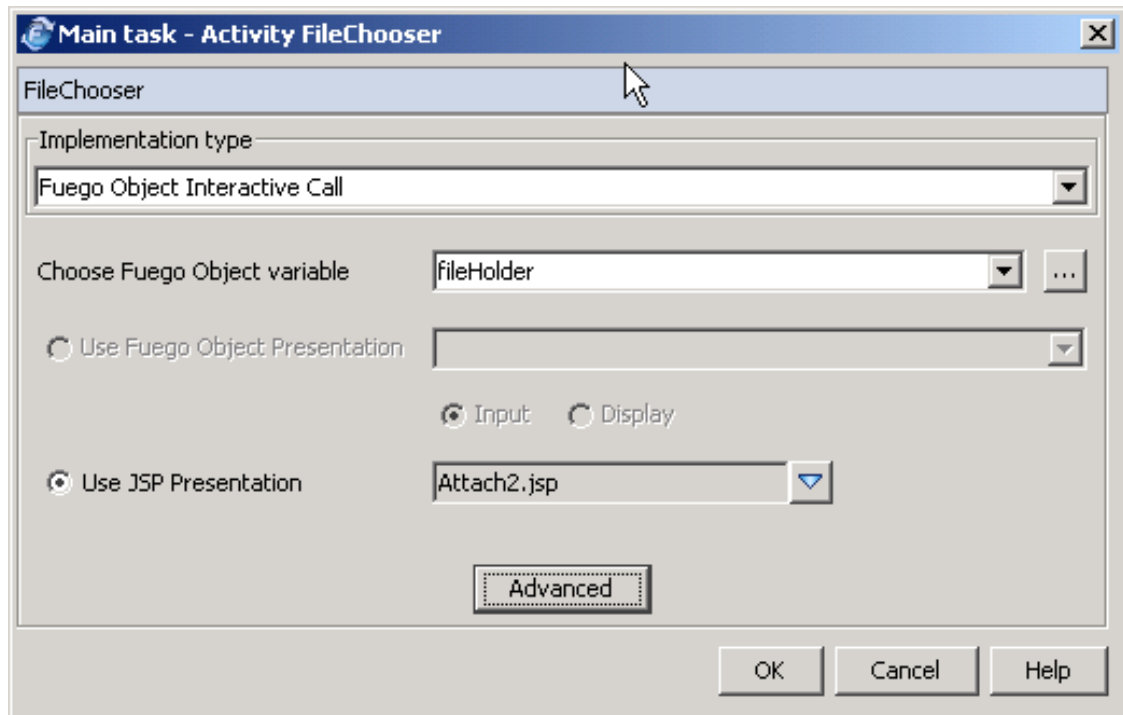
The *loadFile* activity retrieves the content and name of the attached file, by referring it as the name in the jsp given in the tag, "attachment", plus "contents" and "filename" respectively.

```
filename as String = String(attachs["attachment_filename"])
filebytes as Binary = Binary(attachs["attachment_contents"])
```

The *Commit* activity creates an attachment to the instance.

FileChooserJSPFOAssign

The *FileChooser* activity uses the *fileHolder* variable of type "FileHolder" fuego object. It invokes the *Attach2.jsp* which makes reference to the attribute *FileHolder.file* of type Binary, where the attachment will be automatically loaded.



The *loadFile* activity retrieves the content and name of the attached file stored in the *fileHolder* instance variable.

```
person.loadedFileNames[] = fileHolder.file_filename
fileMap[fileHolder.file_filename]=fileHolder.file
```

The *Commit* activity creates an attachment to the instance.

Fuego Objects

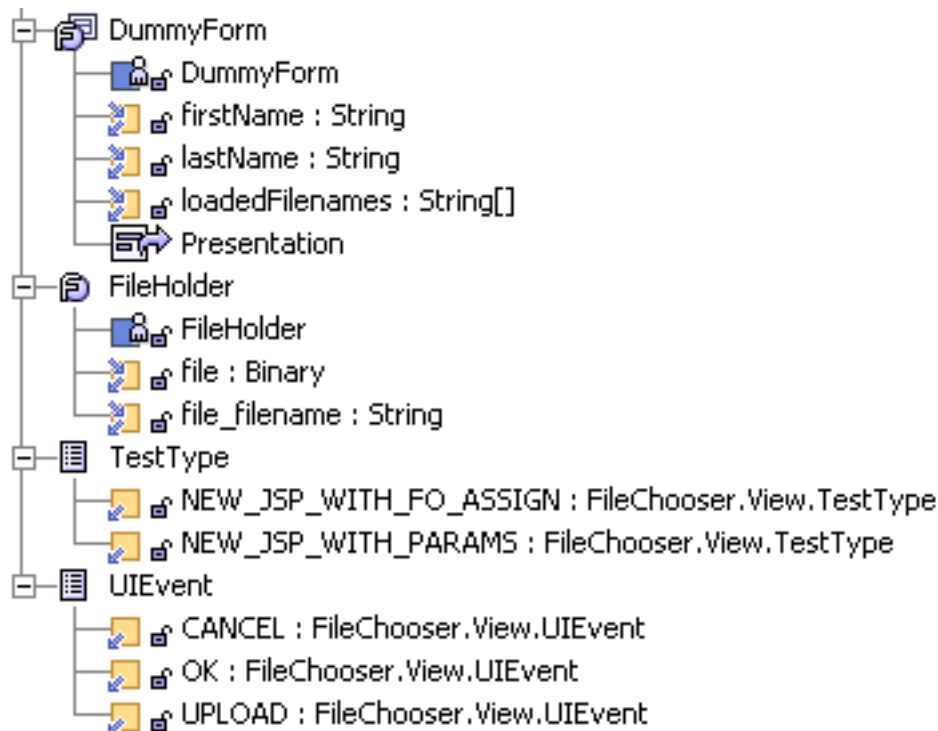
The Fuego Objects used in this examples are **DummyForm**, and the

FileHolder.

The **DummyForm** object is used to load a person data and attached files, through the *dummyForm.jsp* page.

The **FileHolder** object is used in the *FileChooserJSPFOAssign* screenflow, to assign the attached file in the JSP page. It has two attributes, *file* to hold the content of the attachment and *filename* to store the name of the attached file.

Two enumerations are also part of the example catalog. *TestType* to select what example to execute in the *Create* activity of the main process. And *UIEvent* that represents the user selection when the *dummyForm.jsp* page is executed.



Custom Versionables JSP



Customer JSP used in this example are:

- **dummyForm**: Used as presentation to load the *person* data.
- **Attach**: Used in the screenflow *FileChooserJSPwithParams*, in the "FileChooser" activity. Used to load the attachments of the person object. The attachment is received as argument from the jsp page in the *attachment* predefined argument.

```
<input id="attachment"
      type="file" name="attachment" value="">
```

- **Attach2**: Used in the screenflow *FileChooserJSPFOAssign*, in the "FileChooser" activity. Used to load the attachments of the person object. The attachment tag refers to the *FileHolder.file* attribute where the file content is loaded.

```
<input id="attachment" type="file" <f:fieldName
      att="fileHolder.file"/> value="">
```

Transformations

A *Transformation* is a conversion performed on the result of one or more *source expressions*. It generates a new value of a *target* type. This conversion is performed by applying a set of defined rules stored in the *transformation library*.

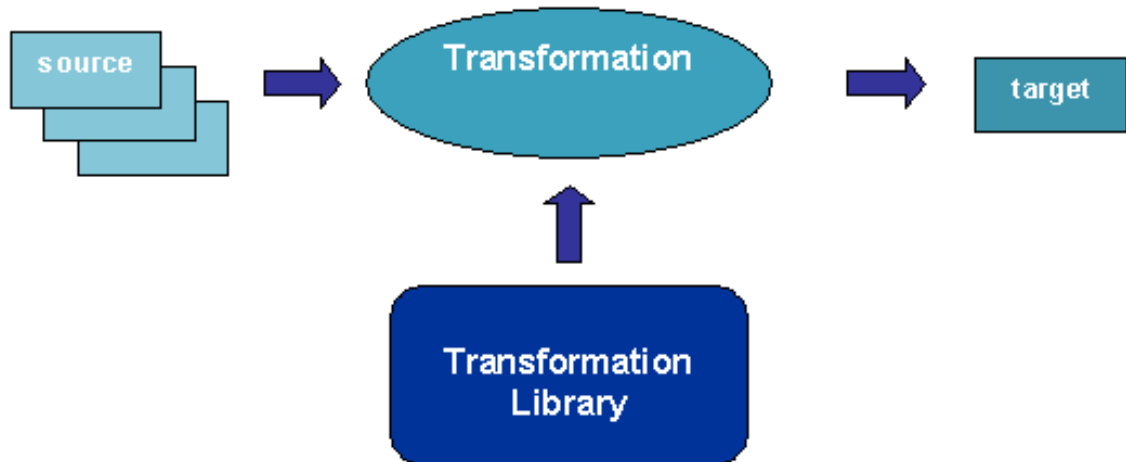
Transformation Elements

The basic elements of a *Transformation* are:

- the *sources*
- the *set of rules* to be applied, which are stored in the *transformation*

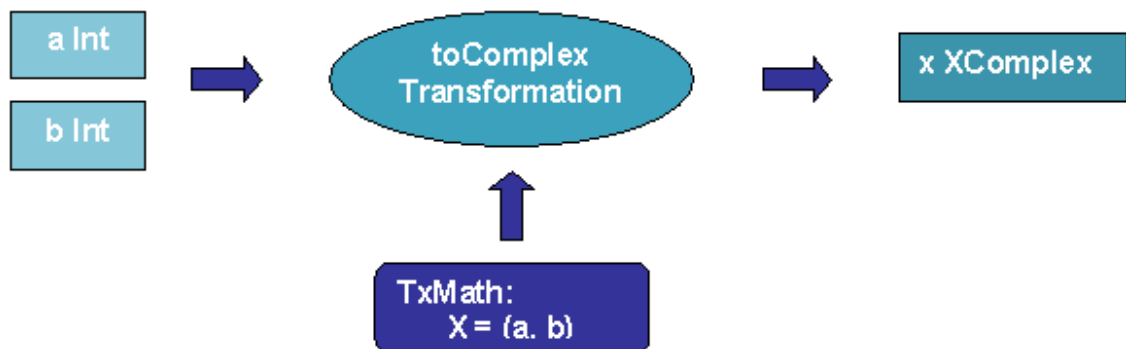
library

- the *target*



As shown in the picture above, the set of rules of the transformation is applied to one or more sources obtaining only one target.

With a simple math example, let's design the *toComplex* transformation where the sources are two *integers* and the target is a complex number.



To complete defining a transformation, it is necessary to specify:

1. A name for the transformation

2. One or more named source types
3. The target type and a name to reference it
4. Local variables to the transformation, if needed
5. Rules to perform the transformation
6. The library where it is stored

Source

The transformation can have one or more sources. They can be of any type: Basic types such as *Int*, *Real*, *Decimal*, even *Binary* and *Any*, or any other component defined in the *Project Catalog*, such as *Java*, *SQL*, *EJB*, *JNDI*, *Web services*, *XML*, *Fuego Objects*, or any other type that FuegoBPM Studio allows you to catalog.

Target

The transformation has only one *target*. It can be, as in the case of the sources, of any of the types allowed.

Target Initialization

The target type will be automatically initialized only if it has a default initialization, that is to say:

- it has a default constructor, or
- it is a predefined type.

If the type does not have a default constructor, it must be explicitly initialized by the user in the *Begin transformation procedure*. See the following section.

Transformation Rules and Library

The *transformation library* is a Fuego Object that contains a set of rules. In our math examples, the transformation library was named *TxMath*. This Fuego Object can contain one or more transformations.

Although a transformation has a name, when it is invoked it is not referenced by name but by the library where it is contained. That is why, so far, transformations with the same source and target definition cannot be contained in the same library.

If more than one transformation contained in a library may be applied, it is the compiler who resolves which one applies according to the more specific types definition. For example, following with the math example, there can be more than a transformation to complex:

Tx Name	Sources types	Target type
<i>intToComplex</i>	(Int, Int)	XComplex
<i>realToComplex</i>	(Real, Real)	XComplex
<i>anyToComplex</i>	(Any, Any)	XComplex

In this case, if you invoked a transformation with sources of type *Int*, any of the above defined could be applied, but as the more specific (Int, Int) is defined, that is the one that the compiler will chose. If only the last two of the list were defined, the more specific one to be applied if your sources type are (Int, Int) *realToComplex* is chosen.

The transformation name is only used for information purposes. The compiler uses it when an error is found. It shows the error indicating the transformation by its name.

These rules are grouped in three sections:

1. Begin Transformation Method
2. Target Component Attributes
3. End Transformation Method


Begin and End Transformation Methods

Pre and post procedures allow the user to execute a custom code to do the transformation. These procedures can contain:

1. References to the source types
2. References to the target type
3. References to local variables (defined as sources)
4. Any valid Method statement

References to source and target types must be made as shown in the following image: **arg**.`[source, target]`

Note

 **arg** qualifier for sources and target is optional if there is not a local variable with the same name.

Target Component Attributes


Field mappings define the rules to associate a value to each field of the target type. These rules can be defined as the direct assignment of a source to a target, an expression (`arg.z.importe * 0.21`) or as a method (`Order.calcOrderPrice`).

The code for these mappings can contain:

1. References to the source types
2. References to the target type
3. References to local variables (defined as sources)
4. Any valid Method statement

References to source and target type must be made as shown in the following image: **arg**.`[source, target]`

Note


 **arg** qualifier for sources and target is optional if there is not a local variable with the same name.

If the target type is not a component type (e.g., Int, Real), some special rules are applied:

1. Only one field mapping is created and it represents the value
2. The name of the field is the name of the target type

Field mappings are not mandatory, as the target can be resolved either in the *Begin Method* or *End Method* custom procedures.

Transformation Definition

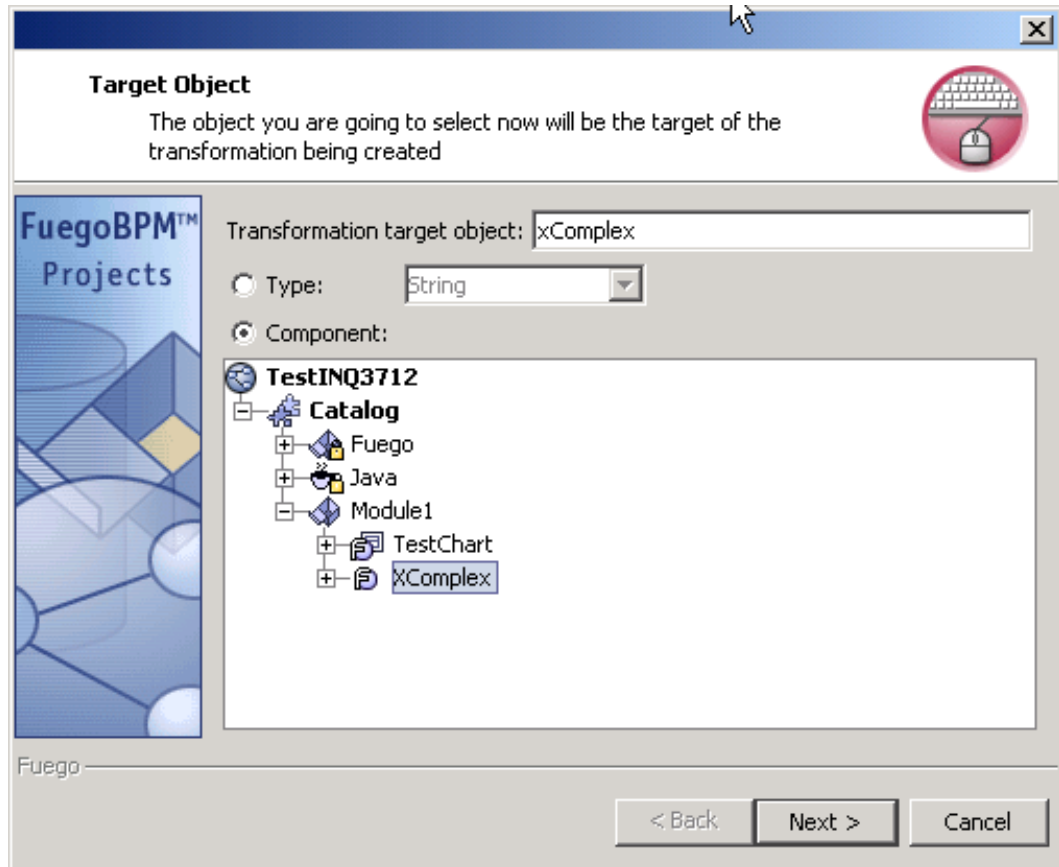
A transformation is identified within a Fuego Object with the following icon: .

Transformation Wizard

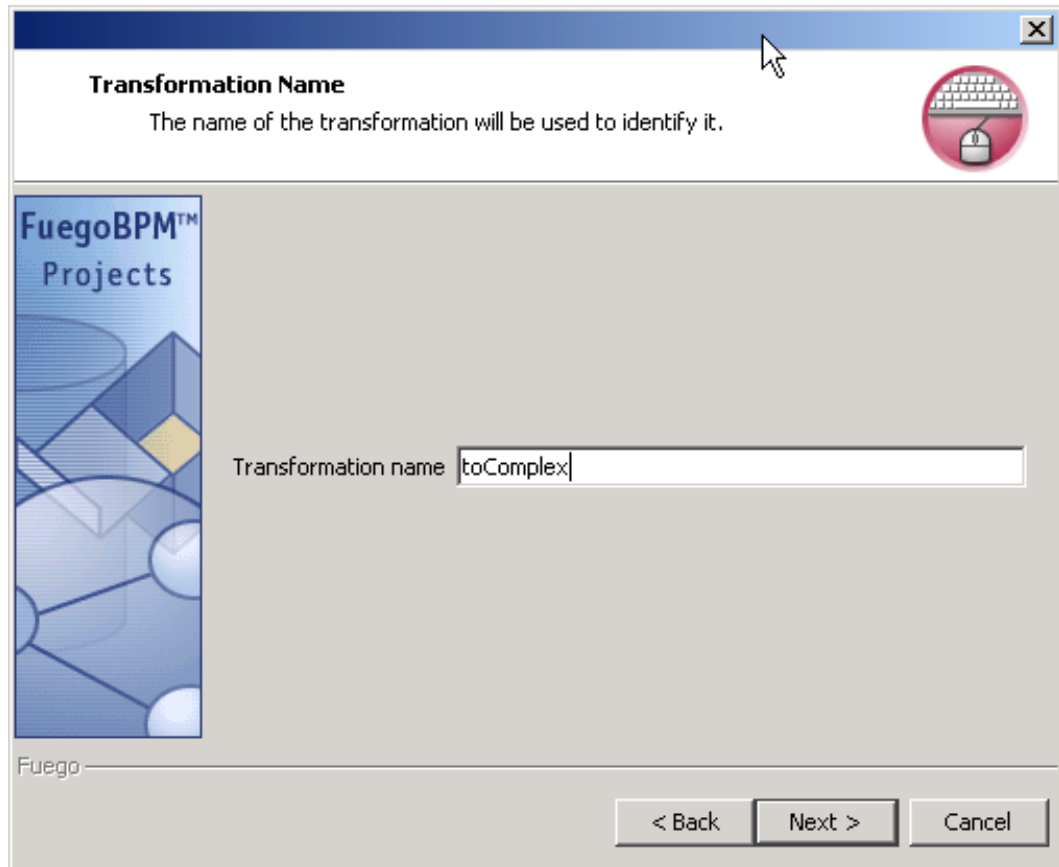
FuegoBPM provides a wizard to help and make the transformation definition easier. Let's follow the steps and define a simple transformation.

Transformations are created in the Fuego Object, which is the transformation library. To create a new transformation:

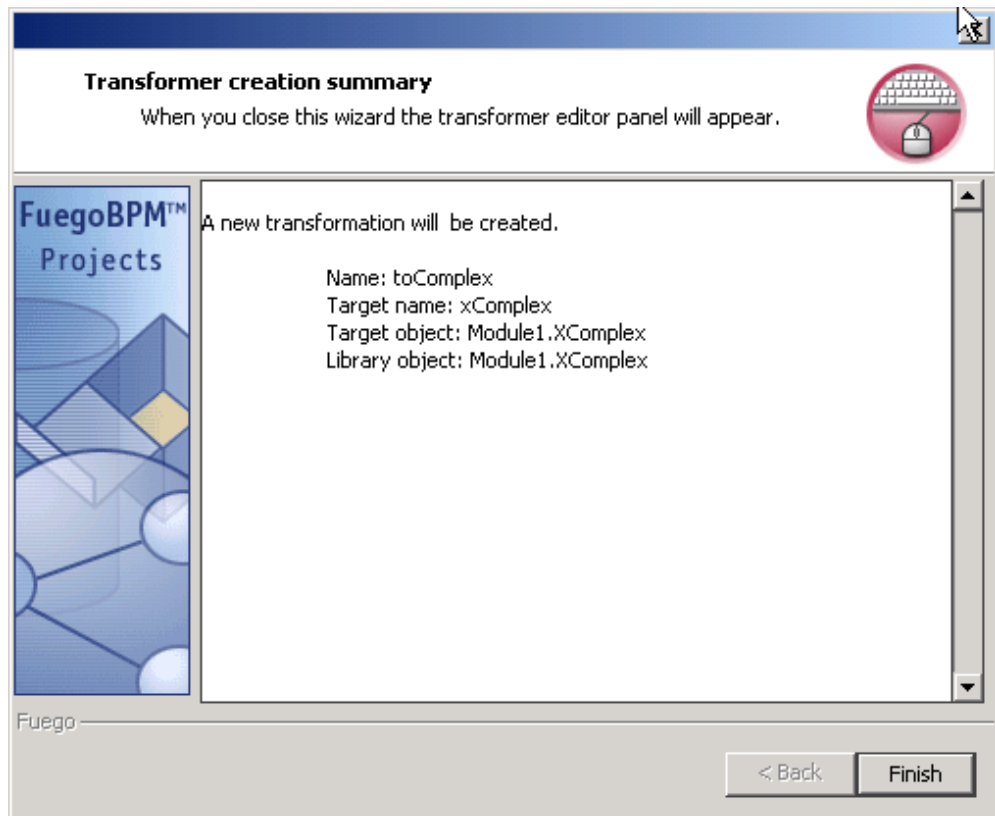
1. Right-click on the Fuego Object you want to add the transformation and select the option **New Transformation** from the menu. The first step of the wizard for transformations definitions is displayed.



2. This step is the *target type* definition. By clicking the *Type* option of the radio button, you can choose any of the basic types or by clicking the *Component* option, select a cataloged component browsing the Project Catalog. Click **Next** to continue.
3. In the next step, type the name you want to give to the transformation. Remember that this name is only informative at compiling time in case an error occurs. It is not used to call the transformation. Click **Next** to continue.



4. The last step shows you the specification of the transformation that will be created. Click **Finish** to end the wizard. It shows:
 - a. Name
 - b. Target name
 - c. Target object
 - d. Library object



5. When the wizard is closed, the Transformation Editor panel is opened to begin building the transformation. See The **Building a Transformation** section for further details on this editor.

Transformations to String

If you are creating a transformation and the target type of the same is String, you can choose the option **New Transformation to String** that appears in the menu by right-clicking on a Fuego Object. By clicking this option, the wizard for transformation creation is not opened but the Transformation Editor Panel is opened instead. This option is available only if a transformation with a String target does not exist within the Fuego Object library.

Handling Transformations

Add a Transformation

To add a transformation, select the option **New Transformation** or **New Transformation to String** from the menu opened when right-clicking on a Fuego Object of your Project Catalog. See section **Transformation Definition**.

Open a Transformation

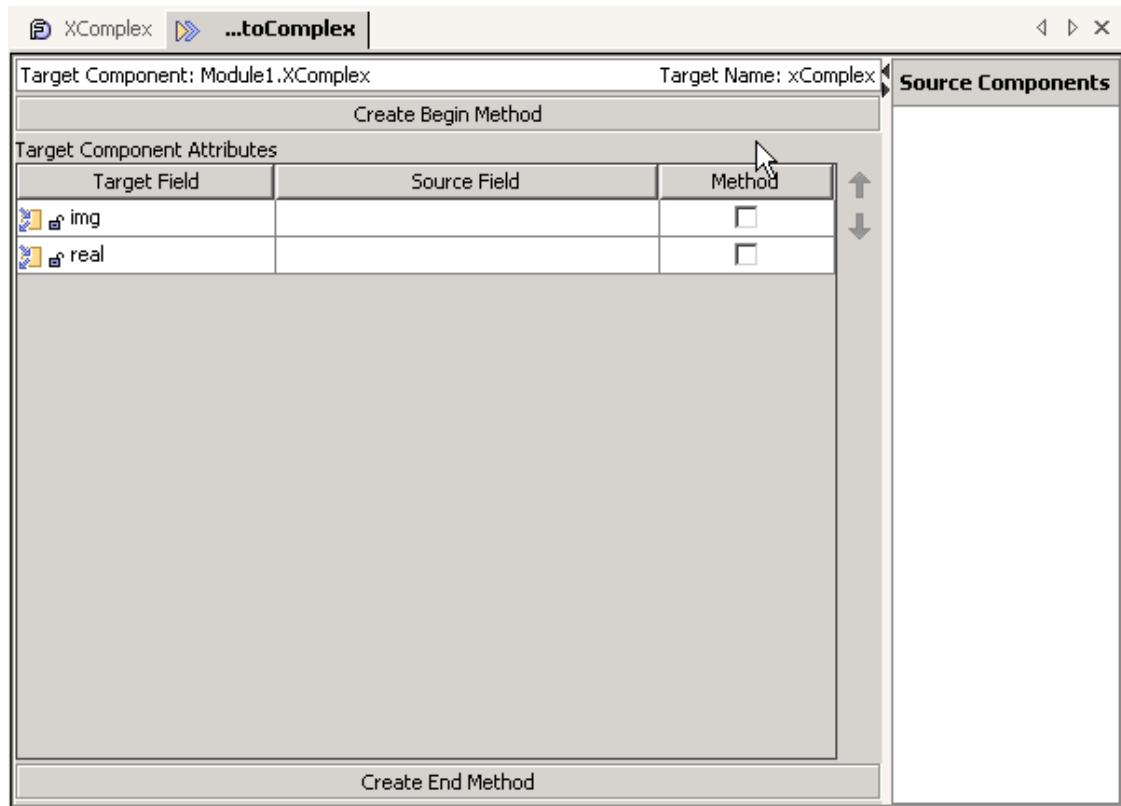
To open a transformation, right-click on the transformation within the Fuego Object Transformation Library and select the **Open** option.

Remove a Transformation

To remove a transformation, right-click on the transformation within the Fuego Object Transformation Library and select the **Remove** option.

Building a Transformation

To complete the transformation construction, you use the *Transformation Editor Panel*. This panel has two working areas, the *target* and the *source* zones.



In the *Source Components* area, you define the different sources you will use to convert into the transformation target and the local variables to the transformation used in the *Begin* and *End* Methods.

To add a new source to the transformation, right-click on the panel source zone and select the option **Add source component**.

Source components can be renamed or removed. To do so, select the source component and right-click. Select the option you want to execute **Remove** or **Rename**.

The *Target* area has the following:

- the *target type* and *name*
- the *Create Begin Method* button
- the *Target Component Attributes* where the target attributes are listed

and where you can define how they are constructed

- the *Create End Method* button

When you select either the button to create the Begin method or the button to create the End method, the Method Editor is opened so that you type its code.

The *Begin Method* can be used to resolve some required data to be set to a target attribute, for example a query to a database to obtain some information based on a source that has to be set to a target attribute; or to combine with a source component, for example the calculation of a discount that has to be added to a price that is a source component.

The *End Method* is used to resolve some required data to set to a target attribute after the mapping has been done, for example the calculation of a tax on the final price target attribute.

Mapping target attributes with source components

For each target attribute listed, three columns are included by default:

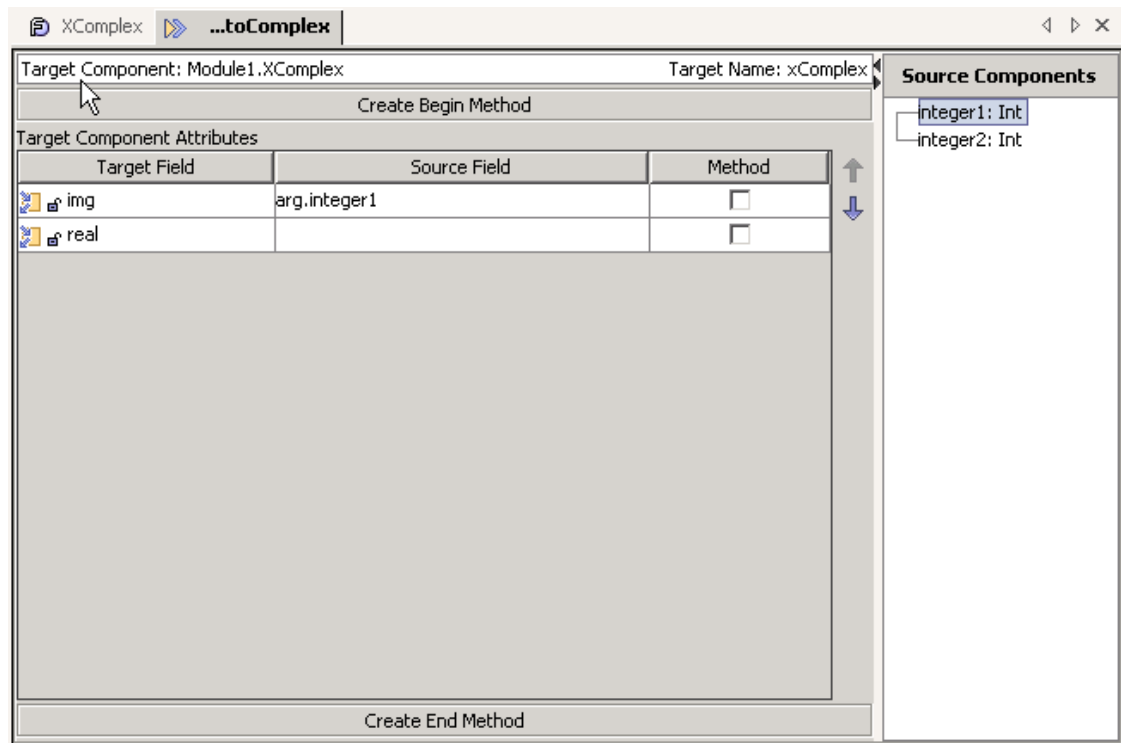
- Target Field, the name of the attribute
- Source Field, the source expression that defines the target, not mandatory
- Method, for the case that the calculation is more complex and that it is not already defined outside the transformation

The *Source Field* can be:

- a source component, *arg.sourceName*

- an expression, *arg.sourceName * 0.10* or *CatalogComponent.method()*
- a method defined within the transformation

By doing drag and drop of source component to a target attribute, you make a direct assignment. Every time a source or target is referenced in an assignment, it must be done using the **arg.** qualifier.

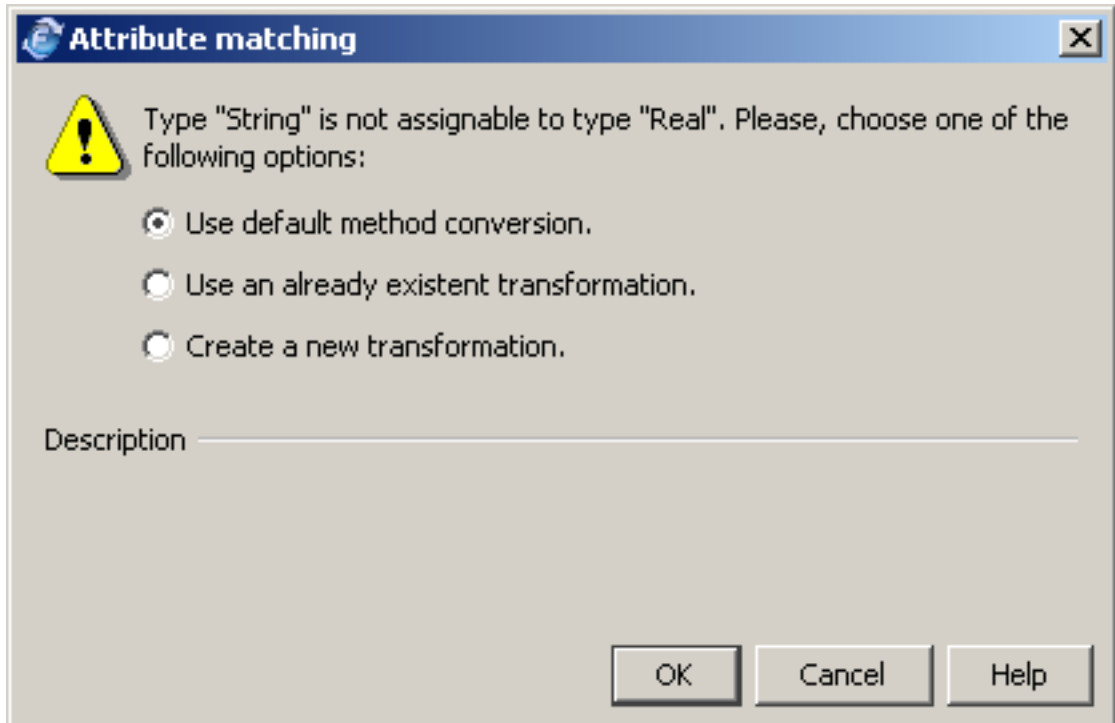


To code an specific method to a target attribute, select the check box in the *Method* column. This enables a button on the *Source Field*. To begin coding, click the button and the Method editor is opened.

Assigning a source component to a target attribute with different types

If you type the *Source Field* and the source component you are referencing were not of the same type as the target field, the checking of the transformation will fail. But, if you do the assignment

by dragging and dropping the source component on the target field, the editor panel will recognize the type differences and will prompt you with a dialog containing three options to correct the differences.



The possible options are:

- ***Use default method conversion*** - if you select this option, the *Source Field* automatically fills with the Method cast sentence that converts the source component into the target attribute type. *arg.sourceName to sourceTYPE : arg.intSTR to REAL* where the source component *intSTR* is *String* and it is being assigned to a Real target attribute.
- ***Use an already existing transformation*** - by choosing this option, a list of possible transformations are displayed to let you choose one of them. Once you do so, the *Source Field* automatically fills with the expression assigning the transformation you have chosen.

- *Create a new transformation* - by choosing this option, a wizard will be opened, with the source and target types preset.

What happens if a transformation field is not set?

In every case (db component heir or not) FuegoBPM sets default value for that field as defined in the attributes definition. If no default was entered then it assumes the following:

- String to "",
- Bool to false,
- Time to 'now',
- Int to 0,
- Decimal to 0,
- Real to 0,
- Fuego Object to null or XObject() depending on the 'not null' flag setting. Note that if the value for the attribute is set or changed in the Fuego Object constructor then it assumes that value.

Syntax for transformation calls

Transformation calls are expressions with the following syntax:

```
<source expression> to <target type>  
    using <transformation library>
```

or

```
<source expression 1>, ..., <source expression N> to  
    <target type> using <transformation library>
```

- *Sources* are expressions which result is transformed to *target type*
- *Target type* is any type defined in the catalog
- *Transformation library* is the component that contains the definition of the transformation

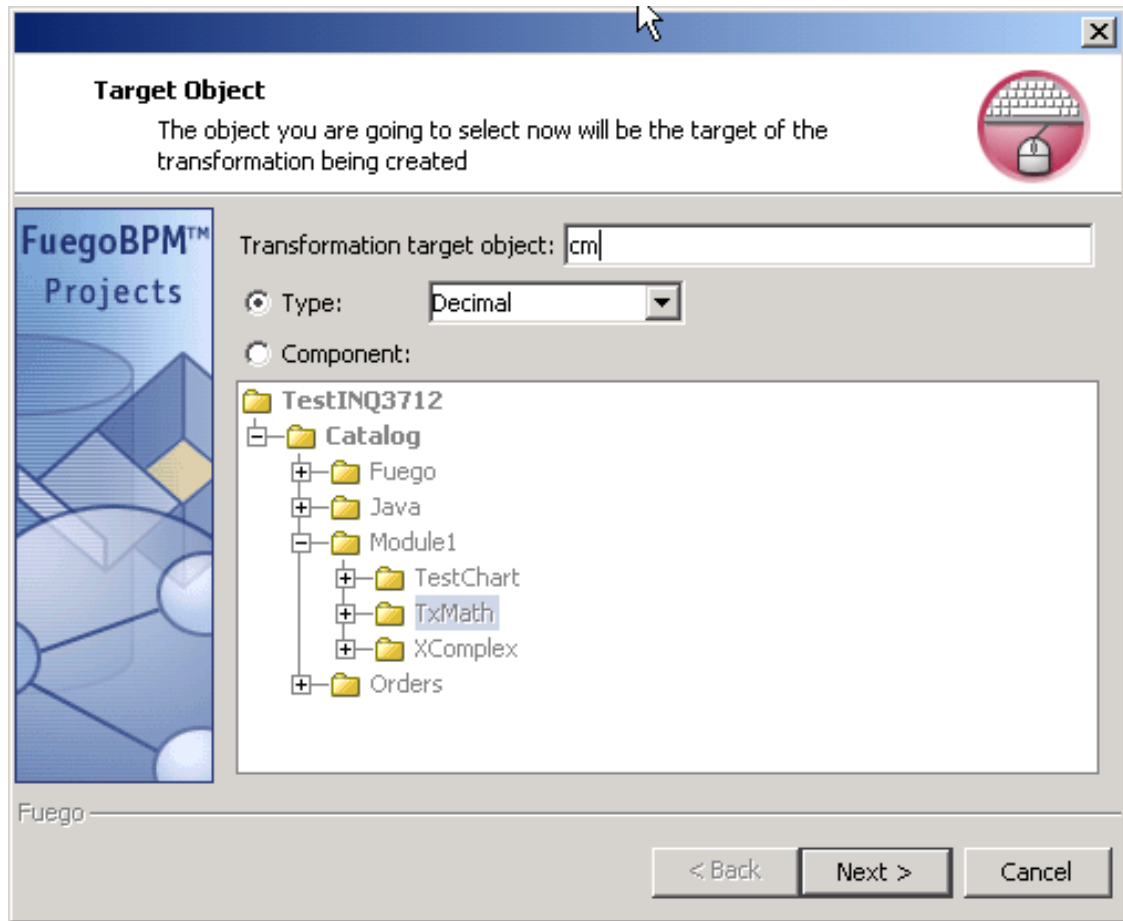
Transformation Examples

Defining the *inchToCentimeters* Transformation

The *inchToCentimeters* transformation converts a value expressed in inches into centimeters.

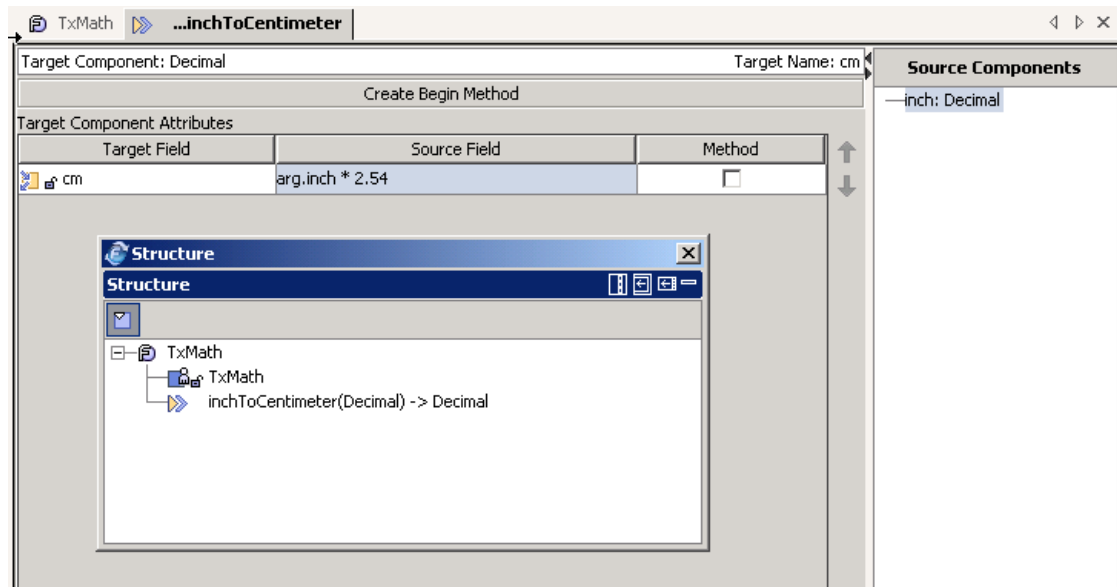
To define it, select the option **New Transformation** from the menu for the Fuego Object transformation library.

Fill the first step of the wizard, with the target name and type, in this case it is a *Decimal* named *cm*.



This transformation has only one source component, the inches to be converted. In the transformation editor, add the source component by right-clicking on the "Source Components" panel and selecting the option "Add source component". Give the name to the source component, in the example "inch" and select the "Decimal" type.

Drag and drop the *inch* source to the source field column of the *cm* target attribute. Edit it and add the expression $* 2.54$.



Calling the *inchToCentimeters* transformation in a Method

```
inch as Int
cm as Int
inch = 1
cm = inch to Int using TxMath
display cm
```

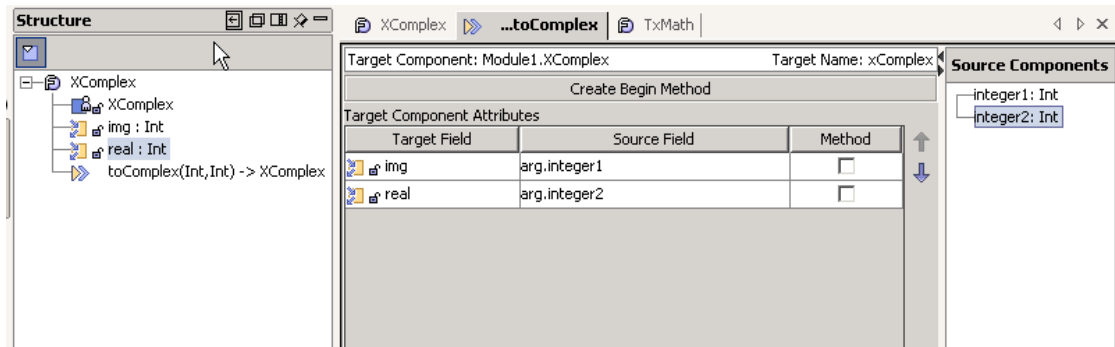
Defining the *toComplex* Transformation

Using the wizard to create the transformation, see the steps described in the section **Transformation Definition** where the steps shown are the ones to define the *toComplex* transformation.

What is the XComplex Fuego Object structure which is the target of this transformation?

It has only two attributes which are the real and imaginative numbers that form the complex number.

toComplex Transformation Editor Panel



Calling the *toComplex* transformation in a Method

```
a as Int
b as Int
x as XComplex

a = 1
b = 2

x = (a, b) to XComplex using TxMath

display (" real: " + x.real + " img: " + x.img )
```

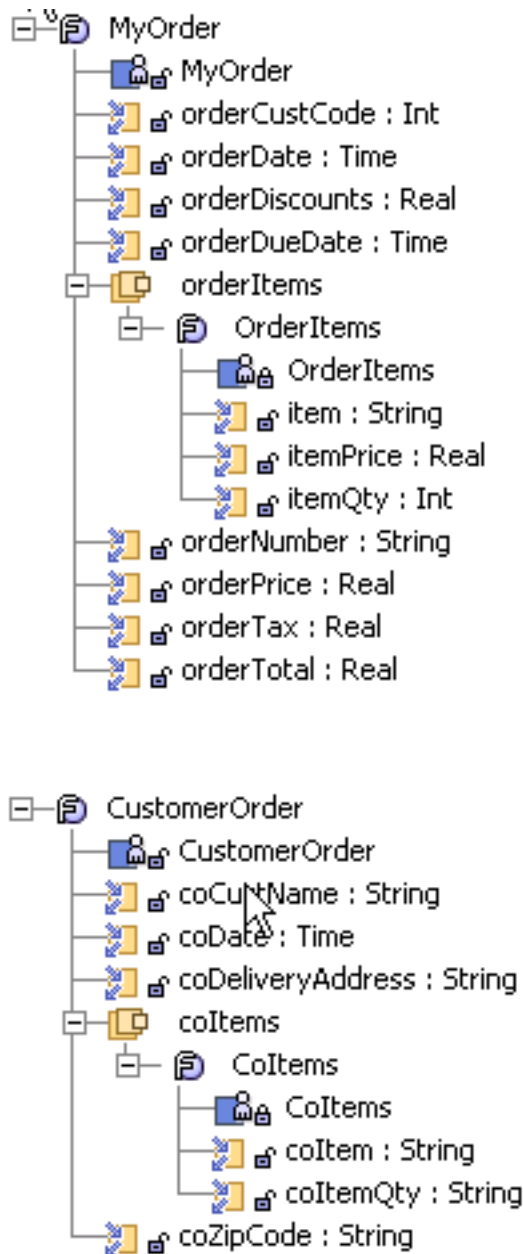
Defining the *toOrder* Transformation

Let's define a transformation which is a little bit more complicated. Suppose we receive a Sales Order from a customer in a different format from the one administrated by our Order Fulfillment system.

Source component and Target attribute

To do so, we define two Fuego Objects, one representing the Order

in your system and the other representing the format recieved from the customers. The *MyOrder* Fuego Object and *CustomerOrder* Fuego Object respectively:



Of course, *CustomerOrder* is the source component and *MyOrder* is the target attribute.

Transformation Rules

The following table shows how *MyOrder* Fuego Object attributes have to be filled in based on the source we have.

MyOrder Attribute	Target type
<i>orderNumber</i>	next available number
<i>orderCustCode</i>	Customer code, based on the customer name
<i>orderDate</i>	Date in the customer order
<i>orderDueDate</i>	Date in the customer order plus 30 days
<i>orderDiscounts</i>	Discount preset to the customer
<i>orderItems.item</i>	Each item received in the customer order
<i>orderItems.itemQty</i>	The quantity per item received in the customer order
<i>orderItems.itemPrice</i>	Based on date and item from our pricing list
<i>orderPrice</i>	Sum of all items price per quantity
<i>orderTax</i>	Based on the zipcode of the delivery address in the customer order
<i>orderTotal</i>	Calculated price minus discount plus tax

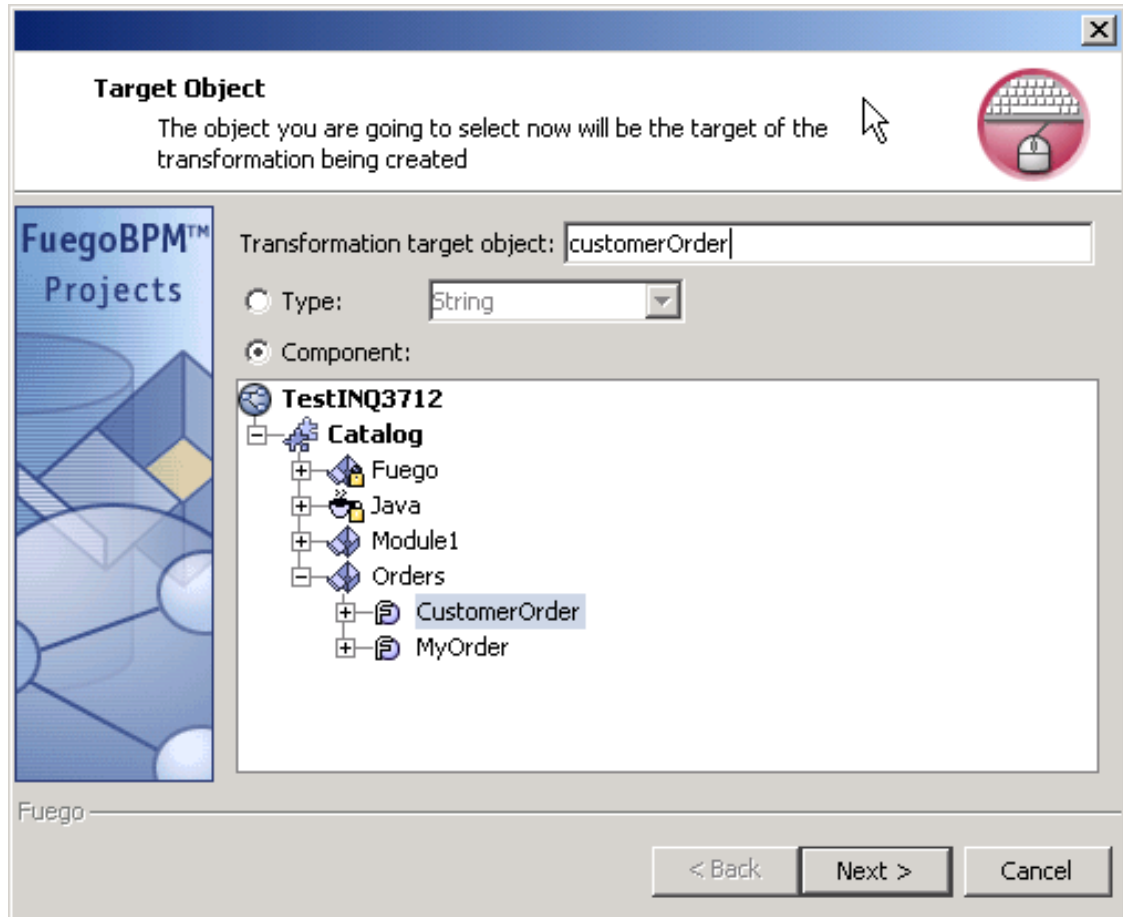
Others:

- The customer delivery address has to be updated in our database.

Transformation Rules Definition

Select the option **New Transformation** from the Fuego Object Transformation Libray shortcut menu. The wizard opens. In the first

step, we have to choose the target type and give it a name. Browsing the catalog *MyOrderFuego* Object is selected and the default suggested name is the one chosen.



Next, in the Transformation Editor Panel, we define the source component and implement the transformation rules. On the *Source component* area, we right-click and browse and select from the project catalog the Fuego Object that represents the customer order *CustomerOrder* FO.

Then, we define the tranformation rules:

- **orderDate**: this rule is implemented by dragging and dropping the *CustomerOrder.coDate* to the target attribute *orderDate*.

- ***orderDueDate***: this rule is implemented by dragging and dropping the *CustomerOrder.coDate* to the target attribute *orderDate*. We edit the expression and add 30 days by typing *+ 30d*.
- ***orderItems*** group: this target attribute is defined using a method. Click the method check box and click the button on the source field column. The Method would be something like the following:

```
i as Int
i = 0

for each item in arg.customerOrder.coItems do

  arg.myOrder.orderItems[i].item = item.coItem
  arg.myOrder.orderItems[i].itemQty = item.coItemQty to Int
  arg.myOrder.orderItems[i].itemPrice =
    findItemPrice(PricingList, item.coItem,
                  arg.customerOrder.coDate)
  i = i + 1
end
```

Later, you will be able to implement it using the *expand* feature.

- ***orderPrice***: this target attribute is defined using a method. Click the method check box and click the button on the source field column. The Method would be something like the following:

```
for each item in arg.myOrder.orderItems do
  arg.myOrder.orderPrice = arg.myOrder.orderPrice
                           + item.itemPrice
end
```

- ***orderTotal***: two implementations are shown to fill this target attribute,
- *source field* with the following expression: *myOrder.orderPrice - myOrder.orderDiscounts + myOrder.orderTax*.
- *End Method*:

```
// order total calculation
arg.myOrder.orderTotal = arg.myOrder.orderPrice -
    arg.myOrder.orderDiscounts +
    arg.myOrder.orderTax
```

Implementing inheritance

Implementing Behavior Inheritance

If a Fuego Object implements *behavior inheritance*, it delegates its implementation to another component.

The component to which a Fuego Object delegates its behavior can be of any of the following types: *SQL*, *Java*, *EJB*, *Fuego Object*, and so on.

A wrapper or Fuego Object that inherits behavior from other components can be built to simplify and speed up the development process. As this wrapper automatically inherits the attributes and methods of the component, it wraps these changes that are automatically available to the wrapper since changes are made to the component.

The only method shown in a Fuego Object is the object's constructor method.

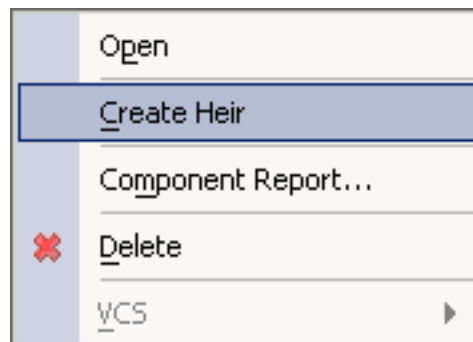
How to create an heir

The heir, or delegate component, can be created in two different ways. It can be created from the component that has a defined behavior or by indicating to the Fuego Object the component to which it delegates the implementation.

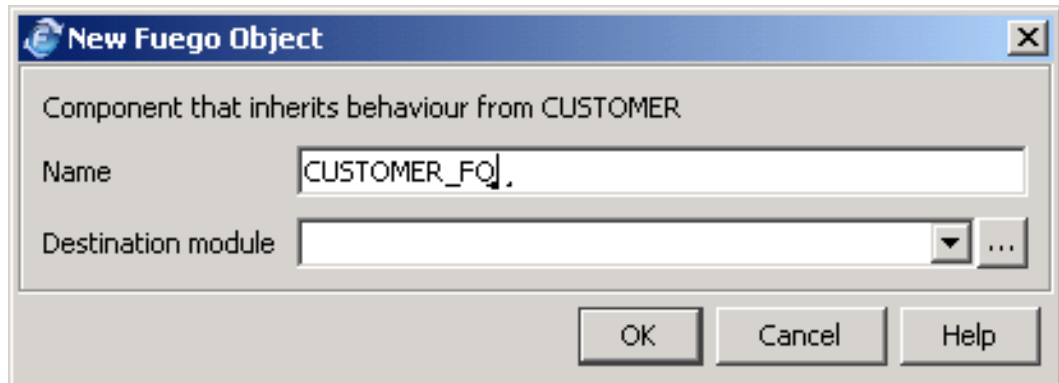
Create an Heir to a Component

To create an *heir* to the component:

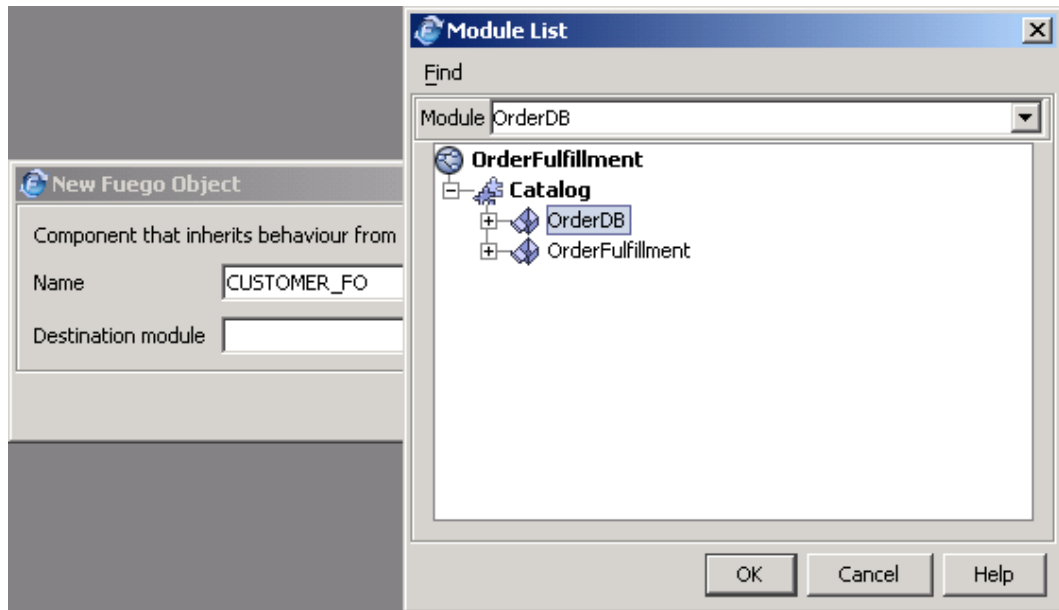
1. Right-click on the component and select the option **Create Heir** from the shortcut menu.



2. Type the name to the new component that inherits the behavior in the dialog that is opened. Browse the project catalog to determine the destination module where it is going to be created by clicking the ... button.



3. Select the module where you will store the new *heir* in the **Module List** dialog and click **OK** to continue.

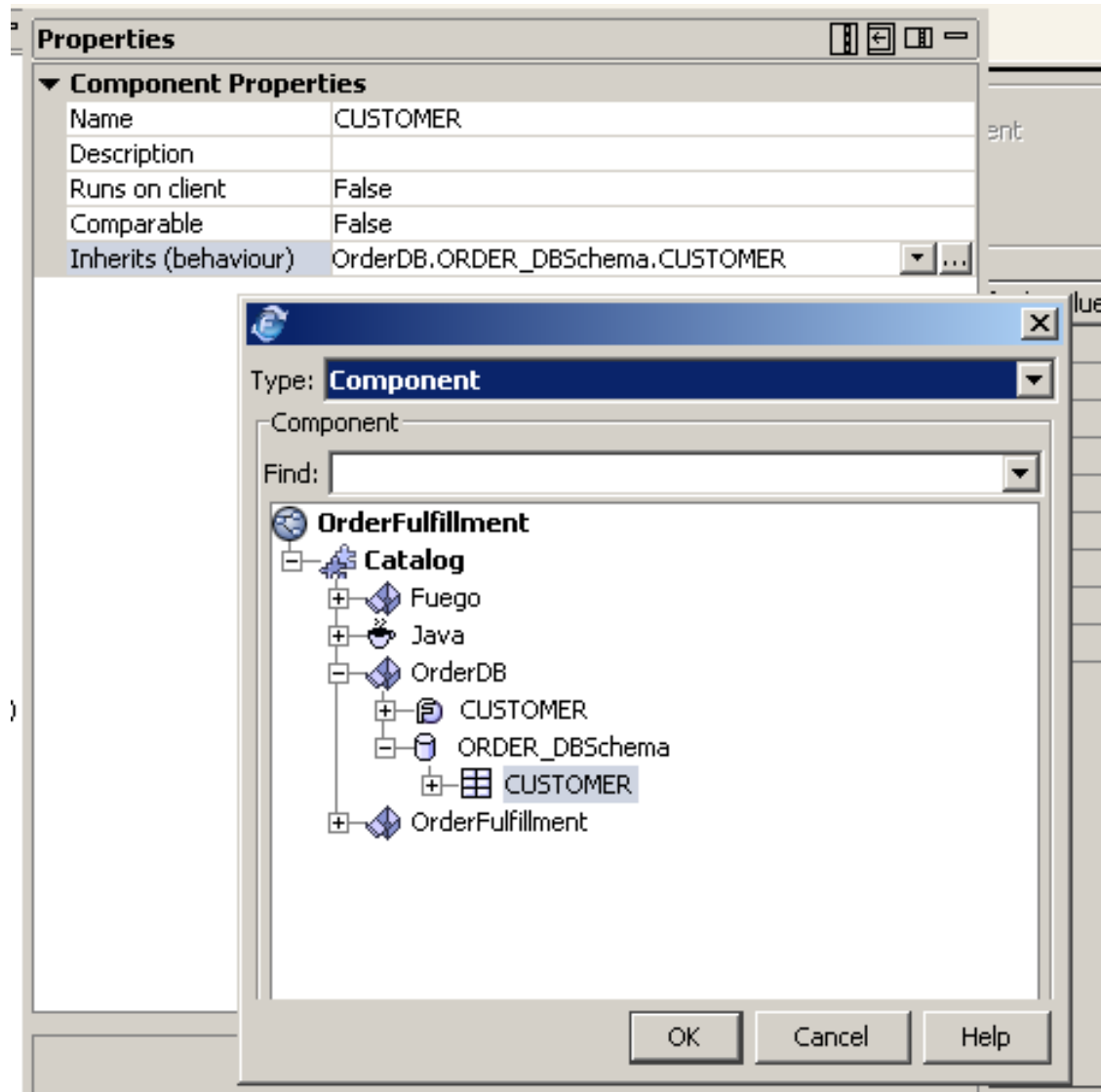


4. The name of the module selected is copied to the first dialog box in the *Destination module* field.
5. The new Fuego Object *heir* component is created.

Define the heir to the Fuego Object

You can define for an already created Fuego Object the component from which it inherits its behavior.

Go to the property flap of the Fuego Object. Then, go to the property *Inherits (behavior)* and browse for the component in the Project Catalog.



What does the Fuego Object heir look like?

The Fuego Object contains the attributes and methods of the component from which it inherits behavior. Open the **Structure** flap or the Fuego Object edition panel to see which attributes and methods have been inherited. You will see them as virtual elements, meaning that you are not allowed to remove or edit them. Anyway, you can add behavior and data to the Fuego Object.

CUSTOMER

Description: ☐ Runs on client

Inherits (behaviour):

Name	Key	Virtual	Type	Not null	Default value
custcode	✓		String(4)	✓	
custname			String(30)	✓	
codpay			Int	✓	
custdisc			Int	✓	
custtype			String(8)	✓	
billadd			String(50)	✓	
shippadd			String(50)	✓	
credlimit			Real	✓	
mail			String(30)	✓	

Inheriting from an SQL introspected component

As any another heir object, it inherits behavior and data structure.

The data structure inherited are the attributed defined in the database table.

An SQL component is a *Fuego.Sql.SqlObject*, from which inherits, the methods:

- load,

- remove, and
- store.

There is also an editable inherited *attribute* **accessDatabase**. The `accessDatabase` default value is set to `true`. If the value is set to `false`, the object is considered as a simple object structure. In this case, any setting value or getting value actions performed on the object do not have effect on the database, but on the object structure.

Please refer to [Configuring SQL Components with no database access](#) for detailed information.

Adding attributes to the Fuego Object heir

You can add attributes to the Fuego Object as you need. Please refer to the [Fuego Object Attributes](#) topic.

Adding and Redefining methods

If you redefine a method, it does not imply that the redefined method of the heir component is invoked, too. You **MUST** explicitly invoke it by implementation.

You can both add and redefine methods to the Fuego Object.

If you are redefining the *load* method inherited from an SQL component, it is not required to redefine it with the same returned type.

Note



In presentation component properties that invoke a Fuego Object method, if some are redefined, only one method is shown and it invokes the closest one to the object in the inheritance chain.

It is very important to bear in mind how a method works when you redefine it. It depends on how you are invoking it and from where. Let's see the different situations, supposing that we have a SQL

component named ORDER and a Fuego Object Order that inherits the behavior of the former:

From a BP Method

A BP-Method code like the following creates an instance of the Fuego Object and executes the *store* method:

```
order as Order()  
order = Order()  
Order.store()
```

If you do,

```
Order.store()
```

it is exactly the same as the first BP-Method code. In this case, it uses the *instance by default* that is automatically generated the first time the Object is invoked. This instance is alive and available in the rest of the code. Next time you invoke a method of the Fuego Object, it will not create it again but instead, it will use the one already created.

From a method of the same component

If you invoke the method *store* within a method of the Fuego Object Order, it being redefined, then:

- Invoking the Fuego Object redefined *store* method:

```
Order.store()
```

In this case, it does not use the instance by default but the *this* instance because it is being invoked within the component. It is the same to do:

```
this.store()
```

- Invoking the SQL component *store* method:

```
ORDER.store()
```

In this case, the store method of the component that contains the behavior is invoked.

Invoking methods when there is more than one level of heir components

When you have two or more levels of behavior inheritance implemented, the component at the second level will not show the implementation of the first component in the chain.

The scenarios explaining different situations are based on three components: a SQL component (SQL), which has a Fuego Object heir (FO2) and a last Fuego Object heir (FO1). The invocation methods are called from a method of the last Fuego Object (FO1).

No redefinition of SQL component method

FO1	FO2	SQL
--	--	<i>store()</i>

Any of the following statements in the *FO1.method()* are exactly the same:

```
FO1.store()
this.store()
SQL.store()
```

Redefinition of SQL component method in the first Fuego Object

FO1	FO2	SQL
--	<i>store()</i>	<i>store()</i>

```
FO1.store()
FO2.store()
this.store()
```

The three sentences invoke the method *store* redefined in the Fuego Object FO2. The *SQL.store()* is invoked only if it has been explicitly included in the redefinition within the FO2.

```
SQL.store()
```

In this case, this sentence does not have the same effect. It will create an instance of the SQL component and execute its *store* method.

Redefinition of SQL component method in both Fuego Objects

FO1	FO2	SQL
<i>store()</i>	<i>store()</i>	<i>store()</i>

```
FO1.store()  
this.store()
```

Both sentences invoke the method *store* redefined in the Fuego Object FO1. The SQL.store() or FO2.store() are invoked only if they have been explicitly included when redefining the method.

```
FO2.store()
```

This sentence invokes the method *store* redefined in the Fuego Object FO2. The SQL.store() is invoked only if it has been explicitly included in the redefinition within the FO2.

```
SQL.store()
```

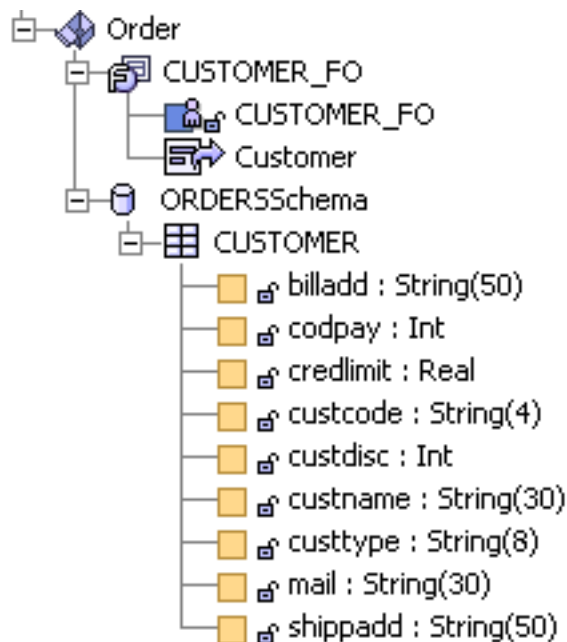
In this case, this sentence does not have the same effect. It will create an instance of the SQL component and execute its *store*

method.

Building a Presentation to the Fuego Object heir

To build a presentation to a Fuego Object heir, follow the same steps indicated in the Fuego Object Presentation topic.

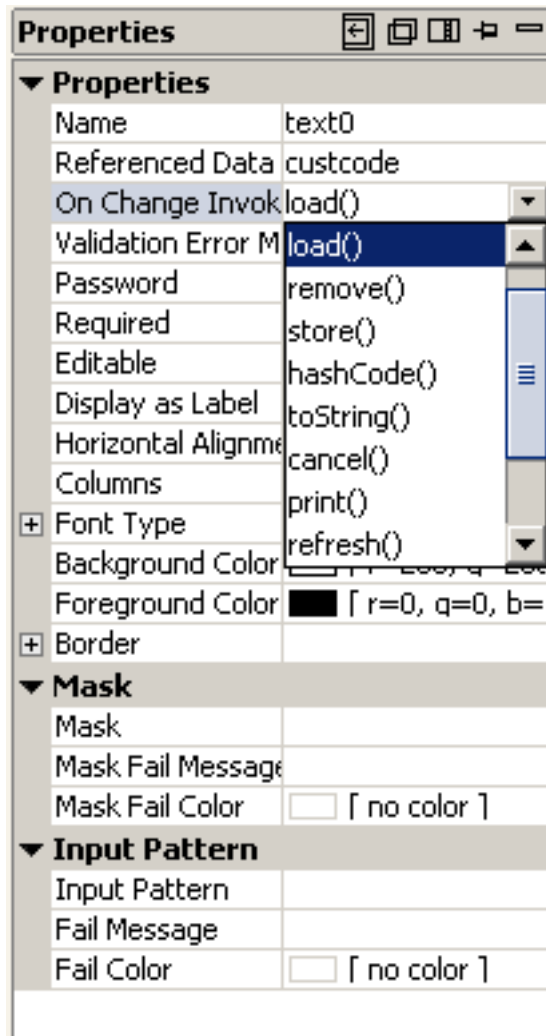
For example, if you are creating a presentation to a Fuego Object heir of a CUSTOMER table named CUSTOMER_FO, you will want to automatically recall all the details for a customer with that number whenever the customer number field is changed.



To do this,

1. Simply select the presentation component that corresponds to the customer number attribute.
2. Click {None} on the field next to the *On Change Invoke* prompt.
3. Select *load()* from the dropdown list. This method was automatically built. It loads/returns a single row of data based on the id passed to it.

4. Now, whenever the order number id field is changed, the *load()* method is run.



The BP-Method to invoke this presentation must contain the *input* statement:

```
customer = CUSTOMER_FO()
input customer using selectedPresentation = "Customer"
```

Insert a new item into the database from the process

In the section above, you have seen how to retrieve information from the database. Let's see how to insert a new customer into the CUSTOMER table using the CUSTOMER_FO Fuego Object heir.

Create the activity in the process, add for the customer the sentences to assign values to its method or invoke the presentation. And, if the user has selected *submit*, call the *store* method.

- Without presentation:

```
customer.custCode = CustomerLib.nextID()  
customer.custCredlimit = 10000  
...  
// fill the rest of the customer table fields  
// and call the store method  
  
customer.store()
```

- With a presentation:

```
customer = CUSTOMER_FO()  
  
input customer using selectedPresentation = "Customer"  
returning userSelected  
  
if userSelected == "submit" then  
    customer.store()  
end
```

Implementing Type Inheritance

How to create an object with type inheritance

To create a Fuego Object with type inheritance,

1. Right-click on the component and select the option **Implement Type** from the shortcut menu.
2. Type the name to the new component that implements type in the dialog that is opened. Browse the project catalog to determine the destination module where it is going to be created by clicking the ... button.
3. Select the module where you will store the new Fuego Object in the Module List dialog and click **OK** to continue.
4. The name of the module selected is copied to the first dialog box in the *Destination module* field.
5. The new Fuego Object implementing type component is created.

A Fuego Object implementing Type Inheritance can only be created based on another Fuego Object, java interfaces and Web Services.

How to define an abstract class

Base abstract classes are those that generally are used like base classes in the inheritance hierarchy. These classes cannot be used to instantiate objects. *Abstract class* 's purpose is to create an appropriate base class, that can be used to inherit other classes. The later classes are the ones that can be instantiate. An inheritance hierarchy does not need to contain an abstract class, however, they are sometimes the first superior level of the hierarchy. For example, in the definition of the *geometrical figure hierarchy*, the *Figure* abstract class would be the first level of the hierarchy. The next level could be *BidimensionaFigure* and *TridimensionalFigure*. And the last level would be the concreat classes for the bidimensional (*Circle*, *Square* and *Rectangle*) and tridimensional figures (*Sphere*, *Cube* and *Tetrahedron*).

A Fuego Object created by implementing type can be defined as an

abstract class by defining at least one of its methods as abstract.

To create an abstract method, go to the Fuego Object and select **New Method** from the right click pop up menu. Once it has been created, open the **Properties** flap for the method and set the *Abstract* property as *true*.

Note



Abstract methods have to be implemented in the classes that inherit from the *abstract* class. If they are not implemented, the Fuego Object does not compile.

Abstract classes cannot be instantiated, it can only be done by instantiating an object of the inherited class.

This *abstract* definition allows you to implement in a Fuego Object both *Implementation inheritance* and *Interface inheritance*. On the one hand, the *implementation inheritance* has the functionality defined in the low levels of the hierarchy, and on the other hand, the *implementation inheritance* has the functionality defined in the high levels of the hierarchy.

Other considerations about Type Inheritance

Every other consideration explained in the section Implementing Behavior Inheritance applies also for those Fuego Object created by *Implementing Type*.

Fuego Objects Example

Example Presentation

An *OrderFulfillment* project is going to be used as example to show Fuego Object's common usages, best practices and recommendations. This project, models, based on a set of processes and components, the business service managing the behavior of people, systems and organizations to orchestrate everything involved from the Customers Administration, the Order Processing, and Stock Administration. Keeping the example simple, you will learn

how to apply Fuego Objects concepts in different scenarios.

This project example can be found in the sample directory of your FuegoBPM Studio 5.X installation with the name *OrderFulfillment*.

Please, refer to Project Processes for further details about the implementation.

Project *DashboardExample* also contains the *Orderfulfillment* project. Definitions are valid also for this project that is used in the *Dashboard* chapter for the examples.

Project Processes

This project has defined the following processes for our example purpose:

- Commercial Information
- Customer Management
- Order Process
- Stock Administration

Project Catalog

The project catalog contains the definitions of some Fuego Objects built to show you how to implement different options and behaviors.

Non-Presentable Fuego Objects

- Utilities
- CustomerFO

Heir Fuego Objects

Fuego Objects that implement behavior inheritance.

- PListH,
- PType
- PayType
- Customer
- Item
- CustomerFO

Presentable Fuego Objects

Presentable Fuego Objects, are those with at least a presentation defined. In our example, the Presentable objects are:

- PayType
- Items
- PTypes
- Customer
- OrderComp

Please, refer to Project Catalog for further details about the implementation.

Example Database structure

Database schema definition language are provided for MS SQL,

CLOUDSCAPE and MySQL. You can find these DLL to create the database in the FuegoBPM Studio *sample* directory of your installation.

```
CREATE TABLE items (itemcode VARCHAR(6)
                     NOT NULL PRIMARY KEY,
                     itemdescr VARCHAR(40));

CREATE TABLE paytype (codpay INT
                       NOT NULL PRIMARY KEY,
                       paydescr VARCHAR(30) NOT NULL,
                       paydisc INT);

CREATE TABLE plisth (validfrom DATE
                     NOT NULL PRIMARY KEY,
                     descrip VARCHAR(30) NOT NULL));

CREATE TABLE pricelist (itemcode VARCHAR(6)
                        NOT NULL REFERENCES items(itemcode),
                        itemprice REAL(6,2),
                        validfrom DATE NOT NULL,
PRIMARY KEY (validfrom, itemcode));

CREATE TABLE customer (custcode INT
                       NOT NULL PRIMARY KEY,
                       custname VARCHAR(30) NOT NULL,
                       codpay INT REFERENCES payType(codpay),
                       custdisc INT DEFAULT 0,
                       custtype ENUM("GLOBAL", "NATIONAL",
                                     "INTERNAT"),
                       billadd VARCHAR(50) NOT NULL,
                       shippadd VARCHAR(50) NOT NULL,
                       credlimit REAL(8,2) DEFAULT 0,
                       mail VARCHAR(30) NOT NULL);

CREATE TABLE stock (itemcode VARCHAR(6)
                    NOT NULL PRIMARY KEY
                    REFERENCES items(itemcode),
                    stock INT NOT NULL DEFAULT 0);

CREATE TABLE stkorder (stktype ENUM("IN","OUT") NOT NULL,
                       stkcode INT NOT NULL,
                       stkdate DATE NOT NULL,
PRIMARY KEY (stktype, stkcode));

CREATE TABLE stkordit (stktype ENUM("IN","OUT") NOT NULL,
                       stkcode INT NOT NULL
                       REFERENCES stkorder (stktype, stkcode),
                       itemcode VARCHAR(6) NOT NULL
                       REFERENCES items(itemcode),
```

```

        qty INT NOT NULL,
PRIMARY KEY (stktype, stkcode, itemcode));

CREATE TABLE orders (ordnum INT
        NOT NULL PRIMARY KEY,
        custcode VARCHAR(4) NOT NULL
        REFERENCES customer (custcode),
        orddate DATE NOT NULL,
        ordduedate DATE NOT NULL,
        orddescr VARCHAR(40) NOT NULL,
        ordprice REAL(8,2) NOT NULL DEFAULT 0,
        ordextchg REAL(8,2) NOT NULL DEFAULT 0,
        ocustdisc REAL(8,2) NOT NULL DEFAULT 0,
        opaydisc REAL(8,2) NOT NULL DEFAULT 0,
        ospecdisc REAL(8,2) NOT NULL DEFAULT 0,
        ordertotal REAL(8,2) NOT NULL DEFAULT 0);

CREATE TABLE orderit (ordnum INT
        NOT NULL REFERENCES orders (ordnumb),
        itemcode VARCHAR(6)
        NOT NULL REFERENCES items(itemcode),
        itqty INT NOT NULL,
        itprice REAL(6,2) NOT NULL,
PRIMARY KEY (ordnum, itemcode));

CREATE TABLE ordlogapps (ordnum INT NOT NULL,
        logtype ENUM("COM","CRD","STK") NOT NULL,
        approved INT NOT NULL DEFAULT 0,
        appdate DATE NOT NULL,
        appby INT NOT NULL,
PRIMARY KEY (ordnum, logtype));

CREATE TABLE shiporder (shipord INT NULL PRIMARY KEY,
        ordnumb INT NOT NULL
        REFERENCES orders (ordnumb),
        shipadd VARCHAR(50) NOT NULL,
        shipdate DATE NOT NULL,
        trancpy VARCHAR(20) NOT NULL,
        tranmail VARCHAR(30) NOT NULL,
        stktype ENUM("IN","OUT")
        NOT NULL,
        stkcode INT NOT NULL
        REFERENCES stkorder (stktype, stkcode));

```

Customer XML schema


```
<?xml version="1.0" encoding="UTF-8" ?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:simpleType name="Address" type="xsd:string"/>
  <xsd:simpleType name="CustomerCode" type="xsd:string"/>
  <xsd:simpleType name="CustomerName" type="xsd:string"/>
  <xsd:simpleType name="CodPay" type="xsd:string"/>
  <xsd:simpleType name="CustDisc" type="xsd:string"/>
  <xsd:simpleType name="CustType" type="xsd:string"/>
  <xsd:simpleType name="Mail" type="xsd:string"/>

  <xsd:group name="shipAndBill">
    <xsd:sequence>
      <xsd:element name="ShipAddress" type="Address"/>
      <xsd:element name="BillAddress" type="Address"/>
    </xsd:sequence>
  </xsd:group>

  <xsd:complexType name="Customer">
    <xsd:sequence>
      <xsd:choice>
        <xsd:element name="CustomerCode"
          type="CustomerCode"/>
        <xsd:element name="CustomerName"
          type="CustomerName"/>
        <xsd:element name="CodPay" type="CodPay"/>
        <xsd:element name="CustDisc" type="CustDisc"/>
        <xsd:element name="CustType" type="CustType"/>
        <xsd:group ref="shipAndBill"/>
        <xsd:element name="Mail" type="Mail"/ >
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:element name="customer" type="Customer"/>

</xsd:schema>
```

Project Processes

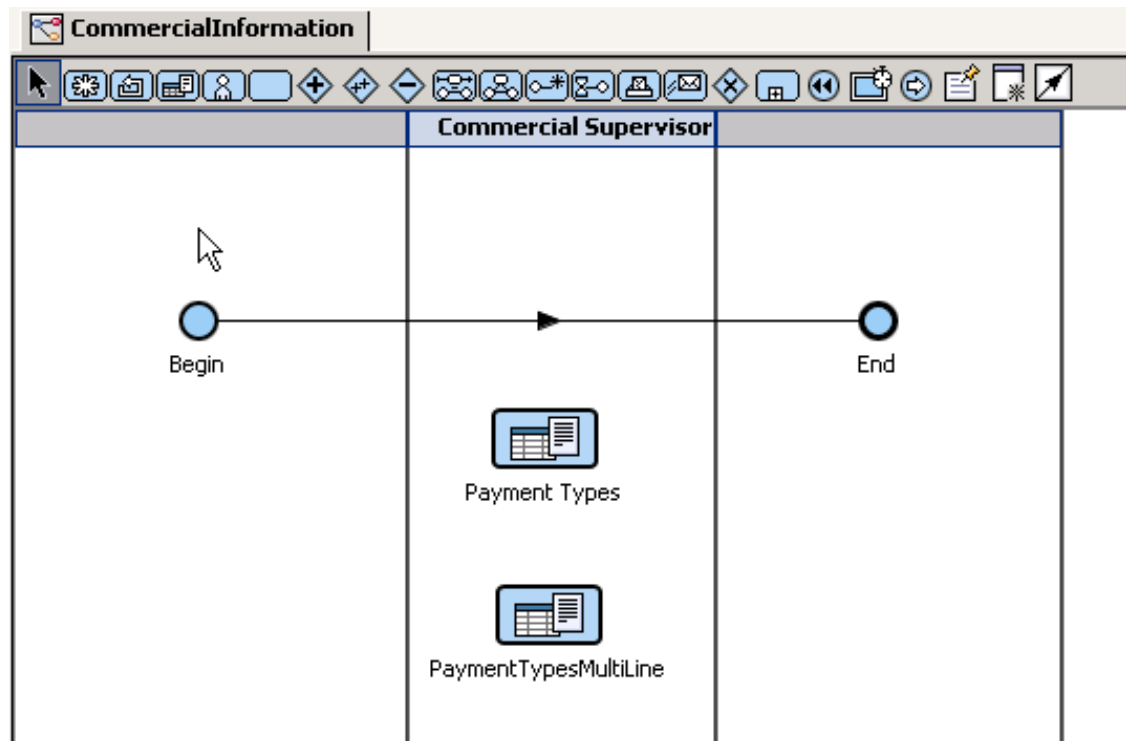
Commercial Information

Process

Using this process, sales representatives or their supervisors update

basic information such as the accepted Payment types by the company.

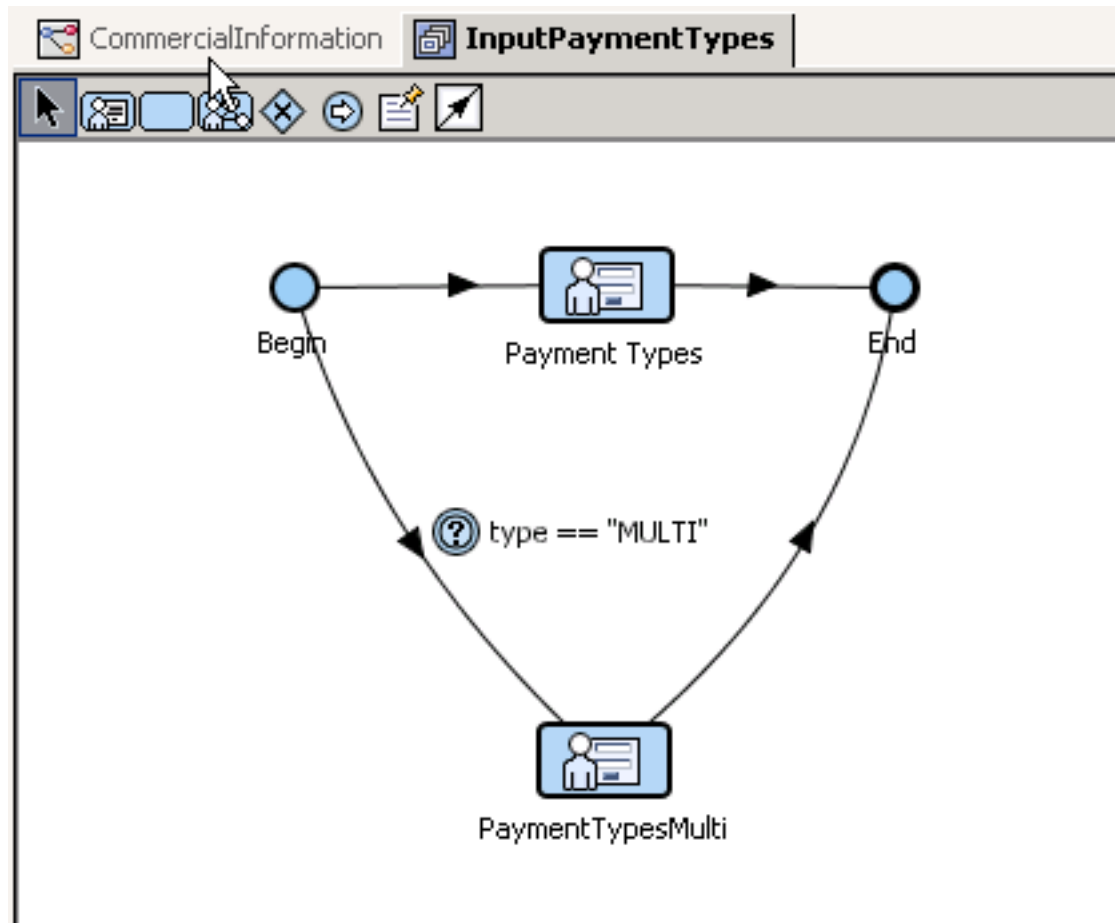
This process has only global activities that invoke to the same screenflow, which executes different Fuego Objects showing different implementations.



Screenflow *InputPaymentTypes*

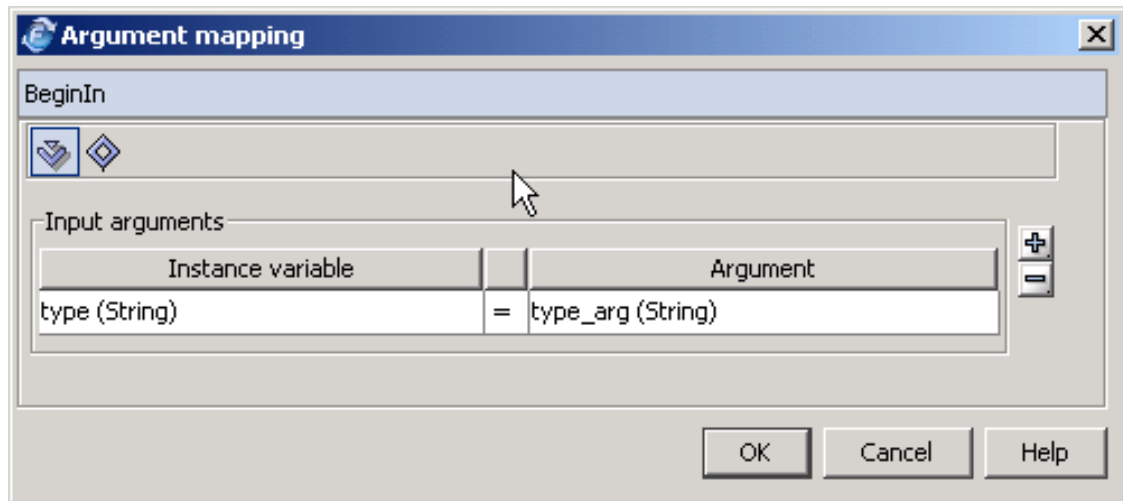
The screenflow, receives as input argument a string which defines which Fuego Object to execute.

- Argument = *ONE*, uses a Fuego Object to add, update or delete Payment types codes from a database. This Fuego Object implementation allows users to work on only one code at a time.
- Argument = *MULTI*, uses a Fuego Object to add, update or delete Payment types codes from a database. This Fuego Object implementation administers all the codes as a group.



Argument mapping is only defined for the Begin activity, as, it does not need any argument to be passed from the screenflow to the global calling activity.

Argument Mapping - Begin Activity



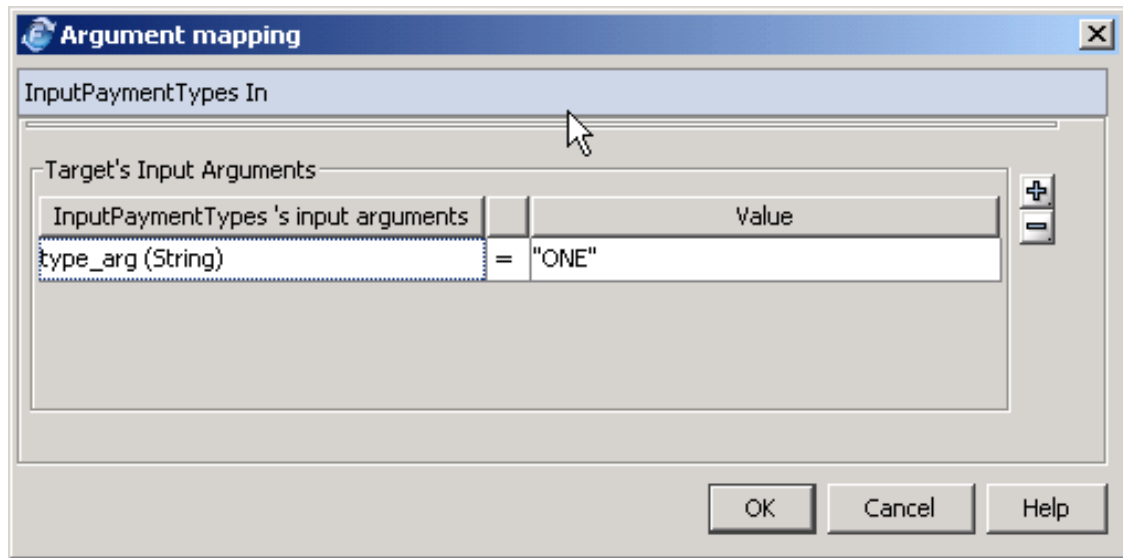
Activities

This process has only global activities that invoke to the same screenflow, which executes different Fuego Objects showing different implementations.

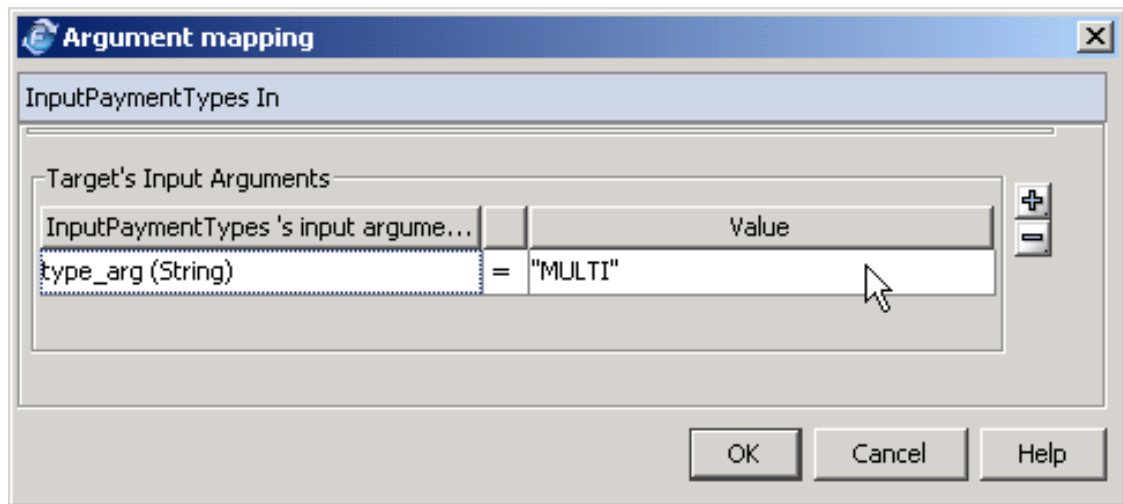
Payment Types Activity: executes the screenflow passing as argument *ONE*.

Payment Types Multiline Activity: executes the screenflow passing as argument *MULTI*.

Argument Mapping - PaymentTypes activity



Argument Mapping - PaymentTypes Multiline activity

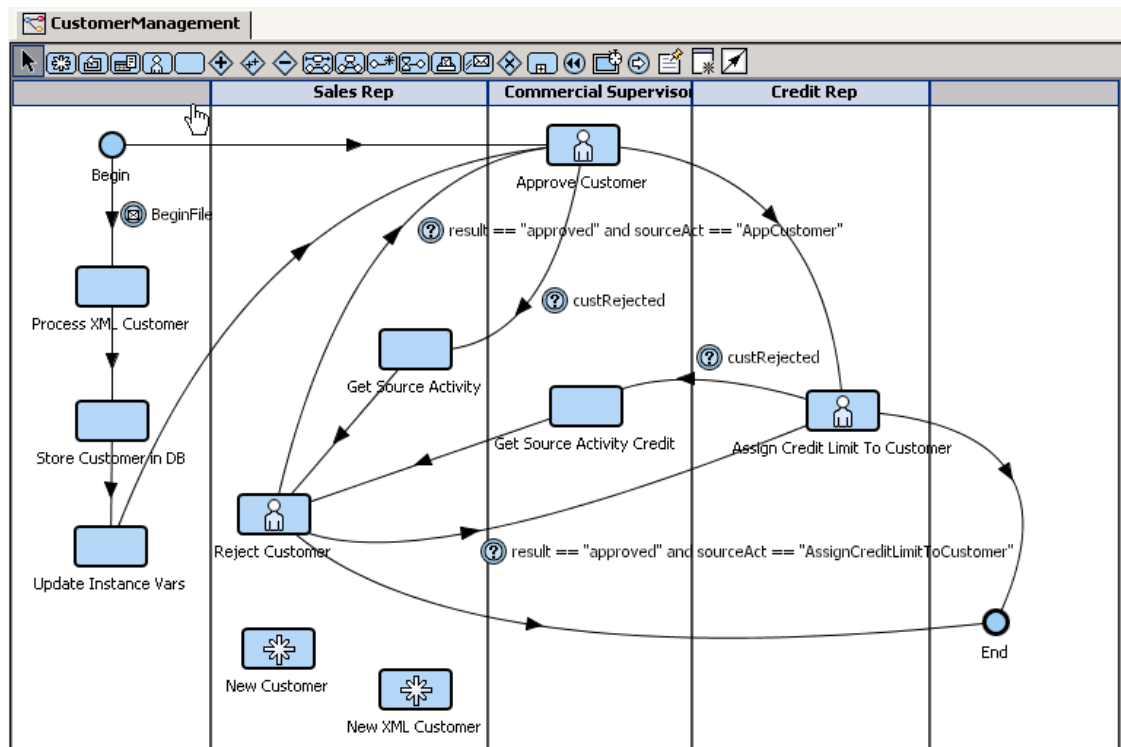


Customer Management

Process

This process models the customer approval or rejection, based on commercial and credit analysis. A new customer can be generated either from an XML file or by entering data using a form. Regardless of the process, the customer is added into the approval flow and can

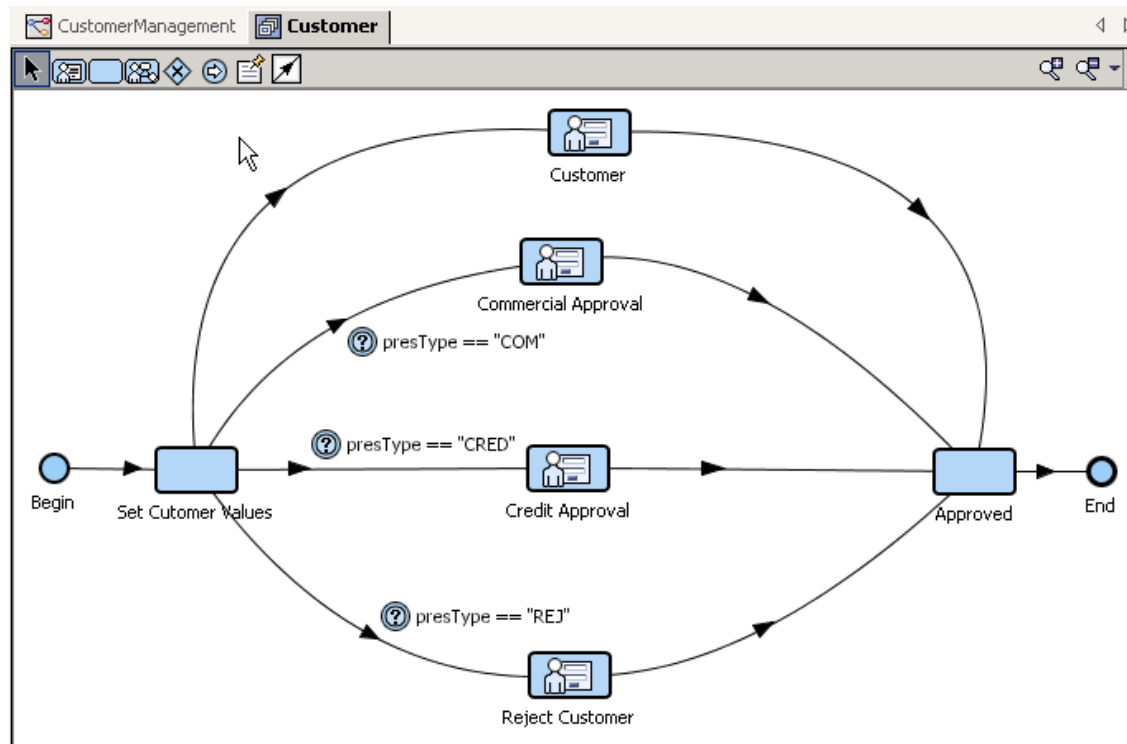
be approved or rejected. If the customer is rejected, the data is sent to the activity that ends the rejection. When the rejection is analyzed, the possibility of getting the customer back to the flow is available. If that is the case, the customer will be automatically sent to the activity from where it was rejected.



The screenflow *Customer* was designed to manage the input, approval or rejection of the customer.

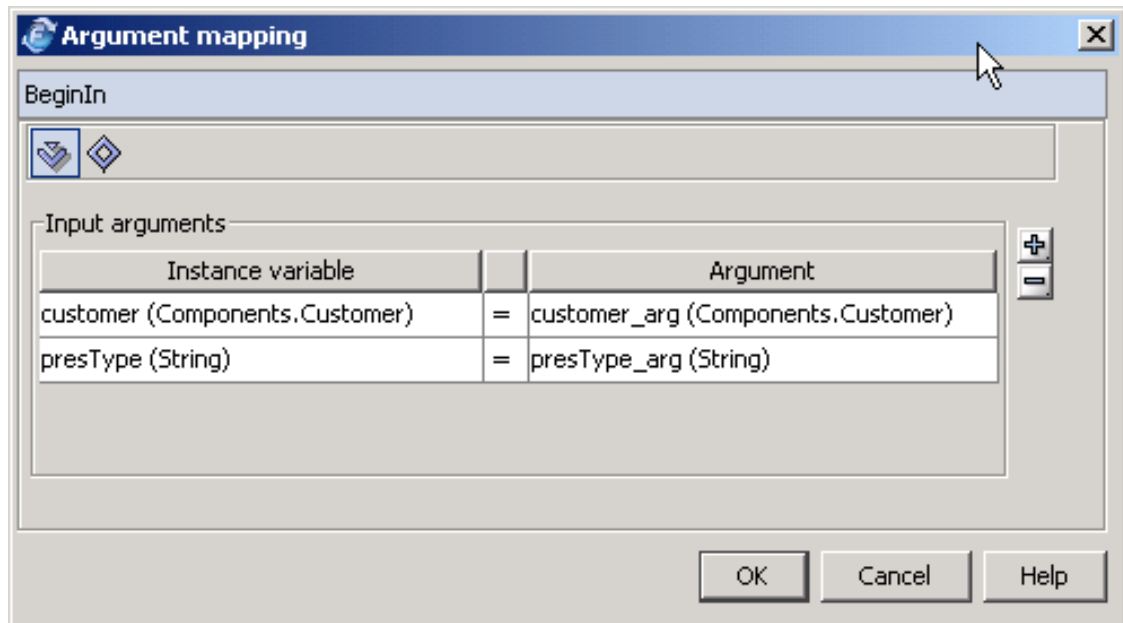
Screenflow *Customer*

This screenflow executes a Fuego Object *Customer*, in the *Components* module of the catalog. This Fuego Object has designed only one presentation to create, approve or reject the customer. Based on this presentation typification (Fuego object attribute *presType*)) the screenflow models the flow to send the customer to the corresponding interactive component call activity.

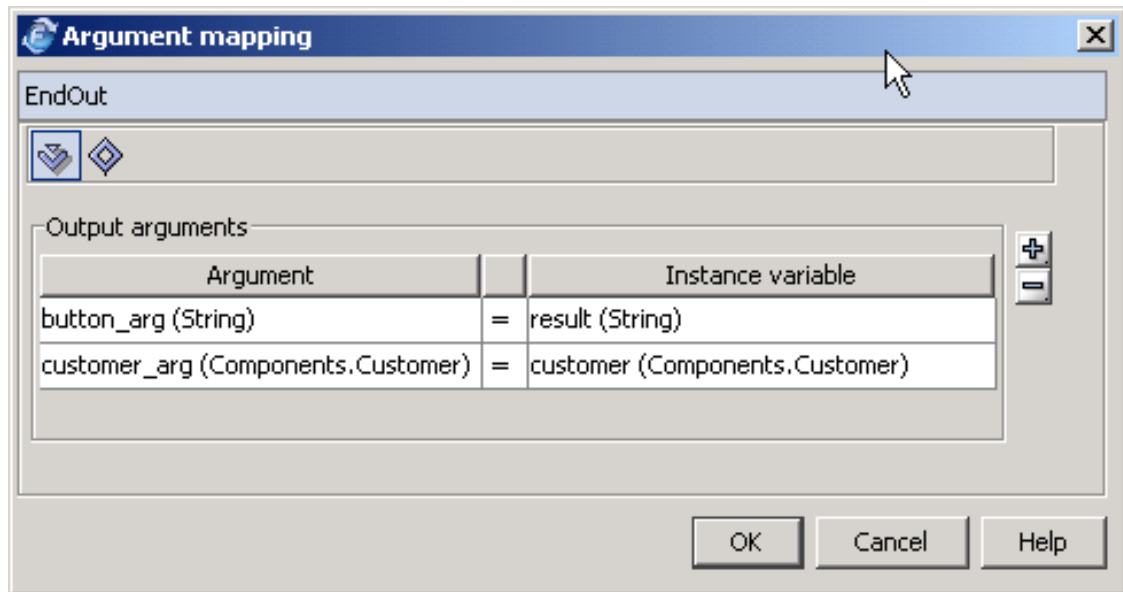


Argument Mapping

- **Begin activity**, receives two arguments:
 - customer: Fuego Object Customer, created in the same argument mapping execution.
 - prestype: String which represents the presentation type mode to execute the Customer Fuego Object, *NEW*, *COM*, *CRED* or *REJ*.

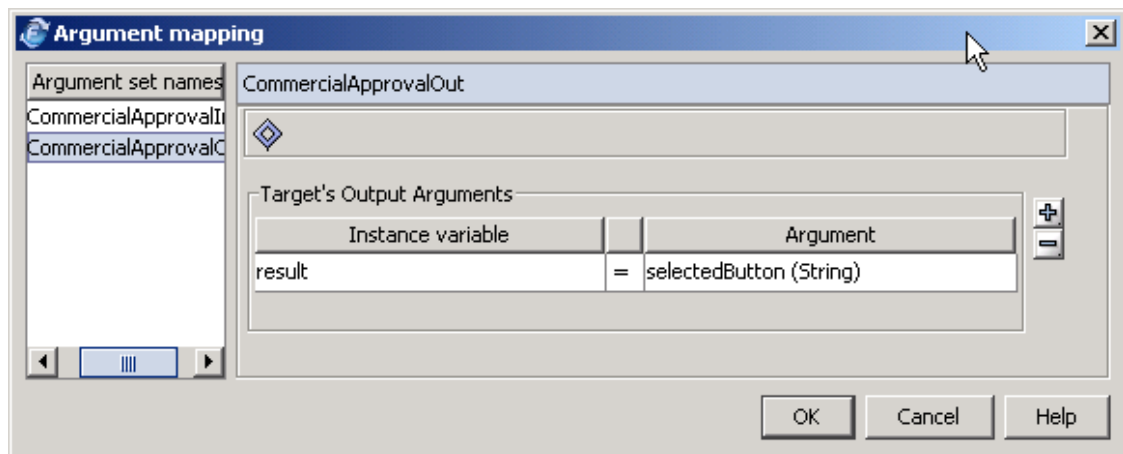


- **End activity**, passes as arguments:
 - customer: Fuego Object CUsermer, updated.
 - result: string that represents the action taken on the customer, "rejected", *approved*, *submit*, or null (""). This string is the value of the selected button in the presentation of the fuego object.



Activities

Customer, *Commercial Approval*, *Credit Approval* and *Reject customer*: execute the *Customer.CustomerPres* presentation in an specific mode. As output argument, they returned the *selectedButton* in the presentation. This value is used later in the *Approved* activity to update the database and set the *action* predefined variable.



Approved activity, stores or removes the customer depending on the taken action, and sets the *action* predefined variable as *OK* or *CANCEL*. This predefined variable value set within the screenflow propagates to the calling process. Find the code below.

```
if result == "approved" then
    store customer
    action = Action.OK
end
if result == "rejected" and customer.prestype == "REJ" then
    remove customer
    action = Action.OK
end
if result == "" then
    action = Action.CANCEL
end
if result == "submit" then
    action = Action.OK
end
```

Activities

All interactive activities invoke the same screenflow *Customer*. However, depending on the activity, some presentation fields have to be enabled and others not. This behaviour is set to the screenflow in the argument mapping with the value of the fuego object attribute *presType*. For example, the credit limit field will only be enabled in the Credit approval activity. This behavior is achieved by setting properties dynamically in the presentation initialization method. The Fuego Object used in this process' activities is *Customer*.

New Customer Activity: A new customer is added to the database and to the customer approval flow. The Fuego Object presentation does not allow (in this step) setting the *discount* and the *credit limit*.

Approve Customer Activity: Through this activity, the user approves or rejects the customer from a commercial point of view through the execution of the *Customer* screenflow. He or she can add information related to commercial issues, such as discount, customer type, code payment type. In this step, the *credit limit* is not enabled.

Assign Credit Limit to Customer Activity: From this activity, the user approves or rejects customers' credit limits through the execution of the *Customer* screenflow. The maximum amount to sell to the

customer can only be set in this activity. In this step, all customer information is editable.

Reject Customer Activity: The customer is rejected or sent back to the approval flow through the execution of the *Customer* screenflow. If the customer is rejected, the corresponding record in the database is removed. In this step, all the customer information is read only.

New XML Customer Activity: This activity allows, by browsing the file system, to process the customer data received in XML format.

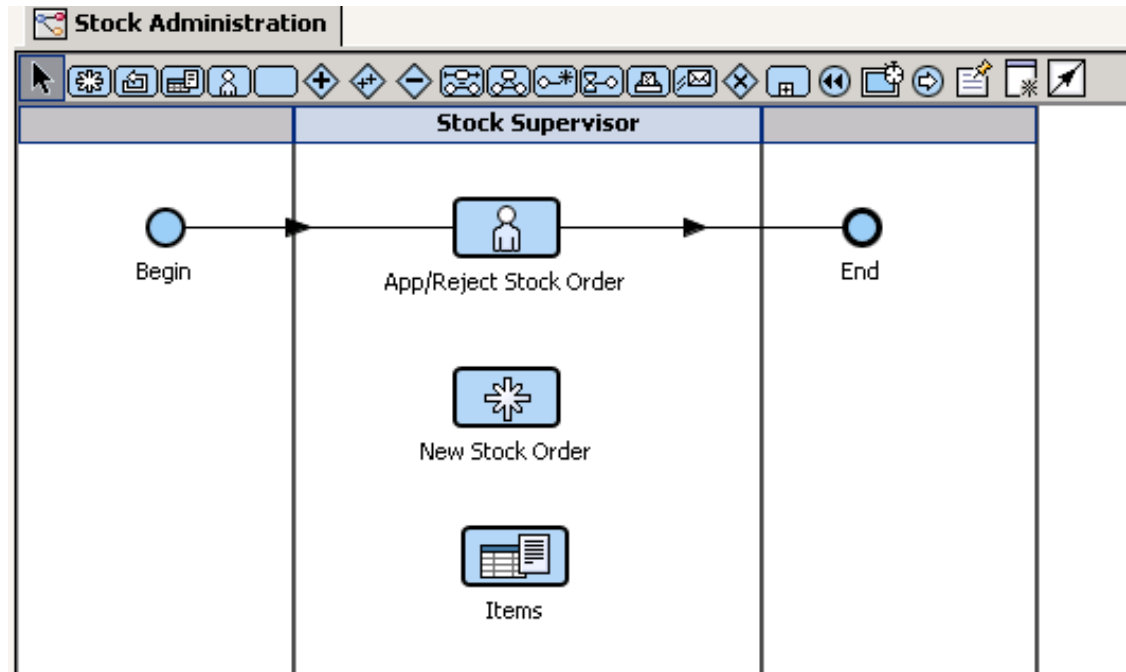
Process XML Customer Activity: This activity loads a customer in XML data format and loads it in an instance variable, making it available to the next activity.

Store Customer in DB Activity: This activity implements a *Component method*, in this case, a Fuego Object method. Implementing argument mapping between the instance variable that contains the customer in XML format and the input argument of the method. See section **Invoking a Fuego Object in a Component Task** in the Designing Using Fuego Objects topic.

Update Instance Vars: This activity updated the instance variables needed in the rest of the approval flow. *description* and *customerID*.

Stock Administration

This is a simple process that allows administration of the items catalog and the stock orders.



Two screenflow were designed:

- Input Items: To administrate the stock item catalog.
- Input Stock Order: To administrate the Stock Orders regardless to their creation and approval.

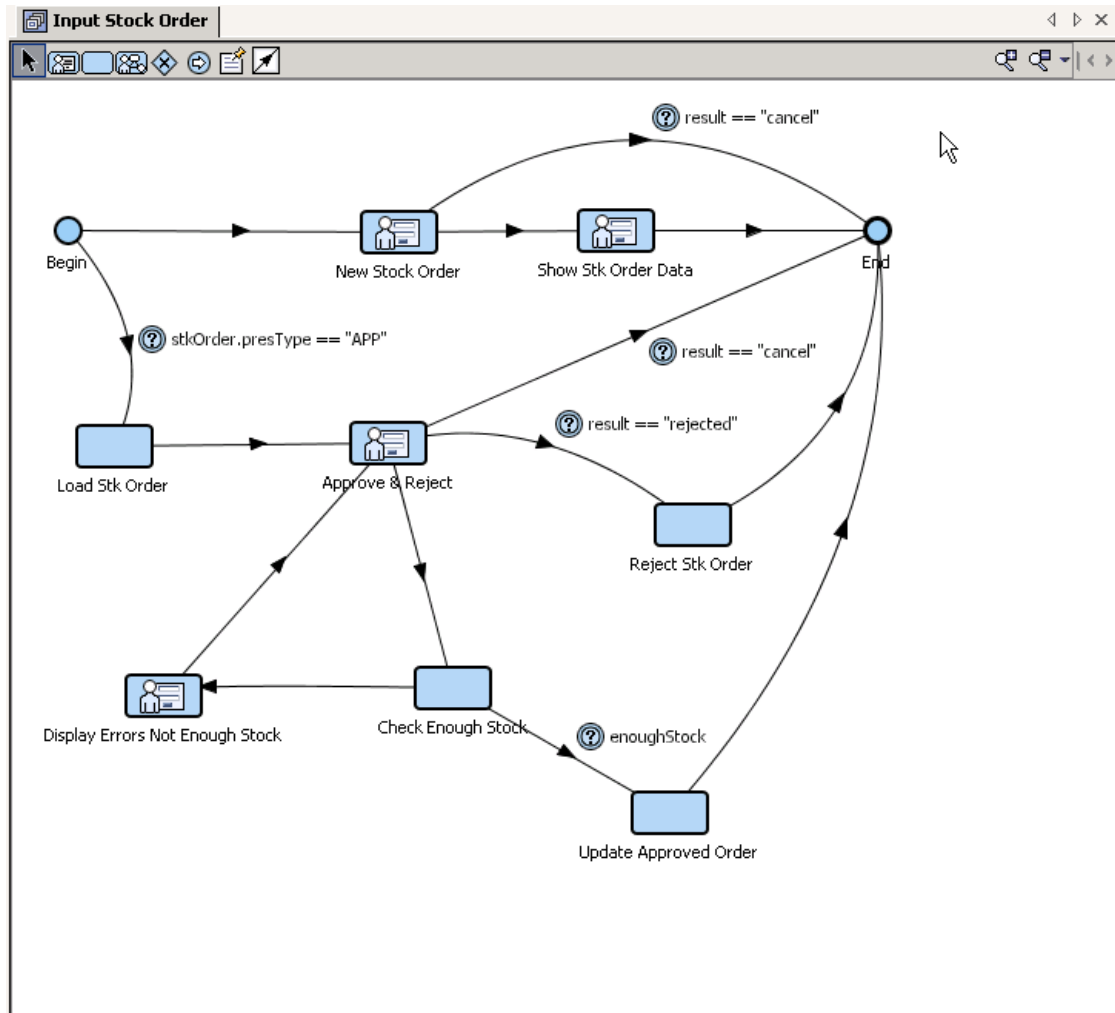
Screenflow *Input Items*

It is a very simple screenflow, with no in/out argument, which replaces an input statement in the calling process by invoking the Fuego Object *Component.Items*.

Screenflow *Input Stock Order*

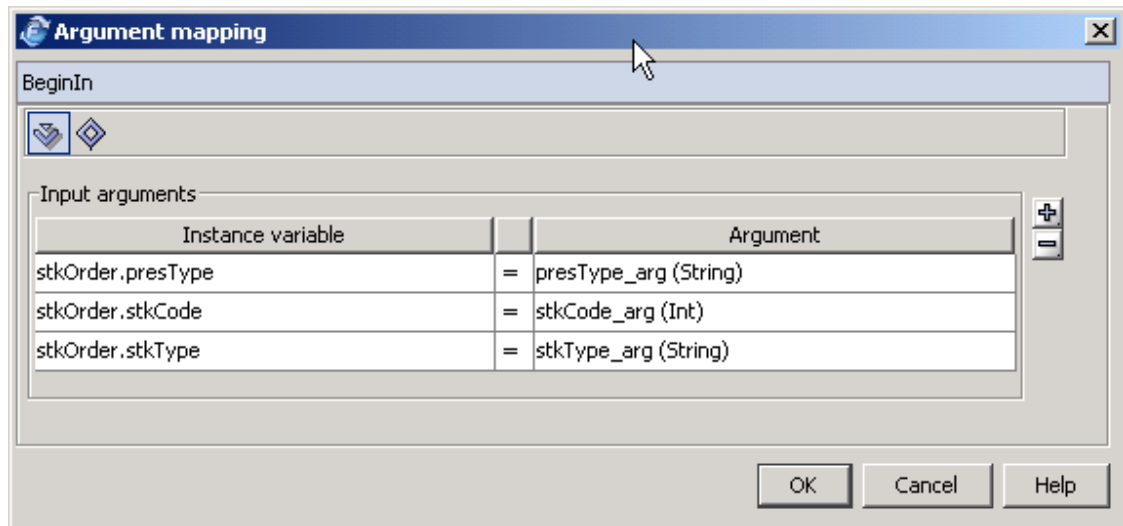
This screenflow executes a Fuego Object *StockOrders*, in the *Components* module of the catalog. This Fuego Object has designed only one presentation to create, approve or reject the stock order. Based on this presentation typification (Fuego object attribute *presType*)) the screenflow models the flow to send the stock order to the

corresponding interactive component call activity.

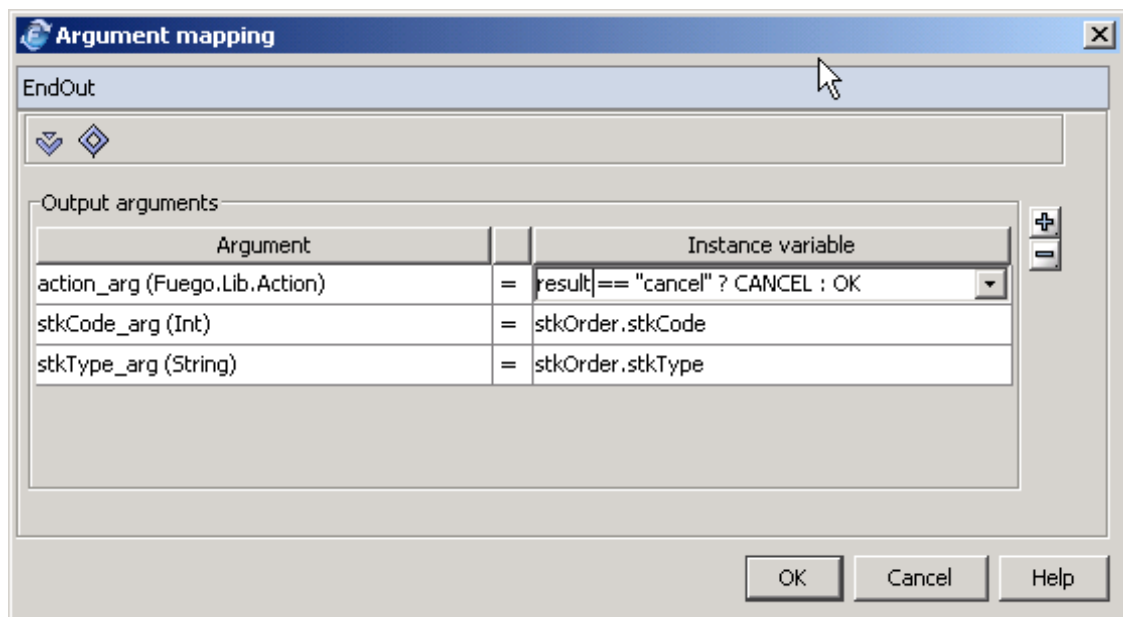


Argument Mapping

- **Begin activity**, receives two arguments:
 - **presType**: String that represents the presentation type mode to execute the Stock Order Fuego Object, *APP* Approve the Stock Order, *NEW* Input a new Stock Order.
 - **stkCode & stkType**: Code and type of the stock order. When *presType* is *NEW* the values are null.



- **End activity**, passes as arguments:
 - stkCode & stkType: Code and type of the stock order generated or precessed. These values are important when the screenflow was executed in *NEW* mode.



Activities

- **New Stock Order:** inputs a new Stock Order using a form. Stock Orders are added but not confirmed. This means that they must always be approved to make it effective. The Fuego Object used is `StkOrders`. The Type and Code of the stock order generated is shown in the next interactive call activity "Show Stk Order Data".
- **Load Stk Order:** if the presentation Type is *APP* then the stock order is loaded into the instance variable *stkOrder*.
- **Approve & Reject:** Approves or rejects the Stock order using the same `StkOrders` Fuego Object loaded in the *stkOrder* instance variable..

If the Stock Order is rejected, then the Stock Order in the database is saved with a special status *DISC*. This is done in the *Reject Stk Order* activity. If the order is approved, then it is made effective verifying previously the existence of stock in case the stock order is of type *OUT*..

When the Stock order is an *IN* type (inputs stock), then no validation is required and the stock is updated adding the quantities specified in the order for each item. However, if the stock order is an *OUT* type, a validation to assure that there is enough stock for each item in the order is done. This validation is done in the *Check Enough Stock* automatic activity. When the validation is ok, the order is saved with the *APP* status and the stock is updated. When there is not enough stock to make the order effective, a message is displayed indicating which items request more stock than is available on hand in the activity *Display errors Not enough stock*. The stock order will be continuously displayed until:

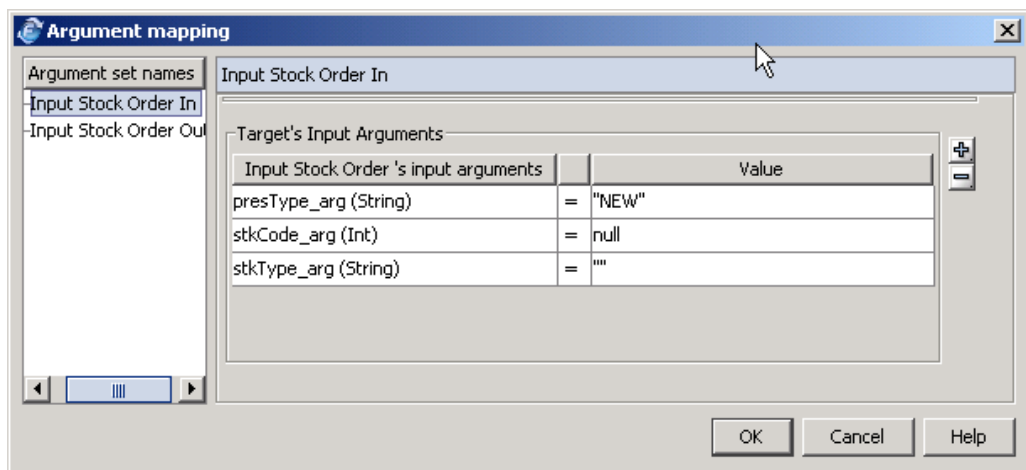
- its execution is cancelled, by selecting the cancel button in the presentation, which returns the string "standby" and in which is based the output argument setting of the predefined variable

action,

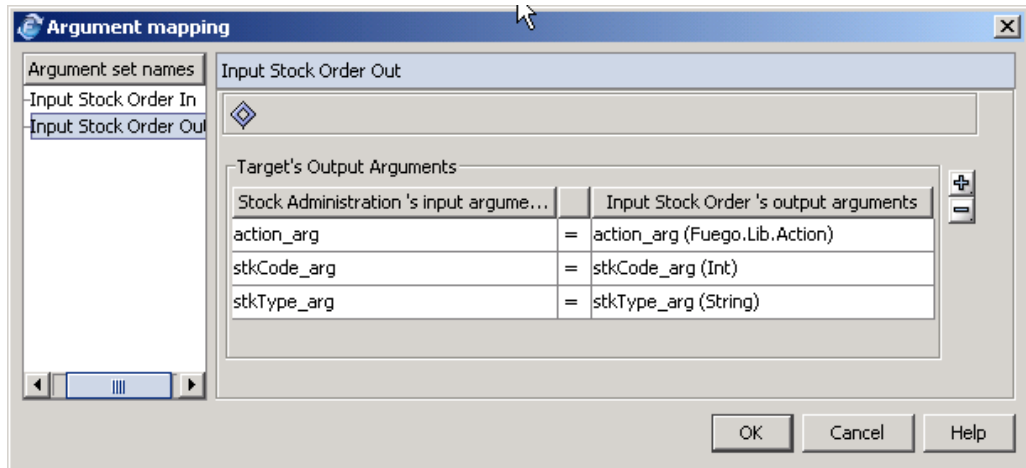
- an approve is performed and there is enough stock, or
- the order is rejected.

Activities

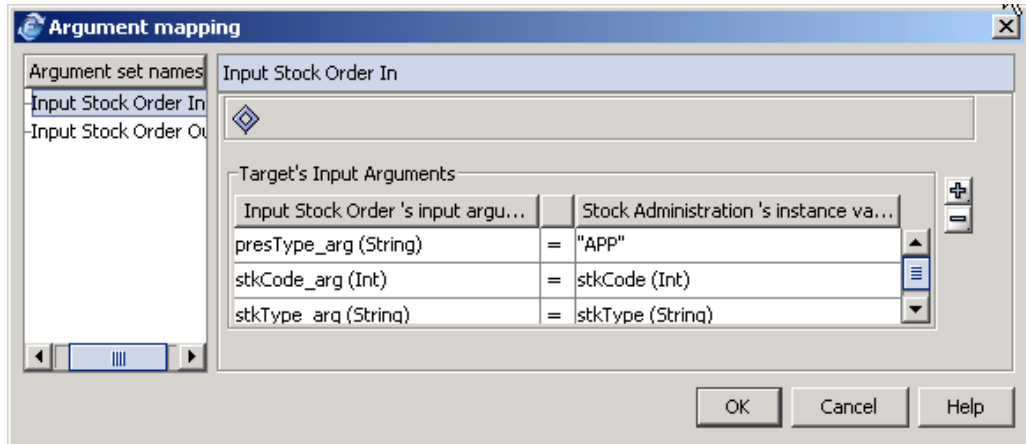
- **Items Activity:** Invokes the *Input Items* screenflow.
- **New Stock Order:** inputs a new Stock Order by invoking the *Input Stock Order* screenflow with argument *NEW*.
- Input argument mapping:



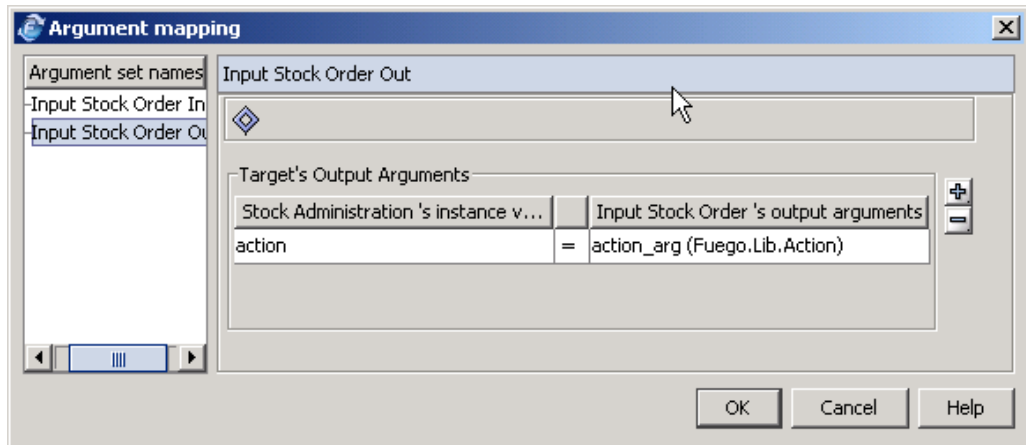
- Output argument mapping:



- App/Reject Stock Order: Approves or rejects the stock order by invoking the *Input Stock Order* screenflow with argument *APP*..
- Input argument mapping:



- Output argument mapping:



Project Catalog

Fuego Objects

In this example we have implemented all Fuego Object types, non-presentable ones: used as a *library*, heir basic ones used both as basic Fuego Object inner of another and presentable Fuego Objects to users, and presentable ones using the previous ones as inner objects.

Non-Presentable Fuego Object

In our example, we have defined two non-presentable Fuego Objects. The *Utilities* fuego object, used as a *library* with methods and attributes that can be used from other Fuego Objects. And the *CustomerFO* Fuego Object which it's only purpose is to store a *CUSTOMER* into the database converting data received in XML format.

- *Utilities* provides:
 - *loadItemCodes* method, that fills its *items* attributes with the cataloged items in the *ITEMS* database table.

- *getItemCodes* method, that fills its *items* attribute using the *loadItemCodes*, only if the attribute is empty.
- *showItems* method, that returns an associative array ready to use in a valid values invocation.
- *getItemDescription* method, that retrieves and returns the description from the database of a given item.
- *loadPaymentTypes* method, that fills its *paytypes* attributes with the cataloged payment types in the *PAYTYPE* database table.
- *getPayCodes* method, that fills its *paytypes* attributes using the *loadPaymentTypes*, only if the attribute is empty.
- *CustomerFO* provides:
 - *customerFromXML* method, that creates a *customerFO*, which inherits behavior from the *CUSTOMER* table, and stores it into the database.
- *StkNum* provides:
 - returns the next code to generate by order type (IN/OUT), through the method *nextCode*.
 - updates the last code generated by order type (IN/OUT), through the method *updateLastCode*.
- *CustomerMINI* is used to load Customer's basic information.

Heir Fuego Objects

PListH and PriceList

These two Fuego Objects, inherit behavior from the tables *PLISTH* and *PRICELIST*, that conformed the business object Pricing List. *PLISTH* contains the header and *PRICELIST* contains the lines of the Pricing List.

PType

Inherits behavior from *PAYTYPE* and is used as inner object from the *PTypes* Fuego Object.

PayType

Inherits behavior from *PAYTYPE* and implements a presentation to manipulate only one Payment at a time.

Customer

Inherits behavior from the *CUSTOMER* SQL component. It is also presentable and is used through all activities in the *Customer Management* process. Its only presentation has been built providing multi functionality according to the different business requirements of the approval flow.

Item

Inherits behavior from the *ITEMS* SQL component.

CustomerFO

Inherits behavior from the *CUSTOMER* SQL component, and implements a method to store it building it from an XML customer component introspected in the Project catalog.

StkOrdit

Inherits behavior from the *STKORDIT* SQL component.

StkOrder

Inherits behavior from the *STKORDER* SQL component.

StkNum

Inherits behavior from the *STKNUM* SQL component.

To define these heir Fuego Objects and after introspecting the database tables into your project catalog (see SQL Components), right-click on the introspected SQL component corresponding to each table, and select the option **Create Heir**. For further information about heir Fuego Objects and how to create them, please refer to Implementing Behavior Inheritance.

Presentable Fuego Objects

Presentable Fuego Objects, are those with at least a presentation defined. In our example, the Presentable objects are:

- PayType
- Items
- PTypes
- Customer
- StkOrders
- OrderComp

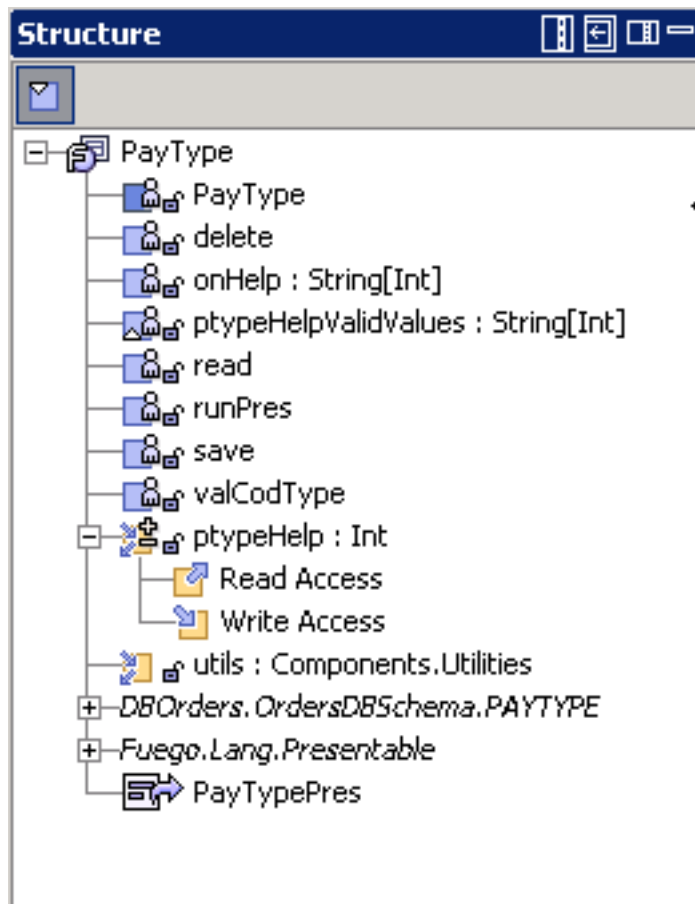
PayType

This example presents:

- behavior inheritance implementation
- dynamic valid values with description
- uses of non-presentable Fuego Objects as *lib*
- inner Fuego Objects as attributes

- refreshValidValues() standard method
- action buttons

This Fuego Object was constructed to add, update, and delete one Payment Type code at a time.



The requirement was to provide the end user a way to add, update and delete Payment type codes by working only on one of them at a time, providing:

- update and removal of existing codes
- addition of new codes

- visualize a list with the existing codes and select one of them
- work, adding, update and removing, without leaving the application after each action

Attributes

- *codpay*, *paydescr*, *paydisc*: inherited attributes from table *PAYTYPE*.
- *ptypeHelp*: this attribute is referenced from the presentation and used to show a dynamic set of valid values. It has dynamic valid values with description retrieved by the *onHelp* method. It has both, the read and write access methods redefined, to retrieve or set the value of the attribute *codpay*. It is used as a view of the real *codpay* attribute.
- *utils*: by defining this attribute the access to the *lib* methods defined in the *Utilities* Fuego object is enabled.

Methods

- *delete*: this method redefines the remove action inherited from the SQL component. The redefinition was necessary to make a reset of the presentation after, a remove is done, to let the user keep on working and refresh the valid values of the existing codes, as the recently removed must disappear from the list of valid values.
- *onHelp*: this method is used to retrieve the list of payment type codes valid values. Referenced from the *ptypeHelp* attribute. It has been implemented by using the *getPayCodes* method, defined in the *Utilities* fuego object. Returns an associative array of type **String[Int]** as is used for a dynamic list with descriptions for an attribute which type is **Int**.

- **read**: this method executes the *load()* method or the *reset()* method depending on the value of the attribute *pTypeHelp*. As this attribute can be set with the null value, the implementation defined is, if the user selects the **null** value then, a new code is to be added, if not, the code has to be retrieved and displayed.
- **save**: this method redefines the store action inherited from the SQL component. The redefinition was necessary to make a reset of the presentation after, a store is done, to let the user keep on working and refresh the valid values of the existing codes, as the recently added code (if new) must appear in the list of valid values.
- **valCodType**: this method searches the payment type code hold in the *codpay* attribute. If this attribute does not exist in the list of valid values, then a new code is about to be added, if not it has to be retrieved and displayed. The search is done using the *Utilities.getPayCodes*, this method has been optimized, to access to the database only when they are not loaded. The list of valid values is always updated because when a *save* or *delete* is done the *refreshValidValues* of the *pTypeHelp* attribute is executed.

Presentation

The presentation defined is named **PayTypesPres**.

Components

- **codpay**:
 - References: **codpay**, this field can be set by editing it, or by selecting a value from the list of valid values in the *pTypeHelp* component.
 - on Change invokes: **valCodType()**, by this invocation it is determined if a load or a reset must be done, according to the

value set. If the value exists, then a load is done, if it does not exist, a refresh has to be done as the value is a new one to be added.

- *ptypeHelp*:
 - References: **ptypeHelp**, as this field has the read and access method redefined, it is like a view of the real *codpay* attribute. When a value is selected from its valid values combo, it is set in as well in the *codpay* attribute.
 - on Change invokes: **read()**, when a value is selected from the combo box, it is loaded, if it is the null value, a reset is performed.
- *padydescr*:
 - References: **paydescr**
- *paydisc*
 - References: **paydisc**

Buttons

- *save*: action button
 - method invoked: **save**, it stores the code, resets the presentation, and refresh the valid values.

- *cancel*: cancel button, to exit the presentation.
- *reset*: reset button, the user can choose to clear the presentation.
- *delete*: action button
 - method invoked: **delete**, it removes the code, resets the presentation, and refreshes the valid values.

Execution

The screenshot shows a web application titled "FUEGO Work Portal" with a navigation bar containing "Welcome, fuego", "Search", "Options", "Help", and "Logout". The main content area displays a form titled "Payment Type". The form contains three input fields: "Payment Type" with the value "3", "Description" with the value "CASH", and "Discount" with the value "10". A dropdown menu is open for the "Payment Type" field, showing a list of options: "CASH", "CCARD", "DCARD", "CASH" (highlighted), and "CHECK". Below the form are four buttons: "Save", "Exit", "Reset", and "Remove". The footer of the page reads "FuegoBPM™ - Work Portal".

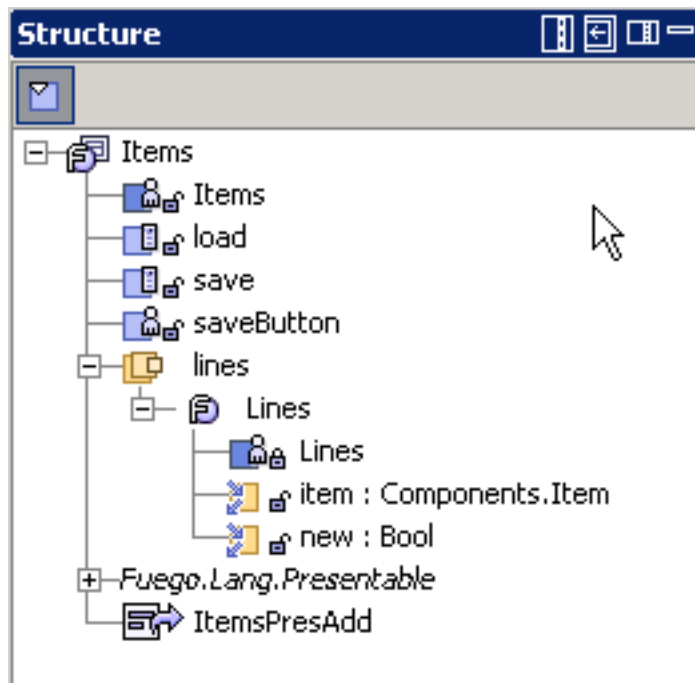
Items and PayTypes

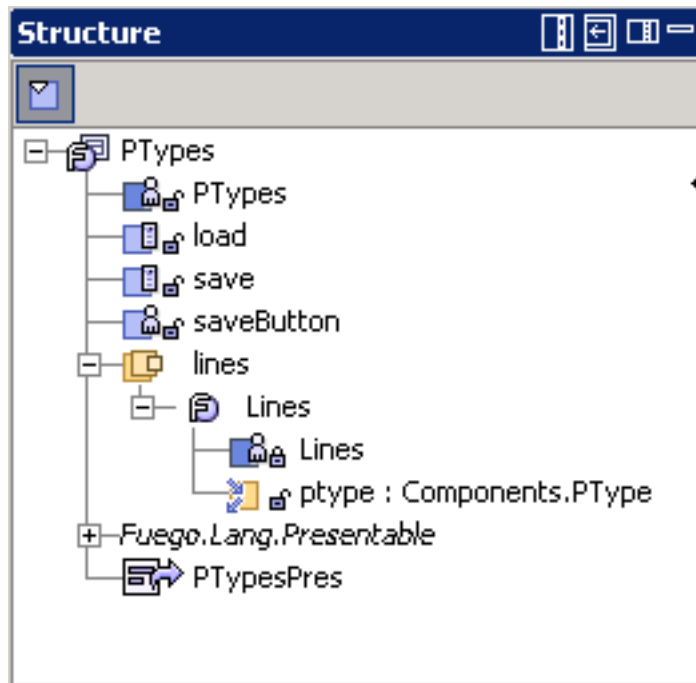
This example presents:

- behavior inheritance implementation
- changing the constructor method
- dynamic valid values with description

- uses of non-presentable Fuego Objects as *lib*
- inner Fuego Objects as attributes
- group attributes
- refreshValidValues() standard method
- action buttons
- array component in presentations
- using inner Fuego Objects from the presentation

These Fuego Objects were constructed to add, update, and delete Items and Payment Types as a group.





The implementation in this case had been done working with all the existing Items/Payment Types at the same time. In order to do this, new Fuego Objects were built to hold all the existing elements in a group attribute, on row each. The main attribute of the group is a component type that inherits behavior from the corresponding SQL Component, *PAYTYPE* and *ITEMS*.

The only difference in the implementation of these two Fuego Objects, is that the Items one, has to create an entry in the *STOCK* table, when a new item is added.

Attributes

- *lines*: group attribute (same name in both Fuego Objects). Each row has is an attribute typified as the basic Fuego Object that inherits behavior from SQL Component, *PType/PAYTYPE* and *Item/ITEMS*.
 - *PTypes* Fuego Object: the *lines* attribute contains a *ptype* attribute of type *PType*.

- *Items* Fuego Object: the *lines* attribute contains an *item* attribute of type *Item*, and the *new* attribute of type Bool, that indicates if the row is new or not. This is used to determine if the row correspondes to a new item code, to create or not the *STOCK* entry with 0.

Methods

- *load*: this method loads using an embedded SQL component all the database records and holds each of them in a row of the group attribute, name in both Fuego Objects *lines*. In the *Item* FO, the **new** attribute is set to false.
- *save*: in this method a store is executed for each element in the *lines* attribute. In the *Item* FO, for those lines with the *new* attribute in true, an entry is generated in the *STOCK* table with value 0.
- *saveButton*: this method is invoked from the action button *save*. The method *save* cannot be invoked as it *runs on server*.
- *constructor: PTypes()/Items()*: in this case, the constructor executes the *load* method, to create the Fuego Object with its group attribute full. When the presentation is opened the array will already contained the existing codes in the database.

Presentation

Both presentations have been built with an array that references to the *lines* group attribute of the Fuego Object.

Each component within each row of the array, references to its equivalent within the inner object attribute of the group.

Components

PTypes presentation: **PTypesPres**

- *rows*, array component, references to *lines* group attribute of the Fuego Object.
- *codpay*, references to *lines[].ptype.codpay*.
- *padydescr*, references to *lines[].ptype.paydescr*.
- *paydisc*, references to *lines[].ptype.paydisc*.

Items presentation: **ItemsPresAdd**

- *rows*, array component, references to *lines* group attribute of the Fuego Object.
- *itemcode*, references to *lines[].item.itemcode*.
- *itemdescr*, references to *lines[].item.itemcode*.

Buttons

- *save*: action button.
 - method invoked: **saveButton**, it stores all rows contained in the group attribute, executing a previous delete of all the existing ones. In the *Items* FO an entry with value 0 is generated in the *STOCK* table.
- *cancel*: cancel button, to exit the presentation.

Execution

Payment Types:

Execution

The screenshot shows the FUEGO Work Portal interface. At the top, there is a header bar with the FUEGO logo, the text "Work Portal", a welcome message "Welcome, fuego", and navigation links "Search - Options - Help - Logout". Below the header, a "Payment Types" dialog box is displayed. This dialog box contains a table with four columns: "Payment Type", "Descrip", and "Discount". There are four rows of data, each with a selection icon on the left. At the bottom of the dialog box, there are "Save" and "Cancel" buttons. The footer of the portal reads "FuegoBPM™ - Work Portal".

	Payment Type	Descrip	Discount
1	1	CCARD	20
2	2	DCARD	15
3	3	CASH	10
4	4	CHECK	5

Items:

The screenshot shows the FUEGO Work Portal interface. At the top, there is a header bar with the FUEGO logo, the text 'Work Portal', a welcome message 'Welcome, fuego', and navigation links 'Search - Options - Help - Logout'. Below the header, the main content area displays a form titled 'Items'. The form contains a table with two columns: 'Item Code' and 'Description'. The table has three rows of data. Below the table, there are two buttons: 'Store' and 'Exit'.

	Item Code	Description
1	1010	Item1010
2	1020	Item1020
3	1030	Item1030

At the bottom of the form, there are two buttons: 'Store' and 'Exit'.

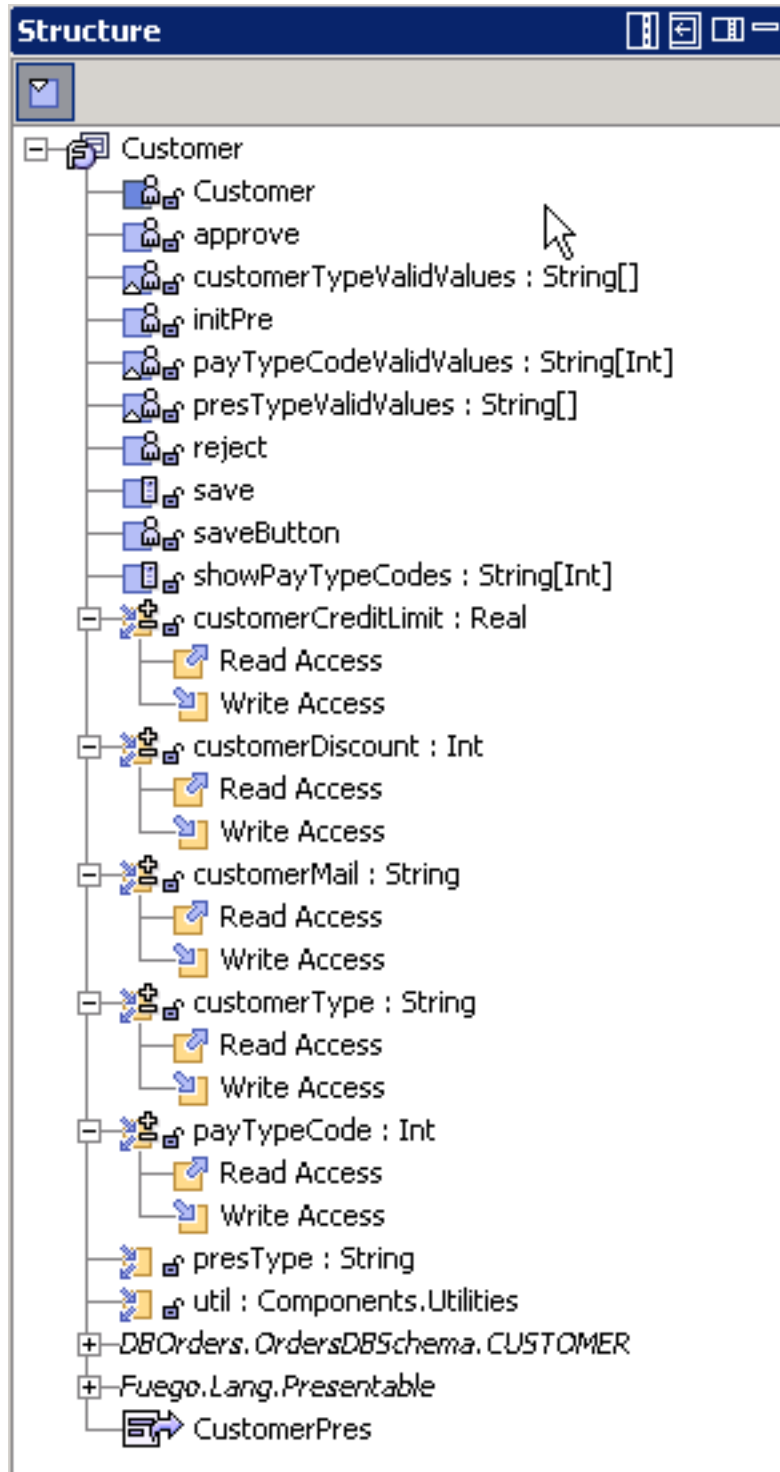
Customer

This example presents:

- behavior inheritance implementation
- dynamic valid values with description
- static valid values
- uses of non-presentable Fuego Objects as *lib*
- inner Fuego Objects as attributes
- group attributes
- usage of the *refreshValidValues* standard method
- defining a Initialization Method
- dynamic set of presentation properties

- action buttons
- array component in presentations
- using inner Fuego Objects from the presentation
- input patterns
- check expression and presentation applicability

This Fuego Object was constructed for customer data input and view purposes. Its presentation does not store or remove the customer from the database. This control has been delegated to the activities where it is used.



This Fuego Objects implements behavior inheritance from the *CUSTOMER* SQL component. This Fuego Object and its presentation are used through all the activities of the customer approval flow. The

same presentation has been designed to be used in all the activities, by enabling or disabling attributes and buttons.

Attributes

- *custcode, custname, codpay, custdisc, custtype, billadd, shippadd, credlimit, mail*: inherited attributes from table *CUSTOMER*.

The next five attributes have been defined in the Fuego Object to be referenced from the presentation. They give the capability of enabling and disabling, and make them mandatory according to activity where the presentation is used. All of them have the *read* and *write* access overwritten to return and set the value from and to the real attribute.

- *customerDiscount*: represents *custdisc*.
- *customerCreditLimit*: represents *credlimit*.
- *payTypeCode*: represents *codpay*. It has a dynamic with description valid values setting. The method *showPayTypeCodes* is used to retrieve the list of valid values.
- *customerMail*: represents *mail*.
- *customerType*: represents *custtype*. It has a list of static valid values. The ones are the same defined in our database example: *INTERNAT, NATIONAL, GLOBAL*.
- *presType*: This attribute is used to indicate what type of action is going to be done through the presentation usage. It has defined a static list of valid values. According to its value, some presentation components properties are dynamically set in the method invoked when the presentation initiates (see in this example: method *initPre*). This attribute is set before the input

statement is used in a BP_Method. See BP_Methods: *newCustomer*, *appCustomer*, *assignCreditLimitToCustomer* and *rejectCustomer*.

- **NEW**: this value is set when it is going to be used as an *add*.
- **COM**: this value is set when it is going to be commercially approved.
- **CRED**: this value is set when it is going to be approved by credit.
- **REJ**: this value is set when its rejection is being analyzed.
- *utils*: by defining this attribute the access to the *lib* methods defined in the *Utilities* Fuego object is enabled.

Methods

- *initPre*: this method is invoked in the *Initialization Method* property of the presentation *CustomerPres*. It gives the presentation functionality according to the value of the *presType* attribute enabling/disabling buttons, making attributes editable or not and setting the presentation name.
- **NEW** (New Customer)
 - disables buttons: *approve* and *reject*
 - makes not editable the fields: *credlimit* and *customerDiscount*
 - dynamically sets the presentation name to: *CustomerPres* = *New Customer*
- **COM** (Commercial approval)

- disables buttons: *submit*
- makes not editable the fields: *credlimit* and *custcode*
- sets the presentation name to: *CustomerPres = Customer Commercial Approval*
- executes the *load* inherited method
- **CRED** (Credit approval)
 - disables buttons: *submit*
 - makes not editable the fields: *custcode*
 - sets the presentation name to: *CustomerPres = Customer Credit Approval*
 - executes the *load* inherited method
- **REJ** (Rejection)
 - disables buttons: *submit*
 - makes not editable all presentation fields
 - sets the presentation name to: *CustomerPres = Reject Customer or Send Back to Processing?*
 - executes the *load* inherited method
- *save*: this method invokes the *store* method inherited from *CUSTOMER*. It searches the next customer code to generate it.
- *saveButton*: As the *save* method is defined as *runs on server*, the

present method has to be defined to be invoked from the presentation, calling the *save* and the *submit* methods.

- *showPayTypeCodes*: this method is used to retrieve the list of payment type codes valid values. Referenced from the *ptypeHelp* attribute. It has been implemented by using the *getPayCodes* method, defined in the *Utilities* Fuego Object. Returns an associative array of type **String[Int]** as is used for a dynamic list with descriptions for an attribute which type is **Int**.

See the usage of these two methods in the section **Fuego Object Methods - Action methods** in this page.

- *approve*
- *reject*

Presentation

Components

- *custcode*: references *custcode* attribute. It is always as *not editable*.
- *custname*: references *custname* attribute.
- *payTypeCode*: references *payTypeCode* attribute.
- *customerDiscount*: references *customerDiscount* attribute.
- *customerType*: references *customerType* attribute.
- *billadd*: references *billadd* attribute.
- *shipadd*: references *shipadd* attribute.
- *creditlimit*: references *customerCreditLimit* attribute. Validates not to

be 0 by implementing a **Check Expression**.

- *mail*: references *mail* attribute. Implements an **input pattern**.

Buttons

- *submit*: action button.
 - method invoked: **saveButton**, it stores the customer into the database using the inherited method **store** and invokes the **submit** method.
- *cancel*: cancel button, to exit presentation.
- *approve*: action button.
 - method invoked: **approve**, it invokes the **submit** method with *approved*.
- *reject*: action button.
 - method invoked: **reject**, it invokes the **submit** method with *rejected*.

Execution

New Customer

The screenshot shows a web browser window titled 'FUEGO Work Portal'. The header bar is blue with the text 'FUEGO Work Portal' on the left, 'Welcome, fuego' in the center, and 'Search - Options - Help - Logout' on the right. A mouse cursor is pointing at the 'Search' link. The main content area contains a form titled 'New Customer'. The form has several input fields and dropdown menus. The 'Customer Name' field is filled with 'Jhon Smith'. The 'Payment Type' dropdown is set to 'CCard'. The 'Customer Type' dropdown is set to 'NATIONAL'. The 'Billing Address' and 'Shipping Address' fields are both filled with 'Park Ave.3910. Springfield'. The 'Mail' field is filled with 'jsmith@smith.com'. At the bottom of the form are two buttons: 'Submit' and 'Cancel'. The footer of the browser window shows 'FuegoBPM™ - Work Portal'.

New Customer	
Customer Code	Customer Name <input type="text" value="Jhon Smith"/>
Payment Type <input type="text" value="CCard"/>	Customer Discount
Customer Type <input type="text" value="NATIONAL"/>	Credit Limit
Billing Address <input type="text" value="Park Ave.3910. Springfield"/>	
Shipping Address <input type="text" value="Park Ave.3910. Springfield"/>	
Mail <input type="text" value="jsmith@smith.com"/>	
<input type="button" value="Submit"/>	<input type="button" value="Cancel"/>

Commercial Customer Approval

The screenshot displays the FUEGO Work Portal interface. At the top, a blue header bar contains the 'FUEGO Work Portal' logo on the left, a 'Welcome, fuego' message in the center, and a navigation menu on the right with links for 'Search', 'Options', 'Help', and 'Logout'. A mouse cursor is positioned over the 'Welcome, fuego' text. Below the header, a central white box with a black border is titled 'Customer Commercial Approval'. This box contains a form with the following fields: 'Customer Code' (value: 17), 'Customer Name' (text input: Jhon Smith), 'Payment Type' (dropdown menu: CCARD), 'Customer Discount' (text input: 5), 'Customer Type' (dropdown menu: NATIONAL), 'Credit Limit' (empty text input), 'Billing Address' (text input: Park Ave. 3910 Springfield), 'Shipping Address' (text input: Park Ave. 3910 Springfield), and 'Mail' (text input: jsmith@smith.com). At the bottom of the form box are three buttons: 'Cancel', 'Approve', and 'Reject'. The bottom of the browser window shows a status bar with the text 'FuegoBPM™ - Work Portal' and a horizontal scrollbar.

Customer Commercial Approval	
Customer Code	17
Customer Name	Jhon Smith
Payment Type	CCARD
Customer Discount	5
Customer Type	NATIONAL
Credit Limit	
Billing Address	Park Ave. 3910 Springfield
Shipping Address	Park Ave. 3910 Springfield
Mail	jsmith@smith.com
<div>Cancel Approve Reject</div>	

Assign Credit Limit to Customer

The screenshot shows a web application titled "FUEGO Work Portal". The header bar includes a welcome message "Welcome, fuego" and navigation links: "Search - Options - Help - Logout". Below the header, there is a "Customer Credit Approval" form. The form contains several input fields and buttons. The fields are organized as follows:

Customer Code	17	Customer Name	Jhon Smith
Payment Type	CCARD	Customer Discount	5
Customer Type	NATIONAL	Credit Limit	1000
Billing Address	Park Ave. 3910 Springfield		
Shipping Address	Park Ave. 3910 Springfield		
Mail	jsmith@smith.com		

At the bottom of the form, there are three buttons: "Cancel", "Approve", and "Reject". The footer of the page reads "FuegoBPM™ - Work Portal".

Reject Customer

The screenshot shows a web application titled "FUEGO Work Portal". The header includes a welcome message "Welcome, fuego" and navigation links: "Search - Options - Help - Logout". Below the header is a toolbar with icons for document, edit, and print. The main content area is a form titled "Customer Credit Approval".

The form contains the following fields and controls:

- Customer Code: 17
- Customer Name: Jhon Smith
- Payment Type: CCARD (dropdown)
- Customer Discount: 5
- Customer Type: NATIONAL (dropdown)
- Credit Limit: 1000
- Billing Address: Park Ave. 3910 Springfield
- Shipping Address: Park Ave. 3910 Springfield
- Mail: jsmith@smith.com

At the bottom of the form are three buttons: "Cancel", "Approve", and "Reject".

The footer of the application reads "FuegoBPM™ - Work Portal".

CustomerFO

This Fuego Object was constructed as a simple example to show through a heir Fuego Object how to add a customer into the database from a customer in XML data format.

Attributes, None.

Methods

customerFromXML: input argument: *custXML*: *Components.CustomerXML.Customer*. Creates a customer and stores it into the database. This method is invoked in the *Store Customer in DB* Activity, as a Component Task.

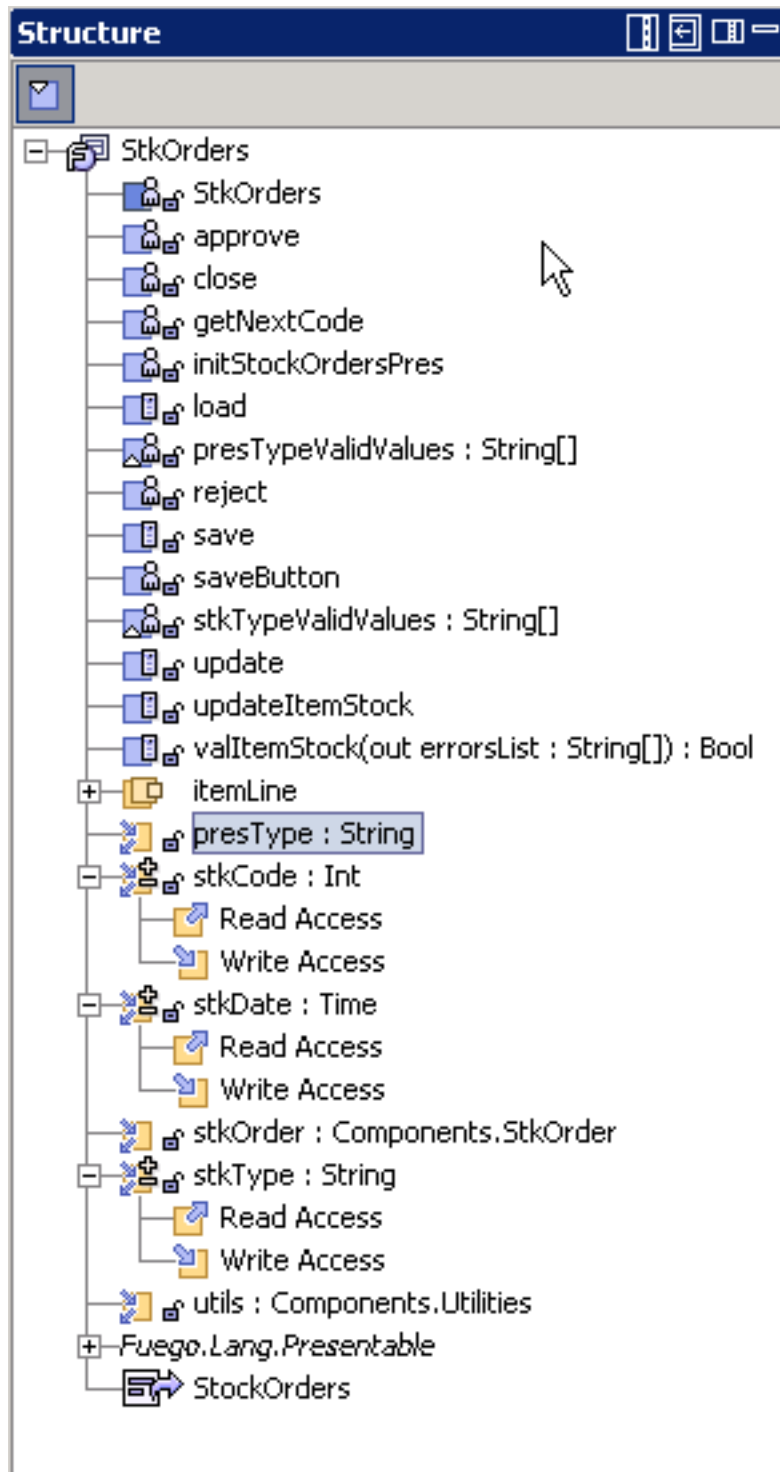
Presentation, None.

StkOrders

This example presents:

- one to many relationship implementation
- dynamic valid values with description
- inner Fuego Objects as attributes
- group attributes
- customizing the Fuego Object and Group constructor
- action buttons
- array component in presentations
- using inner Fuego Objects from the presentation

This Fuego Object was constructed to add and approve Stock Orders.



The functionality we want to provide by building this Fuego Object is:

- generate a stock order updating the number when the save is executed.
- validate the stock order items to be sure that there is enough stock when the stock order type is OUT.
- update the stock per item when the stock order is approved, adding or diminishing the quantity for each item.
- assign an special status to the order, *DISC* or *APP*, when it is rejected or approved.

Attributes

- *stkCode*: a String attribute that represents the stock order code (stkcode).
- *stkType*: a String attribute that represents the stock order type (stktype), it had to be redefined because it required a static list for valid values.
- *stkDate*: a TIME attribute that represents the stock order date (stkdate), it had to be redefined because the precision was needed only to be a date.
- *presType*: This attribute is used to indicate what type of action is going to be done through the presentation usage. It has defined an static list of valid values. According to its value, some presentation components properties are dynamically set in the method invoked when the presentation inits (see in this example: method *initStockOrdersPres*). This attribute is set before the input statement is used in a BP_Method. See BP_Methods: *newStockOrder*, *appRejectStockOrder*.
 - **NEW**: this value is set when it is going to be used as and *add*.
 - **APP**: this value is set when it is going to be approved.

- *stkOrder*: an inner Fuego Object of type *StkOrder*, which inherits behavior from the *STKORDER* SQL component.
- *utils*: by defining this attribute the access to the *lib* methods defined in the *Utilities* Fuego Object is enabled.
- *itemLine*: group attribute. The group attribute is formed by:
 - *item*: an String attribute that represents the attribute **item**.
 - *qty*: an Int attribute, that represents the attribute **qty**.
 - *stkOrdit*: an inner Fuego Object of type *StkOrdit*, which inherits behavior from the *STKORDIT* SQL component.

Methods

- *approve*: it invokes the submit method with *approved*.
- *close*: it invokes the submit method with *standby*.
- *getNextCode*: sets the attribute *stkCode* with the next code to generate.
- *initStockOrdersPres*: method used to initialize the presentation.
 - **NEW** (New Stock Order)
 - dynamically disables buttons: *approve* and *reject*,
 - **APP** (Approval)
 - dynamically disables buttons: *save*,

- *reject*: it invokes the submit method with *rejected*.
- *save*: updates the next stock code for the type and stores the *stkorder* and *stkordit* for each item.
- *saveButton*: invokes the *save* and the *submit* method.
- *update*: updates the Stock Order in the database as when it is approved some changes can be done to it, as delete an item row.
- *updateItemStock*: updates the stock table for each item in the array component.
- *valItemStock*: validates that the stock will be updated correctly, basically when it is an OUT type stock order, it verifies that there is enough stock to diminish.
- *showItemsHelp*: method used as dynamic method with description for the *item* attribute.

Presentation

The presentation defined is named **StkOrders**.

Components

- *stktype*: references the attribute *stkType*.
- *stkcode*: references the attribute *stkcode*. It is not editable.
- *stkdate*: references the attribute *stkDate*.
- *itemline*: array component, references to *itemLine* group attribute of the Fuego Object.
 - *item*: references the attribute *itemLine[].item*.

- *qty*: references the attribute *itemLine[].qty*.

Buttons

- *save*: action button.
 - method invoked: **saveButton**, it stores the stock order into the database invoking the method **save** and invokes the **submit** method to exit the presentation.
- *cancel*: action button.
 - method invoked: **close**, it invokes the **submit** method with *standby*.
- *approve*: action button.
 - method invoked: **approve**, it invokes the **submit** method with *approved*.
- *reject*: action button.
 - method invoked: **reject**, it invokes the **submit** method with *rejected*.

Execution

Stock Order type IN

New Order

FUEGO Work Portal Welcome, fuego Search - Options - Help - Logout

Stock Orders

Stock Order Type & Code 12

Date

	Item	Quantity
1	<input type="text" value="Item1010"/>	<input type="text" value="10"/>
2	<input type="text" value="Item1030"/>	<input type="text" value="40"/>

FuegoBPM™ - Work Portal

FUEGO Work Portal Welcome, fuego Search - Options - Help - Logout

Stock order generated:

IN12

FuegoBPM™ - Work Portal

Order Approval

FUEGO Work Portal Welcome, fuego Search - Options - Help - Logout

Stock Orders

Stock Order Type & Code 12

Date

	Item	Quantity
1	<input type="text" value="Item1010"/>	<input type="text" value="10"/>
2	<input type="text" value="Item1030"/>	<input type="text" value="40"/>

FuegoBPM™ - Work Portal

Stock Order type OUT

New Order

The screenshot shows the 'FUEGO Work Portal' interface. The header bar includes the logo, 'Welcome, fuego', and navigation links: 'Search - Options - Help - Logout'. The main content area is titled 'Stock Orders'. It contains a form with the following fields: 'Stock Order Type & Code' with a dropdown menu set to 'OUT' and a value of '15'; 'Date' with a text input '2/14/2005' and a calendar icon. Below these is a table with two columns: 'Item' and 'Quantity'. The table has one row with '1' in the first column, 'Item1020' in the second, and '10000' in the third. Above the table is a toolbar with icons for adding, editing, deleting, and saving. At the bottom of the form are 'Save' and 'Cancel' buttons. The footer bar reads 'FuegoBPM™ - Work Portal'.

	Item	Quantity
1	Item1020	10000

The screenshot shows the 'FUEGO Work Portal' interface. The header bar is the same as the previous screenshot. The main content area displays the message 'Stock order generated:' followed by 'OUT15'. Below this is an 'OK' button. The footer bar reads 'FuegoBPM™ - Work Portal'.

Order Approval

The screenshot shows the 'FUEGO Work Portal' interface. The main content area is titled 'Stock Orders'. It contains a form with the following fields: 'Stock Order Type & Code' with a dropdown menu set to 'OUT' and a value of '15'; 'Date' with a text input '2/14/2005' and a calendar icon. Below these is a table with two columns: 'Item' and 'Quantity'. The table has one row with '1' in the first column, 'Item1020' in the second, and '10000' in the third. Above the table is a toolbar with icons for adding, deleting, and other actions. At the bottom of the form are three buttons: 'Cancel', 'Approve', and 'Reject'.

	Item	Quantity
1	Item1020	10000

The screenshot shows the 'FUEGO Work Portal' interface with an error message. The message text is 'Not enough Stock for:'. Below this, there is a code block showing '[Item: 1020, Stock: 2310]'. At the bottom of the message area is an 'OK' button.

```
[Item: 1020, Stock: 2310]
```

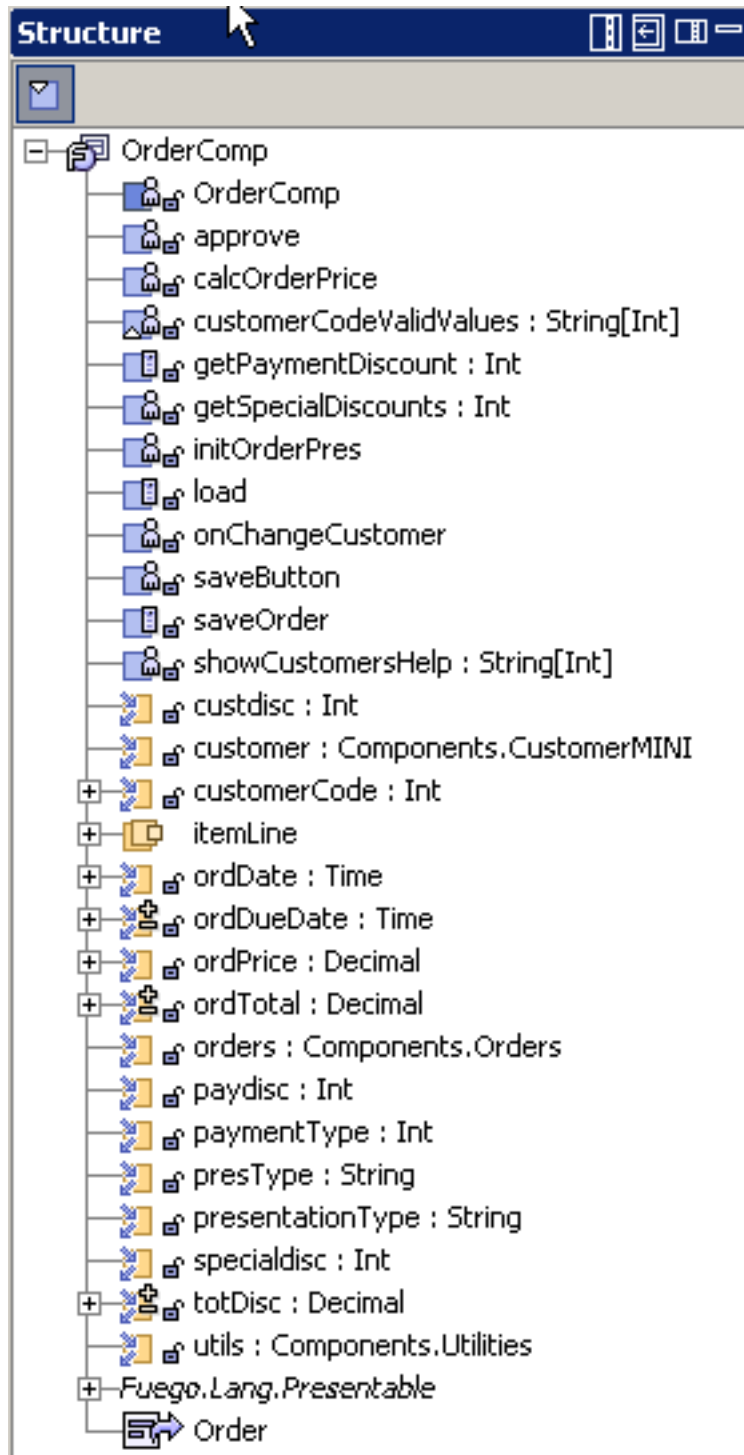
OrderComp

This example presents:

- one to many relationship implementation

- dynamic valid values with description
- inner Fuego Objects as attributes
- group attributes
- customizing the Fuego Object and Group constructor
- action buttons
- array component in presentations
- using inner Fuego Objects from the presentation

This Fuego Object was constructed to add and approve Orders.



Bussiness Object Order status:

- *PEN*: Pending, before the order is billed.
- *BLL*: Billed, once it has been billed.
- *SHP*: Shipped, once it has been shipped.
- *REJ* : Rejected, if it has been Commercially rejected or because a Credit rejection. (orders approvement logs are managed in the process, logging the information to the *ORDAPPLOGS* table).

Attributes

- *orders*: an inner Fuego Object of type Orders, which inherits behavior from the *ORDER* SQL component.
- *orderDate*: a TIME attribute that represents the stock order date (orddate), it had to be redefined because the precision was needed only to be a date.
- *orderDueDate*: a TIME attribute that represents the stock order date (ordduedate), it had to be redefined because the precision was needed only to be a date and its value is equal to the order date plus thirty days, *orderDate + 30d*.
- *ordPrice*: this attribute has the read method overwritten, to calculate its value based on the order attributes, summarizing the order items total amount.
- *ordTotal*: this attribute has the read method overwritten, to calculate its value based on the order attributes. Its value results from adding the order price minus the discounts (customer, payment and special discounts).
- *customer*: inner Fuego Object of type CustomerMINI, to load the basic customer information.
- customerCode

- *custdisc* , *paydisc*, *specialdisc*: percentage of discount to apply to order. This values are loaded in the on change of the customer, once the customer has been selected.
- *totdisc*: calculated by redefining its read access method, as the sum of the discounts of the order.
- *presType*: This attribute is used to indicate what type of action is going to be done through the presentation usage. It has defined an static list of valid values. According to its value, some presentation components properties are dynamically set in the method invoked when the presentation inits (see in this example: method *initOrderPres*).
- **NEW**: this value is set when it is going to be used as and *add*.
- **APP**: this value is set when it is going to be approved.
- **REV**: this value is set when it is going to be only reviewed.
- *utils*: by defining this attribute the access to the *lib* methods defined in the *Utilities* Fuego Object is enabled.
- *itemLine*: group attribute. The group attribute is formed by:
 - *item*: an String attribute that represents the attribute **item**.
 - *itqty*: an Int attribute, that represents the attribute **qty**.
 - *itemTotal*:
 - *orderit*: an inner Fuego Object of type *OrderIt*, which inherits behavior from the *ORDIT* SQL component.

Methods

- *approve*: it invokes the submit method with *approved*.
 - *initOrderPres*: method used to initialize the presentation.
 - **NEW** (New Order)
 - dynamically disables buttons: *approve* and *close*,
 - **APP** (Approve Order)
 - dynamically disables buttons: *save* and *close*,
 - **REV** (Review Order)
 - dynamically disables buttons: *save*, *approve* and *cancel*.
 - *saveOrder*: updates the next stock code for the type and stores the *stkorder* and *stkordit* for each item.
 - *saveButton*: invokes the *save* and the *submit* method.
- order, it verifies that there is enough stock to diminish.
- *showCustomersHelp*: method used as dynamic method with description for the *customer* attribute.
 - *onChangeCustomer*: once the customer is selected, the discounts and attributes which values depend on the customer are loaded.
 - *load*: it loads the order if the presentation type is not *NEW*.

- *getSpecialDiscounts*: returns a discount percentage according to the day of the week of the order date.
- *getPaymentDiscount*: given a payment code it returns its discount percentage.
- *calcOrdPrice*: calculate the order price

```
orders.ordprice = 0
for each item in itemLine
do
    orders.ordprice = orders.ordprice + item.itemTotal
end
orders.ocustdisc = orders.ordprice * custdisc / 100
orders.opaydisc = orders.ordprice * paydisc / 100
orders.ospecdisc = orders.ordprice * specialdisc / 100

totDisc = orders.ocustdisc + orders.opaydisc + orders.ospecdisc
orders.ordextchg = 0
orders.orderTotal = orders.ordprice - totDisc
```

- Item methods
 - *getItemPrice*: once the item has been selected this method is invoked in the on change to find and show the item price.
 - *onChangeQuantity*: once the quantity for the item has been input, the total for the item row and the recalculation of the order price is done.
 - *showItemsHelp*: method used as dynamic method with description for the *customer* attribute.

Presentation

The presentation defined is named **Order**.

Buttons

- *save*: action button. Displays *Save Order*.
 - method invoked: **saveButton**, it stores the stock order into the database invoking the method **save** and invokes the **submit** method to exit the presentation.
- *cancel*: cancel action button. Displays *Cancel*.
- *approve*: action button. Displays *Approve*.
 - method invoked: **approve**, it invokes the **submit** method with *approved*.
- *close*: submit action button. Display *Close*

Execution

New Order

FUEGO Work Portal
Welcome, fuego
Search - Options - Help - Logout

Order

Order Number

Order Date
Order Due Date 3/17/2005

Order Description

Customer
Payment Type 3

Customer Discount % 30
Payment Discount % 10
Special Discounts % 0

	Item	Quantity	Price	Total Item
1	<input type="text" value="Item1010"/>	<input type="text" value="3"/>	30.0000	90.00000000
2	<input type="text" value="Item1020"/>	<input type="text" value="2"/>	20.0000	40.00000000

Total Discounts 130.0000
Total Discounts 39.0000
Total Discounts 13.0000
Total Discounts 0.0000
Total Discounts 52.0000

Order Total 78.0000

FuegoBPM™ - Work Portal

Order Review

FUEGO Work Portal
Welcome, fuego
Search - Options - Help - Logout

Order

Order Number

Order Date
Order Due Date
3/17/2005

Order Description

Customer
Payment Type

Customer Discount %
Payment Discount %
Special Discounts %

Item	Quantity	Price	Total Item
1 <input type="text" value="Item1010"/>	<input type="text" value="3"/>	30.0000	90.00000000
2 <input type="text" value="Item1020"/>	<input type="text" value="2"/>	20.0000	40.00000000

Total Discounts
Total Discounts
Total Discounts
Total Discounts
Total Discounts

130.0000
39.0000
13.0000
0.0000
52.0000

Order Total
78.0000

Close

Order Approval

Order

Order Number: 55

Order Date: 2/15/2005 Order Due Date: 3/17/2005

Order Description: To deliver ASAP

Customer: Peter Drayfus Payment Type:

Customer Discount % Payment Discount % Special Discounts %

Item	Quantity	Price	Total Item
1 Item1010	3	30.0000	90.00000000
2 Item1020	2	20.0000	40.00000000

Total Discounts Total Discounts Total Discounts Total Discounts Total Discounts

130.0000 39.0000 13.0000 0.0000 52.0000

Order Total: 78.0000

Cancel Approve

FuegoBPM™ - Work Portal

Fuego Object Methods - Action Methods

Action methods are invoked from buttons define as *action* type. In this case, you need to specify a method to be invoked to perform the desired action.

How do you get it?

An *Action* can be anything you want, such as an approval or rejection.

Examples of them are included in our *Customer* Fuego Object. The presentation **CustomerPres** is used in the *Approve Customer*, *Assign Credit Limit To Customer* and *Reject Customer*. Through this presentation the analysts visualize the customer information that help them to decide whether or not to approve the customer.

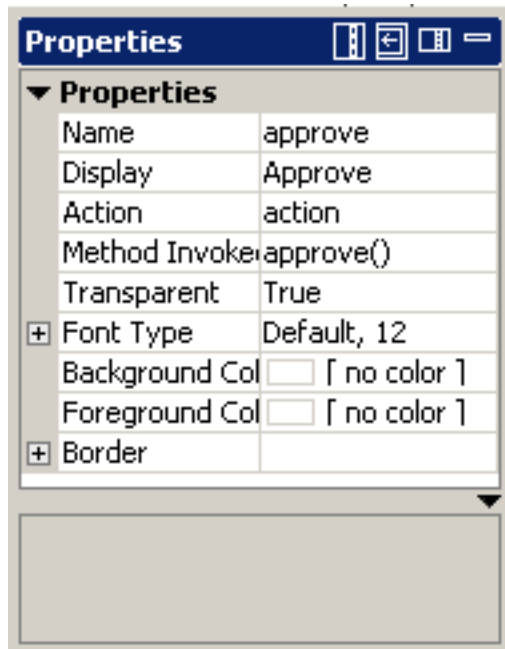
To do so, the presentation has three buttons,

- *Approve*, if this button is selected the customer will be approved and moved to:
 - the next activity *Assign Credit Limit To Customer* or *End* and corresponding the databases will be updated with the required information, if it was being executed in one of the approval activities, or
 - back to activity from where it came, in case it is being executed in the *Reject Customer* activity, and the analyst wants to send it back to the approval flow.
- *Reject*, if this button is selected the customer will be rejected and sent:
 - to the *Reject Customer* activity, if it is being executed in one of the approval activities, or
 - to the *End* activity and is removed from the database, if it was being executed from the *Reject Customer* activity.
- *Cancel*, if this button is selected no processing is done.

Buttons definition

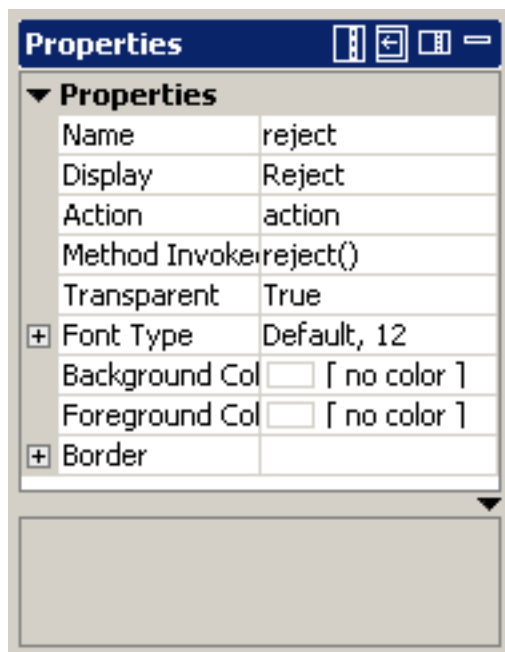
Both of them have been defined as *action* buttons invoking a method. If you remember, the only methods FuegoBPM Studio shows when you are defining an action method, are those with void returned value.

Approve button properties:



It invokes a method name *approve()*.

Reject button properties:



It invokes a method name *reject()*.

Action methods code

Both methods invoke the submit Fuego Object standard method with different value for its argument.

Remember that the *submit* method has only one argument string type, that is the value returned to the calling method. The BP_Method from where the presentation is executed checks the returned value for *approved*, *rejected*, or *canceled*. With that information, the corresponding database is updated.

approve() method code:

```
submit("approved")
```

reject() method code:

```
submit("rejected")
```

Designing Using Fuego Objects

Invoking a Fuego Object in a Component Task

In the example project, a customer received in XML format is processed and added into the database and into the approval process. In order to do this, the main task of the *Store Customer in DB* activity is defined as **Component Method**, as shown below:

Main task - Activity Store Customer in DB

Store Customer in DB

Implementation type

Component

Component method

Component name:

☐ Use instance variable

Component name:

Components.CustomerFO

Browse

Component member

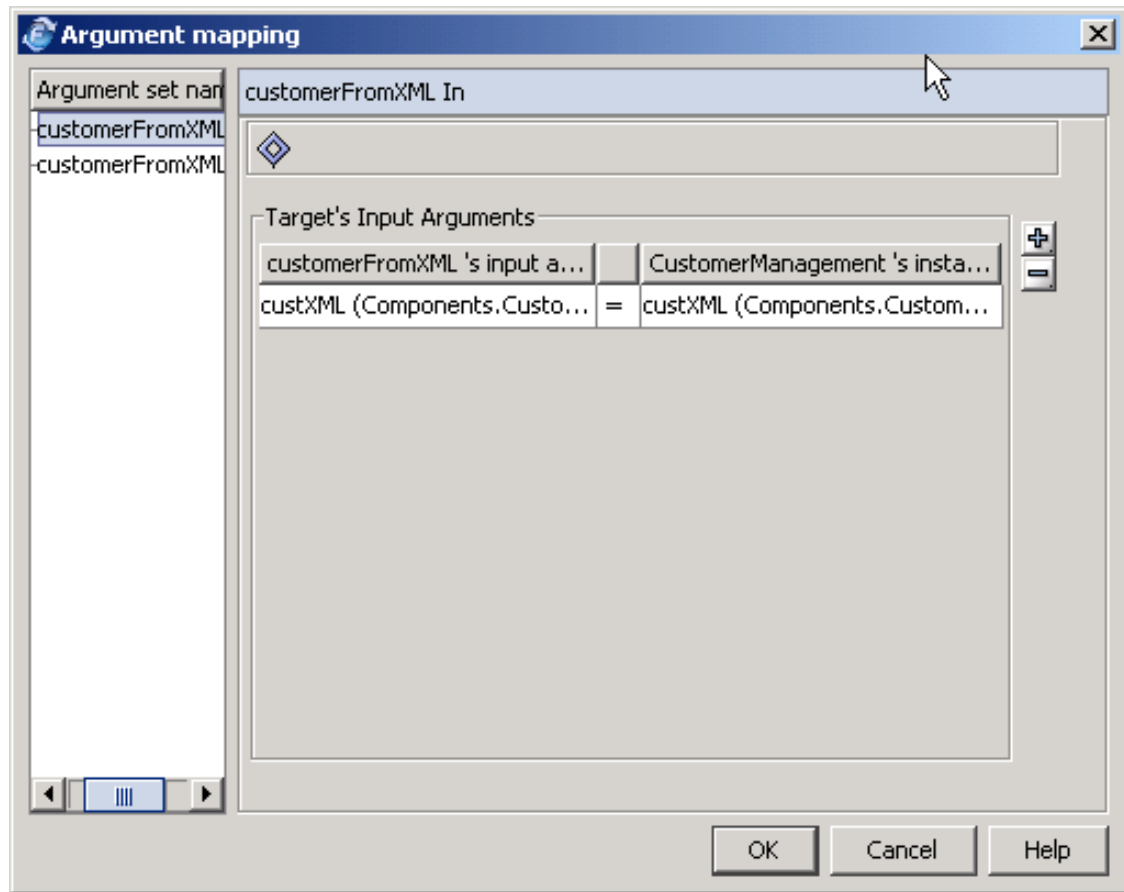
Method Name

customerFromXML(in custXML : Components.CustomerXML.Customer)

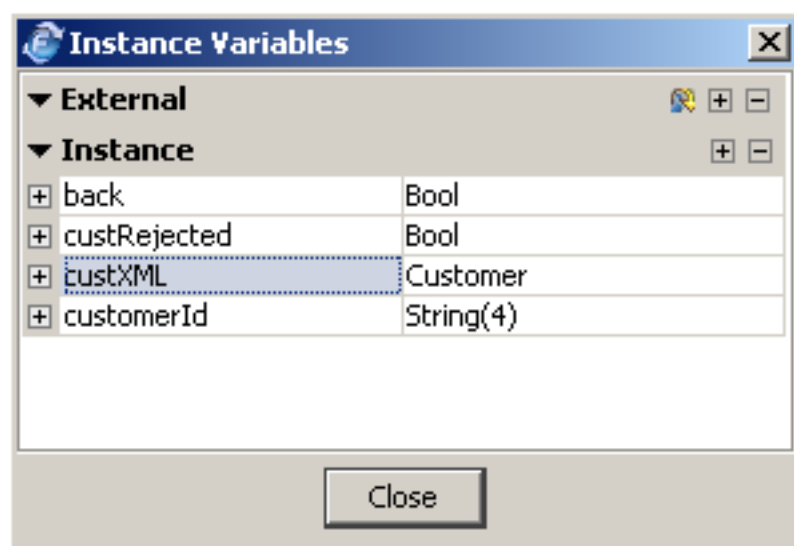
Argument mapping

OK Cancel Help

The *argument mapping* of this activity is defined for *IN* mapping relating the **method argument** to the **instance variable** that contains the customer in XML format, as shown below:



where **instance variables** are:



Using a FO from a BP_method

Fuego Object input

For example, the *pricinglist* method used by the Global activity *Pricing List*, inputs the *PricingList* Fuego Object through its presentation.

```
pl as PricingList
pl = PricingList()

input pl using selectedPresentation = "PricingPres"
```

Fuego Object input plus store

The method *addPaymentTypes* invoked in the Global activity *Payment Types*, inputs the *PayType* Fuego Object through its presentation, and depending on the selected button of the presentation, executes the *store* method to save it to the database.

```
retButton as String
payType as PayType
payType = PayType()

input payType using selectedPresentation = "NewPayType"
    returning retButton = selection

if retButton == "submit" then
    payType.store()
end
```

Fuego Object input and actions according to selected button in the presentation

The method *appCustomer* invoked in the Global activity *Payment Types*, inputs the *Customer* Fuego Object through its presentation. Depending on the selected button of the presentation, executes the *store* method

to save it to the database and sends it to the next approval activity evaluated by credit, which determines whether the instance goes to the reject activity or cancels the execution. This presentation executes in different ways according to the activity from where it is invoked. That is why the creation of the Fuego Object auxiliary variable is done with the *presType* attribute set to *COM*. In this case some buttons and fields of the presentation are disabled.

```
aux as Components.Customer
aux = Components.Customer(presType:"COM",
                           custcode:customerId)

input aux using selectedPresentation = "CustomerPres"
        returning retButton = selection

if retButton == "approved" then
    aux.store()
    custRejected = false
else
    if retButton == "rejected" then
        custRejected = true
    else
        action = CANCEL
    end
end
end
```

Creating a Fuego Object from an XML Component

custXML is an XML component introspected into the Project Catalog. It defines a customer *CustomerXML*. custFO is a *CustomerFO* Fuego Object inheriting behavior from the SQL Component *CUSTOMER*.

```
custFO = CustomerFO()

custFO.custcode = custXML.customerCode
custFO.custname = custXML.customerName
custFO.custtype = custXML.custType
custFO.mail = custXML.mail
custFO.billadd = custXML.billAddress
```

```
custFO.shippadd = custXML.shipAddress  
custFO.codpay = Int(custXML.codPay)  
custFO.custdisc = Int(custXML.custDisc)  
  
store custFO
```

Implementing Business Objects Using external components

COM Components

About COM Components

COM components are software programs that use the Microsoft Component Object Model (COM) technology. COM is a set of standards that specify several constraints that each component should follow in order to comply with the standard (such as which types are valid for a method's argument, how to manage memory, etc.)

COM objects can be created in several different ways and with different tools (for example you could build a COM component using Visual Basic .Net or C++). Several applications expose functionality as COM objects - this includes most of Microsoft's applications (Office, Internet Explorer, etc.) and several other third-party applications (SAP, Adobe Acrobat, etc.)

Integrating with COM Components

If you plan on calling an external application that exposes itself using COM, from your process or Fuego Object methods, make sure you read the COM documentation provided with that application (or contact the software vendor for further information). For example, if you wish to call a Microsoft product, such as Microsoft Excel, you will need to read the COM information that can be found on the Microsoft

Developers Network Internet site.

About FuegoBPM COM Bridge

FuegoBPM COM Bridge is a Windows application that acts as a "bridge" between FuegoBPM applications and COM. Fuego supplies this application to provide all the necessary services to introspect and use COM components. It is in charge of asking COM invocations, type conversions and marshalling of argument across the net.

FuegoBPM COM Bridge is packaged as a separate application to allow you to:

- run your processes in a robust environment by effectively isolating the Server from misbehaving components.
- run the Server in a different host environment from the one used to run COM components (e.g., run your server in a Solaris box and your COM components in one or several Windows servers or even use one COM Bridge with several servers).

Because COM components do not necessarily reside on the same machine as the Server, COM Bridge should be installed on the machine where the component resides. However, this is not absolutely necessary for all components because DCOM components, if properly configured, can be located in a different machine from the one that is running COM Bridge.

There are no special considerations when the process that uses COM Components is deployed in a J2EE/UNIX environment. A FuegoBPM COM Bridge has to be running on a Windows box accessible through the LAN and it will work as any other service (like a data base, for example).

For further information about COM/DCOM, refer to Microsoft's documentation.

Installing FuegoBPM COM Bridge

FuegoBPM COM Bridge comes in two flavors:

- **combridge.exe**: Is a standalone application, which is intended for the development stage of a project. It should be installed on the Process Developer workstation. It will start up whenever a user logs-on.
- **combsvc.exe**: Is the service version of FuegoBPM COM Bridge. It is intended for production environments, where no GUI interaction is required with the COM components or applications that are being automated. After installation, it will start whenever the machine starts.

The development version of FuegoBPM COM Bridge is installed when the FuegoBPM Studio is installed. However, it can be manually installed by running:

```
combridge -install
```

Additionally you could start the combridge in debug mode so an output file is generated. To do that, follow these instructions:

1. Open a terminal, go to "fuego's installation directory"/bin, and stop the COM Bridge by executing:

```
C:\\fuego\\bin> combridge /stop
```

1. Execute COM Bridge with the following options:

```
C:\\fuego\\bin> combridge /debug /log C:\\combridge.log
```

This starts FuegoBPM COM bridge with full debug level and will log its execution to "C:\\combridge.log".

1. Run the problematic CIL again. All operations will be logged to the log file, including in and out arguments.

FuegoBPM COM Bridge Service

Analogously, the service version can be installed by running:

```
combsvc -install
```

This installs COM Bridge as a Windows service; it can be started from the Services Control Panel or from the command line by running:

```
combsvc -start
```

If you want to stop it, run:

```
combsvc -stop
```

And to remove it, run:

```
combsvc -remove
```

For further information, see section: Command Line Options.

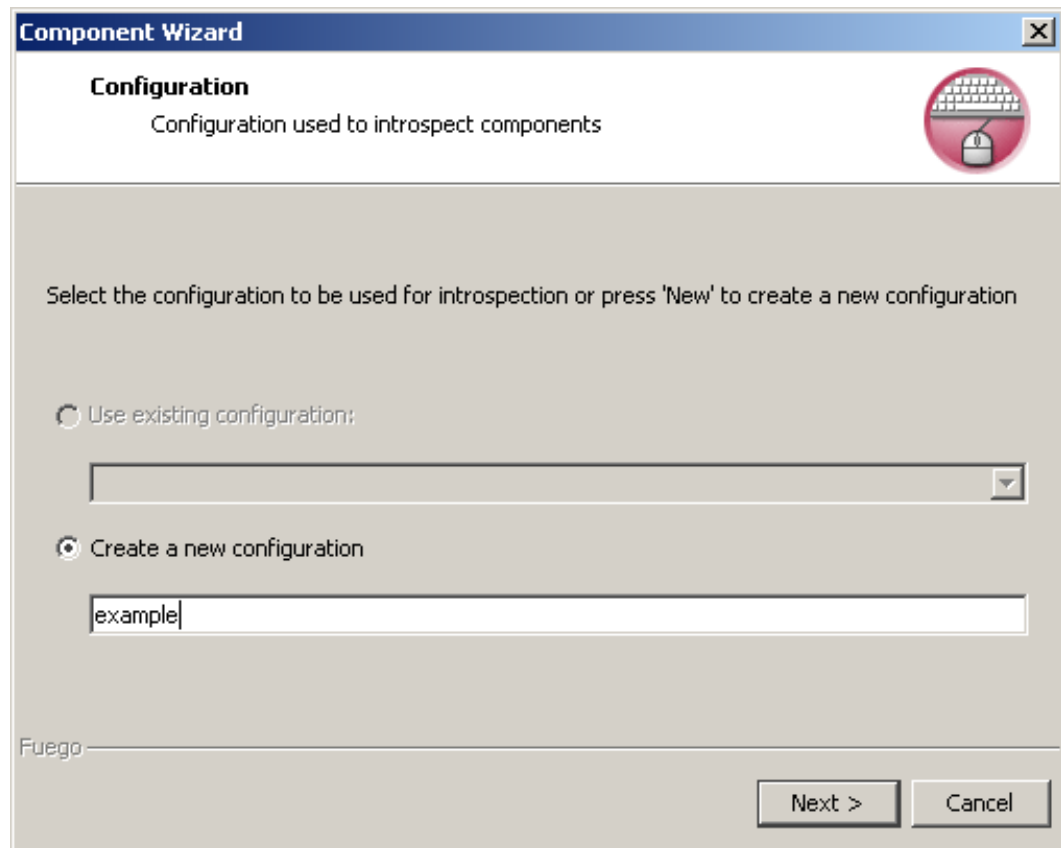
Using COM Components

Before using COM components from processes or Fuego Objects, the components should be cataloged into the project's catalog. When you catalog a component, you are gathering all the necessary information that FuegoBPM Studio needs in order to call and execute it at runtime. This information is also used by FuegoBPM Studio to detect potential errors at compile time, which dramatically reduces the time spent in fixing errors in a process.

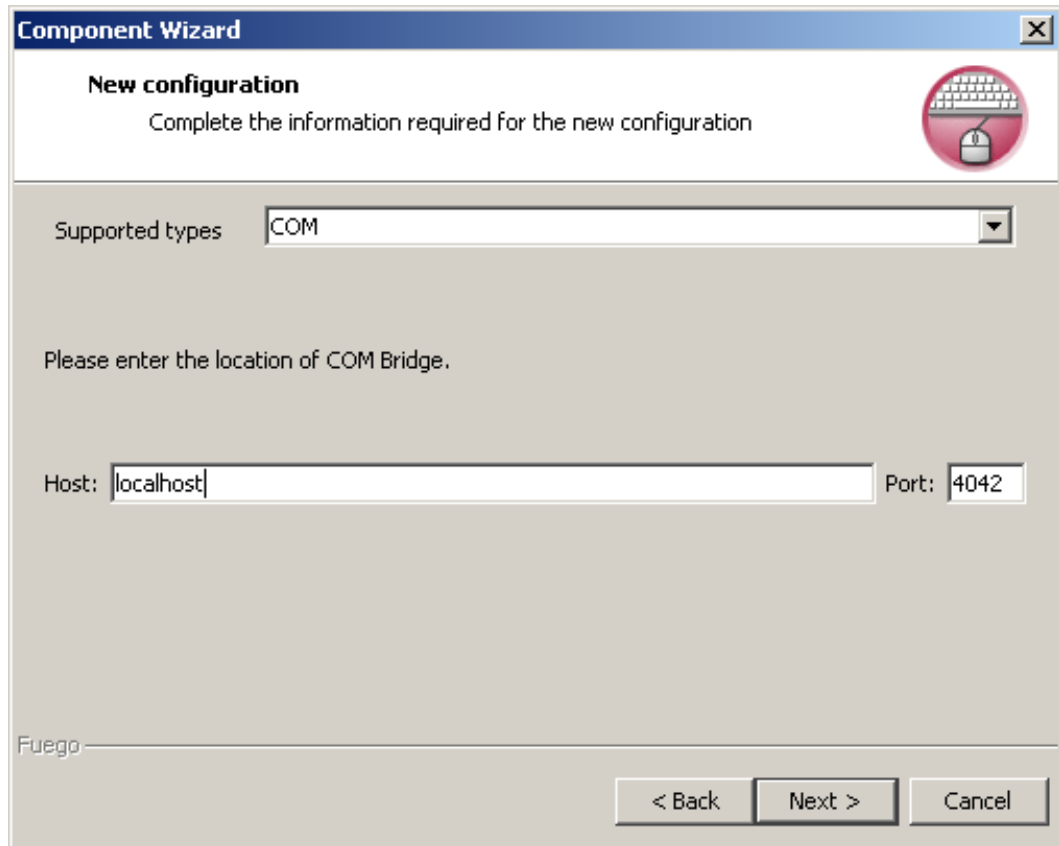
Catalog the COM component in the Project Catalog

To catalog the component in the Project Catalog, a wizard is provided.

1. Right-click on the module where you want to add the COM component and select **Catalogue Component** and then **COM** from the shortcut menu.
2. Select an existing **Configuration** if you are reusing one or check the radio button *Create a new configuration* and type its name.

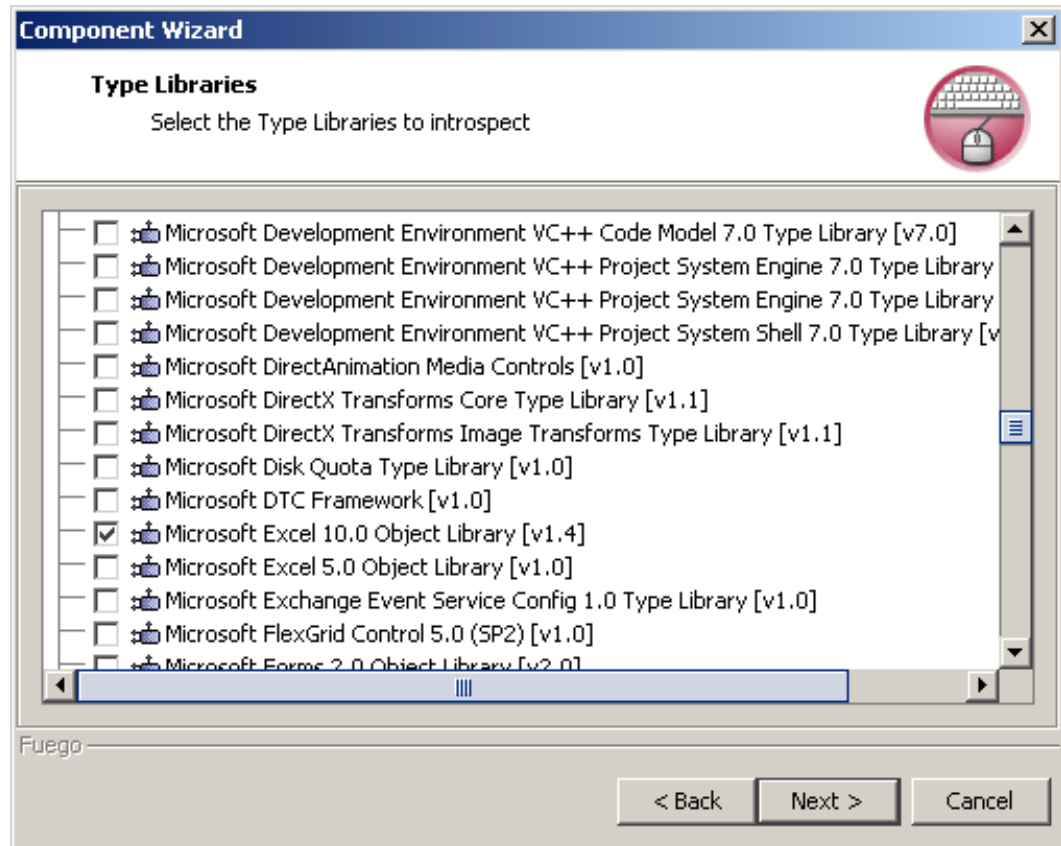


3. Next, indicate the host and port where the COM Bridge has been installed. The default port is '4042'.

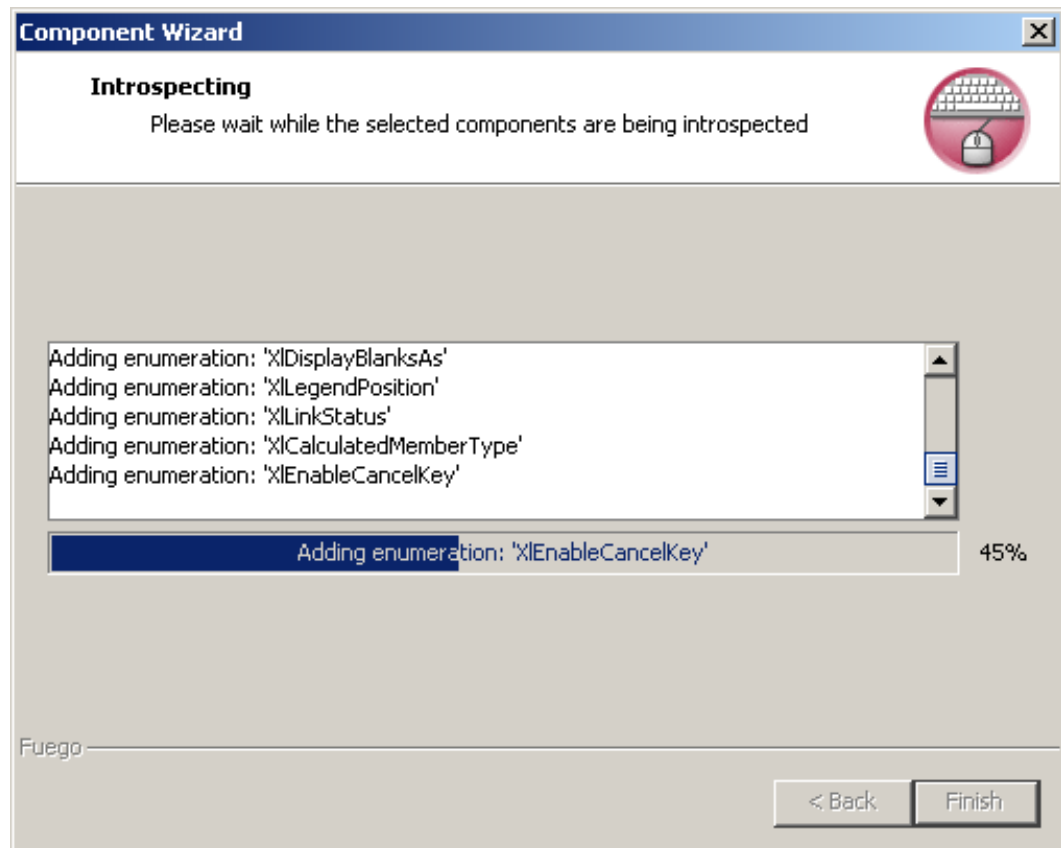


The screenshot shows a Windows-style dialog box titled "Component Wizard". The main heading is "New configuration" with a sub-instruction "Complete the information required for the new configuration". A red circular icon with a keyboard and mouse is in the top right. Below the heading is a "Supported types" label and a dropdown menu currently showing "COM". A text prompt "Please enter the location of COM Bridge." is centered. Below this are two input fields: "Host:" with "localhost" and "Port:" with "4042". At the bottom left is the "Fuego" logo. At the bottom right are three buttons: "< Back", "Next >", and "Cancel".

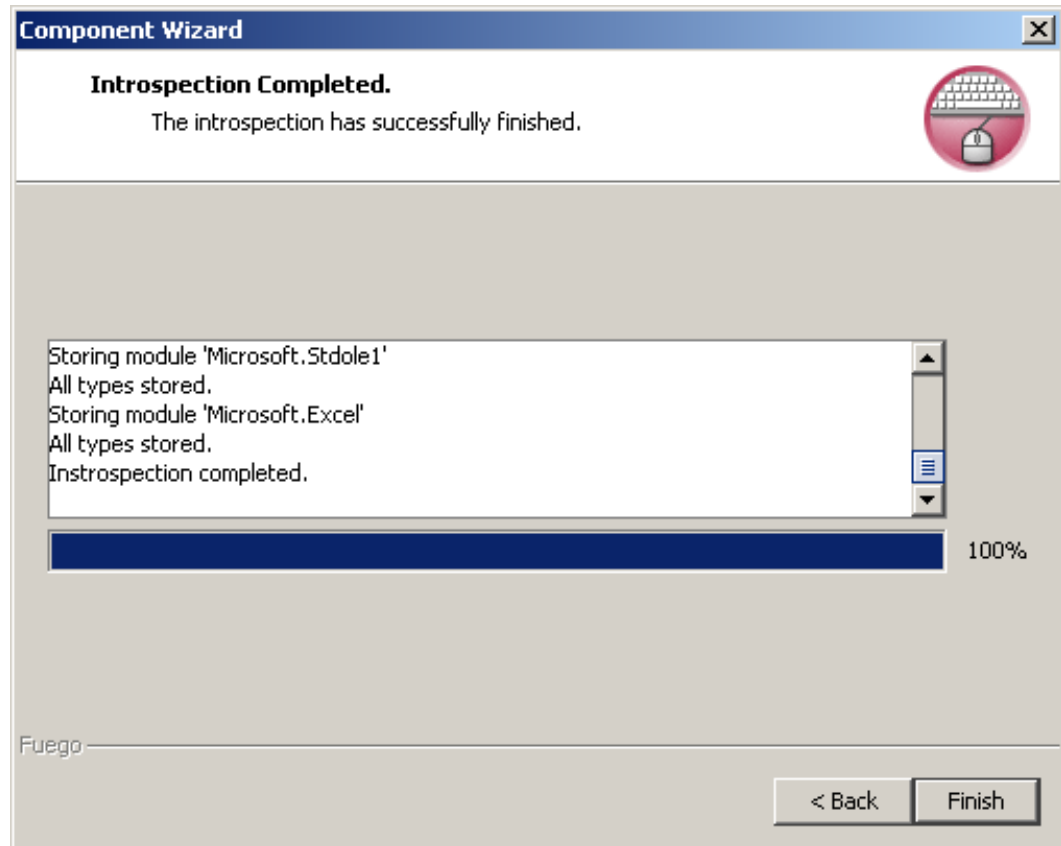
4. The following enables you to choose which type libraries you want to introspect. Every available type library on the machine where COM Bridge is installed is displayed in the wizard dialog. Scroll down the list and select the ones you want to catalog.



5. Click **Next**, the components introspection will begin.



6. Click **Finish**.



Example using an Excel file

To run this example you have to catalog the Microsoft Excel Object Library under a module name MSEXCEL.

The autocommented BP_Method shows how to use Excel from Fuego BPMN. The Excel file example can be found in the Orderfulfillment project under the *sample* directory of your FuegoBPM Studio installation.

```
do
//Open the sheet
open MSEXCEL.Application.workbooks
    using "C:\\tmp\\invoice.xls"

//Set it visible (for debugging purposes)
//MyExcel.Application.visible = true
```



```

//Get the active sheet cells
cells = Worksheet(
    MSEXCEL.Application.activeWorkbook.activeSheet).cells

//Ask for some client data, this could be taken
//from a database
input "Name:" name,
    "Address:" address,
    "State: " state,
    "Zip: " zip,
    "Phone:" phone,
    "Order No.:" orderNo
using title = "Enter product"

//Fill the sheet, the getItem method, takes two args
// (in ths case row and column), and returns a
// MSEXCEL.Range object that matchs the criteria
// for selection
// Fill Name

getItem cells using 4, 3 returning temp
cell = MSEXCEL.Range(temp)
cell.value = name

//Fill Date
getItem cells using 4, 9 returning temp
cell = MSEXCEL.Range(temp)
cell.value = 'now'

//Fill Address
getItem cells using 5, 3 returning temp
cell = MSEXCEL.Range(temp)
cell.value = address

//Fill State
getItem cells using 5, 5 returning temp
cell = MSEXCEL.Range(temp)
cell.value = state

//Fill Zip
getItem cells using 5, 7 returning temp
cell = MSEXCEL.Range(temp)
cell.value = zip

//Fill Order No.
getItem cells using 5, 9 returning temp
cell = MSEXCEL.Range(temp)
cell.value = orderNo

//Fill Phone
getItem cells using 6, 3 returning temp

```

```

cell = MSEXCEL.Range(temp)
cell.value = phone

// Now we enter some data about products
for i in 10..21 do
    qty = 0
    product = ""
    price = 0

//We ask for it
input "Quantity:" qty,
    "Product:" product,
    "Unit Price:" price
    using title = "Enter product",
        buttons = ["Ok", "Finish" ]
    returning buttonPressed = selection

exit when buttonPressed != "Ok"

//And then we fill the sheet
//Fill Qty.
getItem cells using i, 2 returning temp
cell = MSEXCEL.Range(temp)
cell.value = qty

//Fill product
getItem cells using i, 3 returning temp
cell = MSEXCEL.Range(temp)
cell.value = product

//Fill price
getItem cells using i, 8 returning temp
cell = MSEXCEL.Range(temp)
cell.value = price

end

//We finally ask for the shipping cost
price = 0
input "Shipping cost" price
    using title = "Invoice"

//Fill Shipping
getItem cells using 23, 9 returning temp
cell = MSEXCEL.Range(temp)
cell.value = price

display "What do you want to do with this sheet?"
    using title = "Invoice finished",
    options = ["Preview", "Print"],
    default = "Preview"

```

```
        returning buttonPressed = selection

if buttonPressed == "Print" then
    //Print it
    printOut Worksheet(
        MSEXCEL.Application.activeWorkbook.activeSheet)
else
    //Preview it
    MSEXCEL.Application.visible = true
    printPreview Worksheet(
        MSEXCEL.Application.activeWorkbook.activeSheet)
end

//Mark it as saved
MSEXCEL.Application.activeWorkbook.saved = true

//Display that we're done with this (just for debugging,
// so we can take a look at the invoice we just filled
display "Done!"

//Just in case, we ask Excel to quit
quit MSEXCEL.Application

end
```

Example using Word files

To run this example you have to catalog the Microsoft Word Object Library under a module name *ComCatalogation*.

The FBL methods included in this example are part of the *WordExample* project under the *sample* directory of your FuegoBPM Studio installation. The *input.doc* Word document used in the second example is under the *WordExample* project directory, move it to "C:\tmp" to make the process work or modify the line in the FBL to the real location of the file in your installation.

Example 1: Creating a new Word file

The FBL below, shows how to create and fill with text a Word file. This FBL is the one corresponding to the activity "CreateWordDocument" of the "Word Document Creation Example" process.

```
filename = "C:\\WordTest" + id.number + ".doc"

Application.visible = false

// get ready to use Word
worddocs = Application.documents

// create a new Word document
worddoc = add(worddocs)

// find something specific in the document.
// in this case there isn't anything,
// but the rangevar variable is still needed
// for the next command which inserts the text.
rangevar = range(worddoc)

// tell Word where you want to put the sentence.
// I say after rangevar, but as you know from above,
// that isn't very specific.
insertAfter rangevar
    using text = sentence + "\\nFuego is the
        greatest software ever!"

// save as the filename you want.
// We built the name somewhat unique in a variable
saveAs worddoc
    using fileName = filename

// close the newly saved document
close worddoc

// quit
quit Application
```

Example 2: Setting values in an existing Word file

The FBL below, shows how to complete form fields in a Word file used as template, and saved as a new file. This FBL is the one corresponding to the activity "Input Word Document Example" of the analog process.

```
// Initialize variables
custName = customerName
```

```

worddocs = wordappl.documents
Application.visible = false

// Open Word file
open worddocs using
    fileName = "C:\\tmp\\input.doc" returning worddoc

// Initialize the form fields into their object
wordformfields = worddoc.formFields

// Select the form field you want to work on
item wordformfields using
    index = "Text2" returning wordformfield

// Set the value of the form field you selected
wordformfield.result = custName

// Save as a different Word file
saveAs worddoc using
    fileName = "C:\\tmp\\result.doc"

// Quit Word
quit wordappl

```

Command Line Options

Standalone Version

Command	Description
combridge -install	Installs the COM Bridge to start on every login.
combridge -remove	Uninstalls the COM Bridge.
combridge -debug	Disables asynchronous logging.
combridge -stop	Stops another COM Bridge running.
combridge -log filename	Sets the log file name (default is %TEMP%\COMBridge.log).
combridge -?	Displays COM Bridge usage dialog
combridge [option] [port]	The port is the TCP port where COM Bridge is listening to incoming calls. Defaults to port 4042.

Service Version

Command	Description
combsvc -install	Installs the COM Bridge as a service.
combsvc -remove	Uninstalls the COM Bridge as a service.
combsvc -debug [params]	Run the COM Bridge as a console application for debugging.
combsvc -start	Starts the COM Bridge service.
combsvc -stop	Stops the COM Bridge service.
combsvc -?	Displays COM Bridge service commands.

.Net Assemblies Components

About .Net Assemblies

Microsoft® .NET is a vision and set of Microsoft software technologies for connecting information, people, systems, and devices. It enables a high level of software integration through the use of XML Web services—small, discrete, building-block applications that connect to each other as well as to other, larger applications over the Internet. Microsoft Visual Studio® .NET and the Microsoft .NET Framework allow developers to develop XML Web services quickly and integrate them easily with other applications.

Assemblies are the building blocks of .NET Framework applications. An assembly is a collection of types and resources that are built to work together and form a logical unit of functionality. An assembly provides the common language runtime with the information it needs to be aware of type implementations. To the runtime, a type does not exist outside the context of an assembly.

Integrating with .Net Assemblies

If you plan on calling an external application that exposes itself using

.Net Assemblies, from your process or Fuego Object methods, make sure you read the .Net documentation provided with that application (or contact the software vendor for further information).

The .Net bridge

FuegoBPM .Net Bridge is a Windows packaged as a separate that acts as a "bridge" between FuegoBPM applications and .Net Assemblies. Fuego supplies this application to provide all the necessary services to introspect and use .Net components.

The .NET bridge runs as a standalone process. Is itself a .NET application, so it runs in the CLI (the CLI is managed automatically by Windows).

All components called by the bridge, share the same CLI than the bridge itself since they are called using System.Reflection APIs (it follows that they share the same process).

Using .Net Assemblies in FuegoBPM

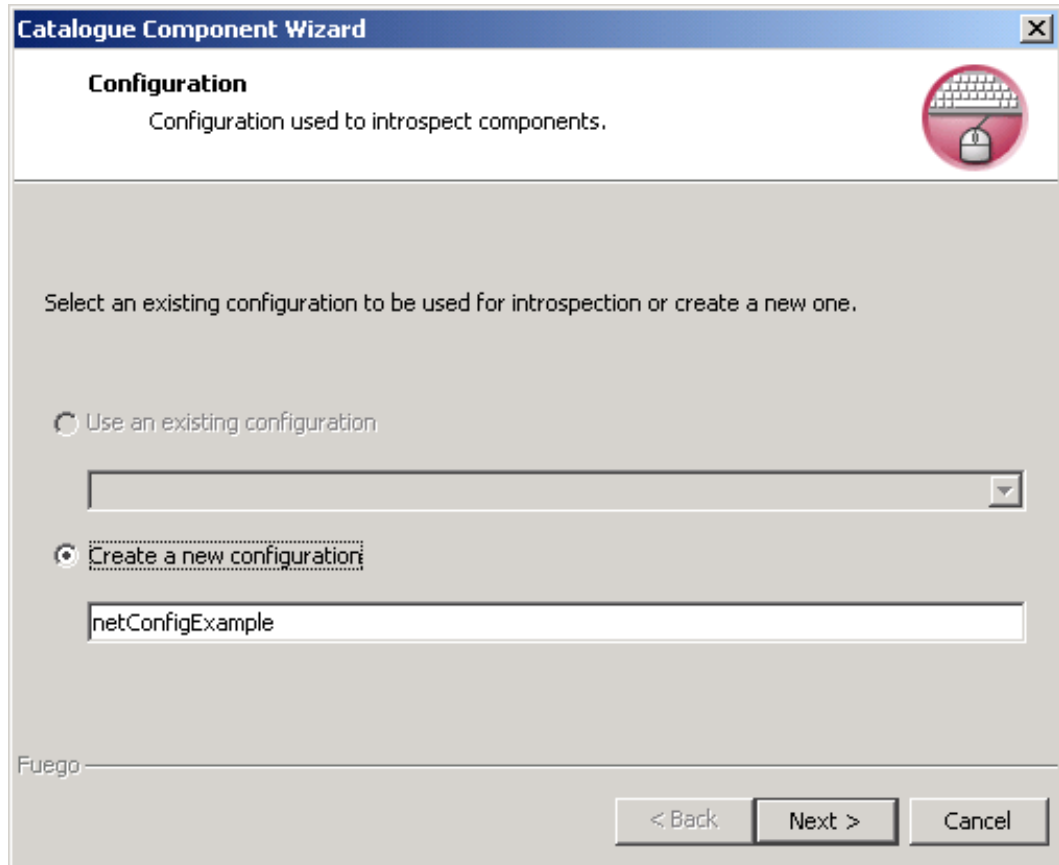
Before using .Net Assemblies from processes or Fuego Objects, they should be cataloged into the project's catalog. When you catalog a .Net assembly, you are gathering all the necessary information that FuegoBPM Studio needs in order to call and execute it at runtime. This information is also used by FuegoBPM Studio to detect potential errors at compile time, which dramatically reduces the time spent in fixing errors in a process.

Introspect the .Net Assembly into the Project Catalog

To catalog the component in the Project Catalog, a wizard is provided.

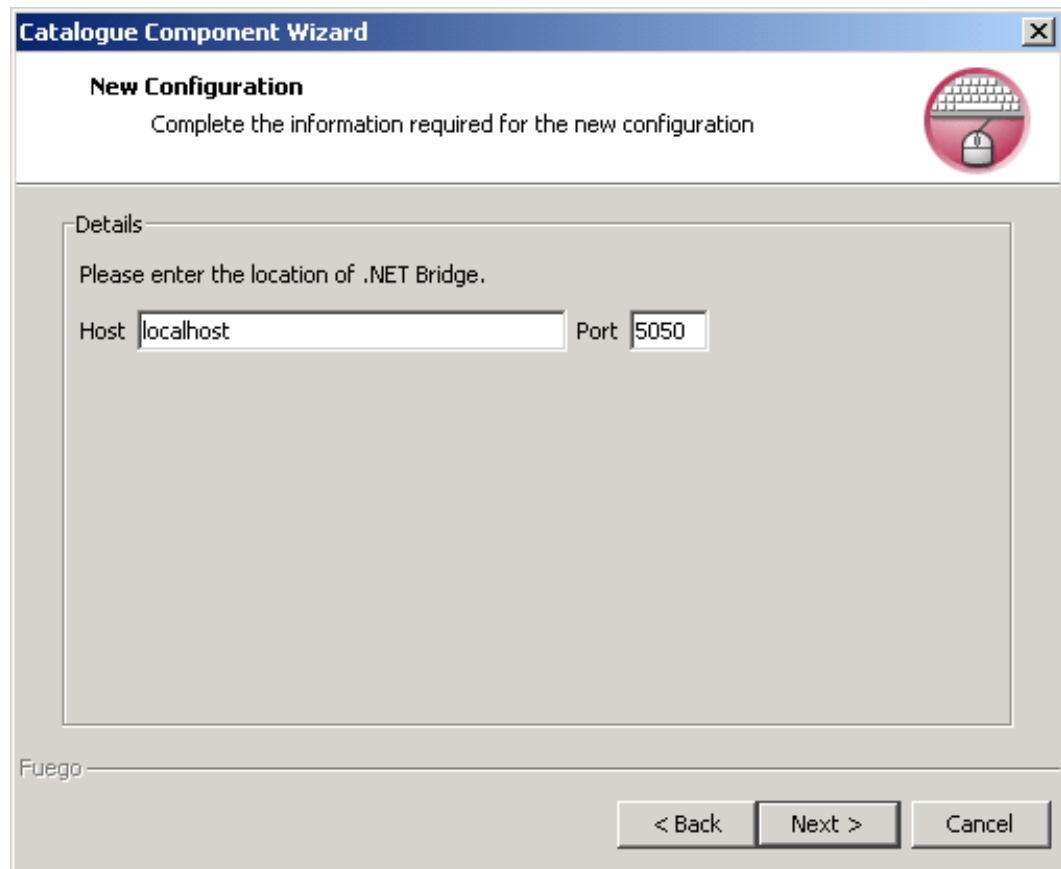
1. Right-click on the module where you want to add the .Net assembly. Select **Catalogue Component** and then **.NET** from the shortcut menu.

2. Select an existing **Configuration** if you are reusing one or check the radio button *Create a new configuration* and type its name.



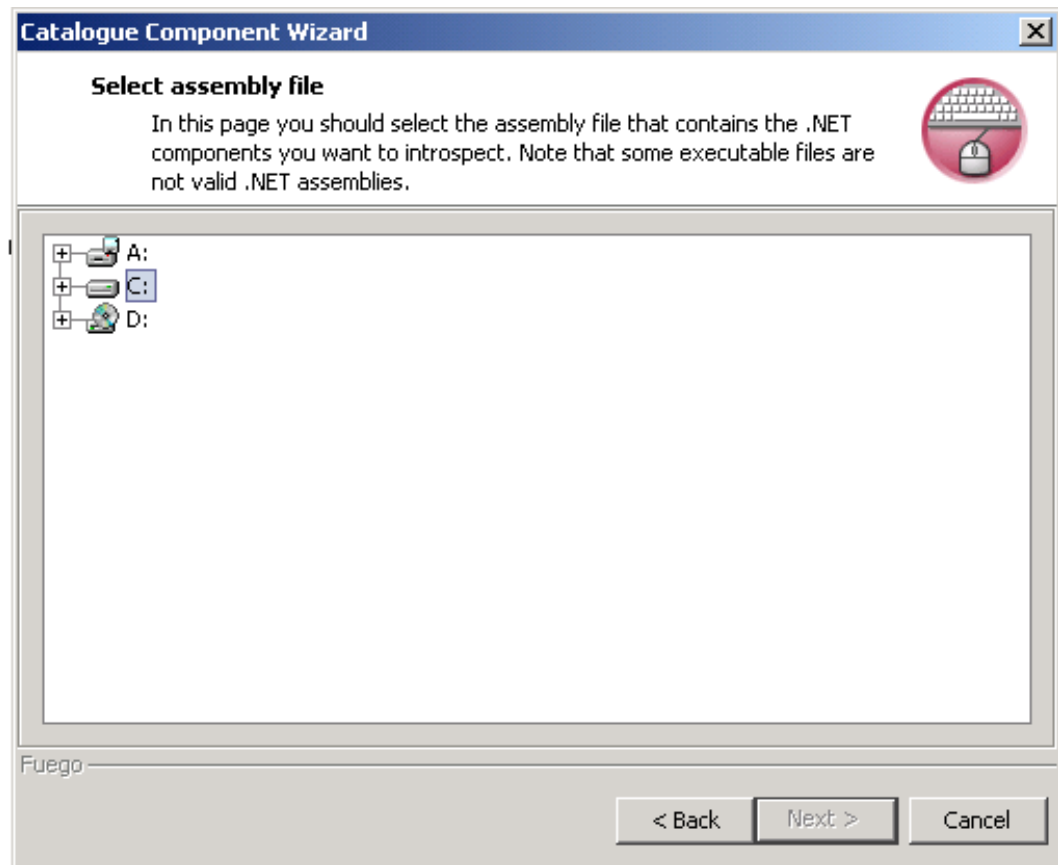
The screenshot shows a window titled "Catalogue Component Wizard" with a close button in the top right corner. The main heading is "Configuration" with a subtext "Configuration used to introspect components." and a keyboard and mouse icon. Below this, a message says "Select an existing configuration to be used for introspection or create a new one." There are two radio buttons: "Use an existing configuration" (unselected) and "Create a new configuration" (selected). Under "Use an existing configuration" is an empty dropdown menu. Under "Create a new configuration" is a text box containing "netConfigExample". At the bottom left is the "Fuego" logo. At the bottom right are three buttons: "< Back", "Next >", and "Cancel".

3. Next, indicate the host and port where the .Net Bridge has been installed. The default port is 5050.

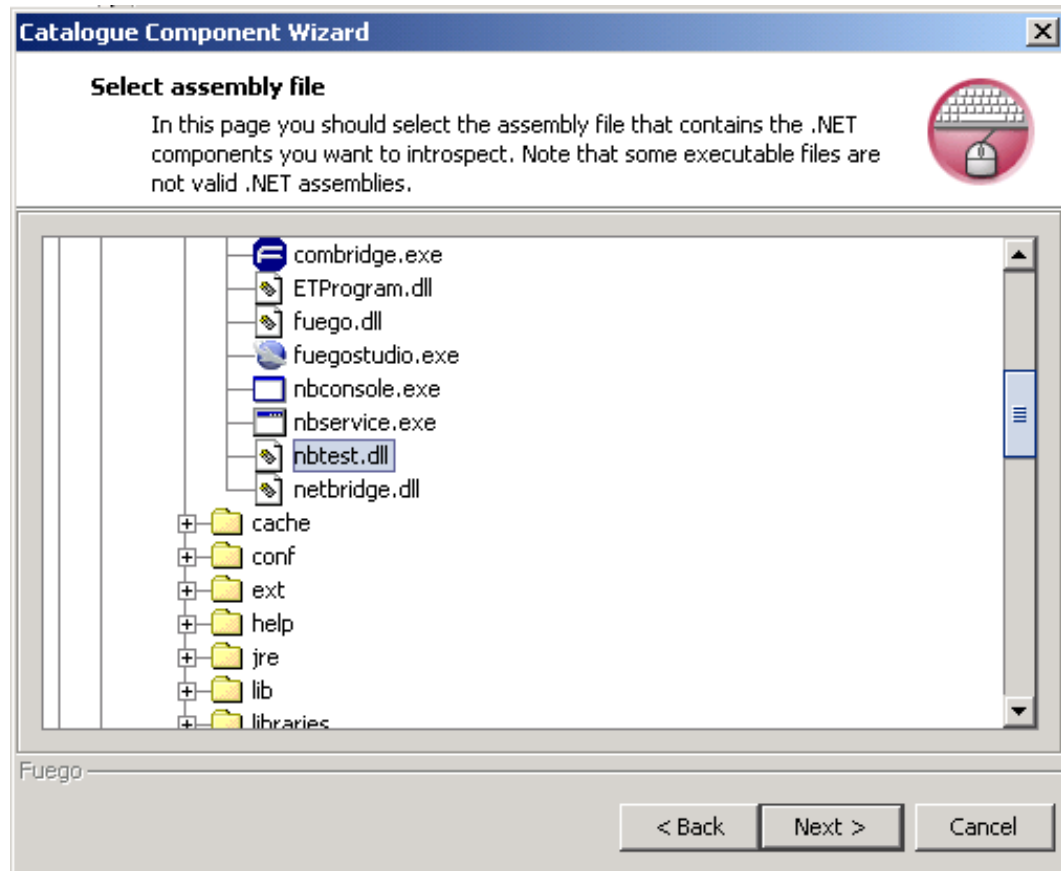


The screenshot shows a Windows-style dialog box titled "Catalogue Component Wizard". Inside, there's a section titled "New Configuration" with the instruction "Complete the information required for the new configuration". A red circular icon with a keyboard and mouse is in the top right. Below, a "Details" section contains the text "Please enter the location of .NET Bridge." and two input fields: "Host" with the value "localhost" and "Port" with the value "5050". At the bottom, there are three buttons: "< Back", "Next >", and "Cancel". The "Fuego" logo is visible in the bottom left corner of the dialog area.

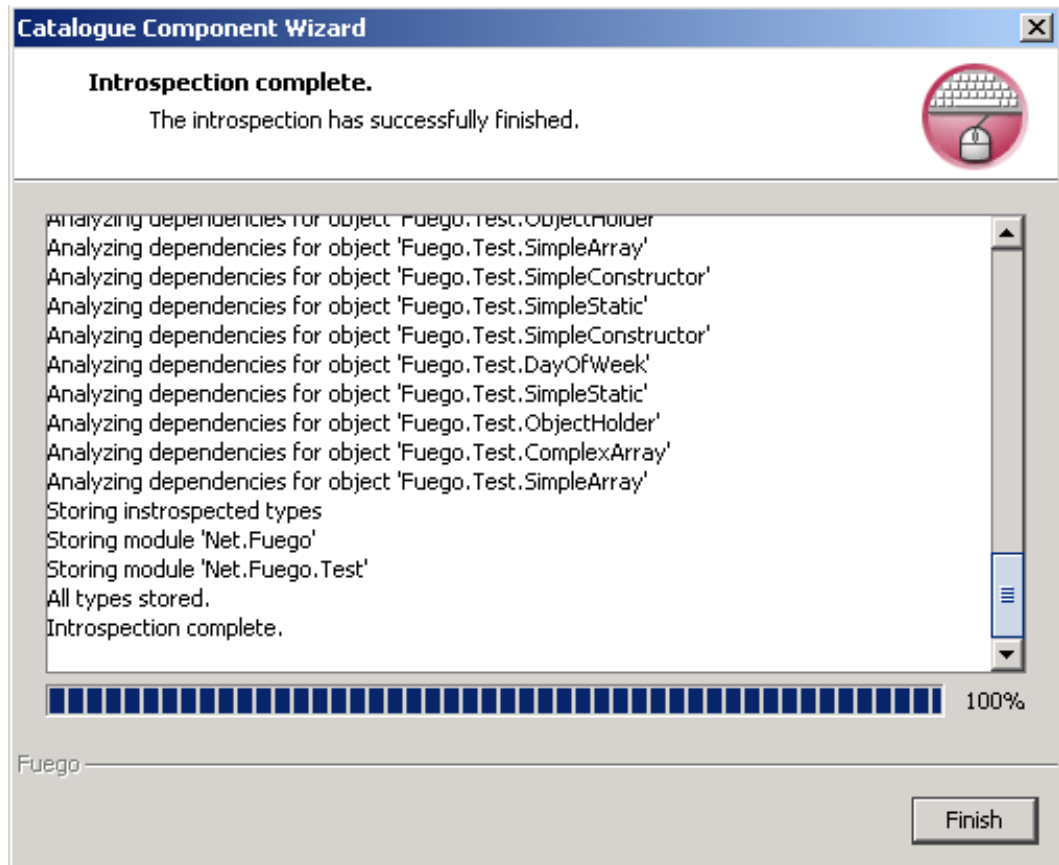
4. The system file in the host where the netbridge is installed and running is shown.



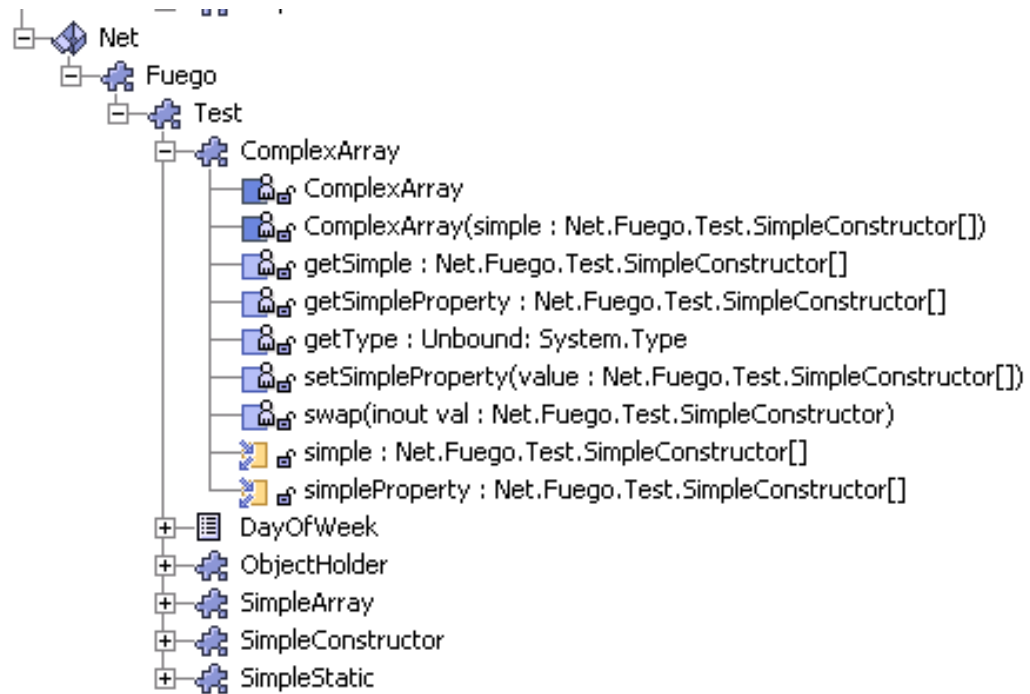
5. Browse and select the *.dll* file that corresponds to the .Net Assembly you want to introspect, and click **Next** to begin the introspection. Not every *.dll* file is .Net file. If you have selected an invalid *.dll* file, an error is displayed. Click **Back** to select the .Net *.dll* file again.



6. Click **Finish** when the introspection ends.



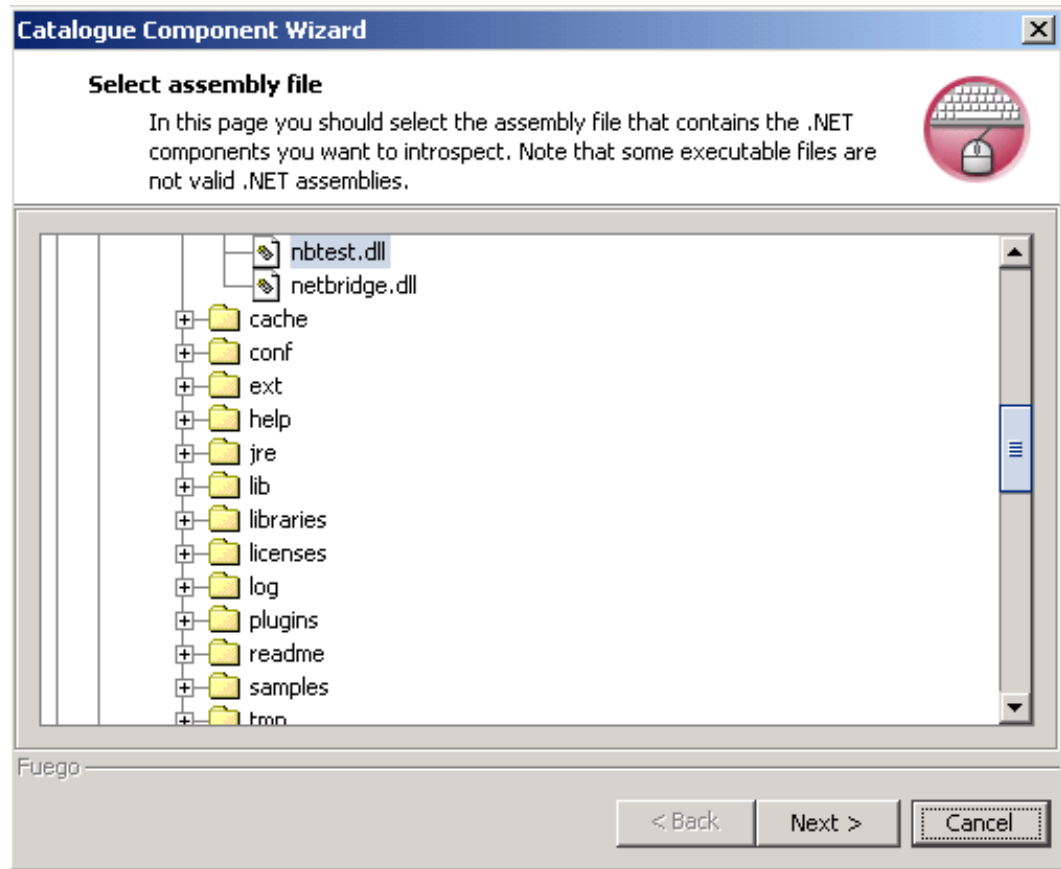
7. The .Net Assembly cataloged is included in the Project Catalog.



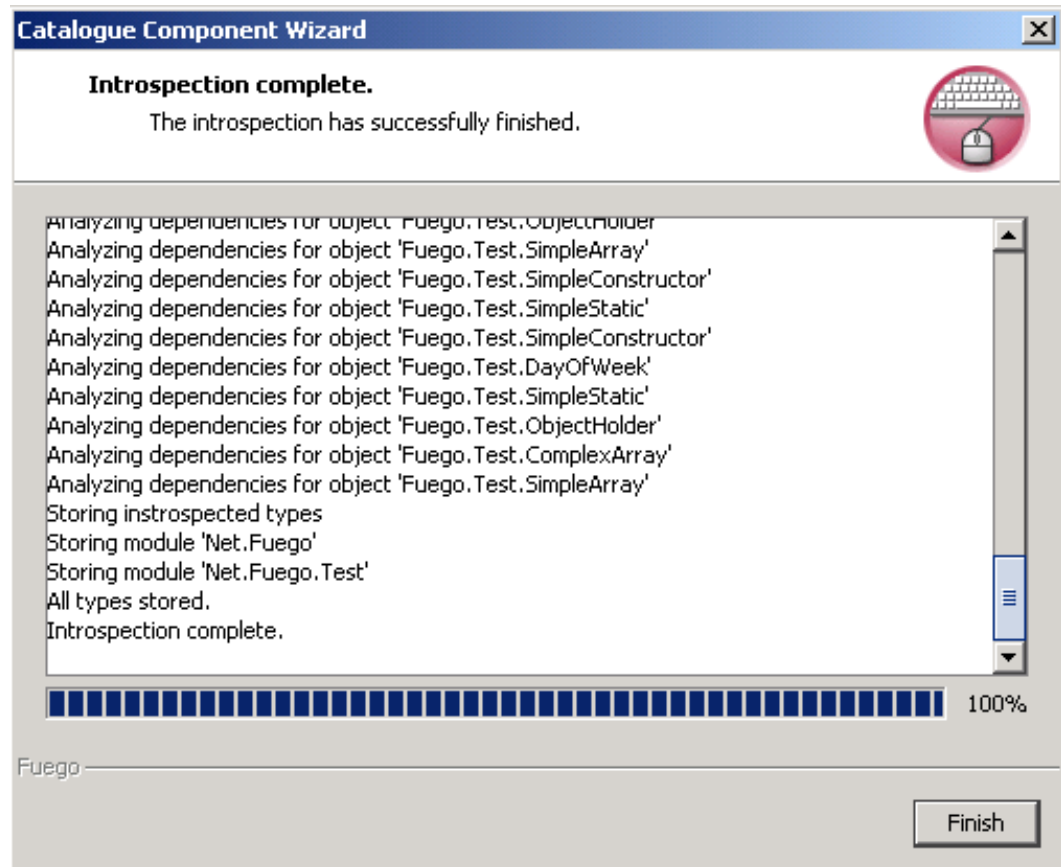
Reintrospecting .Net Assemblies

To reintrospect a .Net Assembly

1. Right-click on the cataloged .Net. Select **Recatalogue Component**. The wizard opens, showing the location and .Net file you originally used to introspect the .Net Assembly. Browse in the file system if the location has changed and click **Next** to continue.



2. The introspection begins. Click **Finish** when ended.



Enterprise JavaBeans ComponentsEJB

The portability and reusability of EJBs make them very attractive to use in a business process to link underlying back-end systems.

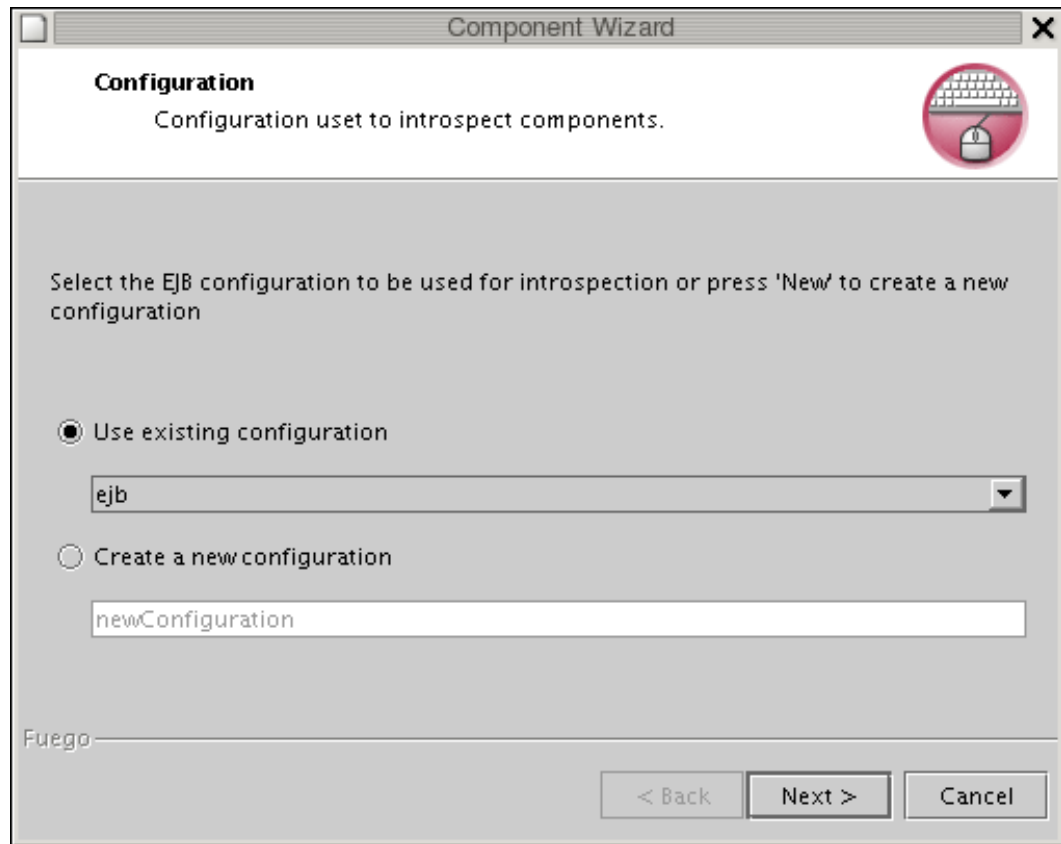
To catalog an EJB, you must:

- Catalog the jar containing the EJB interface definitions as an external resource. By doing this, the jar is automatically copied to the Project lib Repository. Catalog the jar you require as an external resource. For example, if you are working with WebLogic, add the corresponding Web Logic jar that, by default, includes the EJB interface definition.

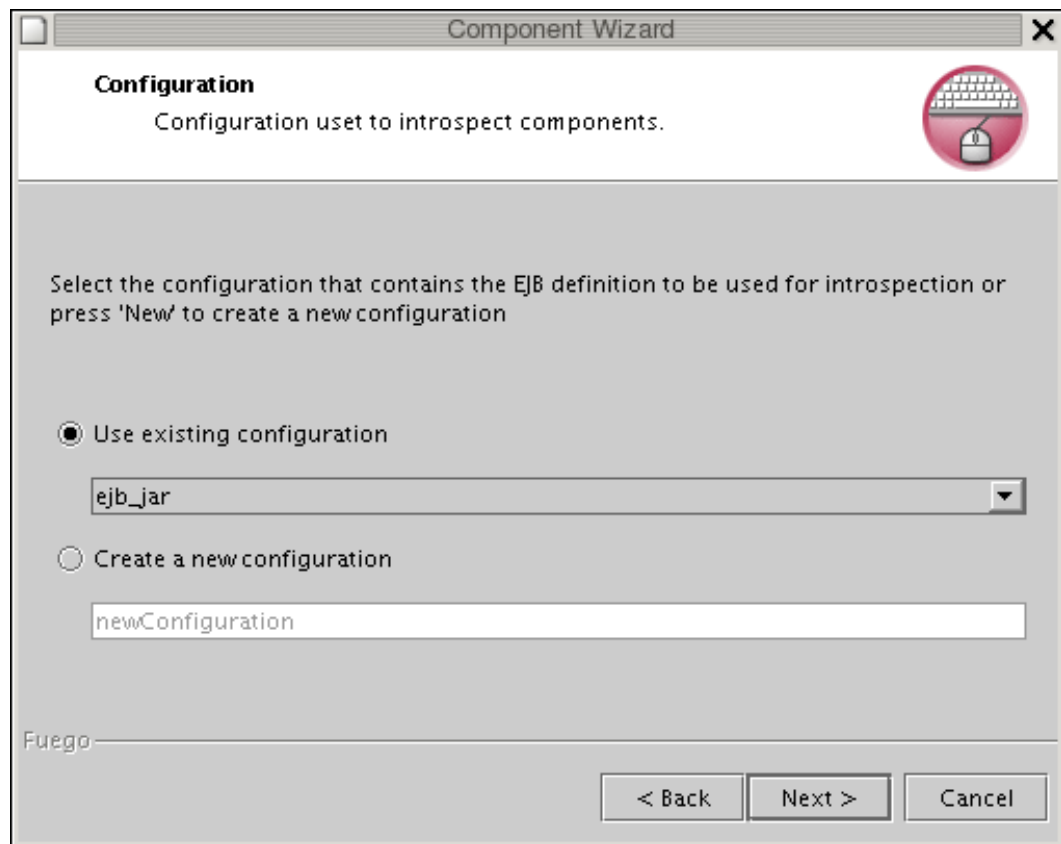
- Catalog the configuration to access the Application Server, filling in the fields with:
 - **Name** and the rest of the fields become active.
 - **Type** configuration type.
 - **Supported types** field automatically filled with *GENERIC_J2EE*
 - **Initial Context Factory** Enter the class name of the JNDI initial context factory used to access the naming service where the home objects are deployed.
 - **URL** Enter the URL of the naming service provider.
 - **Principal & Credentials**, user and password that will be used at connection time.

To catalog an EJB in the FuegoBPM Studio:

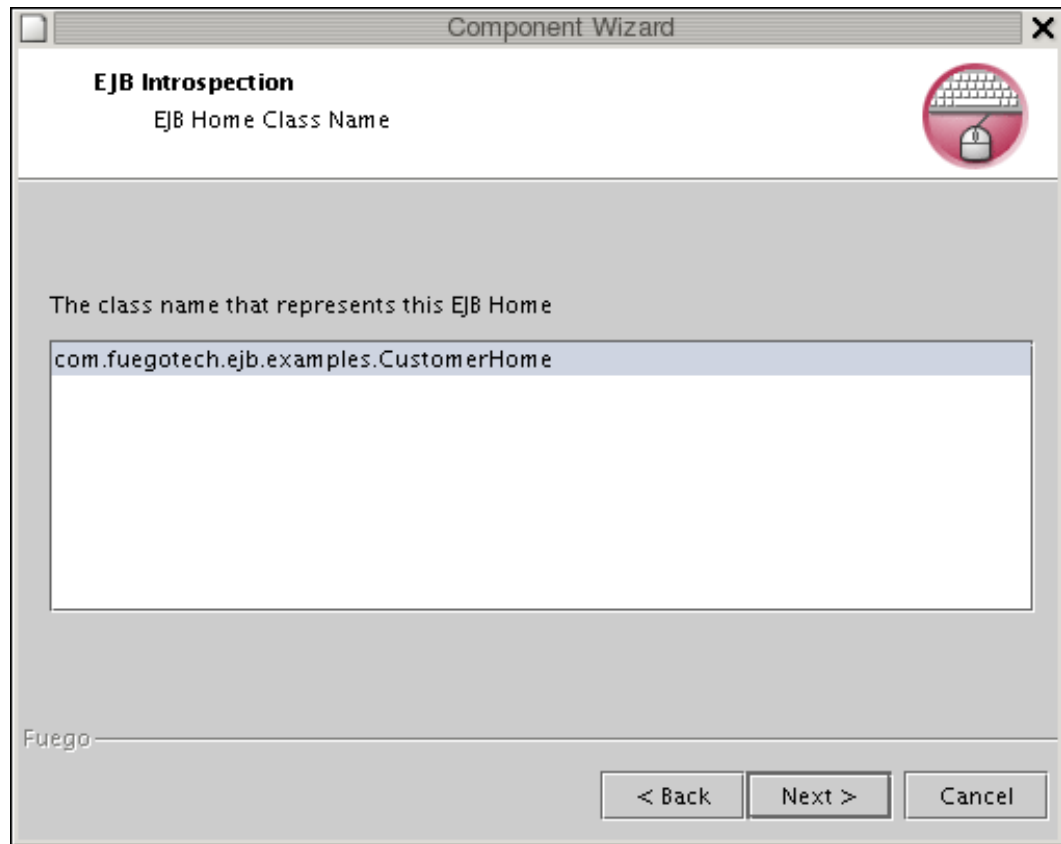
1. Right-click on a module and select **Catalog Component** and then **ejb** from the shortcut menu. The first step of the EJB catalog wizard is displayed.



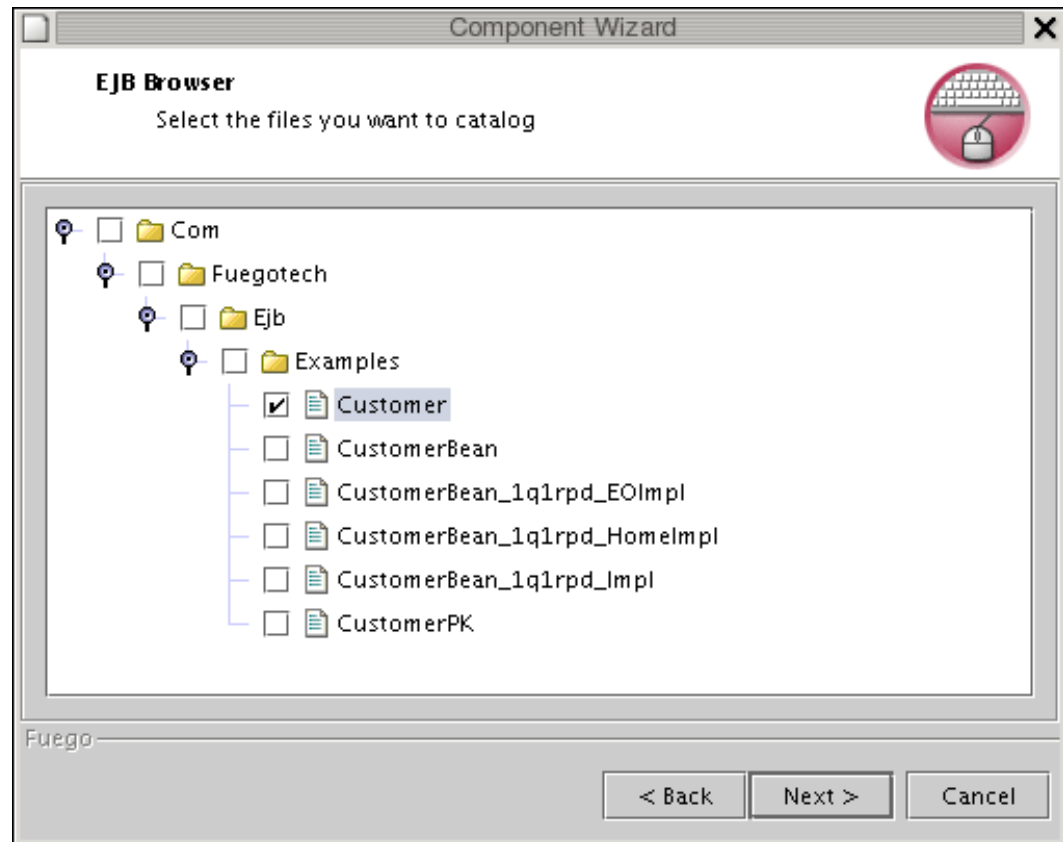
2. Enter the name of the cataloged external resource that contains the EJB interface definition and click **Next** to continue.
3. Enter the name of the configuration you will create or select an existing one. This JCL configuration is the one that contains the jar with the EJB defined by the user.



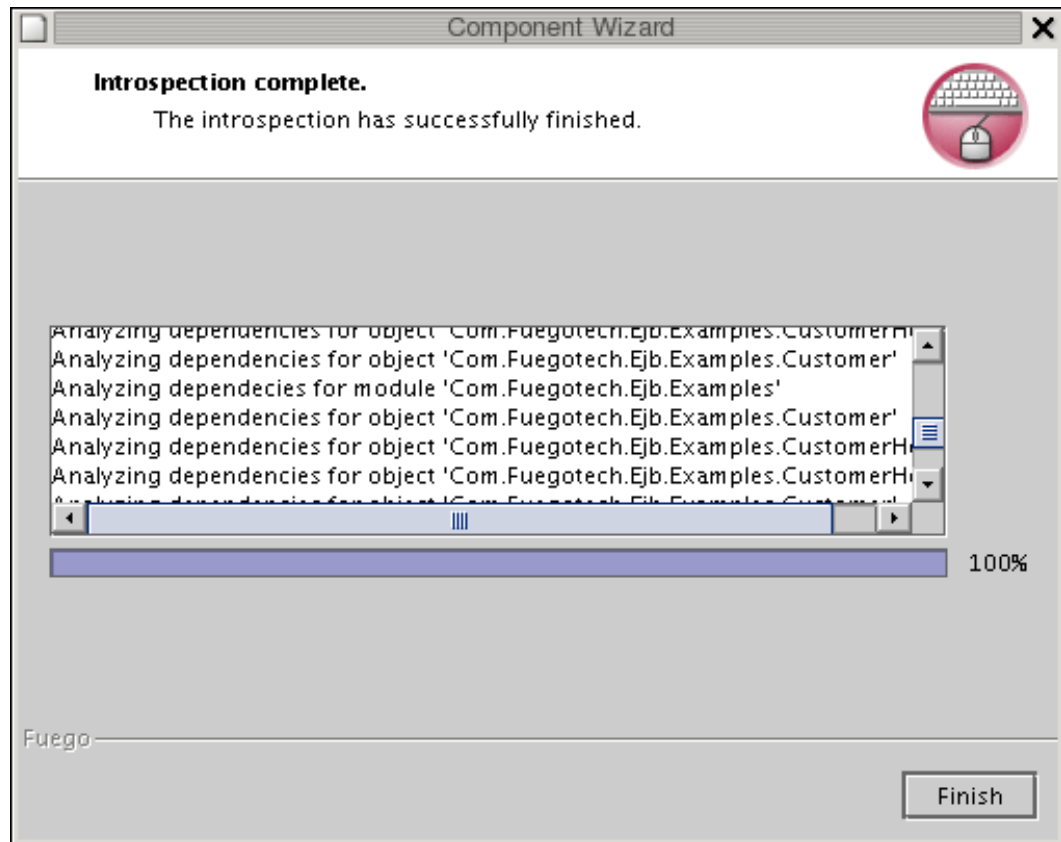
4. The wizard displays the class name that represents the EJB Home. Click **Next** to continue. This class must always be introspected. This is why the wizard does this automatically.



5. The EJB jar file composition is displayed in the next step of the wizard. Select the class you want to catalog by clicking the check box on the left side of its name.



6. Click **Next** to begin with the introspection.



7. When the introspection is finished, click **Finish**.

The module where the EJB has been reintrospected will now display the package class structure.

Using the Component in a Process

After you have added the component to the Project Catalog, you can use it in a business process.

A simple example is:

```
customerHome = (EJB.Com.Fuegotech.Ejb.Examples.CustomerHome)
               EJBHome.locate("EJBConfig")
```

```
customer = customerHome.create("John", "Doe")

display "Customer Name :" + customer.getFirstName() + " " +
        customer.getLastName()
```

The *EJBHome* component is contained in the default **Fuego** module of the Project Catalog. You will always have to use it for the EJB you want to call by applying the *locate* method passing the name of the configuration as an argument you have defined when cataloging the component.

Web Services

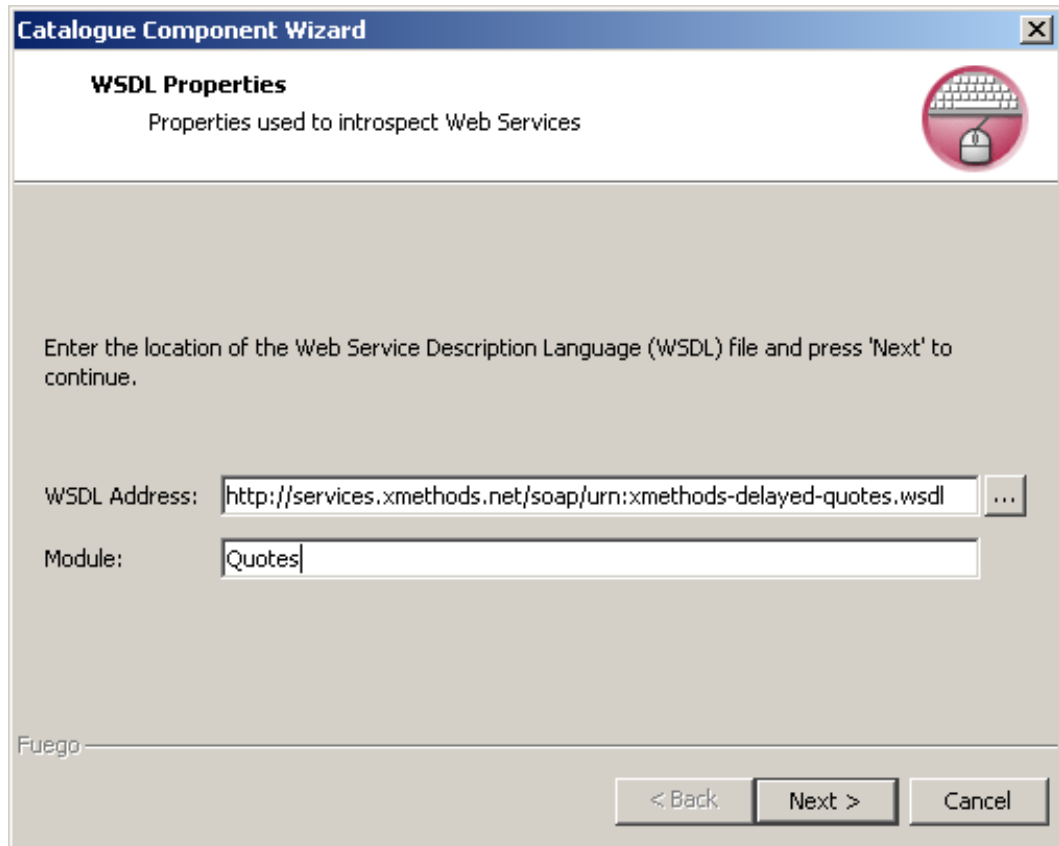
FuegoBPM introspects web services non-intrusively from Web Service Definition Language (WSDL). Any XML components exchanged in service messages are automatically introspected from service descriptions and then mapped to standard components (XML Schema support).

Business and Object Methods developers are able to see service interfaces. An XML or SOAP API is not required to call web services.

Adding a Web Service

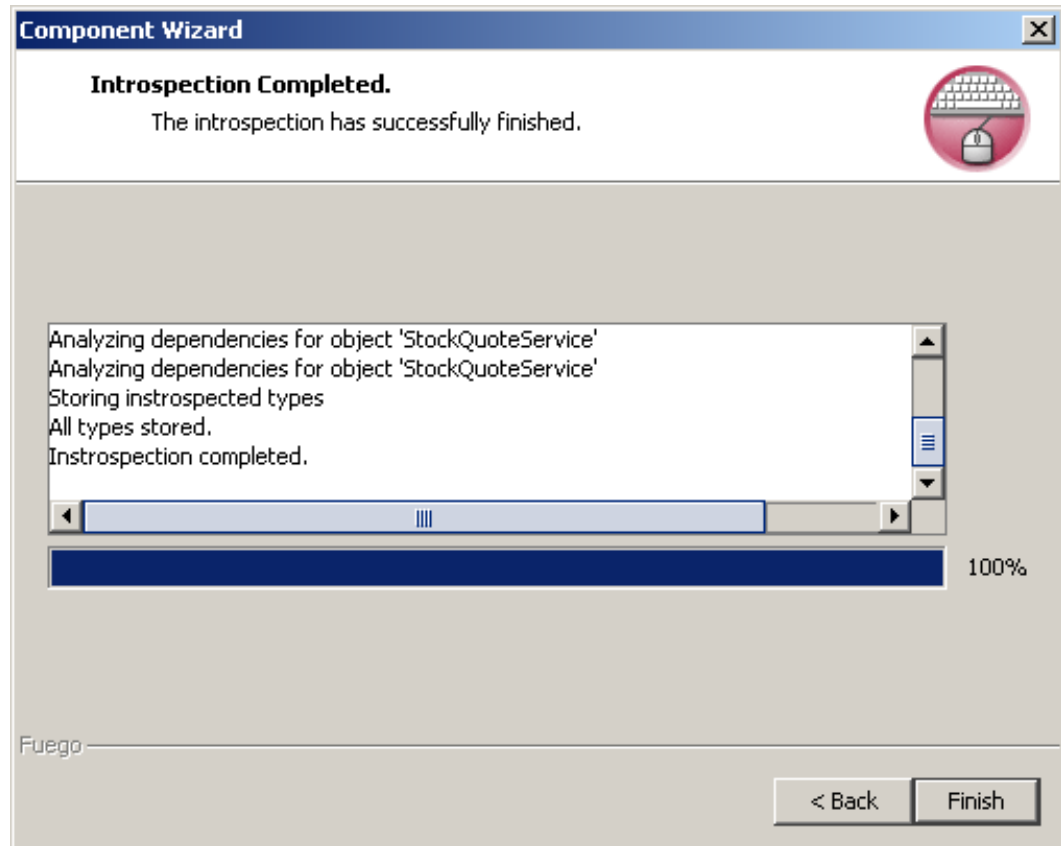
To add a web service:

1. Right-click on the module you want to add the web service to and select **Catalog component** and **web service** from the shortcut menu. The web service wizard is displayed.
2. Enter the complete URL for the web service in the **WSDL Address** field.



The screenshot shows a Windows-style dialog box titled "Catalogue Component Wizard" with a sub-header "WSDL Properties". Below the sub-header is the text "Properties used to introspect Web Services" and a circular icon containing a keyboard and a mouse. The main area of the dialog has a light gray background and contains the instruction: "Enter the location of the Web Service Description Language (WSDL) file and press 'Next' to continue." Below this instruction are two input fields. The first is labeled "WSDL Address:" and contains the text "http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl", followed by a small button with three dots "...". The second is labeled "Module:" and contains the text "Quotes". At the bottom left of the dialog is the "Fuego" logo. At the bottom right are three buttons: "< Back", "Next >", and "Cancel".

3. Enter the name of the module you want to create to hold the cataloged web service in the **Module** field. Click the **Next** button to begin with the introspection.
4. Click the **Finish** button when complete.



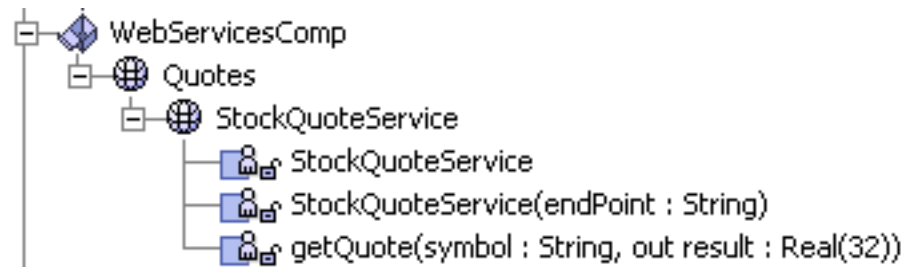
When a web service has been introspected, two new external resources are added to the *External Resource* section of the catalog--one that corresponds to the server access configuration and one that corresponds to the the WSDL file location. If, when introspecting the web service, an existing server configuration matches the one required, it will be reused.

Web Service Example

The following example process uses a development web service available through XMethods. It is a delayed (by 15 minutes) stock quoting service. The process design may include an input statement in the task of a Global activity requesting a stock symbol.

To test a web service example:

- Catalogue the StockQuoteService web service following the wizard.
- Enter `http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl` in the **WSDL file** field of the first step of the wizard.
- After finishing the introspection, the catalog structure displays as in the following image:



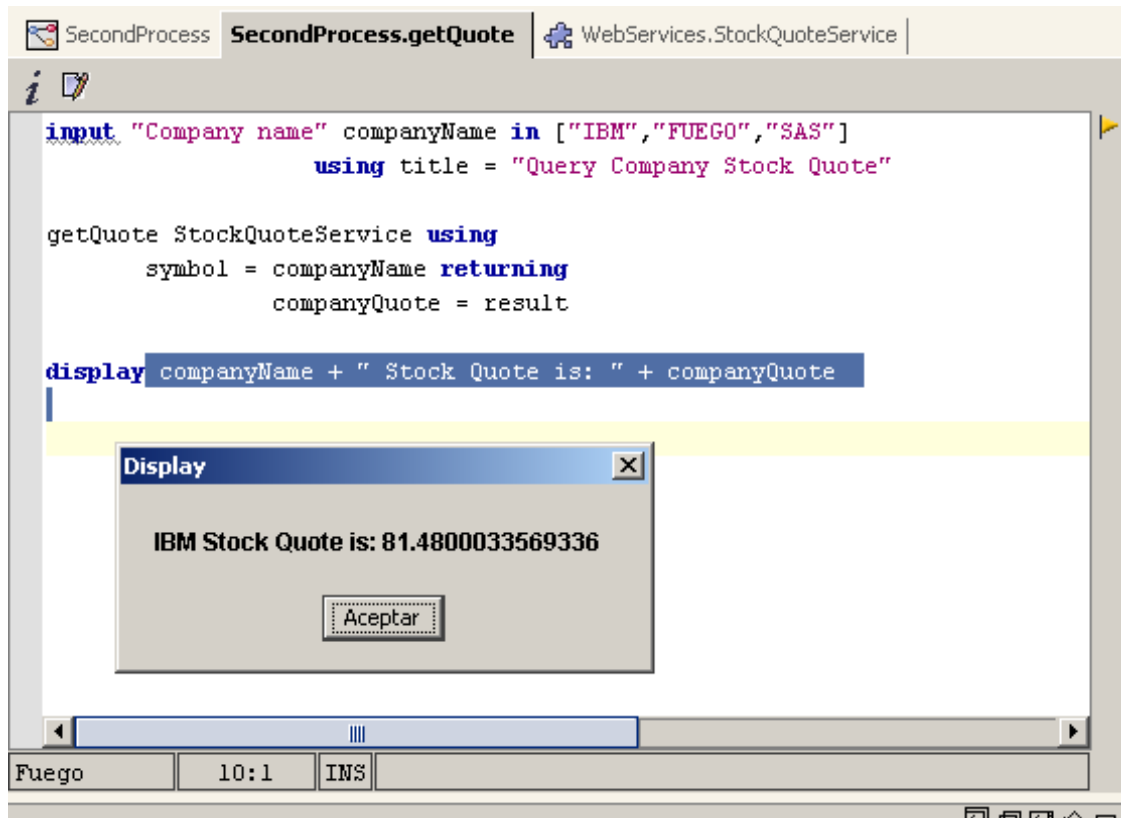
- Open the Business Method task of the Global activity where you want to call the web service.
- Add the following statements to its Method:

```
input "Company name" companyName in ["IBM","FUEGO","SAS"]
    using title = "Query Company Stock Quote"


getQuote StockQuoteService using
    symbol = companyName returning
    companyQuote = result

display companyName + " Stock Quote is: " + companyQuote
```

- You may publish and deploy the process and execute it from Work Portal Application view.
- To test it, run the Method from the Method Editor Debugger. The execution displays as follows:




Tip

 You can drag and drop the Web Service methods to the Method Editor in order to visualize the usage template.

XML Components

XML schemas can be introspected into the component catalog. The XML schema component can be called from your process design to incorporate functionality for typed parsing and typed XML document generation.

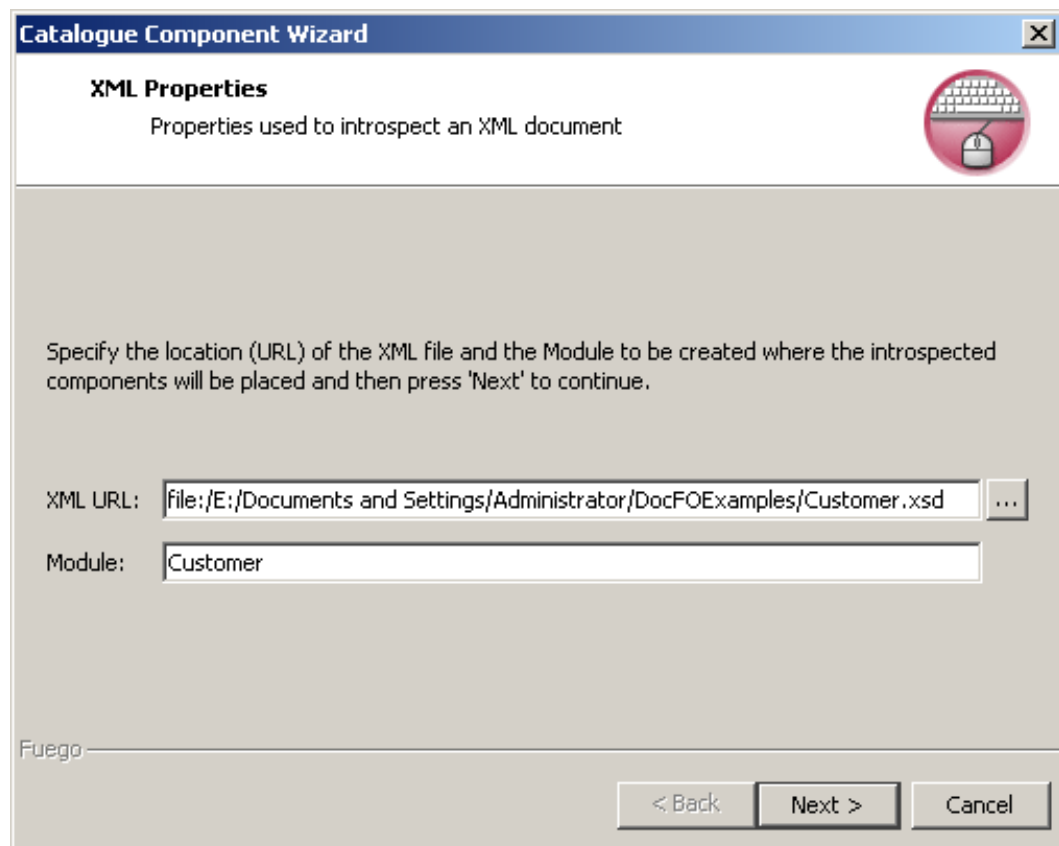
Note

 Schema redefinition is not supported. Only data types from an XML schema are supported. If you only have the XML document, but do not have a schema or you have a DTD file, you can create a schema by using one of many tools available on the Internet in order to create a schema.

Adding an XML Schema to the Project Catalog

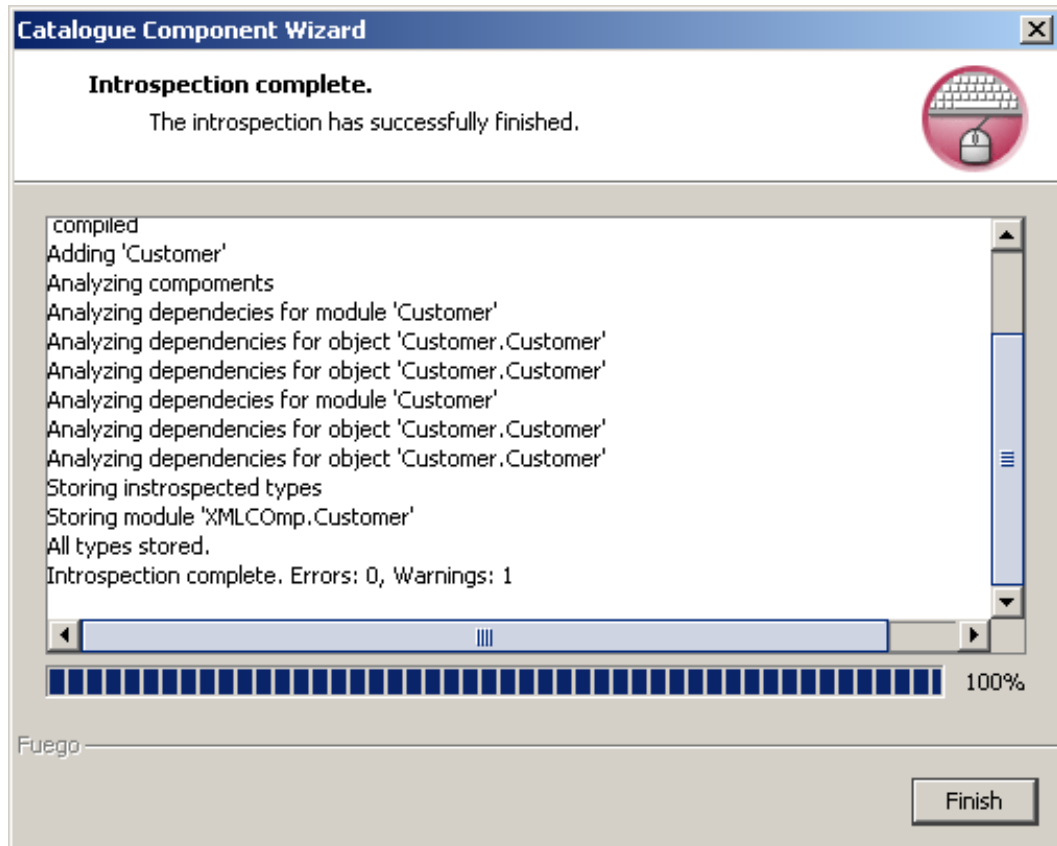
A wizard is provided to catalog an XML schema in the Project Catalog. Perform the following procedure:

1. Right-click on the module where you want to add the XML component and select **Catalogue component** and **xml** from the shortcut menu. The wizard is displayed.
2. Specify the location (URL) of the XML file and the name of the module you want to create to hold the XML components below the module where you started cataloging. Click **Next** to continue.



The screenshot shows a Windows-style dialog box titled "Catalogue Component Wizard". It has a blue title bar with a close button (X) in the top right corner. The main content area is divided into two sections. The top section is titled "XML Properties" and contains the text "Properties used to introspect an XML document". To the right of this text is a circular icon containing a keyboard and a mouse. The bottom section contains the instruction: "Specify the location (URL) of the XML file and the Module to be created where the introspected components will be placed and then press 'Next' to continue." Below this instruction are two input fields. The first is labeled "XML URL:" and contains the text "file:/E:/Documents and Settings/Administrator/DocFOExamples/Customer.xsd". To the right of this field is a small button with three dots "...". The second field is labeled "Module:" and contains the text "Customer". At the bottom left of the dialog box, the word "Fuego" is displayed. At the bottom right, there are three buttons: "< Back", "Next >", and "Cancel".

3. Click **Finish** when complete.




4. The introspected XML schema is shown in the Catalog structure.

Examples

XML Schema Introspection Example

The following is an XML schema example for an XML document that may be used in a business process.

Note

 If you only have the XML document, but you do not have a schema or you have a DTD file, you can create a schema by using one of many tools available on the Internet to create the schema. The following example was generated using one of these tools.

```
<?xml version="1.0" encoding="UTF-8" ?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:simpleType name="Address" type="xsd:string"/>
  <xsd:simpleType name="CustomerCode" type="xsd:string"/>
  <xsd:simpleType name="CustomerName" type="xsd:string"/>
  <xsd:simpleType name="CodPay" type="xsd:string"/>
  <xsd:simpleType name="CustDisc" type="xsd:string"/>
  <xsd:simpleType name="CustType" type="xsd:string"/>
  <xsd:simpleType name="Mail" type="xsd:string"/>

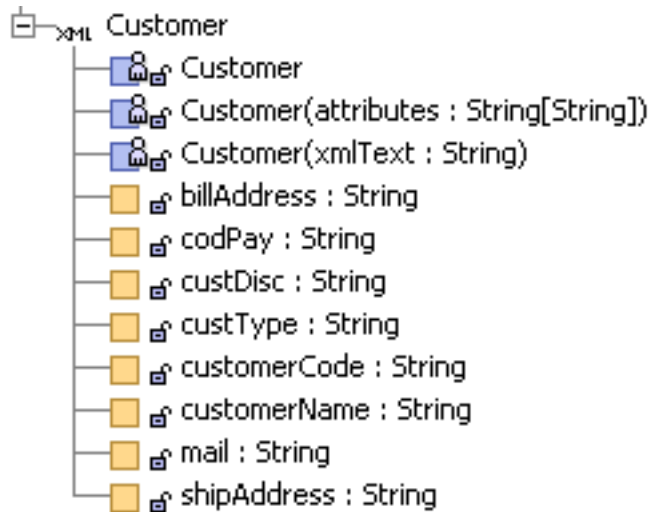
  <xsd:group name="shipAndBill">
    <xsd:sequence>
      <xsd:element name="ShipAddress" type="Address"/>
      <xsd:element name="BillAddress" type="Address"/>
    </xsd:sequence>
  </xsd:group>

  <xsd:complexType name="Customer">
    <xsd:sequence>
      <xsd:choice>
        <xsd:element name="CustomerCode"
          type="CustomerCode"/>
        <xsd:element name="CustomerName"
          type="CustomerName"/>
        <xsd:element name="CodPay" type="CodPay"/>
        <xsd:element name="CustDisc" type="CustDisc"/>
        <xsd:element name="CustType" type="CustType"/>
        <xsd:group ref="shipAndBill"/>
        <xsd:element name="Mail" type="Mail"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>


  <xsd:element name="customer" type="Customer"/>

</xsd:schema>
```

After introspecting the XML schema detailed above, the Project catalog structure appears as shown in the following image:



Tip

 You can drag and drop the XML attributes and methods to the Method Editor in order to visualize the usage template.

Using Introspected XML in a Method

Create a Customer XML file based on the introspected definition

```
customer as XMLComp.Customer.Customer
customer = XMLComp.Customer.Customer()

customerDOS as XMLComp.Customer.Customer
customerDOS = XMLComp.Customer.Customer()

customer.customerCode = "NEW"
customer.customerName = "John Smith"
customer.custDisc = null
customer.billAddress = "Madison 2939 NY"
customer.shipAddress = "Madison 2939 NY"
customer.codPay = null
customer.custType = null
customer.mail = "pp@com.com"

store customer
    using targetFile = "C:/tmp/customer.xml"
```

Load an XML data file containing a customer using the `loadFromUrl` method

The `loadFromUrl` method can receive as parameter an URL, pointing the xml file, like `file://home/sharedDocuments/test.xml` or `http://...`

```
customer as XMLComp.Customer.Customer
customer = XMLComp.Customer.Customer( )

customer.loadFromUrl("file://c:/tmp/customer.xml")
```

While `loadFromUrl()` method to parse XML documents is interesting from a coding perspective, it is recommended to avoid its use on a large XML file as it uses DOM and reads the whole file into memory.

Load an XML data file containing a customer using the `load` method

The `load` method can receive as parameter:

- an URL,
- a path location to the xml file OS dependant. This is, for example : `C:\\documents\\test.xml` or `//home//sharedDocuments//test.xml`, Windows and Unix.
- the xml file content.

How should you read in a large and complex XML file?

Reading a very large and complex XML file into memory all at once is not efficient. It causes large amounts of memory to be consumed and will degrade the Server's performance.

Note

Remember that if you don't have an XSD file, it can be easily generated using an XML editor (like XML Spy).

By having an XSD file that you can introspect, there is a much more efficient technique that does not require that the whole file be read into memory. The code shown below does not read the XML whole file into memory and only accesses the file when it needs to read in more object information from it.

After introspecting the XML , create a new Fuego Object method:

```
// In production, do not hard code a local directory
// like this.
// It is only here in this example for clarity.

customer as XMLComp.Customer.Customer
customer =
  XMLComp.Customer.Customer("file://c:/tmp/customer.xml")

display customer.billAddress
display customer.custDisc
display customer.customerName
```

The above code expects a "customer.xml" XML file in the "c:\tmp" directory. Use the following XML text as this customer.xml file for this example :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<Customer>
  <CustomerCode>NEW</CustomerCode>
  <CustomerName>John Smith</CustomerName>
  <CustDisc></CustDisc>
  <BillAddress>Madison 2939 NY</BillAddress>
  <ShipAddress>Madison 2939 NY</ShipAddress>
  <CodPay></CodPay>
  <CustType></CustType>
  <Mail>pp@com.com</Mail>
</Customer>
```


See more examples in **Fuego Objects Examples** where a Fuego object is created based on XML files.

Java Components

Java components are compiled Java language programs. A *.class file or a *.jar file may be used.

Note



The class file must reference a package name using the *package* statement in the uncompiled Java file.

To add a Java component you must perform the following steps:

Compiling a Java class

Make sure that the following line is in your Java program before you compile it:

```
package PackageName;
```

where *PackageName* is a name, you choose for the package. In the example below, the package name is *dummy* and the sub-package is *lib*.

```
package dummy.lib;

import java.lang.*;
import java.io.*;

public class userException
    extends Exception
    implements java.io.Serializable
{
    public userException(String excMess)
    {
```

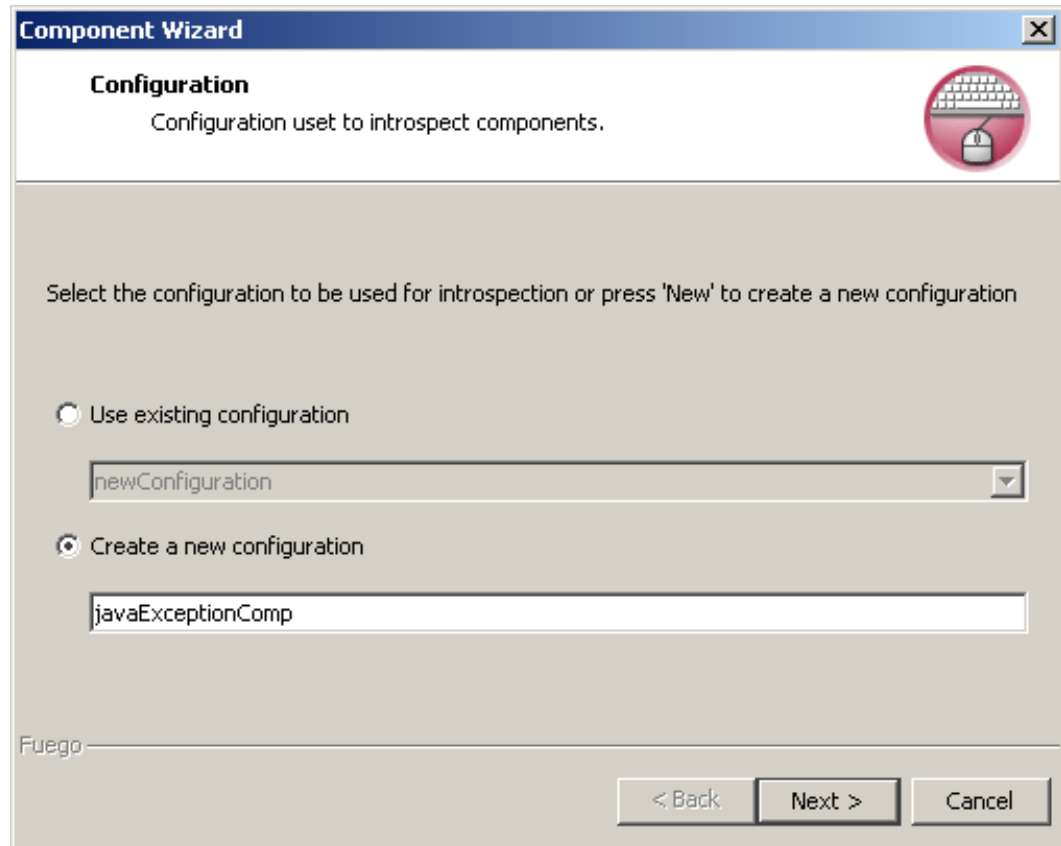
```
        userMessExcep = excMess;  
    }  
    public String userMessExcep = "";  
}
```

Compile your Java program and generate a jar file.

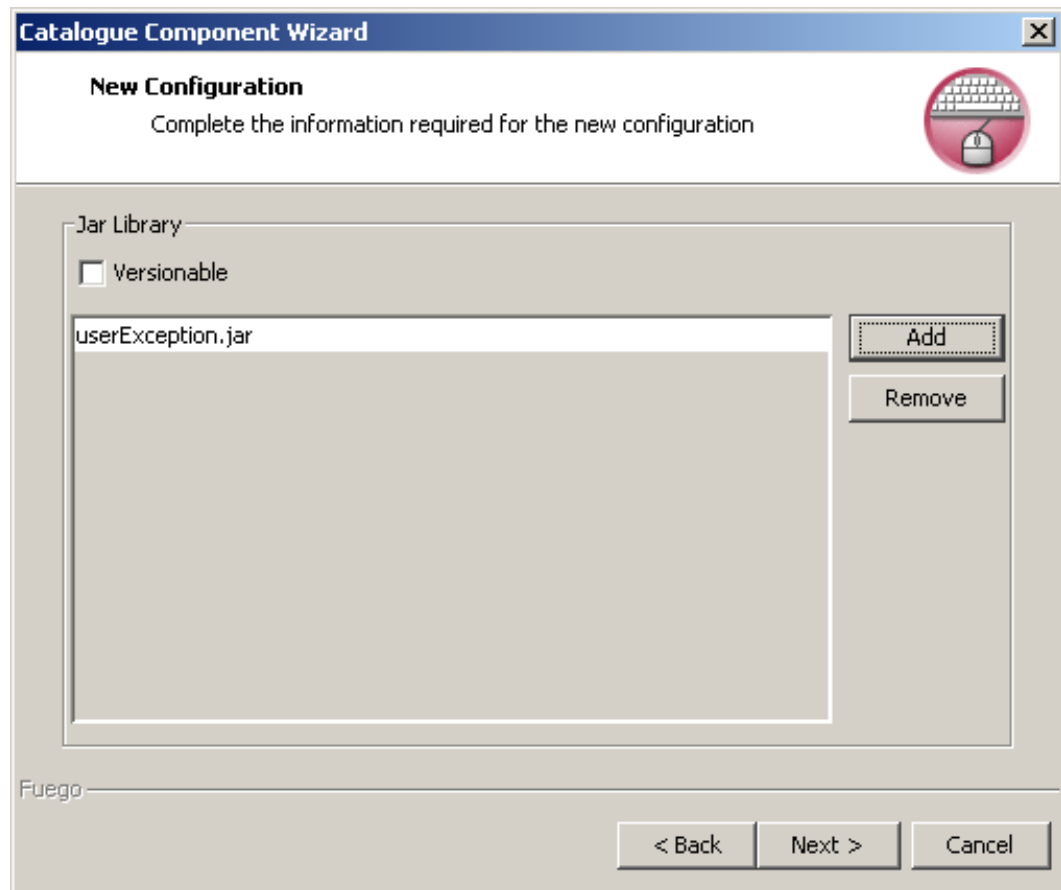
Cataloging a Java component

A wizard is provided to catalog a Java component. Perform the following:

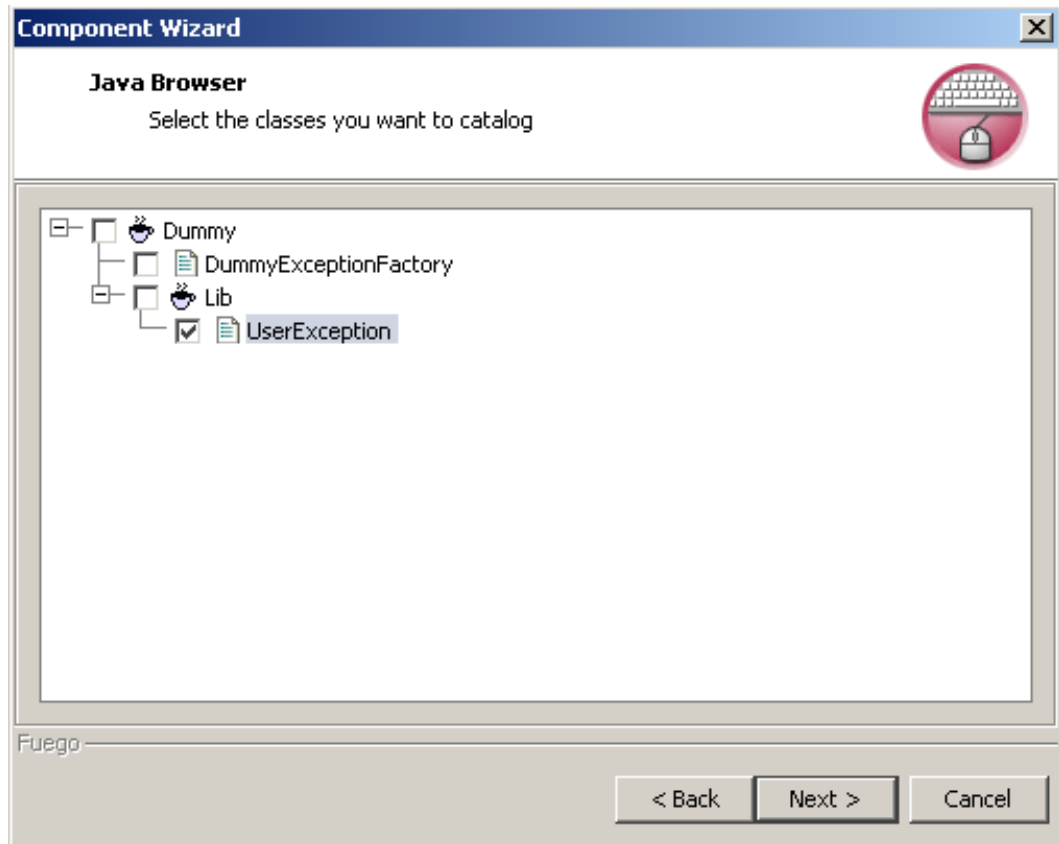
1. You can create a new module to hold your Java component, or right-click on an existing module and select **Catalog Component** and **Java** from the shortcut menu. The wizard is displayed.
2. Use an existing configuration if you want to catalog any other class included in an already specified jar file or create a new one.
3. Click **Next** to continue.



4. Add jars to the library. In order to add a new jar file, click **Add**. The *Project lib Repository: The "Project lib Repository: Select a file to add"* dialog is displayed. Browse to the location of the jar file and click the **Open** button. The file is displayed in the list. The Versionable property indicates that the jars of this library will be stored in the directory and it will generate a new copy of it whenever a jar is replaced by a different one or a jar is removed or added. Usually the jars that should be marked as versionable should be the one that contains user components and are expected to change frequently. Click **Next** to continue.



5. The *Java Browser* step is displayed. Select the classes you want to catalog by clicking the check box next to its name.
6. Click **Next** to begin the introspection.




7. Once the introspection has finished, click **Finish** to complete the wizard.
8. The introspected Java is displayed as part of the Project Catalog.

The jar file is copied to the *lib* directory under the Project directory structure. When you remove a jar from the library, it is also removed from the lib directory.

Please refer to Java Class Libraries for further information on how JCLs are managed by FuegoBPM and their relation with the Control System Version.

Note

 Only the public classes of a JAR are cataloged. Those that are included in the JAR and are not public are not included in the introspection

When to catalog a Java component or define its JAR as an external resource (JCL)

It is only necessary to catalog the classes that will be invoked from a BP-Method or those classes that are invoked in the public API of another Java class invoked from a BP-Method. For example, if the A class is invoked from a BP-Method, which has an attribute of type B, the B class has to be cataloged too.

Using the component as an Instance Variable

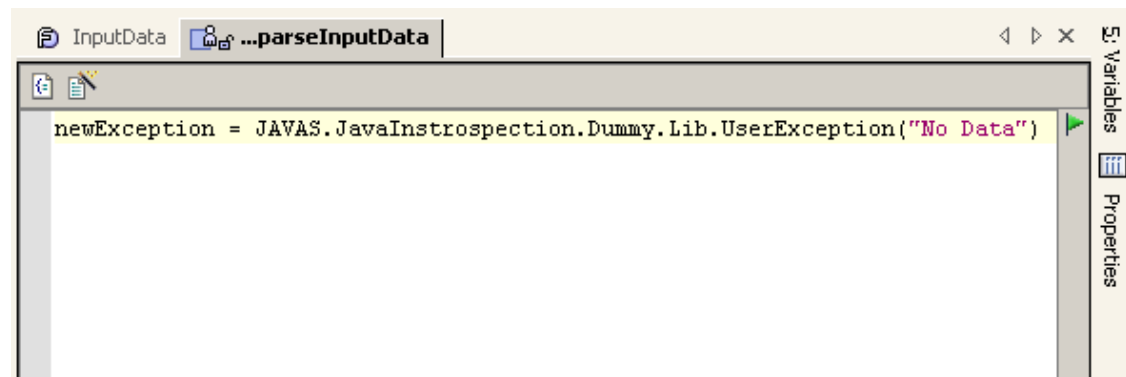
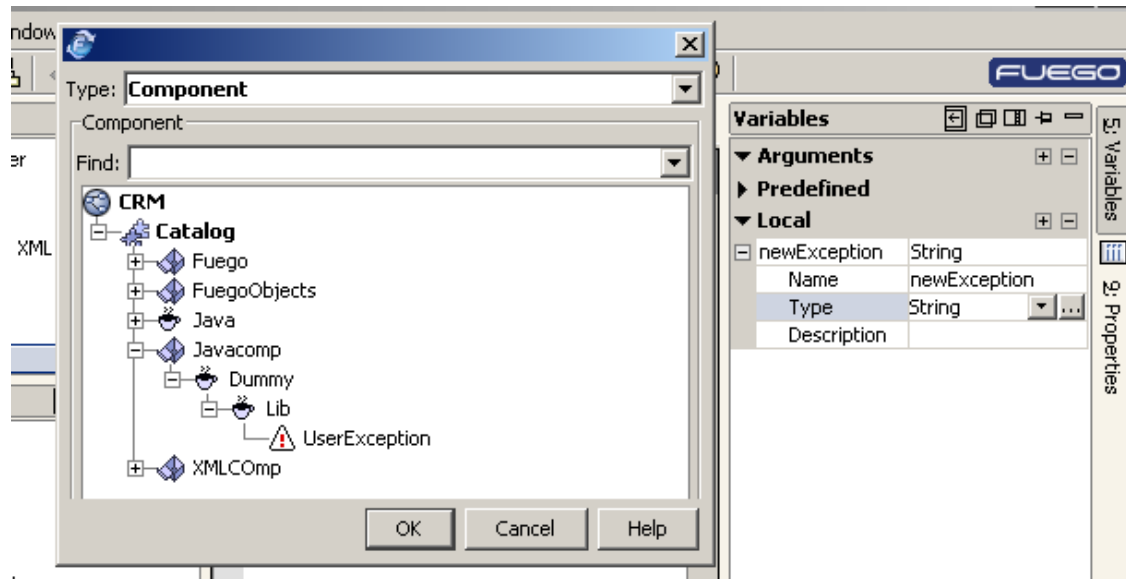
To use a java component as an Instance Variable, it **must** implement **java.io.Serializable**. This allows FuegoBPM to persist the instance variable.

Call the component in a process

Now that you have added the Java component to the catalog, you can use it in the Business Process Methods statements while designing a process.

To use the component in a process

1. Open the task Method where you want to call the Java component.
2. The Java component is available as a type to define variables or to call in your Method statements.



Java Naming Directory Interface Components JNDI

Java Naming and Directory Interface (JNDI) components act as an interface to your processes. One interface can communicate with directory services to retrieve, delete, and add objects.

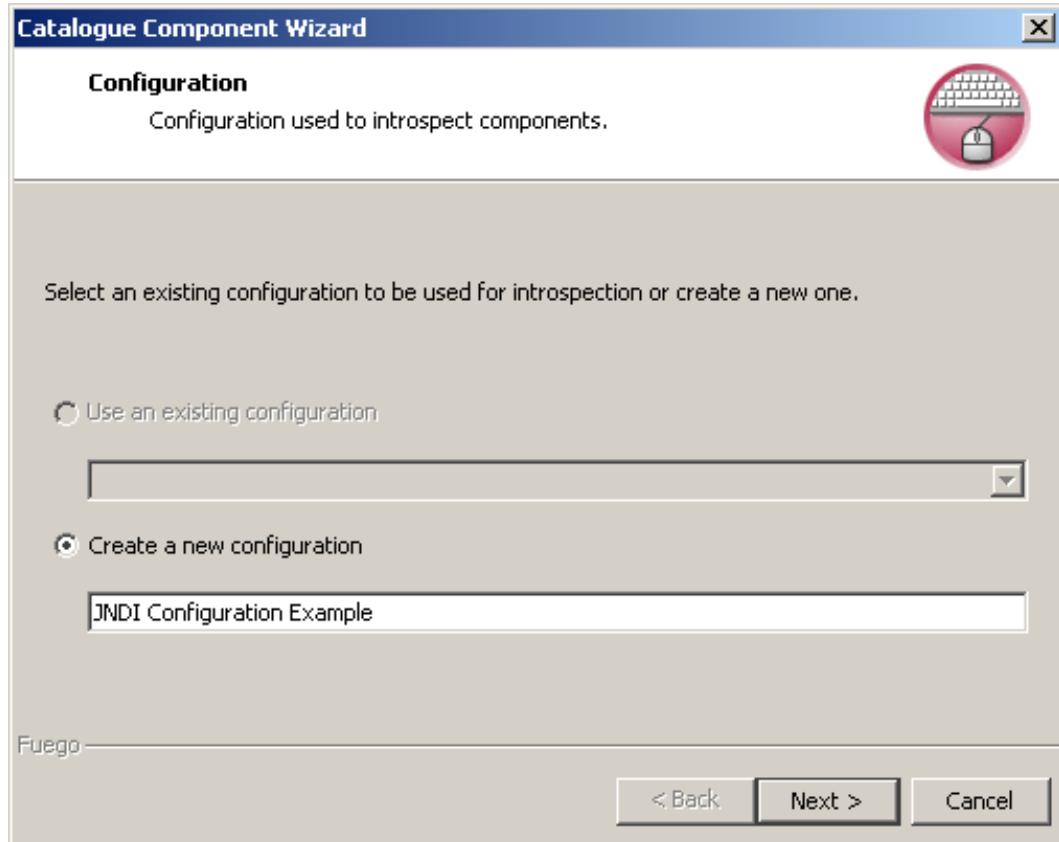
Catalog the JNDI Component

To catalog a JNDI component:

1. Add a new module to hold your component or right-click on an

existing module and select **Catalogue Component** and **jndi** from the shortcut menu.

2. Use an existing configuration or create a new one.



The screenshot shows a dialog box titled "Catalogue Component Wizard" with a close button (X) in the top right corner. The main heading is "Configuration" with a subtext "Configuration used to introspect components." and a keyboard icon. Below this, it says "Select an existing configuration to be used for introspection or create a new one." There are two radio button options: "Use an existing configuration" (unselected) and "Create a new configuration" (selected). Under "Use an existing configuration" is an empty dropdown menu. Under "Create a new configuration" is a text input field containing "JNDI Configuration Example". At the bottom left is a "Fuego" logo. At the bottom right are three buttons: "< Back", "Next >", and "Cancel".

If you choose to create a new configuration, enter the following information:

Catalogue Component Wizard

New Configuration
Complete the information required for the new configuration

Details

Initial Context Factory:

URL:

Principal:

Credentials:

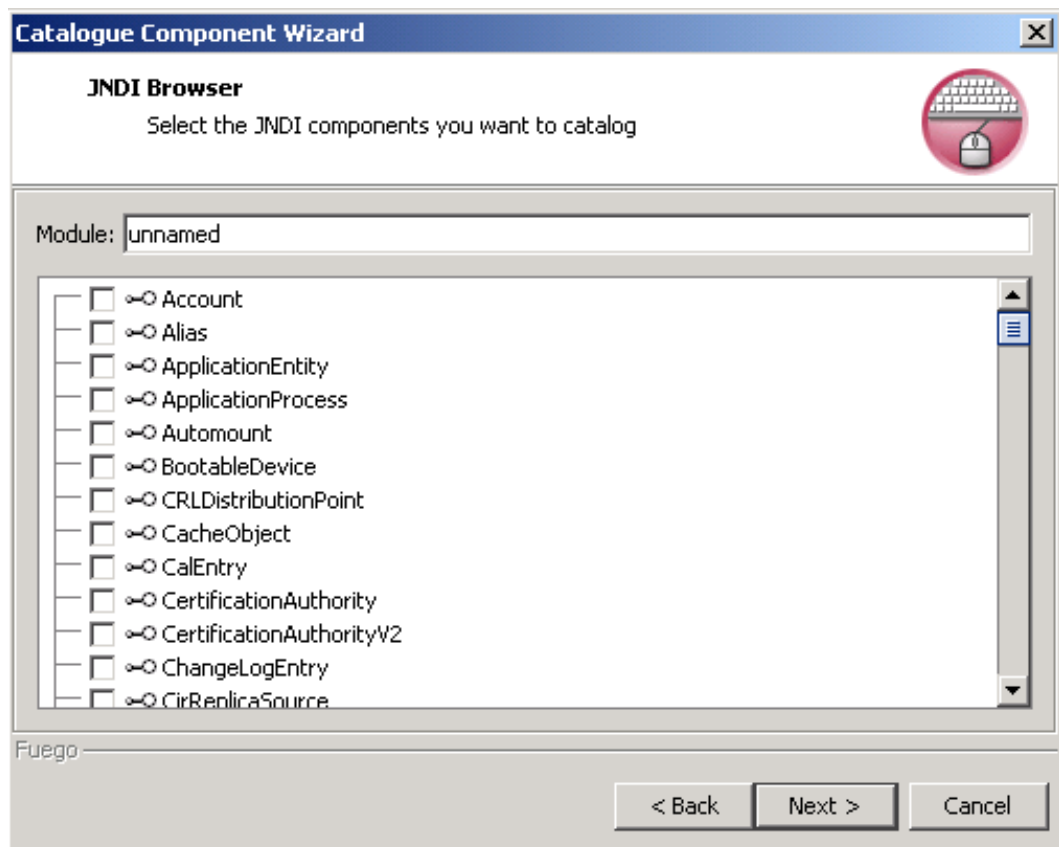
Referrals: (dropdown menu showing: Follow, ignore, throw)

Fuego

< Back Next > Cancel

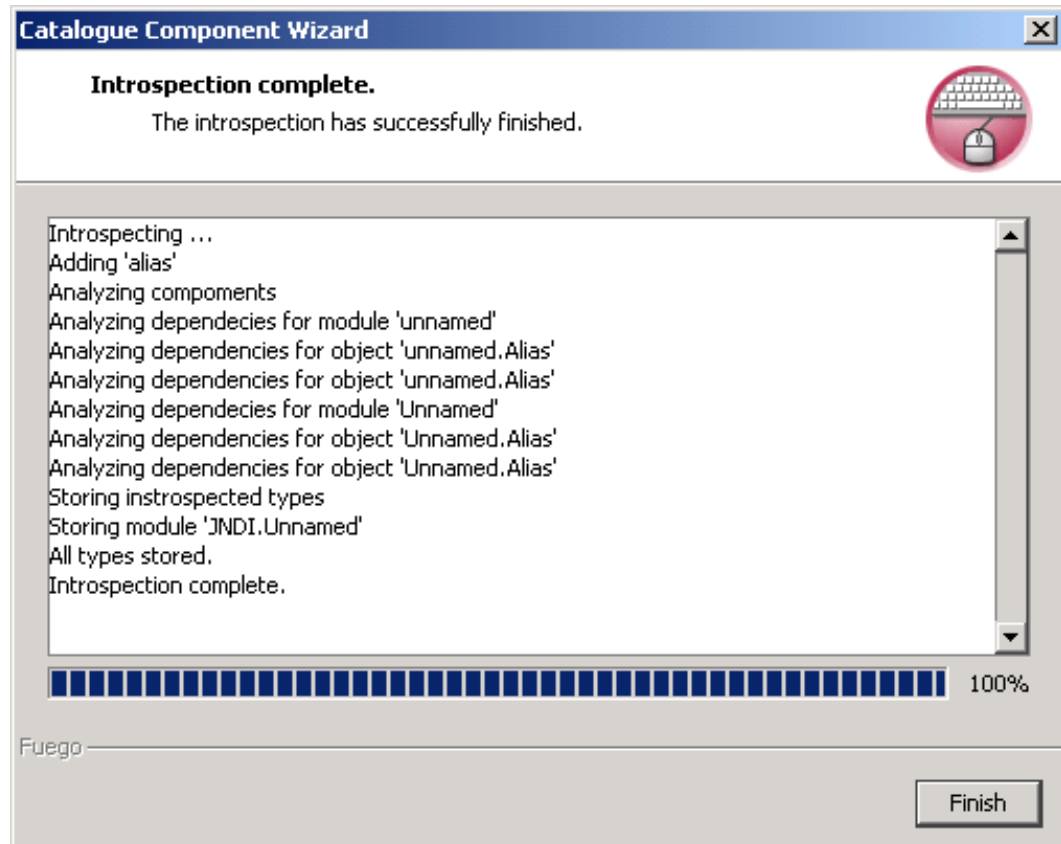
3. The initial context factory can be: `com.sun.jndi.Ldap.LdapCtxFactory` or your own Context factory.
4. The URL for the **URL** field is the URL you use to connect to the directory service. For Netscape iPlanet directory services, the format is generally as follows: `ldap://host:port/o=company,c=country` For Microsoft Active Directory, the format is generally as follows: `ldap://host:port/dc=subdomain,dc=domain,dc=com`
5. The **Principal** is the root distinguished name for the directory service. For example, Netscape iPlanet directory services usually use: `cn=root` Microsoft Active Directory users usually just enter *root*.
6. Enter your password for the directory service in the **Credentials** field.

7. Select the Referral option from the pop up menu. It indicates what action to take when the JNDI directory is distributed and an entry is not found.
 - a. *follow*: the entry will be looked for directly.
 - b. *ignore*: the entry is not looked for, it is as if it doesn't exist,
 - c. *throw*: you will have to catch and manage the exception.
8. Once the configuration has been created/selected, the following screen is displayed where you can select the methods you want to use.



9. Select the check box next to each one of the methods you want to use.

10. Click the **Next** button to continue.
11. Once the introspection has finished, click **Finish** to close the wizard.



Using the Component in a Process

Now that you have added the JNDI component to the catalog, you can use it in the Business Process Methods in FuegoBPM Studio.

To call the component in a process design:

- Open FuegoBPM Studio.
- Open an existing process or create a new process.
- Add an activity to the process.

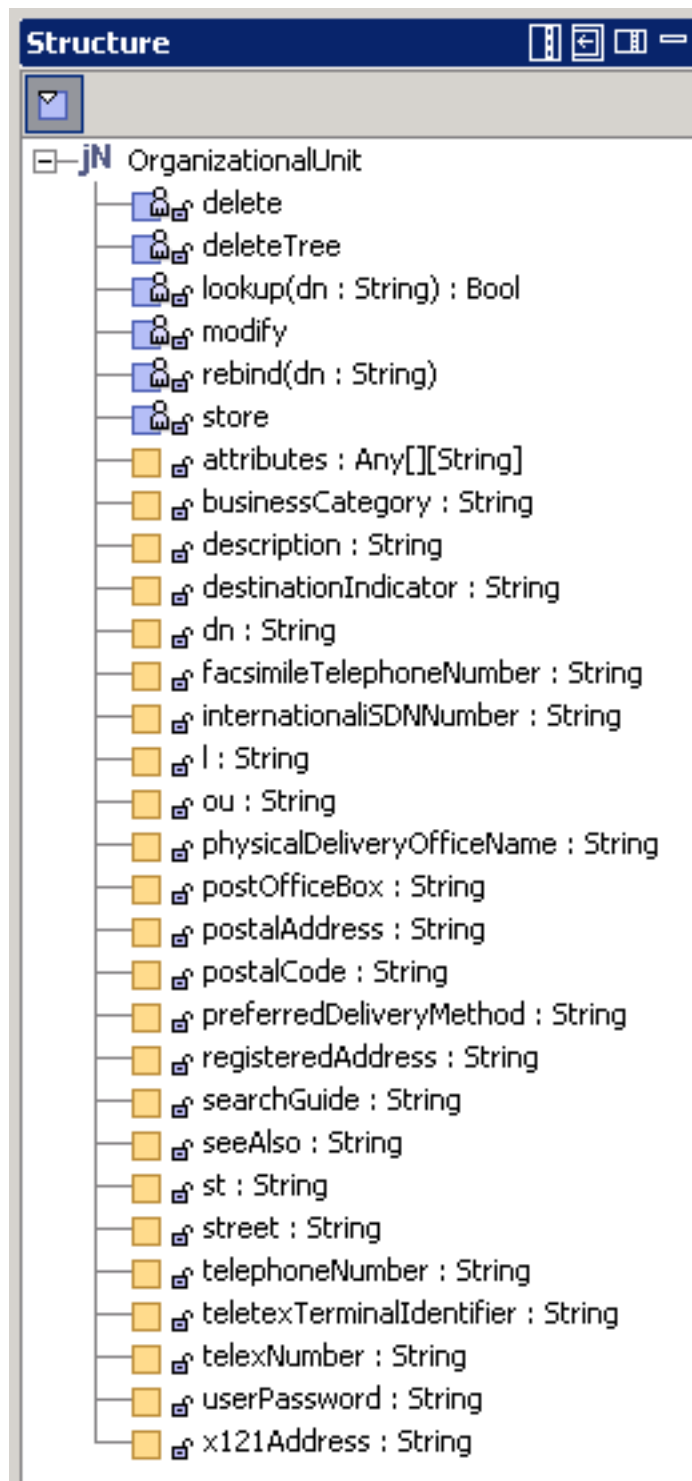
- Open the BP-method Editor by double-clicking on the activity.

Note



If the BP-method Editor does not open when you double-click an activity, you can change your preferences. Choose **File** and then **Preferences** from the menu options. Click the **Activity** tab. In the **On double click show:** field select sources from the drop-down menu. Click the **Ok** button. Next time you double-click an activity, the Editor appears.

- Click the **Component Browser** folder at the bottom of the BP-method Editor window. Your new module should appear on the list.



- Drill down from your new module to see your JNDI component. Drill down from your component to see all the methods you chose

to include.

- Complete your Method with the JNDI component usage, methods, variables, as needed.
- Save and check your method.

BP-Methods Examples

Common where sentences to search within the Directory Service

The first example uses a common *where* statement to search for activities whose activity name is "End". The second example compares directory service entries to two conditions set in the method. The participant must be named "Robert" AND must belong to the "Documentation" Organizational Unit to be considered a match.

```
//common where  
  
for each ad in activitydefinition  
  where activityname = "End"  
  do  
    display "This process has an End activity: " + ad.cn  
  end
```

```
//Operand equivalence and  
for each hp in humanparticipant  
  where hp.ou = "Documentation" and hp.cn = "Robert"  
  do  
    display "The participant: " + hp.cn  
  end
```

Search in the Directory Service by dn

A search by *dn* cannot be done using the *for each* statement, as it is like the primary key of an entry in the Directory Service. To search by *dn* you must use the *lookup* statement, as shown below.

```
data = "c=Argentina"
result = lookup(country, dn : data)

if (result) then
    delete country
end
```

Store a new Item in the Directory Service

This fourth example stores a new item in the directory.

```
o1 = OrganizationalUnit()
o1.ou = "test21"
o1.description = "foo1"
o1.dn = "ou=test21"
o1.postalCode = "1001"
o1.store()
```

Note



When you run JNDI component calls in the Method Debugger, a set of false information is returned. Actual information stored in directory services is not available until runtime, when the process is published and deployed.

SQL Components

The FuegoBPM Studio Project Catalog can introspect SQL tables, which can then be called from Method scripts to write business rules in your process design or Fuego Object methods. In FuegoBPM Studio, when using SQL componets, the operation is in auto-commit mode. This means that the SQL operation is committed although the Method is not successful.

Examples

```
do

firstName = "John"
lastName = "Doe"

description = "User's name is: " + lastName + ", "
              + firstName

INSERT INTO TABLE1 VALUES(firstName, lastName)

on SQLException
action = FAIL
```

Adding a SQL component

Use the following procedure to add a SQL database to the catalog. The procedure shown in the steps below uses an Oracle database as the example. FuegoBPM supports Oracle, MS SQL, Cloudscape, Informix, Sybase ASE, and IBM DB2.

Step 1: Catalog the JDBC driver as an external resource of your project

To be able to catalog a database component, you must first catalog the JDBC driver that corresponds to the database as a JCL external resource. For complete instructions, please refer to External Resources Configuration - Java Class Libraries.

Note



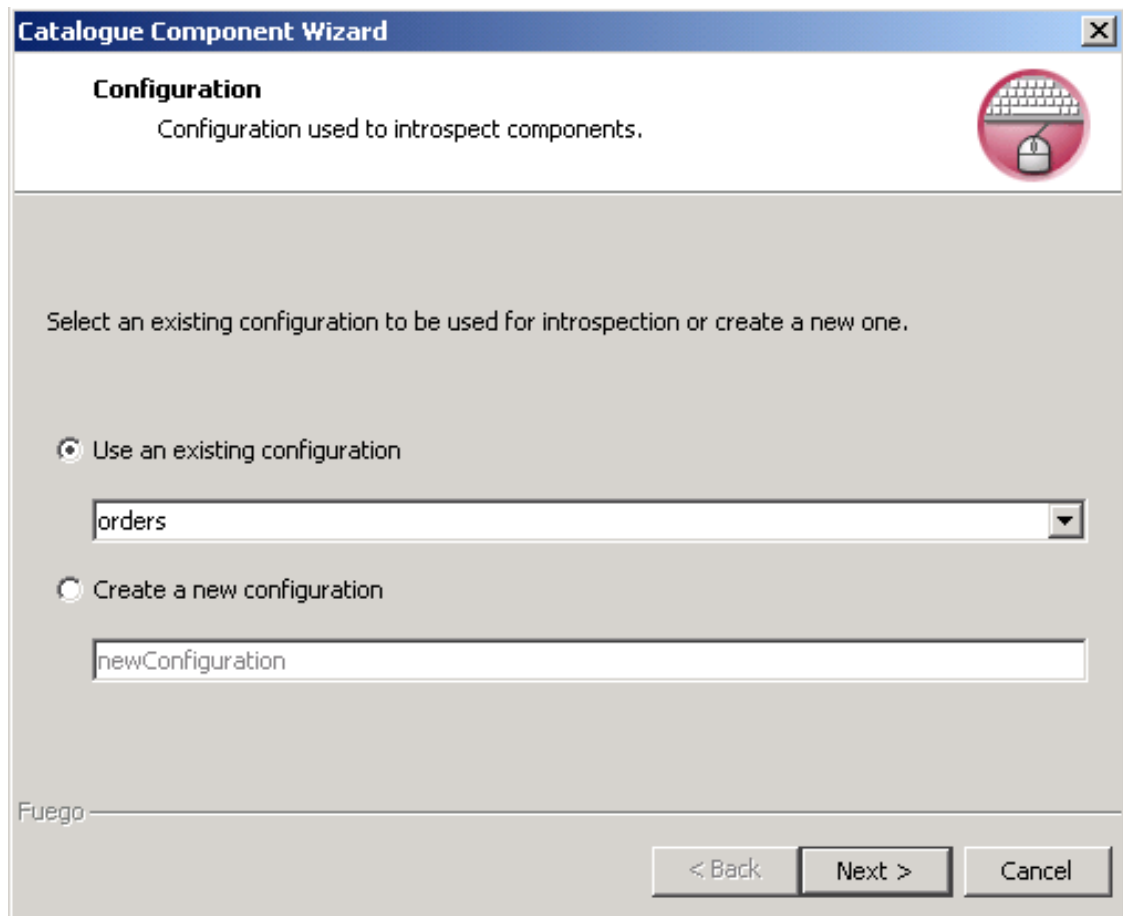
Be sure that the JDBC driver is catalogued as a **Non-versionable** JCL external resource.

Step 2: Catalog the database component

To add a database component

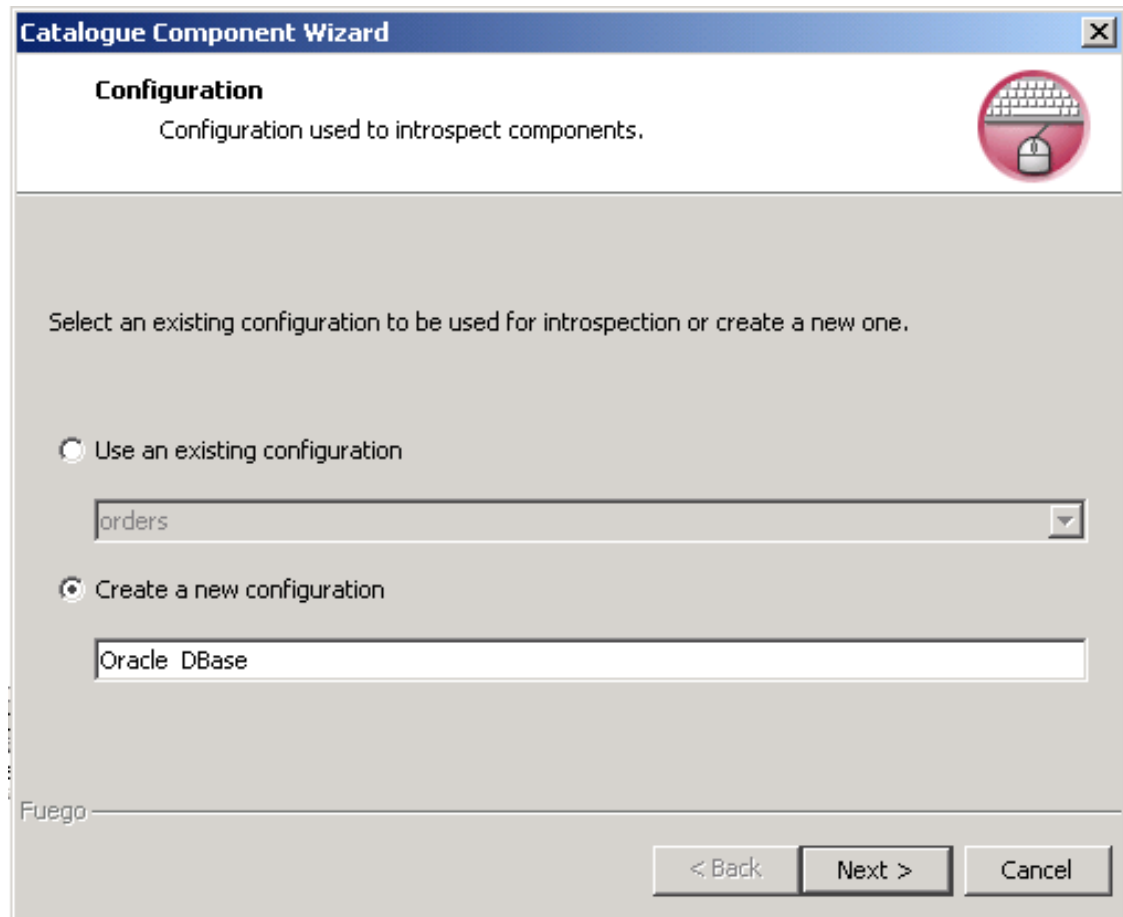
1- Add a new module to hold your database component or right-click on an existing module in the left panel of the Component Manager window and select **Catalogue Component** and then **sql** from the shortcut menu.

The following wizard screen appears.



The screenshot shows a window titled "Catalogue Component Wizard" with a close button in the top right corner. The window has a blue header bar. Below the header, the title "Configuration" is displayed in bold, followed by the subtitle "Configuration used to introspect components." in a smaller font. To the right of the subtitle is a circular icon containing a keyboard and a mouse. The main area of the window is light gray and contains the instruction "Select an existing configuration to be used for introspection or create a new one." Below this instruction are two radio button options. The first option, "Use an existing configuration", is selected with a filled radio button. Below it is a text box containing the word "orders" and a small downward arrow on the right. The second option, "Create a new configuration", is unselected with an empty radio button. Below it is a text box containing the text "newConfiguration". At the bottom left of the window, the word "Fuego" is followed by a horizontal line. At the bottom right, there are three buttons: "< Back", "Next >", and "Cancel".

Use an existing configuration or create a new one.



The screenshot shows a window titled "Catalogue Component Wizard" with a close button (X) in the top right corner. Below the title bar, the word "Configuration" is displayed in bold, followed by the text "Configuration used to introspect components." To the right of this text is a red circular icon containing a white keyboard and a mouse. The main area of the window has a light gray background and contains the instruction "Select an existing configuration to be used for introspection or create a new one." Below this instruction are two radio button options. The first option, "Use an existing configuration", is unselected. Below it is a text box containing the word "orders" and a small downward arrow on the right. The second option, "Create a new configuration", is selected with a filled radio button. Below it is a text box containing the text "Oracle DBase". At the bottom left of the window, the word "Fuego" is visible. At the bottom right, there are three buttons: "< Back", "Next >", and "Cancel".

2- Type a name for the new configuration and click **Next**.

Catalogue Component Wizard

New Configuration
Complete the information required for the new configuration

Supported types: Oracle DataBase

Details

Host: aragorn Port: 1521

User: system Password: *****

SID: strider

Schema: development

Driver Type: thin

☐ Advanced

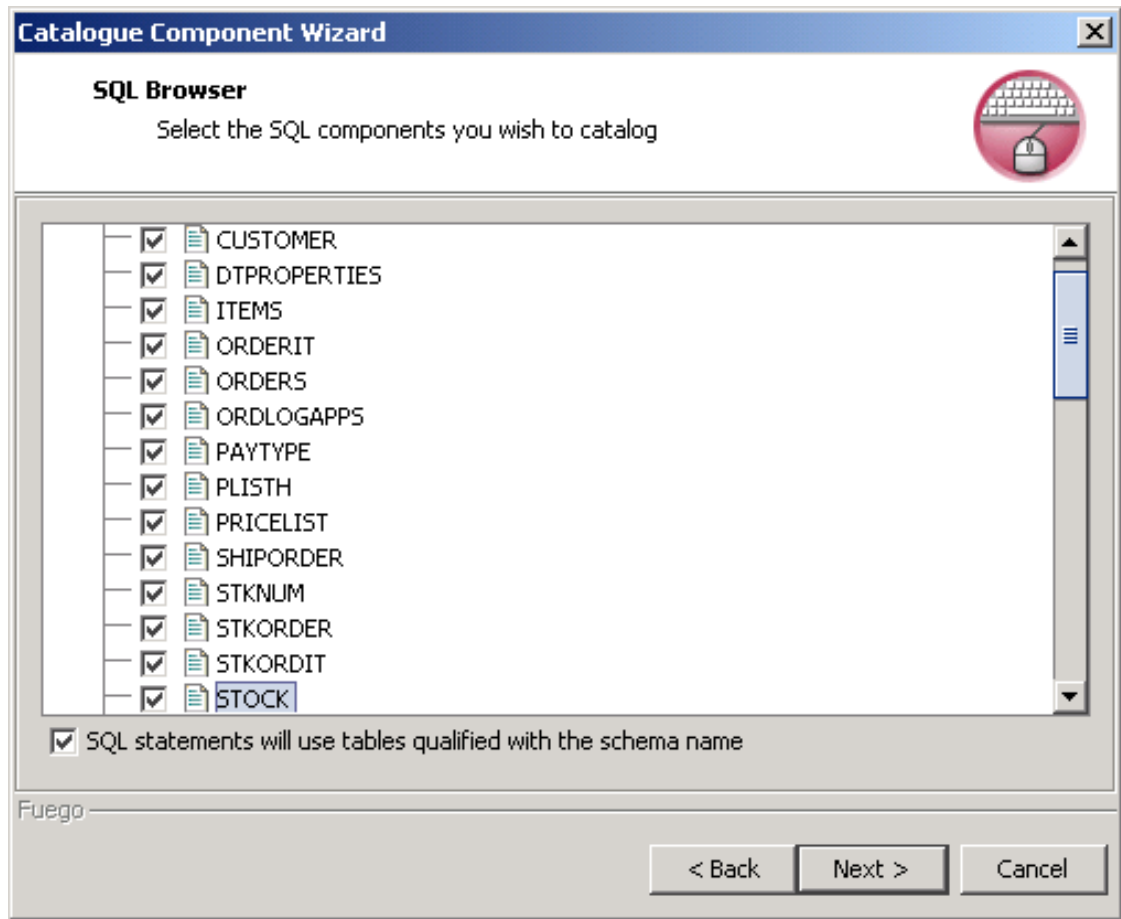
URL: jdbc:oracle:thin:@aragorn:1521:strider

Basic | Advanced | Runtime

Fuego


< Back Next > Cancel

3- Select the database type from the **Database type:** drop-down menu. Oracle is selected in the following example. Fill in the properties.



Select the *SQL statements will use* checkbox when you want to use the schema name to qualify the tables in the sql statements.

Note

 If your BP-method will alter the table by inserting or deleting information, you need to supply a user name and password that have administrator rights to the database.

- **Oracle**

- Type the host name of the server or computer where Oracle resides in the **Host** field.
- Type a port number in the **Port** field or use the default port.

- Type the server ID (also called the SID) in the **Oracle Server Id** field. This is the alias for the Oracle database and can be obtained from your Oracle Database Administrator. This field is not required for other database types. Select the correct database type below for information on such database.
- Schema, if you give a name of a schema, the configuration and introspection will only work on tables of that schema. If you don't give a schema name the schema could be changed in runtime. That is for example a table referenced like *devel.invoice* in your development environment. If in production you had a different schema name, suppose *production* references to *devel.invoice* would work only if you didn't give a name to the schema.
- **Progress** Progress databases require the database name in the **Database** field.
- **Cloudscape** Cloudscape databases require the database name in the **Database** field and the correct schema in the **Schema** field.
- **Informix** Informix databases require the database name in the **Database** field and the server name in the **INFORMIX_SERVER** field. If the schema name has capital letters, you have to uncheck the "SQL statements will use tables qualified with the schema name" checkbox in the "Select the SQL components you wish to catalog" wizard step.

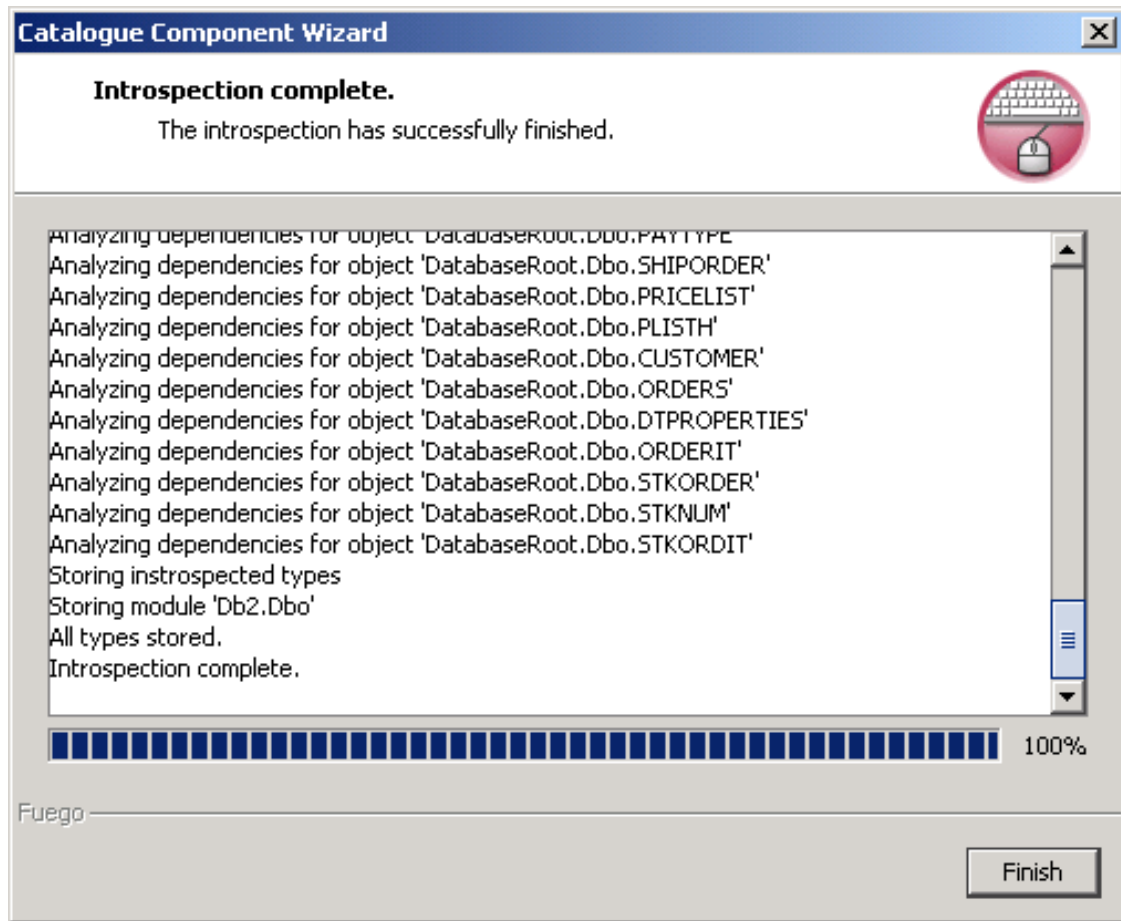
- **Sybase ASE** Sybase ASE databases require the database name in the **Database** field.
- **MS SQL** MS SQL databases require the database name in the **Database** field. **IMPORTANT:** if you are not able to connect, please check with your DBA that the selected port is correct and does support JDBC connections.
- **DB2** IBM DB2 databases require the database name in the **Database** field. **Important Note when using DB2 type 2:** If you are introspecting SQL tables using the DB2 Type2 JDBC driver and this one is registered as an external java library the error *"java.sql.SQLException: No suitable driver"* may be thrown by the DB2 JDBC Driver. The problem manifests after publishing and deploying the project with the Studio embedded Server running. Before publishing and deploying, the connection with DB2 works ok, but once the project is deployed you will start getting this error message. A workaround to prevent this from happening is to stop and start the Studio Embedded Server. The way to prevent this error from happening is to remove the *db2java.zip* from the External libraries, recreate the Studio cache (delete the directory *\$STUDIO/cache* after removing the *db2java.zip* from the project) and copy this file into the JRE's ext directory being used by the FuegoBPM Studio. Most likely you will be using the Studio Embedded JVM that is located inside the Studio installation directory. So you would need to copy the *db2java.zip* file into *\$STUDIO/jre/lib/ext*.

Note



DB2 tables that contain field name *columns* cannot be cataloged.

4- Click **Next**.



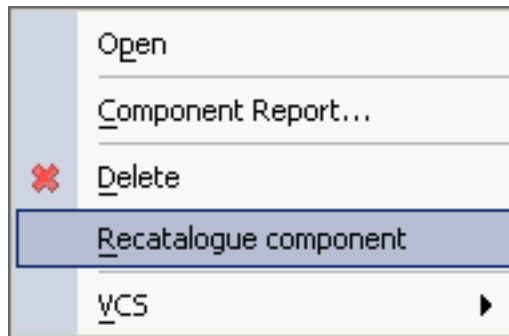
5- Select the Schema/Tables/Procedures you want to catalog and click **Next**.


6- Click **Finish** to close the wizard.

Step 3: Recataloging and adding new tables.

If, for any reason, the tables that you had already cataloged have changed, you can reintrospect them.

Right-click on the schema name created when you first cataloged the component and select the option *Recatalogue component*. By doing this, only the selected tables will be reintrospected or introspected for the first time and added to your Project Catalog.

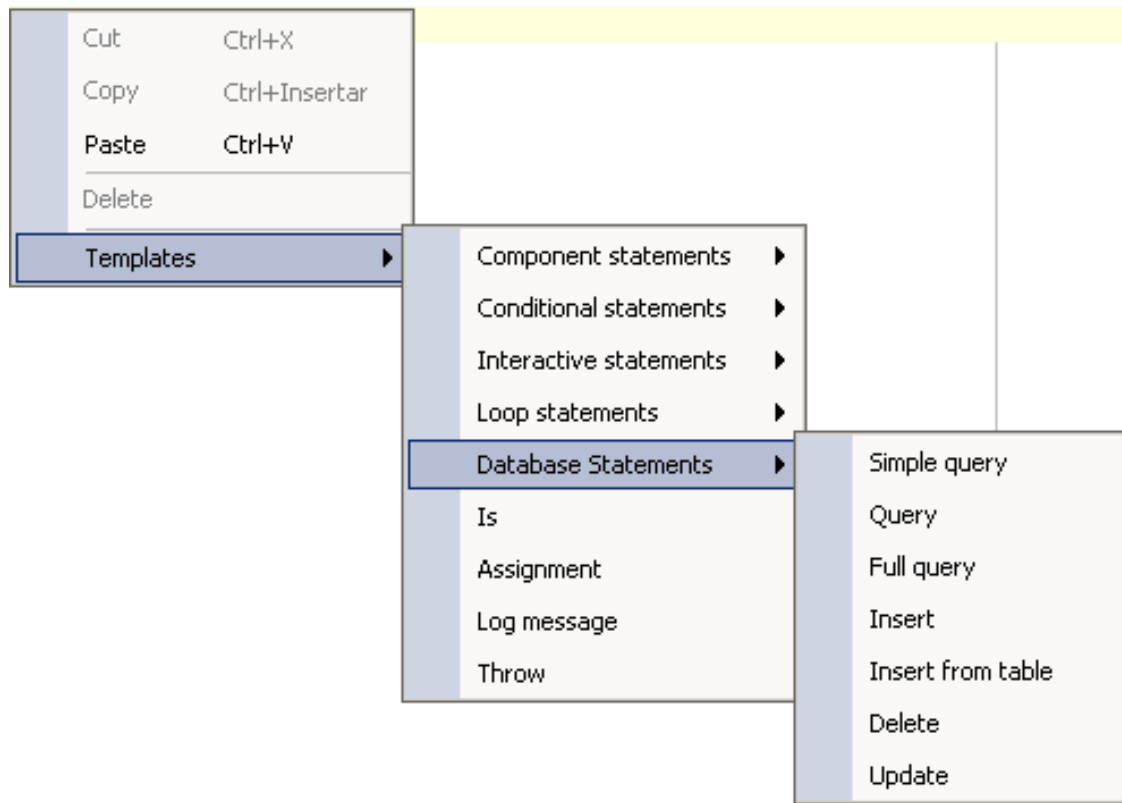
**Note**

 If you are cataloging tables from a MySQL database, the module created to hold the sql components will be named *ConfignameSchema*, since MySQL does not handle the *schema* concept.

Step 4: Use the component in a process

Once you have added a database component to the Project Catalog, you can call and make changes to the database in any of the Business Process Methods or Object Methods of your design.

To call the component from a Business Process Method activity's task method Editor, double-click on the activity task to which you want to add the usage of the SQL component. You can use the Methods template by right-clicking on the Methods Script and selecting the query you need.



1. The query template will paste into the Method scripting panel.
2. Make the required changes.
3. Save and check your Method. Correct errors, if any. Run the Method debugger to check your work.
4. Save your process.

Export the schema structure

The introspected database schema structure can be exported. To do so,

1. Right-click on the schema icon of the introspected SQL components, and select the *Export Schema Structure* option.

2. Browse the directory where the sql file will be stored and click the *Save* button.

The *store* method

insert or update

The *store* method determines automatically if an *update* or an *insert* has to be performed. It attempts to perform the *update* and no record is updated then executes the *insert*. It is depending on the component status that executes the *insert* or *update* first. For example, if the sql component was loaded from the database executing the *load* method, then it executes the *update* first, and the *insert* later. But, if the component was instantiated from a BP_Method and the fields were manually set, the *store* executes the *insert* first and then the *update*.

Server and Client side

If a *store* is performed on the client side, it will fail, as it cannot be execute on the client side.

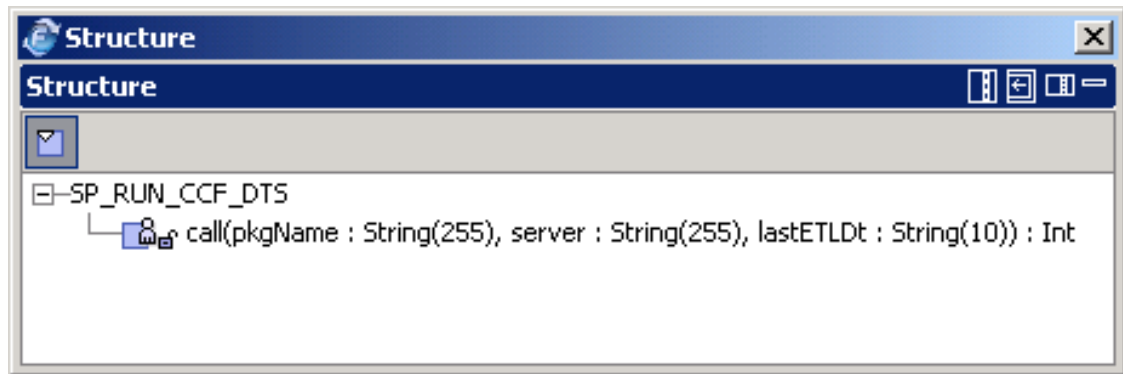
How the store methods work for updates?

All the component fields are updated. There is not *dynamic update* for the changed fields to avoid affect the

statements cache.

Cataloguing a Stored Procedure

To catalog a stored procedure, you must follow the same steps detailed for a sql table. When you introspect a stored procedure, it is catalog under the module and you will visualize it arguments and returned value like shown in the following picture:



Methods Examples

The following are some basic database Method examples:

Query

```
for each { variable } in { table }
where { condition }
do
{ body }
end
```

Insert

```
INSERT INTO { table }({ column1, column2, ... })
VALUES({ val1, val2, ... });
```

Delete

```
DELETE FROM { table }
WHERE { condition };
```

Update

```
UPDATE { table }  
SET { column1 = value1, column2 = value2, ... }  
WHERE { condition };
```

Exists

```
select Name, Skill from EMP where EXISTS  
    select * from EMP where EMP.Name = empName  
group by Name  
having COUNT(Skill) > 1);
```

Furthermore, if the Table has a primary key, the following methods are available:

- `load()` : Loads the record if the primary key is set. Returns a boolean indicating if the record was found.
- `store()` : Inserts or updates the record in the database. The primary key must be set.
- `delete()` : Deletes the record in the database. The primary key must be set.

SQL Components as instance variables

You can use SQL components cataloged in your Project catalog as instance variables. This is very useful when you want to persist data attached to a process instance. You do not have to store each

database value as separate instance variables.

Working with SQL components as Instance Variables

The configuration used to access the database is the one used when the sql component was cataloged.

Actions automatically performed

The *load* action is automatically performed, so you do not have to worry about retrieving the information. The *load* is performed in a lazy way.

Actions that have to be manually performed

You have to manually do the *store* and *remove* of the sql component.

- *store*--you should call the store method if you are adding a new item to the database or if you have modified it during processing. *store* is "intelligent" and will execute an *insert* or *update* depending on the situation.
- *remove* if you need to delete the item from the database. It is not possible to automatically determine this situation.

SQL as instance variable and *accessDatabase*

When a process instance variable is "persisted" (when the task is completed) all their fields are stored into the database. However, SQL objects are a bit more intelligent and decide which fields have to be stored depending on the value of the "accessDatabase" attribute.

If "accessDatabase" is set to true, it assumes that the object is already stored in a database and therefore it only stores its primary key. And if it is false the whole object is stored.

However, these rules do not apply for Screenflow because their variables are not persisted between tasks. Everything works as you are running only one task.

Benefits

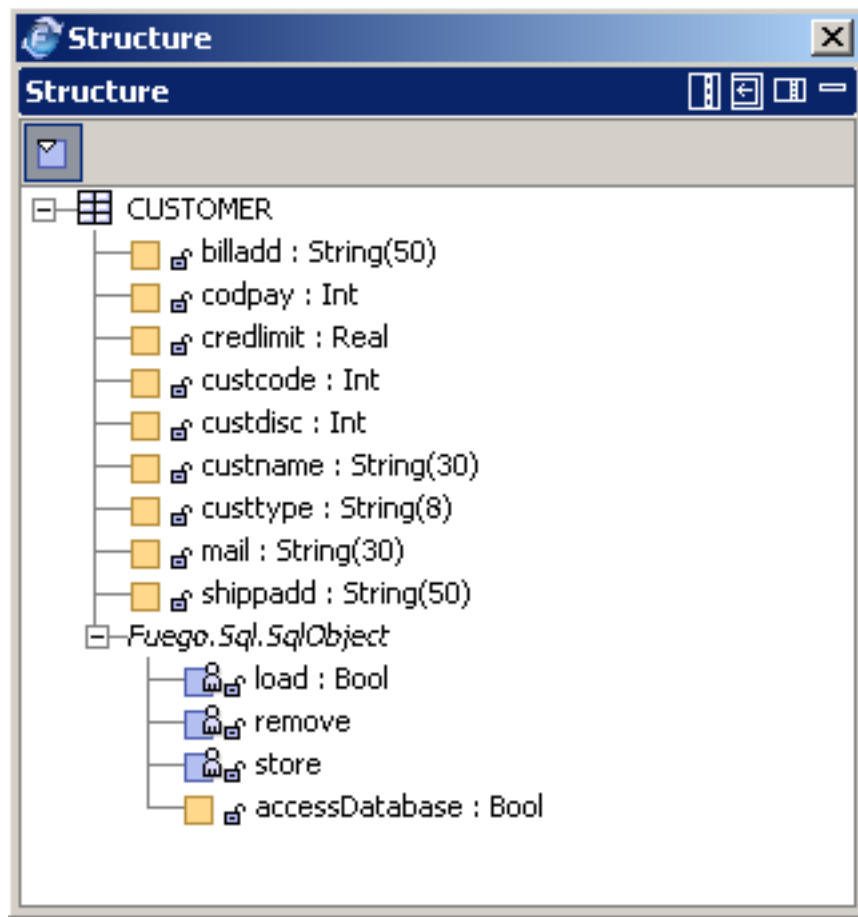
Using a SQL component as an instance variable is a practice that helps you optimize the server database as the only data stored regarding the SQL Component is its primary key. This reduces the storage usage as you do not need to store the data that you are interested in attaching to the process instance in different instance variables. For example, if you want to store the customer's id, its credit limit and discount percentage, you will have to define three instance variables. Instead, if you store the sql component, the only data stored would be the customer id which is the primary key of the identity customer.

Using the SQL component as an instance variable brings the additional benefit of having the data always updated. As you do not keep the information separately from the database, the information may be changed by other users and next time you access the data, it will be updated. If you store information, such as the credit limit of a customer in an instance variable and it is changed, you may be processing with outdated information and you must write extra code to check these changes. Using the SQL component as an instance variable, you always have the latest updated information.

Another important benefit is that the load is automatically performed. More than that, it is done in a *lazy* way. This means that the data is not retrieved until it is required. The *load* is performed the first time an attribute of the component is referenced.

Configuring SQL Components with no database access

SQL introspected components have an extra predefined bool field named *accessDatabase*. You can visualize it by expanding the *Fuego.Sql.SqlObject* entry in the structure panel of the SQL component.



The *accessDatabase* default value is set to **true**. If the value is set to *false*, the object is considered as a simple object structure. In this case, any setting value or getting value actions performed on the object do not have effect on the database, but on the object structure.

The code shown below does not interact with the database:

```
c = CUSTOMER()  
  
c.accessDatabase = false  
// or c = CUSTOMER(accessDatabase: false)  
  
c.custtype = "GLOBAL"  
  
.....
```

***accessDatabase* attribute and methods behaviour**

The SQL methods **load**, **store** and **remove**, force access to the database, by setting the default value to the *accessDatabase* field, that is *true*.

load

When the primary key of an SQL component is filled and another attribute that is not primary key is accessed, an automatic *load* is performed internally. If you want to disable it from working like that, set the *accessDatabase* attribute to false.

store

If the *accessDatabase* field is set to true, when the sql is serialized as a process instance, only the primary key is stored, but, all the sql component fields are store if the *accessDatabase* is set to false".

Note

When the object is serialized as a ProcessInstance variable having the *accessDatabase* field set to false not only the primary is store but also **all** the object fields are. See more information in SQL Components as instance variables.

SQL Queries

FuegoBPM provides out-of-the-box functionality for introspecting tables from relational databases resources (tables, views and store procedures). At times, though, it may be necessary to operate with complex views of those tables or generic SQL queries. As, in some cases, join among different tables are performed to retrieved information localized in different RDBMS tables. FuegoBPM Studio now allows defining these complex queries as a single resource to prevent replicating these complex queries in different areas of the product. These SQL queries can also be customized to receive arguments and make them reusable within a FuegoBPM Project.

SQL queries

It is worth mentioning that only SELECT-type queries are allowed and that the generated components will provide read-only access to the results.

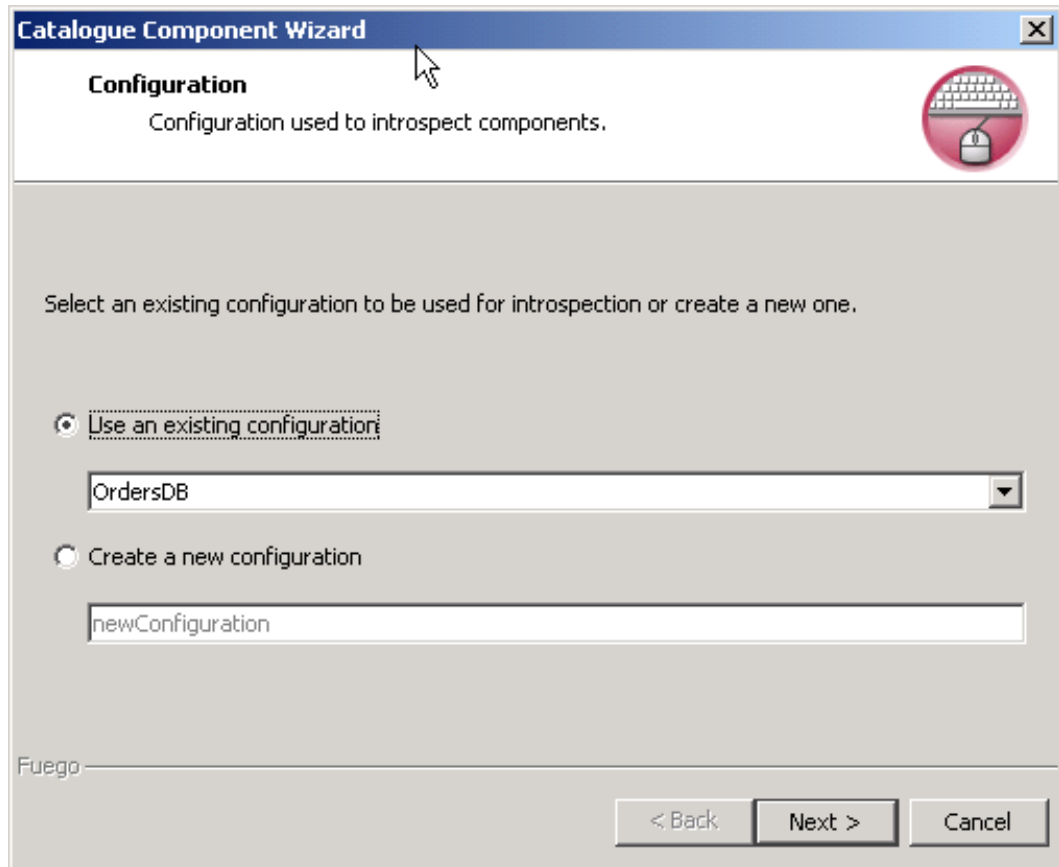
We further classify queries into two categories, named selfcontained and parametric queries. They differ only in that the latter allows the caller to pass parameters to the query at run-time.

Parametric queries, as the name suggests, allow the caller to specialize the query according to values that are determined at run-time, instead of when the query is designed and introspected. As an example, consider the a query, where we want to be able to specify the actual range for the amount column of the orders when the query is executed.

Cataloging SQL Queries

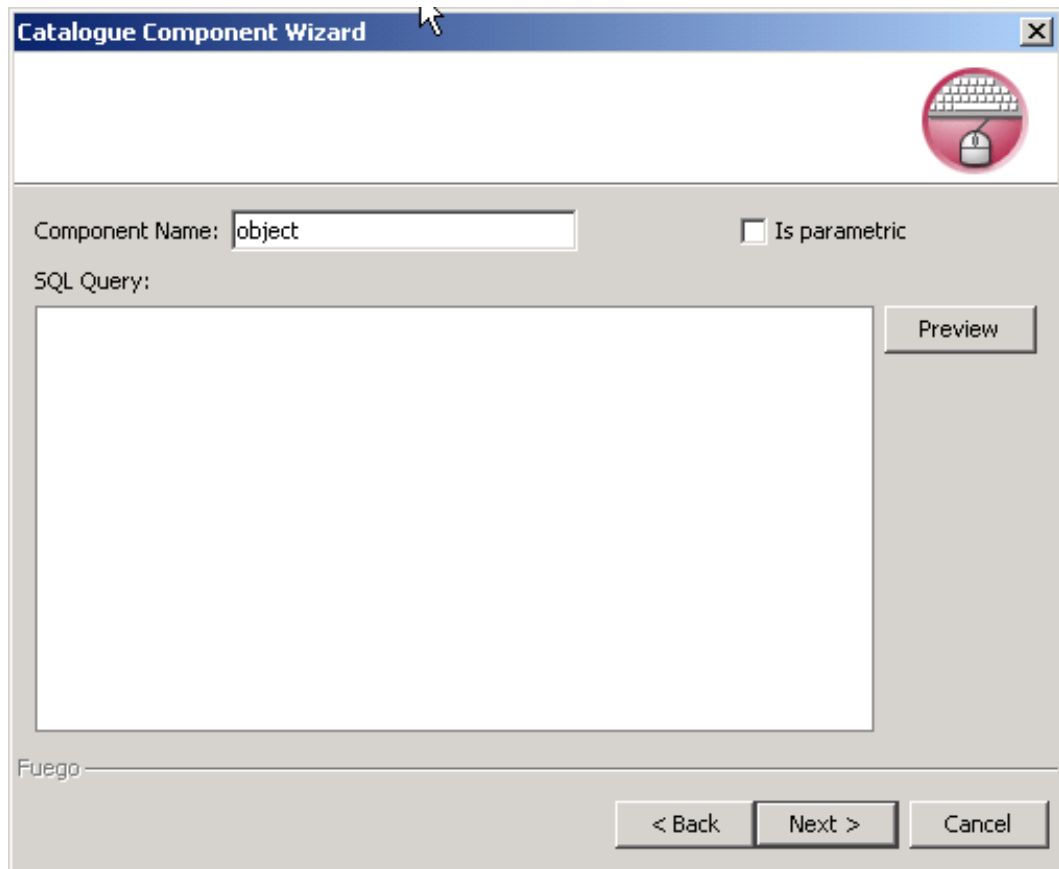
To catalog an SQL Query

1. Right-click on the module where the SQL Query will be stored. Select **Catalogue Component** and then **sql query** from the short cut menu.
2. The first step of the wizard, **Configuration** is displayed.



Select an existing configuration to be used to introspect by clicking **Use an existing configuration** and selecting the indicated one from the combo options, or define a new one by clicking the **Create a new configuration** check box, and click **Next** to continue. The configuration to use is the one defined as **SQL type** and is the one that defines the location and type of the database used to introspect the SQL query. Refer to External Resources - SQL Database for detail on how to configure it.

3. The next step of the wizard is the specification of the sql query you want to introspect.

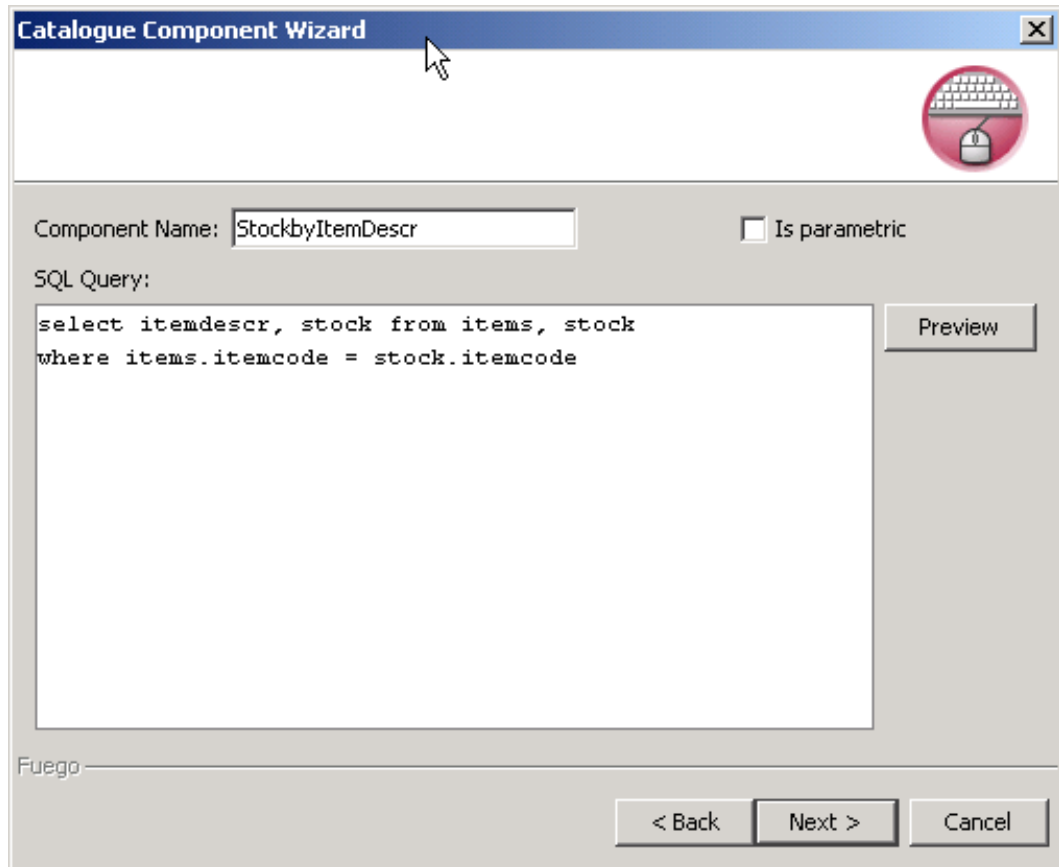


The screenshot shows a Windows-style dialog box titled "Catalogue Component Wizard". It has a blue title bar with a close button (X) in the top right corner. Below the title bar is a white area with a red circular icon containing a keyboard and mouse. The main area has a grey background. It contains a text field labeled "Component Name:" with the text "object" entered. To the right of this field is a checkbox labeled "Is parametric" which is currently unchecked. Below the text field is a large white text area labeled "SQL Query:". To the right of this area is a button labeled "Preview". At the bottom of the dialog, there is a "Fuego" logo on the left and three buttons: "< Back", "Next >", and "Cancel".

Write the name to give the resultant component and begin defining the SQL query to catalogue.

Selfcontained SQL Query

1. Type the query in the **SQL Query** area of the dialog. As the sql is not parametric, leave the *Is parametric* check box unselected.



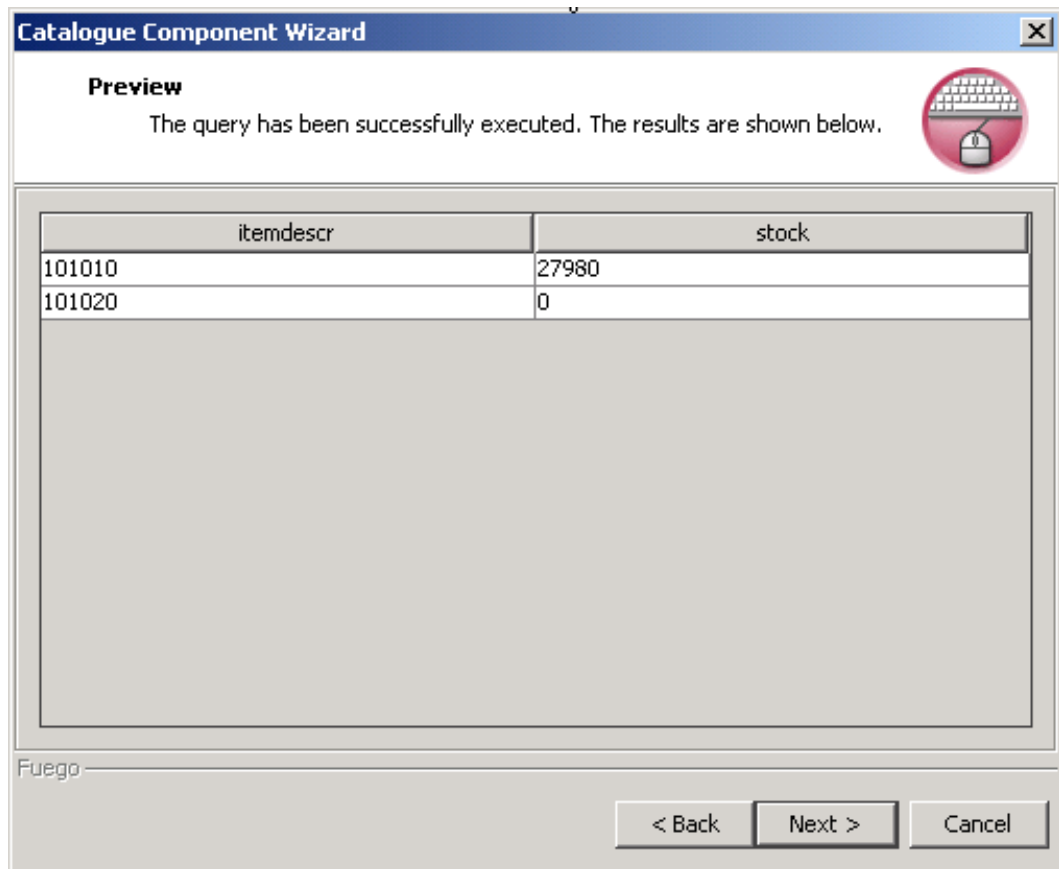
The screenshot shows a Windows-style dialog box titled "Catalogue Component Wizard". It has a blue title bar with a close button (X) in the top right corner. Below the title bar is a large empty rectangular area. To the right of this area is a circular icon containing a keyboard and a mouse. Below the large area is a section with a "Component Name:" label and a text box containing "StockbyItemDescr". To the right of this is a checkbox labeled "Is parametric" which is currently unchecked. Below the text box is a label "SQL Query:" followed by a large text area containing the following SQL query:

```
select itemdescr, stock from items, stock
where items.itemcode = stock.itemcode
```

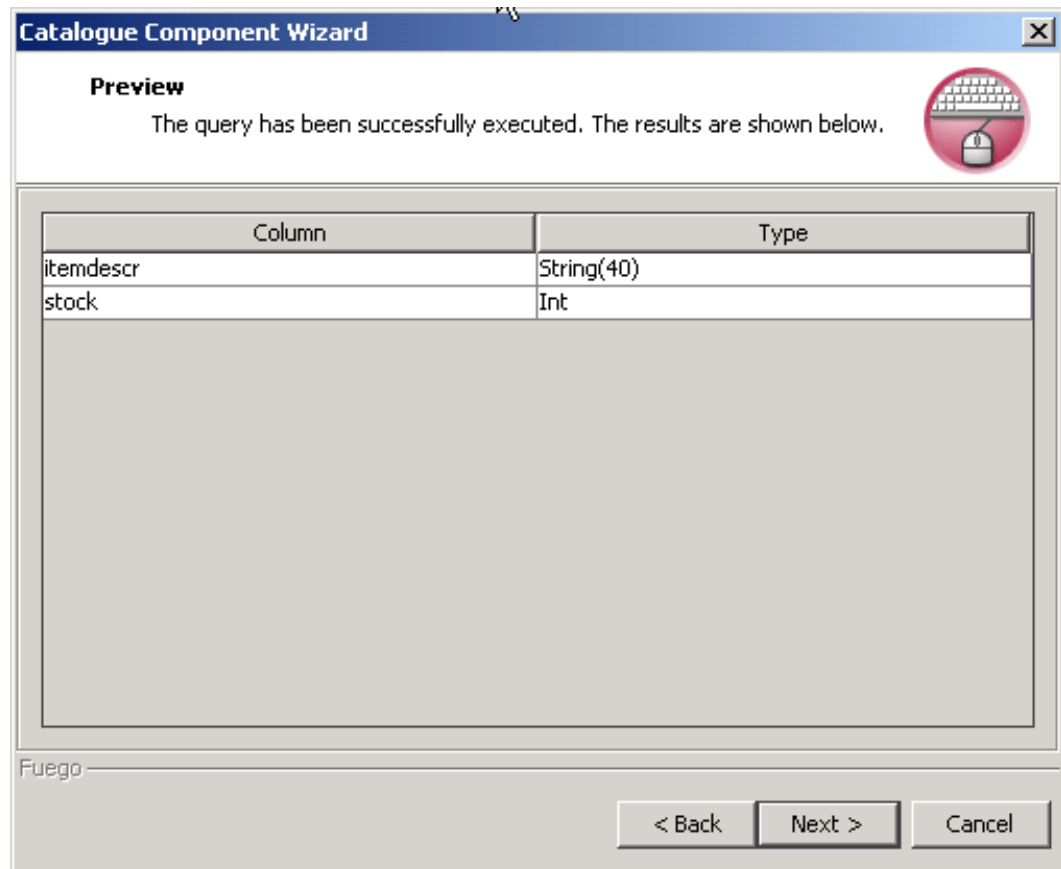
To the right of the SQL text area is a button labeled "Preview". At the bottom of the dialog box, there is a "Fuego" logo on the left and three buttons on the right: "< Back", "Next >", and "Cancel".

The query shown in the catalogation is a simple join between two tables.

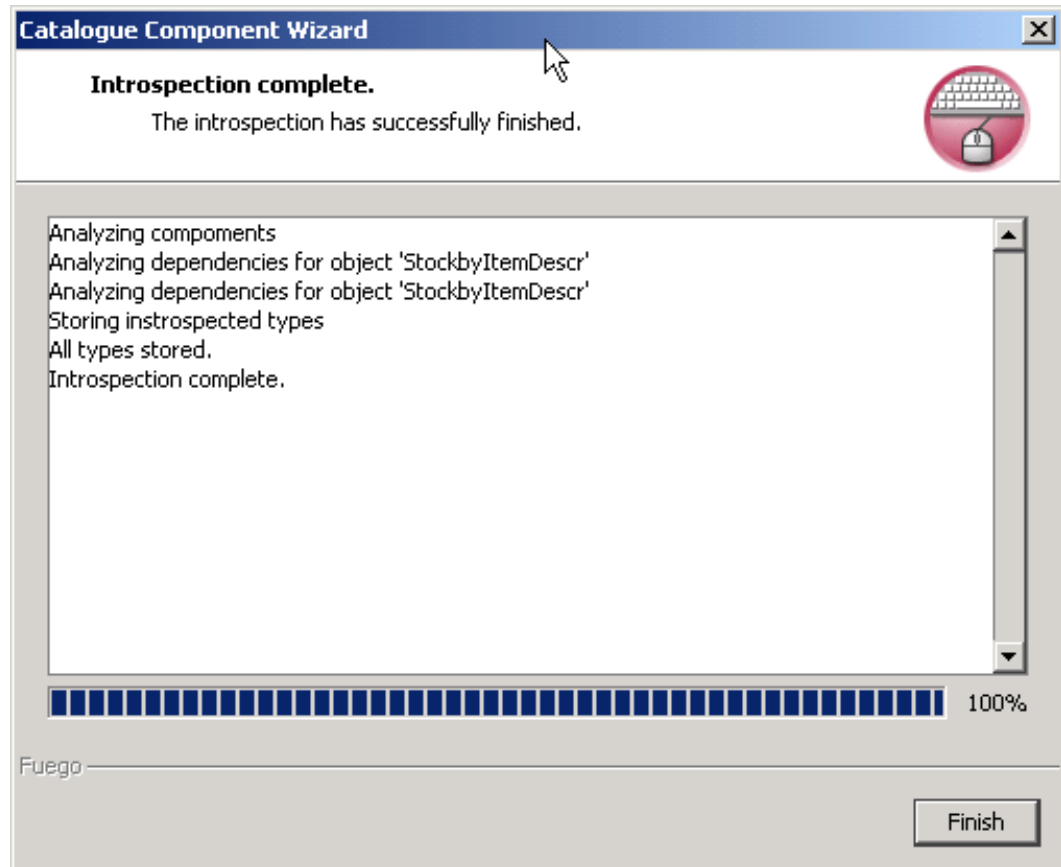
2. Click the **Preview** button to execute the sql, see which are the results and correct the sql if needed.



3. Click **Next**. The introspected resultant attributes are displayed, showing name and type for each one.



4. Click **Finish** when the introspection ends.



Parametric SQL Query.

1. Type the query in the **SQL Query** area of the dialog. As the sql is parametric, select the *Is parametric* check box. Parameters are define like: \$parame_name:type. The parameter type shoul be specified

as in the database, if possible. See list of possible values in the next paragraph. For example, if the value in the database correspondes to a VARCHAR then, define the parameter as \$param:VARCHAR, instead of defining it like \$param:string.

The following are the possible types for a sqlquery argument:

ARRAY, BIGINT, BINARY, BIT, BLOB, CHAR, CLOB, DATE, DECIMAL, DISTINCT, DOUBLE, FLOAT, INTEGER, JAVA_OBJECT, LONGVARBINARY, LONGVARCHAR, NULL, NUMERIC, OTHERREAL, REF, SMALLINT, STRUCT, TIME, TIMESTAMP, TINYINT, VARBINARY, VARCHAR, NVARCHAR, OTHER (OTHER is the default type when the type used is unknown.)

Catalogue Component Wizard

Component Name: ☒ Is parametric

SQL Query:

```
select stkorder.stktype, stkorder.stkdate,
       stkorder.status,
       stkordit.itemcode, sum(stkordit.qty)
from stkorder, stkordit
where stkorder.stktype = stkordit.stktype and
      stkorder.stkcode = stkordit.stkcode and
      stkorder.status = $status:VARCHAR
group by stkorder.stktype, stkorder.stkdate
```

Preview

Fuego

< Back Next > Cancel

The query shown in the catalogation is a join between two tables that summarizes the quantity of items in the stock orders by type and date for orders in a specific status which is the parameter in the example.

1. Click **Next** to enter the parameters value and execute a sql preview.

Catalogue Component Wizard

SQL Query

```
select stkorder.stktype, stkorder.stkdate,
       stkorder.status,
       stkordit.itemcode, sum(stkordit.qty)
from stkorder, stkordit
```

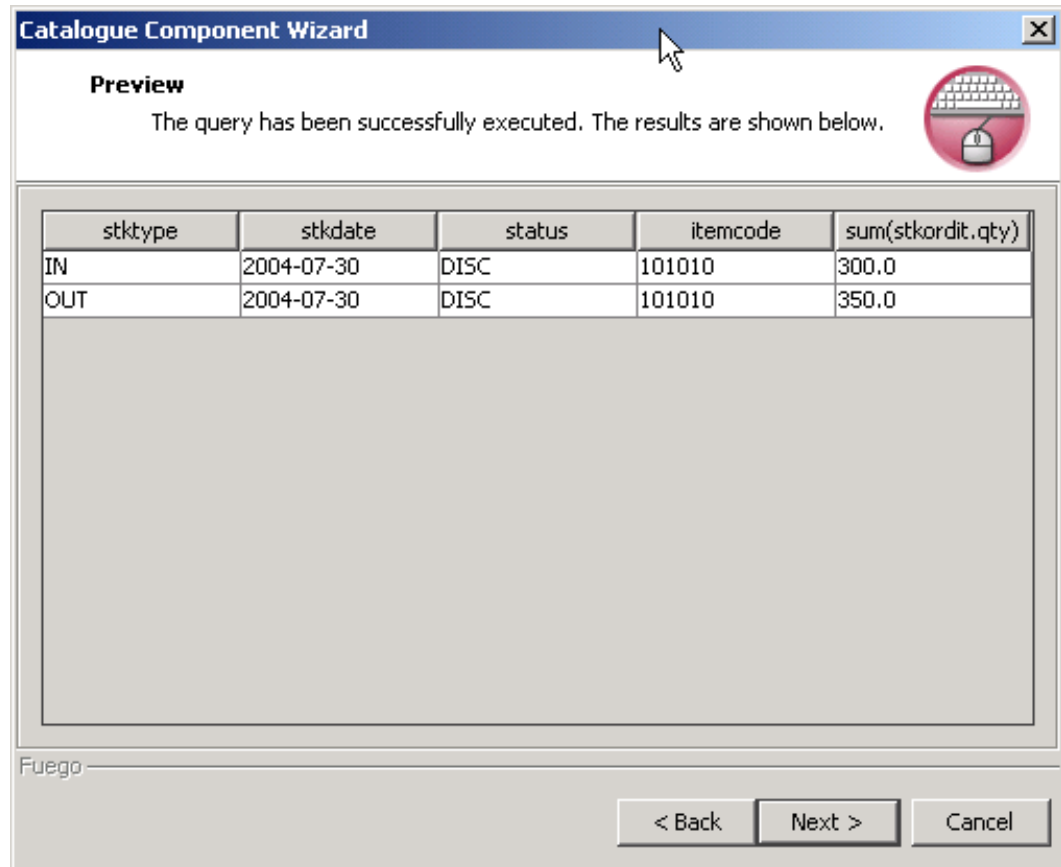
Parameters

Parameter		Value
status [VARCHAR]	=	DISC

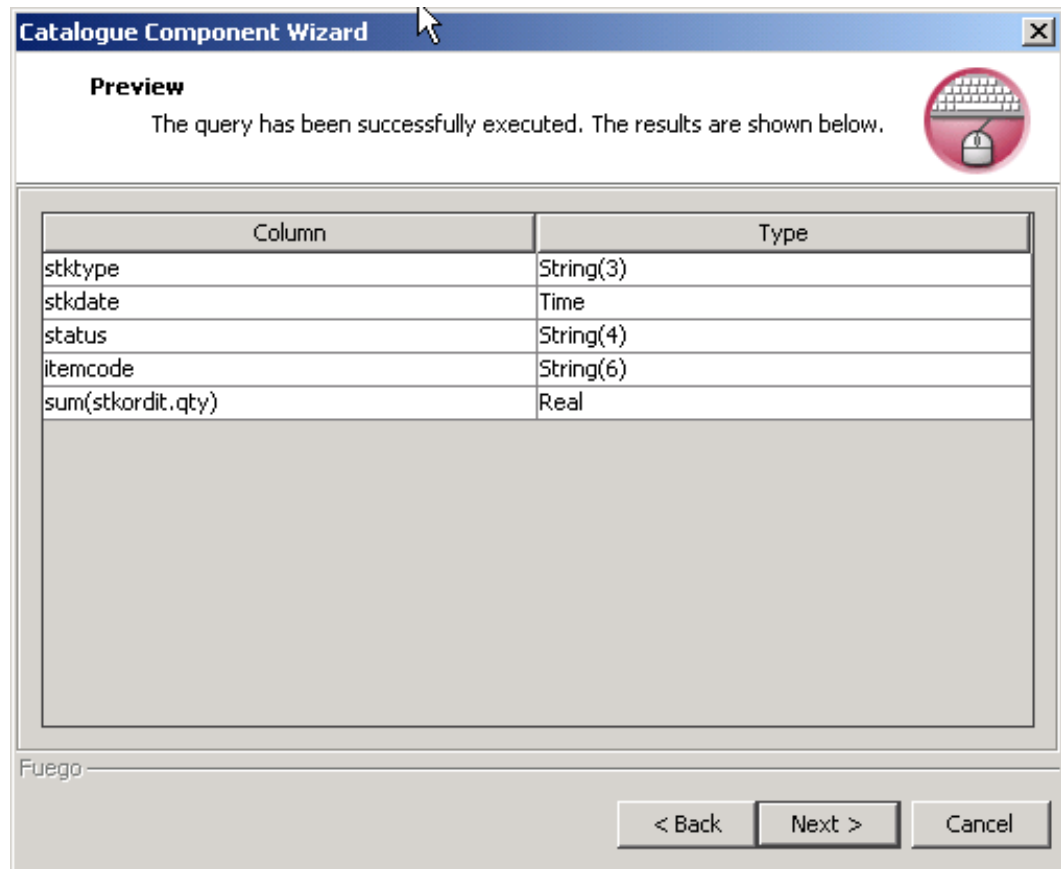
Fuego

< Back Next > Cancel

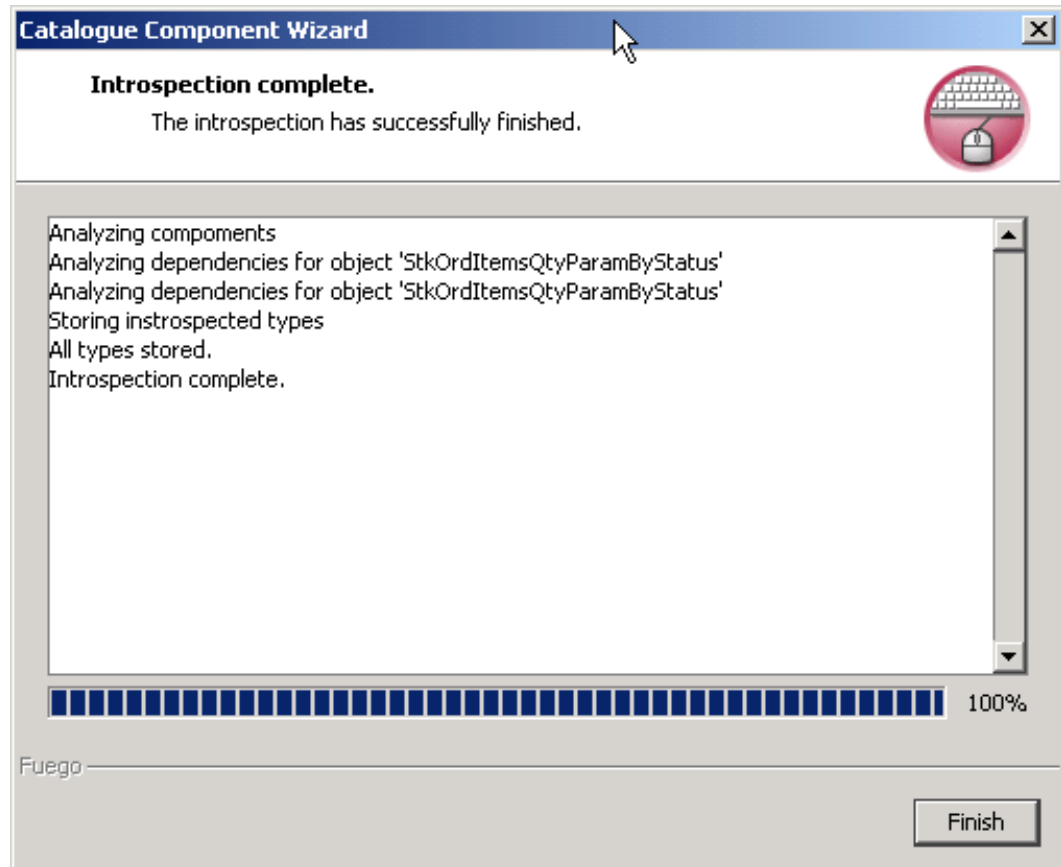
2. Click the **Preview** button to execute the sql, see which are the results and correct the sql if needed.



- Click **Next**. The introspected resultant attributes are displayed, showing name and type for each one.



- Click **Finish** when the introspection ends.



Recataloging SQL Queries

To recatalog an SQL Query

1. Right-click on the already cataloged SQL Query. Select **Recatalogue Component**.
2. Follow the steps detailed in section above.

Cataloging SQL Queries Examples

Cataloguing examples


Other examples using *group by* and *order by*. Having the *stkorder* and

stkordit tables with the following data:

Catalogue Component Wizard [X]

Preview

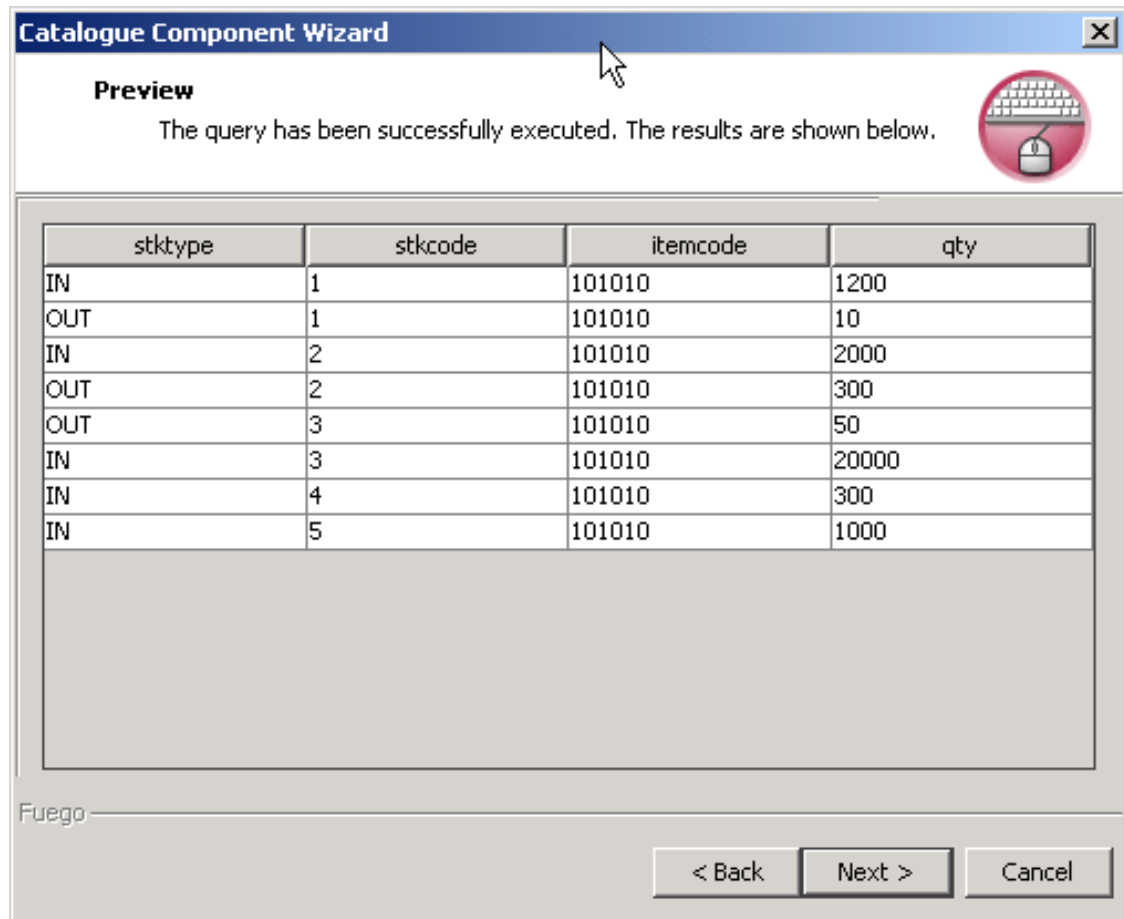
The query has been successfully executed. The results are shown below.



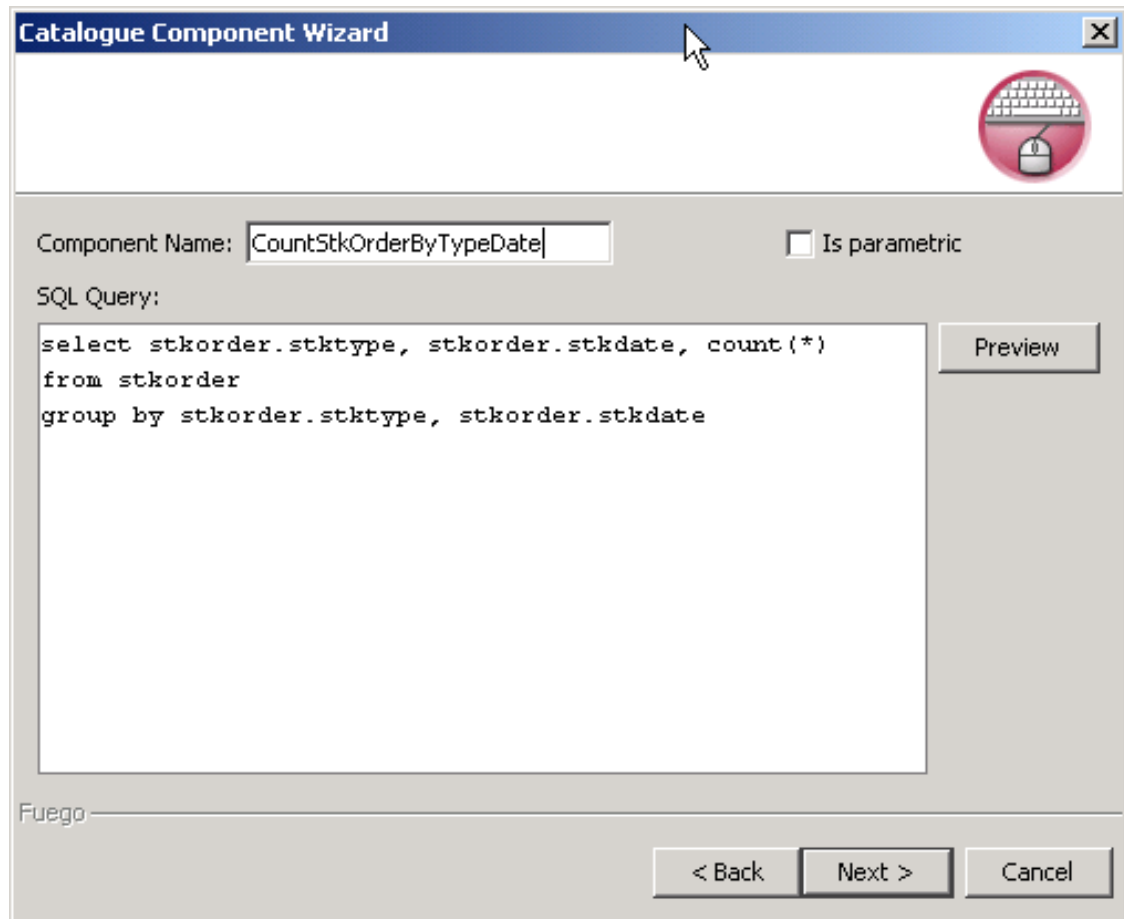
stktype	stkcode	stkdate	status
IN	1	2004-07-01	APP
OUT	1	2004-07-30	APP
IN	2	2004-07-30	APP
OUT	2	2004-07-30	DISC
OUT	3	2004-07-30	DISC
IN	3	2004-07-30	APP
IN	4	2004-07-30	DISC
IN	5	2004-07-30	

Fuego

< Back Next > Cancel



Counting the Stock Order by Type and Date



The image shows a 'Catalogue Component Wizard' dialog box. It has a title bar with the text 'Catalogue Component Wizard' and a close button. Below the title bar is a large empty rectangular area. To the right of this area is a circular icon containing a keyboard and a mouse. Below the large area, there is a 'Component Name' label followed by a text box containing 'CountStkOrderByTypeDate'. To the right of this is a checkbox labeled 'Is parametric' which is currently unchecked. Below the 'Component Name' section is an 'SQL Query' label followed by a large text area containing the following SQL query:

```
select stkorder.stktype, stkorder.stkdate, count (*)  
from stkorder  
group by stkorder.stktype, stkorder.stkdate
```

To the right of the SQL query text area is a 'Preview' button. At the bottom left of the dialog box is a 'Fuego' label followed by a horizontal line. At the bottom right are three buttons: '< Back', 'Next >', and 'Cancel'.

Component Name: ☐ Is parametric

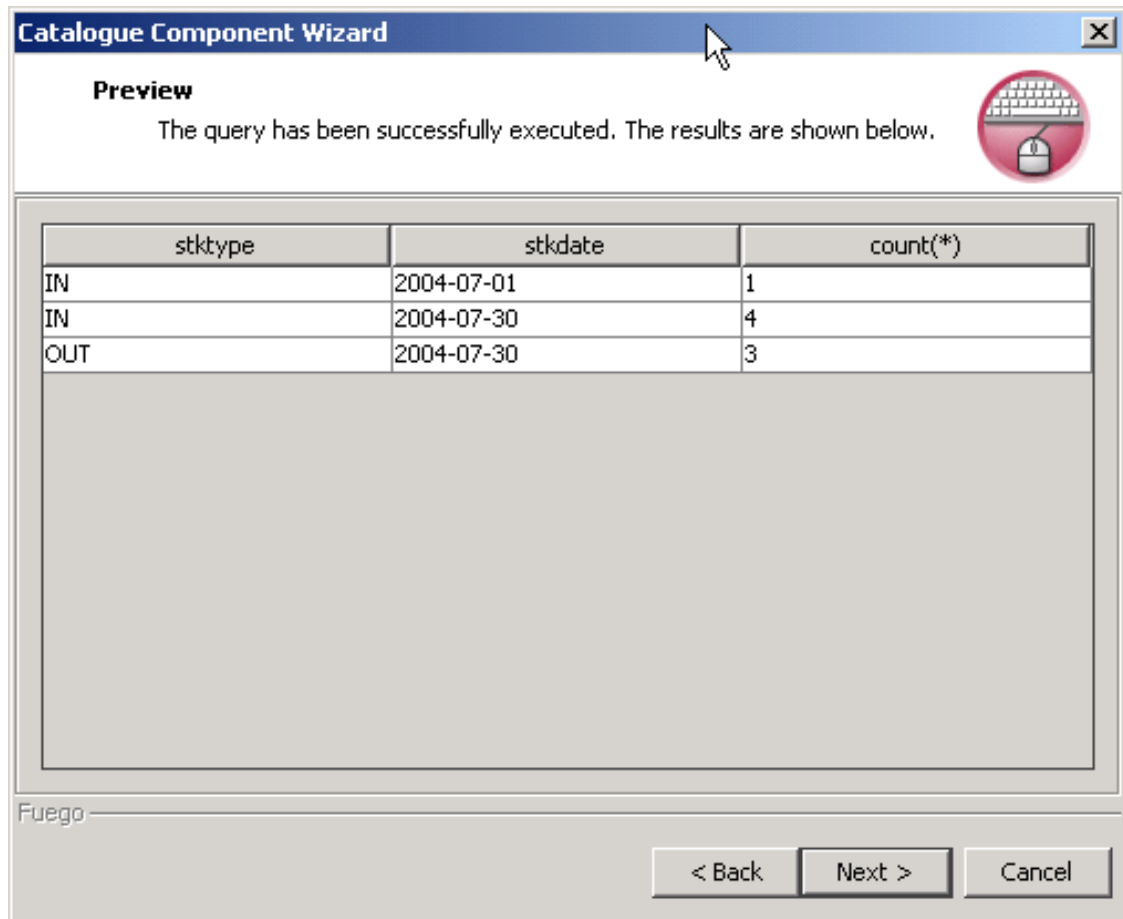
SQL Query:

```
select stkorder.stktype, stkorder.stkdate, count (*)  
from stkorder  
group by stkorder.stktype, stkorder.stkdate
```


Preview


Fuego

< Back Next > Cancel



Sumarizing the total quantity of items in Stock Orders by Type, Date and Status

Catalogue Component Wizard 

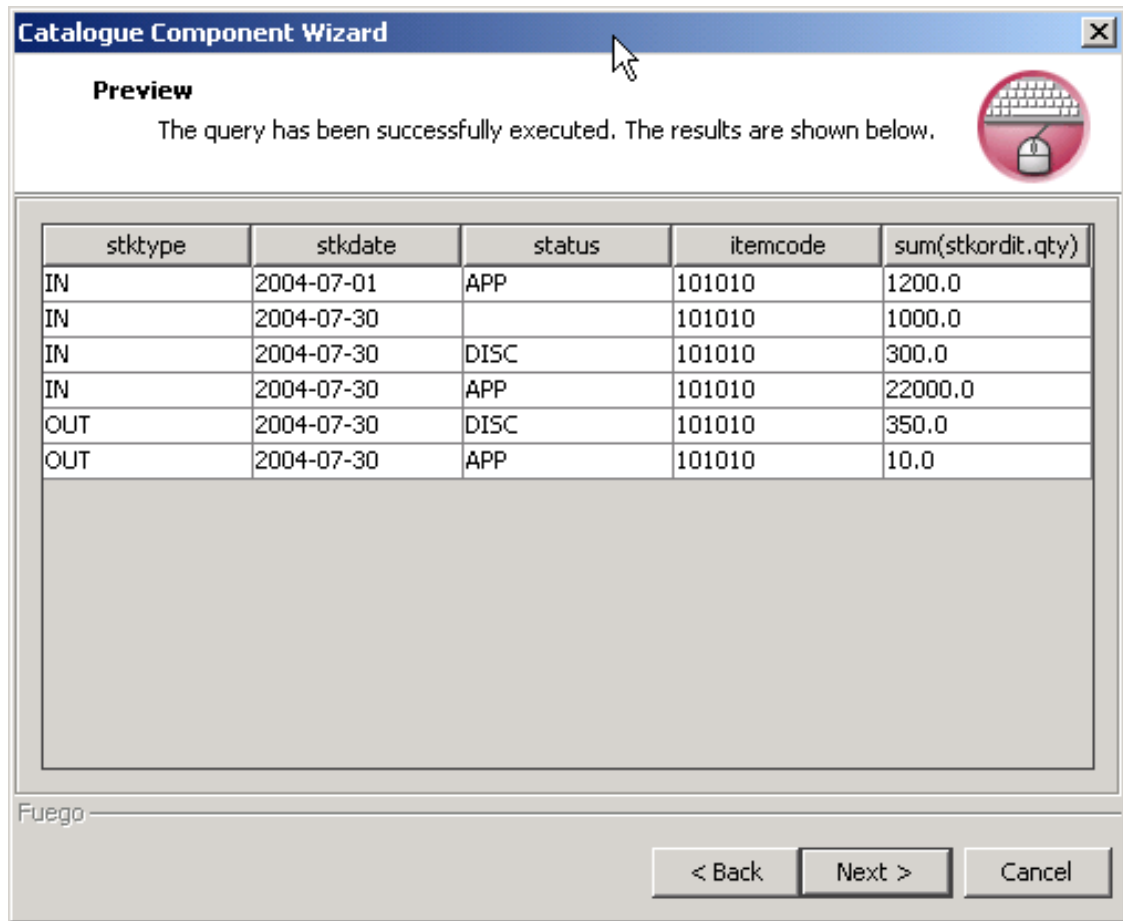


Component Name: ☐ Is parametric


SQL Query:


```
select stkorder.stktype, stkorder.stkdate,  
       stkorder.status,  
       stkordit.itemcode, sum(stkordit.qty)  
from stkorder, stkordit  
where stkorder.stktype = stkordit.stktype and  
       stkorder.stkcode = stkordit.stkcode  
group by stkorder.stktype, stkorder.stkdate,  
         stkorder.status
```

Fuego



Sumarizing the total quantity of items in Stock Orders by Type and Date


Catalogue Component Wizard 



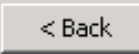
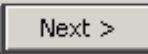
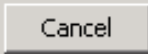
Component Name: ☐ Is parametric

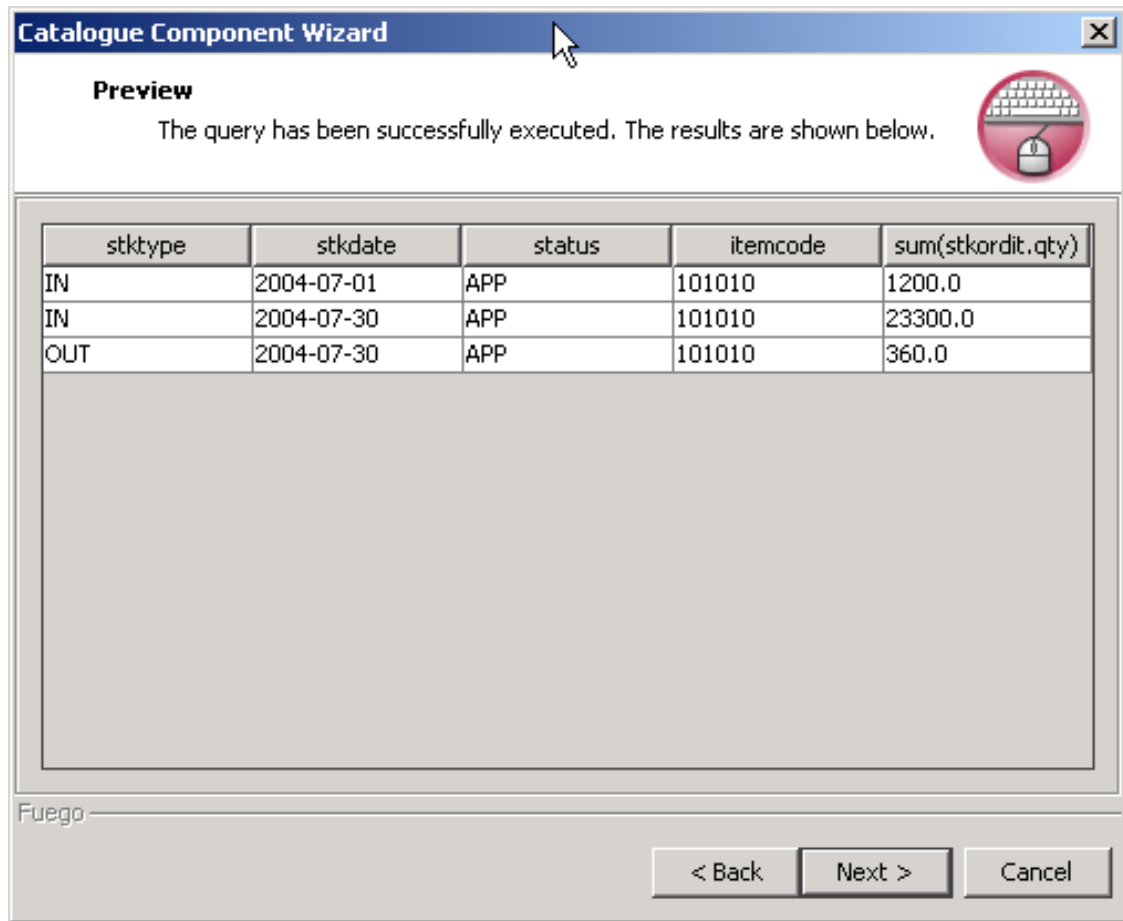
SQL Query:

```
select stkorder.stktype, stkorder.stkdate,  
       stkorder.status,  
       stkordit.itemcode, sum(stkordit.qty)  
from stkorder, stkordit  
where stkorder.stktype = stkordit.stktype and  
       stkorder.stkcode = stkordit.stkcode  
group by stkorder.stktype, stkorder.stkdate
```



Fuego



Using cataloged SQL Queries

Using an introspected SQL Query

In a BP-Method or Fuego Object Method, a SQL query is used with the *for each* statement to obtain each tuple resultant of the query execution.

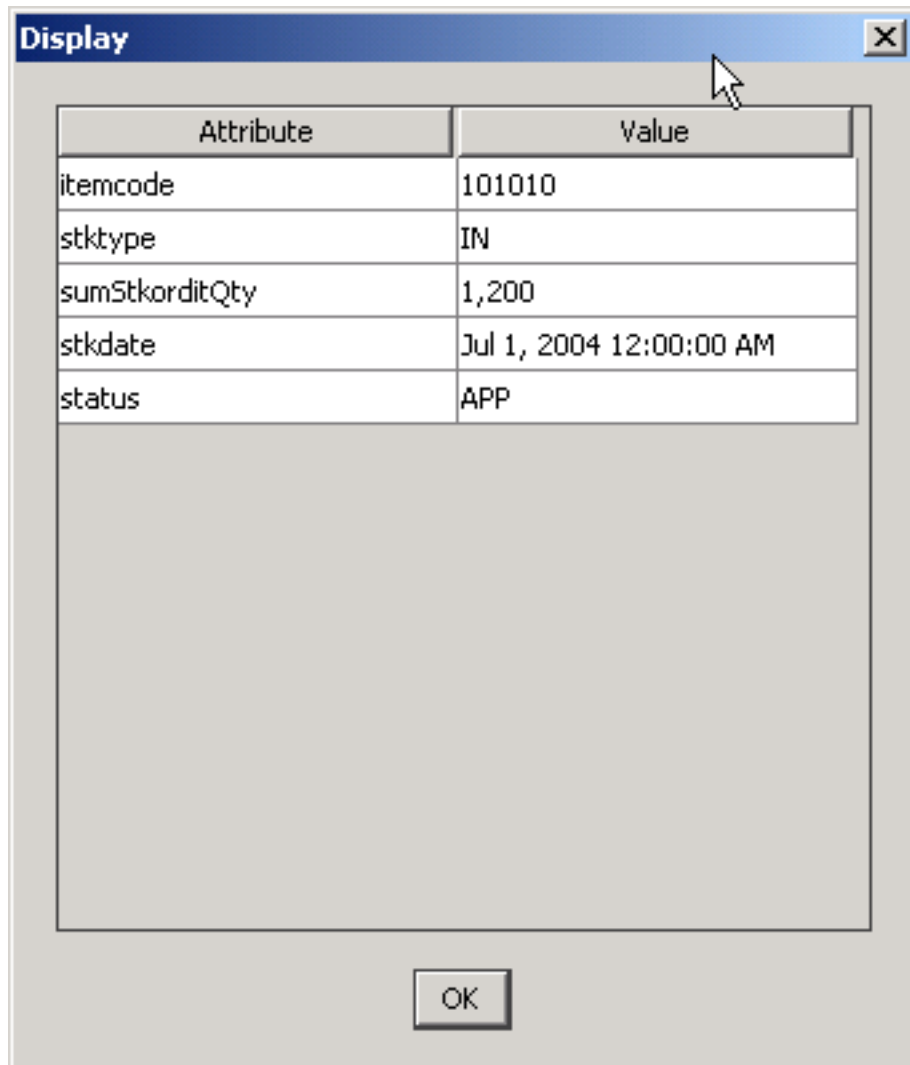
Using a selfcontained sql query

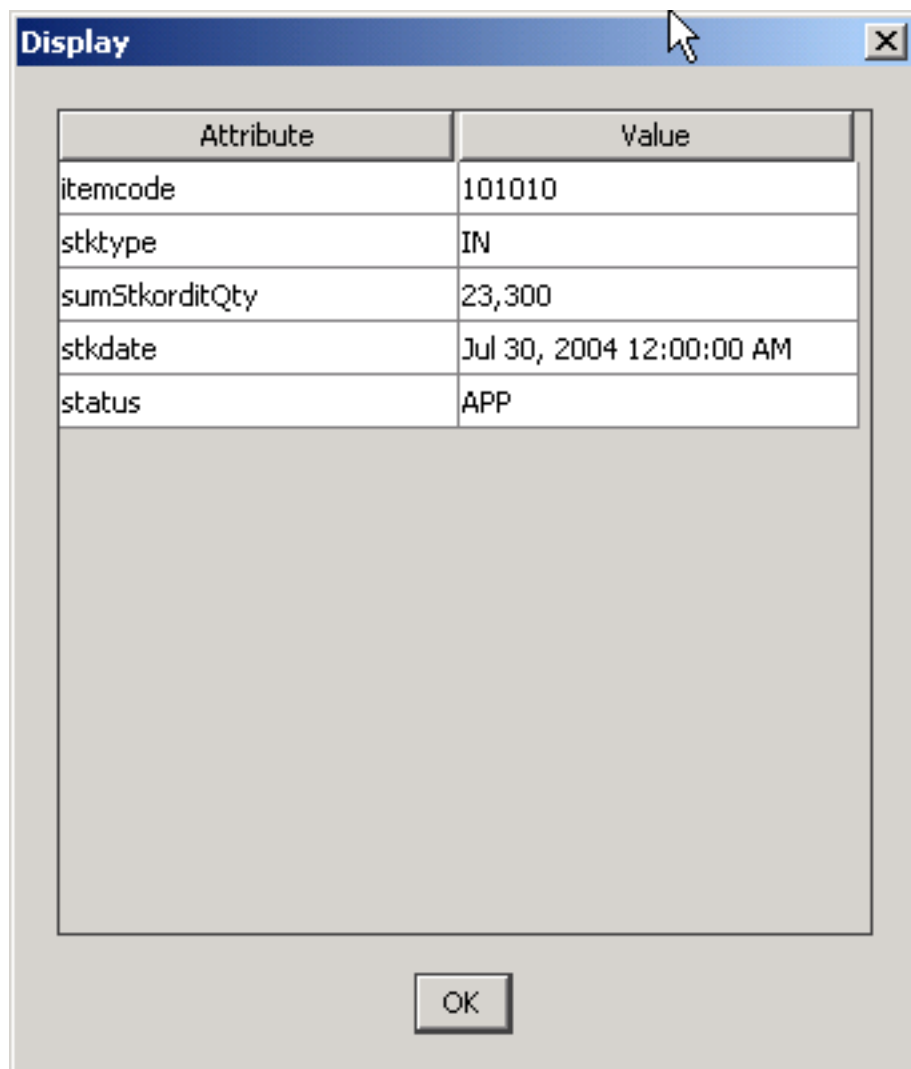
Using in a method the *TotalItemsQtybyTypeDate* sql query catalogued as example, the code would be:

```
for each element in DBOrders.TotalItemsQtybyTypeDate do
  // statements
  display element
```

end

and the display execution will show the following three dialogs:

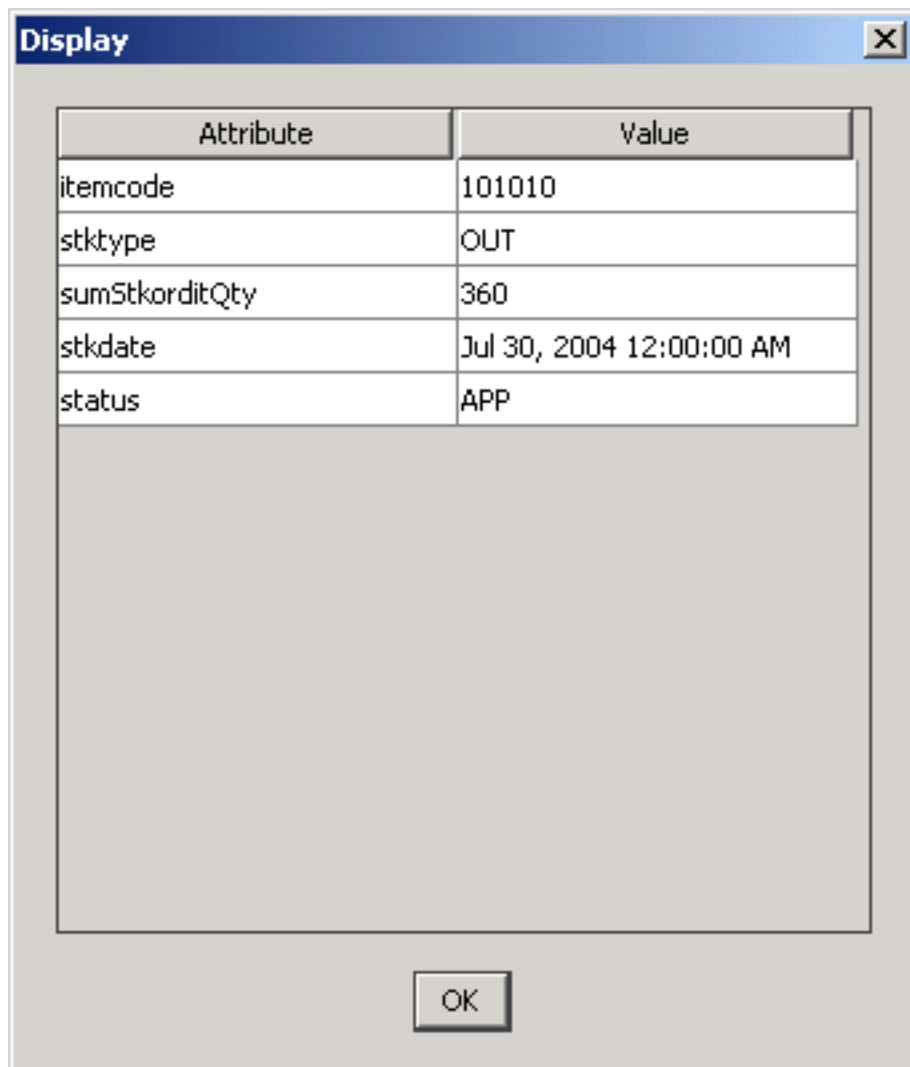




The image shows a 'Display' dialog box with a table containing the following data:

Attribute	Value
itemcode	101010
stktype	IN
sumStkorditQty	23,300
stkdate	Jul 30, 2004 12:00:00 AM
status	APP

Below the table is a large empty rectangular area, and at the bottom center is an 'OK' button.



Attribute	Value
itemcode	101010
stktype	OUT
sumStkorditQty	360
stkdate	Jul 30, 2004 12:00:00 AM
status	APP

OK

Which results are coincident with the preview of the sql query execution. Refer to section **Cataloguing Examples** above, example **Sumarizing the total quantity of items in Stock Orders by Type and Date**

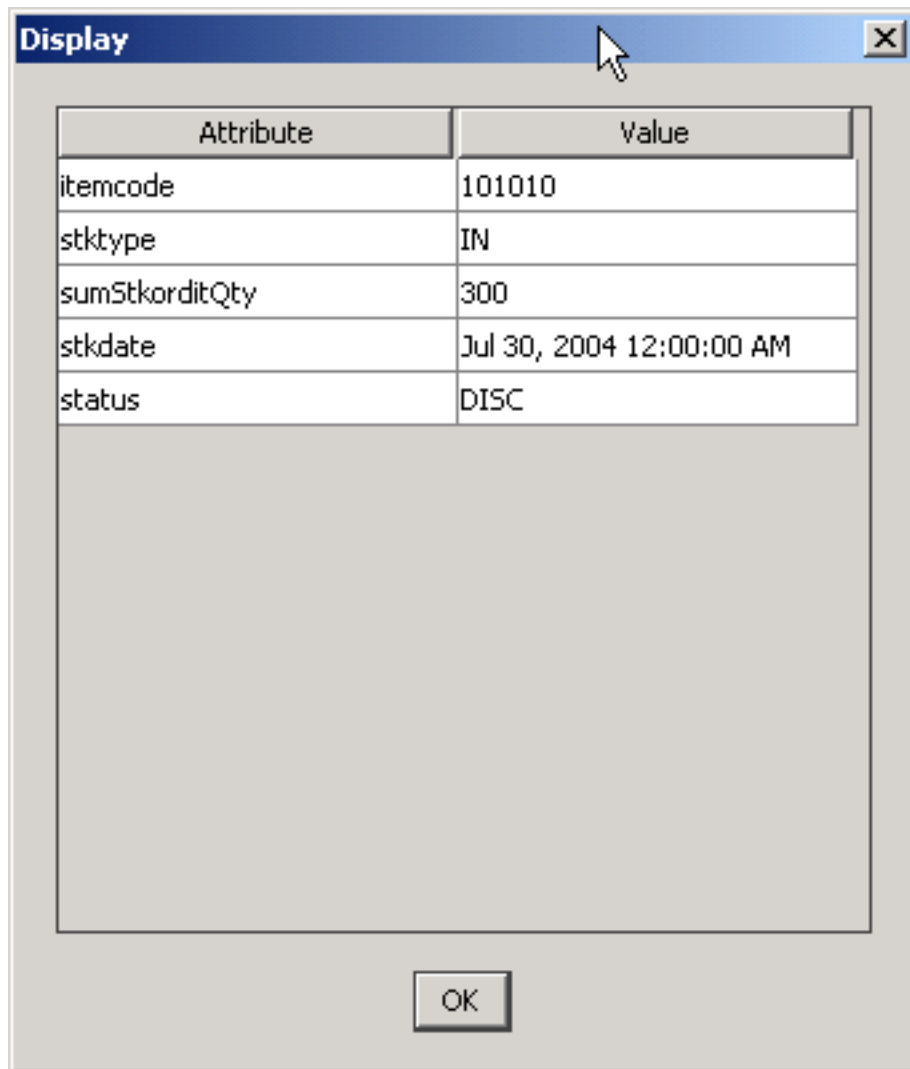
Using a parametric sql query

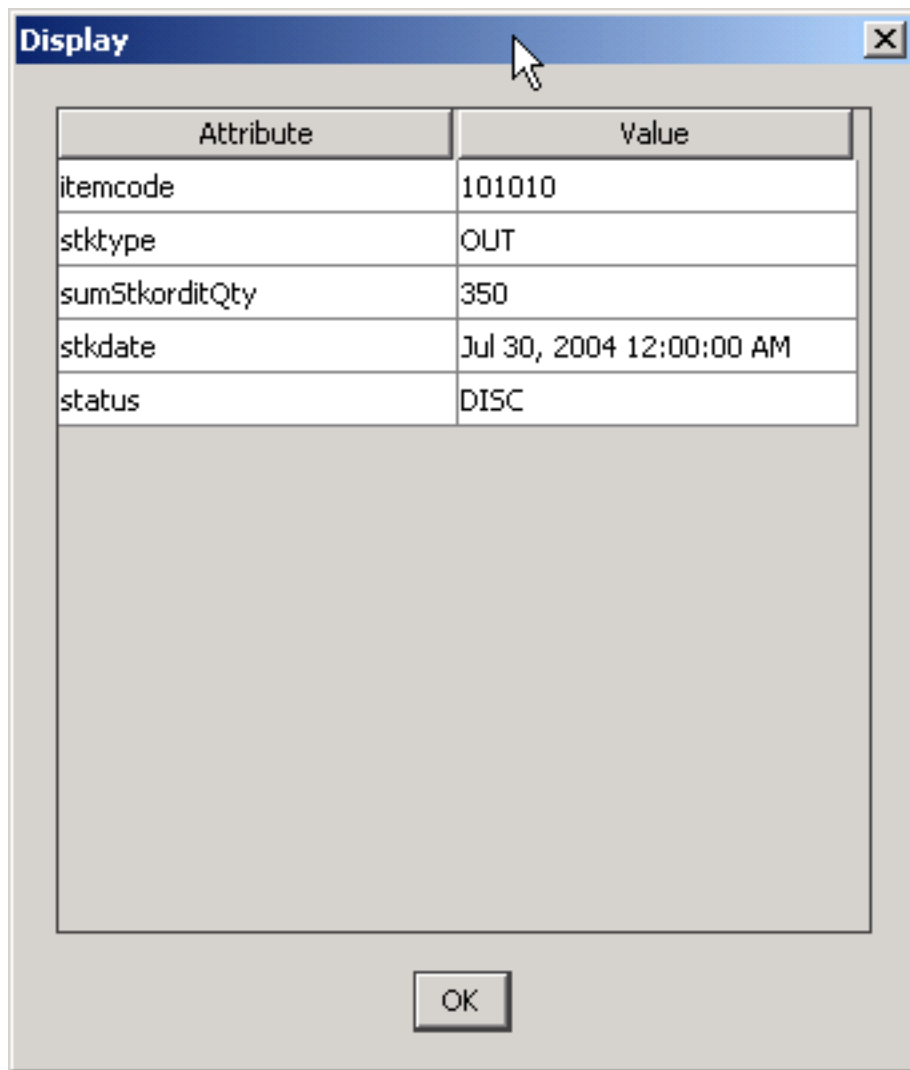
Using in a method the *StkOrdItemsQtyParamByStatus* sql query catalogued as example, the code would be:

```
for each element in
  DBOrders.StkOrdItemsQtyParamByStatus("DISC") do
  // statements
```

```
    display element  
end
```

and the display execution will show the following two dialogs, that are coincident with the parametric catalogation example in the SQL Queries section.





CORBA Components

FuegoBPM Studio allows you to catalog CORBA objects that reside in an Interface Repository. Once cataloged, you can manipulate the components of the CORBA object in your Method tasks in a process design.

Supported CORBA objects

Any CORBA type can be cataloged through FuegoBPM Studio into the Project Catalog. The types include the following:

- Aliases
- Attributes
- Constants
- Enums
- Exceptions
- Interfaces
- Modules
- Operations
- Primitives
- Structs
- Unions
- Values and Boxed values

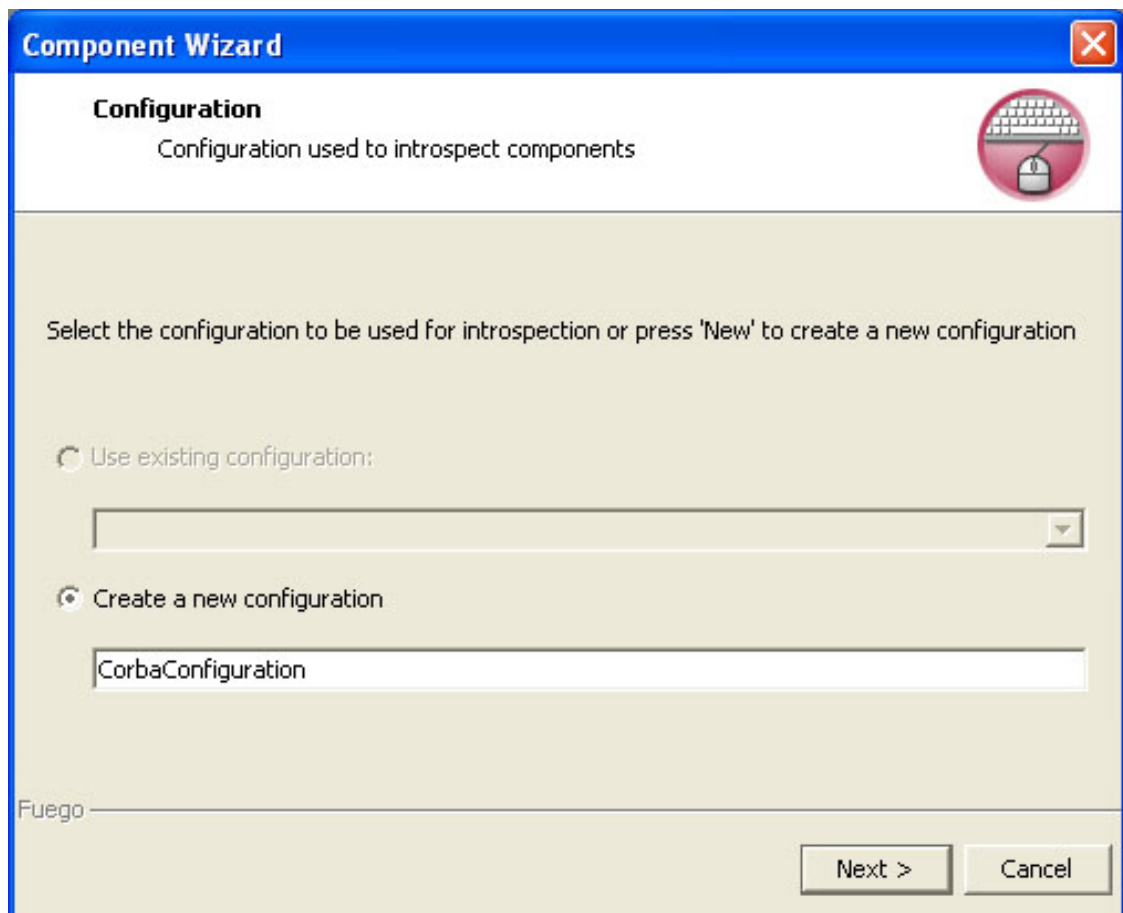
However, the following are the only types supported at run time:

- Interfaces with attributes and operations
- Structs
- Unions
- Sequences
- Enumerations
- Arrays
- Aliases

Adding a CORBA object

A wizard is provided to make the cataloging of a CORBA component easier for the process developer.

1 - Choose a name for the new configuration. Note that creating a configuration allows you to reuse it each time you need to add new components or to reintrospect existing ones.



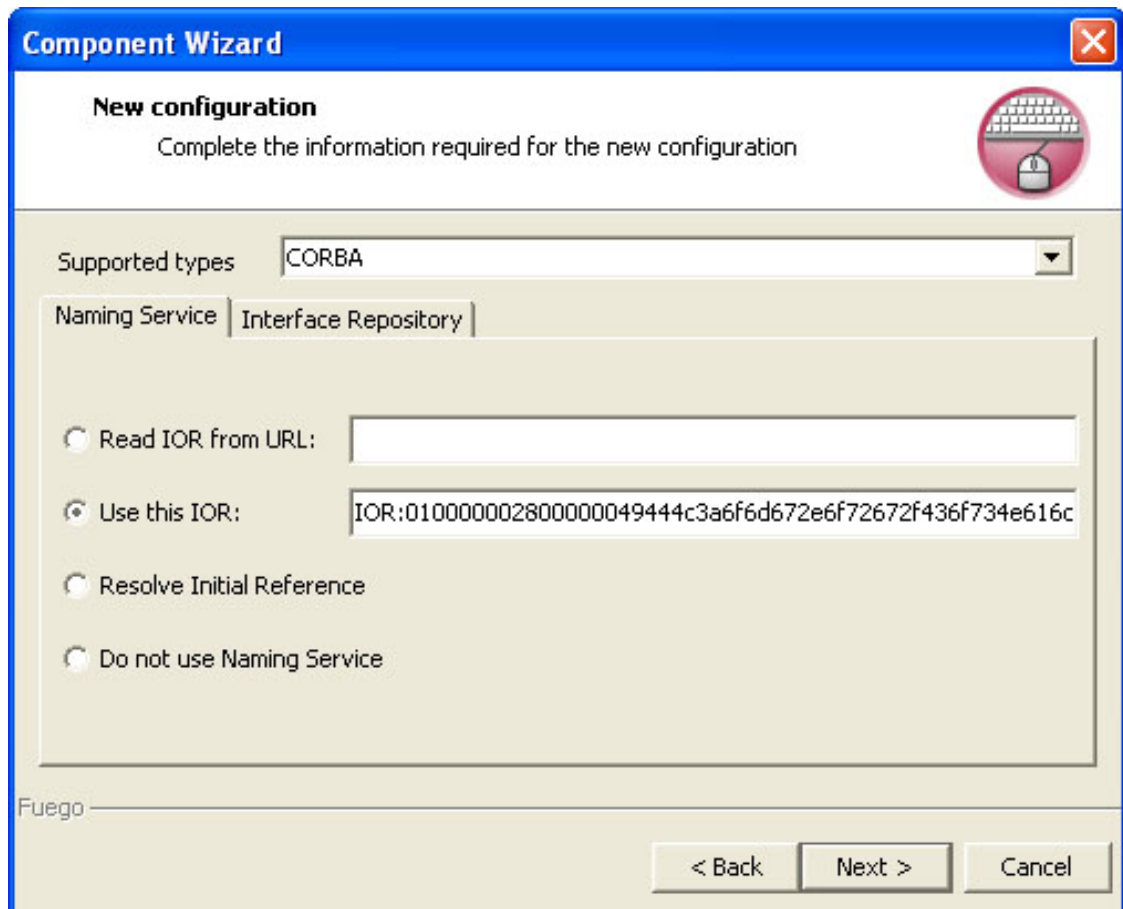
2 - Set Naming Service values:

In this step, you have to configure the values for the Naming Service. There are several options:

- Read IOR from URL: if you have the IOR exported to some service

(for example, a web server) and a URL exists that can be used to fetch it, select this option.

- Use this IOR: specify the IOR directly (**recommended**).
- Resolve Initial Reference: to request the ORB to get the reference of the service trying to resolve its reference. Note that this option is not recommended since it has interop problems when using different ORBs.
- Do not use a Naming Service: select this option if you do not need this service. However, remember that objects used in methods are found using the Naming Service. Hence, you will need to use the IOR for any object to be used in a method - passing it as a parameter in its constructor.



The screenshot shows the 'Component Wizard' dialog box with the title 'New configuration'. Below the title is the instruction 'Complete the information required for the new configuration'. The 'Supported types' dropdown is set to 'CORBA'. The 'Naming Service' tab is selected, and the 'Interface Repository' sub-tab is active. Four radio button options are present: 'Read IOR from URL' (unselected), 'Use this IOR:' (selected), 'Resolve Initial Reference' (unselected), and 'Do not use Naming Service' (unselected). The 'Use this IOR:' option has a text field containing the IOR string: 'IOR:010000002800000049444c3a6f6d672e6f72672f436f734e616c'. At the bottom left, the 'Fuego' logo is visible. At the bottom right, there are three buttons: '< Back', 'Next >', and 'Cancel'.

3 - Set the Interface Repository IOR.

Next, you have to set the values for the Interface Repository. The options are:

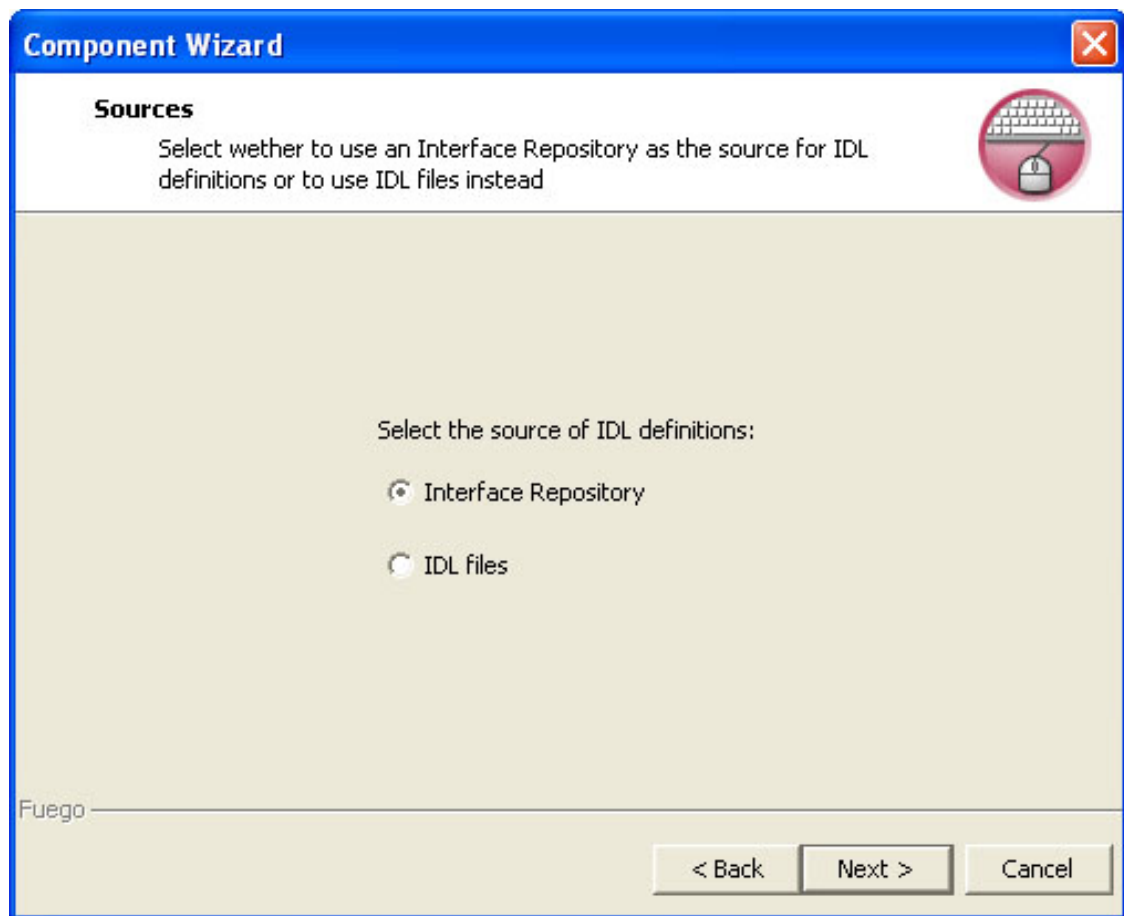
- Read IOR from URL: if you have the IOR exported to some service (for example, a web server) and a URL exists that can be used to fetch it, select this option.
- Use this IOR: specify the IOR directly (**recommended**).
- Resolve Initial Reference: to request the ORB to get the reference of the service trying to resolve its reference. Note that this option is not recommended since it has interop problems when using different ORBs.
- Use FuegoBPM Studio's Interface Repository: this service can be used when an ORB does not have an implementation of the Interface Repository service. Note that this service must be launched separately.

Once the values have been set, you can proceed to the next step.

4 - Select the source of the IDL definitions.

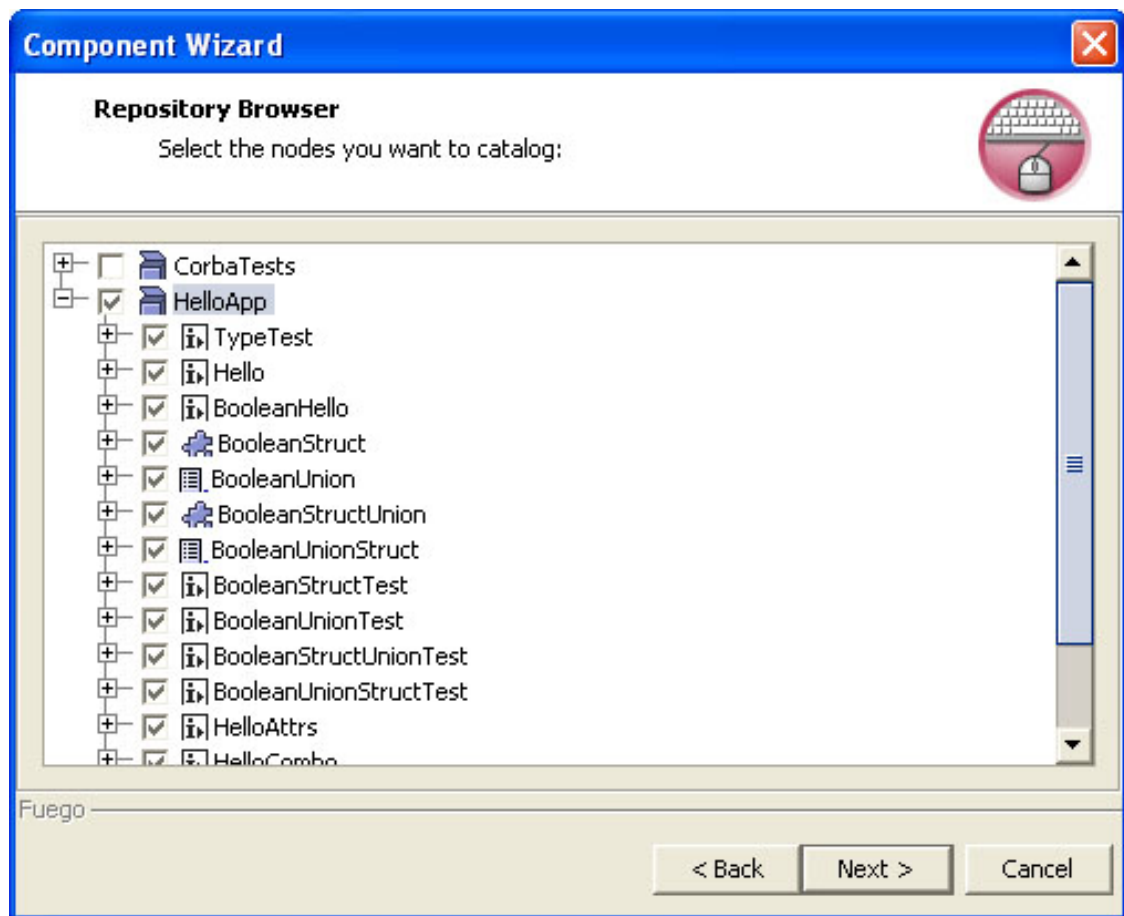
This step allows you to select the source of the IDL files:

- Interface Repository: the IDL definitions are located in this service (**recommended**); the introspector will connect to this service and fetch its contents.
- IDL files: any IDL definitions stored as files. This option **must** be used when the Interface Repository used is FuegoBPM Studio's Interface Repository service. It allows the user to select a number of IDL files, publish them in the repository and use them to catalog components.

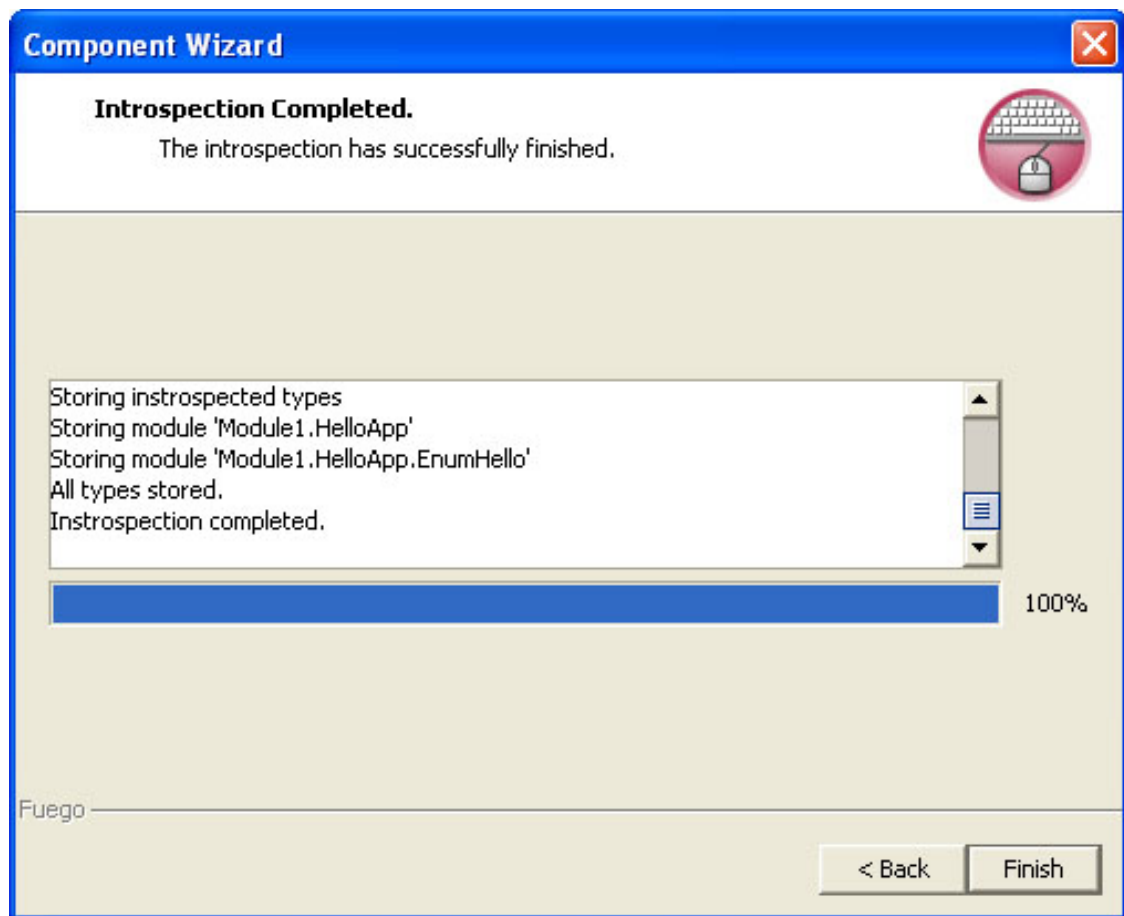


5 - Select the components

Select the components you want to catalog. Remember that if a parent tree is selected, all of its children will also be selected.



6 - Click **Next**. The Wizard is finished!



7 - Click **Finish** to close the Wizard.

See Also

Module Implementation

Sequences

Enumerations

Arrays

Aliases

Callbacks

CORBA and Methods

Module Implementation

The current implementation has several considerations to be kept in mind. The main constraints include POAs: the CORBA module uses Portable Object Adapters (POAs) instead of Basic Object Adapters (BOAs.) BOAs are not supported.

Method considerations

The CORBA services provided by FuegoBPM Studio support the following objects at runtime.

- Interfaces
- Structs
- Unions
- Sequences
- Enumerations
- Arrays
- Aliases

The Boxed Values object is not supported.

Primitive types

The table below shows the primitive types that are implemented and how they are mapped to BP-Method types:

CORBA type	Method type
Boolean	Bool
char	String

CORBA type	Method type
wide char	String
string	String
wide string	String
octet	Int
short	Int
unsigned short	Int
long	Int
unsigned long	Int
long long	Int
unsigned long long	Int
float	Real
double	Real

Out arguments

The OUT argument modifier can be used for every primitive type listed in the table above. However, for *Struct* and *Union* objects there is a limitation, they have to be instantiated **before** being passed as an output argument. Hence, for structures and unions, an example Method is the following:

```
structTest = CorbaTestsTestStruct()
structOutOp paramTest returning
structTest = aTestStruct
```

and

```
testUnion = CorbaTestsTestUnion()
```

```
testUnion.aBoolean = false  
paramTest = CorbaTestsParamTest("ParamTest")  
unionOutOp paramTest returning  
testUnion = aTestUnion
```

Summarizing, just as arguments should be instantiated with IN and INOUT arguments, arguments should also be instantiated with OUT arguments.

Union implementation

Union objects are implemented by using helpers that permit marshalling and unmarshalling to and from the server side. Before a union is marshalled, **any attribute of the union should be set**. In other words,

```
testUnion = CorbaTestsTestUnion()  
  
// the attribute is set before the union is used as  
// an argument  
testUnion.aBoolean = false  
  
paramTest = CorbaTestsParamTest("ParamTest")  
  
unionOutOp paramTest returning  
  
testUnion = aTestUnion
```

This is **required** for all operations involving the use of unions. Furthermore, since unions also have default attributes, they can be used as any other attribute. For example:

```
union ShortUnion switch(short)
{
    case0: string aString;
    case1: wstring aWString;
    default: short defaultShort;
};
```

Then, the default attribute is *defaultShort* and can be used as follows:

```
shortUnion = CorbaTestsShortUnion()
shortUnion.defaultShort = 1
unionTest = CorbaTestsUnionTest("ParamTest")
shortUnionTest unionTest using
aShortUnion = shortUnion
```

If no attribute is set and the union is used in a remote invocation, the execution framework will raise an exception indicating that any attribute should be set before the union is marshalled to the server side.

Struct implementation

The structure implementation is similar to the Union implementation. **All of the structure members should be initialized before use.** This means that no attribute (member) can be **Null** when it is used with remote invocations. Should this happen, the server will respond with a *marshal* error.

Sequences

Sequences are bound in Methods. If the sequence length exceeds its bound, an exception is thrown. For example, in the following Method

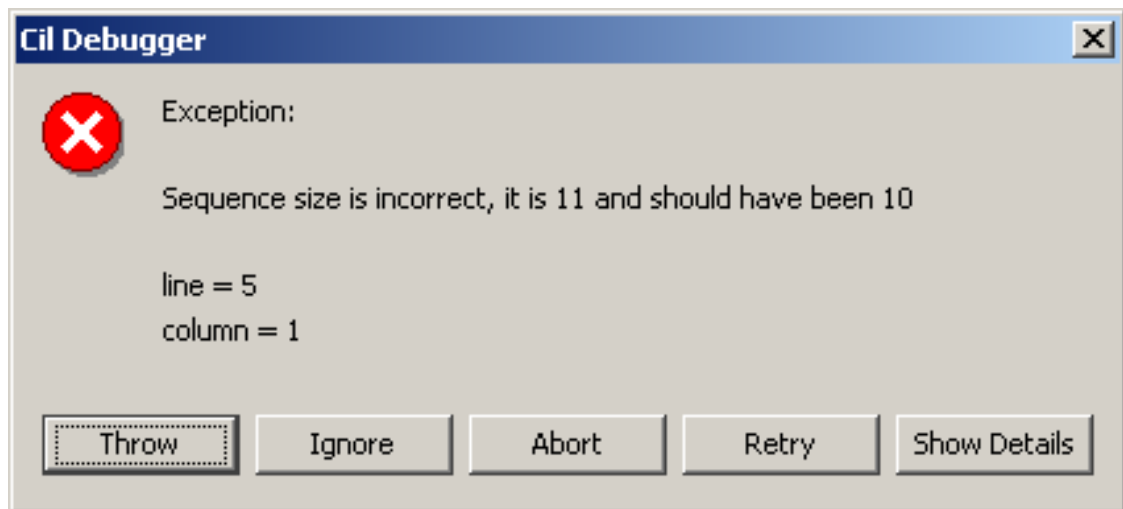
script the sequence has 11 members but can only have 10 members.

```
seqTest    = Module1.CorbaTests.SeqTest("SeqTest")
stringSeq = [ "1", "2", "3", "4", "5", "6", "7", "8",
              "9", "10", "11" ]

boundStringSeqOp seqTest using
aBoundStringSeq = stringSeq returning stringSeq

display stringSeq
```

When this example is run, it displays the following exception on the screen:

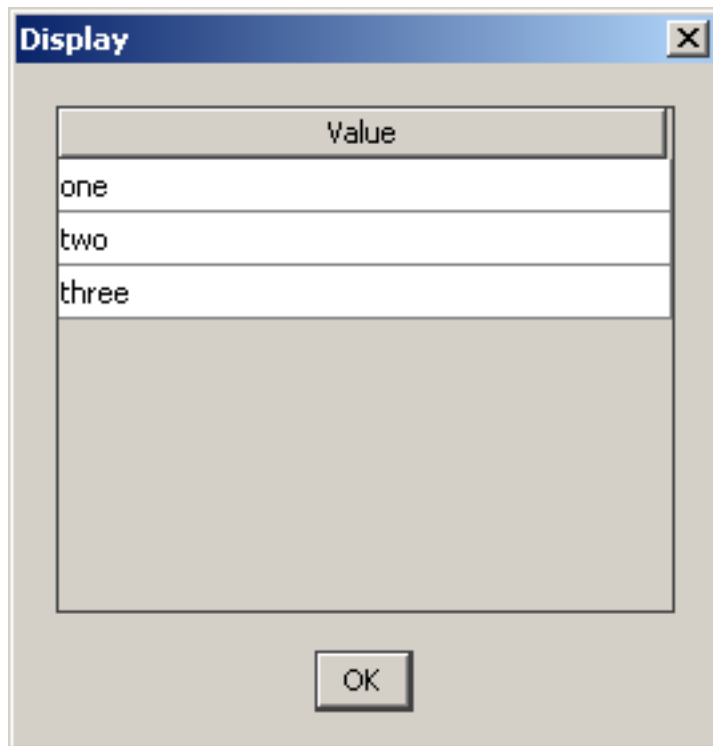


Sequences of primitive types

Any of the primitive types shown in Primitive types can be used to create a sequence. The following example shows how to create a *string* sequence.

```
seqTest = Module1.CorbaTests.SeqTest("SeqTest")
stringSeq = ["one", "two", "three"]
```

```
stringSeqOp seqTest using aStringSeq = stringSeq  
returning stringSeq  
display stringSeq
```



Sequences of structs

The Method script below shows how structure sequences can be used in the Method.

```
ts = 'now' // timestamp  
  
seqTest = Module1.CorbaTests.SeqTest("SeqTest")  
  
// create the first struct  
structOne = Module1.CorbaTests.SeqTest.TestStruct()  
structOne.longMember = 10  
structOne.stringMember = String(ts)
```

```
// create the second struct
structTwo = Module1.CorbaTests.SeqTest.TestStruct()
structTwo.longMember = 20
structTwo.stringMember = String(ts)

// the array of structures
structSeq = [structOne, structTwo]

testStructSeqOp seqTest using
aTestStructSeq = structSeq
```

Sequence of unions

Using sequences of unions is the same as using sequences of structs, as shown in this example:

```
ts = 'now' // timestamp

seqTest = Module1.CorbaTests.SeqTest("SeqTest")

unionOne = Module1.CorbaTests.SeqTest.TestUnion()
unionOne.longMember = 10

unionTwo = Module1.CorbaTests.SeqTest.TestUnion()
unionTwo.stringMember = String(ts)

unionSeq = [unionOne, unionTwo]

testUnionSeqOp seqTest using
aTestUnionSeq = unionSeq
```

Sequences as INOUT/OUT parameters

Sequences can also be passed as OUT or INOUT arguments in operation invocations. Note that when using sequences of structures or unions as OUT arguments, the restriction that these objects must be instantiated first before being used is still applicable, shown as

follows:

```
seqTest = Module1.CorbaTests.SeqTest("SeqTest")

// union is instantiated
testUnion = Module1.CorbaTests.SeqTest.TestUnion()
testUnion.stringMember = "aString"

testUnionSeqOutOp seqTest returning

// union will be received in the sequence
testUnionSeq = aTestUnionSeq

unionOne = testUnionSeq[0]
unionTwo = testUnionSeq[1]
```

Enumerations

CORBA enumerations can be used in many situations. The following example shows how enumerations can be used in operation invocations or as the discriminator of a union.

```
module CorbaTests
{
  interface EnumTest
  {
    enum Slot { s1, s2, s3 };

    union UnionTest switch(Slot)
    {

      case s1: string    stringMember;
      case s2: long      longMember;
      default: boolean   booleanMember;

    };

    void enumOp(in Slot aSlot);

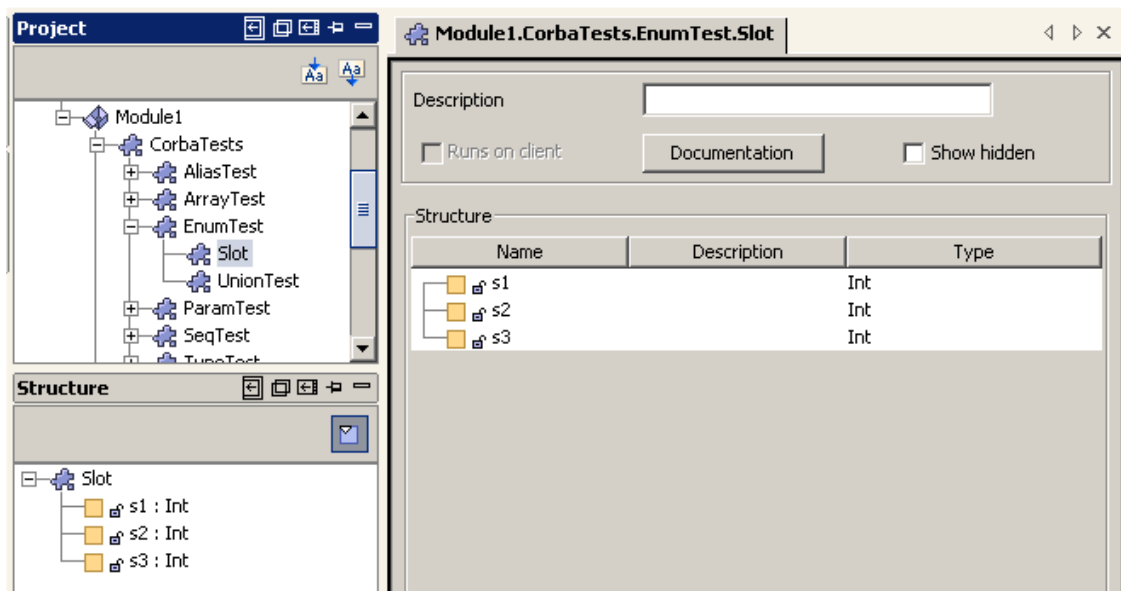
    void unionOp(in UnionTest aUnionTest);

    Slot retEnumOp(in Slot aSlot);

    void outEnumOp(out Slot aSlot);
```

```
void inoutEnumOp(inout Slot aSlot);
};
};
```

Using enumerations in BP-Methods is simple. When you catalog an IDL, you obtain the following:



Enumeration examples

The following examples show how to use the enumeration that appears in the Project Catalog after the IDL is obtained.

Example 1 - using enums as IN arguments

```
s1 = Module1.CorbaTests.EnumTest.Slot.s1
enumTest = Module1.CorbaTests.EnumTest("EnumTest")
enumOp enumTest using aSlot = s1
```

Example 2 - using enums as OUT arguments

```
enumTest = Module1.CorbaTests.EnumTest("EnumTest")  
outEnumOp enumTest returning s3 = aSlot
```

Example 3 - using enums as INOUT arguments

```
enumTest = Module1.CorbaTests.EnumTest("EnumTest")  
inoutEnumOp enumTest using aSlot = s2 returning s4 = aSlot
```

Example 4 - enum as the return type

```
s2 = Module1.CorbaTests.EnumTest.Slot.s2  
enumTest = Module1.CorbaTests.EnumTest("EnumTest")  
retEnumOp enumTest using aSlot = s2 returning s1
```

Example 5 - enum as the discriminator of a union

```
enumTest = Module1.CorbaTests.EnumTest("EnumTest")  
unionTest = Module1.CorbaTests.EnumTest.UnionTest()  
unionTest.longMember = 10  
unionOp enumTest using aUnionTest = unionTest  
unionTest.stringMember = "aString"  
unionOp enumTest using aUnionTest = unionTest  
unionTest.booleanMember = true  
unionOp enumTest using aUnionTest = unionTest
```

Arrays

Unidimensional arrays are supported with CORBA in BP-Methods. An example of arrays is shown in the following IDL.

```
interface ArrayTest
{
    typedef string  StringArray[];
    typedef long    LongArray[];
    typedef boolean BooleanArray[];

    struct TestStruct
    {
        long    longMember;
        string  stringMember;
    };

    union TestUnion switch(boolean)
    {
        case TRUE: long longMember;
        case FALSE: string stringMember;
    };

    typedef TestStruct TestStructArray[];
    typedef TestUnion  TestUnionArray[];
};
```

Array examples

The following examples illustrate different ways to use arrays with CORBA objects.

Example 1 - arrays as IN arguments

```
arrayTest = Module1.CorbaTests.ArrayTest("ArrayTest")  
stringArray = [ "one", "two", "three",  
                "four", "five" ]  
  
stringArrayInOp arrayTest  
    using aStringArray = stringArray  
    returning stringArray  
  
display stringArray
```

Example 2 - arrays as OUT arguments

```
arrayTest = Module1.CorbaTests.ArrayTest("ArrayTest")  
stringArray = [ "one", "two", "three",  
                "four", "five" ]  
  
stringArrayOutOp arrayTest  
    returning stringArray = aStringArray  
  
display stringArray
```

Example 3 - arrays as INOUT arguments

```
arrayTest = Module1.CorbaTests.ArrayTest("ArrayTest")  
stringArray = [ "one", "two", "three",  
                "four", "five" ]  
  
stringArrayInoutOp arrayTest  
    using aStringArray = stringArray  
    returning stringArray = aStringArray
```

Aliases

Aliases are not considered as attributes. Instead, when an alias

usage is found, it is replaced with the original definition of the alias.

The following example illustrates how aliases are cataloged.

```
interface AliasTest
{
typedef char Null;
typedef float Money;

enum TestEnum { e1, e2, e3 };

struct TestStruct
{
string stringMember;
long    longMember;
};

union TestUnion switch (TestEnum)
{
case e1: TestStruct structMember;
default: Null empty;
};

Money moneyInOp(in Money amount);
Money moneyOutOp(out Money amount);
Money moneyInoutOp(inout Money amount);
};
```

The following table shows how the objects are structured after cataloging this interface. After cataloging this interface the objects are structured as follows:

- *AliasTest*
 - Operations (Name/Argument Type):
 - *moneyInOp: Real*
 - *moneyOutOp: Real*
 - *MoneyInoutOp: Real*

- Attributes (Name/Type): None
- *TestEnum*
 - Operations (Name/Argument Type): None
 - Attributes (Name/Type):
 - *e1: Int*
 - *e2: Int*
 - *e3: Int*
- *TestStruct*
 - Operations (Name/Argument Type): None
 - Attributes (Name/Type):
 - *stringMember: String*
 - *longMember: Int*
- *TestUnion*
 - Operations (Name/Argument Type): None
 - Attributes (Name/Type):
 - *structMember: TestStruct*

- *empty: String*

As you can see, all references to aliases have been replaced and the type of alias has been set wherever it was used. Thus, the objects can be used as any object inside a Method.

Callbacks

When IDL files are cataloged from the Interface Repository, a list of interfaces is displayed. Select those interfaces that are normally used as callbacks (the Notification interface in the example).

Once the wizard has been finished, the module keeps this list of callback interfaces internally. In fact, the list is loaded when the module is instantiated.

How to Use Callbacks

1. Catalog a CORBA interface with callback methods.
2. Right-click on the interface node and select the action 'ImplementInterface'.

This will create a Fuego Object in the same module of the Callback interface. This is the object you should pass to the CORBA object. Then, CORBA will call the callback method that will be implemented in the Fuego Object.

As a *Callback* is **Asynchronous**, you should put your method on hold until the callback is called. In order to do that, there is a component **Fuego.Utills.Event** with two methods: *wait()* and *trigger()*. The *wait* can also receive an Interval that indicates the time to wait before raising an Exception.

The following examples show how to use them:

- Invoking method:

```
newUserAccount = UserAccount()  
  
newUserAccount.userName = "John"  
newUserAccount.password = "Smith"  
  
//here we instantiate the Notification Implementation  
notifImpl = NotificationImpl();  
  
//we set the event attribute  
//Note: this attribute was added to the Fuego Object  
//after Implementing the interface  
notifImpl.event = Fuego.Util.Event("Test Event")  
manager = Manager("AccountManager")  
  
manager.createUserAccount(notifImpl, newUserAccount);  
  
waitFor notifImpl.event using timeout = '20s'
```

- Callback method

```
trigger event
```

CORBA and Methods

After you have cataloged CORBA objects, you can use them in BP Methods.

CORBA Methods examples

The following examples are provided to demonstrate the use of CORBA objects in Methods. Note that these examples only show the client side.

Example 1 - "Hello World"

Consider the following IDL:


```
interface Hello  
{  
    string sayHello();  
    oneway void shutdown();  
    void op1(in string arg0);  
    void op2(inout string arg0);  
    void op3(out string arg0);  
};
```

The Method is written as follows:

```
helloApp = Hello("Hello")  
sayHello helloApp returning ret  
display ret
```

The interface constructor receives one argument: the name of the object bound in the Naming Service. When a server is started, CORBA objects are registered in the Naming Service through a name. This name must be used when a lookup of that object is performed. In this example, the name is **Hello**.

Note

 You can also specify the IOR of the object instead of its name.

After the object is constructed, the method *sayHello* is called. The method returns a *String* that is stored in the variable **ret**.

Once the invocation is complete, the variable is displayed with the display statement resulting in the following dialog box:

**Example 2 - Using arguments**

Using the same object as in example 1, the following example calls *op1* with a *String* argument:

```
helloApp = Hello("Hello")  
op1 helloApp using  
arg0 = "testArg0"
```

Three types of argument modifiers are supported:

- IN
- OUT
- INOUT

The example shown above demonstrates an IN modifier. The following examples illustrate the use of OUT:

```
interface ParamTest
{
void outOp(out boolean arg);

void mulOutOp (out boolean arg0, out string arg1, out long arg2);
};
```

```
outOp paramTest returning
aBoolean = arg
```

and

```
mulOutOp paramTest returning
aBoolean = arg0, aString = arg1, aLong = arg2
```

Finally, an INOUT argument modifier is illustrated in the Method below:

```
helloApp = Hello("Hello")
ret = "Test"
op2 helloApp using arg0 = ret
display ret
```

Structs and Unions can also have OUT or INOUT modifiers. However,

there are special issues to consider. See CORBA Module Implementation for further information.

Example 3 - Using attributes

Consider the following IDL:

```
interface HelloAttrs  
  
{  
  
  attribute string attr1;  
  
  readonly attribute string attr2;  
  
};
```

Attributes of CORBA objects can be used as follows:

```
h2 = HelloAttrs("HelloAttrs")  
h2.attr1 = "test"  
display h2.attr1
```

Example 4 - Object arguments and simple return types

In addition to the IDL in the using attributes example above, the following attribute is used to illustrate this example.

```
interface HelloCombo  
  
{  
  
  string op1(in HelloAttrs attrs);  
  
  HelloAttrs op2();  
  
  HelloAttrs op3(in HelloAttrs attrs);  
  
};
```

op1 is used in the following Method:

```
a = HelloAttrs("HelloAttrs")
h = HelloCombo("HelloCombo")
op1 h using
attrs = a returning ret
display ret
```

When you run the previous Method in the Method Debugger, the following dialog box appears.



Example 5 - Using Structs

Structures are not different from other objects except for the fact that they do not need to be bound to a remote object. Instead, they are kept in the client side. Still, they can be used as arguments or can be returned values of invocations.

The following IDL illustrates this example:

```
struct PersonStruct
{
string first_name;
string last_name;
```

```
HelloAttrs hello_attrs;  
  
};  
  
interface Person  
{  
  
void set(in PersonStruct _aPerson);  
  
PersonStruct get();  
  
};
```

A Method that uses this IDL is as follows:

```
s = PersonStruct()  
s.firstName = "John"  
s.lastName = "Smith"  
  
display s.firstName  
p = Person("Person")  
set p using aPerson = s  
get p returning s  
  
display s.firstName
```


Since the server side implementation modifies the result of the structure, when the first display is reached in the Method Debugger it shows:



When the object is returned, the display shows:



Note

 When an object is passed as a parameter, it is marshalled to the server side. When an object returns, an internal copy (clone) of the object is generated and the values read from the stream are set into the cloned object. This is important when Union objects are used, as illustrated in **Example 6 - using unions**.

Example 6 - Using Unions

To illustrate this example, consider the following IDL:

```
union BooleanUnion switch(boolean)
{
case TRUE: boolean aBoolean;
case FALSE: string aString;
};
```



```
interface BooleanUnionTest

{

void op1(in BooleanUnion aBooleanUnion);

BooleanUnion op2();

};
```

For this example, the Union is set to use the boolean attribute and then pass it as a parameter to the op1 defined in the BooleanUnionTest interface:

```
booleanUnion = BooleanUnion()
booleanUnion.aBoolean = true
display booleanUnion.aBoolean
display booleanUnion.aString
booleanUnionTest = BooleanUnionTest("BooleanUnionTest")
op1 booleanUnionTest using
aBooleanUnion = booleanUnion
op2 booleanUnionTest returning booleanUnion
display booleanUnion.aBoolean
```

When you run the previous Method in the Debugger, the first display statement shows "True", which is correct. However, the second one shows "Null Value". The reason is that Unions are based on the last attribute set. In other words, before getting a value from a union, the user must set a value first. Any attempt to get a value that has not been previously set will result in a *Null* return.

When the operation is called, the union is sent to the server side, modified by it and returned to the client. Then, the union is cloned and, based on the current attribute (the last one that was set), the

Union is rebuilt.

In this case, the last display will appear:



This is the result of the modification made on the server side implementation.

Example 7 - Combining operations, Unions and Structs

As we have seen before, it is quite simple to create and use CORBA objects. Now, a more complete example of what can be achieved with CORBA will be presented. The main idea is to combine structures, unions and operations.

The IDL for this example is:

```
struct BooleanStruct
{
    string field1;
    boolean field2;
};

union BooleanUnion switch(boolean)
{
    case TRUE: boolean aBoolean;
```

```
case FALSE: string aString;

};

union BooleanUnionStruct switch (long)
{
case 0: boolean aBoolean;
case 1: BooleanStruct aBooleanStruct;
case 2: BooleanUnion aBooleanUnion;
default:
boolean anotherBoolean;
};

interface BooleanUnionStructTest
{
void set(in BooleanUnionStruct aBooleanUnionStruct);
BooleanUnionStruct get();
};
```

The Method for this example is:

```
booleanStruct = BooleanStruct()
booleanStruct.field1 = "field1"
booleanStruct.field2 = true

booleanUnion = BooleanUnion()
booleanUnion.aBoolean = true
```

```
booleanUnionStruct = BooleanUnionStruct()  
booleanUnionStruct.aBoolean = true
```

```
// first case: testing union 'aBoolean'  
  
// attribute  
booleanUnionStructTest =  
    BooleanUnionStructTest("BooleanUnionStructTest")  
  
set booleanUnionStructTest using  
aBooleanUnionStruct = booleanUnionStruct  
booleanUnionStruct.aBoolean = false  
  
get booleanUnionStructTest returning booleanUnionStruct  
if not booleanUnionStruct.aBoolean then  
    display "Method failed!"  
    exit  
end
```

```
// second case: testing union 'aBooleanStruct'  
  
// attribute  
booleanUnionStruct.aBooleanStruct = booleanStruct  
  
set booleanUnionStructTest using  
    aBooleanUnionStruct = booleanUnionStruct  
    booleanUnionStruct.aBooleanStruct = null  
  
get booleanUnionStructTest returning booleanUnionStruct  
booleanStruct = booleanUnionStruct.aBooleanStruct
```

```
if not booleanStruct.field1 == "field1" then
    display "METHOD failed!"
    exit
end

if not booleanStruct.field2 then
    display "METHOD failed!"
    exit
end
```

```
// third case: testing union 'aBooleanUnion'

// attribute
booleanUnionStruct.aBooleanUnion = booleanUnion

set booleanUnionStructTest using
aBooleanUnionStruct = booleanUnionStruct
booleanUnionStruct.aBooleanUnion = null
get booleanUnionStructTest returning booleanUnionStruct
booleanUnion = booleanUnionStruct.aBooleanUnion
if not booleanUnion.aBoolean then
    display "Method failed!"
    exit
end
```

This example modifies the BooleanUnionStruct object and uses the BooleanUnionStructTest to send and receive it, to and from the server side. This Method is used to test marshaling/unmarshaling of local objects.

Cataloguing SAP BAPIs

What is a SAP BAPI?

Business Application Programming Interfaces (BAPIs) are standard SAP interfaces that enable software

vendors to integrate their software into the mySAP Business Suite. BAPIs are technically implemented using

RFC (Remote Function Call) enabled function modules inside SAP systems.

BAPIs are defined in the Business Object Repository (BOR) as methods of to SAP business objects that

perform specific business tasks. They allow integration at business level, not technical level. This makes it much

easier to find suitable BAPIs compared to non-BAPI based function modules.

How does FuegoBPM integrate with SAP?

FuegoBPM uses the SAP Java Connector (JCo) to integrate with its BAPI library to access SAP business objects.

The SAP Java Connector (JCo) is a toolkit that allows Java applications to communicate with SAP systems. JCo

is a high-performance encapsulation of the RFC Library that supports all features of RFC. It combines an easy

to use API with unprecedented flexibility and performance. It can be used to implement BAPI based integrations.

You also need additional files to be able to use this integration:

If you are using FuegoBPM Studio in a Windows environment:

- *librfc32.dll*, it has to be located under the *system32* directory, and
- *sapjcorfc.dll*, it has to be located under the *ext* directory of the

FuegoBPM Studio installation.

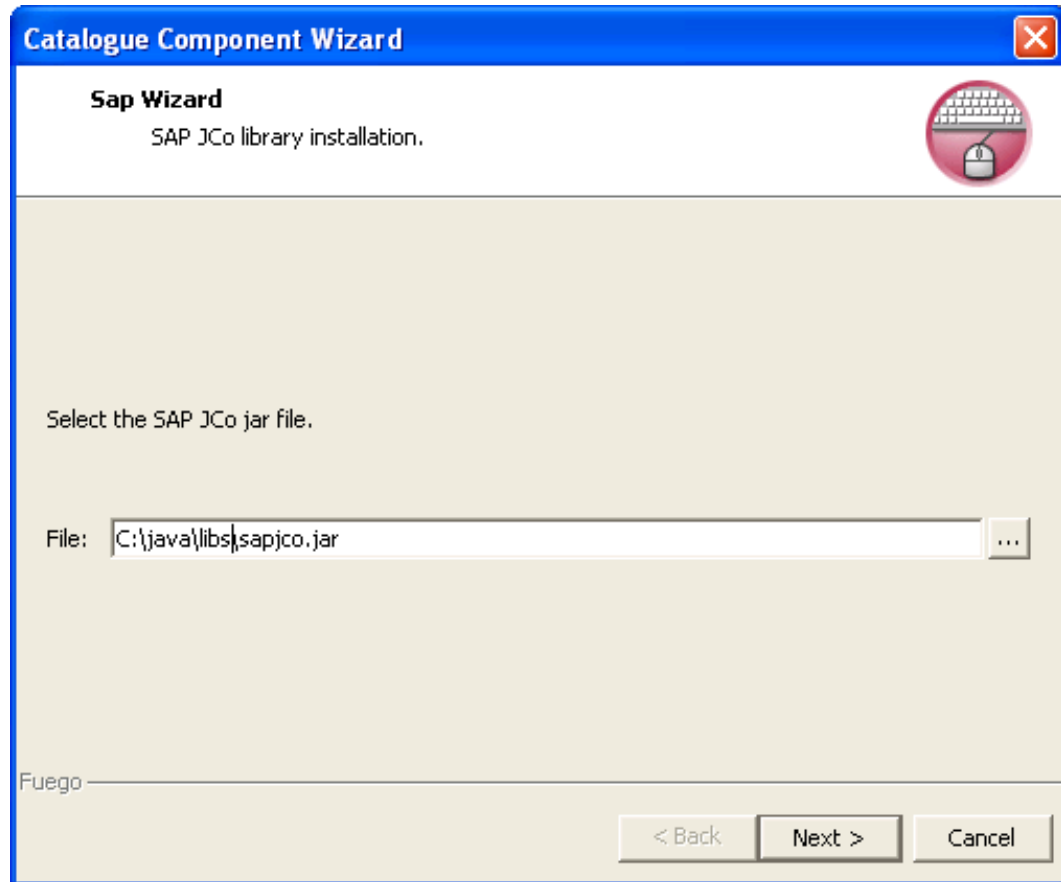
If you are using FuegoBPM Studio in a Unix environment:

- Copy the file *librfccm.so* to your system file and add the folder that contains it to environment variable LD_LIBRARY_PATH,
- *libsapjcorfc.so*, it has to be located under the *ext* directory of the FuegoBPM Studio installation.

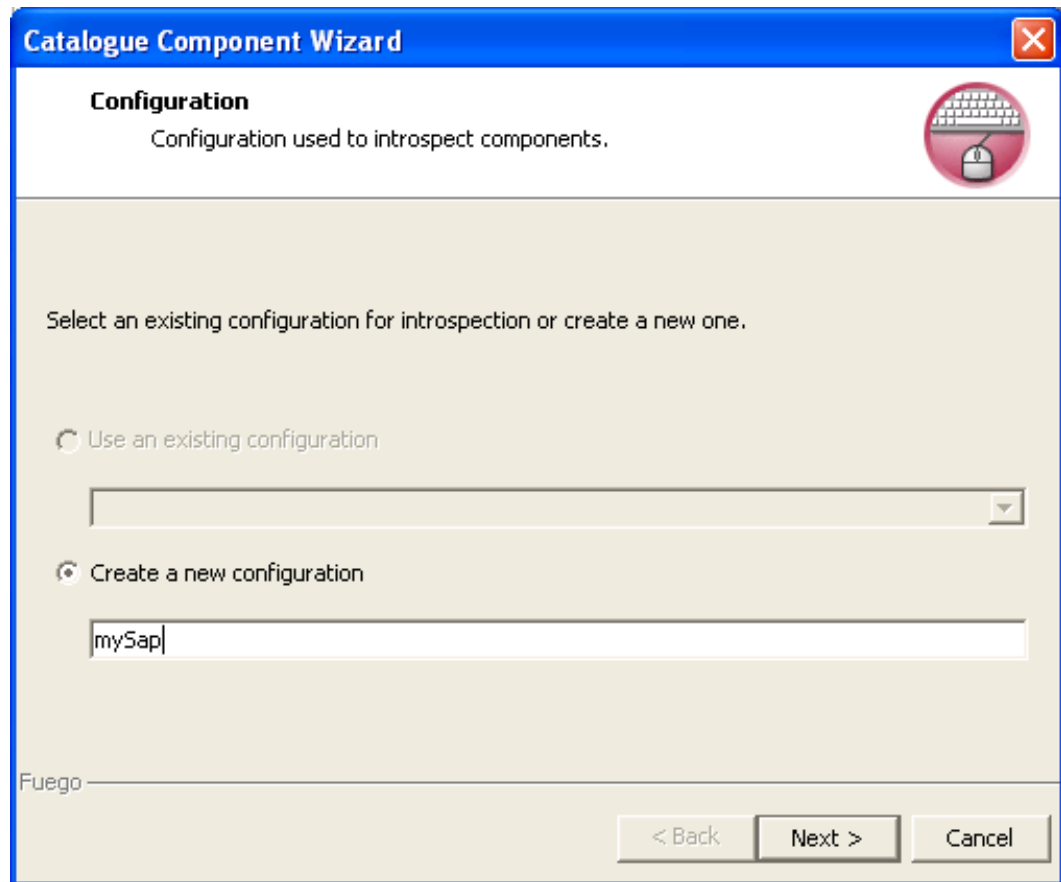
Introspecting SAP BAPIs

To introspect SAP BAPIs,

1. Right-click on the module you want to add the BAPI and select the options *Cataloge Component/ SAP* from the popup menus. The first time you add a BAPI with FuegoBPM Studio, the SAP JCo is required.



- . Browse the file system to the location where the jar file is and click **Next**. The application needs to be restarted. Next time you catalogue a BAPI, this step does not appear and the introspection begins in step 2.
2. When adding a BAPI, a configuration to connect to the SAP host is required. If you already have one defined, select it, if not you can create it in this step, give a name to it and click **Next**.



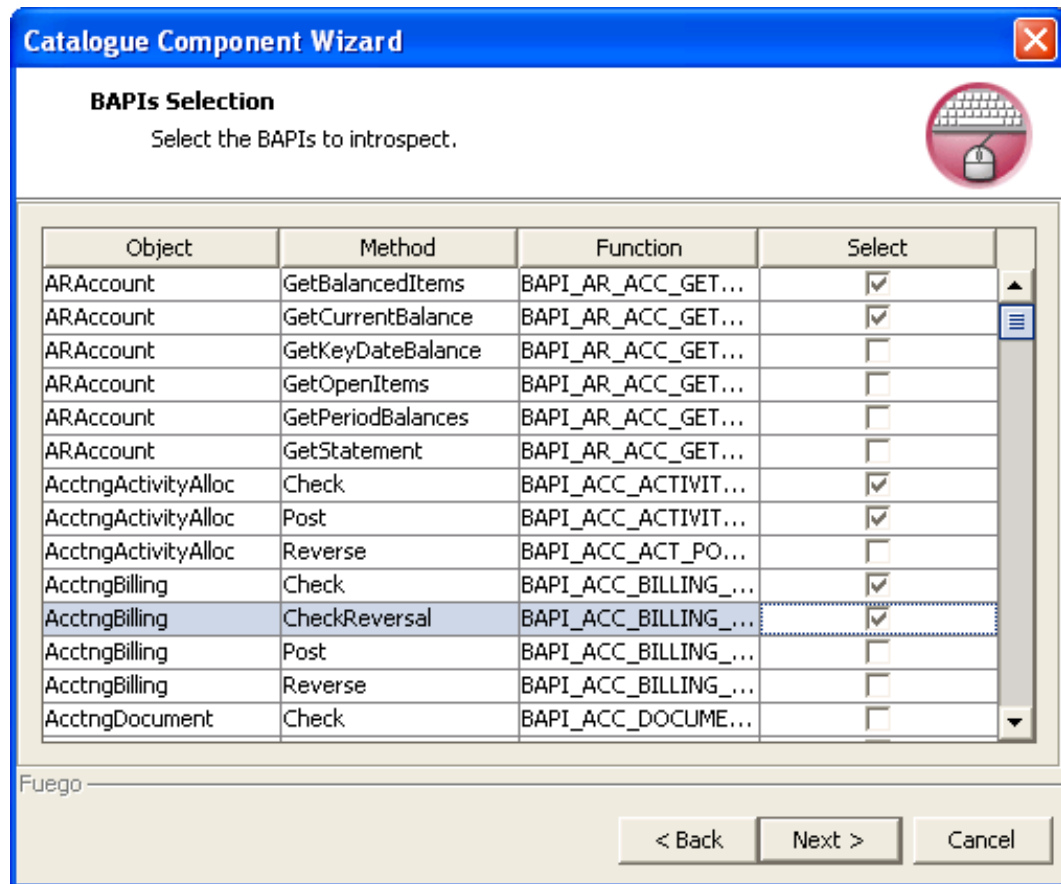
The screenshot shows a window titled "Catalogue Component Wizard" with a blue header bar. Below the header, the word "Configuration" is displayed in bold, followed by the text "Configuration used to introspect components." In the top right corner of the window, there is a red circular icon containing a keyboard and a mouse. The main area of the window has a light beige background and contains the instruction "Select an existing configuration for introspection or create a new one." Below this instruction, there are two radio button options. The first option, "Use an existing configuration", is unselected. Below it is a text box with a dropdown arrow on the right. The second option, "Create a new configuration", is selected. Below it is a text box containing the text "mySap". At the bottom left of the window, the word "Fuego" is followed by a horizontal line. At the bottom right, there are three buttons: "< Back", "Next >", and "Cancel".

3. If you are creating the configuration to connect to the SAP host, the following data is required:

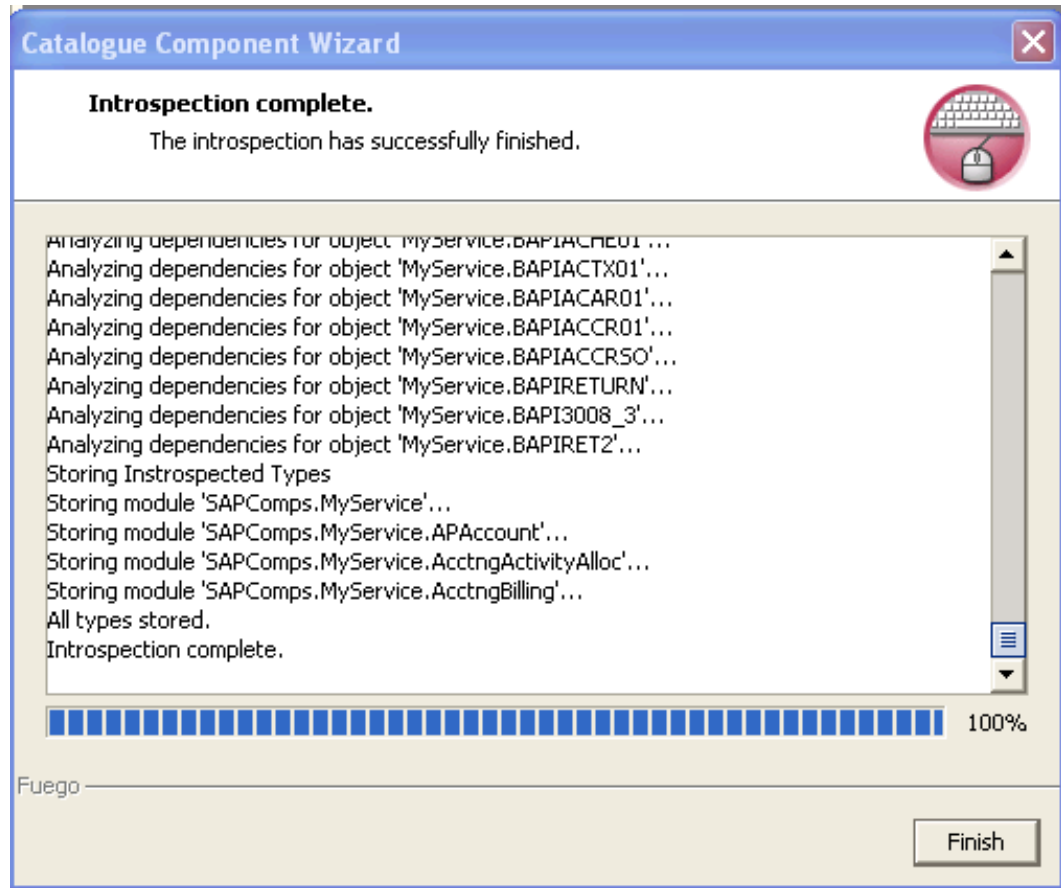
The screenshot shows a Windows-style dialog box titled 'Catalogue Component Wizard' with a blue header bar. Below the title bar, the text 'New Configuration' is displayed with a mouse cursor icon pointing to it. Underneath, a subtitle reads 'Fill in the information required for the new configuration.' To the right of the text is a circular icon containing a keyboard and a mouse. The main area of the dialog is a light beige color and contains a 'Details' section with a list of input fields. These fields are labeled 'Service Name', 'Client', 'User ID', 'Password', 'Host', 'System Number', 'Language', and 'Pool Size'. Each field has a corresponding text box with the following values: 'myService', '100', 'fuego000', '*****', 'host.name.net', '00', 'EN', and '1'. At the bottom of the dialog, there is a status bar with the text 'Fuego' on the left and three buttons on the right: '< Back', 'Next >', and 'Cancel'.

Field	Value
Service Name	myService
Client	100
User ID	fuego000
Password	*****
Host	host.name.net
System Number	00
Language	EN
Pool Size	1

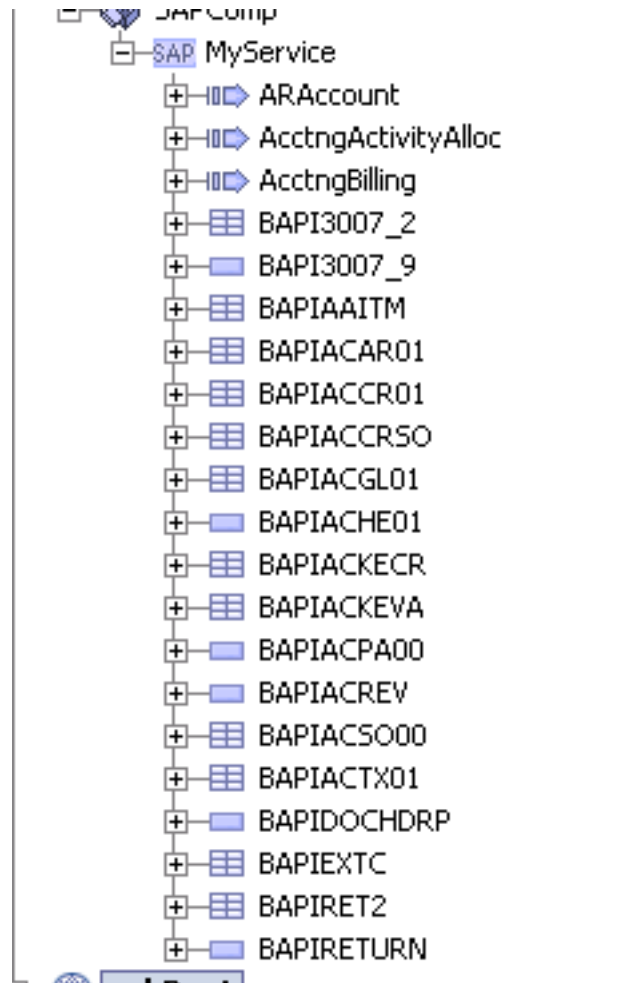
- a. **Service Name** : Is the name under where the introspected BAPIs will be hold in the Project Catalogue.
 - b. SAP information to connect to the host where it is located:
Client, User ID, User Password, Host, System Number, Language
 - c. **Pool Size**: Quantity of opened connections to access SAP with the defined configuration.
4. Once the connection is stablished with SAP, the list of available BAPIs appears. Select those you need and click **Next**, the instropction begins.






- Once the introspection is completed, click **Finish** to end.



6. All the introspected BAPIs appear under a component with the name of the *Service Name* of the configuration used to instropect.



In the example, under *mySAP* different elements have been created.

1. , objects created base on the *ObjectName* of the BAPIs selected when introspecting. The BAPIs are created as methods inside this objects.
2. , the TABLES, that can be iterated.
3. , SAP Structures, these are objects with attributes.

Enumerations

An enumeration is a way to add your own predefined variable that is going to be used throughout the life cycle of a process. The variable is defined to be an equivalent to a specific list of keywords that are significant for the process or your company's business logic.

For example, FuegoBPM Studio uses four enumerations for process building. *Action* is the enumeration containing the keywords that indicates the state of a Method task as the FuegoBPM Enterprise Server processes it.

When you expand the *Fuego* module, you will find enumerations in some of its modules, as *Action* is in *Lib*. Expand the module and implementation until you can see the Action enumeration.

You can see that the Action variable can only be equivalent to OK, FAIL, RELEASE, CANCEL, REPEAT, ABORT, BACK, or SKIP. If you try to enter anything different in your Method statements, such as Action = GOOD, the Method debugger will deliver an error message. Since GOOD is not a member of the enumeration, it cannot be used with Action.

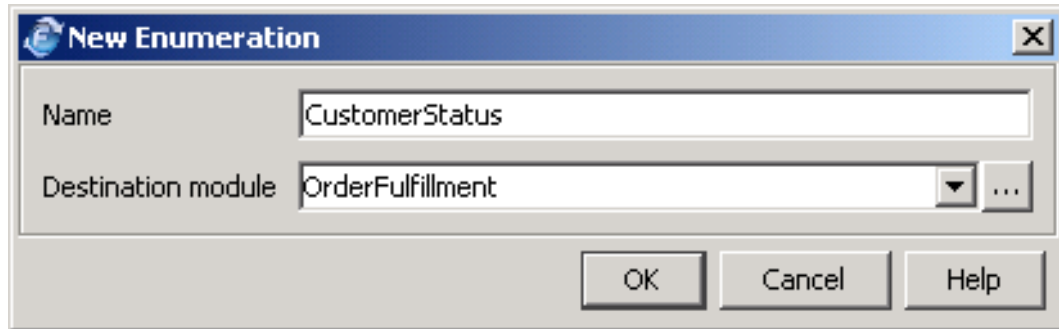
Enumerations make reading programs much easier. The keywords that have been added under an enumeration are truly for human benefit. The Server is really seeing numbers. In the list above, OK = 1, FAIL = 2, RELEASE = 3, CANCEL = 4, REPEAT = 5, ABORT = 6, BACK = 7, and SKIP = 8. However, when you read through a program Action = 1 does not make much sense. However, Action = ABORT is clear from the human perspective.

Adding an enumeration

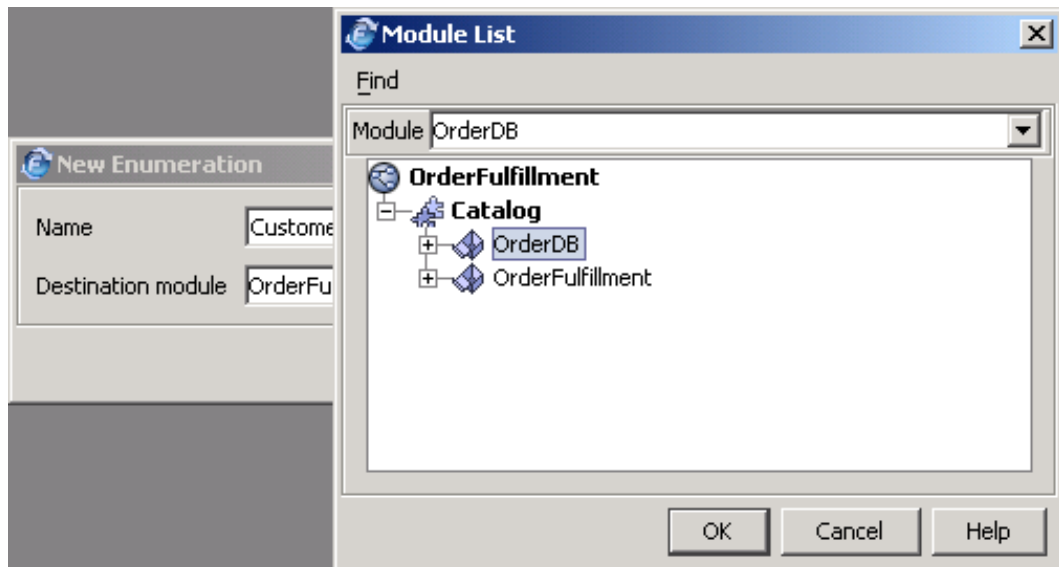
To add an enumeration to the Project Catalog,

1. If you wish, add a new module to the Project Catalog to contain your enumeration. Otherwise, right-click on an existing module and select **New Enumeration** from the shortcut menu.

2. A dialog box appears requesting a name for the enumeration.



3. Enter an **Enumeration name** and click **Ok**. The name of the module from where you begin the creation is displayed in the *Destination Module* field. To change the destination module, browse the Project Catalog by clicking the *three dots* button.



4. A new profile appears in the right panel of the FuegoBPM Studio.

The screenshot shows a window titled "CustomerStatus" with a toolbar containing icons for add, delete, up, and down. The form has two text input fields: "Enumeration name" (containing "CustomerStatus") and "Enumeration description" (empty). Below these is a checked checkbox labeled "Is Sequential". A table with a header "Name" contains four rows: "ACTIVE", "INACTIVE", "COMMERCIAL_ACCEPTANCE_REVISION", and "CREDIT_ACCEPTANCE_REVISION".


Name
ACTIVE
INACTIVE
COMMERCIAL_ACCEPTANCE_REVISION
CREDIT_ACCEPTANCE_REVISION

5. Enter an **Enumeration description**.
6. Select or clear the **Is Sequential** check box. If this option is selected, the keywords you enter will be automatically numbered beginning with zero (0). If the option is disabled, a second column will appear next to the Name column, where you can enter a value that is appropriate, such as a Java error code number.
7. Click the *add* (plus icon) button to add a line.
8. Click inside the line and type in a keyword.
9. Repeat steps 5-6 until you have added all your keywords.
10. Click the **Save** button.

11. To *remove* or *move* an enumeration value, use the icons on the left side of the panel.



Note

 The keywords can be in capital or lowercase letters, depending on what you prefer.

Since the *Is Sequential* check box is checked, these words are automatically numbered: ACTIVE = 0, INACTIVE = 1, COMMERCIAL_ACCEPTANCE_REVISION = 2, and CREDIT_ACCEPTANCE_REVISION = 3.

If you disable the **Is Sequential** check box, a Value column appears and you may enter your own numbers manually.

The screenshot shows a software window titled "CustomerStatus" with a standard Windows-style title bar (minimize, maximize, close buttons). Inside the window, there are two text input fields: "Enumeration name" containing "CustomerStatus" and "Enumeration description" which is empty. Below these fields is a checkbox labeled "Is Sequential" which is currently unchecked. Underneath the checkbox is a toolbar with four icons: a plus sign, a minus sign, an up arrow, and a down arrow. Below the toolbar is a table with two columns: "Name" and "Value". The table contains four rows of data. The first row is "ACTIVE" with value "1". The second row is "INACTIVE" with value "2". The third row is "COMMERCIAL_ACCEPTANCE_REVISION" with value "3". The fourth row is "CREDIT_ACCEPTANCE_REVISION" with value "4", and this row is highlighted with a blue background. Below the table is a large, empty rectangular area.

Name	Value
ACTIVE	1
INACTIVE	2
COMMERCIAL_ACCEPTANCE_REVISION	3
CREDIT_ACCEPTANCE_REVISION	4

Exporting an enumeration

You can export an enumeration to use it in other projects or share it with other developers.

To export an enumeration,

1. Right-click on the enumeration and select the option **Export**.
2. The *Select Object file...* dialog opens. Browse the location in your file system and give a name to the zip file to generate.
3. Click the **Save** button to export the enumeration.

Importing an enumeration

To import an enumeration,

1. Right-click on the module and select the option **Catalogue Component / Enumeration** or from the **File** menu select the option **Import / Enumeration**.
2. The *Module list* dialog opens. Select the module where you want to import the enumeration and click **OK**.
3. The *Select Object file...* dialog opens. Browse the location in your file system where the zip file is located.
4. Click the **Open** button to import the enumeration.

When importing an enumeration into the catalog, it may not be imported into the selected module. It depends on the source version of the enumeration you are importing.

If you are importing a component from a prior version the *zip*, *xcdl* or *xml* file does not include the module information, that is why it is added to the selected module.

Components files of this same version, can be *xcdl* files without module information or *zip* files with module information, that's why the first ones are added to the selected module and the second ones are added to the original module (if the module does not exist it is created).

Using enumerations as Fuego Objects attributes types

You can define attributes in a Fuego Object as an enumeration created in the project catalog. These attributes can be referenced from Fuego Objects presentations from *Combo* and *Radio Button* components.

Using enumerations in a process

Once you have added an enumeration to the Project Catalog, you can use the variable in a process. First, you must add an instance variable to the process, thus making the variable type the name of the enumeration.

To use an enumeration in a process,

1. Open with the Method Editor for the process task to which you want to add the enumeration.
2. Open the **Variables** frame and add the variable to the Instance Variables (if you want the variable to be available process-wide) or Local Variables (only available for the activity).
3. Enter the variable name in the **Name** field. Click **Browse** and use the catalog component browser to browse for the enumeration. Click **Ok**.

Once the variable has been added, it can be used in the Business Process Method script. For example, if *customerStatus* = *INACTIVE*, a method can be called to remove the customer from an active customers database. The Method might look like the following:

```
if customerStatus is INACTIVE then
//call method to remove the customer from the database
removeCustomer CustomerMaintenance
end
```

The removeCustomer method might look like the following:

```
DELETE FROM CUSTOMER
```

```
WHERE custCode= arg.custCode
```

Or it can be implemented in order to remove it from the main database and add it to the historical database.

Cataloging Exceptions

Business (or User) Exceptions are defined by the individual programmer and they are expected to be thrown in a normal execution. Examples of this kind of exception could be: Not Enough Money, Account Is Closed, or Operation Not Authorized.

Possible sources for Business exceptions are:

- User code (BP Method or Fuego Objects). Through the use of the 'throw' statement.
- Cataloged components with their own exceptions (for example, a Web Service with an exception defined).

Note



Business Exceptions must be cataloged in the Project Catalog.

For further information, please refer to Handling Exceptions Using Fuego Blocks.

Adding an Exception to the Project Catalog

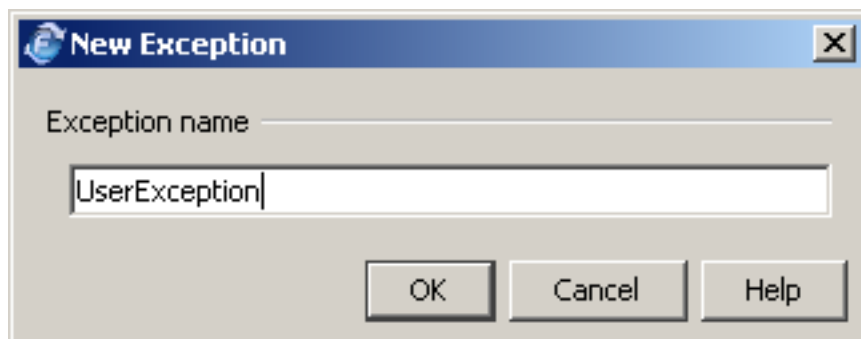
An exception used in a Business Method will not be recognized unless you catalog it in the Project Catalog. Exceptions contained in a jar that you are cataloging are automatically added to the Catalog. You can also create new Exceptions. There are no restrictions on the name you can give to the exception but it is strongly recommended that you include the word Exception in it. When you create a new


exception, it is cataloged as a Fuego Object so that the exception has the same properties and usability benefits as Fuego Objects.

For further details about Fuego Objects, please refer to *Implementing Business Objects*.

To create a new exception in the Project Catalog

1. Right-click on the cataloged module in which you want to catalog the exception and select the *New Exception* option from the menu.
2. Type the name of the exception you are cataloging. The name of the module from which you begin to catalog the exception is copied to the *Destination module* attribute.



3. The newly cataloged exception is added to the module and it is identified with the icon .

Implementing Business Objects Using Fuego Blocks

FuegoBPM Studio supports adapters to common business applications, such as FTP, e-mail, Microsoft Excel, Files, SQL, and so on. These adapters are several standard Java components that are ready to use. They are located in the module **Fuego** in the Project Catalog.

These Fuego Blocks or Fuego standard components are organized in

different groups according to the functionality provided.

Auth

This module bundles security related components. It contains the component SecureStore that is a component to manage user passwords.

Chart

Components to build FuegoBPM Dashboards are contained in this module of the FuegoBPM catalog. These components provide:

- data sources creation to build charts of both XY and XYZ types,
- representation of elements needed to build a graphic, as the type and dimension.

Com

All Automation components inherit from the AutomationObject component contain in this module. This is the same principle as used in the Java programming language. All Java objects inherit all information from the Java Object class.

Corba

This module contains an invokeable class that wraps a CORBA object. All operations performed over these objects are executed through this class.

Dynamic

This module contains a set of components to perform dynamic invocation of Java objects, evaluation of dynamically written boolean expressions and some utility components. Usage of this module's components is discouraged if there is an alternative way to

implement the same functionality.

Ejb

This module contains a component that represents the EJB object home object. This component has to be use for every EJB catalogued into the project's catalog.

Fdi

This module groups all the components that are used to add or update information from the Directory Service.

Io

This module contains different components to manage Files such as Binary files, Client files, Delimited files, Text files, Properties files and others, from activities in your process.

Lang

This module contains components that provide functionality to manipulate:

- time objects as "Day", "Month", "Week", "Interval" of times and "Times" themselves,
- presentable Fuego Objects and the validation exception that occurs when a required expression of a Fuego Object attribute does not check.

Lib

This module groups a great number of components. They mainly provide the way to manage different objects within FuegoBPM for a given instance such as: Activity, Attachments, Events, Calendar Rules, Participant, Process, among others.

Msg

This modules contains components that handle JMS Messages and components that act as listeners of events.

Net

Contains the components that allow you to interact with Java server pages (JSP) and FTP, Telnet, and Web servers through activities in your process.

NetX

Contains advanced web components that allow you to interact with web servers, manage cookies and http responses.

Papi

Contains components that in a high level provide PAPI functionality. PAPI is the Java interface used by client applications that need access to Fuego processes. The applications in the Fuego suite (like FuegoBPM Studio, FuegoBPM Console and FuegoBPM Work Portal) use PAPI to communicate with the process servers.

Sql

This module contains components related to the execution, validation of SQL statements and to catalog SQL tables.

- **DynamicSQL:** can be used to run any valid SQL statement on any database existing as an external resource,
- **BAMQuery:** executes SELECT type queries on the FuegoBPM BAM database,
- **SQLException:** used if you want to perform an action when an error in a SQL statement occurs before exiting the FBL,

- **SQLObject**: object from which every cataloged table inherits.

Ui

This module groups components that facilitates User Interface such as a File Chooser, output information into a Table, and display option in a Menu.

Util

This module contains different components that are useful for different purposes such as, run batch programs, open an URL, run an Ant script, manage a Document, represent a specific geographical, political, or cultural region, among others.

Xml

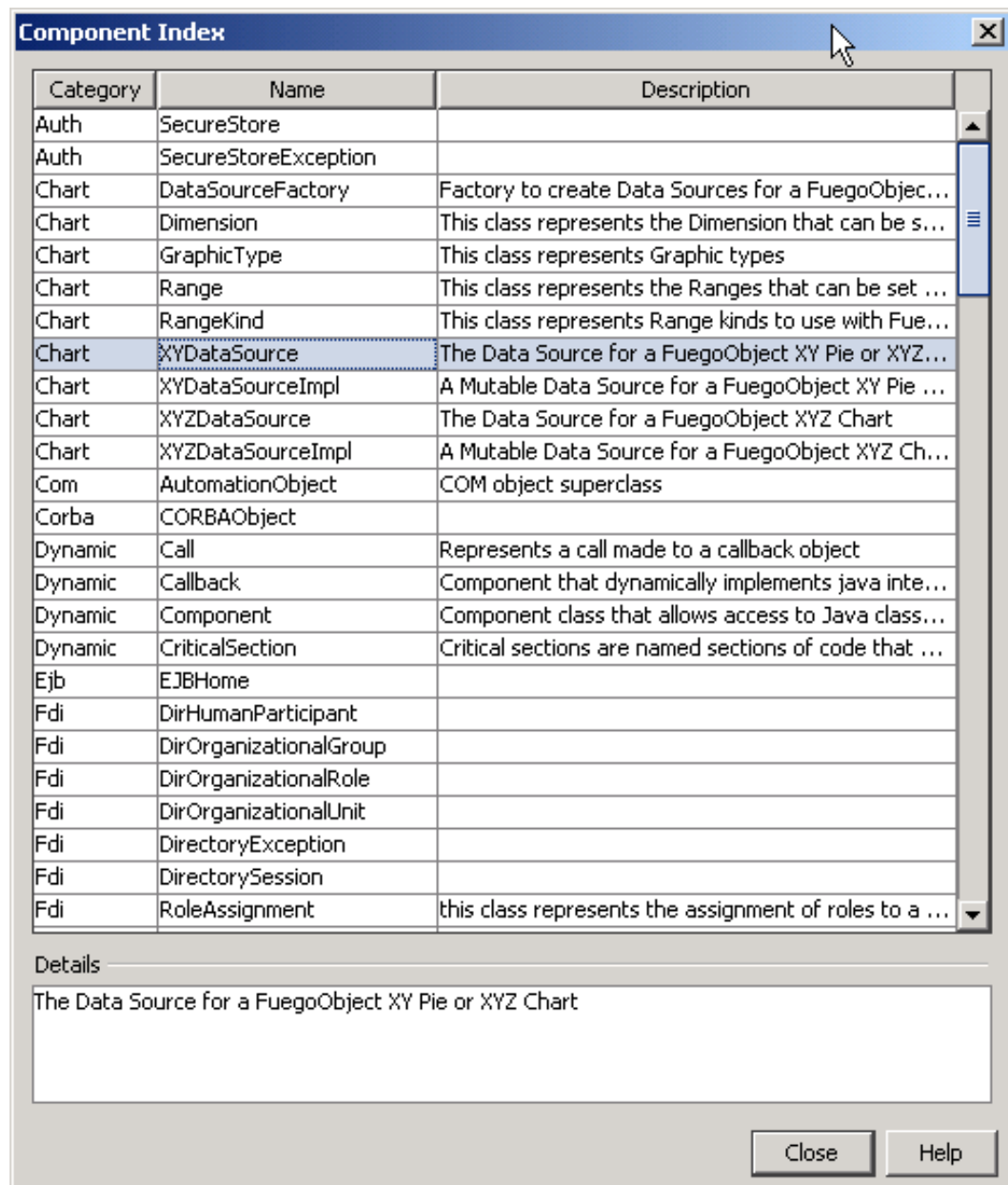
Contains components that allow you to view and manipulate XML documents.

- **DynamicXML**: it serializes/deserializes FObject components in/from an XML string,
- **XMLDocument**: it wraps a DOM tree structure with methods to initialize the tree structure and print it,
- **XMLNode**: it is a wrapper for an element node on a DOM tree structure (used with XMLDocument),
- **XMLException**: it wraps every checked exception thrown in underlying layers of the DOM tree structure when using XMLDocument and XMLNode.
- **XMLObject**: all introspected XML components inherit from it,
- **XPath**: evaluates XPath expressions

Fuego Blocks Components Index

To get the list of all the Fuego Blocks Components,

1. Go to the **Help** menu and select the **Component Index** option. A dialog opens with the list of the components, showing for each one the category or module to which it belongs to, and a brief description about it.



2. If you double click on the component, a new tab in the main panel opens for that component.

Fuego Blocks Documentation

The documentation for each Fuego Block provided is available in FuegoBPM Studio.

To find information about any Fuego Block,

1. Select and open a Fuego Block from the *Fuego* module in the project catalog.
2. The documentation for the Fuego Block is available by opening the *Documentation* tab.
3. The information is provided for the Fuego Block and for each of its methods.

To find information about a method,

1. Select a method in the component structure panel.
2. Change to the tab in the Documentation flap, which now has the name of the selected method.

Documentation and *Use cases* are provided for both, Fuego Blocks, and their methods.

External Resources Configuration

When your process design uses resources other than those provided by FuegoBPM Studio, it is necessary to configure the way in which

FuegoBPM Studio can connect to them.

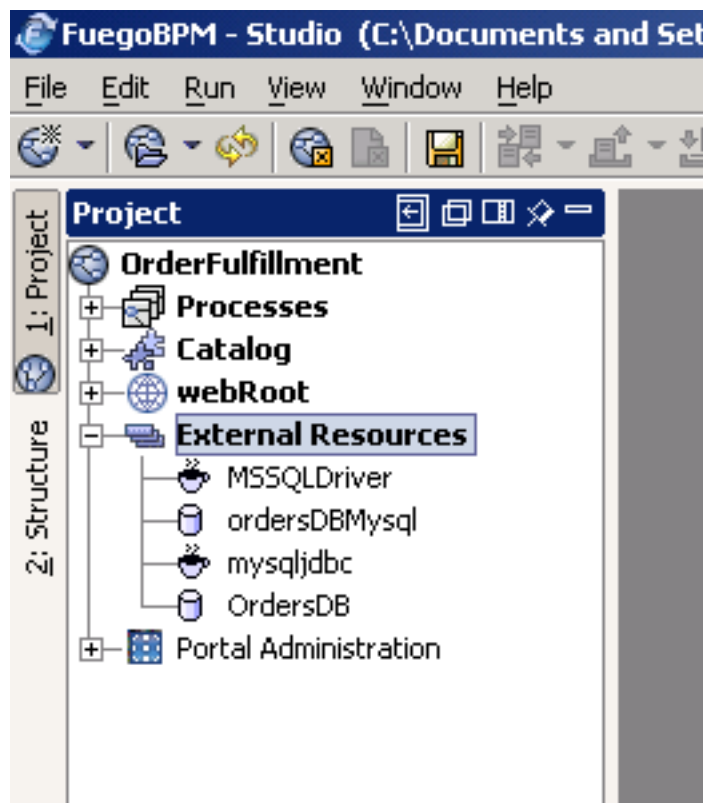
For every project created, **FuegoBPM Studio** creates a directory in which **External Resource** configurations can be stored. Every new **External Resource** configuration is a reusable set of data that allows FuegoBPM Studio to connect to an external resource. The external resources defined here can be used when cataloging components.

External resource types include SQL database, COMBridge, CORBA, Naming and Directory Service, J2EE Application Server, Enterprise JavaBean, Java Class Library, Server, and Web Service.

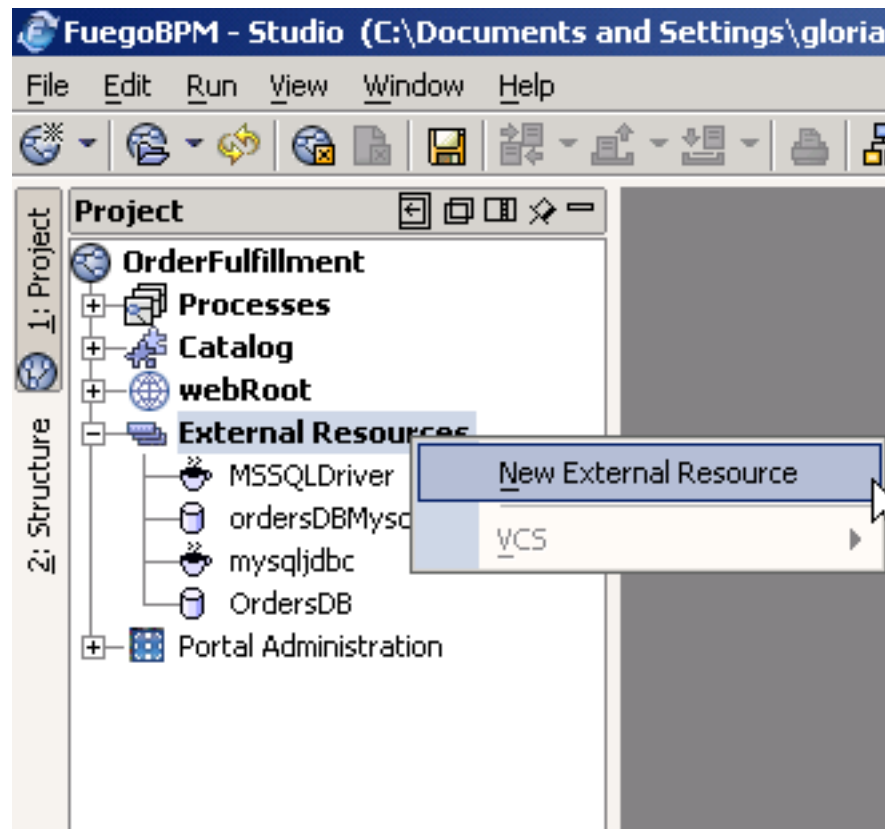
The information needed to create a configuration varies according to the configuration type that has been selected.

To create a new External Resource Configuration

1. Go to your Project **External Resources** directory.



2. Right-click on **External Resources** and select **New External Resource**.



3. The **New External Resource** dialog appears.

New External Resource
Complete the fields to create a new External Resource.

Name: database

Type: SQL database

Supported types: Oracle JDBC

Details

Host: ut-server [PORT] 1521

SID: utfuego

User: system

Password: *****

Driver Type: thin

☐ Advanced

URL: jdbc:oracle:thin:@ut-server:1521:utfuego

OK Cancel Help

4. Enter the configuration name in the **Name** box.
5. Select one of the available types from the **Type** drop-down list.

Type: COMBridge

Supported types:

- CORBA
- Enterprise JavaBean
- J2EE Application Server
- Java Class Library
- Naming And Directory Service
- SQL database
- Server Configuration
- Web Service

Host: localhost

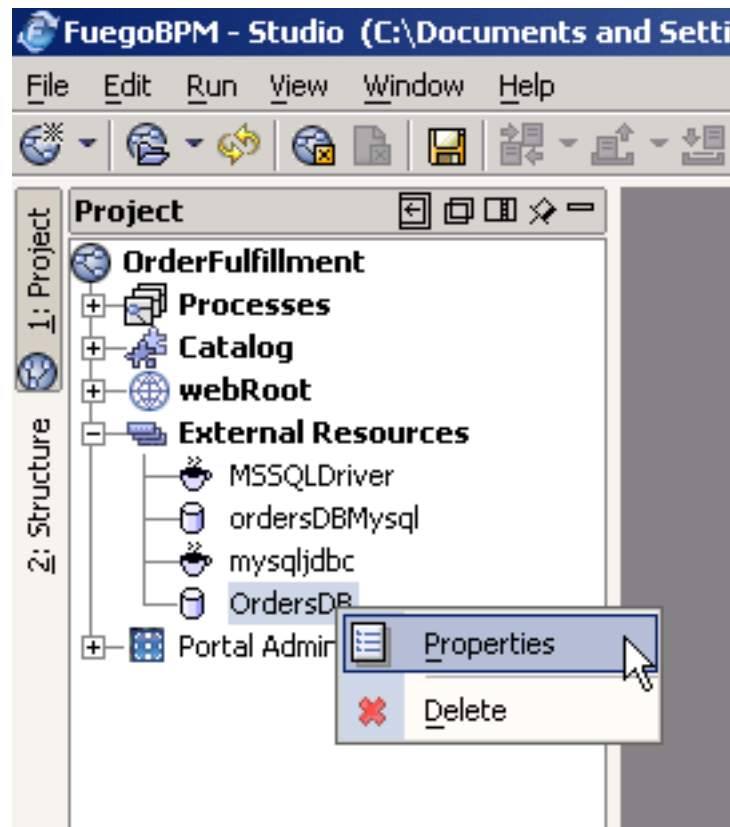
6. The rest of the information required depends on the configuration type.
7. When all the required information is entered, click **Ok**. The

external resource configuration created is added under the **External Resources** directory in your project.

External Resource configurations can be edited at any time. Changes made to the external resource configurations affect the way in which cataloged components that use the edited external resource configuration connect.

To edit an External Resource

1. Go to **External Resources** directory in your project and select the external resource you want to change.
2. Right-click on the external resource and select **Properties** from the shortcut menu.



3. Make all the changes in the **Edit External Resource** dialog and

click **OK** when finished.

Edit External Resource
Modify the values of the External Resource

Name:

Type:

Supported types:

Details

Host: [PORT]

SID:

User:

Password:

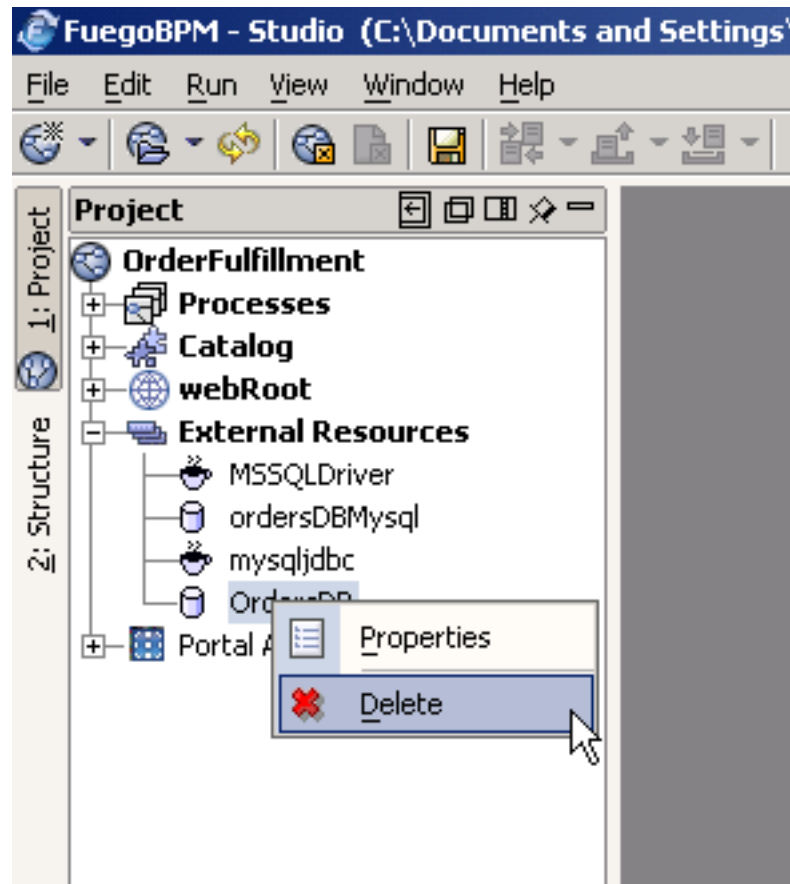
Driver Type:

☐ Advanced

URL:

To delete an External Resource

1. Go to **External Resources** directory in your project and select the external resource you want to delete.
2. Right-click on the selected external resource and select **Delete** from the menu displayed.



Configurations required information

The information required by FuegoBPM Studio to connect to an external resource varies depending on the external resource type you select. Click one of the following types to view a detailed description of the data required in each case.

- **COMBridge**

- **CORBA**

- **Enterprise JavaBean**
- **J2EE Application Server**
- **Java Class Libraries**
- **Naming and Directory Service**
- **SQL Database**
- **Server**
- **Web Service**

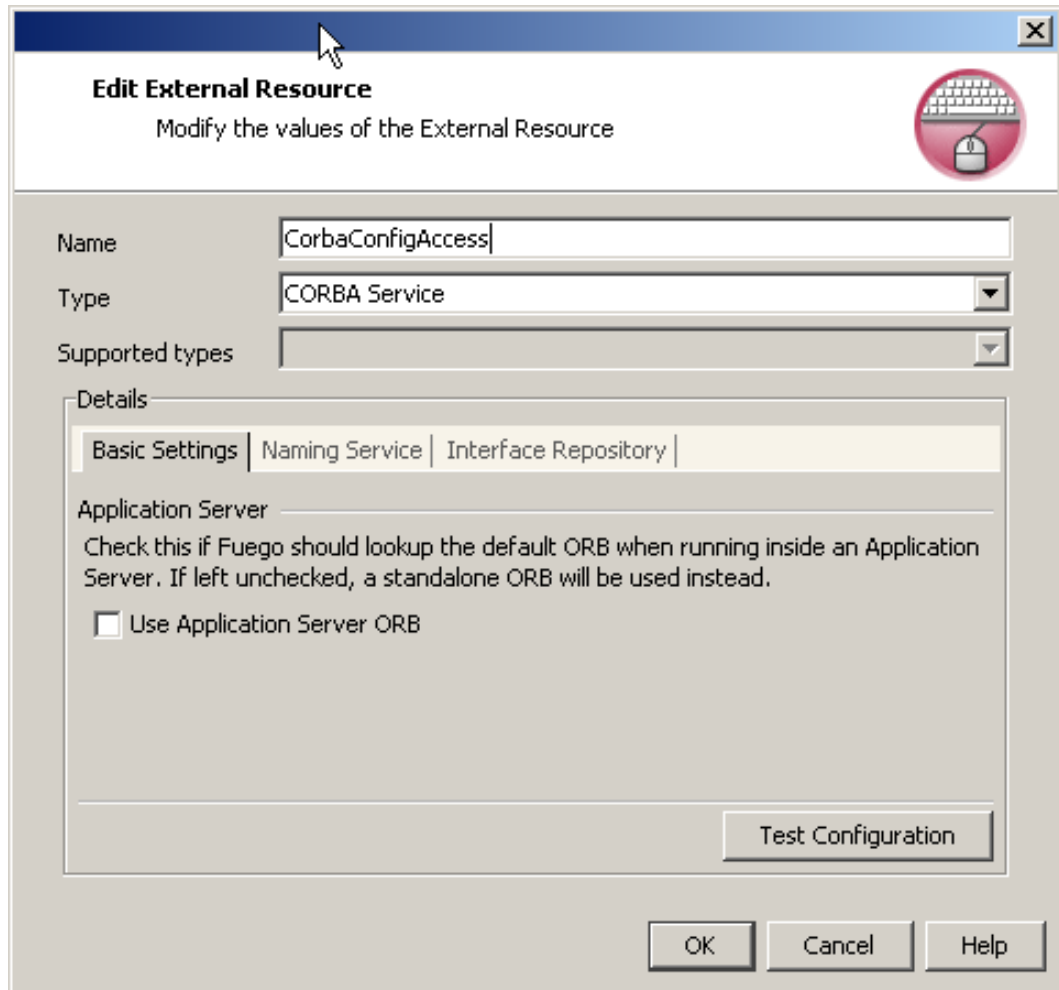
CORBA

FuegoBPM Studio allows you to catalog CORBA objects that reside in an Interface Repository. Once cataloged, you can manipulate the components of the CORBA object in your Method tasks in a process design.

To catalog a CORBA object, you need a configuration of CORBA type. Note that creating a configuration allows you to reuse it each time you need to add new components or to reintrospect existing ones.

To create a CORBA configuration,

1. Right click on the **External Resources** entry of the FuegoBPM Project tree. Select **New External Resource**.
2. Give a name to the configuration you are about to create and select the CORBA type.



3. **Basic Settings**
4. **Naming Service:** There are several options:
 - a. Read IOR from URL: if you have the IOR exported to some service (for example, a web server) and a URL exists that can be used to fetch it, select this option.

- b. Use this IOR: specify the IOR directly (**recommended**).
- c. Resolve Initial Reference: to request the ORB to get the reference of the service trying to resolve its reference. Note that this option is not recommended since it has interop problems when using different ORBs.
- d. Do not use a Naming Service: select this option if you do not need this service. However, remember that objects used in methods are found using the Naming Service. Hence, you will need to use the IOR for any object to be used in a method - passing it as a parameter in its constructor.

Edit External Resource
Modify the values of the External Resource

Name: CorbaConfigAccess

Type: CORBA Service

Supported types:

Details

Basic Settings | **Naming Service** | Interface Repository

☐ Read IOR from URL:

☒ Use this IOR: IOR:010000002800000049444c3a6f6d672e6f72672f436

☐ Resolve Initial Reference

☐ Do not use Naming Service

OK Cancel Help

5. **Interface Repository IOR:** The options are:

- a. Read IOR from URL: if you have the IOR exported to some service (for example, a web server) and a URL exists that can be used to fetch it, select this option.
- b. Use this IOR: specify the IOR directly (**recommended**).
- c. Resolve Initial Reference: to request the ORB to get the reference of the service trying to resolve its reference. Note that this option is not recommended since it has interop problems when using different ORBs.
- d. Use FuegoBPM Studio's Interface Repository: this service can be used when an ORB does not have an implementation of the Interface Repository service. Note that this service must be launched separately.

Edit External Resource
Modify the values of the External Resource

Name: CorbaConfigAccess

Type: CORBA Service

Supported types:

Details

Basic Settings | Naming Service | **Interface Repository**

☐ Read IOR from URL

☒ Use this IOR: IOR:010000001e00000049444c3a4152545f4946522f495

☐ Resolve Initial Reference

☐ Use Fuego Interface Repository

localhost : 1099

OK Cancel Help

SQL Database

The required fields change based on the **Supported Type** selected. The following fields are common to most databases. However, some of the databases might not require all of them to be filled in.

The FuegoBPM Studio platform supports the following databases:

- Cloudscape
- Generic JDBC Version 1
- IBM DB2 AS/400 JDBC
- IBM DB2 JDBC
- IBM DB2 JDBC (Type 2)
- IBM DB2 OS/390 JDBC
- Informix JDBC
- MsSQL JDBC (Microsoft Driver)
- MsSQL JDBC (i-net Driver)
- MySQL JDBC
- Oracle JDBC
- PointBase JDBC
- Postgresql JDBC
- Remote JDBC
- Sybase JDBC

- Sybase SA JDBC

If you are configuring an access to a Server database, please refer to Appendix A - Server database considerations in the System Administration Guide for details.

Cloudscape

FuegoBPM Studio supports Informix Cloudscape, version 3 or higher. Cloudscape is a pure Java RDBMS, so it has built-in JDBC support.

The Cloudscape database settings are as follows:

- **Schema** - The schema of the database is created automatically. The name cannot be changed.
- **Host** - This field cannot be modified. The default is that of the local host.
- **Database** - The full name of the database and the directory where it will be created.
- **Administrator User** - The user name to access the database.
- **AdminPassword** - The administrator password to access the database.

IBM DB2

The IBM DB2 JDBC Driver is available from IBM Corporation as part of the IBM DB2 database.

When creating a configuration to access this database type you must specify the following:

- The database host.

- The port where the database server is listening.
- The database name.
- The database user and password. This user should exist in DB2 and have sufficient rights to create schema and tables, which are used to store information.

IBM DB2 AS/400 JDBC and IBM DB2 JDBC

The IBM DB2 database settings are as follows:

- **Host** - The host name of the machine or server where the database resides.
- **Port** - The port number where the database resides.
- **Schema** - The schema of the database is created automatically. The name cannot be changed.
- **User** - The user name to access the database.
- **Password** - The user's password to access the database.
- **Database** - The name of the database.
- **Administrator user** - The administrator user name to modify the database.

Note



The user and password must exist in the DB2 database. DB2 does not permit you to create users externally.

IBM DB2 JDBC (Type 2) and IBM DB2 OS/390 JDBC

The IBM DB2 database settings are as follows:

- **Schema** - The schema of the database is created automatically. The name cannot be changed.
- **User** - The user name to access the database.
- **Password** - The user's password to access the database.
- **Database** - The name of the database.
- **Administrator user** - The administrator user name to modify the database.

Informix

The Informix database settings are as follows:

On the Basic tab

- **Host** - The host name of the machine or server where the database resides.
- **Port** - The port number where the database resides.
- **Database** - The name of the database.
- **Server** - The server name.
- **User** - The user name to access the database.
- **Password** - The user's password to access the database.

On the Advanced tab

- **Root dbspace name** - The Root database space name.

Microsoft SQL Database)

The MS SQL database settings are as follows. Using a JDBC driver is optional.

- **Schema** - The schema of the database is created automatically. This field cannot be modified.
- **Host** - The host name of the machine or the server where MS SQL resides.
- **Port** - The port number where database resides.
- **User** - The user name to access the database. This field cannot be modified.
- **Password** - The administrator password to access the database.
- **Administrator user** - The administrator user name for the database.

MS SQL (i-net Driver)

The MS SQL database settings are as follows. Using a JDBC driver is optional.

- **Host** - The host name of the machine or the server where MS SQL resides.
- **Port** - The port number where database resides.
- **Database** - Database to be accessed.
- **User** - The user name to access the database.
- **Password** - The administrator password to access the database.

- **Use sql7 & Use cursors always** - check this options if you want to enable the *sql7* and/or *useCursorsAlways* properties of the driver.

Troubleshooting the driver

If you are having any of the problems below, see how the *sql7* and *useCursorsAlways* properties have to be set.

Temporary tables: usage of temporary tables, such as *#table-name*, is broken with the *sql7* property. You may get an exception like:

```
[DBS90011]Invalid object name '#table-name'.  
NextException:  
[DBS90011]Statement(s) could not be prepared.  
  
fuego.components.SQLException:  
[DBS90011]Invalid object name '#table-name'.  
NextException:  
[DBS90011]Statement(s) could not be prepared.  
.....
```

To avoid this problem, set the **Use sql7** property to **false**.

Introspecting a table: if you get the following message when you are introspecting a table:

```
Unicode data in a Unicode-only collation or ntext data  
cannot be sent to clients using DB Library (such as ISQL)  
or ODBC version 3.7 or earlier
```

It may be cause because the table has fields typify as:

- tinyint,
- smallint

- bigint
- money
- smallmoney
- bit
- cursor
- sql_variant
- table
- timestamp
- uniqueidentifier
- text
- ntext
- image
- nchar[(n)]
- nvarchar[(n)]

In that case, the driver provider recommends to set the *sql7* property of the driver to true. To do so, check the *Use sql7* property when defining the access configuration.

ClassCastException and SQLException: No result sets were produced by 'SELECT...': if you get any of these exceptions, set the *Use Cursors Always* to false.

MySql JDBC

The MySQL database settings are as follows:

- **Host** - The host name of the machine or server where the database resides.
- **Port** - The port number where the database resides.
- **Database** - The name of the database.
- **User** - The user name to access the database.
- **Password** - The user's password to access the database.

Oracle

The Oracle database settings are as follows:

- **Host** - The host name of the machine or server where Oracle resides.
- **Port** - Default port number for Oracle.
- **User** - The user name set up by the Oracle system administrator. (Automatically created when the Server is created.) Each Server is related to an Oracle user. The user name in the Oracle instance is preceded by the prefix *fldb_* and ends with the name of the Server. The user name is defined in the Execution Console when the Server is created. Once defined, the name cannot be changed because other FuegoBPM Suite applications reference it.
- **Password** - The administrative password for the Oracle database.
- **SID** - System identification for database; also used to connect to database. Sometimes called Oracle ID.
- **Schema** - if you give a name of a schema, the configuration and introspection will only work on tables of that schema. If you don't give a schema name the schema could be changed in runtime. That is for example a table referenced like `devel.invoice` in your

development environment. If in production you had a different schema name, suppose production references to devel.invoice would work only if you didn't give a name to the schema.

- **Driver type** - The type of drive that Oracle uses (thin or oci8).
- **Advanced** - See below.

On the Advanced tab

- **Tablespace** - Some database administrators divide databases into tablespaces to control and maintain table sizes. If your company uses tablespace names, enter the appropriate name here. Leave the field blank if there are no tablespaces and a default tablespace will be created. When the user name is created in the Execution Console, the user creation statement reference tablespaces. Consequently, it is necessary to define the tablespaces.
- **Temporary Tablespace** - Enter the appropriate temporary tablespace name here. This field is going to be used by FuegoBPM Server's database to perform temporary indexing for some access. TEMP of type TEMPORARY. This tablespace performs temporary operations for the Server.
- **Administrator user** - Enter the administrator user name for the Oracle database. This is the user name that will be used to create the Server's database user in the Oracle instance. The password will be required at user creation time.
- **Profile** - A profile is a set of limits on database resources. If you assign the profile to a user, that user cannot exceed the established limits in the profile. This allows the administrator to limit the actions of a particular Oracle user. The Oracle administrator may have different profiles set for different groups of users so that there is control over what each group is

authorized to use and over which resources from the database a particular group will have.

- **Use Timestamp for Date columns** - When you select this property it makes the DATE fields work like TIMESTAMP. If the property is not selected, the DATE SQL fields type store only the day and the TIMESTAMP field type store both, day and time.

PointBase JDBC

The PointBase database settings are as follows:

- **Host** - The host name of the machine or server where the database resides.
- **Port** - The port number where the database resides.
- **Database** - The name of the database.
- **User** - The user name to access the database.
- **Password** - The user's password to access the database.

Postgresql JDBC

The MySQL database settings are as follows:

- **Host** - The host name of the machine or server where the database resides.
- **Port** - The port number where the database resides.
- **Database String** - The name of the database.
- **User** - The user name to access the database.

- **Password** - The user's password to access the database.


Remote JDBC

If you need to configure XA compliant SQL components you have to integrate a Remote JDBC external resource.

This is implemented when you want your J2EE application server to handle and administrate the database connections. If this is the case, configure first the J2EE external resource to access to the Application Server, see J2EE Application Server. And then a Remote JDBC external resources giving:

- **Database Type** - The database sub-type, which type of database you will be connecting to.
- **J2EE Configuration** - The J2EE configuration previously defined, to access to the Application Server that will manage the connections to the databases.
- **Lookup Name** - The JNDI lookup name that the connection to the database was given in the Application Server.

Note

 if you are in FuegoBPM Studio environment, define the J2EE configuration as a GENERIC J2EE, as it is external to the FuegoBPM Studio. If you are working in a FuegoBPM Enterprise J2EE, that is deployed in the same J2EE application server that handles the connections to the database, remember to redefine the external resource to access to it into LOCAL, as it is the same application server.

Sybase

The Sybase database settings are as follows:

- **Schema** - The schema of the database is created automatically. This field cannot be modified.
- **Host** - The host name of the machine or the server where Sybase resides.
- **Port** - The port number where database resides.
- **User** - The user name to access the database. This field cannot be modified.
- **Password** - The administrator password to access the database.
- **Administrator user** - The administrator user name for the database.

Generic JDBC Version 1

It is possible, for those database not directly supported, to use a generic JDBC if the corresponding drivers are available.

The Generic JDBC Version 1 database settings are as follows:

- **JDBC Driver** - Java class that registers the JDBC driver.
- **URL** - The JDBC connection URL.
- **User** - The user name to access the database.
- **Password** - The administrator password to access the database.

For example, if you want to access to a Cloudscape 5.0 database, enter the following information :

JDBC Driver=com.ibm.db2j.jdbc.DB2jDriver

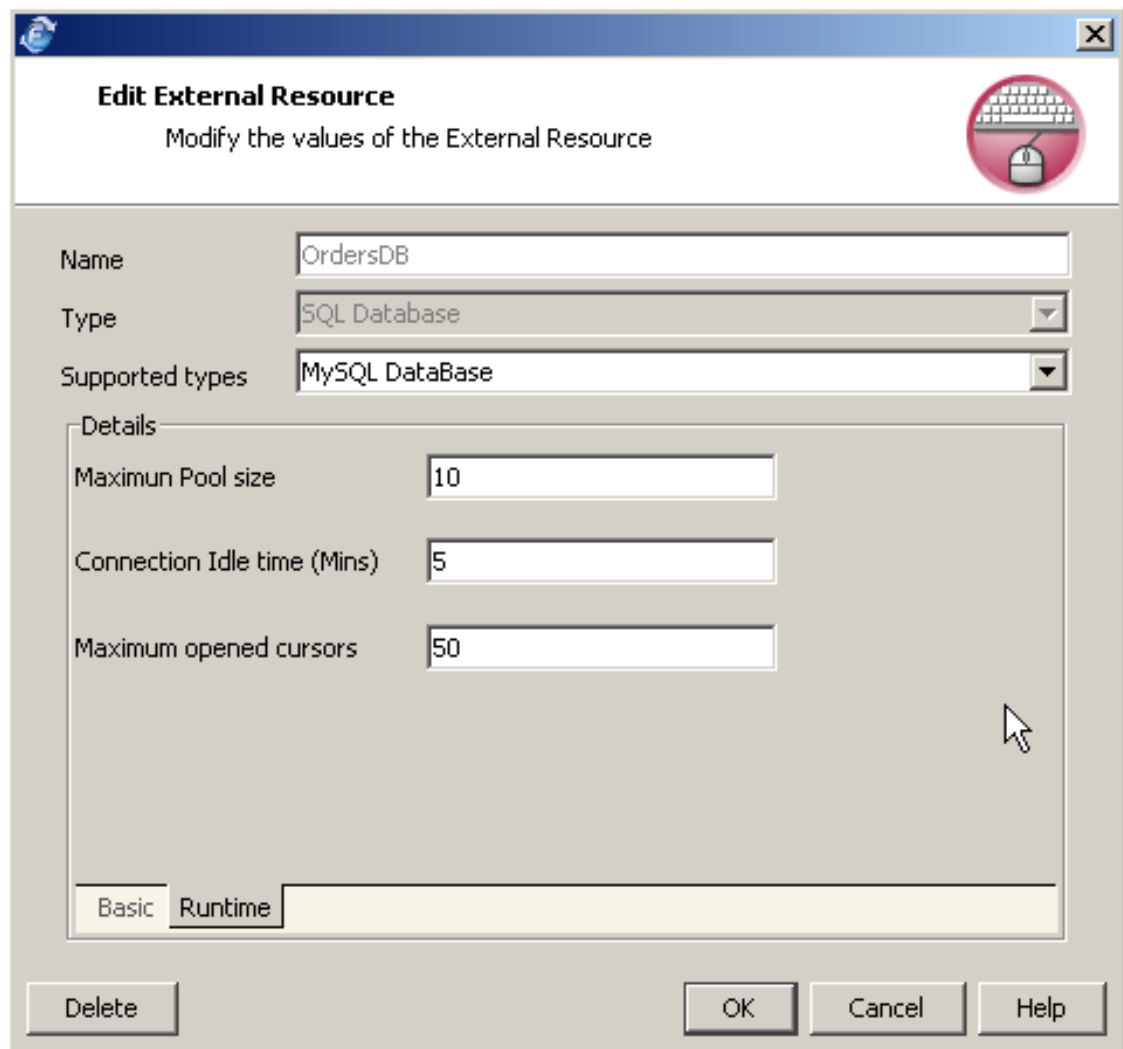
URL=jdbc:db2j:PATH

In addition to that, you will need to catalog the corresponding JDBC drive: *db2j.jar*

Runtime Configuration

Additionally, you can configure runtime properties.

In FuegoBPM Studio, select the runtime tab:



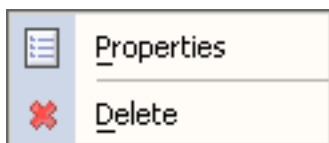
In the FuegoBPM Web Console, refer to the runtime section:

Runtime	
Maximum Pool size	<input type="text" value="10"/>
Connection Idle time (Mins)	<input type="text" value="5"/>
Maximum opened cursors	<input type="text" value="50"/>

- **Maximum pool size:** define the maximum pool size for the SQL configurations.
- **Connection idle time (mins):** The connection is closed after the defined time.
- **Maximum opened cursors:** define the maximum number of opened cursors specified in the data base. This value is related to the maximum pool size. The number of cursors is divided in between the number of maximum pool size and each connection will manage that number of cursors. For example if you have 500 maximum opened cursors and the maximum pool size is 50, therefore each connection can have maximum 10 opened cursors.

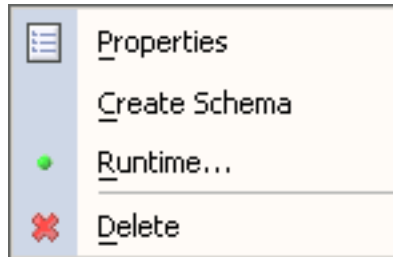
Actions on SQL External Resources

By right-clicking on the external resource a menu is displayed:



- **Properties:** To open the external resource definition dialog
- **Delete :** To delete the external resource.

When the database type of the SQL External Resource is *Cloudscape* two other options:



These two options provide an easily way to build a project with a database environment to test, for example when you don't have connectivity.

- **Create Schema:** To create a cloudscape database in the specified path.
- **Runtime ...:** To browse and select a file ".sql" with sql senteces to execute on a database, as *create schema...*, "insert into...", etc.

COMBridge

COM components are software programs that use Microsoft Component Object Model (COM) technology. COMBridge is a Windows application that acts as a "bridge" between FuegoBPM Suite applications and COM. It provides all the necessary services to introspect and use COM components.

- **Host** - indicates the host where the COM Bridge has been installed.
- **Port** - the port where COM Bridge is located. The default port is '4042'.

Naming and Directory Service

Java Naming and Directory Interface (JNDI) configurations serve as an interface to your process. It interfaces with directory services to retrieve, delete, and add objects stored in the directory service.

- **Initial Context Factory** - the initial context factory can be `com.sun.jndi ldap.LdapCtxFactory` or your own Context factory.
- **url** - the URL you use to connect to the directory service. For Netscape iPlanet directory services, the format is generally as follows: `ldap://host:port/o=company,c=country`. For Microsoft Active Directory, the format is generally as follows: `ldap://host:port/dc=subdomain,dc=domain,dc=com`.
- **Principal** - the root distinguished name for the directory service. For example, Netscape iPlanet directory services usually use `cn=root`. Microsoft Active Directory users usually just enter *root*.
- **Credentials** - your password for the directory service in the **Credentials** field.

J2EE Application Server

J2EE Application Server configurations are required to create Enterprise JavaBeans components.

Two types are supported to define a J2EE Application Server external resource, select one of them depending on your environment:

- **GENERIC J2EE**: This type is used when a process needs to connect to a different Application Server than the one where the process is deployed and running. In case the process is deployed in a FuegoBPM Standalone Enterprise Server, the Generic J2EE is used to connect to the specified App. Server

- **Local J2EE:** This type is used when the process is deployed in a J2EE environment, and the resources are located in the same Application Server.

Indicate:

- **Initial Context Factory** - the class name of the JNDI initial context factory used to access the naming service where the home objects are deployed
- **URL** - the URL of the naming service provider
- **Principal & Credentials** - user and password that will be used at connection time

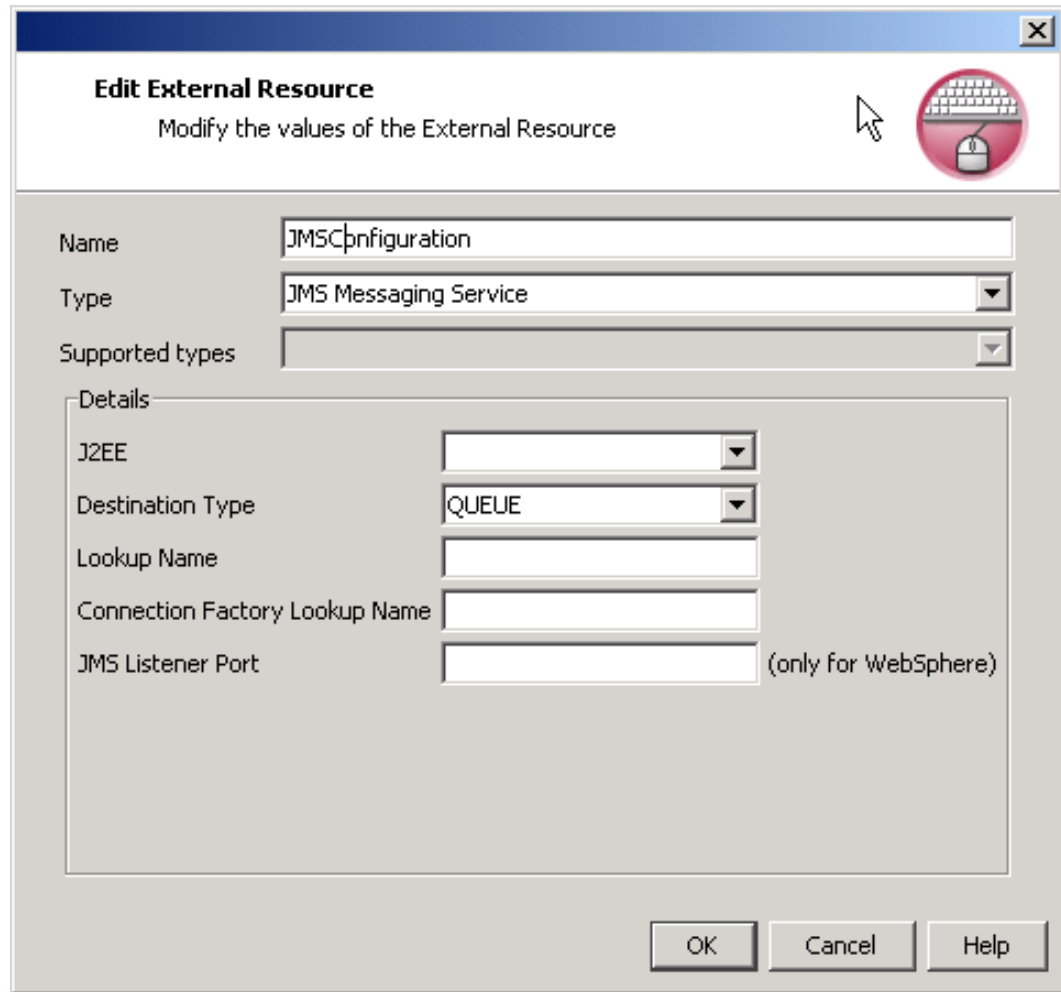
JMS Messaging Service

To catalog a JMS Messaging Service configuration,

1. Type the configuration name, select the type *JMS Messaging Service*.
2. *J2EE:* JMS Messaging service runs on an application server, indicate here the configuration to access it. If the case is that the FuegoBPM Server is an EJB based, and the JMS will run in the same application server that the server, indicate here the already existing configuration defined for the server, as shown in the example picture below.
3. *Destination Type:* Indicate the type of JMS Service, **Topic** or **Queue**.
4. *Lookup Name:* The lookup name of the service in the application service.
5. *Connection Factory Lookup Name:* the connection factory for the JMS

service.

6. *JMS Listener Port* the JMS listener port, configurable only for WebSphere.



Edit External Resource
Modify the values of the External Resource

Name: JMSPnfiguration

Type: JMS Messaging Service

Supported types:

Details

J2EE: [dropdown]

Destination Type: QUEUE

Lookup Name: [text field]

Connection Factory Lookup Name: [text field]

JMS Listener Port: [text field] (only for WebSphere)

OK Cancel Help

Enterprise JavaBean

The portability and reusability of Enterprise JavaBeans (EJBs) make them very attractive to use in a business process to link underlying back-end systems to the business process. In order to be able to catalog an EJB, you must first catalog the configuration to access the Application Server.

- **J2EE Connector** - the configuration to connect to the Application Server
- **Lookup Name** - the name of the EJB

Java Class Libraries

An external library is a file that contains Java classes and resources that are required in your project, typically jar or zip files. External libraries are stored in the project directory structure under the lib directory and administered via the Version Control System (VCS).

What are the benefits of having external libraries associated by project?

Having external libraries associated by project allows the user to work with different versions of the same component using two different projects.

Other key benefits are the memory usage and the improvement of start time. Why load one hundred components when you need only ten for the actual project?

How do the Java class libraries work?

Once you have created a Java class library (JCL), the files defined in the configuration are made available for the project, meaning that the files are added to the project classpath. Removing files from a JCL may require you to reload the project because some classes of the file may still be in use.

Files that are in the project lib directory but are not used in any JCL are not added to the project classpath.

Tips on how to create different JCL

You may classify your library files by category, such as JDBC drivers, EJB files, and custom Java classes in different JCLs, to provide a

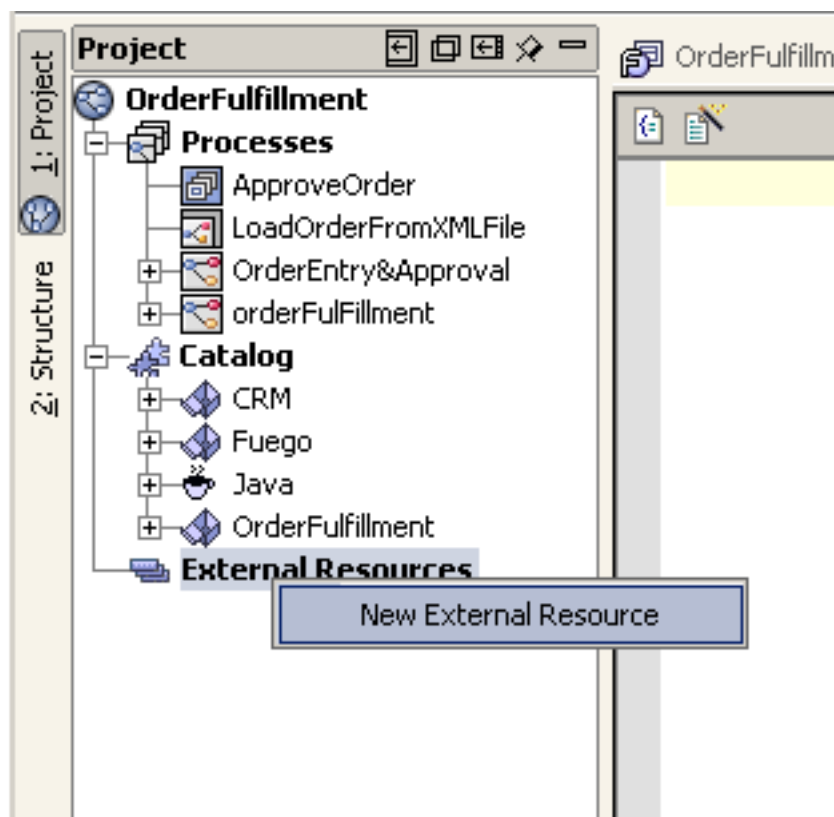
logical separation of your files. Nevertheless, you may create only one JCL with all the files you require.

How can I add other external resources needed in my project?

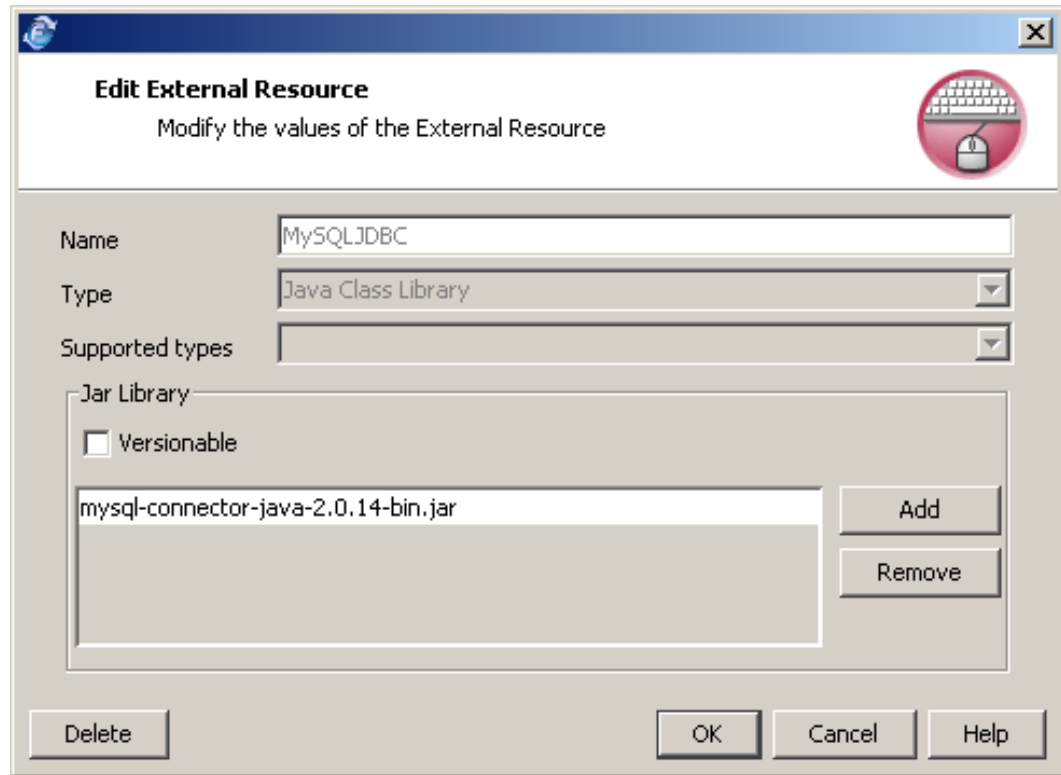
When you catalog an external resource, the files are automatically copied to the project *lib* directory if they are located anywhere else.

To add an external resource to your project,

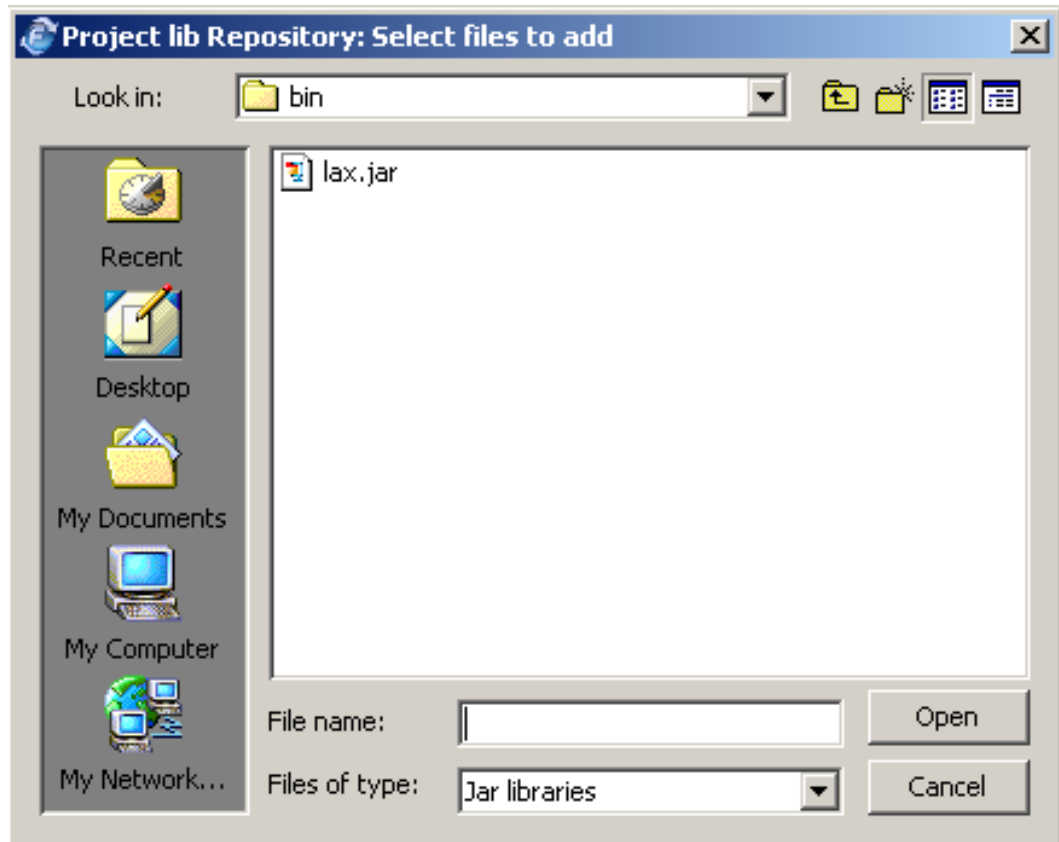
1. Right-click on the **External Resources** entry in the navigation tree in the Project panel and select the **New External Resource** option.



2. The next wizard step is displayed. Give a name to the configuration you are creating and select *Java Class Library* as type.



3. Click the **Add** button to begin adding files. The **Project lib Repository: Select Files to add** dialog is displayed. Browse the directory system and select the files you want to add. Click the **Open** button to continue.



4. Click the check box on the left of the jar file name in the main dialog to include it in the external resource you are generating.
5. Indicate if the library is versionable or not by checking the *Versionable* box. See detailed explanation for this typification in next section **Versionable and Non-versionable libraries**
6. Repeat this sequence for all the files you want to add to the resource and click **OK** when finished.
7. When you finish, there will be a new entry in the External resources entry of the project tree.



If you go to the *lib* directory under the project directories structure, you will notice that the zip file has been automatically copied.

Other Actions

- Click the **Remove** button to remove a jar file from the external resource.
- Click the **Delete** button to remove the external resource configuration.

Versionable and Non-versionable libraries

Versionable libraries are those that may change as processes evolve. Consequently, it is required that each version of the process to be tied to a specific version of the library.

Libraries should be tagged as versionable when the goal is to prevent an update to the library from affecting the behavior of an old version of the process or of a different process that depends on the same components. They could be thought of as integral pieces of a project in the same way the component catalog or processes are.

Non-versionable libraries, on the other hand, are those that don't typically change over time. Or, if they do, the intention is for them to affect all versions of all processes deployed in the server . In contrast with versionable libraries, these can be thought of as infrastructure components or extensions to Fuego needed to support the execution of processes.

Adding a JDBC driver as a JCL external resource

JDBC drivers added as JCLs external resources **have to** be configured as a **Non-Versionable** library.

VCS and Java Class Libraries catalogued as External Resources

Please refer to the Version Control System topic to find detailed information.

Java classes: Endorsed Standards Override Mechanism

An endorsed standard is a Java API defined through a standards process other than the Java Community Process (JCP).

In order to take advantage of new revisions to endorsed standards, developers and software vendors may use the *Endorsed Standards Override Mechanism* to provide newer versions of an endorsed standard than those included in the java Platform.

Classes implementing newer versions of endorsed standards should be placed in JAR files.

According to the override mechanism implementation, you could set the system property *java.endorsed.dirs*, that specifies one or more directories that the Java runtime environment will search for such JAR files. If no value is set for *java.endorsed.dirs*, then JAR files are looked in a default standard location:

- `java-home\\lib\\endorsed` [Microsoft Windows]
- `java-home/lib/endorsed` [Unix]

Here *java-home* refers to the directory where the runtime software is installed.

Endorsed Standars Override MEchanism in FuegoBPM Studio

If you have to use standard java class libraries that have been redefined in FuegoBPM Studio, you must implement the endorsed

override mechanism.

In FuegoBPM Studio copy the redefined classes to the *studio_home/jre/lib/endorsed* directory, where *studio_home* is the top-level directory where FuegoBPM Studio is installed.

For example, if you have to use the CORBA classes redefined in the *visibroker.jar* file, copy this file to the *jre/lib/endorsed* directory under your FuegoBPM Studio top-level installation directory.

Server

This type of external resource is used to configure access to a server.

A Server configuration is created automatically when a web services is introspected. But, if a server configuration with the same characteristics already existed when introspecting, instead of creating a new one, the existing configuration is used.

- **URL:** Enter the URL of web service provider, including **port** if you want.

Web Service

Web services configurations are required to create Web services components. A *web service* configuration requires a *Server configuration*. When you introspect a web service a *Web service* configuration is automatically created, so generally you would not required to create this type of configuration manually. But edit it to, for example, change the development server to the production one.

- **Server Configuration:** Server configuration to access the web service provider.
- **Path:** Path where the Web Service Description Language (WSDL)

file is located.

Chapter 16. Project Portal Administration

Project Portal Administration

The **Project Portal Administration** entry has two elements to work on.

- **Views**, to define views according to the functional requirements.
- **Presentations**, to generate new presentations to relate to the future generated views.



Default Views and Presentations

Default Views

Default views are not included in this step of definition. Default views are generated when the project is deployed and can only be edited and visualized from the FuegoBPM Portal Console. They cannot be visualized when defining the project in the FuegoBPM Studio.

To know more about default views refer to the FuegoBPM Portal Console documentation (included in the FuegoBPM Studio documentation as well) or the FuegoBPM System Administration Guide.

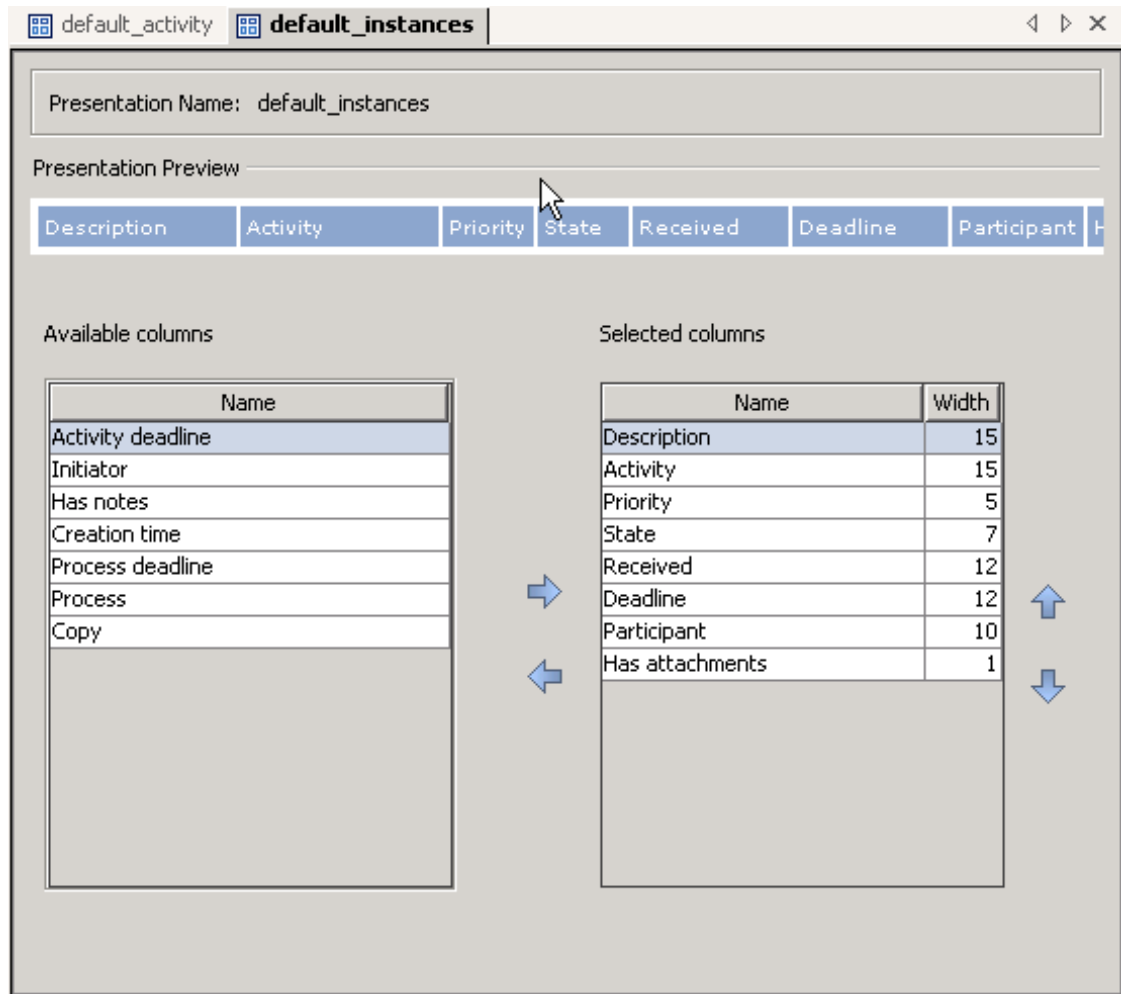
Default Presentations

Presentations are the layout with which a view will be displayed.

FuegoBPM always provides two default presentations: *default_instances* and *default_activity*.

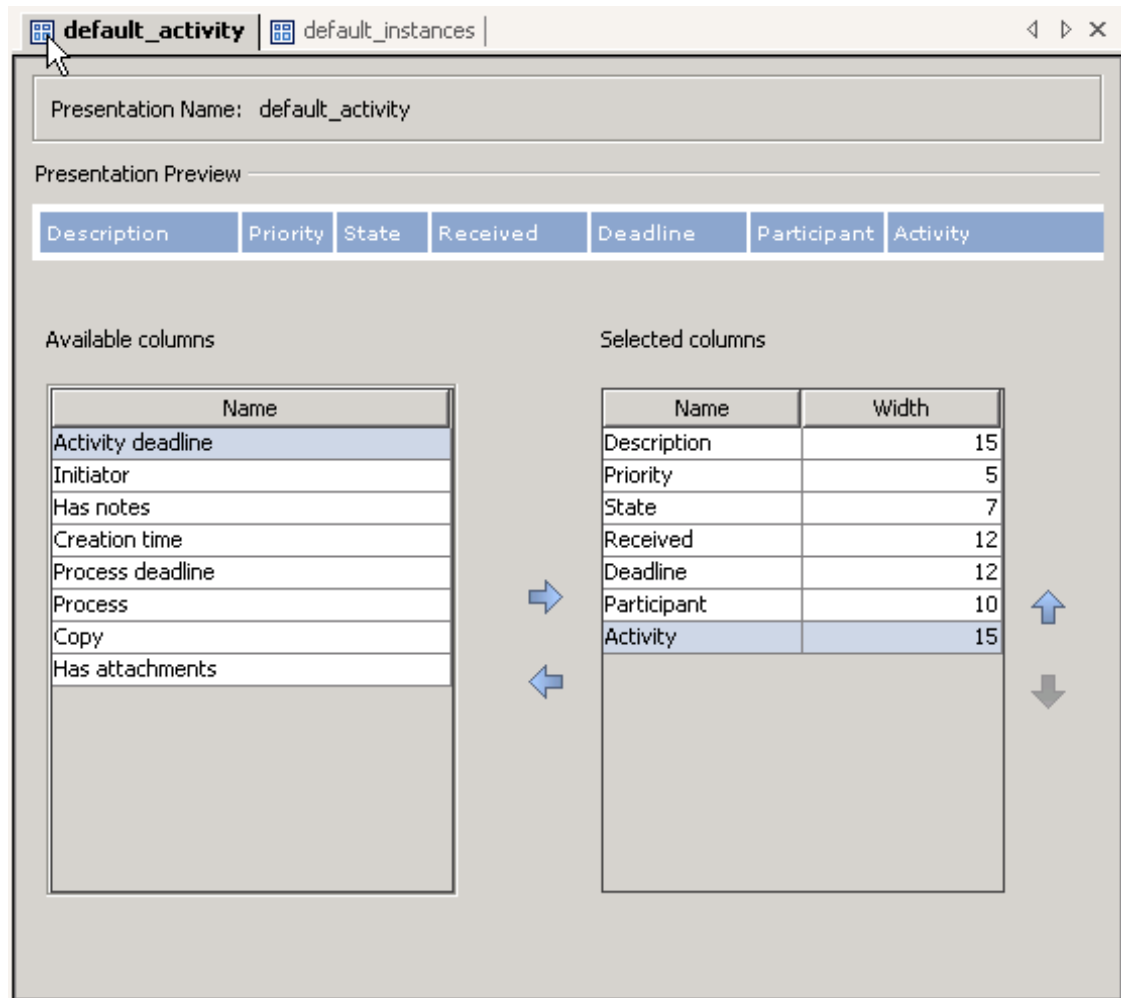
- *default_instances*, this presentation is proper to display views that correspond to a process level. The activity column shows in which activity the process instance is at a certain time. This makes sense only if the view is a process level one, that is to say, several instances of one or more process, are included in the view. This information is not relevant for an activity level view because the activity will always be the same for all the rows displayed in the presentation.

The picture below shows the default presentation columns and the available ones to modify it if required.



- *default_activity*, this presentation is proper to display activity level views. The difference with default instances_presentations is that it does not include the activity column.

The picture below shows the default presentation columns and the available ones to modify it if required.



Editing a default presentation

Although these presentations have a default set of columns, they can be modified according to the needs.

See [Edit a presentation](#)

Deleting a default presentation

Default presentations cannot be deleted completely. If you delete a default presentation, the next time you open the project the default presentation will be restored to its original configuration. A deleted default presentation will be published with its default configuration.

Custom Views

There are five different types of views:

- **Instances** (Process level)
- **Activity Instances**
- **Applications**
- **Attachments**
- **Folders**

To find out more information about the different views types please refer to the FuegoBPM Portal Console documentation (included in the FuegoBPM Studio documentation as well) or the FuegoBPM System Administration Guide.

Views can have a parent view or be top level. In the Work Portal nested views are shown when clicking over their parent. Views can be nested indefinitely.

The parent view is defined when creating the view. By default the creation dialog has no parent view selected. You can select a parent view from the combo box shown in the creation dialog.

Parent views are defined with the single purpose of defining how views will be grouped when showing them to the user. They do not define inheritance neither they group views under a congruent definition (i.e. group them by process, conditions, or external variables values).



There are no restrictions in the type of view to select as parent view. For example there would be no problem in setting an application view as the parent of an instances view.

Creating a View

1. Right-click on the **Views** category. Select the **Create view** option from the menu. The **Create view** dialog opens.
2. Type the view *Id*.
3. Select a *parent view* from the combo box, or select *no parent* if the view is a top level one. The list displayed in the combo box contains all the existing views.
4. Select the view type, *Applications*, *Attachments*, *Instances*, *Activity Instances* or *Folders*, and click **OK** to create the view. The new view will open in a new tab.

Editing a View

Note

 If the  icon is displayed next to the view you want to edit, this means that the view is read-only and you cannot modify it.



1. Double click on the view, or right-click and select **Open** to open the view. A new tab with the selected view will open.
2. According to the type of the edited view you will be able to do some or all of the following changes:
 - assign a process
 - assign an activity
 - assign a role
 - change view properties
 - add, edit or remove labels

- select a presentation to display the view
- create a filter

Please see Views Attributes section to learn more about how to configure a view.

Deleting a View

Note

 If the  icon is displayed next to the view you want to delete, this means that the view is read-only and you cannot modify it.

To delete a view

1. Double click on the view or right-click and select **Delete** to remove the selected view. A confirmation dialog will appear.
2. Click **Yes** to delete the presentation.

To remove views from the FuegoBPM Enterprise once that the project has been deployed, without undeploying it first, you must use the FuegoBPM Portal Console and remove it manually.

For example, let's say there is an instance view called "View1" in the project. The project is published and deployed in Enterprise. Then later you decide to remove "View1" in Studio and you create a "View2". Then updated project is published and deployed in Enterprise without undeploying. If you don't manually delete the View1 via the Enterprise Portal Console, the view will still be displayed in the work portal. The user will see View1 and View2.

Views Attributes

Attribute: Processes

Description:Processes included in the view.

View Type:All, except Folder

Attribute: Activity

Description:Activity included in the view.

View Type:Activity Instances

Attribute: Roles

Description: List of roles available to assign to the view. Default views may have no roles assigned when they were generated from designer without the Generate Views checked or if the generation of the views where no run from the Execution Console. In this case the System administrator has the option to assign the roles and make available the view for end users. Only roles related to the selected process/es will be related. If the view is a folder type, only the roles with process assigned will appear.

View Type: All

Attribute: Is Hidden

Description: The System Administrator may define the view as hidden and the end user may enable it by the Work Portal option in the Options window, Show hidden views.

View Type: All

Attribute: Is Read Only

Description: The system Administrator may define the view as read only. The end user will not be able to modify it in his own Portal.

View Type: All

Attribute: Enable Children

Description:If this check box is selected when the view is saved, all its children views will be updated to the same exact roles assigned to

it.

View Type:Instance

Attribute: Label

Description:Labels may be defined for the view for the different available languages

View Type: All

Attribute: Presentation Preview

Description:Presentation used for the view. Columns and its disposition. Any default presentation or user defined may be used.

View Type: Instances Activity and Instances.

By combining the attributes *Is Hidden* and *Is Read Only*, the end user will be able to do the following:

Is Hidden	Is Read Only	End User Actions
True	True	May be visible by selecting the "Show hidden views" in the Work Portal Options window but cannot be modified.
True	False	May be visible by selecting the "Show hidden views" in the Work Portal Options window, and can be modified.
False	True	Is always visible but cannot be modified
False	False	Is always visible and can be modified.

Processes

☒ CustomerManagement ☒ OrderFulfillment

Roles

☐ Sales Representative

View Properties

☐ Is hidden ☐ Is read only

Labels

Language

☒ English

Presentation

Preview

Description	Activity	Priority	State	Received	Deadline	Participant

Filter Options

Get instances assigned to:

☐ Case sensitive matching

Conditions

☒ Match all of the following

Add condition:

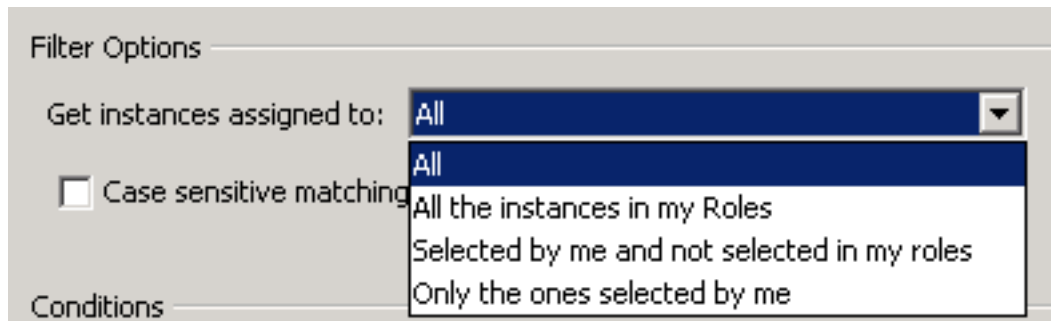
Filter Options

This section gives you the possibility of filtering the instances included in the view according to whom they are assigned.

- Possible values for the *Get Instances Assigned To* filter are:
 - **All**: All existing instances will be get, not matter to whom they

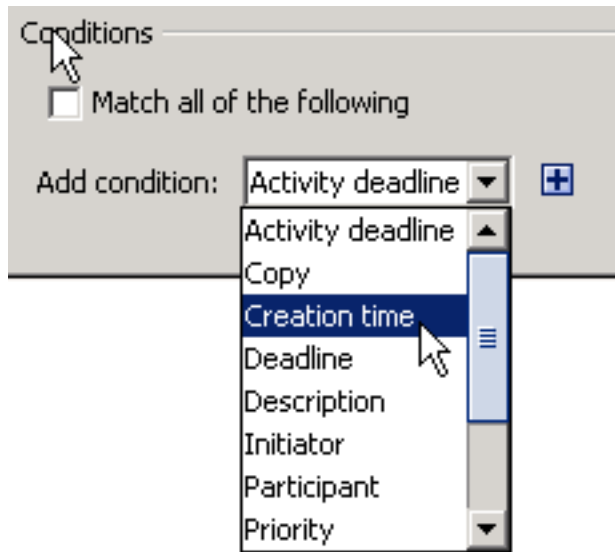
are assigned, or if they are not assigned to a participant.

- **All the instances in my roles:** Instances assigned to participants which belong to any of the roles assigned to the participant logged in the Portal.
 - **Selected by me or not selected in my roles:** instances assigned to the logged participant or, instances in activities assigned to the logged participant's role that are not selected by any other participant.
 - **Only the ones selected by me:** Instances assigned to the logged participant
- The *Case Sensitive Matching* option applies to those conditions containing text, these are *Participant* and *Description*.



Conditions

The conditions defined here will determine which instances are to be included in the view.



If the check box *Match all of the following* is selected an instance will be included in the view only if all the conditions defined are fulfilled. Otherwise the instance will be included if any of the defined conditions is true.

Conditions may refer to:

- *Activity deadline*
- *Copy*
- *Deadline*
- *Description* (case sensitive or not depending on the selected option)
- *Participant* (case sensitive or not depending on the selected option)
- *Process deadline*
- *Priority*, possible values: Highest, High, Normal, Low, Lowest.
- *Received*
- *State*, possible values: Activity completed, Exception, Grabbed, Running, Suspended.

- *Activity* This condition is available only if just one process was selected, possible values will be all activities included in the process.
- *External Variables*, The value of the processes' external variables can be used as condition. The format of the condition will be displayed depending on the type of the selected external variable. For example if the String variable *customerName* is externalized, the variable's name, CustomerName, will appear in the list of possible conditions to add. If this condition is added then the value field and the comparison combo will be the ones corresponding to a String condition.

Custom Presentations

In addition to default presentations, *default* and *default_activity* presentations which are always part of the **Project Portal Administrator**, users can define new presentations according to their functional requirements. These presentations will be available to display any of the existing views.

Creating a new presentation

1. Right-click on the **Presentations** category, select the **New presentation** option.
2. In the dialog displayed type the *Presentation* name or live the default one, and click **OK**. An new entry appears in the **Presentations** category and a panel showing the presentation data is opened in a new Tab on the left. The presentation edition panel shows the presentation's name, its preview, the columns selected to be displayed in the presentation and the available ones which can be added during edition.
3. To add a column to the presentation, from the *Available Columns* table select the column you want to add. Then click on the **right**



arrow button to add it to the selected columns. To remove a selected column, select it from the "Selected Columns" table and then click on the **left** arrow button.

4. Order the columns to be displayed, by selecting a column from the "Selected Columns" table and the clicking the up or down arrow. The default width for each selected column is shown in the **width** column and can be edited by clicking over it.

The preview panel shows how the presentation will look like after the changes are applied and the presentation is published.

Editing a presentation

Note



 If the  icon is displayed next to the presentation you want to edit, this means that the presentation is read-only and you cannot modify it.

To edit a presentation

1. Double or right click on the presentation and select **Open** from the menu to open the presentation. The selected presentation is displayed in the edition panel on the right.
2. Make the desired changes.
3. Save the project.

Deleting a presentation

Note

 If the  icon is displayed next to the presentation you want to delete, this means that the presentation is read-only and you cannot modify it.

To delete a presentation

1. Right click on the presentation and select the delete option from the short cut menu. A confirmation dialog displays.
2. Click **Yes** to delete the presentation.

Chapter 17. FuegoBPM Dashboard

FuegoBPM Dashboard - Overview

FuegoBPM Dashboard allows you to define specific control functions based on stored information. FuegoBPM Studio allows you to use Fuego Objects and their presentations to build Dashboards.

Using Fuego Objects to build a dashboard allows you to design control functions to retrieve information from the:

- Transactional database of the application,
- Statistical database of the application,
- FuegoBPM BAM,
- FuegoBPM Data Store, or
- Any other database that contains information you need to manage.

Fuego Object presentations include a set of special components. Each dashboard presentation component has different properties. These properties make it easy to modify the information displayed by the dashboard presentation component.




Special dashboard components in Fuego Object presentations are:

- *Gauge*
- *Pie*
- *Chart*

The Dashboard's graphical bar charts display certain metrics or statuses which are all displayed in distinctive colors. As the information changes, new items (bars on the graph below are automatically added on the horizontal (X) axis. Their status changes are automatically reflected in the vertical (Y) axis.

FuegoBPM Dashboard Components

Fuego Object presentations include a set of three components types designed for use when building a dashboard:

Icon	Component
	Gauge
	Pie
	Chart

Each of these components has one or more related methods that belongs to the same Fuego Object. This method retrieves the information that is shown in the component. The returned argument's quantity is the key to relate the method to the component. The interval of time to refresh the information is configured as a property of the component.


Gauge

The *Gauge* component is used when you want to show a numeric value only. An example of this would be a method related to the Gauge component called *average customer approval time*. In this example, this method returns the average time required for the instance representing a customer to reach the *End* activity in a customer management process. Other examples are a method that returns *quantity of invoices with total value higher than x* or *quantity of rejected stock orders*.

Gauge component properties

General

- **Name** - Set the component name.
- **Method Invoked** - Set the method that will be invoked to obtain the value displayed. Gauge components are only allowed to relate a method that returns a number (any type, Real, Int, Decimal, etc.).
- **Graphic Type** - Set the graphic type to show the value. Possible graphic types are:
 - **Speedometer**
 - **Thermometer**
- **Refresh Rate** - Set the interval of time to automatically refresh the measurement.
- **On Click** - Set the method that will be invoked when a *click* is performed on the graphic.
- **On Control Click** - Set the method that will be invoked when a *control click* is performed on the graphic.
- **Background color** - Set the color displayed in the background behind the graphic.
- **Value color** - Set the color in which the value is displayed.
- **Value Font** - Set the font used for the value.
- **Outline color** - Set the color of the graphic outline.
- **Content color** - Set the color of the graphic content.

- **User Refresh** - Allows user to perform a manual refresh. If this property is set to true the  icon appears in the right bottom corner of the graphic.
- **Type Selection** - If this property is set to true, when executing the Dashboard, the user will see a combo box listing all possible graphic types.
- **Unit** - Set the name of the unit in which the value is expressed. This unit appears beside the value in the graphic.

Boundaries

Values to set the boundaries in your graphic. (Parentheses show default values.)

- **Range**
 - **Min** (0)
 - **Max** (100)
- **Normal Range**
 - **Min** (50)
 - **Max** (70)
- **Warning Range**
 - **Min** (70)
 - **Max** (90)

- **Critical Range**

- **Min** (90)
- **Max** (100)

Color

Set the colors in which the different zones of the graphic are displayed.

- **Critical color**
- **Warning color**
- **Normal color**

Size

Use these properties to give the selected graphic a preferred size.

- **Width**
- **Height**

The minimum dimension is 150x150, therefore lower values will be ignored (and the minimum one will be set).

Title

Properties to set the graphic's title.

- **Title**

- **Font Type**
- **Title Background**
- **Title Foreground**


Pie

The *Pie* component is used when you want to show a list of values, such as *amount of customers by customer type*. In this example, the method returns an array of values, one per each customer type.

Pie component properties

General

- **Name** - Set the component name.
- **Method Invoked** - Set the method that will be invoked to obtain the value displayed. Pie components are only allowed to relate a method that returns a *Fuego.Chart.XYDataSource* component.
- **Graphic Type** - Set the graphic type to show the value. Possible graphic types are:
 - **Pie**
 - **Pie 3D**
- **Refresh Rate** - Set the interval of time to automatically refresh the measurement.
- **On Click** - Set the method that will be invoked when a *click* is performed on the graphic.

- **On Control Click** - Set the method that will be invoked when a *control click* is performed on the graphic.
- **Background color** - Set the color displayed in the background behind the graphic.
- **Value color** - Set the color in which the value is displayed.
- **Value Font** - Set the font used for the value.
- **Outline color** - Set the color of the graphic outline.
- **User Refresh** - Allows the user to perform a manual refresh. If this property is set to true the  icon appears in the right bottom corner of the graphic.
- **Type Selection** - If this property is set to true, when executing the dashboard the user will see a combo box listing all possible graphics types.
- **Shadow color** - Valid only when the selected graphic is 3D. The color in which the graphic shadow is displayed.
- **Plot Background** - Set the graphic area background color.
- **Plot Outline Color** - Set the graphic area outline color.
- **Content color** - Set the color of the graphic's content.

Size

Use these properties to set the size you prefer for the selected graphic.

- **Width**
- **Height**

The minimum dimension is 150x150, therefore lower values will be ignored (and the minimum one will be set).

Title

Properties to set the graphic's title.

- **Title**
- **Font Type**
- **Title Background**
- **Title Foreground**


Chart

The *Chart* component is used when you want to show a list of values for two dimensions, such as *quantity of stock orders rejected by order type*.

Chart component properties

General

- **Name** - Set the component name.
- **Method Invoked** - Set the method that will be invoked to obtain the value displayed. Chart components are only allowed to relate a method that returns a *Fuego.Chart.XYZDataSource* component, included in
- **Graphic Type** - Set the graphic type to show the value. Possible graphic types are:
 - **Multiple Pie**

- **Multiple Pie 3D**
 - **Bar Chart**
 - **Bar Chart 3D**
 - **Stacked Bar Chart**
 - **Stacked Bar Chart 3D**
 - **Area Chart**
 - **Stacked Bar Chart**
 - **Line Chart**
 - **Waterfall Chart**
-
- **Refresh Rate** - Set the interval of time to automatically refresh the measurement.
 - **On Click** - Set the method that will be invoked when a *click* is performed on the graphic.
 - **On Control Click** - Set the method that will be invoked when a *control click* is performed on the graphic.
 - **Background color** - Set the color to be displayed in the background behind the graphic.
 - **Value color** - Set the color for which the value is displayed.
 - **Value Font** - Set the font used for the value.
 - **Outline color** - Set the color of the graphic outline.
 - **User Refresh** - Allows user to perform a manual refresh. If this property is set to true the  icon appears in the right bottom corner of the graphic.

- **Type Selection** - If this property is set to true, when executing the dashboard the user will see a combo box listing all possible graphics types for displaying the information.
- **Shadow color** - Valid only when the selected graphic is 3D. The color in which the graphic shadow is displayed.
- **Plot Background** - Set the graphic area background color.
- **Plot Outline Color** - Set the graphic area outline color.
- **Autorange includes 0** - The 0 value is include in the axis ranges of the graphic. Only valid for graphics that are not *Pies*, only for: *Bar*, *Bar Chart 3D*, *Stacked Bar Chart*, *Stacked Bar Chart 3D*, *Area Chart*, *Stacked Area Chart*, *Line Chart* or *WaterFall Chart*.
- **Content color** - Set the color for the graphic content.
- **Orientation** - Valid only for bar graphics. Set the orientation of the displayed bar.
 - **Horizontal**
 - **Vertical**

Size

Use these properties to set the size you prefer for the selected graphic.

- **Width**
- **Height**

The minimum dimension is 150x150, therefore lower values will be ignored (and the minimum one will be set).

Title

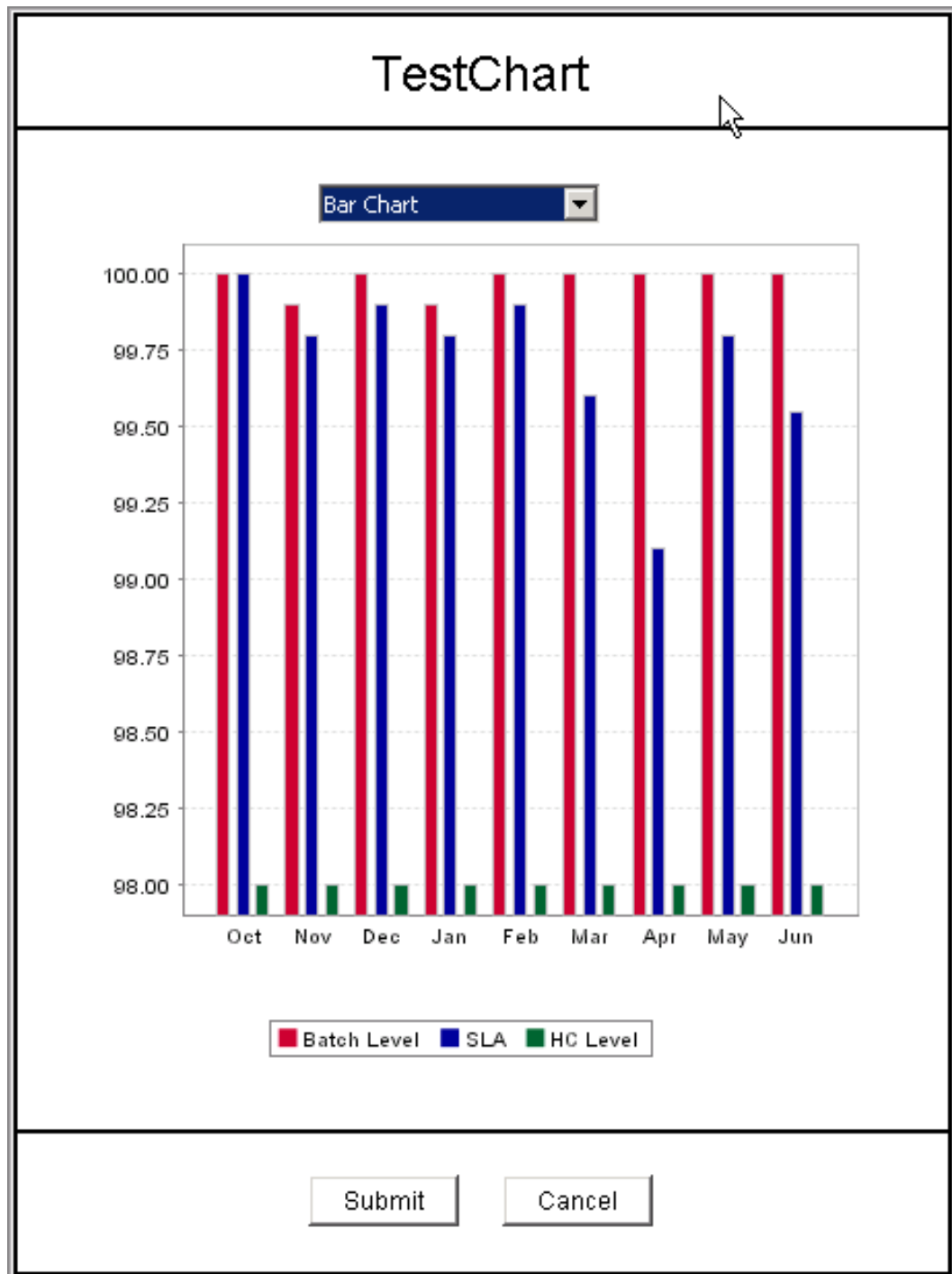
Properties to set the graphic's title.

- **Title**
- **Font Type**
- **Title Background**
- **Title Foreground**

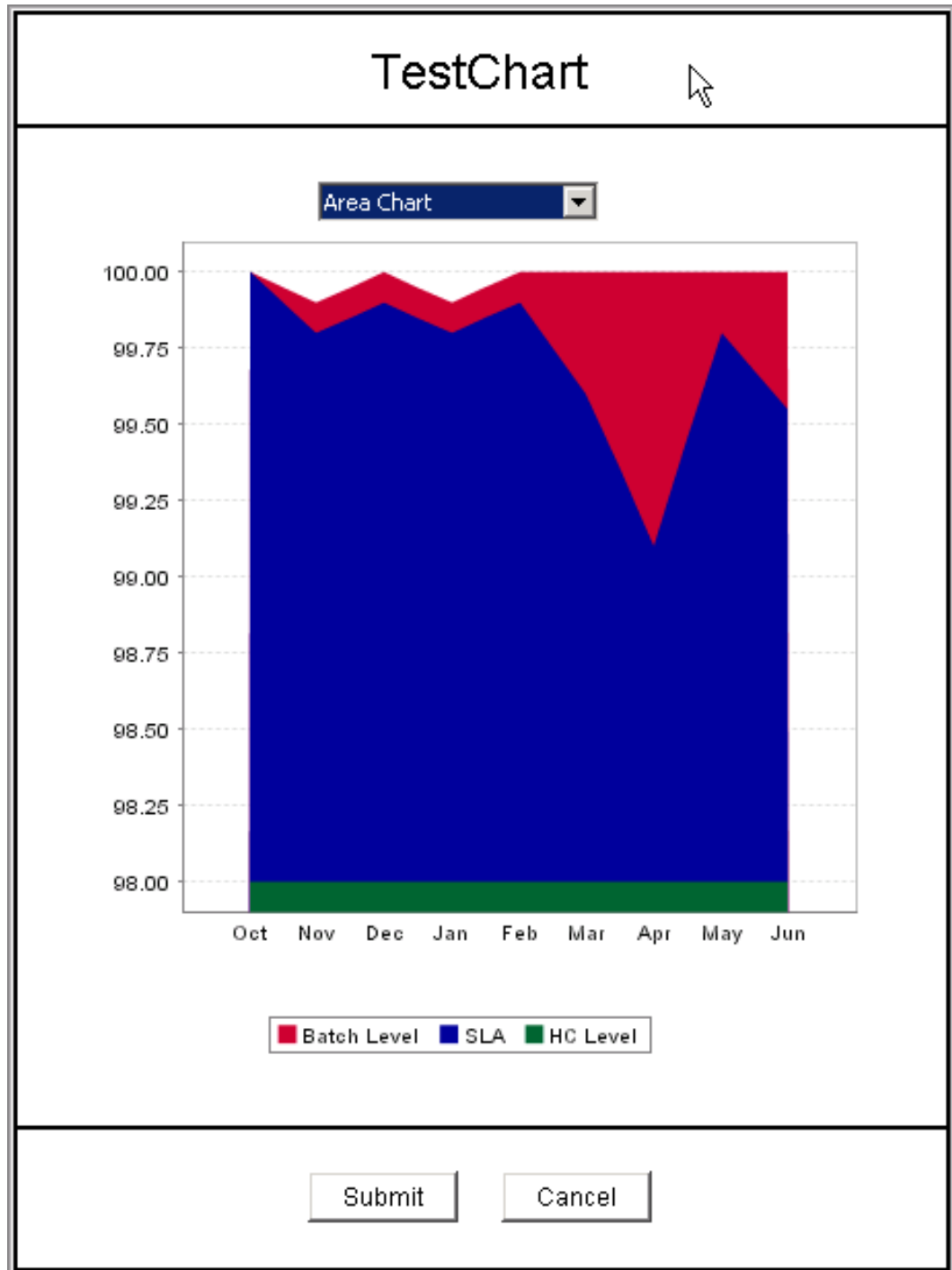
Autorange include 0 Examples:

Suppose that the graphic will only include values from 97.5 to 100.5, it will look like shown in the pictures below:

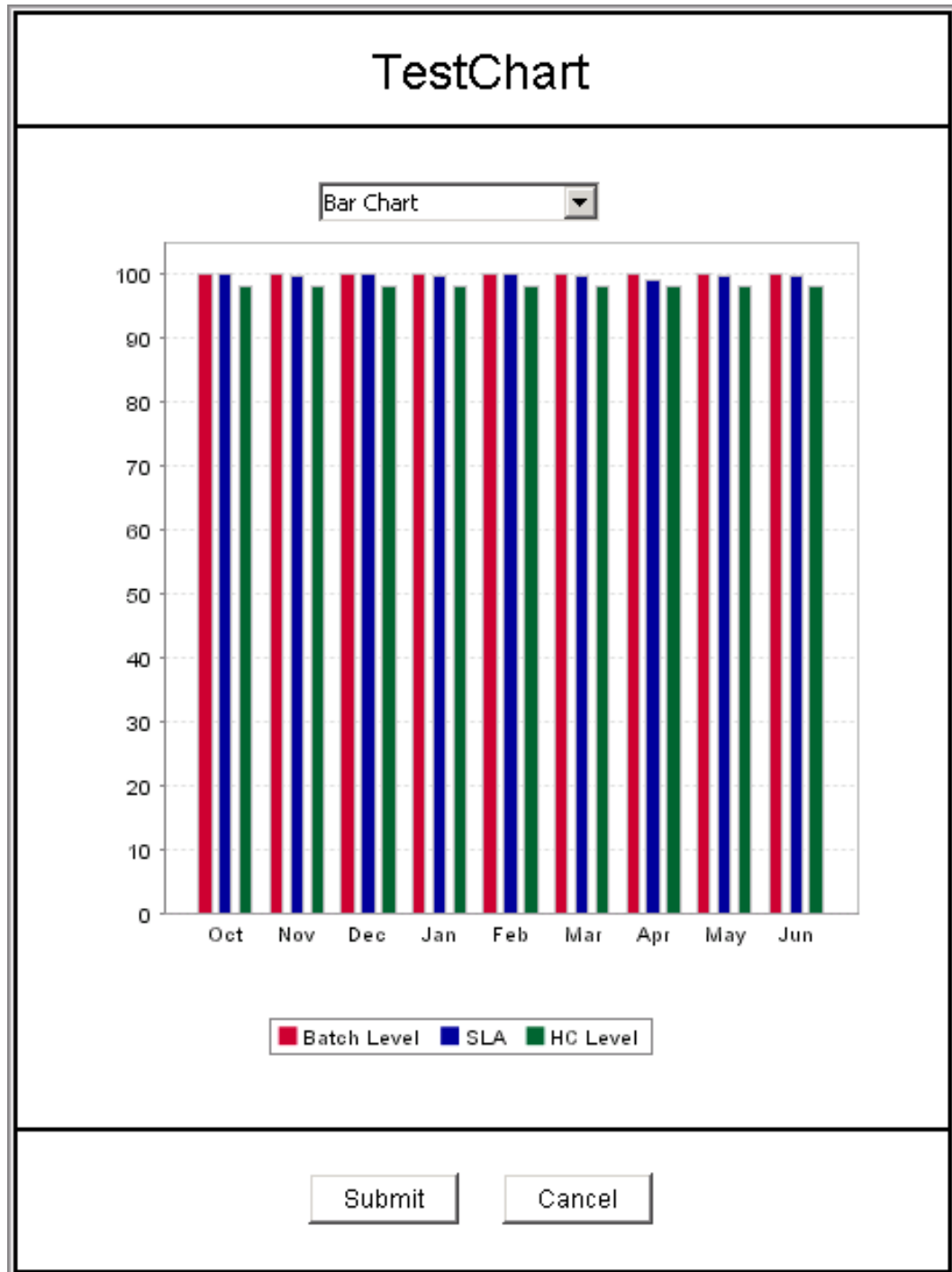
1. Autorange set to false, 0 is not included, graphic type "Bar Chart":



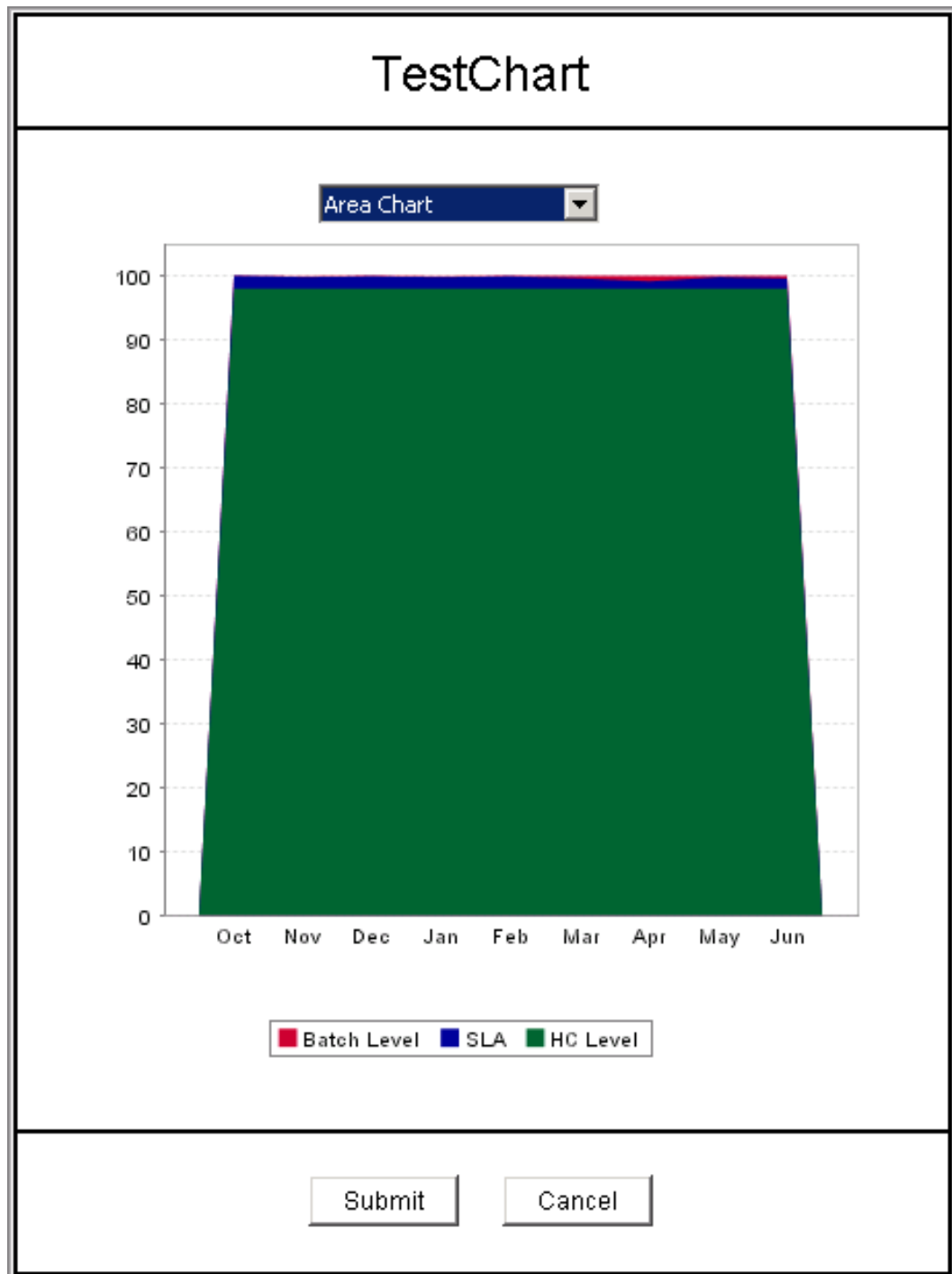
2. Autorange set to false, 0 is not included, graphic type "Area Chart":



3. Autorange set to false, 0 is included, graphic type "Bar Chart":



4. Autorange set to false, 0 is included, graphic type "Area Chart":



Of course, in this example, including the 0 value in the graphic does not make sense as it intends to show a range of values. Set this property according to your needs.

FuegoBPM Dashboard Standard Methods

FuegoBPM Dashboard are built using presentable Fuego Objects. The following table shows the standard list of methods that allow you to modify dashboard components properties at runtime.

Method Name & Description	Applies to
<i>getGaugeRange</i> : To dynamically get the gauge component range set.	Gauge.
Arguments	Returned Value
String gaugeId, Fuego.Chart.RangeKind rangeKind	Fuego.Chart.RangeKind

Method Name & Description	Applies to
<i>getGraphicSize</i> : To dynamically get the component size values set.	Gauge, Pie, Chart
Arguments	Returned Value
String graphicId	Fuego.Chart.Dimension

Method Name & Description	Applies to
<i>getGraphicType</i> : To dynamically get the graphic type set to the dashboard component.	Gauge, Pie, Chart
Arguments	Returned Value
String graphicId	Fuego.Chart.GraphicType

Method Name & Description	Applies to
<i>isGraphicTypeSelection</i> : To dynamically know if the component is set to give	Gauge, Pie, Chart

Method Name & Description	Applies to
the user the possibility of selecting the graphic type during runtime.	
Arguments	Returned Value
String graphicId	Bool

Method Name & Description	Applies to
<i>isGraphicUserRefresh</i> : To dynamically know if the component is set to give the user the possibility of refresh the graphic data during runtime.	Gauge, Pie, Chart
Arguments	Returned Value
String graphicId	Bool

Method Name & Description	Applies to
<i>setGaugeRange</i> : To dynamically set the range values to a gauge component.	Gauge
Arguments	Returned Value
String gaugeId, Fuego.Chart.RangeKind rangeKind, Fuego.Chart.Range range	void

Method Name & Description	Applies to
<i>setGraphicSize</i> : To dynamically set the size values to a dashboard component.	Gauge, Pie, Chart
Arguments	Returned Value
String graphicId, Fuego.Chart.Dimension size	void

Method Name & Description	Applies to
<i>setGraphicType</i> : To dynamically set the graphic type to a dashboard component during runtime.	Gauge, Pie, Chart
Arguments	Returned Value
String graphicId, Fuego.Chart.GraphicType graphicType	void

Method Name & Description	Applies to
<i>setGraphicTypeSelection</i> : To dynamically give the user the possibility of selecting the component graphic type during runtime.	Gauge, Pie, Chart
Arguments	Returned Value
String graphicId, Bool typeSelection	void

Method Name & Description	Applies to
<i>setGraphicUserRefresh</i> : To dynamically give the user the possibility of refreshing the component data during runtime.	Gauge, Pie, Chart
Arguments	Returned Value
String graphicId, Bool userRefresh	void

***show* method**

The show method opens a Fuego Object presentation. Its execution is asynchronous and can be performed from a running presentation by a Fuego Object method. When a presentation is invoked by the show method from another presentation and it is submitted, the

caller presentation is shown again. Presentation shown with the *show* method can include editable components. In dashboards, the method can be used from another method as a *on-click* or *on-control-click* to implement a drill-down showing the new information in a different presentation.

Find an example in the Building some FuegoBPM Dashboard examples section of this documentation, Example 3.

***openURL* method**

This method opens a given URL in a browser window.

In dashboards, this method allows to show a FuegoBPM Work Portal view related to the dashboard. For example, suppose your dashboard graphic shows instances in exception. You could implement the opening the view where the instance is, to process it.

Find examples of usage, in the related section Building some FuegoBPM Dashboard examples.

Methods Invoked from Dashboard Components

Method Invoked

The method retrieves the data set that is shown in the dashboard graphic. The return value of this method changes depending on the component.

- **Gauge:** The method must return a number.
- **Pie:** The method must return a *Fuego.Chart.XYDataSource*
- **Chart:** The method must return a *Fuego.Chart.XYZDataSource*

On Click Method

This method is invoked when the user clicks on the dashboard graphic. Depending on your business needs, this method can do anything you need it to.

One of its common uses is to implement a drill-down. To do this, the *Method invoked* method must be invoked in the *On Click* method code to reload the data set shown in the dashboard graphic. It is done by using the *refreshGraphicImage* standard method.

Any method can be invoked from the *On Click*. If the method has a returned value, it will not be taken into account.

These methods must be defined as *Server side method: false*.

On Control Click Method

This method is invoked when the user *control-clicks* on the dashboard graphic. Depending on your business needs, this method can do anything you need it to.

One of its common uses is to implement the return of a drill-down implemented in the *On Click Method*. To do this, the *Method invoked* method should be invoked in the *On Control Click* method code to reload the data set shown in the dashboard graphic. It is done by using the *refreshGraphicImage* standard method.

Any method can be invoked from the *On Control-Click*. If the method has a returned value, it will not be taken into account .

These methods must be defined as *Server side method: false*.

Methods *On Click/On Control Click* input arguments

These methods can received different number of input arguments depending on the dashboard component to which it is related.

Component	Qty of Arguments
<i>Gauge</i>	0
<i>Pie</i>	0 or 1
<i>Chart</i>	0 or 2

See FuegoBPM Dashboard Standard Methods to find information about the *refreshGraphicImage*.

The section Building some Fuego Dashboard Examples shows the usage by implementing a dashboard for the three types of methods, *Method Invoked*, *On Click Method* and *On Control Click* method.

Changing Business Rules from a Dashboard

Rules based on Business Parameters can be programmatically changed by using the *BusinessParameter* component in the *Fuego.Lib* package. See the component documentation within the Studio

From a dashboard, for example, it would possible to change the Business Parameter through an input component and a refresh of the presentation. The dashboard widgets would change according to the business parameter.

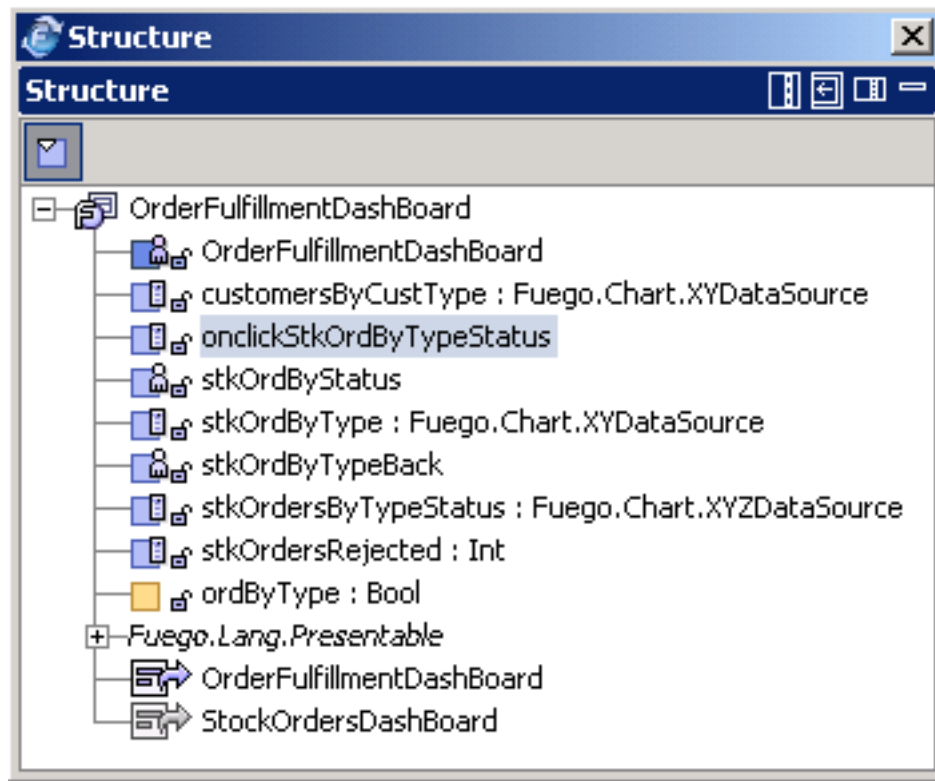
Refer to Business Parameters to learn more about them.

Building some simple Fuego Dashboard Examples

This topic explains how a Fuego Object can be designed as a dashboard. Its presentation uses the three special components needed to build a dashboard: *gauge*, *pie* and *chart*.

The Fuego Object use in these examples is named *OrderFulfillmentDashBoard* as it is the Dashboard design for that project

OrderFulfillment.fpr.



This presentable Fuego Object has two dashboards defined. The first one, set as default presentation, is the *OrderFulfillmentDashBoard* which shows the three possible components to build a dashboard.

The second dashboard has all the possible methods properties defined.

The following section explain the attributes, methods and presentation of the Fuego Object for each defined dashboard.

Example 1: *OrderFulfillmentDashBoard*

This example presents:

- Gauge Component,
- Pie Component,

- Chart Component,
- Method Invoked,
- On Click method.
- Usage of dashboard standard methods *refreshGraphicImage*.

Attributes

No attributes are used in this dashboard.

Methods

customerByCustType: This method returns an object containing the quantity of customers by customer type from the application database using a SQL embedded sentence. This object has to be defined as *Fuego.Chart.XYDataSourceImpl*.

```
tableModel = XYDataSourceImpl("")
for each element in
    SELECT custtype, count(*)
    FROM CUSTOMER
    GROUP BY custtype
do

    addValue tableModel
    using value = element.columnexpr1,
        columnHeader = element.custtype
end

return tableModel
```

onclickStkOrdByTypeStatus:

```
refreshGraphicImage this
    using graphicId = "stkOrdByTypeStatus"
```

stkOrdersByTypeStatus: This method returns an object containing the the quantity of Stock Orders grouped by type and status. This object has to be defined as *Fuego.Chart.XYZDataSourceImpl*.

```
tableModel = XYZDataSourceImpl()
for each element in
    SELECT stktype,status, count(*)
    FROM STKORDER
    GROUP BY stktype, status
do
    addValue tableModel
    using value = element.columnexpr2,
        rowHeader = element.stktype,
        columnHeader = element.status
end
return tableModel
```

stkOrdersRejected: This method returns the quantity of rejected Stock Orders from the application database. Returns an *Int*.

```
stkOrdRej as Int
for each element in
    SELECT count(*)
    FROM STKORDER
    WHERE status = "DISC"
do
    stkOrdRej = element.columnexpr0
end
return stkOrdRej
```

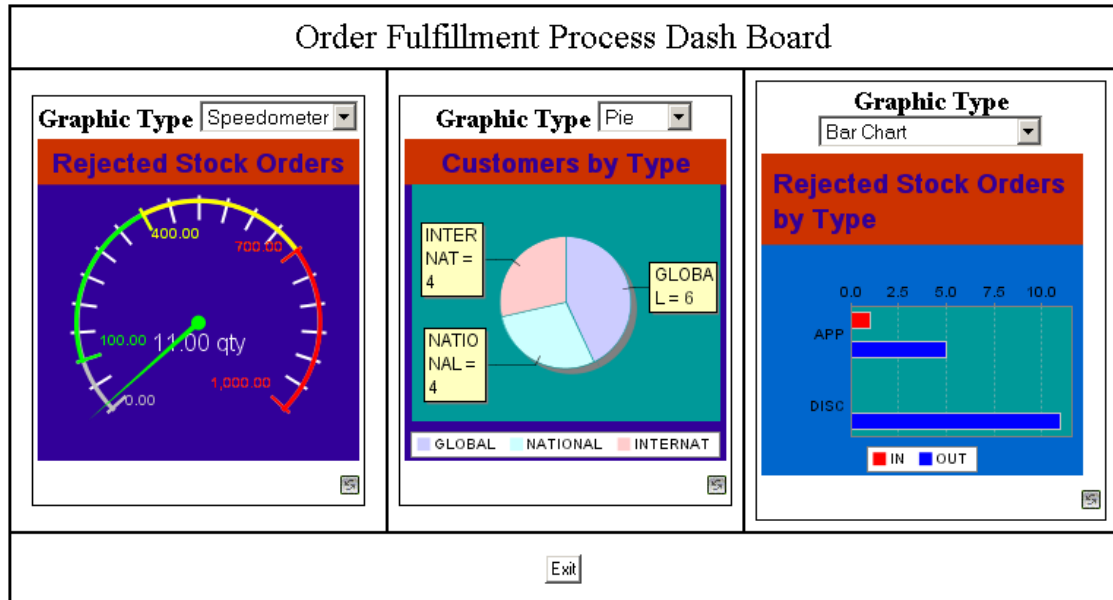
Presentation components

Components

- ***rejStkOrd*** : Gauge component
 - The *gauge* component implemented in this example shows the number of *rejected stock orders* in a Stock Administration process.
 - Colors, sizes, and fonts have been customized.
 - *User Refresh* and *Type Selected* properties are set to true.
 - *Method Invoked*: The method invoked to fill the data set is *stkOrdersRejected*. This method returns an *Int*. The method code is an sql embedded to retrieve the information from the application database. That is why it is defined as a *runs on server* method. (See methods details in section above.)
- ***custByType***: Pie component
 - The *pie* component implemented in this example shows the number of *customers by type* in a Customer Management process.
 - Colors, sizes, fonts have been customized.
 - *User Refresh* and *Type Selected* properties are set to true.
 - *Method Invoked*: The method invoked to fill the data set is *customersByCustType*. This method returns a *Fuego.Chart.XYDataSource* object. The method code is an sql embedded to retrieve the information from the application database. That is why it is defined as a *runs on server* method. (See methods details in section above.)
- ***stkOrdByTypeStatus***: Chart component

- The *chart* component implemented in this example shows the number of *rejected stock orders by type* in a Stock Administration process.
- Colors, sizes, fonts have been customized. *User Refresh* and *Type Selected* properties are set to true.
- *Method Invoked*: The method invoked to fill the data set is *stkOrdersRejectedByType*. This method returns a *Fuego.Chart.XYZDataSource* object. The method code is an sql embedded to retrieve the information from the application database. That is why it is defined as a *runs on server* method.(See methods details in section above.)
- *On Click*: The method invoked on the on-click is *onclickStkOrdByTypeStatus*. This methods implements a load refresh of the graphic data by using the standard method *refreshGraphicImage*. (See methods details in section above.)

Executing the *OrderFulfillmentDashBoard*



Example 2: *StockOrderDashboard*

This example presents:

- Pie Component,
- Method Invoked,
- On Click method.
- On Control Click method.
- Usage of dashboard standard methods *refreshGraphicImage*, *isGraphicUserRefresh*, *setGraphicUserRefresh*.

Attributes

- *ordByType*: This attribute is used by the *Method Invoked* and the action methods *On click* and *On Control Click*. Depending on its value

the method that loads the graphic with data executes a different SQL statement. The *On click* and *On Control Click* methods changes it value to change the logic.

Methods

stkOrdersByType: This method returns an object containing the the quantity of Stock Orders grouped by type or by status. This object has to be defined as *Fuego.Chart.XYDataSourceImpl*. The method based on the attribute *ordByType* it executes a different SQL statement. The *ordByType* attribute changes its value when clicking or control clicking on the graphic when the related methods to those action are executed.

```
tableModel = XYDataSourceImpl("")

if ordByType then
  for each element in
    SELECT stktype, count(*)
    FROM STKORDER
    GROUP BY stktype
  do
    addValue tableModel
    using value = element.columnexpr1,
        columnHeader = element.stktype
  end
else
  for each element in
    SELECT status, count(*)
    FROM STKORDER
    GROUP BY status
  do
    addValue tableModel
    using value = element.columnexpr1,
        columnHeader = element.status
  end
end
return tableModel
```

stkOrdersByStatus: This method is invoked in the *On click* action. It changes the value to the *ordByType* attribute and executes the *refreshGraphicImage* standard method which invokes the method that loads the data to the graphic, *stkOrdBytype*.

The method dynamically sets some component attributes, like *title* and the *user refresh*.

```
result as Bool
ordByType = false
// Dynamic Sets
// Change the title
setText this
    using componentId = "piel",
        text = "Stock Orders By Status"

// Changes the User Refresh property
result = isGraphicUserRefresh(graphicId : "piel")
if result then
    setGraphicUserRefresh this
        using graphicId = "piel",
            userRefresh = false
end

//Reload data
refreshGraphicImage this
    using graphicId = "piel"
```

stkOrdersByTypeBack: This method is invoked in the *On Control click* action to reload the graphic with the information or Orders by Type, the original shown data. It changes the value to the *ordByType* attribute and executes the *refreshGraphicImage* standard method which invokes the method that loads the data to the graphic, *stkOrdBytype*.

The method dynamically sets some component attributes, like *title* and the *user refresh*.

```
result as Bool
ordByType = true
// Dynamic Sets
```

```
// Changes the title
setText this
    using componentId = "piel",
    text = "Stock Orders By Type"

// Changes the User Refresh property
result = isGraphicUserRefresh(graphicId : "piel")
if !result then
    setGraphicUserRefresh this
        using graphicId = "piel",
        userRefresh = true
end

//Reload data
refreshGraphicImage this
    using graphicId = "piel"
```

Presentation

- *stkOrdByTypeStatus*: Pie component
 - This *pie* component implemented in this example shows initially, the number of *stock orders by type* in a Stock Administration process.
 - Colors, sizes, fonts have been customized. *User Refresh* and *Type Selected* properties are set to true.
 - *Method Invoked*: The method invoked to fill the data set is *stkOrdByType*. This method returns a *Fuego.Chart.XYDataSource* object. The method code is an sql embedded to retrieve the information from the application database. That is why it is defined as a *runs on server* method. (See methods details in section above.)
 - *On Click*: The method invoked on the on-click is *stkOrdByStatus*. This methods implements a load refresh of the graphic data by using the standard method *refreshGraphicImage*. The method sets

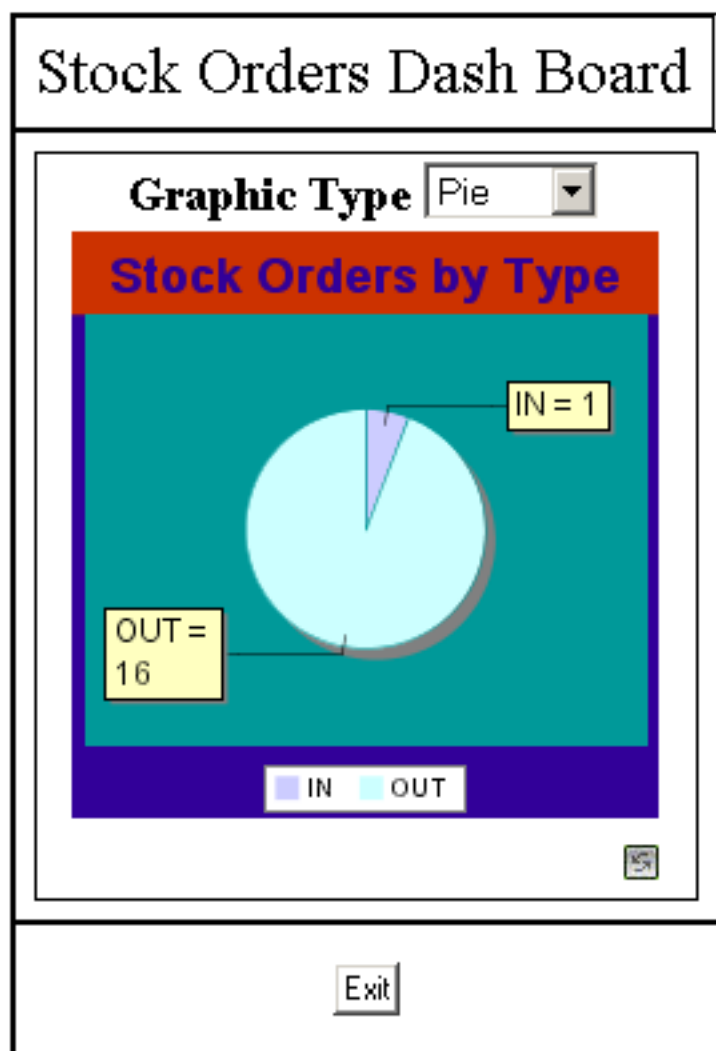
dinamically some properties of the component. (See methods details in section above.)

- *On Control Click*: The method invoked on the on-click is *stkOrdByTypeBack*. This methods implements a load refresh of the graphic data by using the standard method *refreshGraphicImage*. The method sets dynamically some properties of the component. (See methods details in section above.)

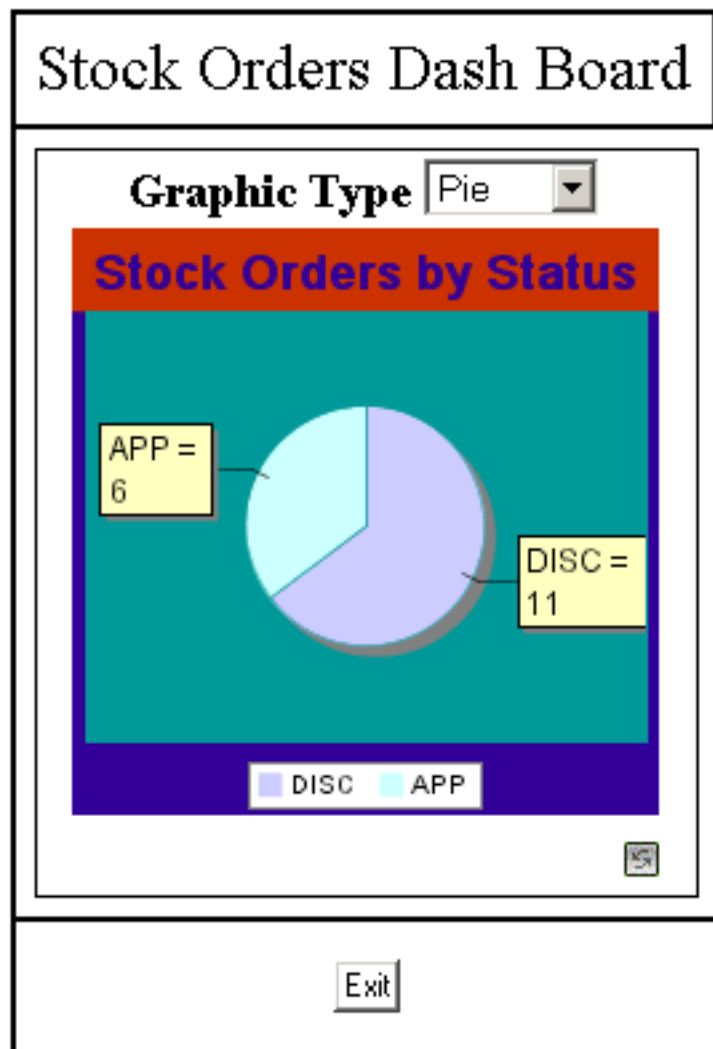
By implementing these two last methods, the user can visualize the Orders by type or by status.

Executing the *StkOrdersDashBoard*

- Initial loaded and *On Control Click* executions:



- *On Click* execution:



Example 3: Dashboard using the *show* method

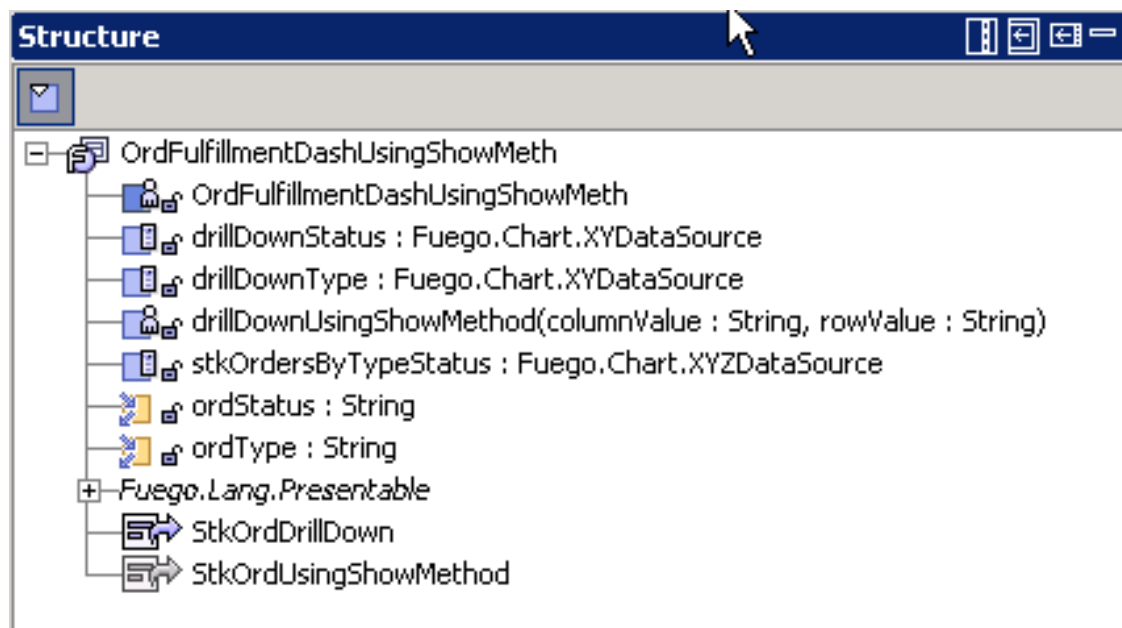
This example presents:

- Pie Component,
- Method Invoked,

- On Click method, receiving arguments,
- *show* method usage.

The Fuego Object used as example is *OrdFulfillmentDashUsingShowMeth* present in the *DashBoardExampleFR* project distributed with the installation. You can find it under the *sample* directory of your FuegoBPM Studio installation directory.

The Fuego Object looks like:



Attributes

- *ordType*: attribute used to store the type value selected in the main presentation when the drill down is performed. This attribute is used in the *drillDownType* method.
- *ordStatus*: attribute used to store the type value selected in the main presentation when the drill down is performed. This attribute is used in the *drillDownType* method.

Methods

stkOrdersByTypeStatus: This method returns an object containing the the quantity of Stock Orders grouped by type and status. This object has to be defined as *Fuego.Chart.XYZDataSourceImpl*.

```
tableModel = XYZDataSourceImpl()  
for each element in  
    SELECT stktype,status, count(*)  
    FROM STKORDER  
    GROUP BY stktype, status  
do  
    addValue tableModel  
    using value = element.columnexpr2,  
           rowHeader = element.stktype,  
           columnHeader = element.status  
end  
  
return tableModel
```

drillDownUsingShowMethod: This method is invoked in the on-click property of the component *stkOrdByTypeStatus*. It is implemented receiving two arguments which are the values of the *column* and *row* where the click was done. The order in which the values are received is: column, row. The received values are set to the Fuego Object attributes *ordType* and *ordStatus*. This method invokes the *show* method which displays a new presentation. The second presentation opened uses the Fuego Object attributes set with the column and row in their component invoked methods.

```
ordType = rowValue  
ordStatus = columnValue  
show this  
    using instance = this,  
           presentationName = "StkOrdDrillDown"
```

drillDownStatus: This method returns an object containing the quantity of Stock Orders per type in the status selected in the drill down.

```
tableModel = XYDataSourceImpl("")
for each element in
    SELECT stktype, status, count(*)
    FROM STKORDER
    WHERE status=ordStatus
    GROUP BY stytype, status
do
    addValue tableModel
    using value = element.columnexpr2,
        columnHeader = element.stktype
end
return tableModel
```

drillDownType: This method returns an object containing the quantity of Stock Orders per status in the type selected in the drill down.

```
tableModel = XYDataSourceImpl("")
for each element in
    SELECT status, stktype, count(*)
    FROM STKORDER
    WHERE stktype = this.ordType
    GROUP BY status, stktype
do
    addValue tableModel
    using value = element.columnexpr2,
        columnHeader = element.status
end
return tableModel
```

Presentation

StkOrdUsingShowMethod

- ***stkOrdByTypeStatus***: Pie component
 - This *pie* component implemented in this example shows

initially, the number of *stock orders by type* in a Stock Administration process.

- Colors, sizes, fonts have been customized. *User Refresh* and *Type Selected* properties are set to true.
- *Method Invoked*: The method invoked to fill the data set is *stkOrdByType*. This method returns a *Fuego.Chart.XYDataSource* object. The method code is an sql embedded to retrieve the information from the application database. That is why it is defined as a *runs on server* method. (See methods details in section above.)
- *On Click*: The method invoked on the on-click is *drillDownUsingShowMethod*, which implements the *show* method to open the second presentation *StkOrdDrillDown*.

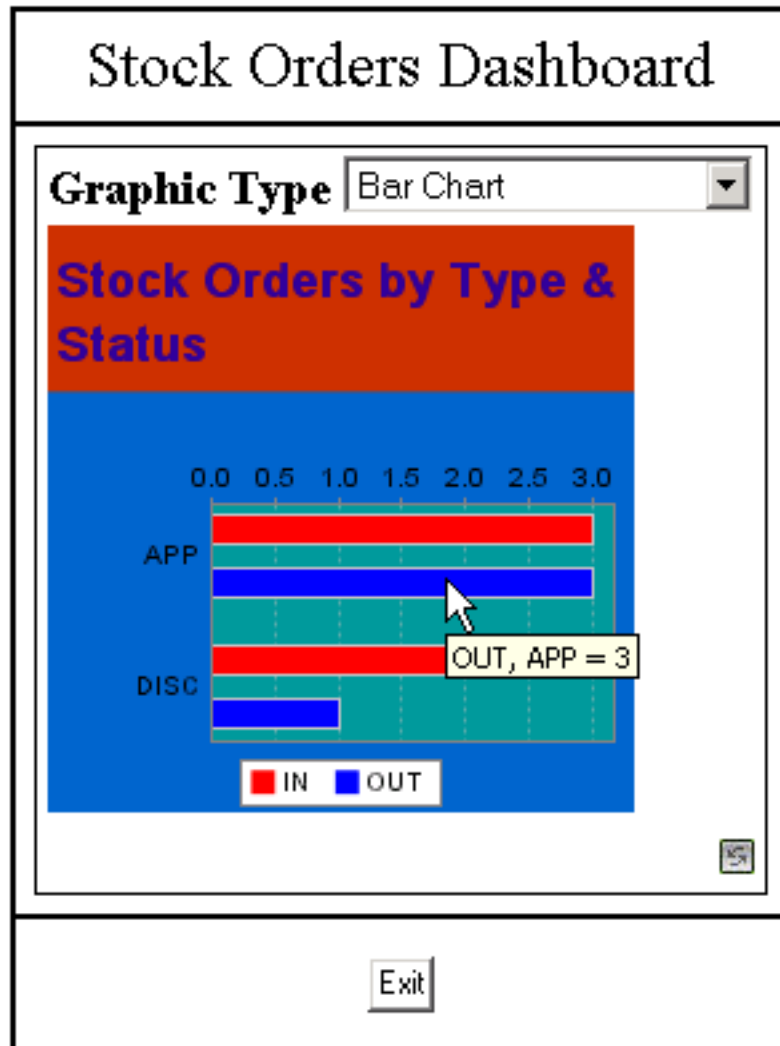
StkOrdDrillDown

- *ordType*: text component. Show the type corresponding to the bar click in the main presentation, which is stored in the *ordType* attribute of the Fuego Object.
- *ordStatus*: text component. Show the type corresponding to the bar click in the main presentation, which is stored in the *ordType* attribute of the Fuego Object.
- *ordersByType*: Pie component
 - This *pie* component implemented in this example shows initially, the number of *stock orders by type for the status selected in the main presentation bar*.
 - Colors, sizes, fonts have been customized. *User Refresh* properties are set to true.

- *Method Invoked*: The method invoked to fill the data set is *drillDownType*.
- ***ordersByStatus***: Pie component
 - This *pie* component implemented in this example shows initially, the number of *stock orders by status for the type selected in the main presentation bar*.
 - Colors, sizes, fonts have been customized. *User Refresh* properties are set to true.
 - *Method Invoked*: The method invoked to fill the data set is *drillDownStatus*.

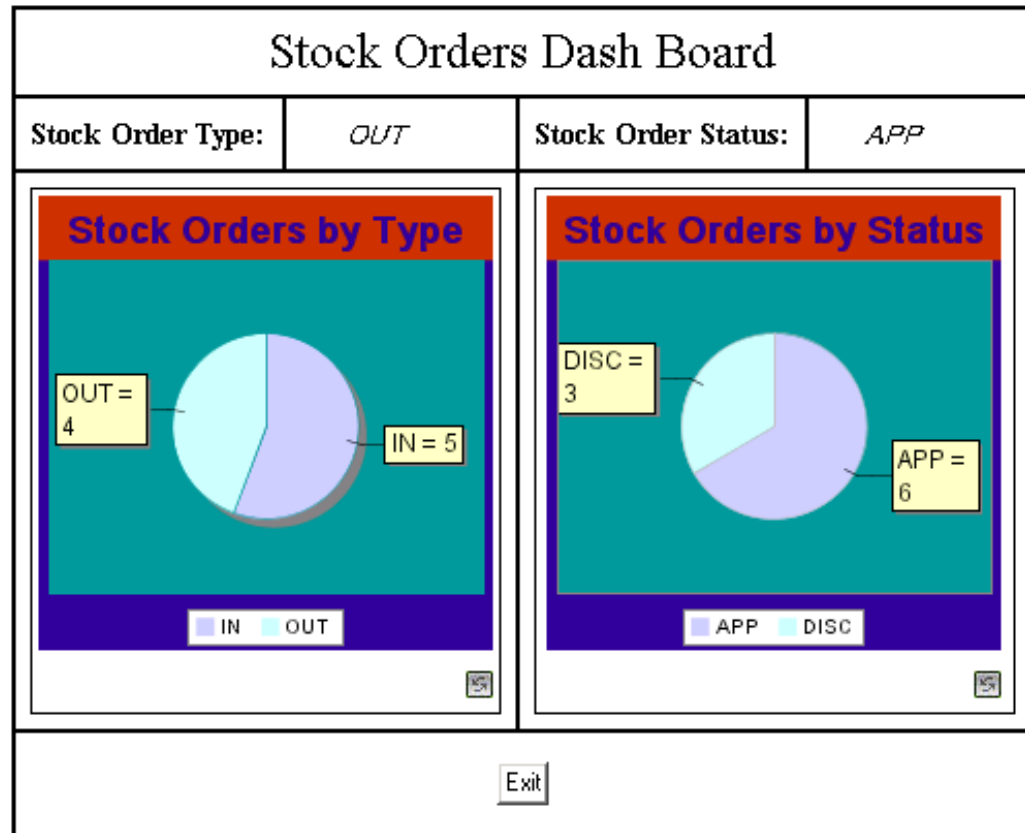
Executing the Presentations

- StkOrdUsingShowMethod



After selecting the bar corresponding to status=*APP* and type=*OUT*, the following presentation is displayed.

- StkOrdDrillDown



Building a BAM Dashboard using the Wizard

BAM reduces this flood of data and creates meaningful graphical views of real-time information. It consists of a set of graphical charts designed in the FuegoBPM Studio that take measurements of Key Performance Indicators (KPI) during the execution of a process.

The company's processes are controlled by a FuegoBPM Server that automates the integration of the company's applications, databases and people. Each step through the company's processes is automatically measured and monitored.

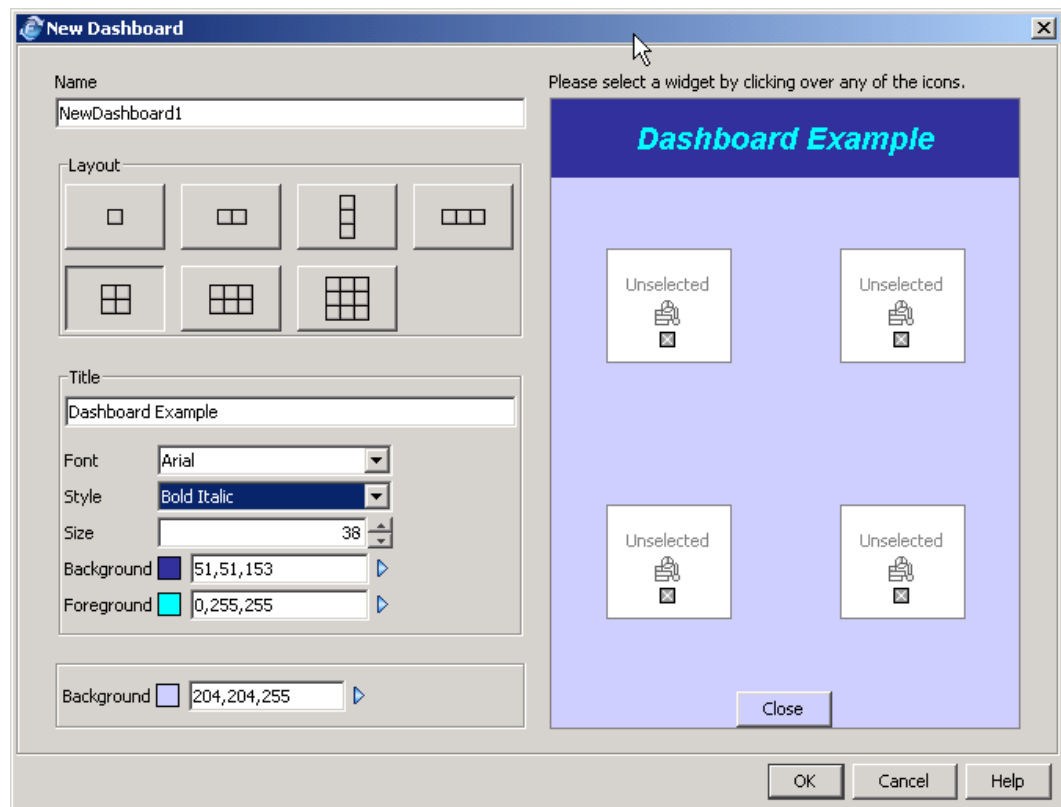
As instances flow through FuegoBPM activated processes, BAM reads real-time data from the FuegoBPM Server. The Dashboard's

graphical charts display certain metrics displayed in distinctive colors. When statuses reach a critical state the colors change and the decision maker is alerted. See BAM Configuration to learn how to set BAM in FuegoBPM Studio environment.



BAM Dashboard Creation Wizard

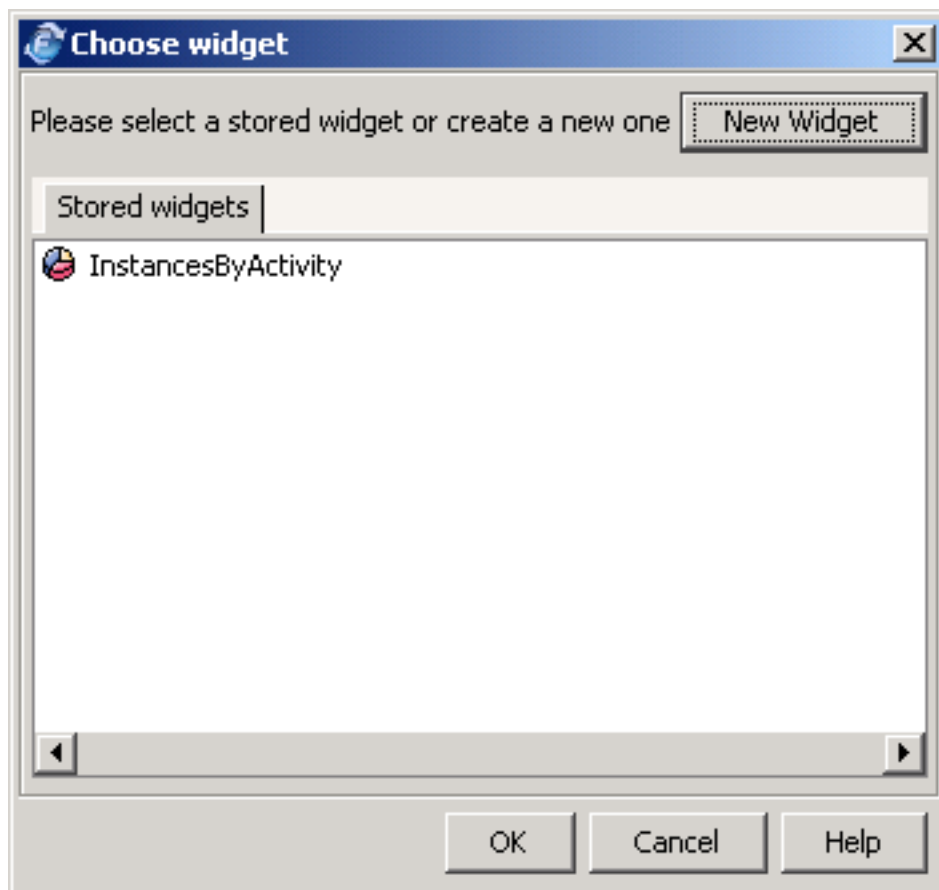
To create a dashboard

1. Right-click on the project catalog module you want to add the dashboard to. Select the **New Dashboard** option.
2. The dashboard layout panel opens.



3. From this panel you can:
 - a. give a name to the dashboard,

- b. select the layout from the available defaults by clicking the graphic for each one,
 - c. edit the dashboard title and set its foreground and background properties and
 - d. set the dashboard background color. The default *Close* button included in the dashboard generation, inherits this property.
4. Add , remove  the template widgets by clicking on the respective icons.
5. When you are adding a widget to the dashboard, the **Choose Widget** dialog opens.



Select the widget template from the preexisting ones or create a

new widget template by clicking the **New Widget** button.

6. When you have finished adding the widgets to the dashboard, click **OK** to close the wizard. The dashboard is created under the module you have chosen. The dashboard is a Fuego Object. To modify a dashboard you can modify its methods and presentations if needed. Remember that if you change a presentation once it has been created, the change is only reflected for that dashboard.

Creating a Template Widget

Defining the DataSource

The first pane in the creation widget panel, is to define the datasource. In this pane you define:

- The template widget name,
- The process used to get the information from,
- The data type
 - Activity workload,
 - Activity Performance, or
 - Process Performance.
- Activity. If the information refers to one activity and which one or, if it refers to all the process activities.
- Dimensions.
- Measure.

Dimensions states can be: totalized or filtered.

Business Measurements can be: max, min, avg, count, sum.

Data source Definition

1. Activity Workload

- a. One Activity: activity or start/stop measurement mark can be included.
 - i. Dimensions: Role, Participants, Dimension Business Variables (range or string).
 - ii. Measurements: Number of instances, Average time task, Mean time task, Measurement Business Variables .
- b. All Activities
 - i. Dimensions: Role, Participants, Dimension Business Variables (range or string), Activities.
 - ii. Measurements: Number of instances, Average time task, Mean time task, Measurement Business Variables .

2. Activity Performance

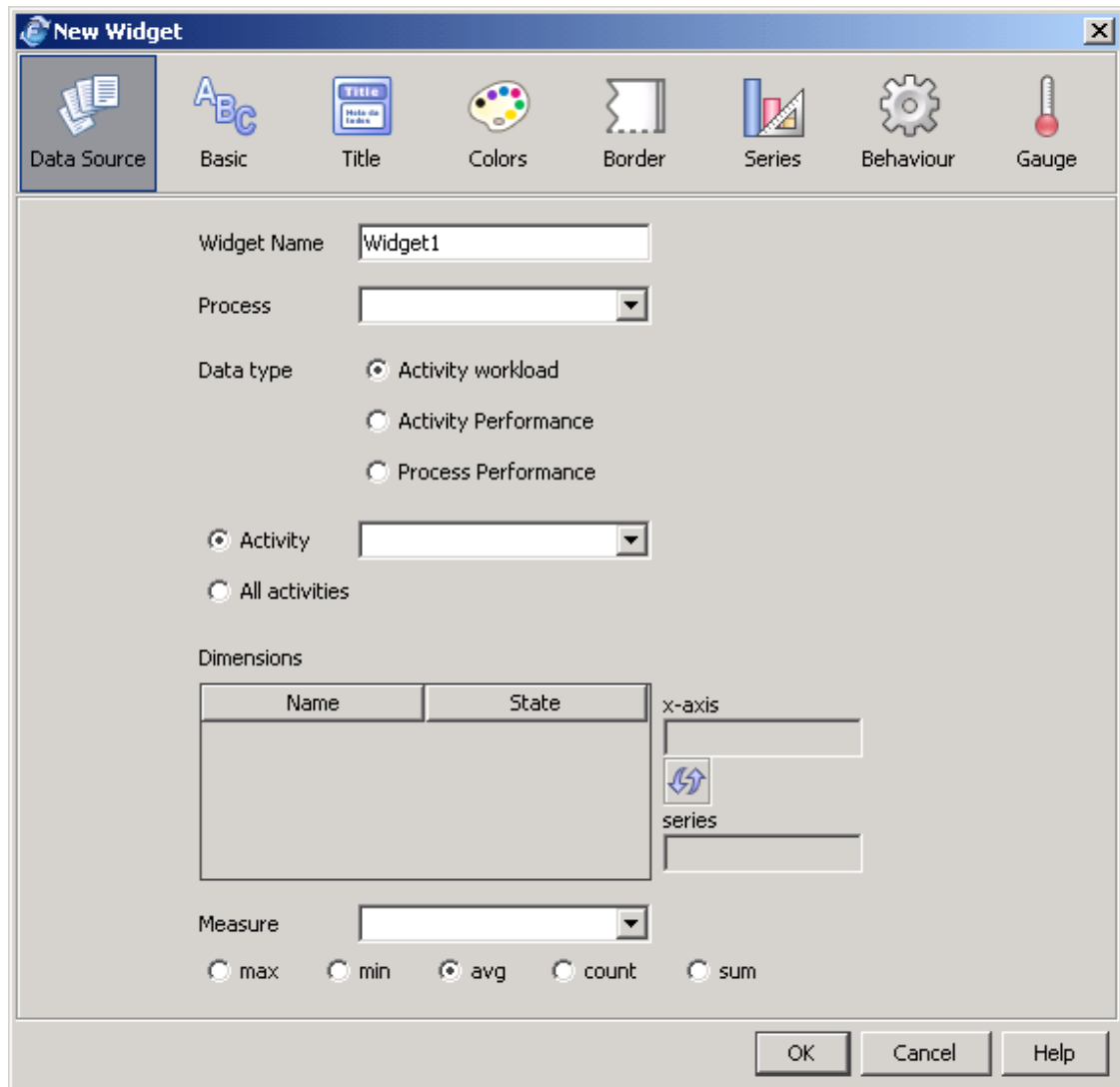
- a. One Activity: activity or measurement mark can be included.
 - i. Dimensions: Role, Participants, Dimension Business Variables (range or string).
 - ii. Measurements: Task time and Measurement Business Variables

b. All Activities

- i. Dimensions: Role, Participants, Dimension Business Variables (range or string), Activities.
- ii. Measurements: Task time and Measurement Business Variables

3. Process Performance

- a. Dimensions: Dimension Business Variables
- b. Measurements: Task time and Measurement Business Variables



The 'New Widget' dialog box features a toolbar with icons for Data Source, Basic, Title, Colors, Border, Series, Behaviour, and Gauge. The 'Data Source' tab is selected. The form contains the following fields and options:

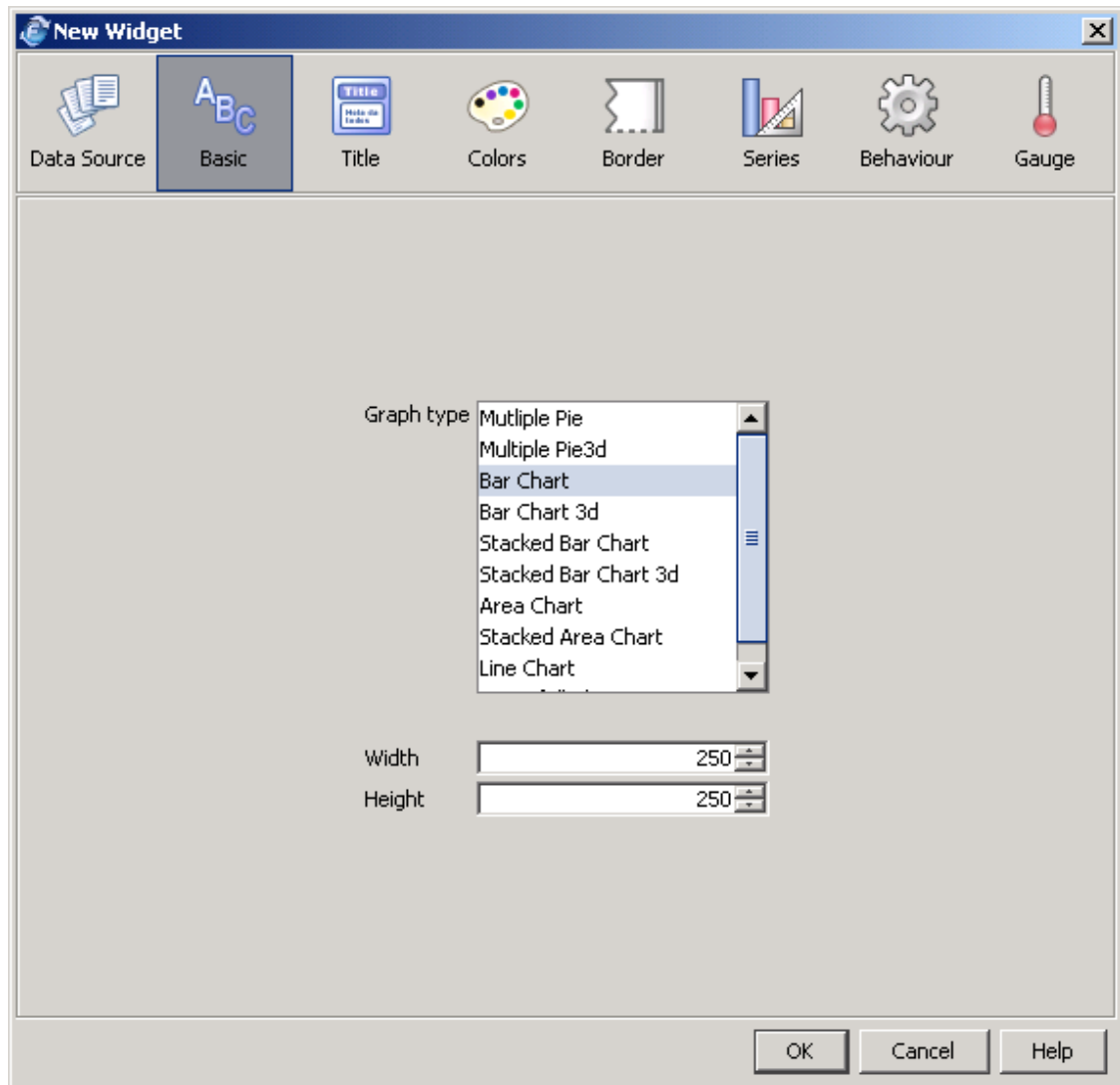
- Widget Name:** A text field containing 'Widget1'.
- Process:** A dropdown menu.
- Data type:** Radio buttons for 'Activity workload' (selected), 'Activity Performance', and 'Process Performance'.
- Activity:** Radio buttons for 'Activity' (selected) and 'All activities', with a dropdown menu next to 'Activity'.
- Dimensions:** A table with two columns: 'Name' and 'State'.

Name	State
------	-------
- x-axis:** A text field.
- series:** A text field, preceded by a swap icon.
- Measure:** A dropdown menu.
- Measure options:** Radio buttons for 'max', 'min', 'avg' (selected), 'count', and 'sum'.

Buttons for 'OK', 'Cancel', and 'Help' are located at the bottom right.

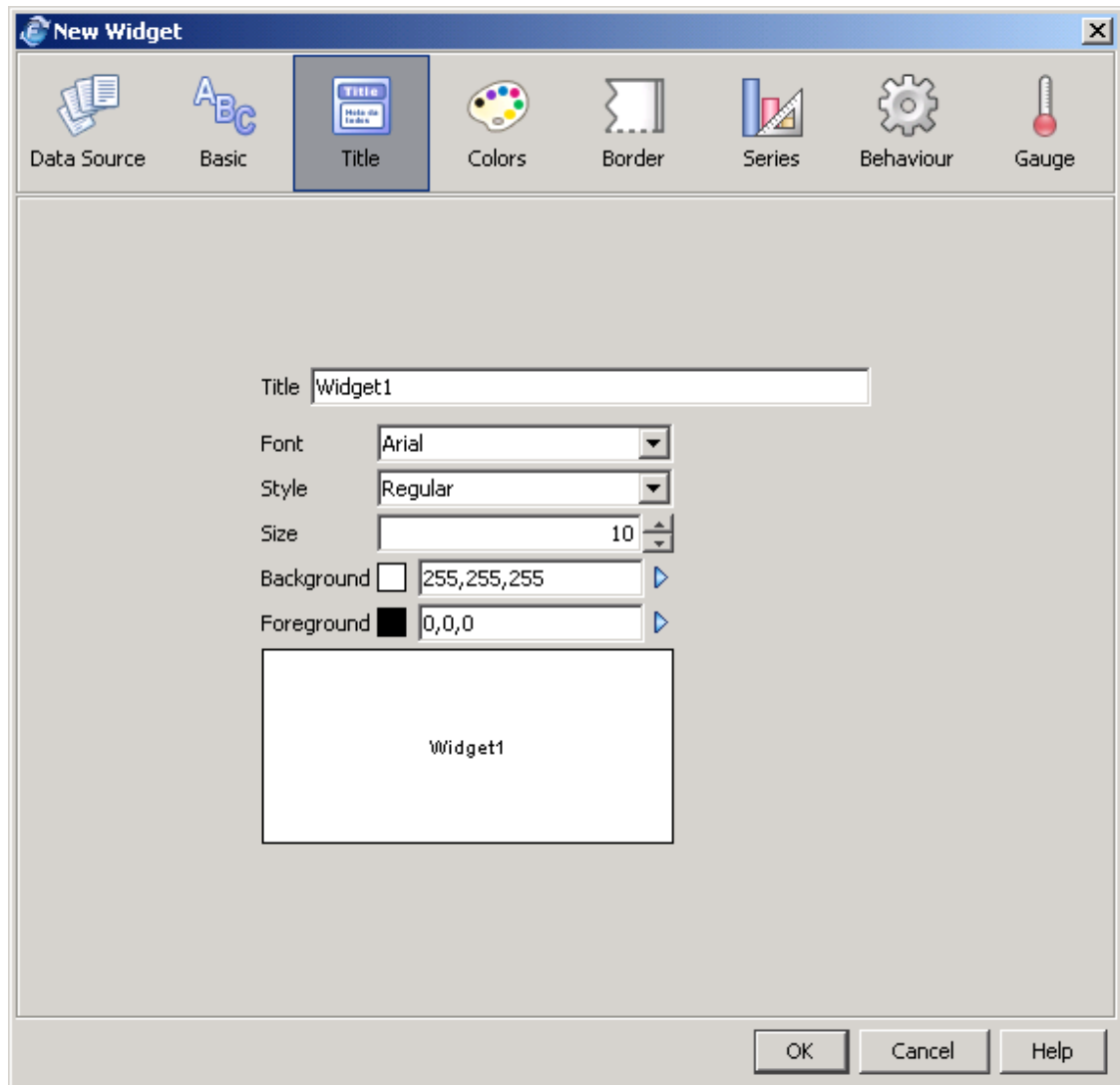
Graph

Define the graph type for the widget and its size. Graph types shown are only those that can be related with the type of the built data source.



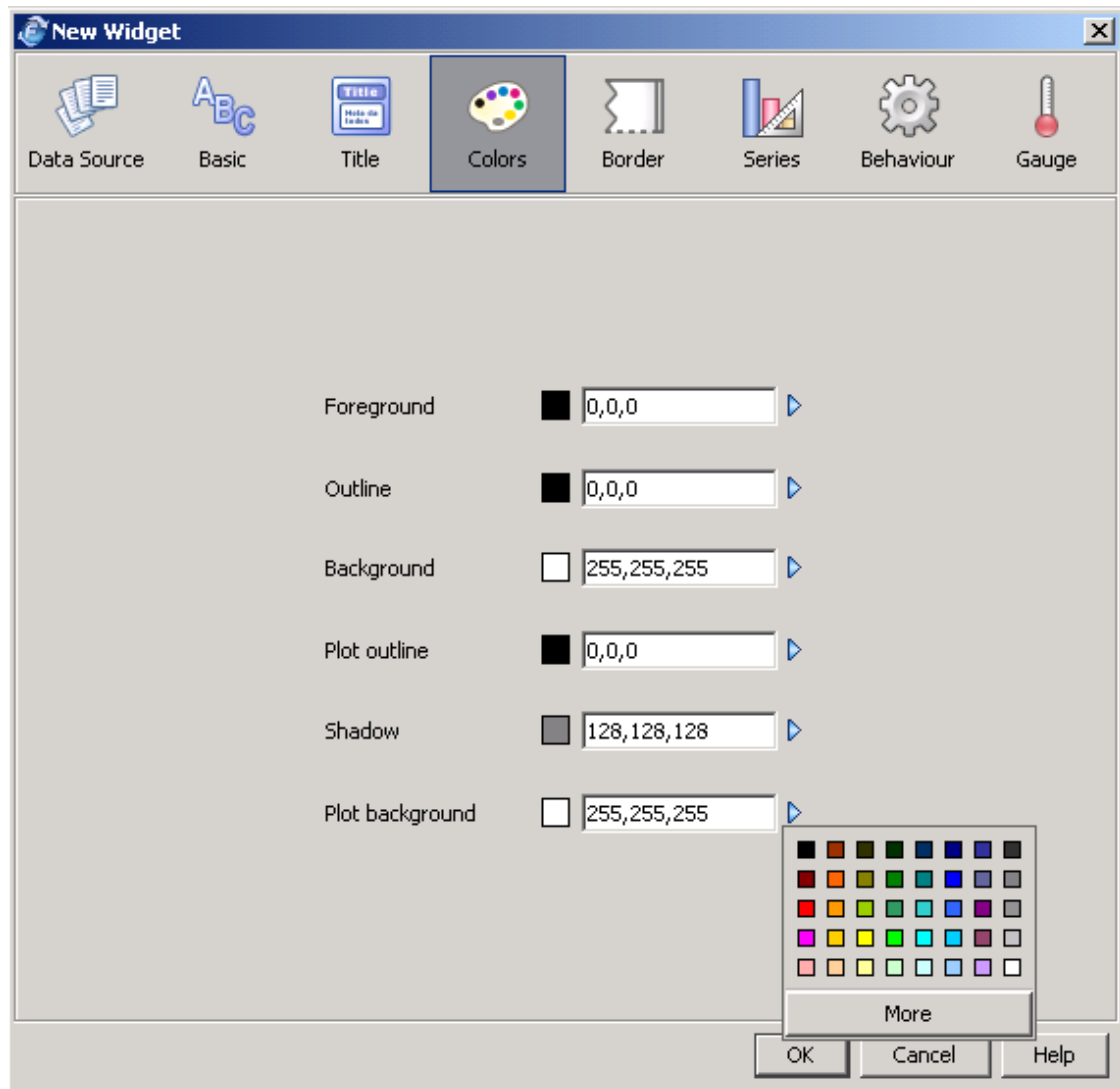
Tittle format

Define the widget tittle, its format in font, size and style. You can select a color for the foreground and background of the tittle.



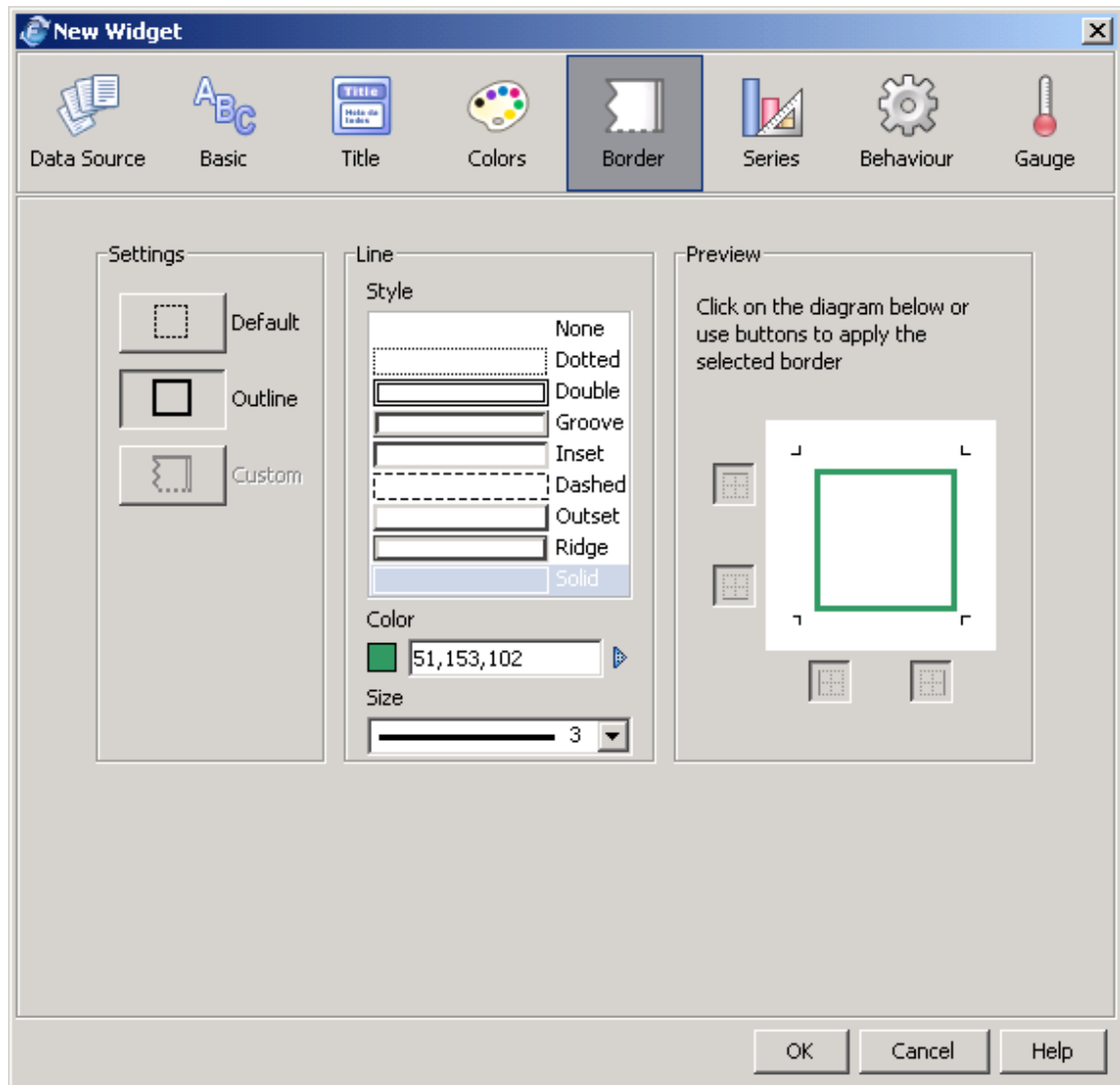
Colors

Define the colors in which the widget will be displayed.



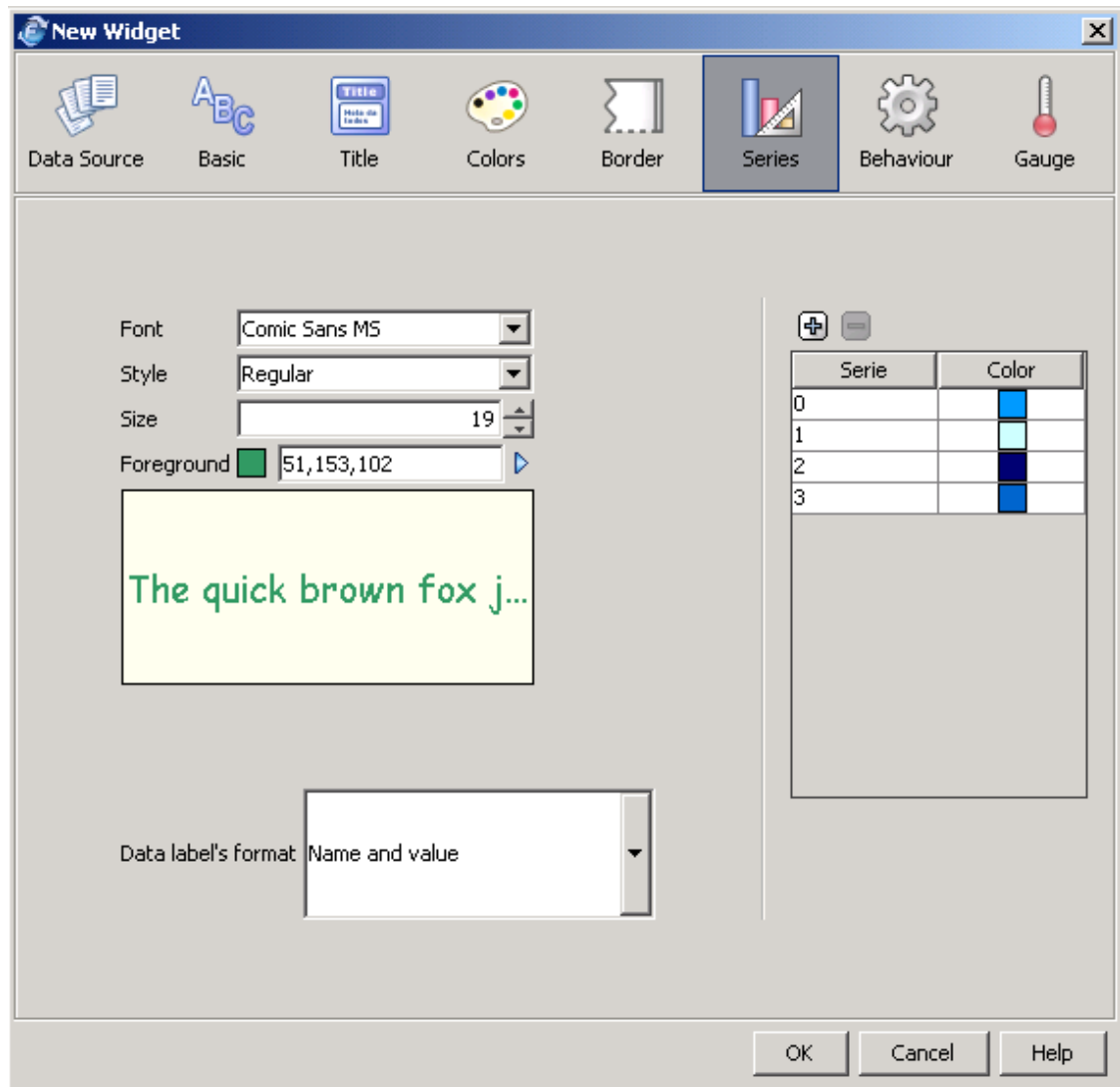
Border properties

Define the border properties. Border properties are defined using the same **Border Editor** that is used for Fuego Object Presentations. Refer to Fuego Object Presentation Components Border Editor.



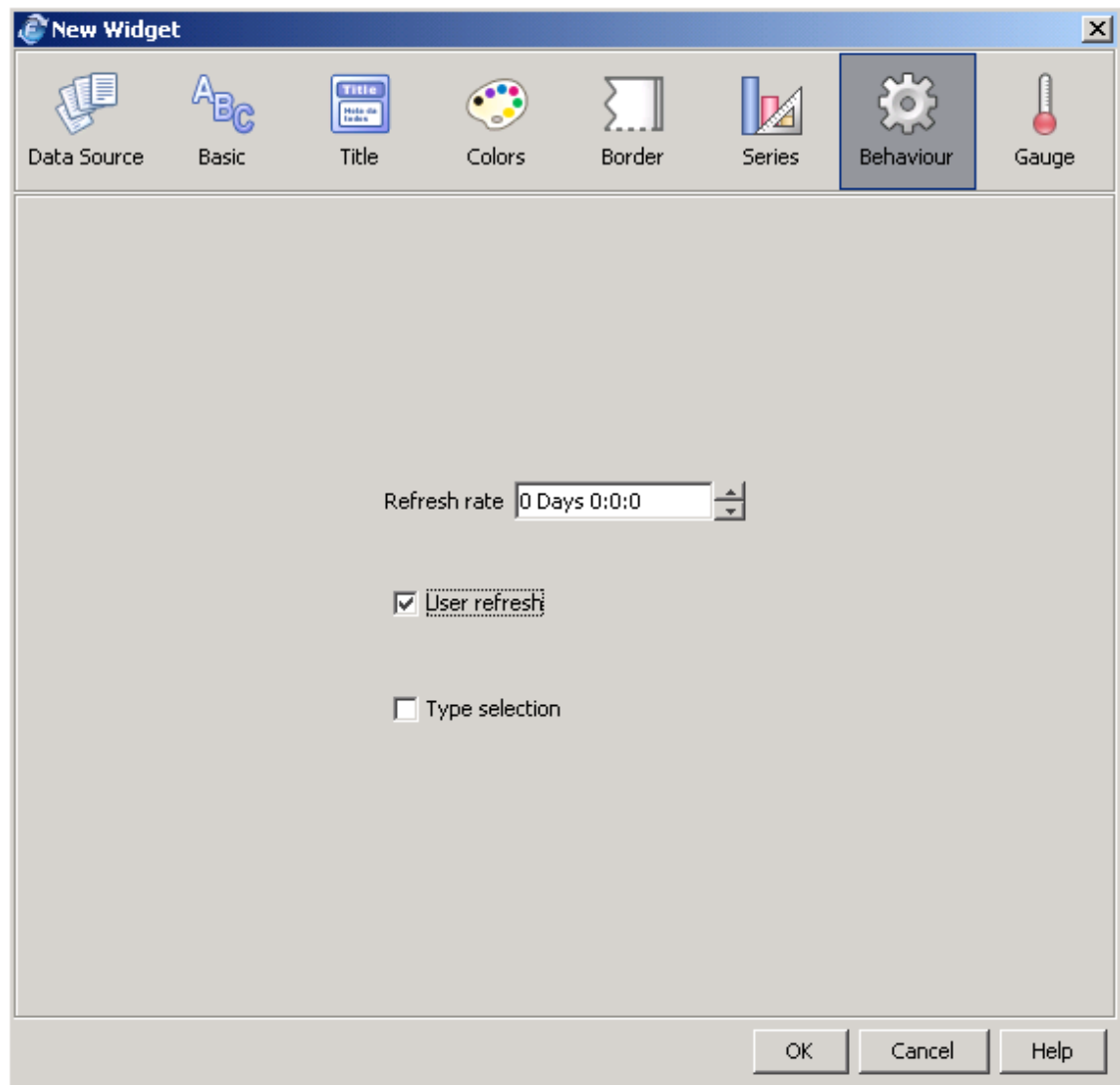
Series properties

Define the widget series properties, their format in font, size and style. The foreground color and the colors to display each serie in the graph.



Behaviour

- Define the graph refresh rate,
- If the user can refresh the dashboard manually at runtime, and
- If the user can change the graph of the dashboard at runtime.



Defining properties for a gauge widget

If the widget you are defining is a gauge, then go to the **Gauge** pane to define its ranges values and colors to display each portion of the gauge graph, *Total*, *Normal*, *Warning* and *Critical*.

New Widget

Data Source Basic Title Colors Border Series Behaviour **Gauge**

Total range

Minimum 0

Maximum 300

Normal range

Minimum 0

Maximum 100

0,255,0

Warning range

Minimum 101

Maximum 200

255,255,0

Critical range

Minimum 201

Maximum 300

255,0,0

OK Cancel Help

Building a BAM Dashboard Manually

BAM and Data Store database schemas

BAM and Data Store database schemas are exactly the same. The only difference is their name. Tables in the BAM database use the prefix *bam_*. For example, *processperformance* for their Data Store schema and *bam_processperformance* for the BAM schema.

Refer to section Data Store and Business Activity Monitoring, to find the database schema and explanation for its tables and fields.

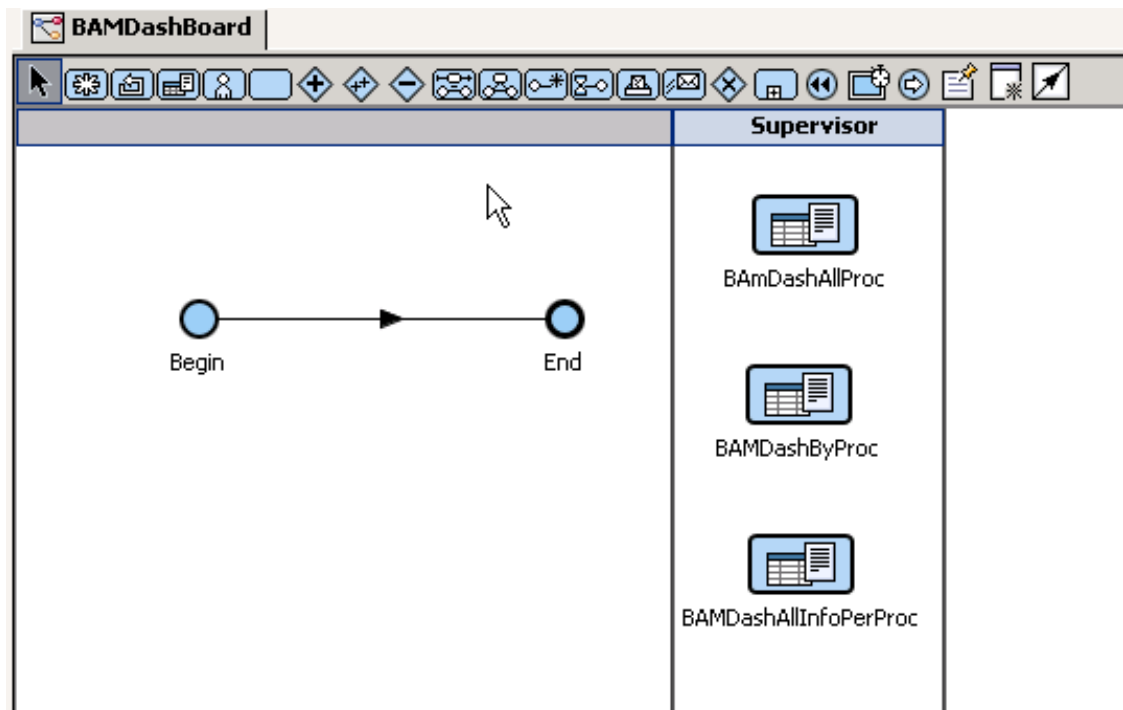
Building a BAM dashboard manually

The only difference in building this dashboard is that the database accessed is the one of the BAM.

This example presents three different ways to show the information contained in the *bam_processperformance* table.

The example is distributed with the installation, its name is *Fuego55DashBoardExample.fpr* and you will find it in the *sample* directory under the FuegoBPM Studio's installation directory.

The process designed to show the built BAM dashboards is named *BAMDashboard*. It has three global activities which implementation type defined is *Show Dashboard*.



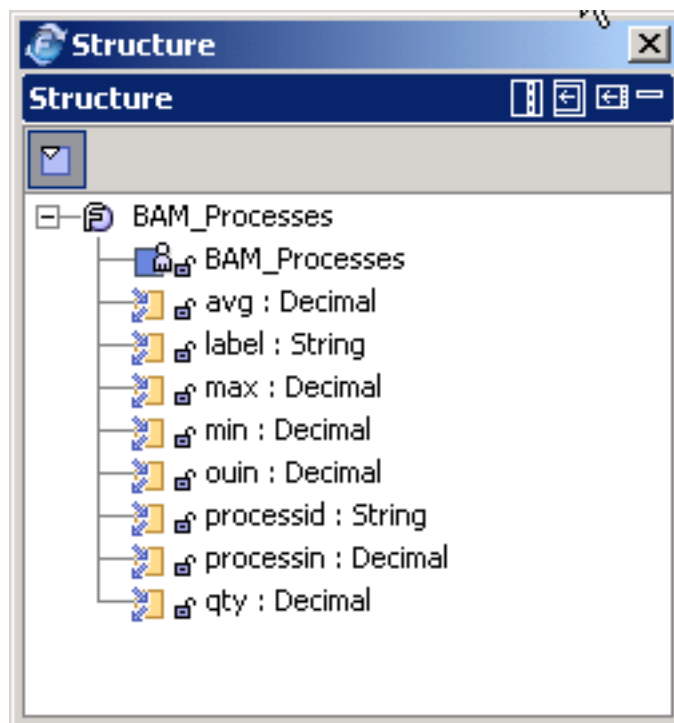
The example has three fuego objects.

- **BAM_Processes**: it inherits from the *bam_process* table, used as base to build the associative array with the calculated information per process.

- ***BAMDashboardProcessPerformance***: contains the presentations with the three different options presented.

BAM_Processes

This Fuego Object inherits behaviour from the database table *bam_process*. It is used to hold the information required to show in the dashboard fuego object per process. It is build only with attributes. An associative array which elements are this fuego object type is build to retrieve all the information, average time, maximum time, minimum time and quantity of instances, at the same time.



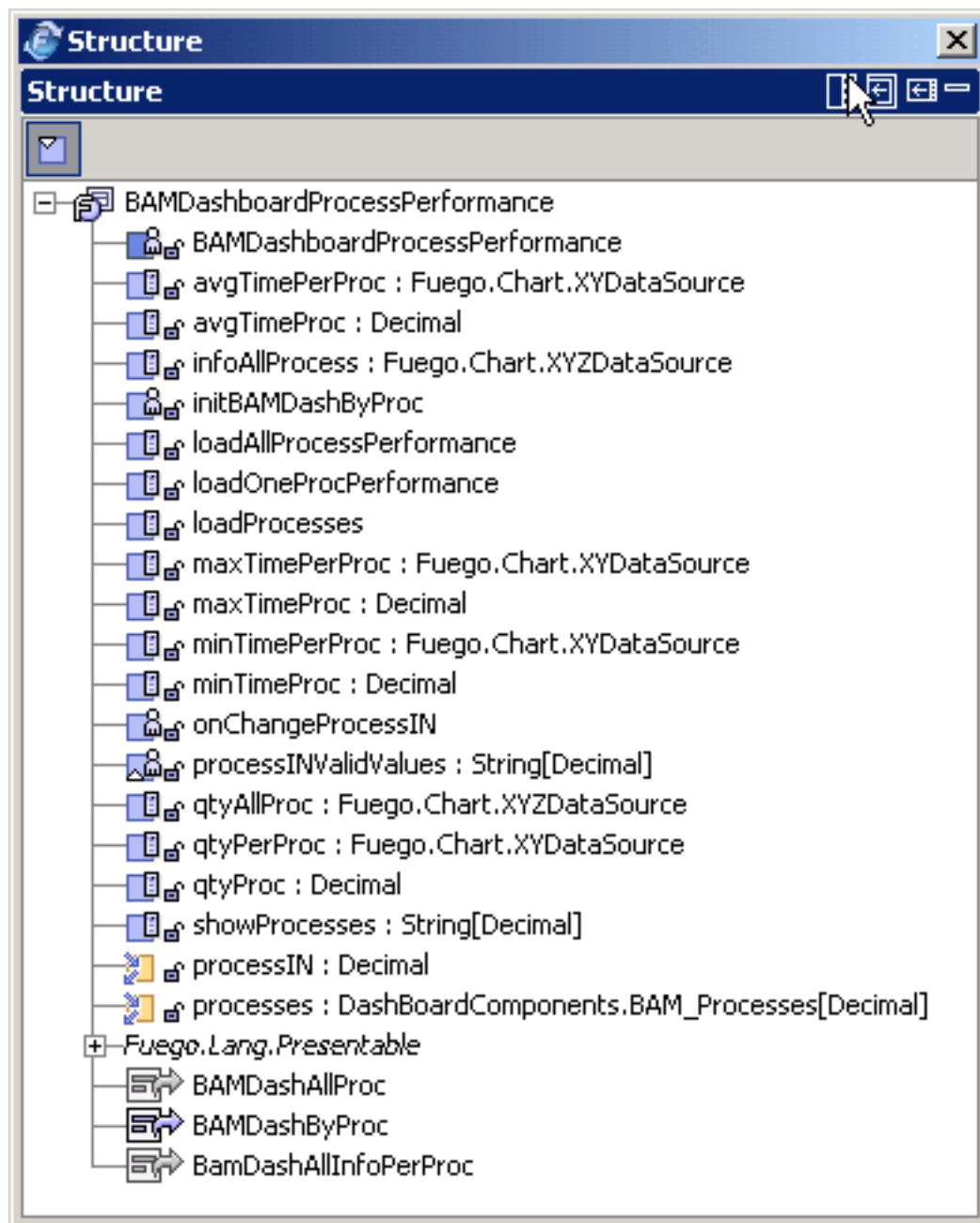
Attributes

- *ouin*: Organizational Unit.
- *processin*: Process numeric identification.

- *processid*: Process alphabetic identification.
- *label*: Process Name.
- *avg* : Holds the average time in which the instances were processed.
- *max*: Holds the maximum time in which the instances were processed.
- *min*: Holds the minimum time in which the instances were processed.
- *qty*: Holds the quantity of processed instances.

BAMDashboardProcessPerformance

This Fuego Object has three presentations with different ways of showing the same information, processing time: average, minimum and maximum and processed instance quantity.



The example is explain by going through each presentation and giving the details of attributes and methods used for each of them.

Building the Examples

BAMDashByProc

This dashboard displays, using gauge components, the information for one processes. The users can select one process from the list of possible deployed processes.

Attributes

- *processIN*: this attribute is used from the presentation to select the process for which the user wants to execute the dashboard.
- *processes*: associative array. The index is the processIN. The element `processes[processIN]` type `DashboardComponents.BAM_Processes[Decimal]` is `BAM_Processes.DashboardComponents.BAM_Processes[Decimal]`.

Methods

- *loadProcesses*: Completes the attribute *processes* with the content of the BAM table *bam_processes*. This table holds the identification of the processes taken into account for the BAM.
- *showProcesses*: method used to show the valid values for the processIN attribute.
- *initBAMDashByProc*: initializes the presentation setting the gauge components as not visible.
- *onchangeProcessIN*: method invoked in the onchange of the processIN component of the presentation. In this method the loading of the information is done according to the process selected and the gauge component are set to visible.
- *loadOneProcPerformance*: selects the maximum and minimum processing times, calculates the average time and calculates the quantity of processed instances. This is only done for the selected process, and, it is store in the associative array *processes*.

- ***avgTimeProc***: returns the average time for the selected process from the associative array already loaded in the *processes* attribute.
- ***maxTimeProc***: returns the maximum time for the selected process from the associative array already loaded in the *processes* attribute.
- ***minTimeProc***: returns the minimum time for the selected process from the associative array already loaded in the *processes* attribute.
- ***qtyProc***: returns the quantity of processed instances for the selected process from the associative array already loaded in the *processes* attribute.

Presentation

Components

- ***avgTime***: gauge component. Invokes the method *avgTimeProc*.
- ***maxTime***: gauge component. Invokes the method *maxTimeProc*.
- ***minTime***: gauge component. Invokes the method *minTimeProc*.
- ***qtyProc***: gauge component. Invokes the method *qtyProc*.

Buttons

- ***exit***: cancel action button.

Global Activity that invokes it

The activity is named *BAMDashByProc* and its main task definition is as shown in the picture below:

Main task - Activity BAMDashByProc

BAMDashByProc

Implementation type
Show Dashboard

Dashboard Display Properties
Allows to show the dashboard in a new window of the browser
☒ Open dashboard in new window
Shows the full control toolbar of the browser for the new window (only if Open in New Window property is enabled)
☐ Show browser's full control toolbar

Component method
Fuego Object Name
Select a Fuego Object and a presentation to be shown as a Dashboard in the Work Portal.
Component name:
DashBoardComponents.BAMDashboardProcessPerformance
Browse

Fuego Object Presentation
Presentation name BAMDashByProc

OK Cancel Help

Second Example: Presentation BAMDashAllInfoPerProc

This FuegoBPM dashboard, shows the information for each process.

Attributes

- *processes*: associative array. The index is the processIN. The element type is *BAM_Processes.DashboardComponents.BAM_Processes[Decimal]*.

Methods

- ***loadAllProcessPerformance***: loads the information about time and processed instances for all the processes already loaded in the *processes* associative array of the *utils* attribute.
- ***qtyPerProc***: retrieves an *XYDataSource* containing the process name and processed instance quantity.
- ***maxTimePerProc***: retrieves an *XYDataSource* containing the process name and maximum processing time.
- ***minTimePerProc***: retrieves an *XYDataSource* containing the process name and minimum processing time
- ***avgTimePerProc***: retrieves an *XYDataSource* containing the process name and average processing time.

Presentation

Components

- ***avgTime***: bar chart component. Invokes the method *avgTimePerProc*.
- ***maxTime***: bar chart component. Invokes the method *maxTimePerProc*.
- ***minTime***: bar chart component. Invokes the method *minTimePerProc*.
- ***qtyProc***: bar chart component. Invokes the method *qtyPerProc*.

Buttons

- ***exit***: cancel action button.

Global Activity that invokes it

The activity is named *BAMDashAllInfoPerProc* and its main task

definition is as shown in the picture below:

Third Example: Presentation BAMDashAllProc

This fuego dashboard, shows the information for all processes.

Attributes

- *processes*: associative array. The index is the processIN. The element type is *BAM_Processes.DashboardComponents.BAM_Processes[Decimal]*.

Methods

- *loadAllProcessPerformance*: loads the information about time and processed instances for all the processes already loaded in the *processes* associative array of the *utils* attribute.
- *infoAllProcess*: retrieves an *XYZDataSource* containing the process name as row and maximum, minimum and average processing time as columns.
- *qtyAllProcess*: retrieves an *XYZDataSource* containing the process name as row and the quantity of processed instances as columns.

Presentation

Components

- *infoAllProcess*: bar chart component. Invokes the method *infoAllProcess*.
- *qtyAllProc*: bar chart component. Invokes the method *qtyAllProc*.

Buttons

- *exit*: cancel action button.

Global Activity that invokes it

The activity is named *BAMDashAllProc* and its main task definition is as shown in the picture below:

Main task - Activity BAmDashAllProc

BAmDashAllProc

Implementation type
Show Dashboard

Dashboard Display Properties
Allows to show the dashboard in a new window of the browser
☒ Open dashboard in new window
Shows the full control toolbar of the browser for the new window (only if Open in New Window property is enabled)
☐ Show browser's full control toolbar

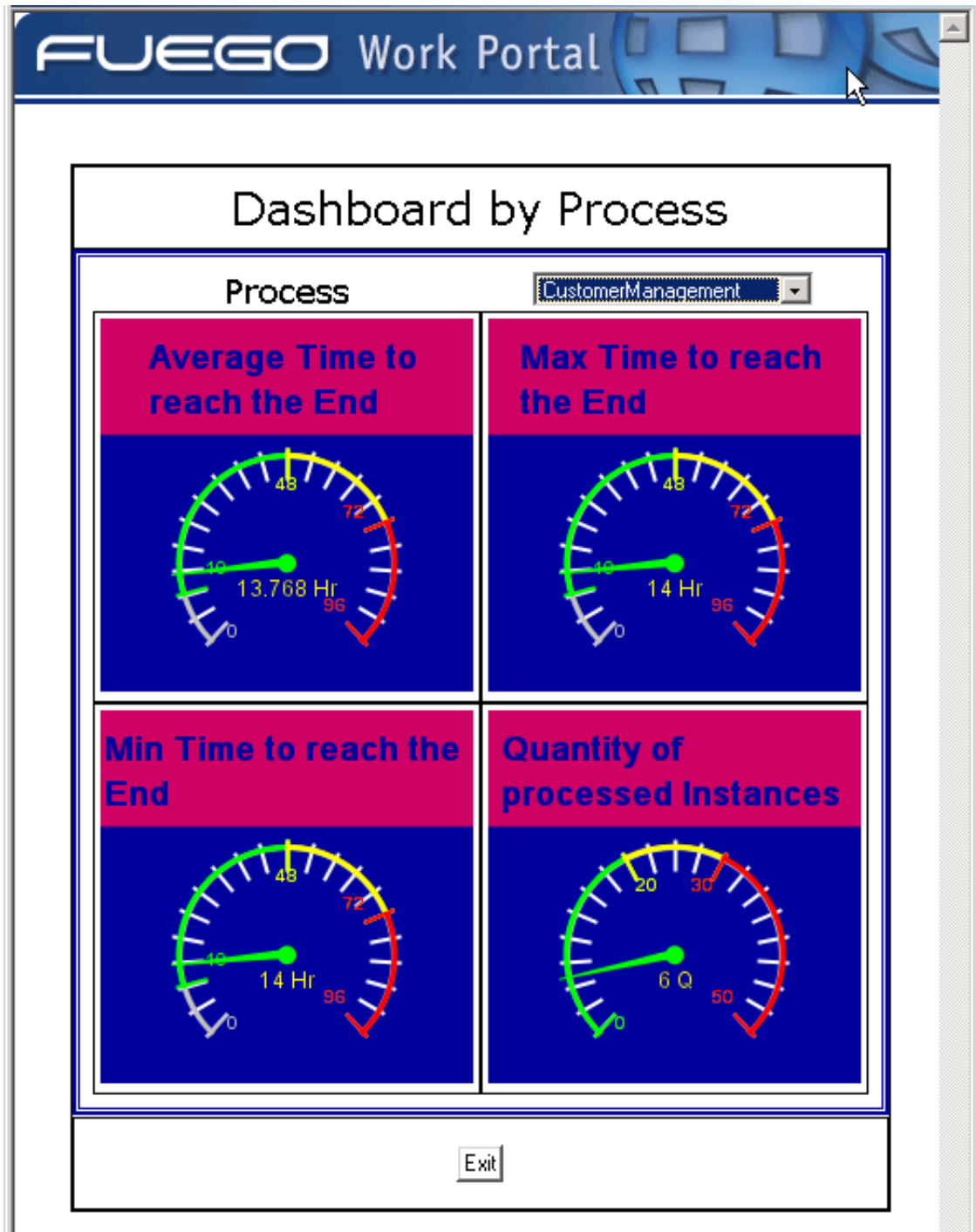
Component method
Fuego Object Name
Select a Fuego Object and a presentation to be shown as a Dashboard in the Work Portal.
Component name:
DashBoardComponents.BAMDashboardProcessPerformance
Browse
Fuego Object Presentation
Presentation name BAMDashAllProc

OK Cancel Help

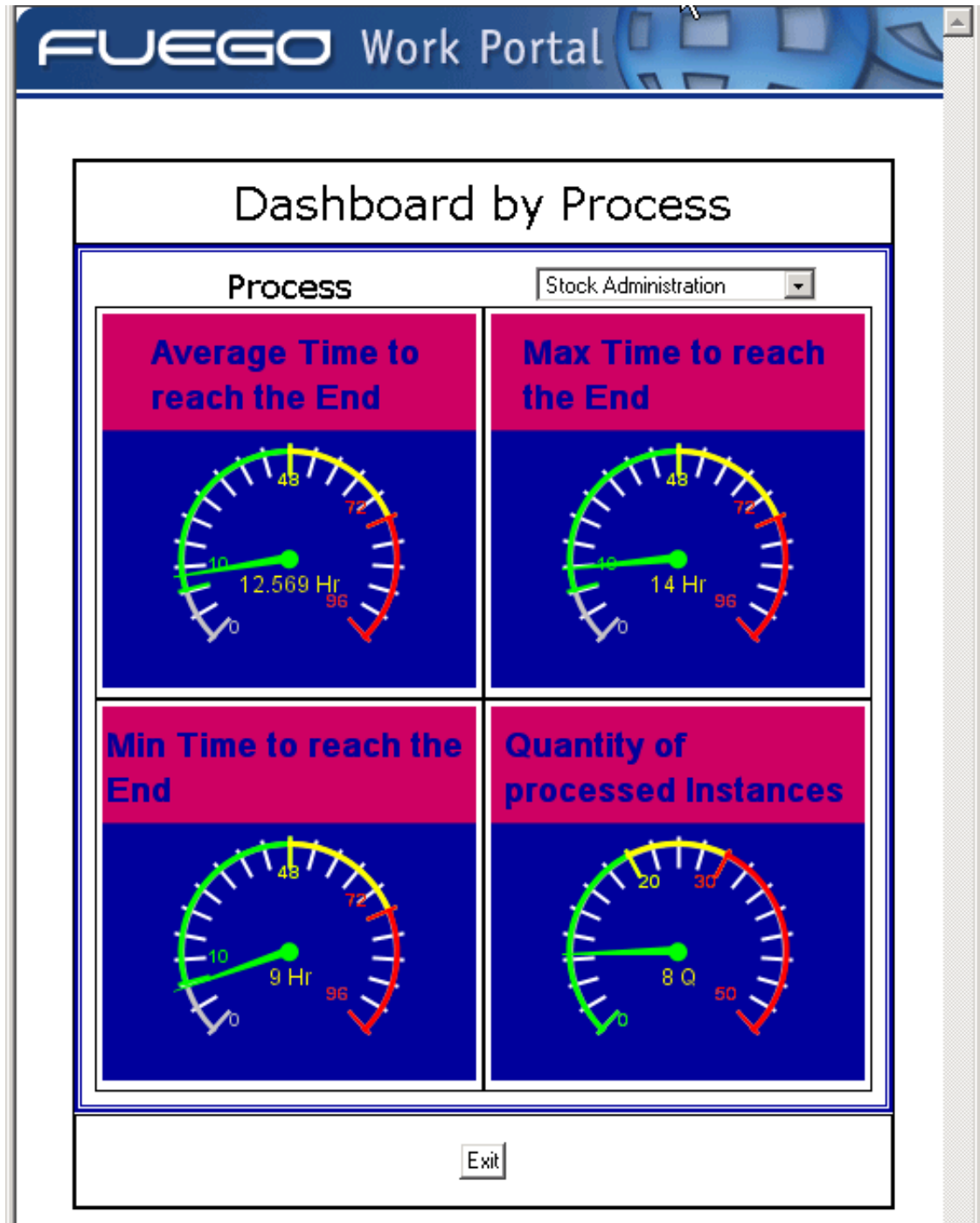
Executing the dashboards

BAMDashByProc

Executing the dashboard for the *Customer Management* process.

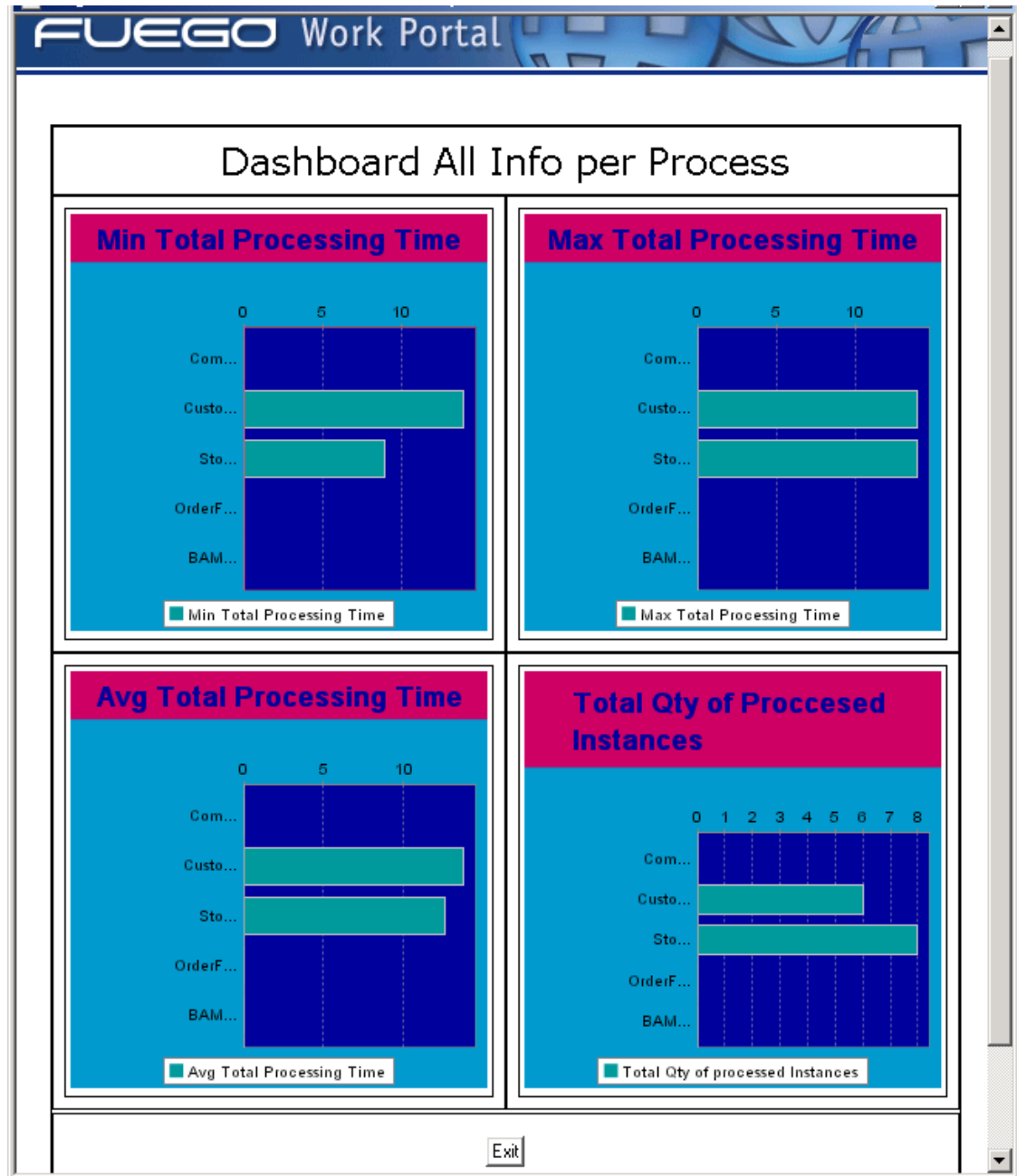


Executing the dashboard for the *Stock Administration* process.



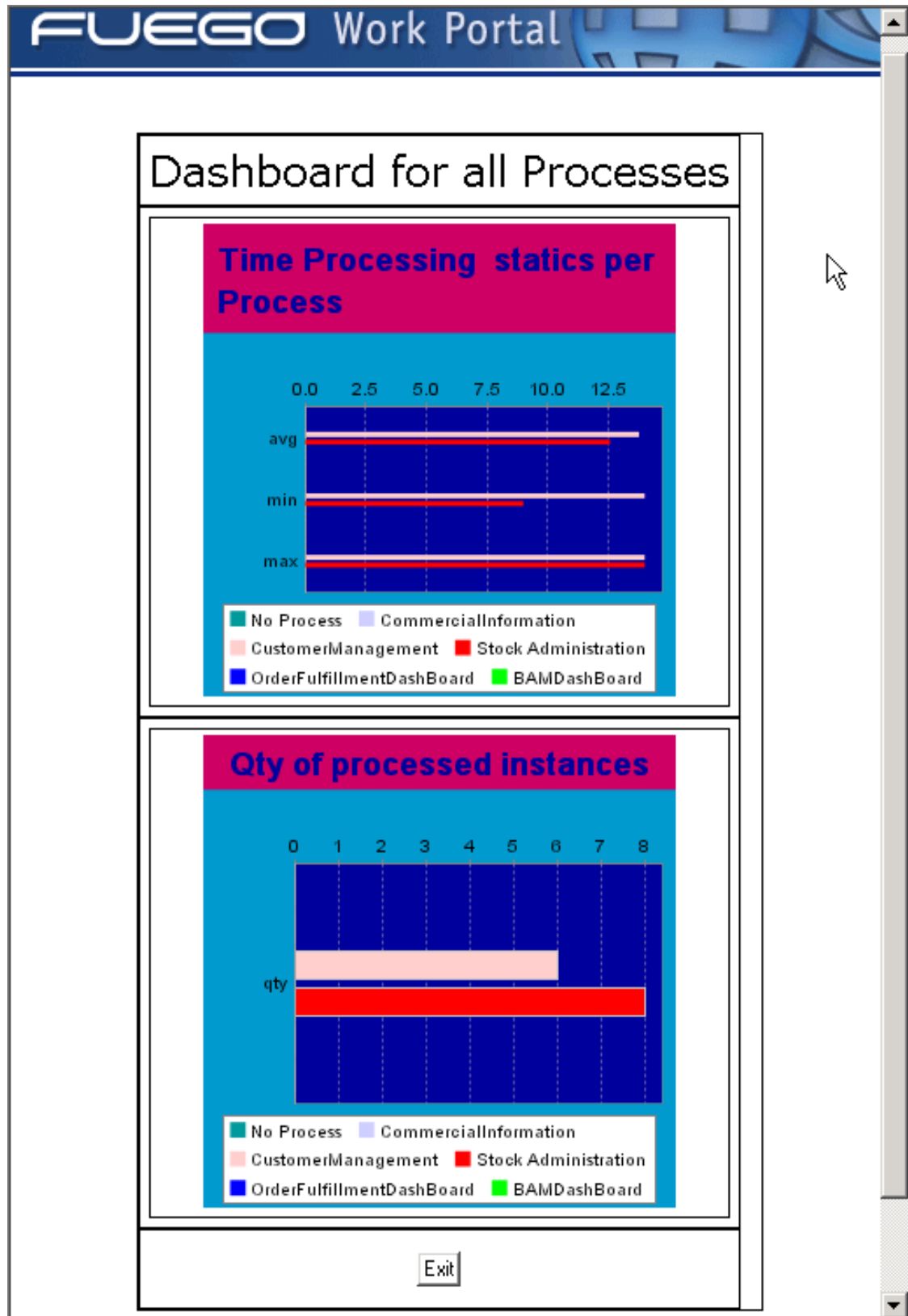
BAMDashAllInfoPerProc

Executing the dashboard that shows each Key Performance Indicator for all processes.



BAMDashAllProc

Executing the dashboard that shows all process information.



Using the "ClientBusinessProcess" block to build a FuegoBPM Dashboard

The *DashBoardExample.fpr* project distributed with the installation, contains also an example using the *ClientBusinessProcess* Fuego Block to access the FuegoBPM Server data.

This FuegoBlock should be used only in **client side** methods invoked from Fuego Object presentations created for Dashboards. It connects to the current PAPI session using the credentials (user and password) available from the dashboard execution. Through this BusinessProcess you can get the ProcessService that allows to perform operations over processes, instances and filters.

Example presentation

The same *CustomerManagement* process is used in this case as in the Fuego Object examples and in the other dashboard examples.

Processes

CustomerManagement: The process was modified to include the *handling exception* flow. As the dashboard example for this case is to show the *Quantity of Instances in Exception*.

OrderFulfillmentDashboard: This example was added to this project in the *Instances in Exception* global activity.

Catalog

To build this example three new components where added to the catalog.

StatusType: This is an enumeration used to set the status in an external instance variable. This external variable is used later in the dashboard invoked method to filter the instances with the status in *EXCEPTION*.

CustomerException: This exception is used from the *Final Customer*

Info Check interactive activity in the *CustomerManagement* process. The exception is thrown when the final user indicates that the information is not ok. By doing this, we forced these instances to go to the *handling exception* flow.

DashboardUsingClientBusinessProces: This is Fuego Object Dashboard used to show the quantity of instances in exception.

Building the *DashboardUsingClientBusinessProces* Fuego Object

This object has only one method and only one presentation.

Build the method that retrieves the number of instances

Create a method to the Fuego Object. This method's code, in our example named *fillInstancesInException*, retrieves the quantity of instances in exception. Set the method property *Return Type* as **Int**.

Step by step, the code is:

1. connect to the server where the *CustomerManagement* process is deployed.
2. create an instance filter for the process we have connected.
3. obtain the set of external variables of the process.
4. build the instance filter as wished.
5. get the instances that match the filter, and
6. return the number of instances that matched the filter.

```
// #1
clientBP as ClientBusinessProcess
clientBP=ClientBusinessProcess()
```

```
connectTo clientBP
    using processId = "CustomerManagement"

instF as InstanceFilter
instF = InstanceFilter()

//#2
create instF
    using processService =
        clientBP.processService

//#3
variables = clientBP.processService.vars

//#4
addAttributeTo instF
    using variable = "PREDEFINE_ACTIVITY",
        comparator = Comparison.IS,
        value = "ExceptionHandler"

addAttributeTo InstanceFilter
    using variable = "estado",
        comparator = Comparison.IS,
        value = "EXCEPTION"

//#5
result = getInstancesByFilter(clientBP, instF)

//#6
return length(result)
```

where :

- **result** is a local variable defined as *Instance[]*, and
- **variable** is a local variable defined as *String[]*.

Tip



To check if you are defining correctly the filters go to the *Search* in the *FuegoBPM Work Portal* and perform a search with the conditions you need. Then edit the source code of the result search page and at the bottom you'll see how the filter is constructed. You should program your filter according to this one.

Build the dashboard presentation

1. Create a presentation to the Fuego Object. And add a dashboard component of the *gauge* type as we are showing the *number of instances in exception*.
2. Select the *Graphic Type*, **Thermometer** or **Meter**, according to your preferences.
3. Set the component properties, color, size, boundaries.
4. Select the *Method Invoked* to the component. In this case only one method is shown as the Fuego Object only has the *fillInstancesInException* method defined which return an **Int**.

The Fuego Object dashboard is ready to be invoked from the global activity.

Build the method that retrieves the number of instances COMPLETED or in the End Activity

To get the number of instances in the End Activity or those which are *COMPLETED*, you must set the filter scope with the status *ONLY_COMPLETED* and the participants roles set to *ALL*. The roles must be set to *ALL* because if you search instances assigned to you or to your roles, *COMPLETED* or *ABORTED* instances are not retrieved. The method code would be like the following:

```
connectTo ClientBusinessProcess
    using processId = "CustomerManagement"

create InstanceFilter
    using processService =
        ClientBusinessProcess.processService

InstanceFilter.searchScope =
    SearchScope(ParticipantScope.ALL,
        StatusScope.ONLY_COMPLETED)
```

```
result = getInstancesByFilter(  
    ClientBusinessProcess, InstanceFilter)  
  
return length(result)
```

Predefined attributes name to use to set or add attributes are:

Attribute	Predefined name
Activity	PREDEFINE_ACTIVITY
Activity deadline	PREDEFINE_ACTIVITY DEADLINE
Copy	PREDEFINE_COPY
Creation time	PREDEFINE_CREATION TIME
Deadline	PREDEFINE_DEADLINE
Description	PREDEFINE_DESCRIPTION
Has attachments	PREDEFINE_HAS_ATTACHMENTS
Has notes	PREDEFINE_HAS_NOTES
Initiator	PREDEFINE_AUTHOR
Participant	PREDEFINE_PARTICIPANT
Priority	PREDEFINE_PRIORITY
Process deadline	PREDEFINE_PROCESS DEADLINE
Received	PREDEFINE_RECEIVED TIME
State	PREDEFINE_STATUS

How to drill-down to Instance Data from a BAM Dashboard

A portal view containing the instance data can be opened from a dashboard as well as from any Fuego Object.

To do so you will have to implement a *drill-down* method to invoke in the *On Click* or *On Control Click* property of the widget in the

presentation.

You click or control click on the graphic to drill down on an specific bar or slice that represents some attribute, for example, the activities, the company departments, etc. When you click, the attribute value is passed as argument to the method invoked in the *OnClick* or *On control click*.

All the attributes defined in the view as parametric, have to be set using the *setParametricValueTo* method, as shown in the example below, section *Method drillDown*. In this example the activity is set as parametric.

You could set as parametric both, predefined attributes as activity, participant, description, etc., or external variables.

Predefined attributes name to use to set or add attributes are:

Attribute	Predefined name
Activity	PREDEFINE_ACTIVITY
Activity deadline	PREDEFINE_ACTIVITY DEADLINE
Copy	PREDEFINE_COPY
Creation time	PREDEFINE_CREATION TIME
Deadline	PREDEFINE_DEADLINE
Description	PREDEFINE_DESCRIPTION
Has attachments	PREDEFINE_HAS_ATTACHMENTS
Has notes	PREDEFINE_HAS_NOTES
Initiator	PREDEFINE_AUTHOR
Participant	PREDEFINE_PARTICIPANT
Priority	PREDEFINE_PRIORITY
Process deadline	PREDEFINE_PROCESS DEADLINE
Recieved	PREDEFINE_RECEIVED TIME
State	PREDEFINE_STATUS

Example Instance by Activities

This example is included in the *DashboardDrilldownInstanceData* project distributed with FuegoBPM Studio. You can find it in the *sample* directory of the FuegoBPM Studio installation.

The FuegoBPM Dashboard defined is a *gauge* type. It shows the quantity of instances per activity. By drill-down clicking on each slice of the pie, a view is opened showing the instances data.

Method *instanceByActivityQuery*

This method returns an *iterator* containing the quantity of instances by activity, using the *BAMQuery*. When this method is invoked the query specified is executed.

```
//SQL query
query as String
query = "select activityId, SUM(quantity) as quantity
from bam_Workload, bam_Activities
where bam_Workload.activityIn in (select activityIn
from bam_Activities where processIn in (select processIn
from bam_Processes where processId like ?)) and
    snapshotTime=(select lastSnapshotTime
from bam_LastSnapshot) and
    bam_Workload.activityIn=bam_Activities.activityIn
group by activityId"

//SQL query arguments
queryArgs as Any[]
queryArgs = ["TestDashboards%"]

//Executing query
bamQuery as BAMQuery
bamQuery = BAMQuery()
iterator as Iterator(Any[String])
iterator = execute(BAMQuery, sql : query,
    inParameters : queryArgs)
return iterator
```

Method *instancesByActivityDataSource*

This method retrieves the data to be shown in the dashboard widget

obtained by executing the method *instanceByActivityQuery* shown below. This is the method invoked in the gauge component, property *Method invoked*.

```
//Getting data from DB
iterator as Iterator[Any[Any]]
iterator = instancesByActivityQuery()

//Filling data source
result as Fuego.Chart.XYDataSourceImpl
result = Fuego.Chart.XYDataSourceImpl("activity");
for each row in iterator
do
  addValue result using value = Decimal(row["quantity"]),
                        columnHeader = String(row["activityId"])
end
return result
```

Method *drillDown*

This method is invoked in the presentation when clicking on a pie slice. It is related to the gauge component of the presentation in the *On Click* method property. The method opens in the same window of the dashboard the view containing the instance data.

```
connectTo ClientBusinessProcess
  using processId = "/TestDashboards"

//ClientBusinessProcess.processService

do
  //Parametric filter ////////////////////////////////////////////
  instanceFilter = getFilterFor(
    ClientBusinessProcess.processService,
    viewId : "paramAct")

  setParametricValueTo instanceFilter
  using variable = VarDefinition.activityid,
    value = activity
  ////////////////////////////////////////////

  //View URL //////////////////////////////////////////////////
  result = getQueryStringFor(
    ClientBusinessProcess.processService,
```

```
viewId : "paramAct")
//////////

//Just to open the url
openURL this
using url = result,
    newWindow = false,
    standardBrowser = false

on e as Java.Lang.Exception
    showError this
    using error = e.message
end
```

As this example opens the information in the user defined view *paramAct* which is parametric, the filter has to be set as parametric (portion of code marked between the comment *// Parametric filter ///*). In this example the activity is set as parametric.

The same attributes have to be define as parametric in the view definition. See the *paramAct* view definition in section **View paramAct**.

View *paramAct*

As shown in the picture below, the *paramAct* view is defined in the *Conditions* area of the view panel as *parametric by Activity* by selecting the *Is parametric* check box and *Activity* as condition.



See more information about the *openURL* and *show* methods in FuegoBPM Dashboard Standard Methods and Fuego Object Methods.

Recommendations

When to use PAPI

When you are sure that you are accessing information that PAPI has cached, PAPI can be faster.

Nevertheless, have in mind that:

- PAPI does not cache terminated instances.
- In processes with HUGE number of instances PAPI does not cache, so it would be accessing the server database to get the instance information.

When to use BAM tables

BAM tables are better to handle consolidated data.

- They can be offloaded to a different machine to avoid performance problems.
- They can consolidate information from different servers.
- You can work over the BAM database adding Indexes, Triggers, etc., to manipulate the information and the access to it or to improve performance.

Chapter 18. Version Control System

Version Control System

The Version Control System (VCS) implements a client to perform version control operations like add, remove, commit, and update into a common repository. It is designed to operate with different VCS providers such as CVS, Source Safe, and others.

FuegoBPM VCS is very useful for the administration of FuegoBPM Studio projects. This includes processes, components, and special files.

By using VCS, you can record the history of your project. For example, bugs sometimes creep in when software is modified and you might not detect the bug until some time after you made the modification. Using VCS, you can easily retrieve old versions to see exactly which change caused the bug. Sometimes this can be a great help.

VCS also helps you if you are a member of a group working on the same project. But take into consideration that VCS is not:

- a build system
- a substitute for management
- a substitute for developer communication

Also, VCS does not have a built-in process model to ensure that changes or releases go through various steps, with various approvals as needed.

CVS can be used from inside FuegoBPM Studio. If you are using another VCS provider, as VSS, you must administrate your project from outside FuegoBPM Studio. In appendix A, Implementing FuegoBPM Studio VCS using Visual SourceSafe you can find how to

operate with VCS provider. The indications on what files you must administrate are valid to any VCS provider you choose to use to administrate your FuegoBPM Project.

Project Source elements

There are several source components that participate in a typical FuegoBPM project. Below you will find a brief explanation of each type.

- FuegoBPM Processes (.xpdI files), Procedures (.xadI files) and Screenflows (.xsdI)
- FuegoBPM Components: Fuego Objects, Enumerations and Exceptions (*.xcdI)
- External Components: External Resources (*.xml) and Libraries (*.jar)
- Fuego Configuration (server, organization, preferences, and views - *.xml)
- FuegoBPM Simulation: Process Simulation Model (*.xsim) and Project Simulation model (*.xpsi)
- FuegoBPM Dashboard Template widgets (*.xwdI)
- Java Servers Pages used as an interface between the end user and the Fuego Object (*.jsp) and their resources.

Not all the files included under a FuegoBPM project need to be stored into the repository. Some of these files contain information used by FuegoBPM Studio to improve its performance, or build and execute its runtime environment.

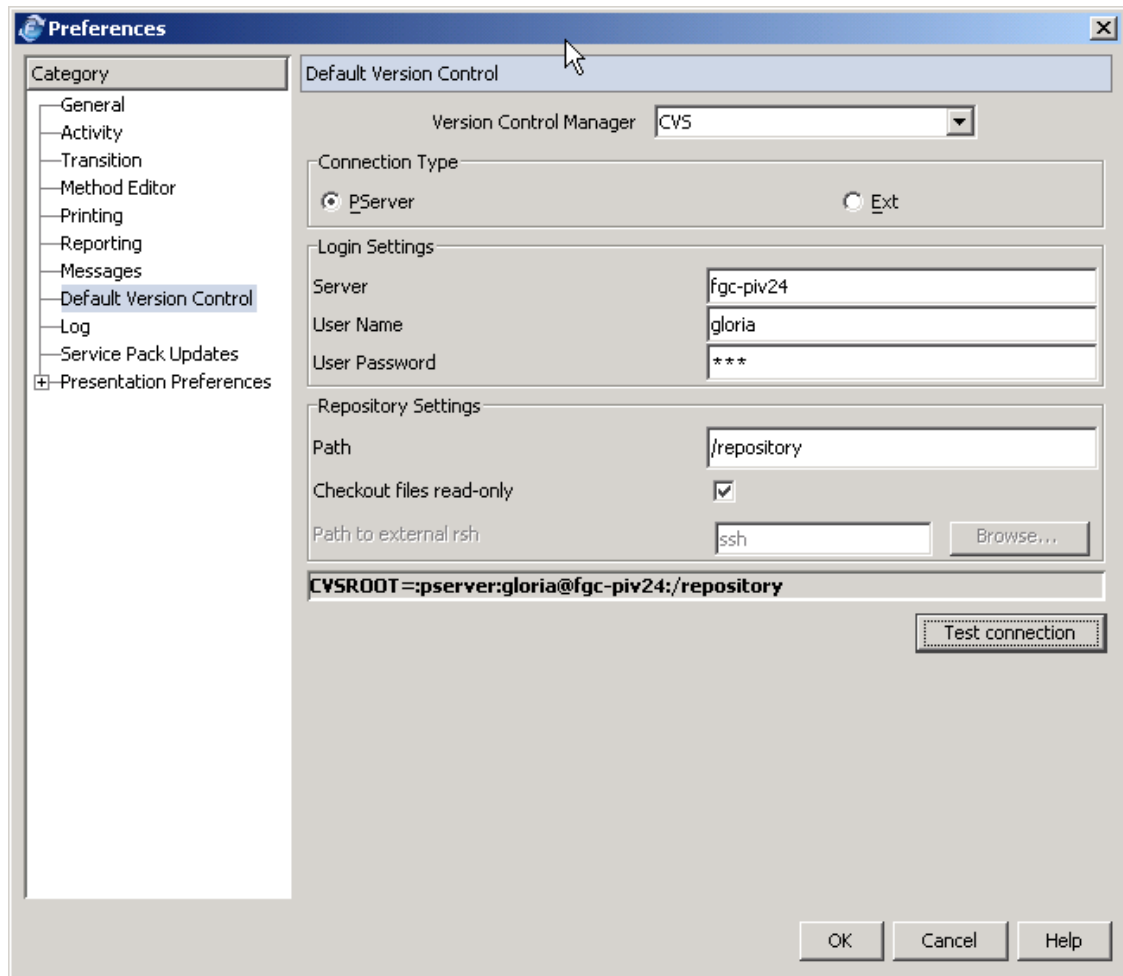
There are other folders that are required to be checked into VSS. These folders are:

lib	Contains all external libraries used by the project (i.e. Microsoft SQL Server JDBC Drivers)
config	Contains configuration XML files
componentCatalog	Contains all the components defined in the project
processes	Contains the definition files for processes, procedures and screen flows
dashboard	Contains the definition files for template dashboard widgets
simulation	Contains the definition files for Process Simulation Models and Project Simulation model
webroot	Contains the Java Servers Pages used as an interface between the end user and the Fuego Object

VCS Properties

How to configure VCS in the Studio

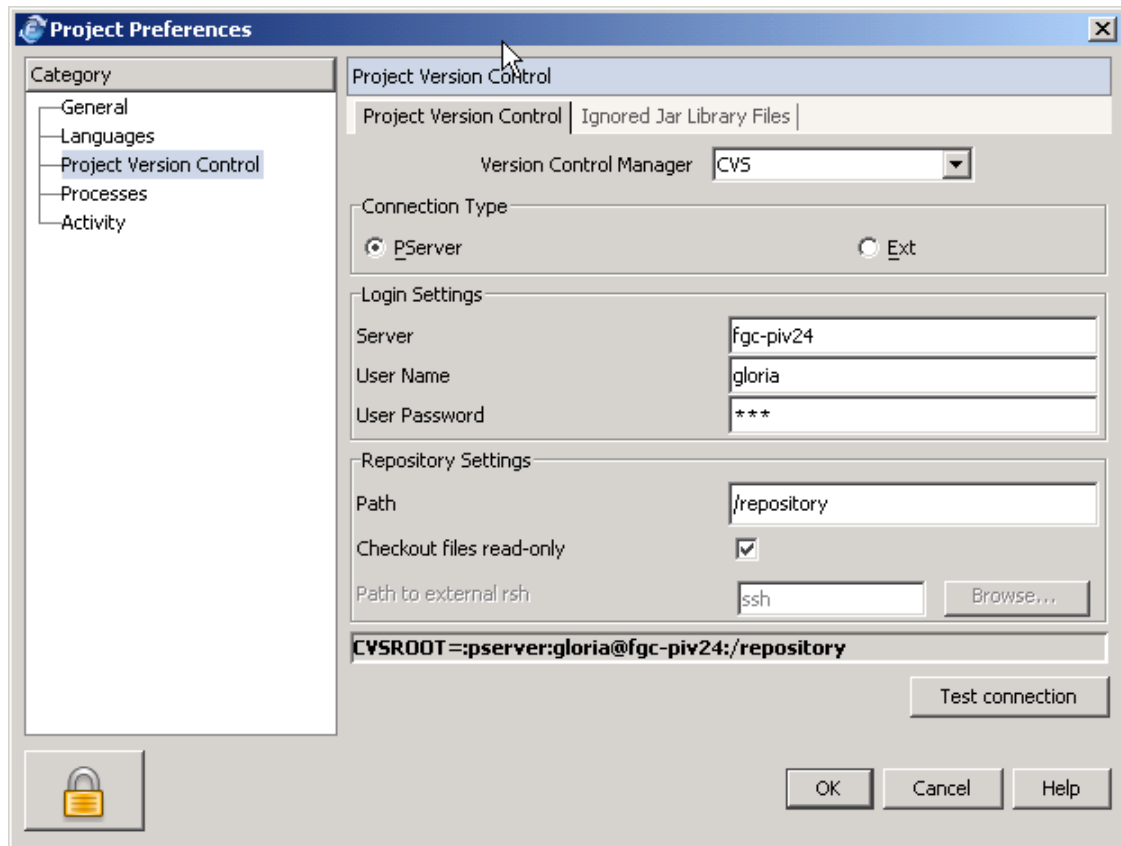
VCS can be configured in the Studio Preferences, **Preferences** option in the **File** menu. The provider to be used is defined in this option. Go to the *Default Version Control* category on the right side of the dialog and set the corresponding values to your VCS.



The configuration you define in this option is the default VCS configuration for your FuegoBPM Studio, but each project VCS's configuration can vary. If the project configuration has not been defined when you define it, you can use the default definition or set a different one.

How to configure VCS for a Project

To define a different VCS configuration from the default one for a Project, go to the main **File** menu and select the **Project Preferences** option. Select the *Project Version Control* category and set the corresponding values for this particular project.



Checkout for files read-only: Select this check box if you want to check out your project as read-only. Have in mind, that if you check it out as read-only, you are not able to update the files and check them back into the repository.

Ignored Jar Library Files tab: This tab shows the jar library files added to the project. Select those that you want to administrate with the VCS from FuegoBPM Studio. By default, any of the jars will be marked to administrate with VCS.

Using CVS as Repository

General Description of CVS

CVS is one of the version control systems you can use as a repository. The CVS repository stores a complete copy of all the files and directories that are under version control.

Normally, you never have direct access to any of the files in the repository. Instead, you use CVS commands to get your own copy of the files into a working directory and then work on that copy.

When you have finished a set of changes, you check (or commit) the files back into the repository. The repository now contains the changes you have made. It also records exactly what you have changed, when you have changed it and other similar information. Note that the repository is not a subdirectory of the working directory, or vice versa; they should be in separate locations.

Using FuegoBPM Studio VCS, you will interact with the CVS repository using FuegoBPM Studio capabilities.

Implementing the repository with CVS

This section explains how to configure and use the CVS implementation. After selecting CVS as the VCS provider, you must configure the CVS properties:

The screenshot shows a 'Version Control Manager' dialog box with a dropdown menu set to 'CVS'. Below this, there are three main sections: 'Connection Type', 'Login Settings', and 'Repository Settings'. In the 'Connection Type' section, the 'PServer' radio button is selected, and the 'Ext' radio button is unselected. The 'Login Settings' section contains three text fields: 'Server' with the value 'FGC-PIV24', 'User Name' with the value 'FGC', and 'User Password' with the value '*****'. The 'Repository Settings' section contains a 'Path' text field with the value '/studio'. Below these sections, a text field displays the generated CVSROOT string: 'CVSROOT=:pserver:FGC@FGC-PIV24:/studio'. A 'Test connection' button is located at the bottom right of the dialog.

Version Control Manager	
CVS	
Connection Type	
<input checked="" type="radio"/> PServer	<input type="radio"/> Ext
Login Settings	
Server	FGC-PIV24
User Name	FGC
User Password	*****
Repository Settings	
Path	/studio
CVSROOT=:pserver:FGC@FGC-PIV24:/studio	
Test connection	

Special Note

If you are configuring *cvs ext* with SSH RSA based authentication and it requires an user intervention to input a password, it will depend on the operative system how it is asked to be input.

In *Linux*, it depends on how the *SSH_ASKPASS* environment variable is set. If you run FuegoBPM Studio on a terminal and the SSH is configured to ask for a password (for example not using an ssh agent), you might think that FuegoBPM Studio is not responding, but in fact it is waiting for an user to input the password.

About the Manager for CVS:

- **Connection Type**

- *PServer*: Available.
- *Ext*: Available.

- **Login Settings**

- *Server*: Name of the server where the CVS Repository is running.
- *User Name*: Name of the user in repository.
- *User Password*: Password for the user in the repository.

- **Repository Settings**

- *Path*: Indicate the root path of the repository. The catalog and processes directories are located in this path.

All the above information will form the **CVSROOT** variable.

Once you have configured the CVS connection settings, test the connection by clicking the **Test** button to ensure that the configuration is correct.

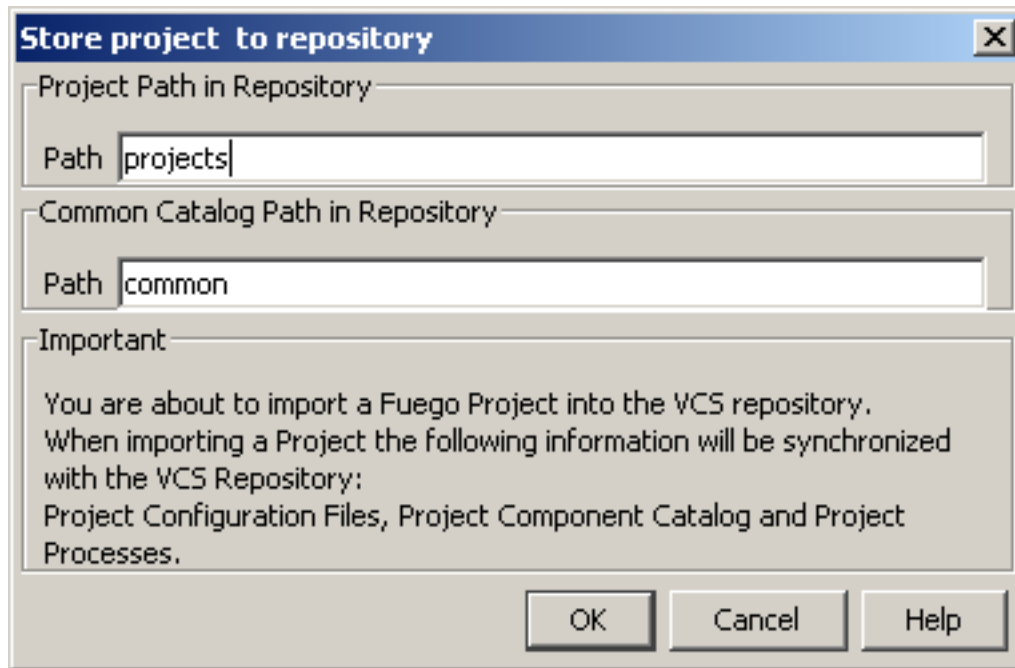
Supported Operations

Store Project to the Repository

The first step to begin using the VCS functionality is to add the project to the repository. Do this:

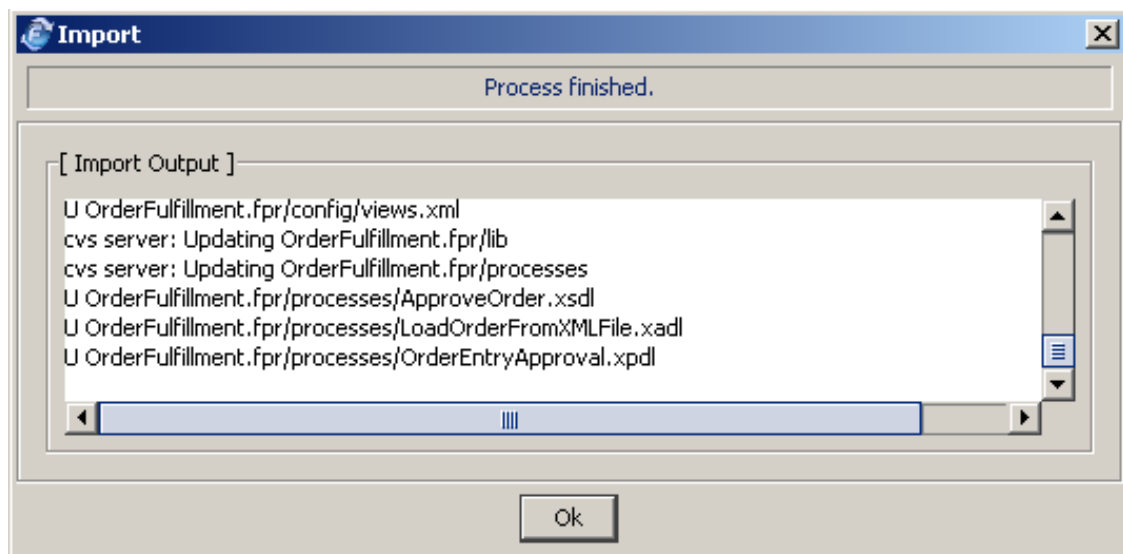
- From the menu opened by right-clicking on the Project entry in the project navigator, **VCS** option and then **Store Project to Repository**, or
- from the main **File** menu, **Store Project to Repository** option

The *Store Project to Repository* dialog is opened, showing the Project and Common catalog paths where your project will be updated. These paths are *projects* and *common* by default, but you can change them if you want.



If you are storing a new project when you have already stored another, the Common catalog entry will not be shown as it already exists.

A dialog showing the updating actions is opened. Once the updating has finished, click **OK** to continue.



Storing the project to the repository adds and commits the whole

project, processes, and components. After performing the *Store project to repository*, you do not need to perform an *add* and *commit*. However, you will need to *add* and *commit* any new element added to the project after the initial store.

Please refer to Common Catalog for further information.

Storing a Project in CVS

The next step after configuring the CVS properties is to **Store the project to repository** available in the context menu of the project tree or in the main **File** and **VCS** menu option. This operation imports the whole project into the CVS repository.

Let's see a general example of how the project is stored into a CVS repository:

Project name: Test.fpr

CVSROOT directory: */usr/myCVSRepository*

If the project path name is left blank, the project directory will be created directly under CVSROOT directory: */usr/myCVSRepository/Test.fpr*.

If the field is filled in with a directory hierarchy like *myprojects/develop*, the project will be imported in */usr/myCVSRepository/myprojects/develop/Test.fpr*.

In both cases, a new module is created in the CVS repository in the *CVSROOT/modules* files:

Test.fpr Test.fpr

or

Test.fpr /usr/myCVSRepository/myprojects/develop/Test.fpr

Note



For further information, see How modules are created in CVS.

After the project is imported to the repository, a check out operation is done on this project so now your local project is administrated.

Add

Adds a component or a module to the repository. To confirm the addition, the commit command must be executed. If the added file is a process folder or a catalog user module, all internal files will be added recursively.

To add an element to the repository, select the **Add** option from the **VCS** menu by right-clicking on the element.

Available for:

- Project.
- Processes.
- Folders that represent a directory. Directories can be added to the repository but the commit operation is required.
- One individual process.
- Catalog.
- Catalog Module.
- One individual component.

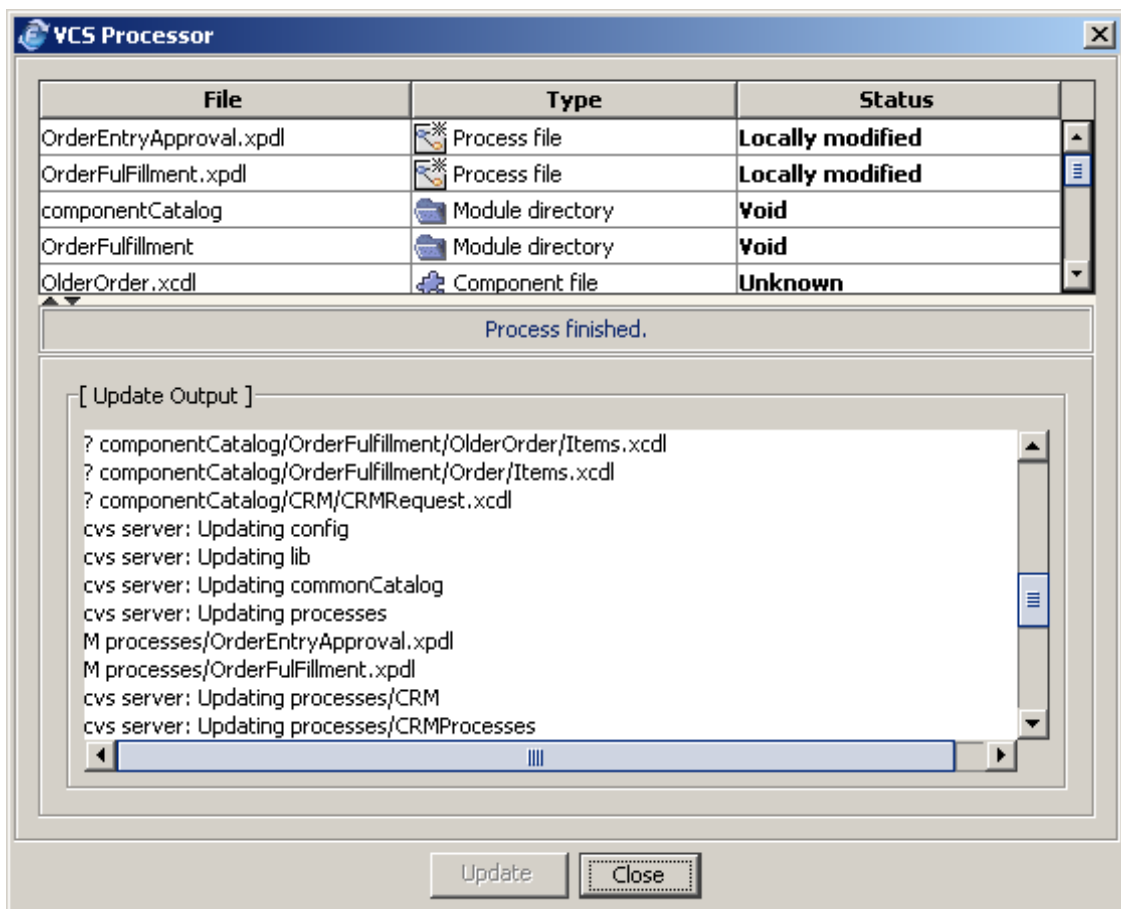
Update

Updates the local copy of a file with the last version in the repository. After you have run checkout to create your private copy of the project from the common repository, other developers may change the central source. From time to time, when it is convenient in your development process, you can use the update command from within your working directory to reconcile your work with any

revisions applied to the repository since your last checkout or update.

To update the project or any of its elements from the repository, select the **Update** option from the **VCS** menu by right-clicking on the element you have chosen.

The VCS Processor dialog is opened listing the files included in the selection to be updated. The information shown for each file is its type and status. Click the **Update** button to complete the operation. The result of the update is shown in the **Update Output** section of the dialog. Click **Close** to quit.



Update keeps you informed of progress by printing a line for each file, preceded by one character indicating the status of the file:

- *U file*: The file was brought up to date with respect to the repository. This is done for any file that exists in the repository but not in your source and for files that you have not changed but are not the most recent versions available in the repository.
- *P file*: Like 'U', but the CVS server sends a patch instead of an entire file. This accomplishes the same as *U* but using less bandwidth. The file has been added to your private copy of the sources and will be added to the source repository when you run commit on the file. This is a reminder for you that the file needs to be committed.
- *A file*: The file has been added to your private copy of the sources and will be added to the source repository when you run commit on the file. This is a reminder for you that the file needs to be committed.
- *R file*: The file has been removed from your private copy of the sources and will be removed from the source repository when you run commit on the file. This is a reminder for you that the file needs to be committed.
- *M file*: The file is modified in your working directory. *M* can indicate one of two status for a file you are working on: either there were no modifications to the same file in the repository, so your file remains as you last saw it, or there were modifications in the repository as well as in your copy, but they were merged successfully, without conflict, in your working directory. CVS will print some messages if it merges your work and a backup copy of your working file (as it looked before you ran update) will be made. The exact name of that file is printed while update is running.
- *? file*: file is in your working directory but does not correspond to anything in the source repository and is not in the list of files for CVS to ignore.

Available for:

- Project
- Processes
- Folders
- One individual process
- Catalog
- Catalog Module
- One individual component

Commit

Use **commit** to actually check the file into the repository. The added files are not placed in the source repository until you use commit to make the change permanent.

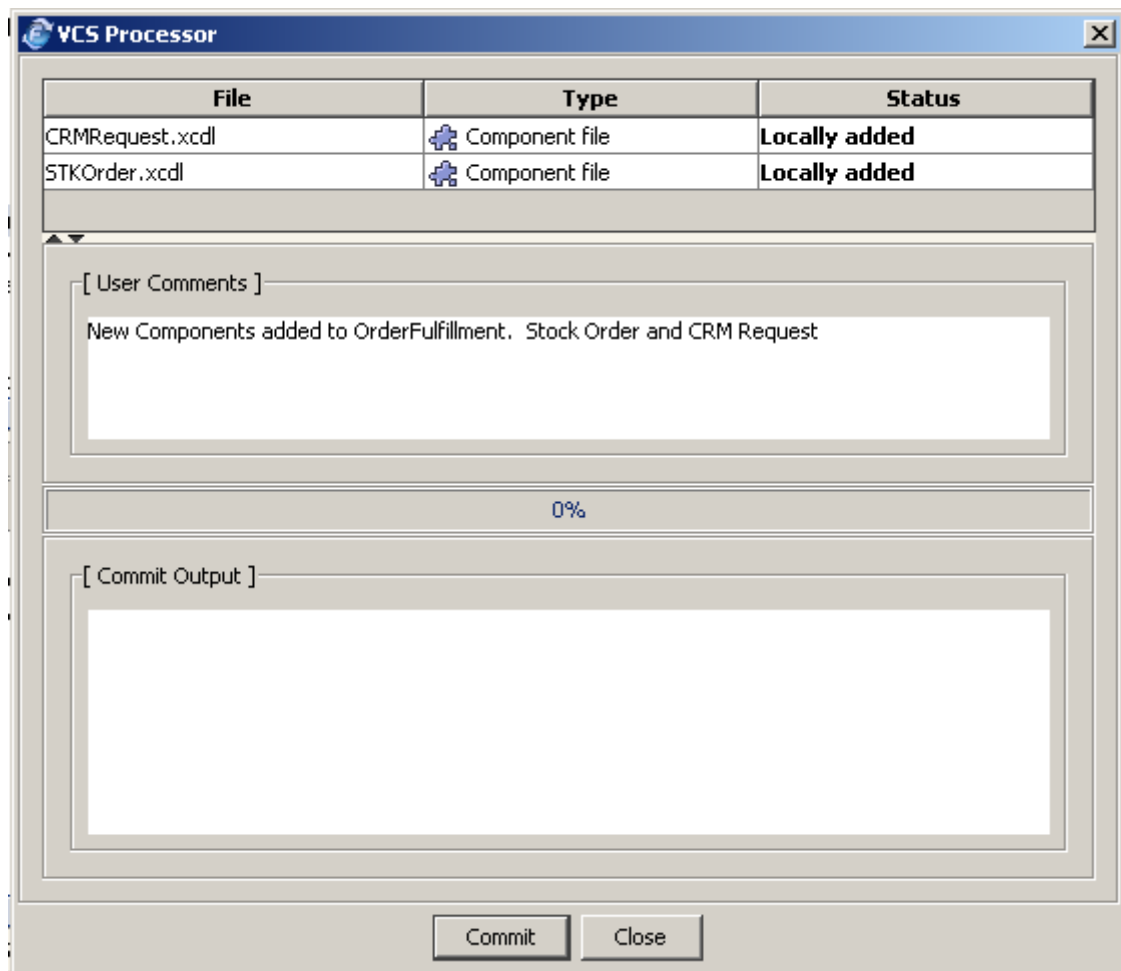
To commit an element to the repository, select the **Commit** option from the **VCS** menu by right-clicking on the element.

The commit dialog is opened. Three sections form this dialog. In the *File* section, all the elements about to commit are listed. For each one the file name, type and status is shown. Some of the file types are:

- Process
- Screenflow
- Procedure
- Configurations

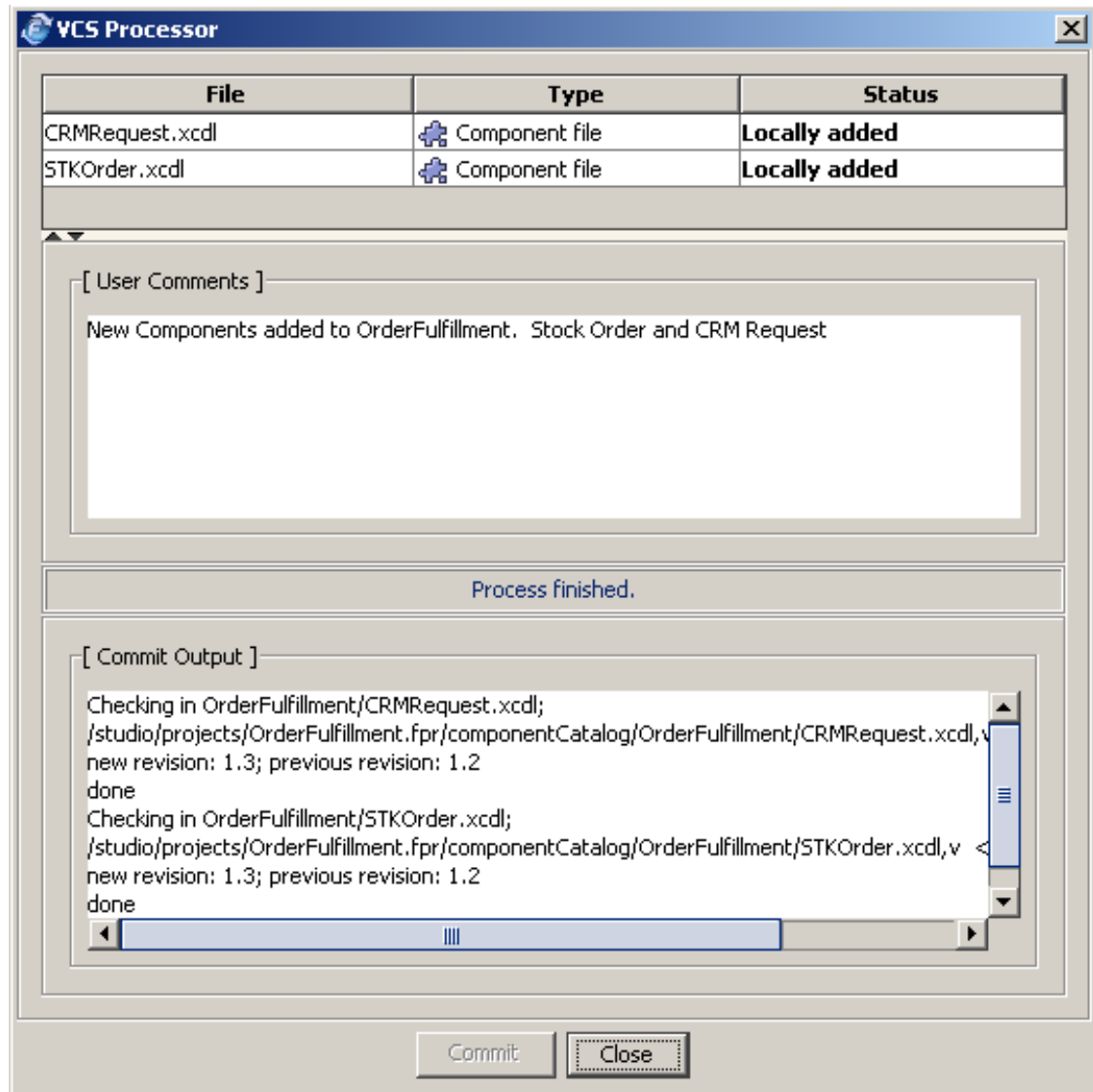
- Server configurations
- Organization settings
- Views
- Component

In the *User Comments* section, users can write the comments referring to the commit they are doing. And the *Commit Output* sections shows the result of processing the *commit*.



Those files that have been added will show the *Locally added* status. For further information about the status, please refer to the *status*

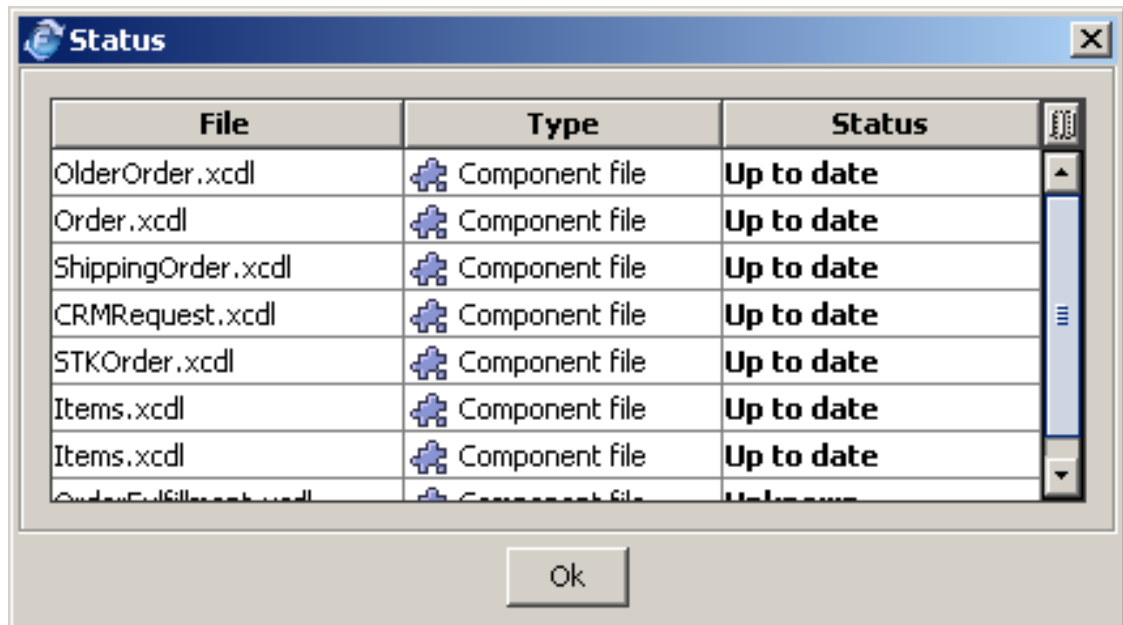
operation.



Click the **Commit** button to proceed. The advance percentage is displayed while the commit is being processed. The result of the operation is shown in the *Commit Output* portion of the dialog.

Click **Close** to end the dialog when the process has finished.

To confirm that the commit was successful, run the status operation on the elements committed. They must show the *Up to date* status.

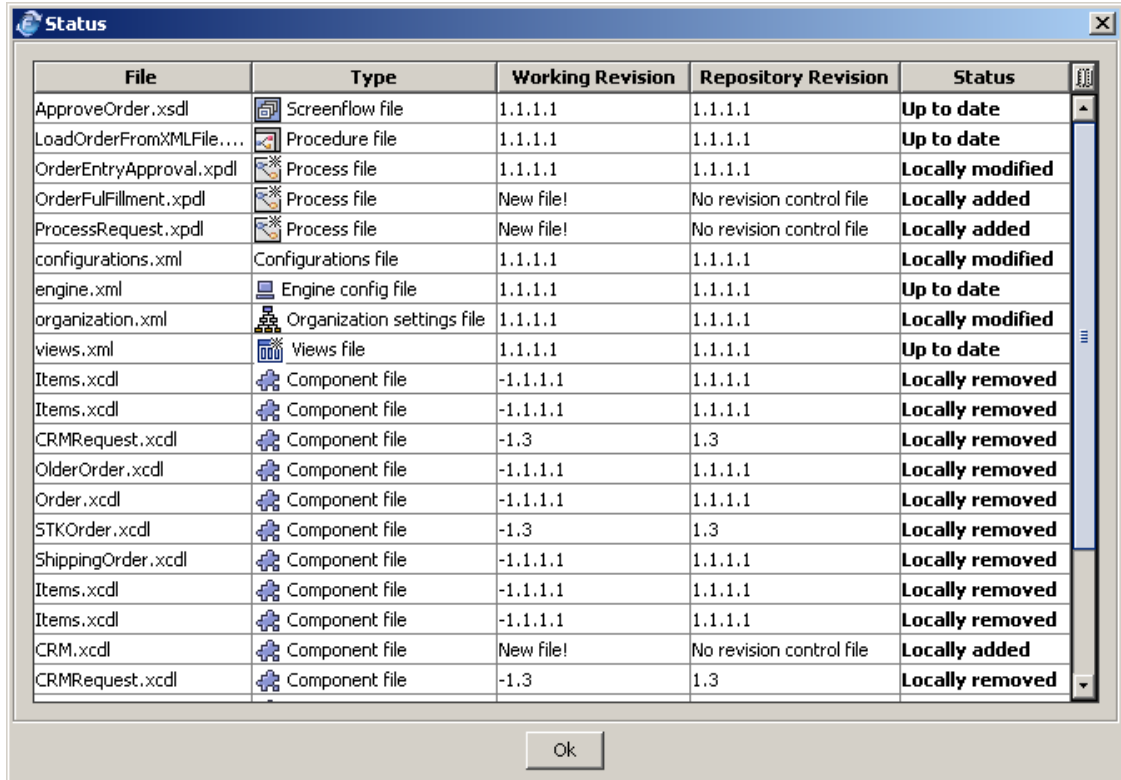
**Available for:**

- Project
- Processes
- Folders
- One individual process
- Catalog
- Catalog Module
- One individual component

Status

Based on the operations you have performed on a checked out file and the operations that others have performed to that file in the repository, a file can be classified into a number of status.

To see the status of your project elements in the repository, select the **Status** option from the **VCS** menu by right-clicking on the element label.



File	Type	Working Revision	Repository Revision	Status
ApproveOrder.xsdl	Screenflow file	1.1.1.1	1.1.1.1	Up to date
LoadOrderFromXMLFile....	Procedure file	1.1.1.1	1.1.1.1	Up to date
OrderEntryApproval.xpdl	Process file	1.1.1.1	1.1.1.1	Locally modified
OrderFulfillment.xpdl	Process file	New file!	No revision control file	Locally added
ProcessRequest.xpdl	Process file	New file!	No revision control file	Locally added
configurations.xml	Configurations file	1.1.1.1	1.1.1.1	Locally modified
engine.xml	Engine config file	1.1.1.1	1.1.1.1	Up to date
organization.xml	Organization settings file	1.1.1.1	1.1.1.1	Locally modified
views.xml	Views file	1.1.1.1	1.1.1.1	Up to date
Items.xcdl	Component file	-1.1.1.1	1.1.1.1	Locally removed
Items.xcdl	Component file	-1.1.1.1	1.1.1.1	Locally removed
CRMRequest.xcdl	Component file	-1.3	1.3	Locally removed
OlderOrder.xcdl	Component file	-1.1.1.1	1.1.1.1	Locally removed
Order.xcdl	Component file	-1.1.1.1	1.1.1.1	Locally removed
STKOrder.xcdl	Component file	-1.3	1.3	Locally removed
ShippingOrder.xcdl	Component file	-1.1.1.1	1.1.1.1	Locally removed
Items.xcdl	Component file	-1.1.1.1	1.1.1.1	Locally removed
Items.xcdl	Component file	-1.1.1.1	1.1.1.1	Locally removed
CRM.xcdl	Component file	New file!	No revision control file	Locally added
CRMRequest.xcdl	Component file	-1.3	1.3	Locally removed

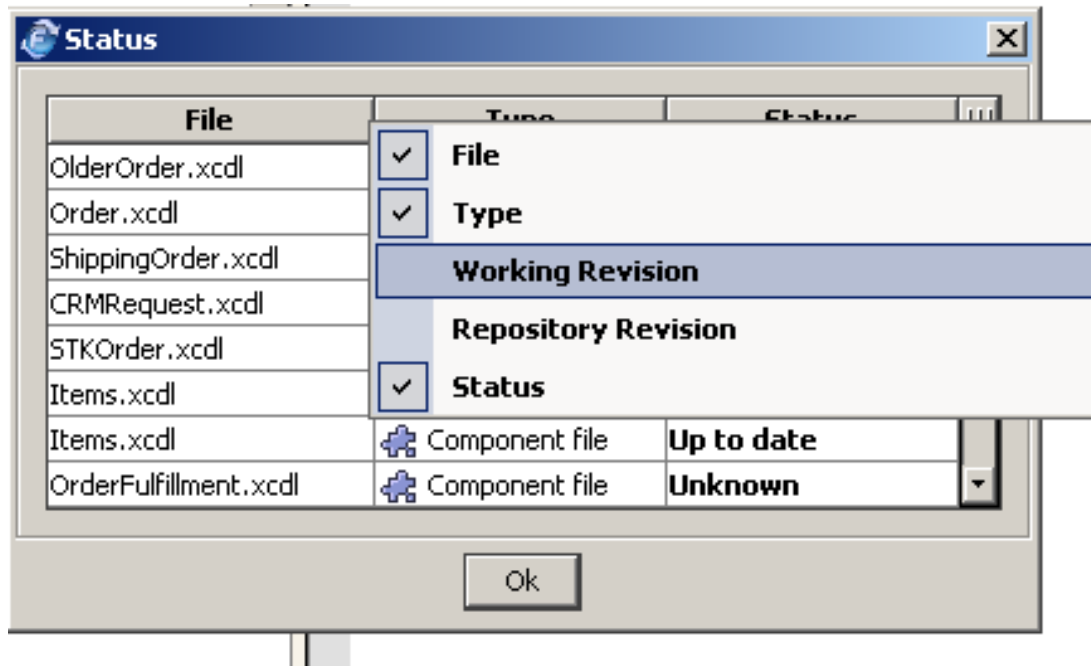
The possible status reported by the status command for a file are:

- **Up to date:** The file is identical to the latest version in the repository.
- **Needs Patch:** There is a new version of the object in the repository and the local copy is older but not modified.
- **Needs Merge:** There is a new version of the object in the repository and the local copy needs to be updated.
- **Needs check out:** The file is out of sync with the repository and needs to be updated.
- **Locally Modified:** The local version has been modified but your

changes have not been committed yet to synchronize with the repository.

- **Locally added:** The file has been added to the repository but not committed yet.
- **Locally removed:** The file has been removed from the repository but not committed yet.
- **Had Conflicts on Merge:** When the local object is locally modified and there is a newer version in the repository, by doing an update the local object can remain merge-errors.
- **Unknown:** CVS does not know anything about this file. The file has not been administrated yet.
- **No in Repository:** Although it seems to have previously been in the repository, now the object is not longer present. This happens when the object has been externally removed from the CVS.
- **Void:** The element is a directory, a folder in the Project tree.
- **Error:** An error has occurred when processing the project element.

The columns contained in the dialog can be modified by clicking the columns icon on the top right corner of the dialog box. The list of the possible columns to be included is listed. Simply select or deselect the columns you want to include.

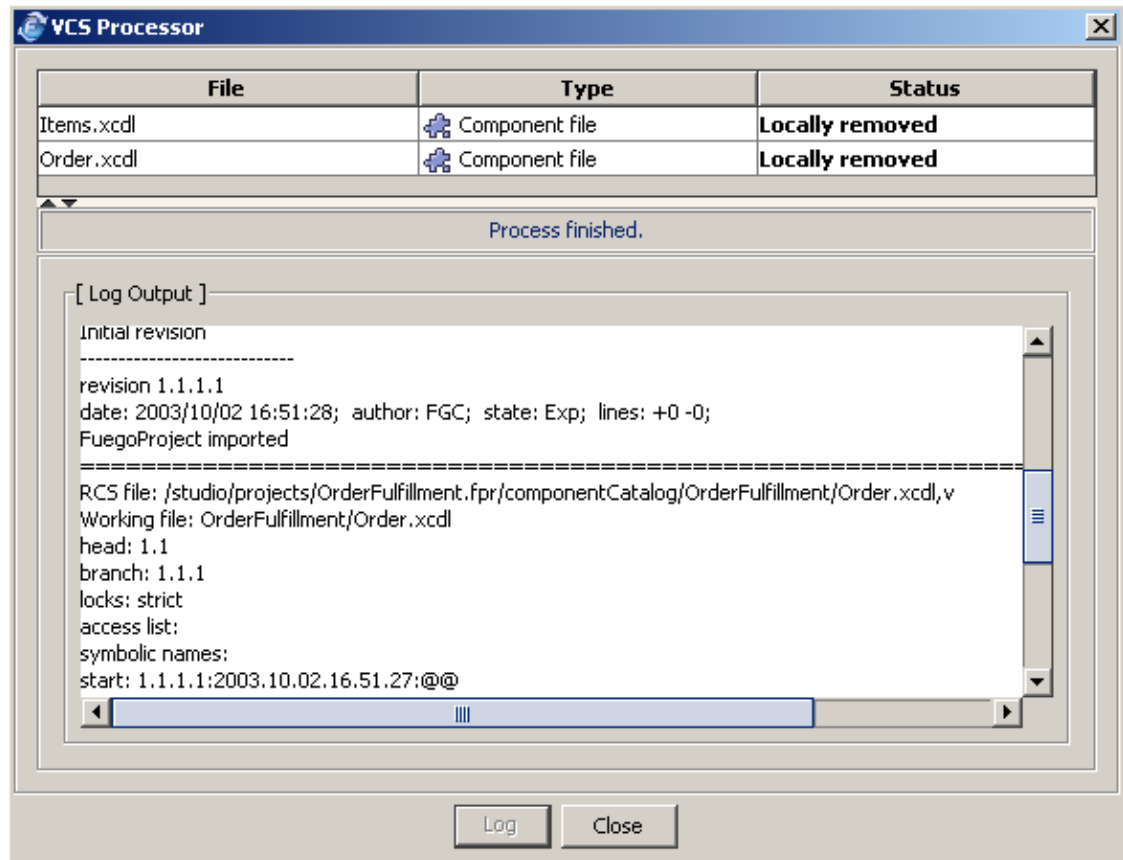
**Available for:**

- Project
- Processes
- Folders
- One individual process
- Catalog
- Catalog Module
- One individual component

Log

To view/visualize the log of operations that have been done on an element or group of elements, select the **Log** option from the **VCS** menu by right-clicking on the element.

The dialog is opened showing the list of elements contained within the level where you have run the operation (Project, Catalog, Process, Module, Component, etc.). Click the **Log** button to see each element's log.



Each element log is separated by a complete line of = characters.

Available for:

- Project
- Processes
- Folders
- One individual process

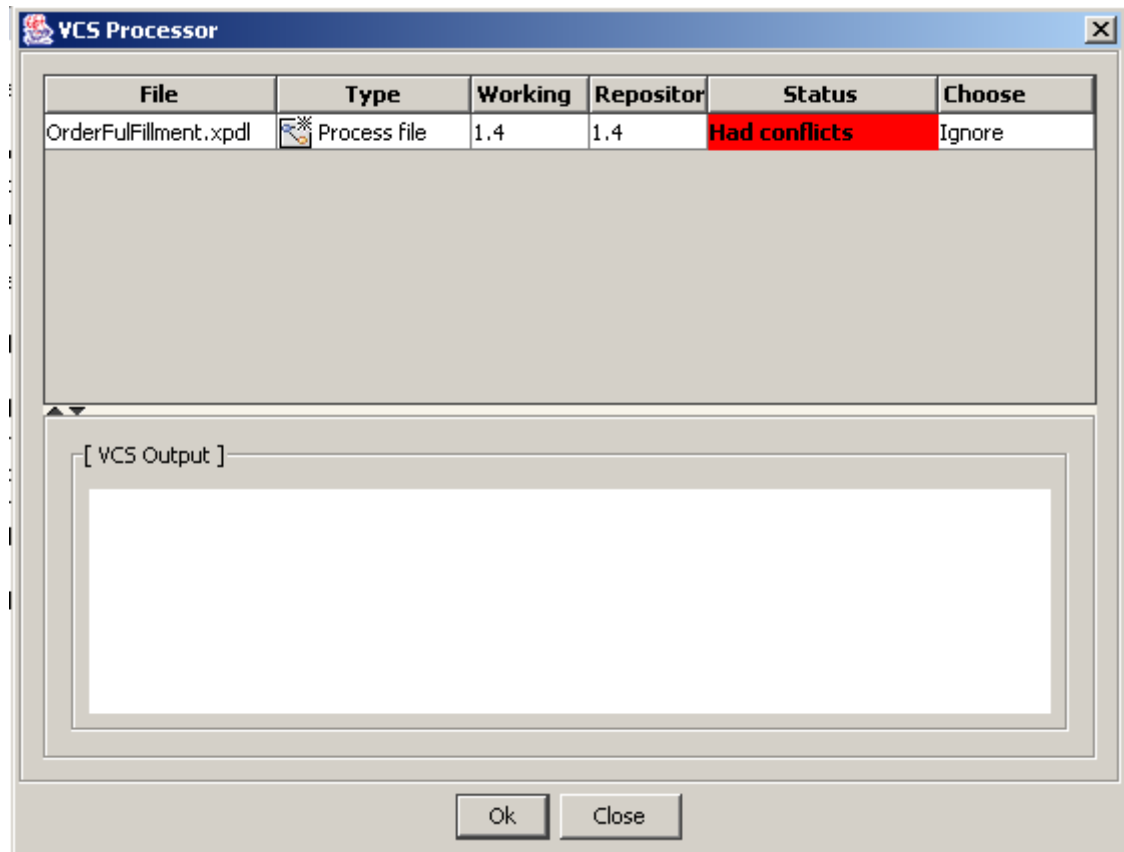
- Catalog
- Catalog Module
- One individual component

Solve Conflicts

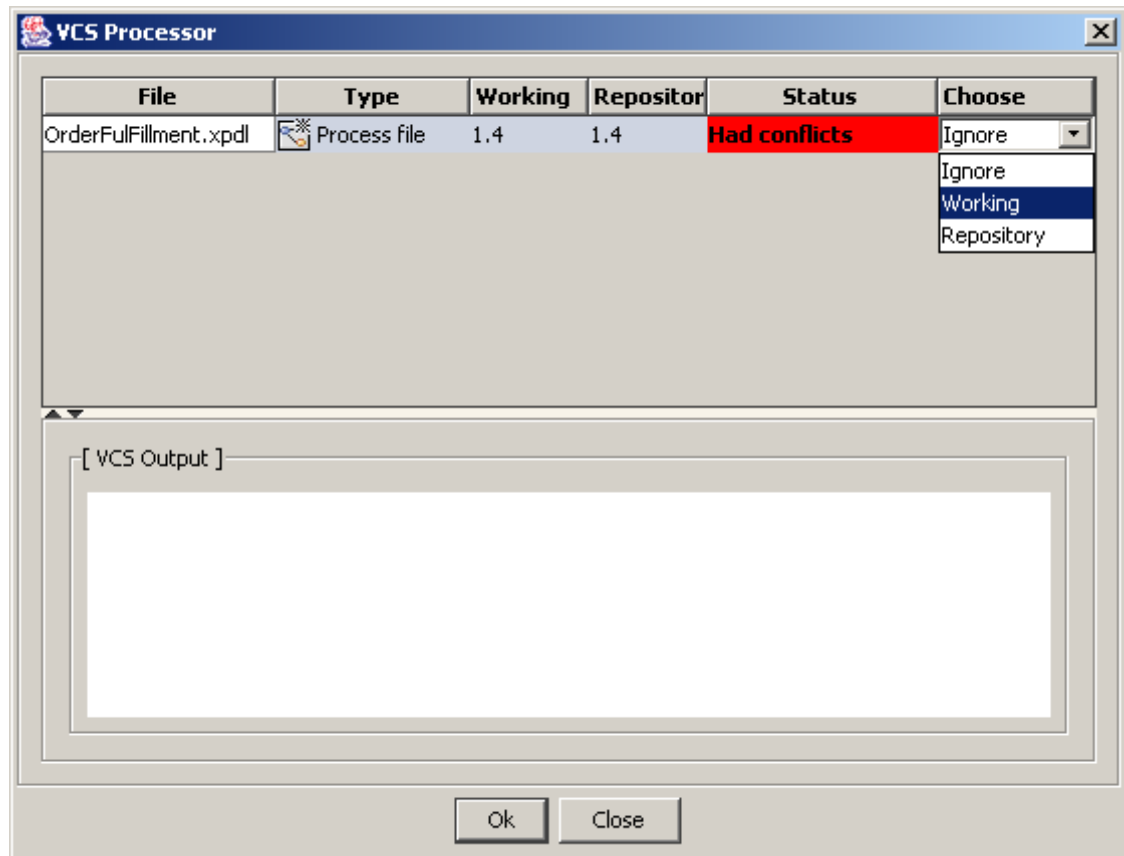
When you are making an update of a project element, some conflicts may appear. This probably happens because you have made some changes in your working directory that are not compatible with other changes made and committed by another developer.

To see the conflicts between your local project and the one in the repository, select the **Solve Conflicts** option in the **VCS** menu by right-clicking on the project name in the navigator.

The list of files or project elements with conflicts appears in the dialog box. The file name, type, working version, repository version and status are shown for each one. The status in this case will be **Had conflicts**.

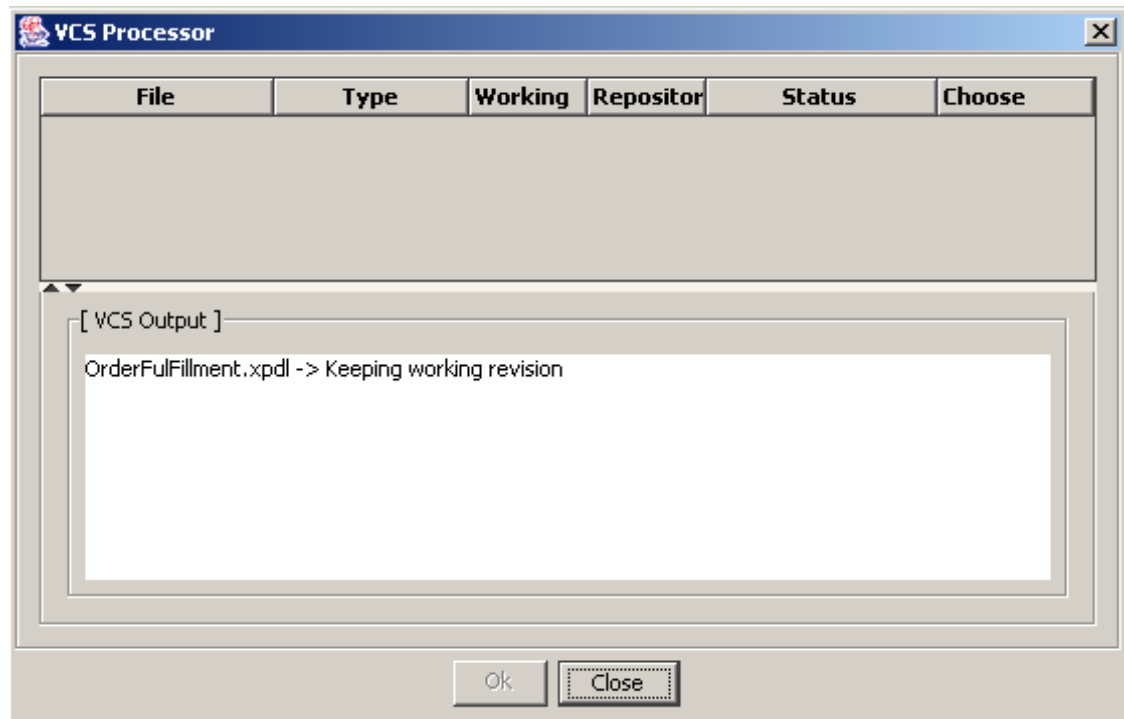


The last column of the dialog is *Choose*. Clicking on the column value, a popup with the possible actions to take is displayed.



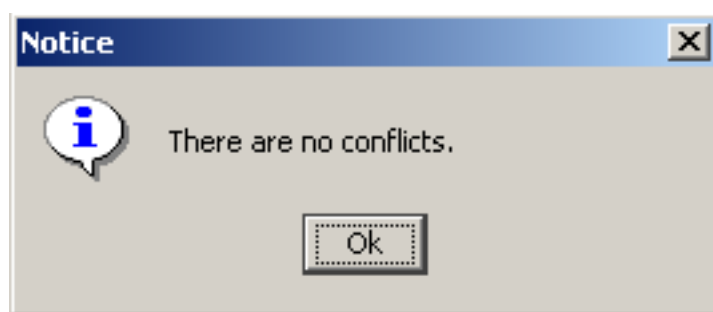
A merge is not provided, so you will be able to choose only among:

- *Ignore*, you can choose to ignore the conflict by now
- *Working*, you will keep the version in your working copy
- *Repository*, you will keep the version in the repository



If the developer is an experienced user, he can edit the XML file manually and fix the conflicts.

If you are trying to *Solve conflicts*, but there are no conflicts at all, the following notice dialog is displayed.



Available for:

- Project
- Processes

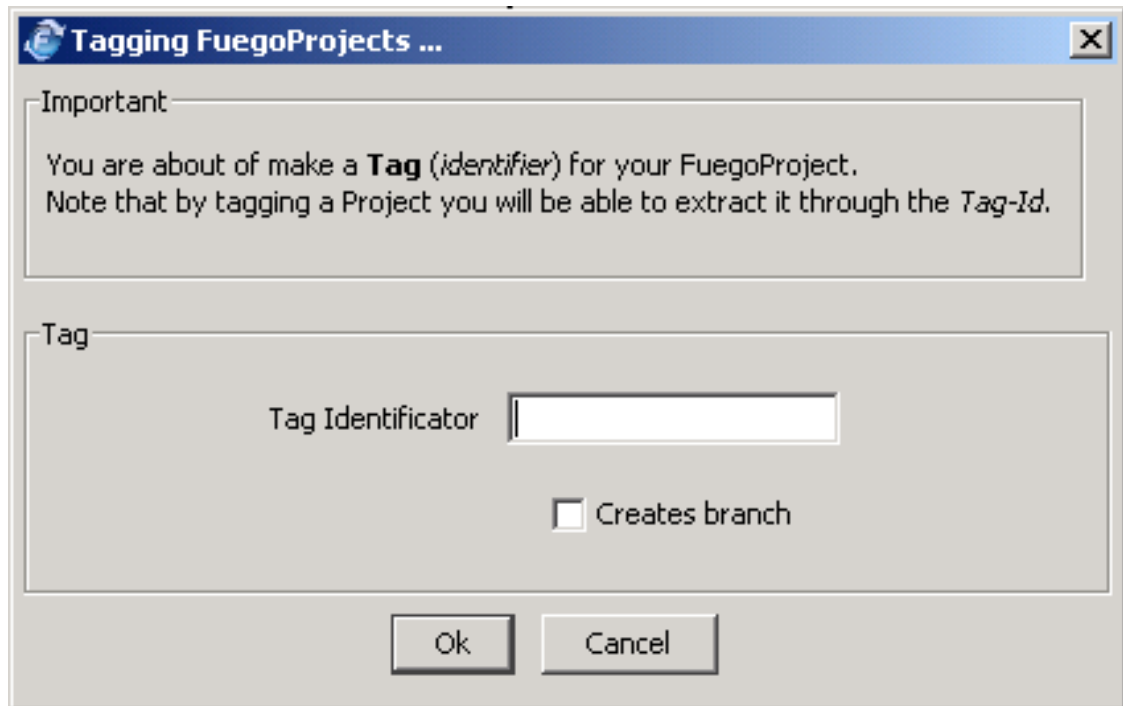
- Folders
- One individual process
- Catalog
- Catalog Module
- One individual component

Tag and Branches

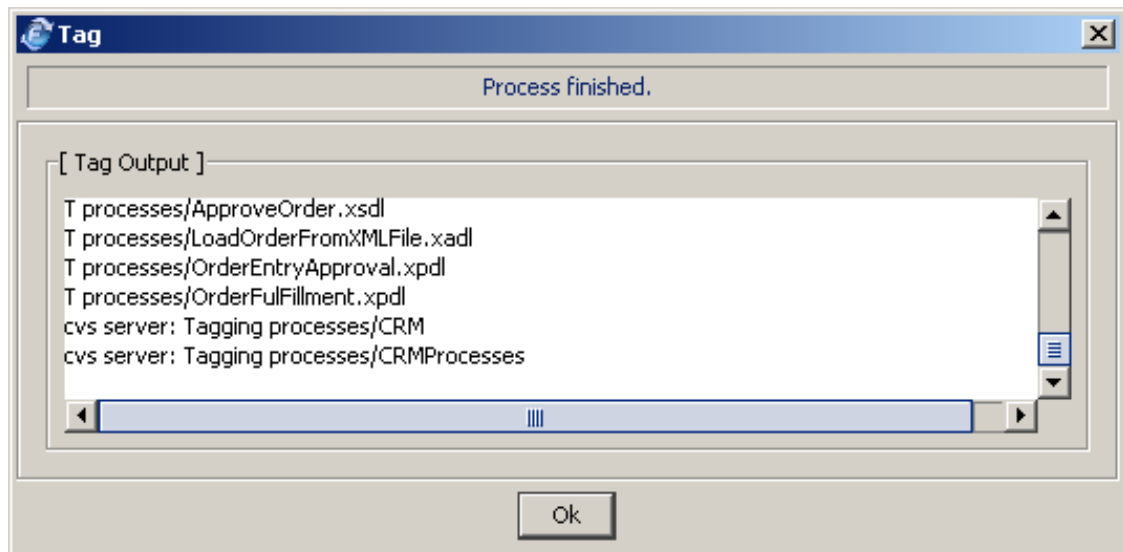
This operation is only available at the project level and it will create a tag (label name) marking all the components and processes of the project checked in the repository (last version) with a label for later retrieval.

To create a tag for the project, select the **Tag** option from the **VCS** menu by right-clicking on the Project entry of the navigator.

The Tag dialog is opened. Give the name to the tag you are about to create, typing it in the *Tag Identifier* field and clicking **OK** to continue. If you want this tag to be a branch, select the *Creates branch* check box.




The result of the tag operation is displayed in the next dialog, showing the result of the process in the *Tag Output* section.



Click **OK** to close the dialog and end the tag process.

From the moment a tag is created for the project, you can retrieve it by indicating the tag name.

Note

 If the tag is created as a branch, this creation is only done on the repository. To begin working on that branch an **Import From Repository** with that branch **must be performed**, the working copy of the project is not modified.

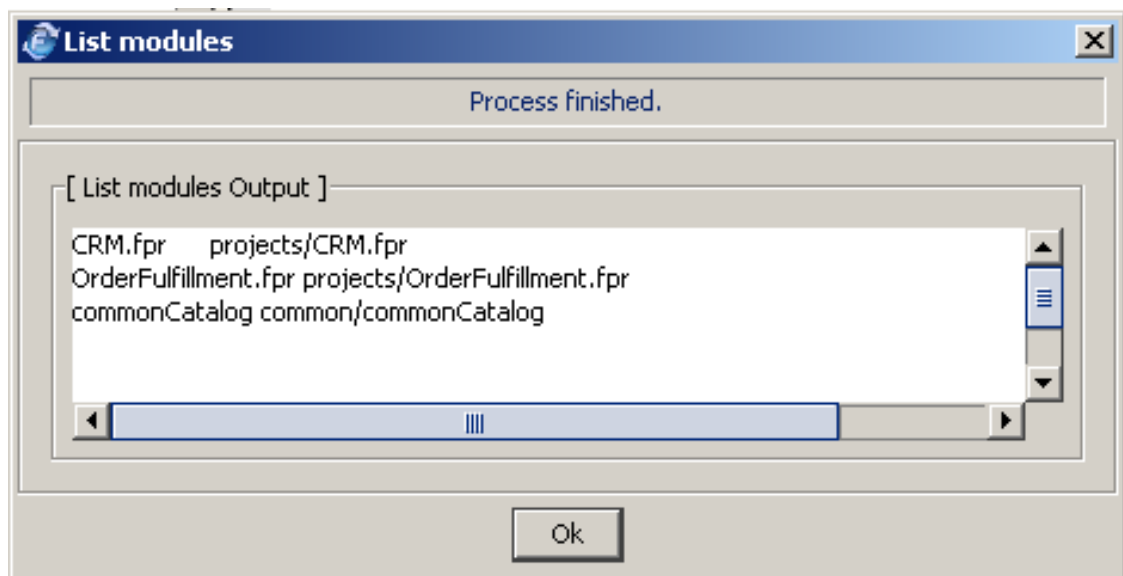
The **Project Tag** menu action on a project gets the sticky tag for the project, if it has one.

Available for:

- Project

Modules

This operation shows you all the existing modules in the same repository as the project.



Available for:

- Project

Project external resources and VCS

If the project is synchronized with a Repository VCS, its configurations are automatically added and removed from the VCS.

Specific *commit*, *update*, and other operations cannot be done on an external resource. They have to be performed at Project level. So, if you have catalogued a new external resource, the add to the VCS is done automatically, but the commit has to be done from the project. If you perform an *update*, it will show you that the external resource has been added but not committed yet.

Configurations

Configurations are administrated in the *configurations.xml* file that you will see as one of the files synchronized with the VCS repository.

Import a Project from the Repository

If you need to create a new project by checking out a project that is already administered by the VCS, select *From VCS Repository* option on the **File** and then **Import** menus.

A wizard opens,

- In the first dialog, select the provider of VCS to use and configure the necessary properties to connect to the repository. The default VCS configuration set is shown.

VCS Manager Information
Enter the connection information according to the selected VCS manager.

Version Control Manager: CVS

Connection Type:
☒ PServer ☐ Ext

Login Settings:
Server: FGC-PIV24
User Name: FGC
User Password: *****

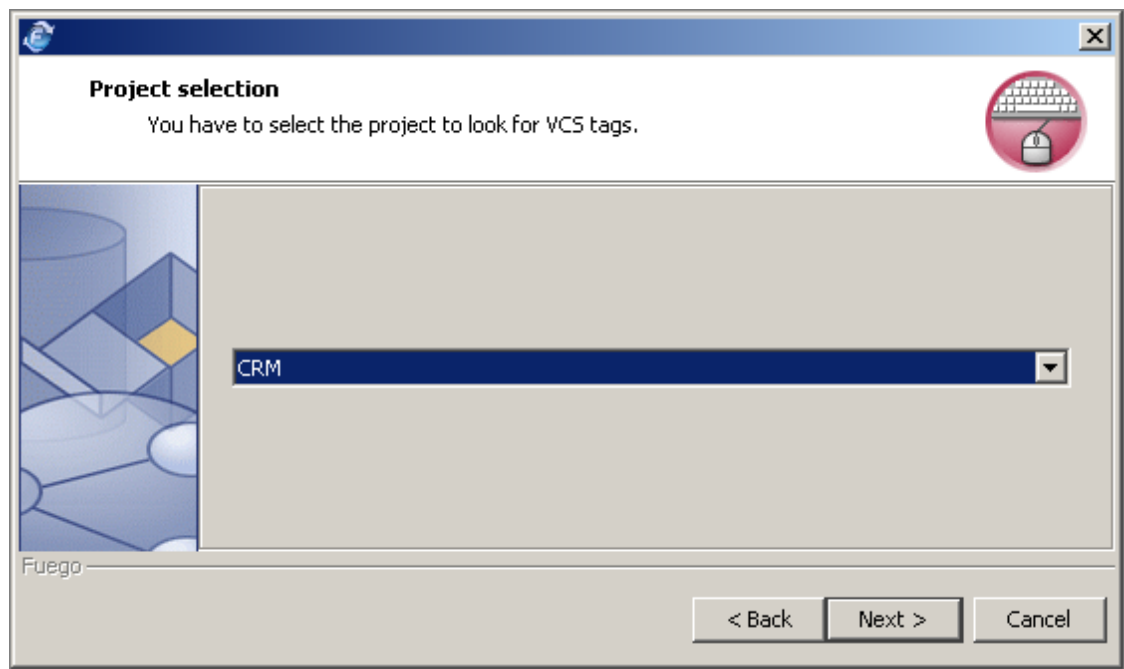
Repository Settings:
Path: /studio

CVSROOT=:pserver:FGC@FGC-PIV24:/studio

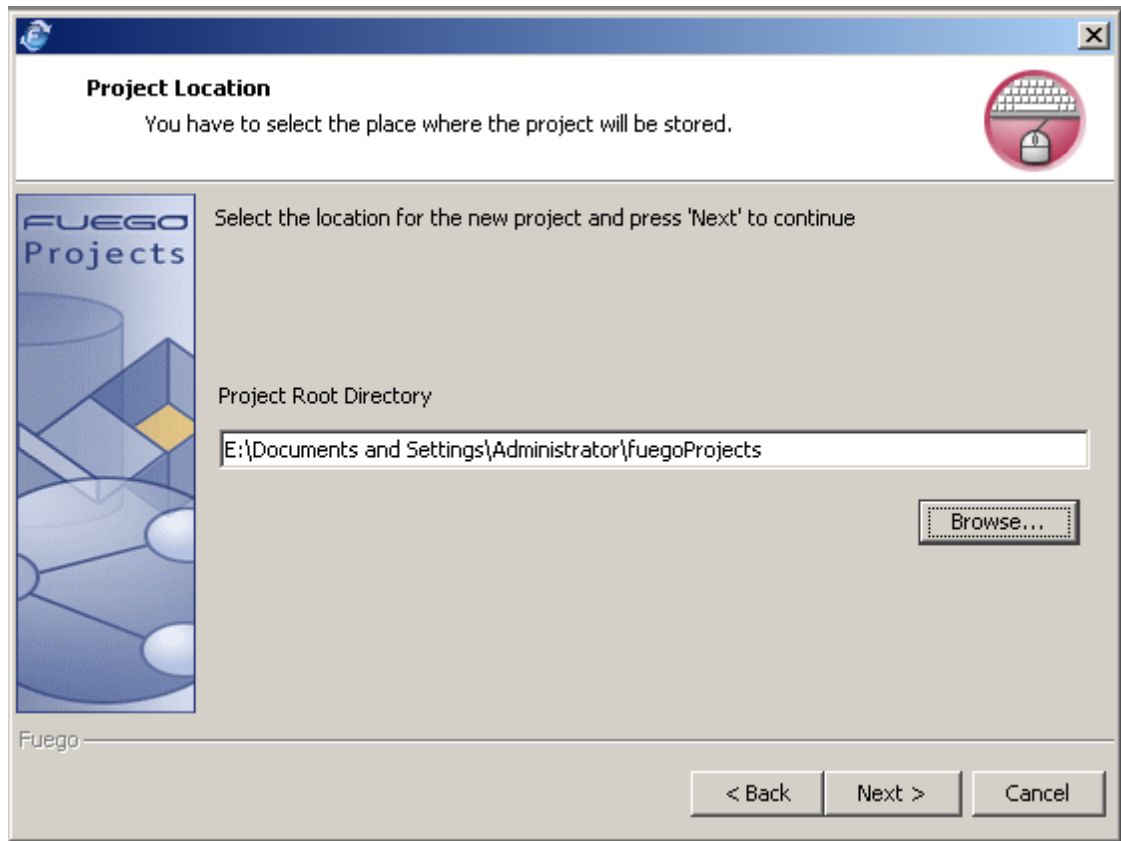
Test connection

< Back Next > Cancel

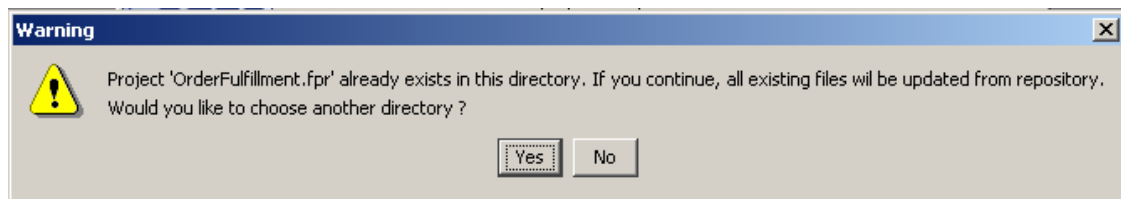
- Next, a list with all the administered projects is displayed. This list is taken from the CVS modules list. Select the project you want to import.



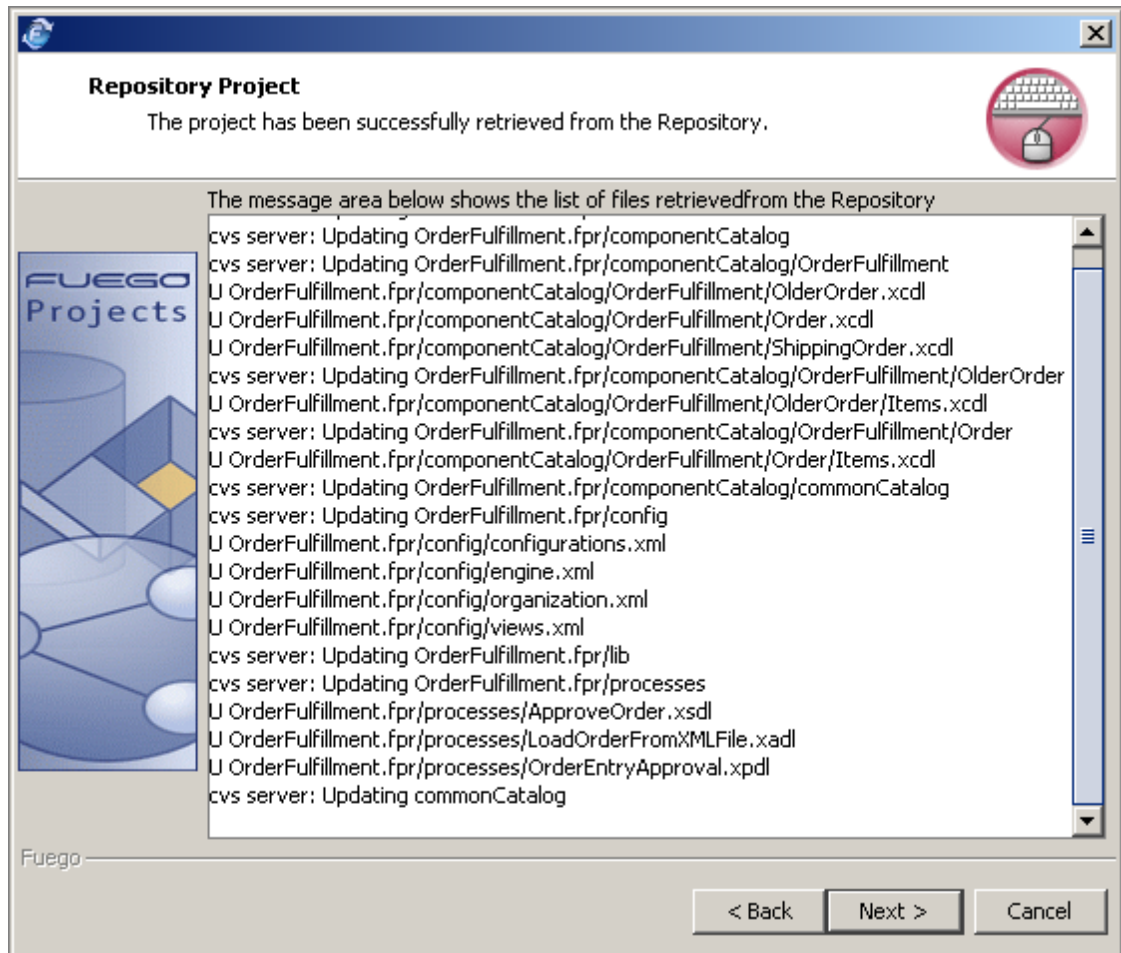
- Select a destination folder where the project will be checked out and click **Next** to proceed.



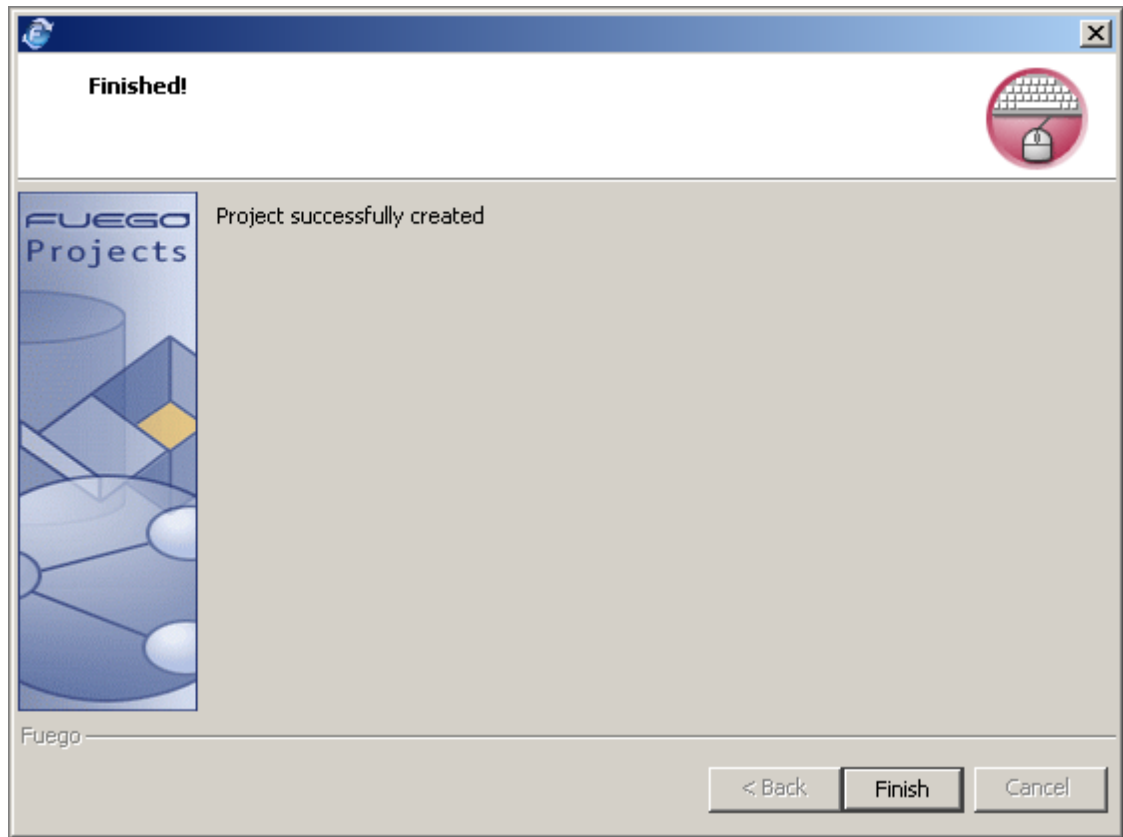
- If a project with the same name already exists in the folder you have selected, a warning is displayed. You can either change the location or continue.



- The next dialog shows the check out and project creation. Click **Next** when finished.



- The Finished! final dialog step is displayed. Click **Finish** to end the Wizard.




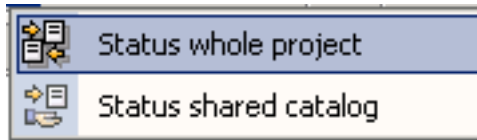
Note


If the project is already checked in but was not done by the FuegoBPM Studio tool, you must create a CVS module manually with the name of the project and the relative directory (from CVSROOT) to the corresponding project root directory.

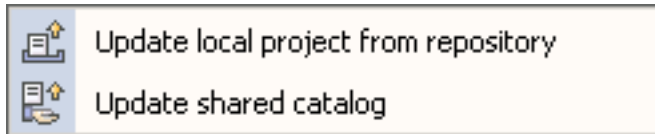
General operations


There are three special buttons in the FuegoBPM Studio toolbar to perform massive operations.

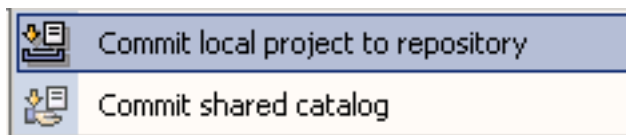
- **Project status:** 
 - Clicking the down arrow beside this icon displays a menu with status options for project or the shared catalogs.



- **Update local project from repository:** 
 - Clicking the down arrow beside this icon displays a menu with update options for the project and for the shared catalog.



- **Commit local project to repository:** 
 - Clicking the down arrow beside this icon displays a menu with options to commit the project or the shared catalog.



Common Catalog

FuegoBPM Studio administers a common catalog in order to share useful components with other users in other projects. Basically, there are two ways of sharing a common catalog:

- By using the VCS functionality
- By having a shared file system

The common catalog is under the FuegoProject directory. The directory name is *commonCatalog*.

The common catalog is created in the repository the first time a project is imported to it. While importing, you will create a relative directory, and after it is created, all consecutive projects will use this *commonCatalog*.

Note



Note for CVS Implementation: A CVS module is created with the *commonCatalog* name.

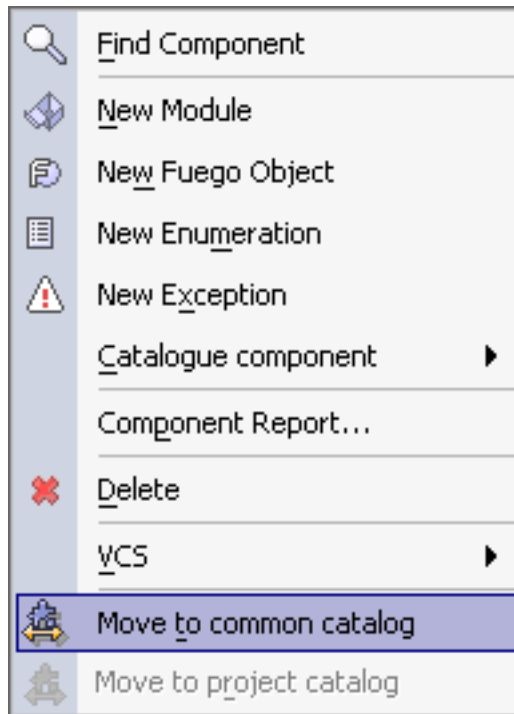
In FuegoBPM Studio, the common catalog's components are displayed with a different color (blue) to help the user distinguish them.

Operations

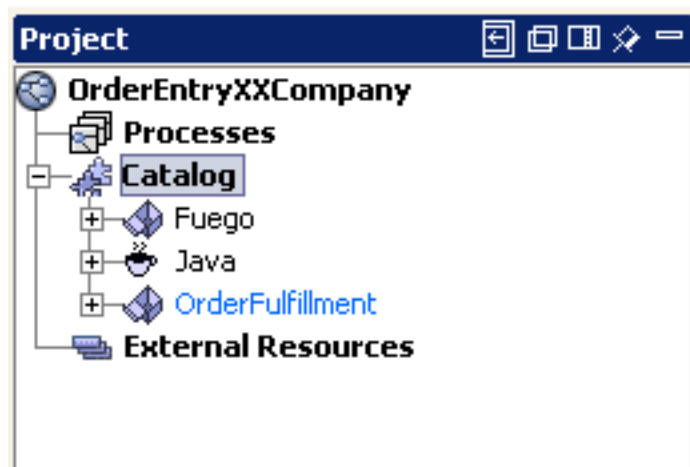
The objects the user can move to the common catalog and vice versa are the containers placed at the top level of the catalog. In other words, these objects are the modules located immediately under the catalog root node.

Move to common catalog

If the object is in the Project Catalog, it can be moved to the common catalog by right-clicking and selecting the option **Move to shared catalog**.

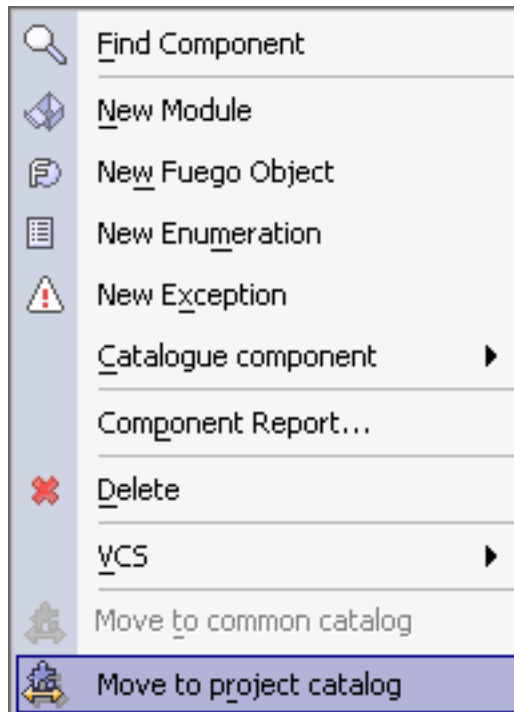


When the object is in the common catalog, it is displayed in blue.



Move to Project Catalog

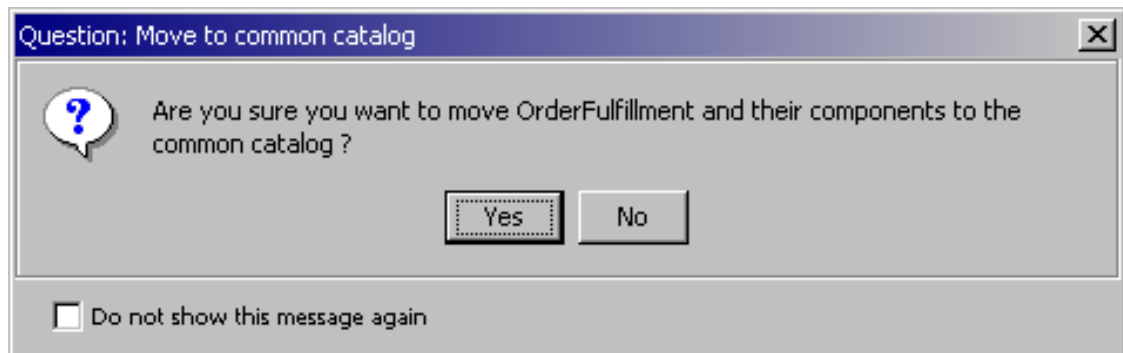
If the object is in the Common catalog, it can be moved to the Project Catalog by right-clicking and selecting the option **Move to project catalog**.



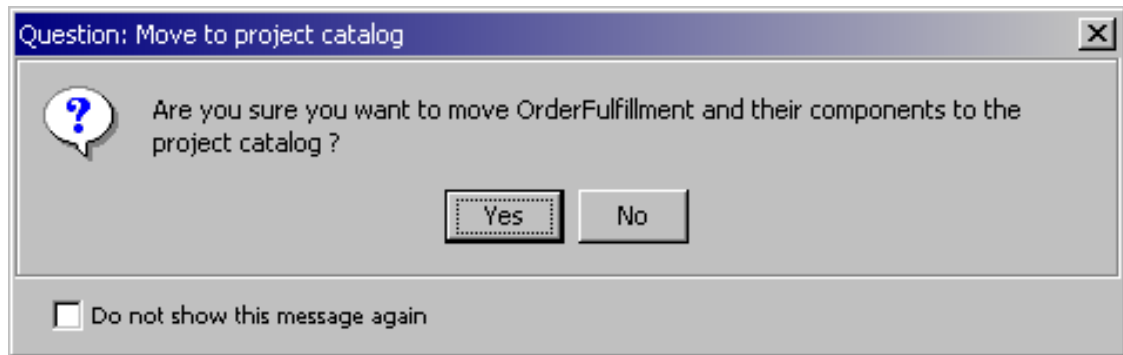
General operations

If the object that the user is trying to move is already synchronized with VCS, the user will be warned about this situation and will have the possibility to perform accordingly.

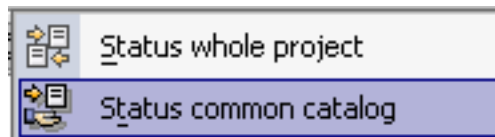
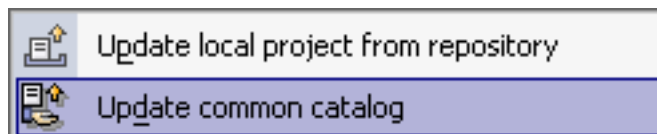
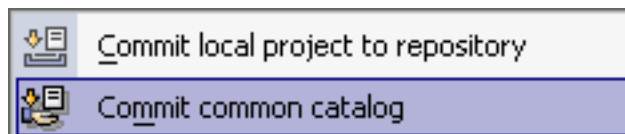
The confirm dialog varies depending on the action you are performing:



or



If the common catalog is implemented using VCS, the typical VCS operations like *status*, *update*, or *commit* can be performed on the components. In order to do this, choose the appropriate option from the VCS menu or click the arrow next to the action icon enabled in the main toolbar and select the option that corresponds to the common catalog.

Status:**Update:****Commit:**

How to reuse a common catalog

If you are working on a project and you need to use components already present in the common catalog, see the following steps:

1. Be sure that the catalog module you want to extract has already been committed and updated from the original project.
2. Perform an **Update common catalog** or an **Update local** on the project or catalog tree level.
3. The **Update** dialog displays. Select which components you want to extract from the common catalog by selecting the *Include* check box in the top pane of the dialog.
4. Select the **Update** button. The module/s containing the selected components appears in the project catalog tree entry.

Chapter 19. Data Store and Business Activity Monitoring

Data Store and Business Activity Monitoring

FuegoBPM Data Store and FuegoBPM Business Activity Monitoring contain information about instances processing performance and processes workload.

Using them you can obtain information about:

- Instances processing performance for processes: The time in which an instance reached the "End" activity of the process.
- Instances processing performance for activities. The time in which an instance was processed in the activity and flew to the next activity. Total processing time for the instance in the activity.
- Quantity of instances and their average time since they were created and their average time to be processed per activity.

Although both, Data Store and BAM, have the same database structure the data they contain is different.

FuegoBPM Data Store database archives the information. It is stored on daily basis and its granularity may be hourly or daily depending on how it is configured. On the other hand, data contained in FuegoBPM BAM, is not historical. It is updated several times during the day and the information has a caducity time. BAM database can be configured to be updated between short intervals of time. This makes the information contained in the BAM to be almost online.

Usually, BAM information should be access from FuegoBPM Dashboard and Data Store information should be access using OLAP. FuegoBPM Data Store and FuegoBPM BAM can be used both,

separately or together, it is something that would be implemented depending on the business needs.

FuegoBPM BAM Benefits

- **Visibility of Real-Time and Recent Trends** - FuegoBPM's Business Activity Monitoring (BAM) gives line of business managers and IT system administrators visibility and analysis of strategic process information.
- **Key Performance Indicators**, immediate, actionable business insight that previously might have taken days, weeks or months to compile and analyze.
- **Control and manage ongoing business operations.** Issues impacting the business are quickly identified and dealt with promptly and efficiently.
- **Respond quickly** to change based on business events as they occur and head off the bad things before they occur.
- **Capture** the big picture.
- **Zoom** in on cross-process metrics with real-time analysis to determine which processes are creating bottlenecks or which customer is most profitable.
- **Drill down** on a single object to understand complete status of a particular item, such as an order or claim.
- **Measure and Monitor** service-level agreements.
- **Business Agility** – change faster than the business changes. BAM improves decision making by providing easy to understand information graphical views that monitor business rule events and anomalies. BAM can proactively identify situations that will likely require action to be taken - before the action is required.
- **Executive Dashboard** - One of the benefits of FuegoBPM is the

potential to measure and monitor each activity of a process. In many cases, valuable strategic measurements can be derived directly from data collected during the execution of an activity, accumulating direct costs and the number of times it executes. The information in this "Executive Dashboard" provides a real-time view of the performance of the business, becoming the eyes and ears for those responsible for business processes.

Business variables

Instances Variables can be seen in the BAM structure, as well as in the Data Store, only if they are defined as *Business Variables*. They can be defined to be shown as a *dimension* or a *measure* in the BAM and Data Store.

Business variables defined as *dimension* are reflected in the:

- *TaskPerformance*,
- *ProcessPerformance*, and
- *Workload* tables of the BAM database structure.

Those defined as *measure* are only reflected in the *Workload* table of the BAM database. See BAM database structure in section below.

To learn more about *Business Variables*, please refer to the section *Using Variables-Business Variables* below **Defining a Process** in the *FuegoBPM Studio* documentation.

Measurement Marks

The Business Analyst designs in the process where to measure times or persist business variables values. These checkpoints or measurement marks are represented in the process by *Measurement Mark widgets* related to process transitions. When the server routes the

instance through a transition with a Measurement Mark, it performs all the checkpoints associated with the transition including the list of business variables which values the user wants to persist.

To learn detailed properties of Measurement Marks and how to include them into a process, refer to the section *Activities / Other Activities / Measurement Marks* in FuegoBPM Studio Documentation.

Where is the measurement stored in the BAM or Data Store?

This depends on the Measurement mark type, as follows:

- Measurement Mark type **Snapshot Start**: it is stored in the Workload.
- Measurement Mark type **Snapshot Stop**: it is stored in the Task Performance.
- Measurement Mark type **Snapshot Start & Stop**: it is stored in the Task Performance.

Where can I see measurement marks?

Measurement Marks can be seen in the audit trail from the FuegoBPM Work Portal as shown in the picture below. The elapsed time is shown only in the stop mark type. And business variables included in the measurement mark definition are shown in both the start and stop mark type. Of course, whether or not, the measurement mark are shown as start and stop separately will depend on the process design.

Audit trail					Help
Marine Supply Order Fill > End > Diving Supply OrderFill4					
Activity	Event	Responsible	Date	Copy	
Create Order	Completed		Nov 16, 2004 7:30:44 PM	0	
Review Order	Completed		Nov 16, 2004 7:30:44 PM	0	
Check Freight	Completed		Nov 16, 2004 7:31:23 PM	0	
ShippingStartMeasure	Processing		Nov 16, 2004 7:50:35 PM	0	
Measurement Start	Measurement Start	John Smith	Nov 16, 2004 7:50:35 PM	0	
	'orderAmount' = 630.00		Nov 16, 2004 7:50:35 PM	0	
Ship Product	Completed		Nov 16, 2004 7:50:35 PM	0	
ShippingStopMeasurement	Processing		Nov 16, 2004 7:50:52 PM	0	
Measurement Stop	Measurement Stop	John Smith	Nov 16, 2004 7:50:52 PM	0	
	'Elapsed Time' = 17s		Nov 16, 2004 7:50:52 PM	0	
	'orderAmount' = 630.00		Nov 16, 2004 7:50:52 PM	0	
End	Completed		Nov 16, 2004 7:50:53 PM	0	

Measurement Mark Started

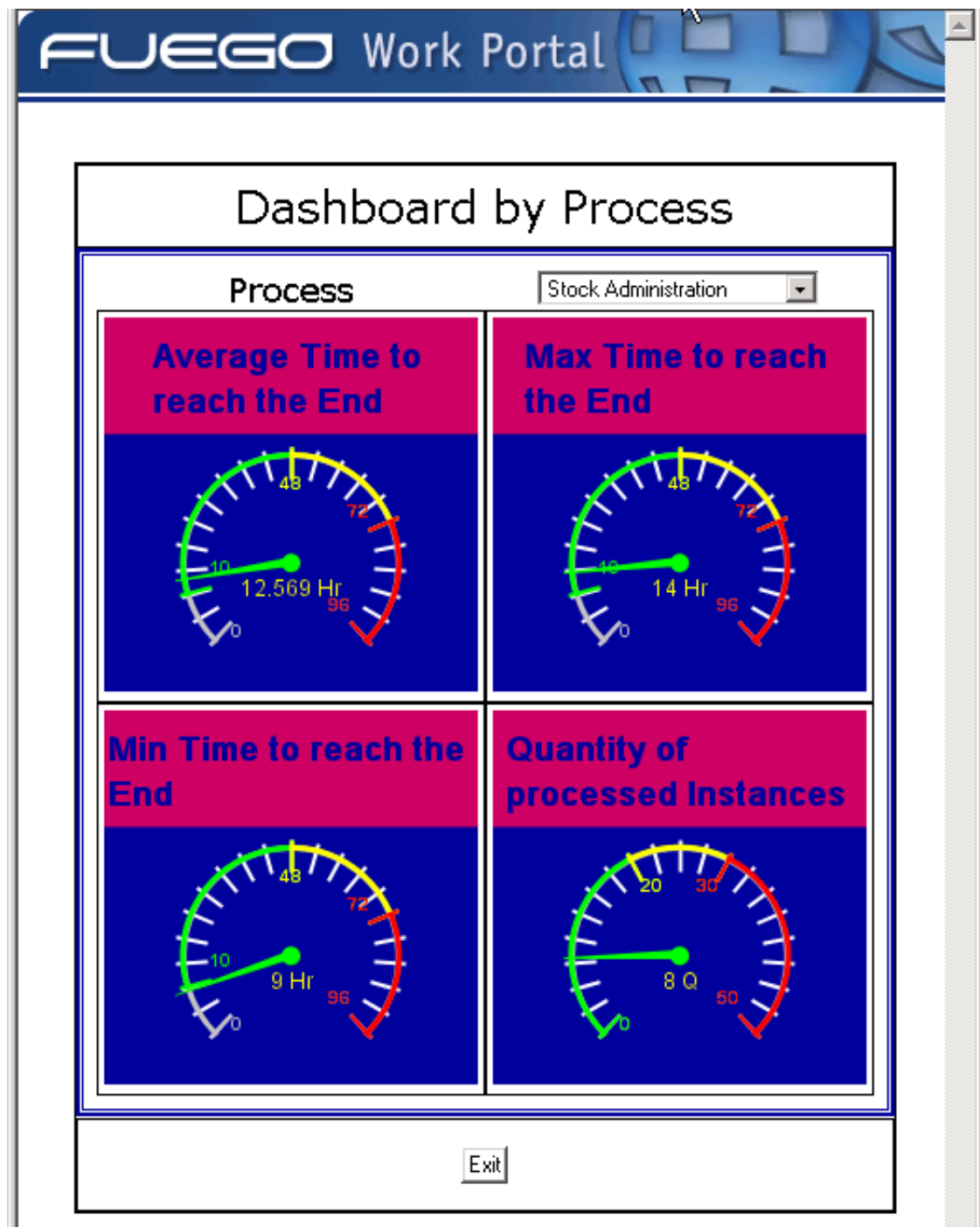
Measurement Mark Stopped

Using the BAM and FuegoBPM Dashboard

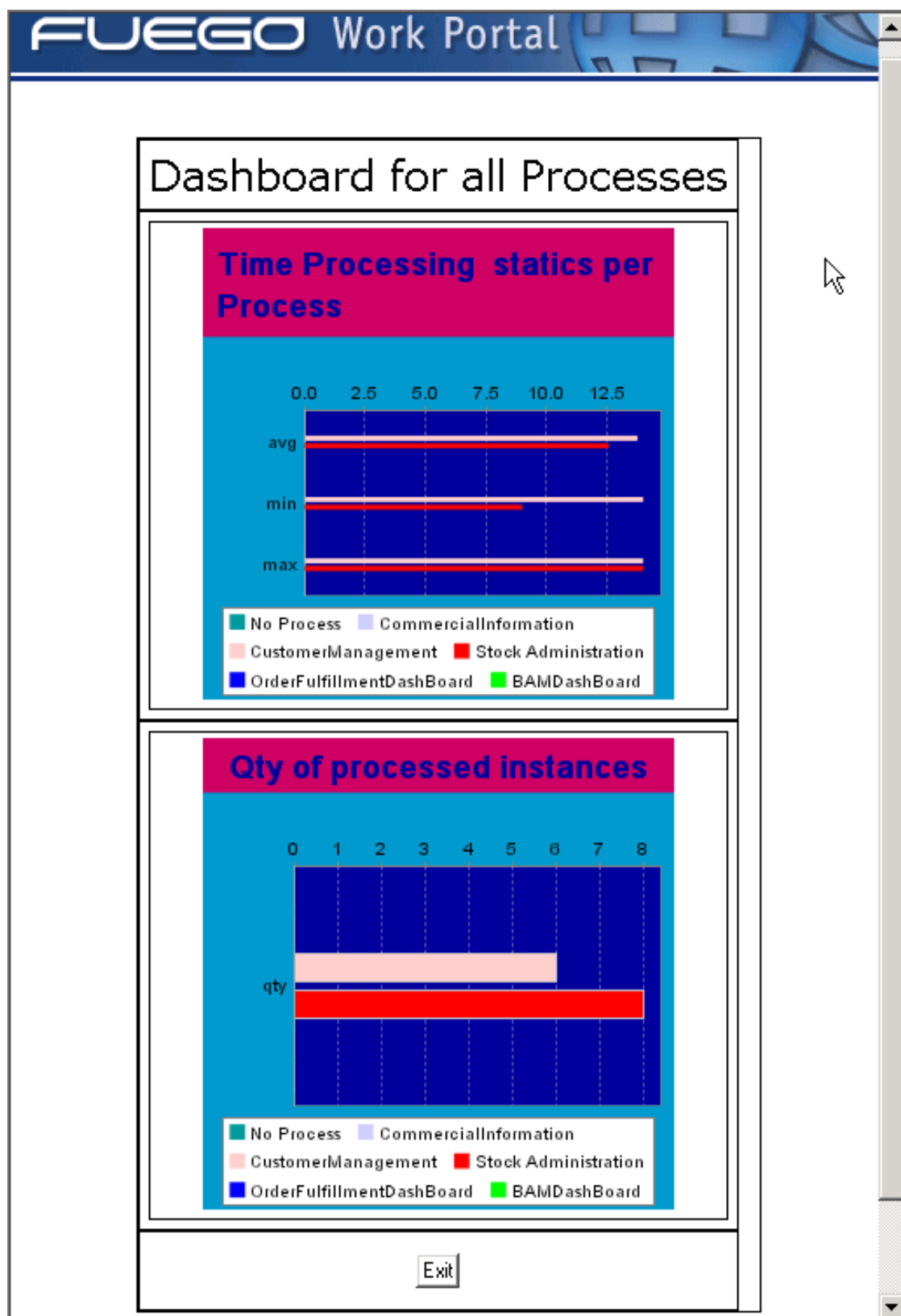
BAM reduces this flood of data and creates meaningful graphical views of real-time information. It consists of a set of graphical charts designed in the FuegoBPM Studio that take measurements of Key Performance Indicators (KPI) during the execution of a process.

The company's processes are controlled by a FuegoBPM Server that automates the integration of the company's applications, databases and people. Each step through the company's processes is automatically measured and monitored. This measurements can be graphically shown using FuegoBPM Dashboards. Some examples are shown below:

Example I: showing KPI per process

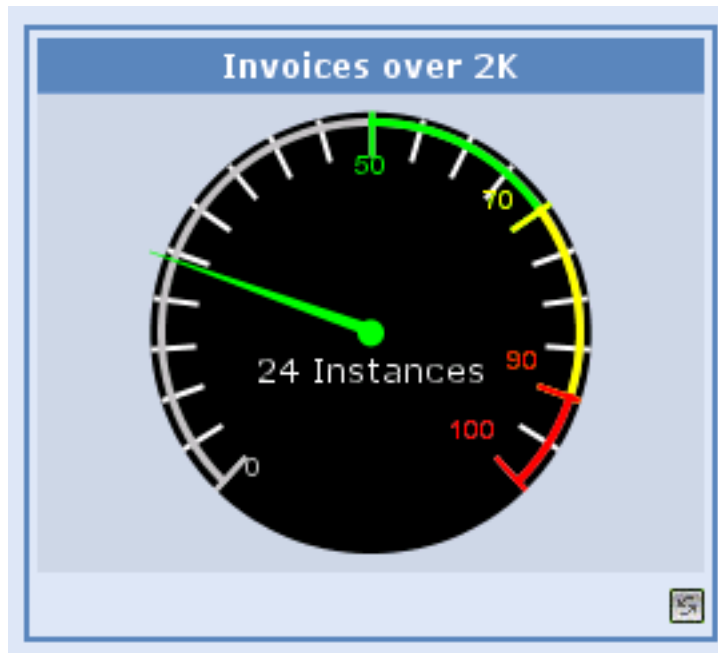


Example II: showing KPI for all processes



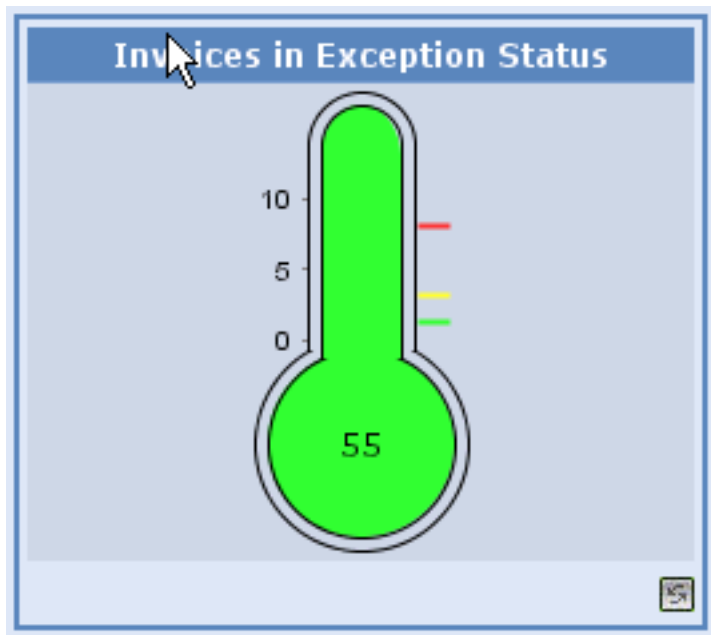
Example III: invoices over 2K

This example was built using instances variables



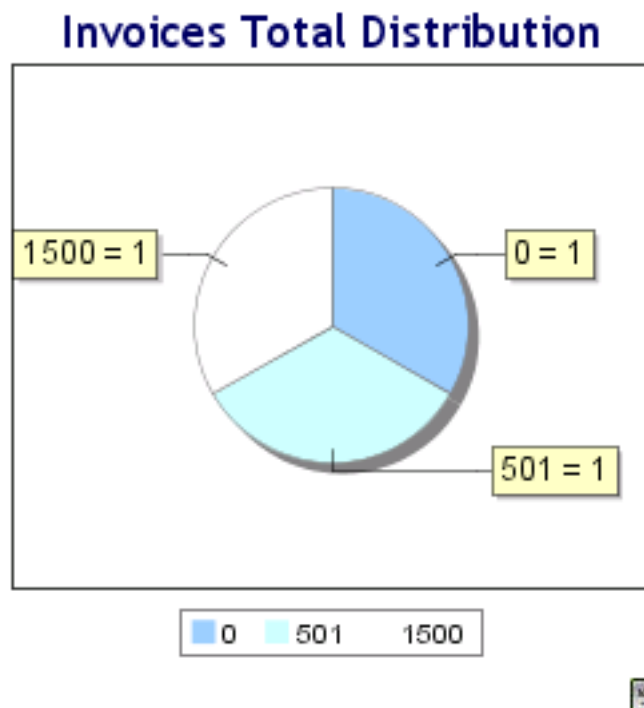
Example IV: instances in exception status

Shows the quantity of instances in the process in exception status.



Example V: Invoices total distribution

Example built on a business instance variable containing the invoice amount



Example VI: Orders in process

Quantity of orders (instances) being processed.



Please refer to FuegoBPM Dashboard, section *Building a FuegoBPM Dashboard using FuegoBPM BAM* in the FuegoBPM Studio to find an example using the BAM database.

Databases structure

Both databases, BAM and Data Store, have the exactly same structure.

OUs

- **ouIn**, DECIMAL(10), not null
- **parentIn**, DECIMAL(10), not null
- **name**, STRING(255, not null

primary key(ouIn)

Roles

- **roleIn**, DECIMAL(10), not null
- **roleId**, STRING(255) not null

primary key(roleIn)

Participants

- **participantIn**, DECIMAL(10), not null
- **participantId**, STRING(255), not null
- **ouIn**, DECIMAL(10), not null
- **displayName**, STRING(255)

primary key(participantIn)

foreign key(ouIn, referencedTable="OUs")

Processes

- **ouIn**, DECIMAL(10), not null
- **processIn**, DECIMAL(10), not null
- **processId**, STRING(255), not null
- **label**, STRING(255), not null

primary key(processIn)

foreign key(ouIn, referencedTable="OUs")

Activities

- **activityIn**, DECIMAL(10), not null
- **activityId**, STRING(255), not null
- **processIn**, DECIMAL(10), not null
- **label**, STRING(255)

primary key(activityIn)

foreign key(processIn referencedTable="Processes")

Workload

This table contains a record showing the quantity of instances, their average time since they were created and their average time waiting in the *activityIn* to be processed. This information is shown for an activity, role, participant, and if it is the case, the activity in a subprocess where the "child" instances created by the *activityIn* are.

- **snapshotTime**, TIMESTAMP, not null
- **activityIn**, DECIMAL(10), not null
- **roleIn**, DECIMAL(10), not null
- **participantIn**, DECIMAL(10), not null
- **origActivityIn**, DECIMAL(10), not null It is the activity in a subprocess,in which the "child" instances created by the *activityIn* are, when the snapshot is done. See example below.
- **waitActivityIn**, DECIMAL(10), not null. In a subprocess case,

this activity is the *subflow* or *process creation* activity that create the instances taken into account in the current record. See example below.

- **quantity**, DECIMAL(10), not null Quantity of instances in the activity waiting to be processed at snapshot time.
- **avgTimeTask**, DECIMAL(10), not null. Average time, in seconds, of the instances waiting in the activity to be processed.
- **avgTimeProcess**, DECIMAL(10) not null. Average time, in seconds, of the instances since they were created.

foreign keys

- activityIn, referencedTable="Activities"
- waitActivityIn, referencedTable="Activities"
- origActivityIn, referencedTable="Activities"
- roleIn, referencedTable="Roles"
- participantIn, referencedTable="Participants"

An example showing the origActivityIn and waitActivityIn

Suppose there are three processes.

- **Process A** with a *Process Creation 1* activity, that initiates instances in the **Process C**.
- **Process B** with a *Process Creation 20* activity, that initiates instances in the **Process C**, and,
- **Process C** with the activity *Interactive 100*, that has 20 instances

waiting to be processed in the moment the snapshot is done. 15 of these instances have been created from *Process A - Process Creation 1* and 5 have been created from *Process B - Process Creation 20*.

The content of the *workload* table would be:

activityIn= *Interactive 100*, waitActivityIn= *Process Creation 1*, quantity 15

activityIn= *Interactive 100*, waitActivityIn= *Process Creation 20*, quantity 5

and

activityIn= *Process Creation 1*, origActivityIn= *Interactive 100*, quantity *N*

activityIn= *Process Creation 20*, origActivityIn= *Interactive 100*, quantity *M*

(where *N* and *M* are the quantity of instances that there are in processes *A* and *B* respectively).

Note

 **If there is a waitActivityIn record, there is an origActivityIn record, and vice-versa.**

There is a case in which a record in the *workload* table may have information in both fields *origActivityIn* and *waitActivityIn*. When a process is a subprocess of another and it has a subprocess as well.

For example, suppose there are 3 processes. *A*, *B* and *C*, where *B* is a subprocess of *A* and *C* is a subprocess of *B*. In this case, workload records of the activities in process *B*, might have *waitActivityIn* information of process *A*, and *origActivityIn* information of process *C*.

TaskPerformance

This table contains a record for each instance that was processed in the *activityIn*, in the *roleIn* by the *participantIn*.

- **activityIn**, DECIMAL(10), not null
- **roleIn**, DECIMAL(10), not null
- **participantIn**, DECIMAL(10), not null
- **completionDate**, DECIMAL(10), not null. Date in which the instance was processed in the activity and flew to the next activity. To maintain the coherence between the data, the *completionDate* is stored in *GMT-0*, as there might be different servers running with different hours.
- **taskTime**, DECIMAL(10), not null. Total processing time, in seconds, for the instance in the activity.

foreign keys

- activityIn, referencedTable="Activities"
- roleIn, referencedTable="Roles"
- participantIn, referencedTable="Participants"

ProcessPerformance

- **processIn**, DECIMAL(10), not null
- **completionDate**, TIMESTAMP, not null Date in which the instance reached the *End* activity of the process. To maintain the coherence between the data, the *completionDate* is stored in *GMT-0*, as there might be different servers running with different hours.
- **taskTime**, DECIMAL(10), not null. The *taskTime* field is the time, in seconds, in which the instance was processed

foreign key(processIn, referencedTable="Processes")

LastSnapShot

This is a view of the *workload* table. And contains the time in which the BAM updater was executed for the last time.

- **lastsnapshot**, TIMESTAMP, not null

Processes used to update the BAM and Data Store database

Not all the deployed processes are used to update the BAM and Data Store database. Only those processes that have been define as *Generate Events for all activities* are taken into account when the BAM or Data Store updater is run.

If you don't want to register events for all the activities, remember to set as generate events those activities you are interested in storing information about activities and tasks in the *workload* and *taskperformance* tables. If it is the case, remember to set the *Begin* and "End" activities to load information on performance. And be careful with split-joins circuits, as if you don't set both as generate events, the information loaded will not be accurate.

See **Generate Events** in the FuegoBPM Studio documentation under the **Designing a Process** chapter for more details.

BAM Configuration

FuegoBPM Studio introduces BAM capabilities in the built-in Studio Server. This will enable developers with the ability to test the real-time Dashboards consuming the BAM Metrics generated by the FuegoBPM Server.

BAM in FuegoBPM Studio has to be configured as an Server preference. Refer to BAM Properties under the Defining Server and

Runtime Properties chapter to learn about this.

Chapter 20. Defining Organizational Resources


Organization

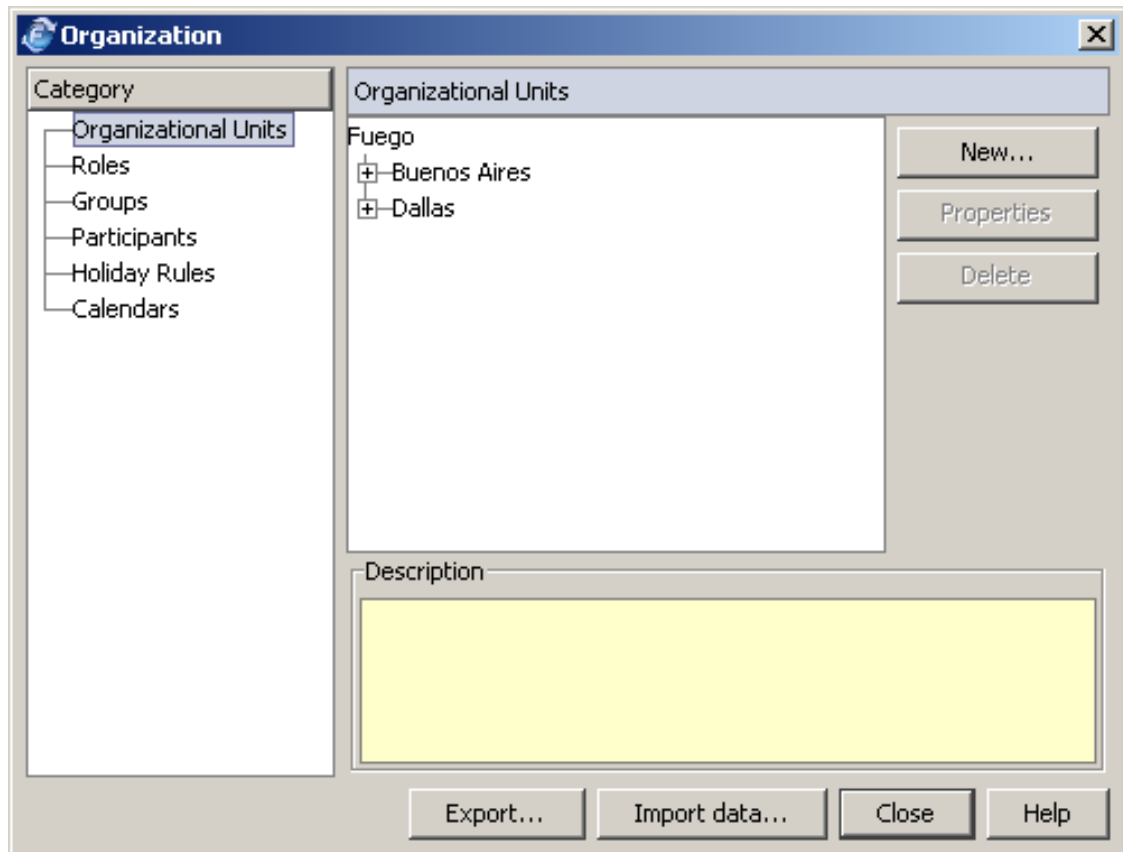
Every project created with FuegoBPM Studio has its own defined organization structure. The Organization window works as an organization administrator. It allows you to logically define and maintain your organization's structure, either for testing and simulation purposes or to define the real structure of your organization for a production environment. The following types or categories of objects are managed by the Organization window:

Category	Description
Organizational units	Represent departments or divisions within the organization. You can assign participants and assign calendar rules to an organizational unit. You can also deploy processes under an organizational unit.
Organizational Roles	Represent job functions performed within the organization. Roles are assigned to participants or groups and this association defines the permissions the participants have when executing tasks of FuegoBPM Studio processes through Work Portal.
Organizational Groups	Represent a profile. Groups have members associated to them and can be assigned roles and other groups.
Participants	The organization members that participate in any task within a business process.

Category	Description
Holiday Rules	Define the organization's non-working days. These rules inform the FuegoBPM Enterprise Server that there is an exception to the normal calendar rules on certain days of the year.
Calendar Rules	Define the organization's work week and work schedule. Calendar rules can be assigned to organizational units.

The Organization window

The **Organization** window work space is divided into two navigation panels. The left navigation panel shows all the different categories of objects you can create in the organization. The right navigation panel lists all the objects of each category that are already created and allows you to create new objects as well as to delete existing ones. Once you have opened your project in FuegoBPM Studio, click on the **File** menu and then select the **Organization** menu item. This option is only enabled when a project is currently open. You can also launch the **Organization** window from FuegoBPM Studio toolbar clicking on the **Organization** icon : the Organization window displays all its working components.



Left Panel	Right Panel
<p>Displays all the different categories of objects you can have in the organization</p>	<p>When you click on a category in the left panel, the list of objects for that category appears in the right panel. The information is displayed as different views showing a tree or a list of items in a table, depending on the category you click. The New button is always enabled to create new objects for the selected category. After you select one of the objects on the list, two buttons--Properties and Delete--are enabled for you to modify or delete the selected object.</p>

When the project server is started from FuegoBPM Studio (select **Run->Start Server** from the menu options), all the Organization structure defined using the Organization window is copied to an isolated environment where the server executes processes.

Changes introduced while the Server is running will not be updated to the runtime environment until the Server is stopped and started or until a **Refresh Server Data** operation is performed. Depending on how **Runtime** server properties are set, the **Refresh Server Data** operation might be performed automatically after introducing changes to the organization structure.

See Server Properties for further information on how the runtime environment is updated with the latest changes. Some changes might require users currently logged in to Work Portal first log out before having the changes available in their Work Portal sessions. Refer to Refreshing Server Data for further details on this topic.

At the bottom of the window, the **Export** button allows you to export the organization structure data to a file. The **Import data** button allows you to import the organization structure information from a file generated in another project or in previous FuegoBPM Studio versions.

Note



In this online help system, organizational data is referred to as objects or entries.

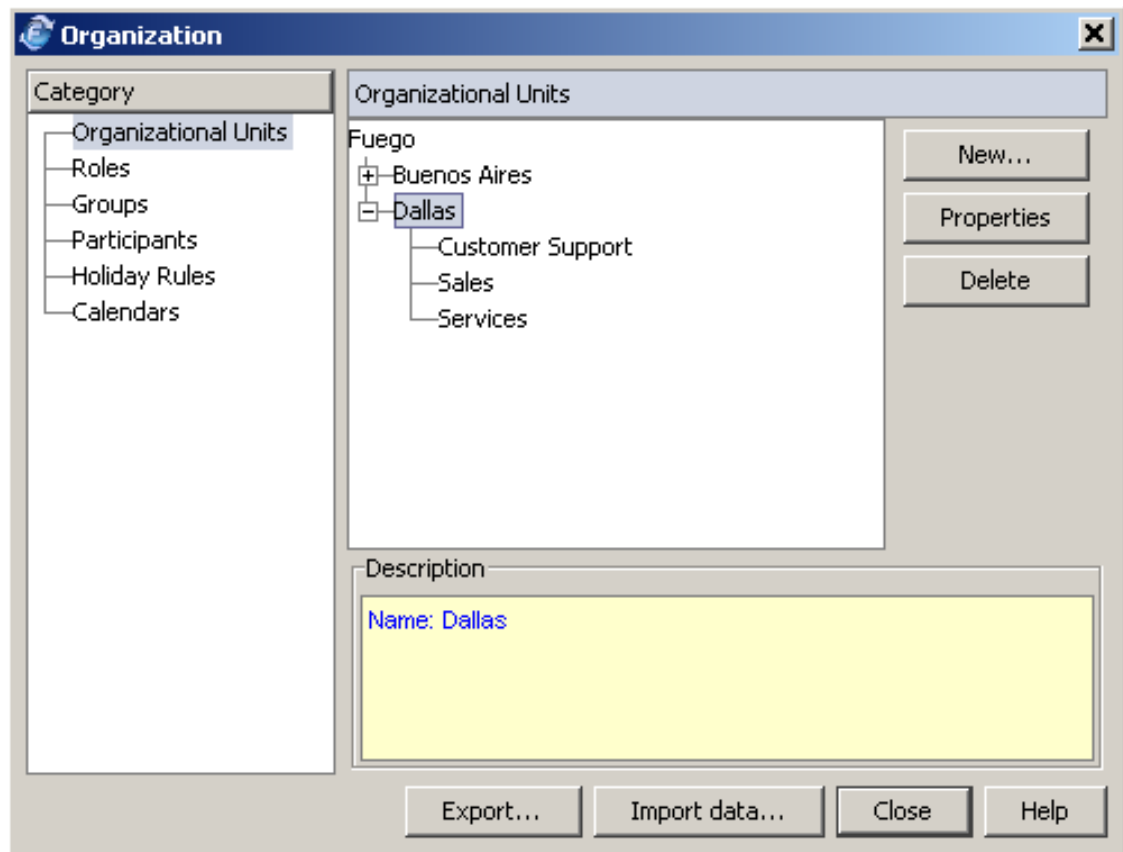
Organization objects views

The Organization window provides different views that allow you to define and update the attributes or properties for each type of object stored in the directory server.

The following views are provided:

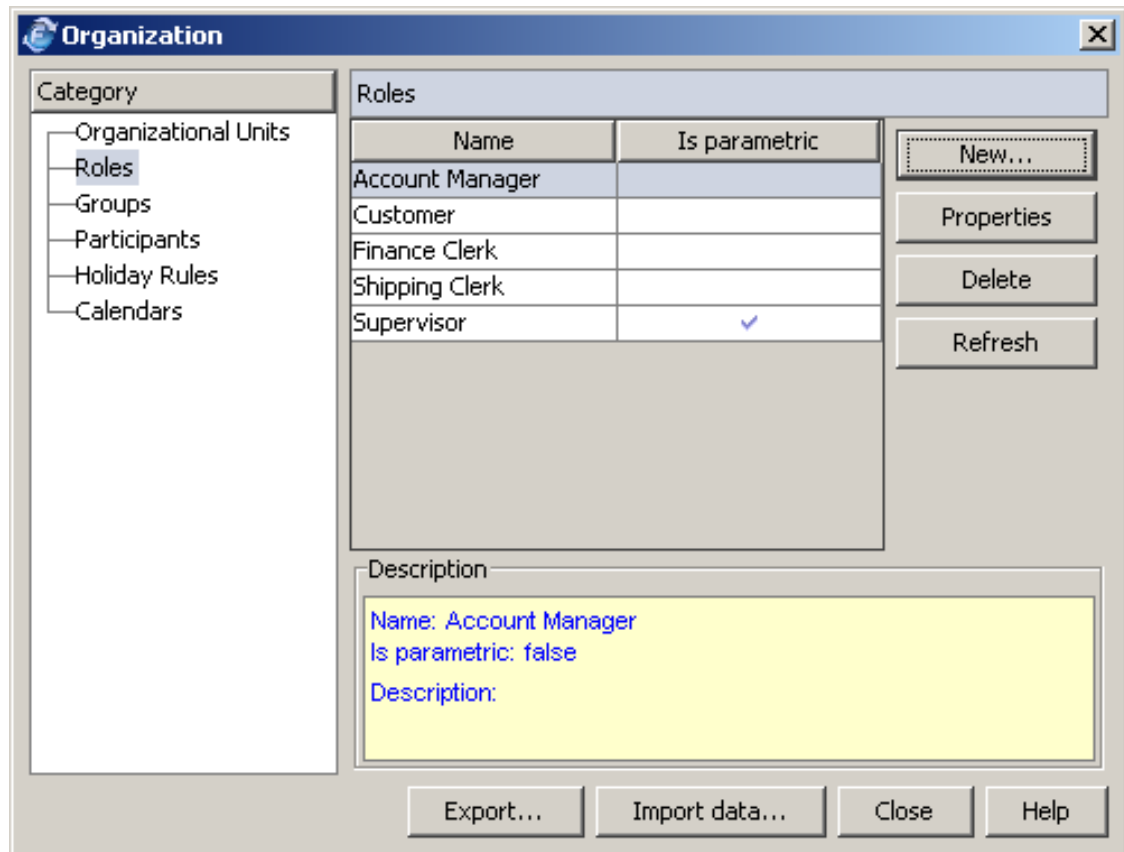
Organizational Units

The Organizational Units view is the default view that is displayed when you open the Organization window. The Organizational Units view allows you to define, update, and delete organizational unit information. In this view, the information is shown in a tree to represent the hierarchical relationships between organizational units. After selecting one of the nodes in the tree, you can click on the **Properties** button to edit the organizational unit information.



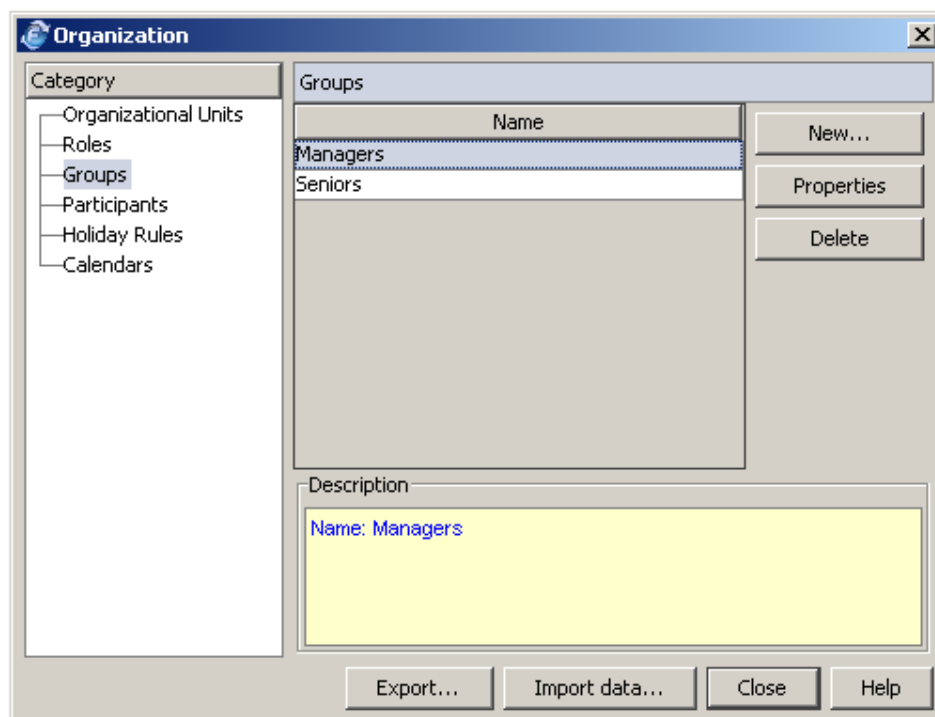
Organizational Roles

The **Roles** view displays a listing of roles defined in the organization. The Properties button displays a new window with the role data.



Organizational Groups

The **Groups** view displays all the groups defined in the company.



Participants

Participants view displays all users defined in the organization and stored in the directory service. When you click on the **Properties** button, a new window opens displaying all the participant properties.

Organization

Category

- Organizational Units
- Roles
- Groups
- Participants**
- Holiday Rules
- Calendars

Participants

Name	Testing
Jane Doe	✓
John Smith	
Mary Jones	
Robert Adams	

New...
Properties
Delete

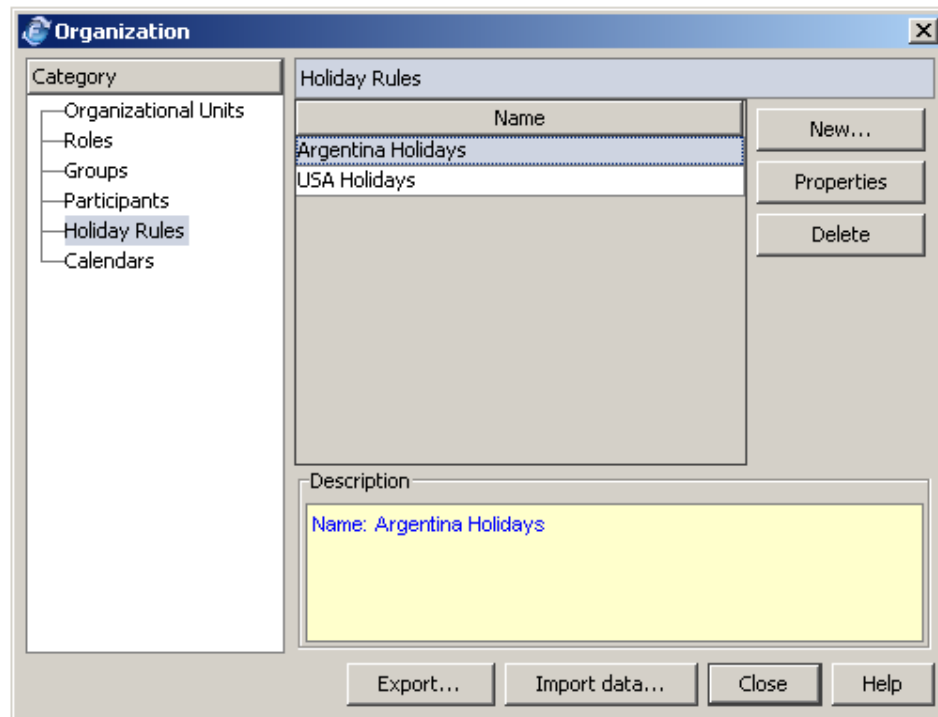
Description

Name: Robert Adams
Quantity: 1
Cost per Hour: 0.0
Efficiency: 80

Export... Import data... Close Help

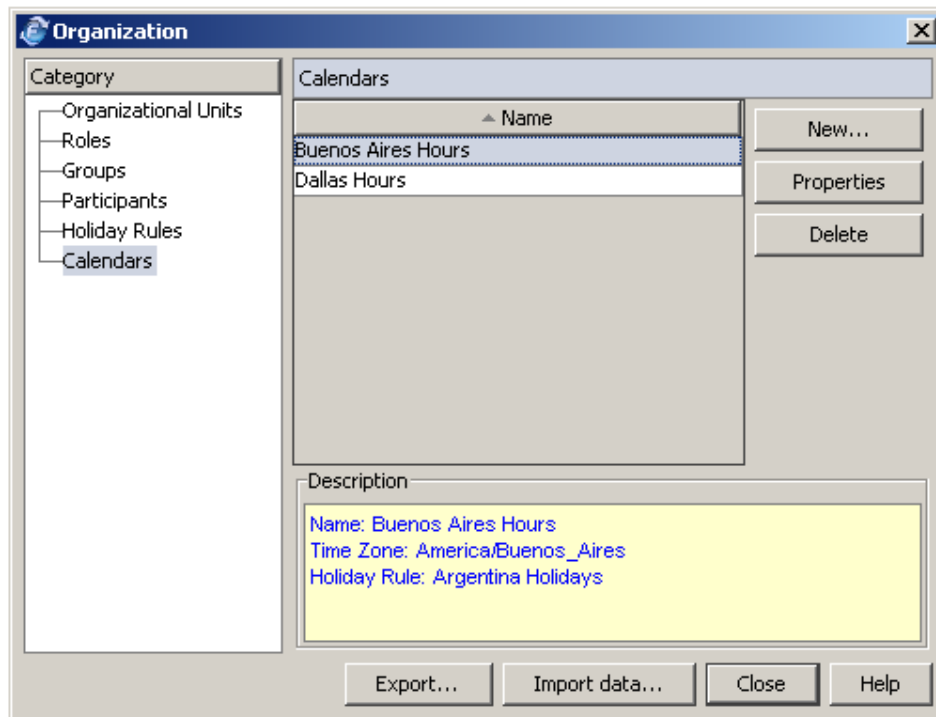
Holiday Rules

The **Holiday Rules** view displays all the Organization's defined holiday rules.



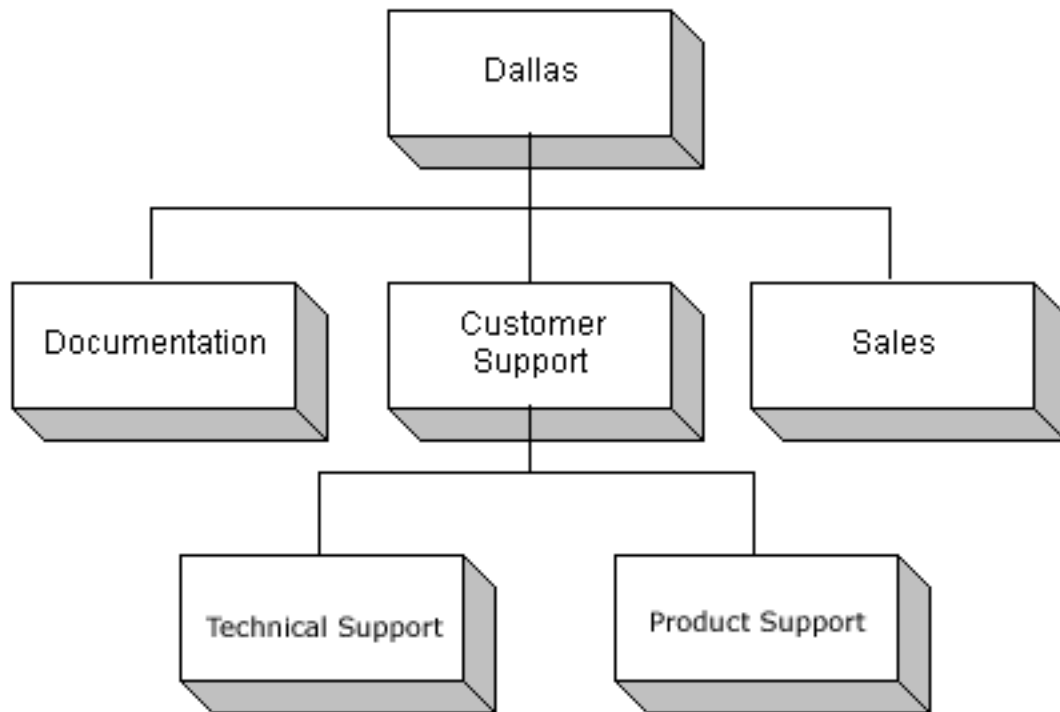
Calendar Rules

The **Calendar Rules** view displays all company calendar rules.



Organizational Units

Organizational units are, typically, departments or divisions within an organization. Organizational units can be organized in a hierarchy, for example:



In this hierarchy, Dallas contains the Customer Support, Sales, and Documentation organizational units. Customer Support contains Product Support and Technical Support organizational units. Once the organizational units have been defined, the employees of the company defined as FuegoBPM Studio participants might be assigned to one of the organizational units in the hierarchy. Processes can be deployed for one of the organizational units defined here so that only participants in that organizational unit and in lower levels within the hierarchy are able to perform tasks in a process.

Every Organizational Unit might have a different calendar rule associated to it. This allows the FuegoBPM Enterprise Server to take into account time zones and working schedules set for the Organizational Unit where processes are deployed and to calculate deadlines accordingly.

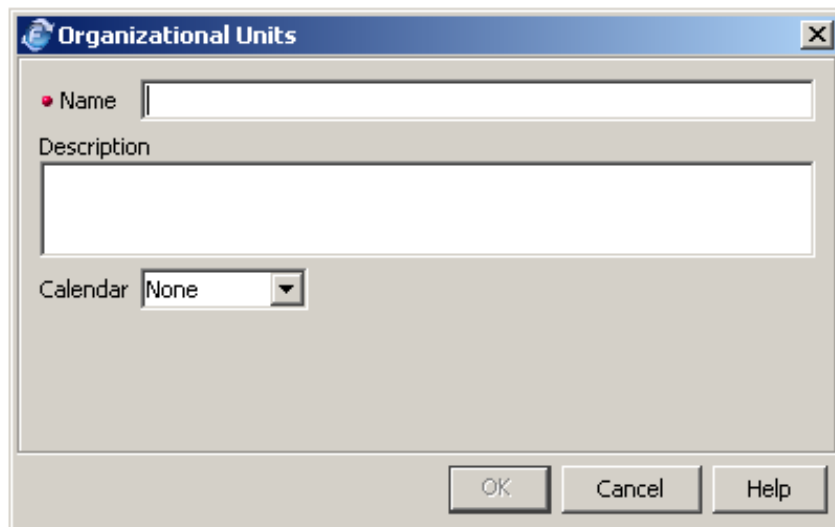
The Organizational Units view in the Organization window allows you to define the organizational hierarchy and the organizational units' properties. Remember that all the changes introduced to the organizational structure require a **Refresh Server Data** operation to

be available in processes executions if the server is currently running. Refer to Refreshing Server Data for further information on this topic.

Creating Organizational Units

To add an organizational unit

1. Open the Organization window. Next, select Organizational Units category in the left panel, and the organization tree appears. If no organizational unit has been defined, you will only see the root Organization. The organization name displayed as the root node in the tree is the one you enter when creating the project. Click the **New** button: A blank organizational unit profile is displayed in a new window.



2. Type the **Organizational Unit Name** and a **Description** .
3. You can assign calendar rules at any organizational level (organization or organizational unit). The Server uses the calendar rule defined for an organizational unit in order to calculate due dates for instances of processes deployed in this organizational unit. For further detailed information on how

FuegoBPM Studio uses calendar rules to calculate deadlines, see Calendar Rules. To assign a calendar rule to the organizational unit, select the rule from the drop-down box.

4. Click **Save** .

Note that if you do not have any organizational unit selected when you click **New**, the new organizational unit is created under the root organization node in the tree. If you need to create a hierarchical organizational unit, be sure that the organizational unit you want to be the parent of is selected in the organization tree when you click **New**.

Note

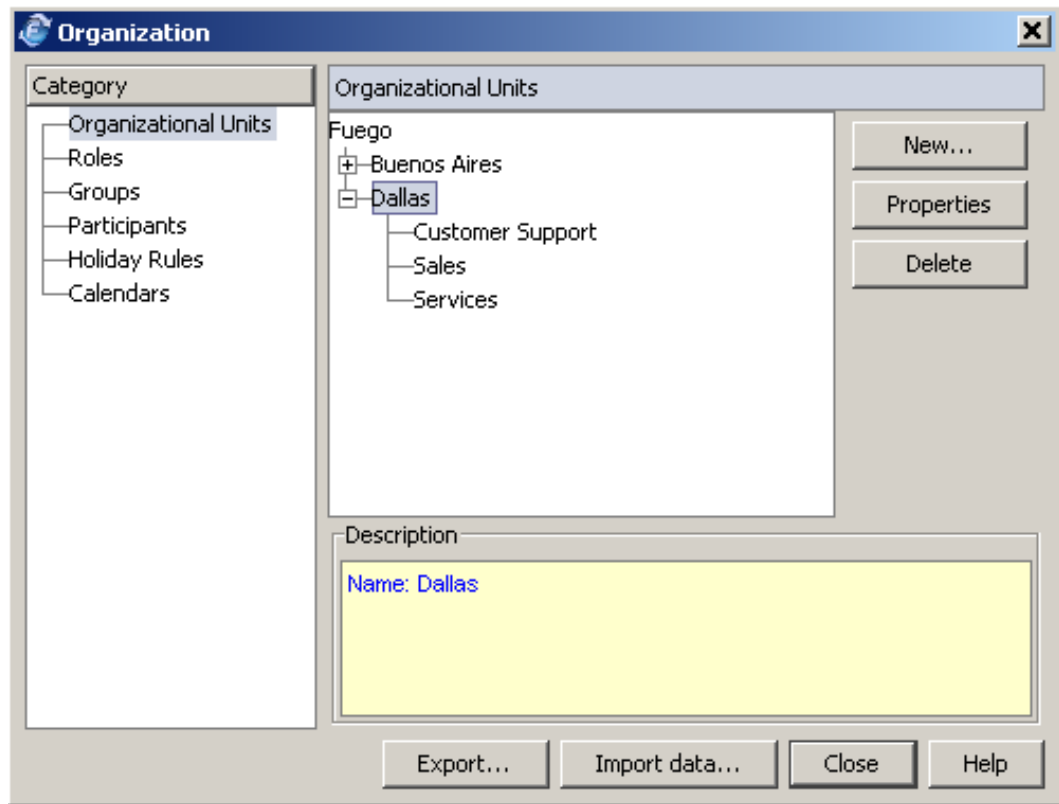


The calendar rule set to an Organizational Unit **does not** affect the time zone used to display information to participants belonging to that organizational unit. The time zone taken into account to display dates in Work Portal is the one set in the Work Portal *options* window. Options, including time zones, can be unique for each user.

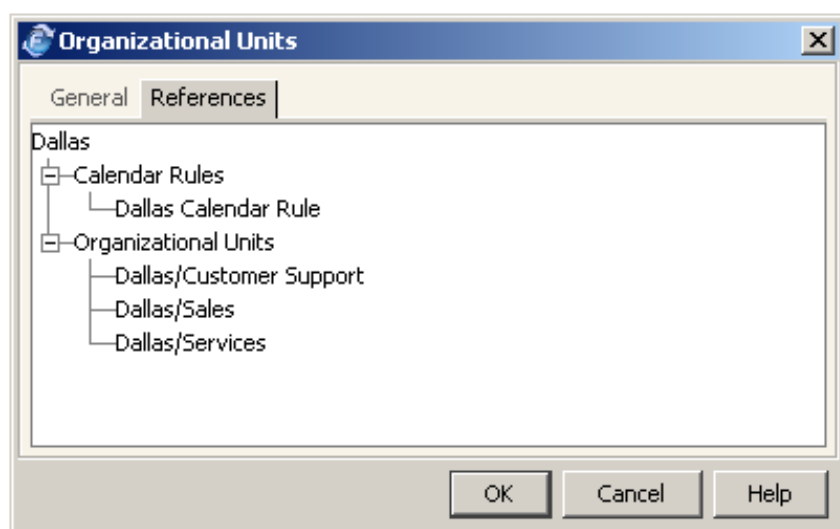
Editing Organizational Units

To edit an organizational unit

1. Open the Organization window. Next, select Organizational Units in the left panel.



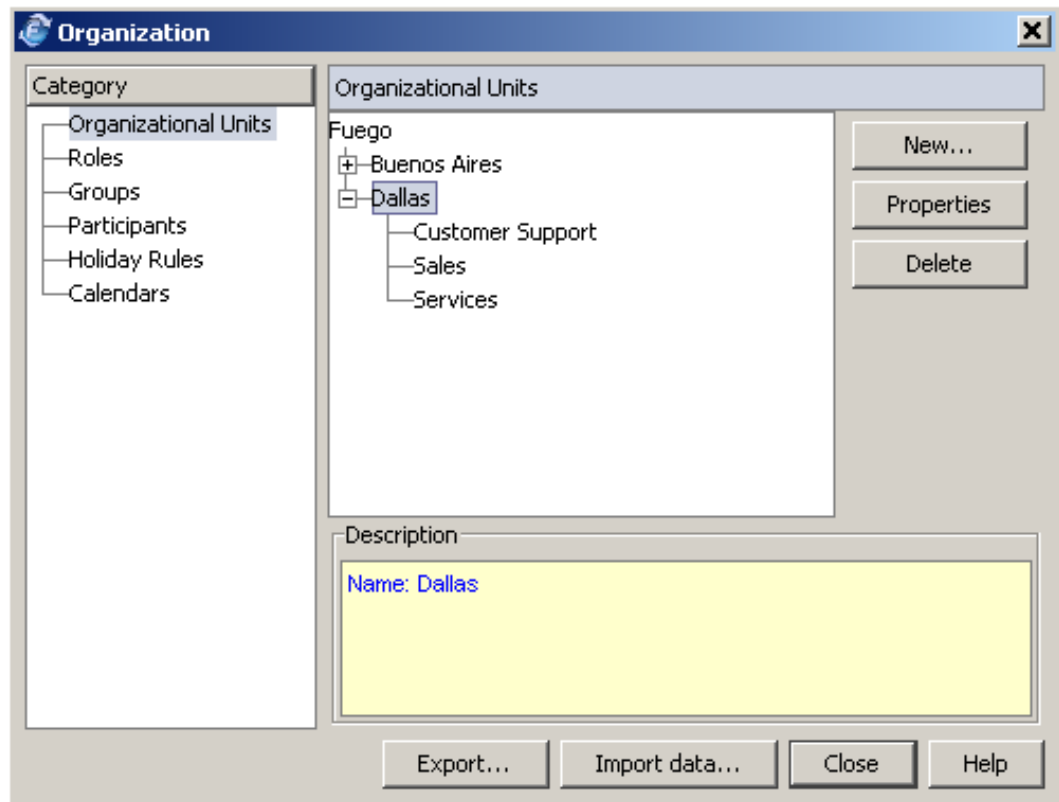
2. Select the **Organizational Unit** you want to edit from the Organization tree displayed in the right panel.
3. Click **Properties**.
4. Make all the changes required in the **General** tab.
5. Click **Save** to save the changes.
6. If you need to check how other objects in the organization use the organizational unit being edited, click the **References** tab. The references information is displayed.



Deleting Organizational Units


To delete an organizational unit

1. Open the Organization window. Next, select Organizational Units in the left panel.



2. Select the **Organizational Unit** you want to delete from the Organization tree displayed in the right panel.
3. Click **Delete**.

Note

 **Important Note** : When deleting an organizational unit to which participants belong, the participants are automatically moved to the first level in the organization.

Deleting Organizational Units when processes are deployed to it

FuegoBPM Studio allows you to make a process public for a specific Organizational Unit so that only participants belonging to that organizational unit can work on that process.

If you delete an organizational unit with processes deployed to it, the deployment information for the processes is automatically changed. Selecting **Run -> Server Preferences -> Deployment** from the **FuegoBPM Studio** menu options shows that, after deleting the organizational unit, the deploy preferences for the processes currently deployed to the unit are changed to be deployed to the root organization instead. Remember, for information displayed in **Deploy Preferences** to be applied in the runtime environment, you must republish the project. For further information on this issue, see Deployment.

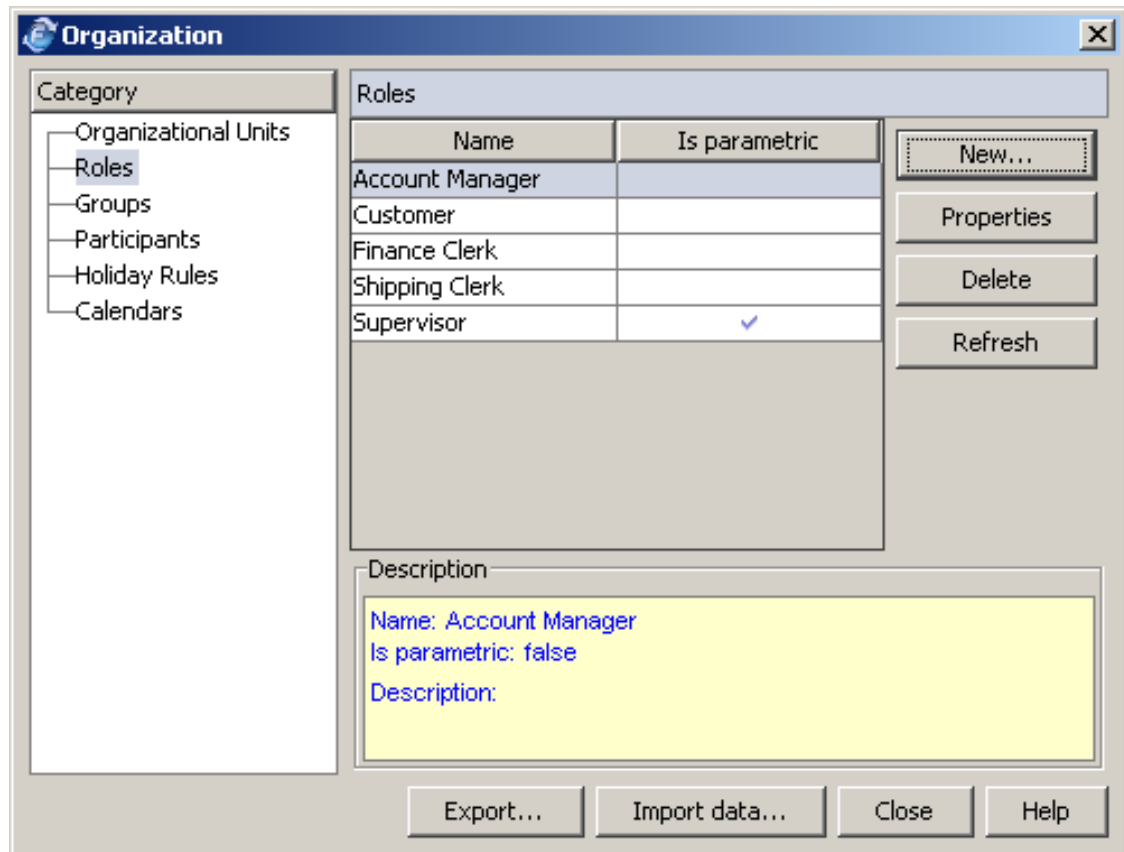
Organizational Roles

A role in the organization is a title that describes the activities performed by employees of the company. Employees are defined as FuegoBPM Studio participants. Examples of roles include Accounts Manager, Sales Clerk, or Customer.

A role in a process defines a job function for work including activities that do not require human intervention. Roles defined in the process are stored independently of the role information defined in the organization. This separation allows designers to develop processes as templates making them reusable in different organizations. However, in order to ease process deployment, FuegoBPM Studio ensures that an organizational role always exists for each user-defined role of the process and handles the mapping between them automatically using the role names. When using FuegoBPM Enterprise runtime edition, the mapping between process roles and real organization roles must be made manually when publishing the process.

Organizational role information can be viewed by selecting the **Roles** category in the Organization window's left panel. The Roles view in the **Organization** window allows you to create, modify, and delete roles in the organization. Take into account that any changes made to the **Organization** window while the FuegoBPM Enterprise Server is running requires you to refresh the Server data. Select **Run-> Refresh Server Data** to make the changes available when

executing processes. Refer to Refreshing Server Data for further information on this topic.



Creating Roles

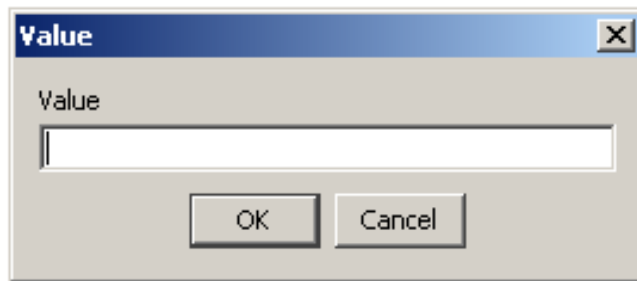
There are two ways to create a new role. Using the organization window, you can create a role to be used in the entire project. The second way is to create a role while designing a process. The former creates an organizational role. From that moment on, that role can be referred to by any process role in the project. The latter not only creates the organizational role to make it available for the rest of the processes, but it also maps that organizational role to the process role that is being designed.

To create a role from the Organization window

1. Open the Organization window.
2. Select the **Roles** category in the left panel, the roles view appears in the right panel. All roles defined also appear.
3. Click the **New** button. A blank Role profile is displayed in a new window:

The screenshot shows a 'Roles' dialog box. It has a title bar with the text 'Roles' and a close button. The main area contains a 'Name' text field, a 'Description' text area, an 'Is parametric' checkbox, and a 'Values' section. The 'Values' section includes a text box with the instruction 'Values splitting this role into subroles. They conform the set of valid values for the instance variable associated to this role at design time.', a table with one header 'Value', and 'Add' and 'Delete' buttons. At the bottom are 'OK', 'Cancel', and 'Help' buttons.

4. Type the **Role Name** and a **Description**.
5. If this Role is a **parametric role**, enable the **Is Parametric** check box. If the role is defined as parametric, you must enter at least one parametric value. In order to add a new parametric value, click **Add** next to the Values list. The **Value** dialog box appears.

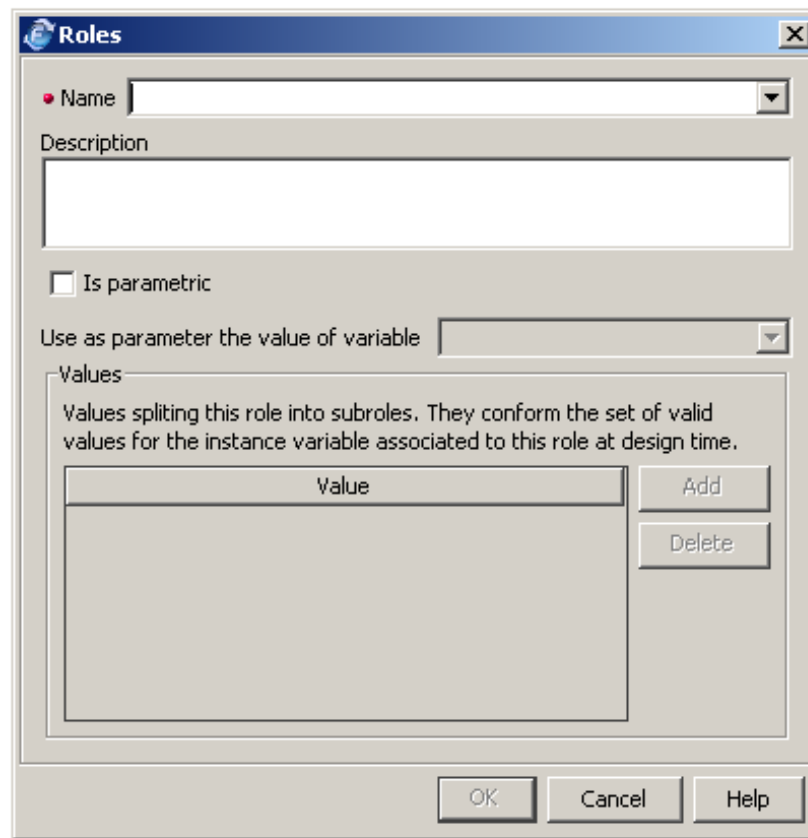


Type a name for this parametric value and click **OK**.

6. Click **Save**.

From the design development space

1. Right-click on any place in the design workspace. Select **Add Role** and then **New** from the shortcut menu. A blank Role profile is displayed in a new window.

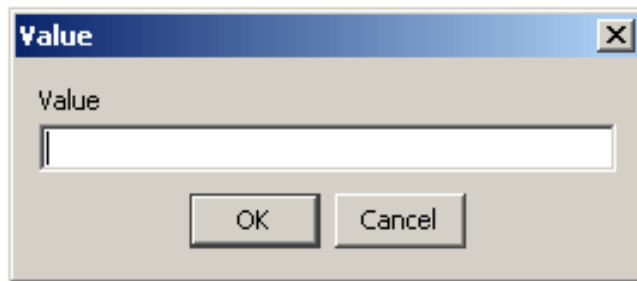


The image shows a dialog box titled "Roles". It contains the following fields and controls:

- Name:** A text input field with a dropdown arrow on the right.
- Description:** A large text area.
- Is parametric:** A checkbox.
- Use as parameter the value of variable:** A text input field with a dropdown arrow on the right.
- Values:** A section containing a text description: "Values splitting this role into subroles. They conform the set of valid values for the instance variable associated to this role at design time." Below this is a table with one header "Value" and one empty row. To the right of the table are two buttons: "Add" and "Delete".

At the bottom of the dialog box are three buttons: "OK", "Cancel", and "Help".

2. Type the **Role Name** and a **Description**.
3. If this Role is a **parametric role**, enable the **Is Parametric** check box. Select the instance variable whose values will be used as parameter to determine (at runtime) the sub-role in charge of performing tasks located in the parametric role. The check process operation fails if the role is not associated to a suitable variable. In the event that the instance variable has not been created, you can leave this field blank and edit the role information later after creating the variable that you will associate to this parametric role. If the role is defined as parametric, you must enter at least one parametric value. In order to add a new parametric value, click the **Add** button next to Values list. The **Value** dialog appears.



Type a name for this parametric value and click **OK**.

4. Click **Save**.

Note



Be aware that once you have defined a **non-parametric** role, you will not be able to convert it to a **parametric** role.

Parametric roles

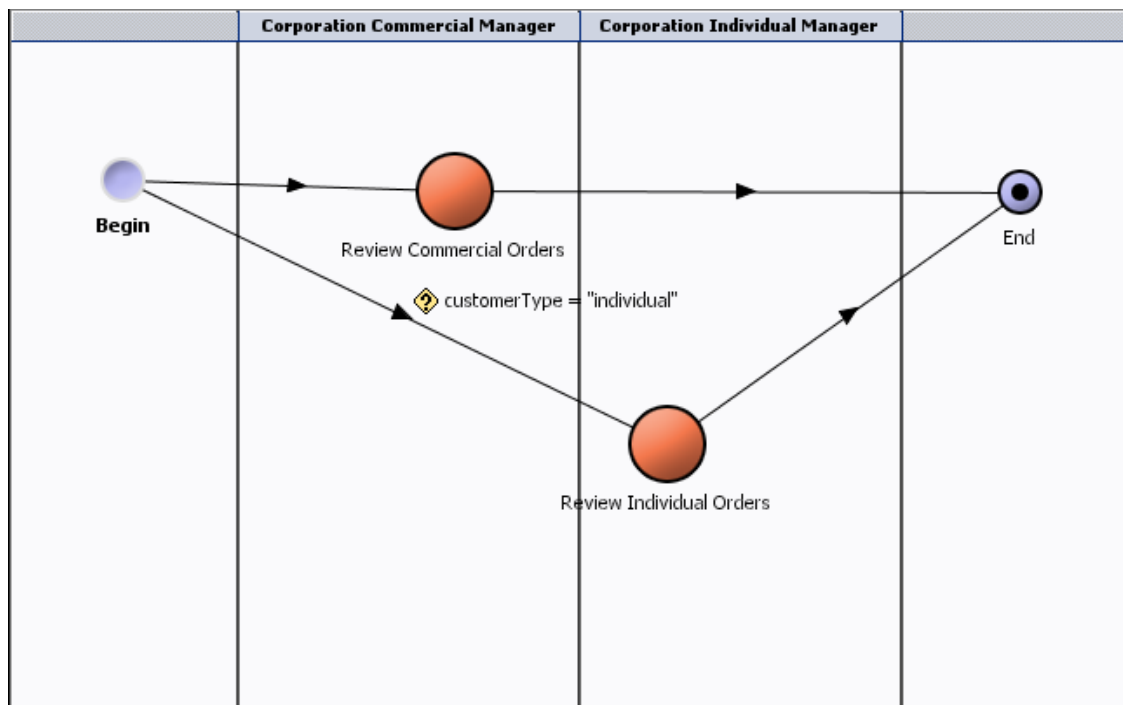
Parametric roles accommodate business circumstances requiring different groups of people to perform similar activities. They allow you to determine the group of people who can perform a process task at runtime instead of doing it at design time. This eliminates the need for redundant activities that vary only by the person or group that performs the activity.

Parametric roles also eliminate cumbersome conditional statements and split in the process design. Parametric roles make processes easier to read and understand as well as reduce design effort.

A parametric role example

In a Supply Chain process of a company, there is a task to review orders that can be performed by two different groups of people depending on the order being made by customer type.

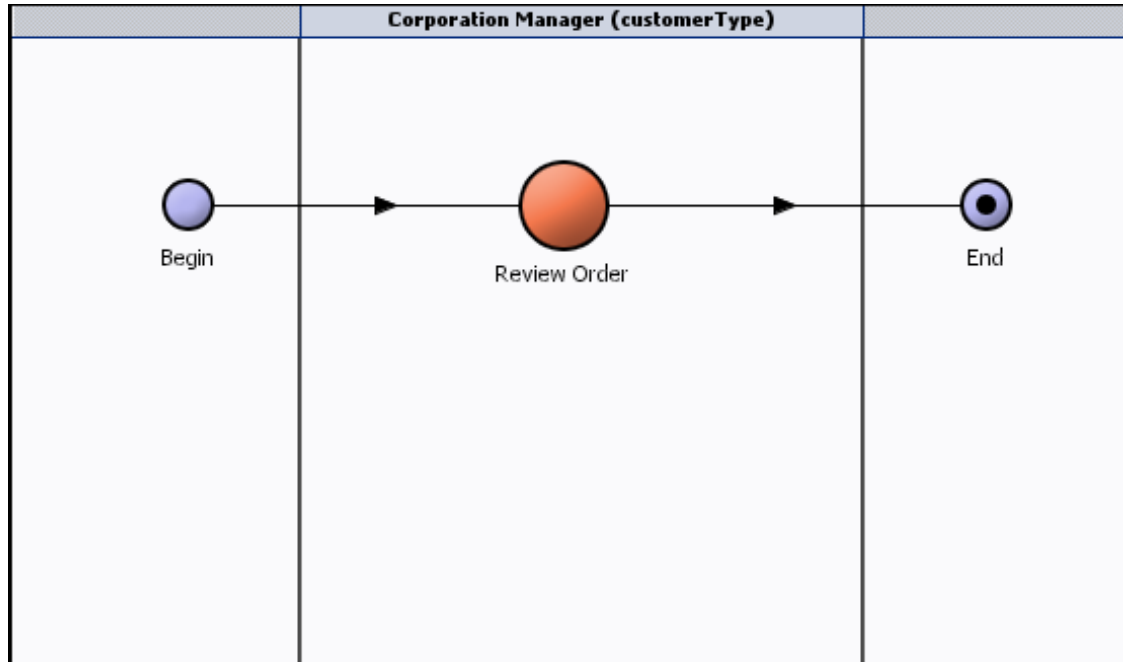
The example below shows the process design to model this situation without using parametric roles. The process has two activities performing exactly the same tasks to review orders. The activities are assigned to two different roles: one for commercial account managers and another for individual account managers. In addition, the process contains some logic in order to determine where the orders should be sent to be approved depending on the customer type. In this case, a conditional transition decides where to send the order.



The picture below shows how the same process can be modeled in a much simpler way by using only one role defined as parametric. Two parametric values were added to the role using the **Role** category in the **Organization** window, in this case, **individual** and **commercial** values. In the process design, if you right-click on the role column, the variable "customerType" is defined as the parameter of the role.

The role was assigned to commercial account managers through the **Participants** category in the **Organization** window by selecting the parametric value "commercial", and to individual account managers by selecting the parametric value "individual".

When processing the orders, the variable *customerType* is checked and all Commercial customer orders appear in commercial account managers' Work Portal queues while all Individual customer orders appear in individual accounts managers Work Portal queues.



Note that no logic is needed in the process design to determine where the order for review is to be sent. As you can see, the activity to review orders is defined only once, thus making the process design much cleaner.

Editing Roles

To edit a role

1. Select **Organization** from the **Project menu**.
2. Click on the **Roles** category in the left panel.
3. Click on the role you want to edit in the roles list displayed in the right panel.

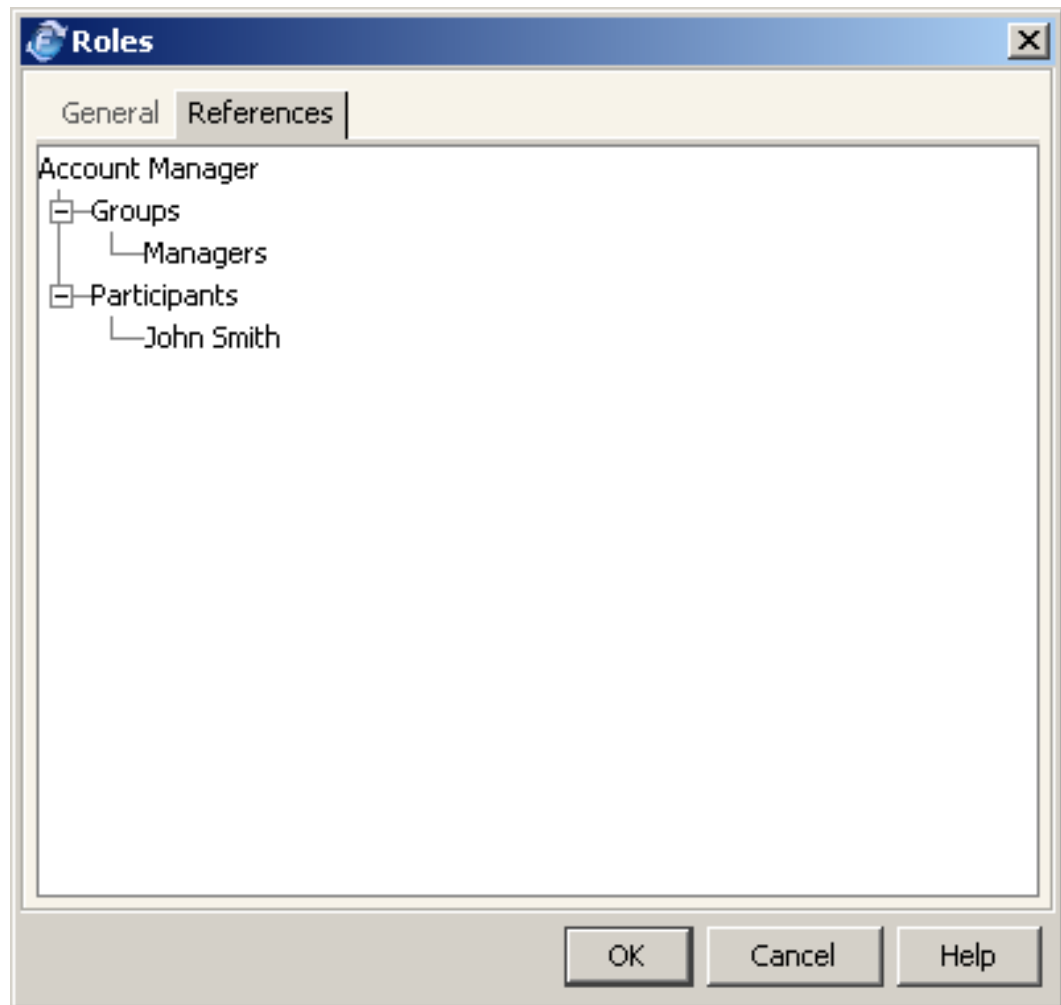
4. Click the **Properties** button.
5. Change data by clicking the **General** tab in the edit role window.

The screenshot shows a dialog box titled 'Roles' with a close button (X) in the top right corner. It has two tabs: 'General' (selected) and 'References'. The 'General' tab contains the following elements:

- A 'Name' field with a red dot icon, containing the text 'Account Manager'.
- A 'Description' text area, currently empty.
- An unchecked checkbox labeled 'Is parametric'.
- A section titled 'Values' with a description: 'Values splitting this role into subroles. They conform the set of valid values for the instance variable associated to this role at design time.'
- A table with one header 'Value' and one empty row below it.
- Two buttons, 'Add' and 'Remove', to the right of the table.

At the bottom of the dialog are three buttons: 'OK', 'Cancel', and 'Help'.

6. If you need to check references of the role before changing or deleting the role, click the **References** tab. The information on how other organization objects refer to the role you are editing is displayed.



Deleting Roles

To delete a role

1. Select **Organization** from the **Project menu**.
2. Click on the **Roles** category in the left panel.
3. Click on the role you want to delete in the roles list displayed in the right panel.

4. Click the **Delete** button.

Note



The role can be deleted provided that no process is using it. In addition, take into account that all the references to the deleted role are also removed, including role assignments to participants and groups.

Refreshing Roles

The **Refresh** button is used to check all processes and determine if any role within a process is not defined in the roles list. If so, the role is added.

This is only valid if you have imported a project or checked it out (update your project) from the repository. If new processes containing new roles were added, these are not added to your project until you open the process in the Studio designer and either publish the project or click the **Refresh** button.

Setting a default Participant in a Role

You can dynamically set the default participant for a Role. This has to be done using the **Role component**.

When an instance arrives to an activity that belongs to a role, and this role has a default participant set (previously set in an activity's BP-Method using the component), the instance is automatically selected by that participant.

For example if you know in **activity A** that when the instance arrives to any activity that can be processed by the Role *Credit Rep*, this instance has to be processed by *John Smith* that is the supervisor, then you have to set in the **activity A BP-Method**, the default participant for Role *Credit Rep* as *John Smith*.

Warning



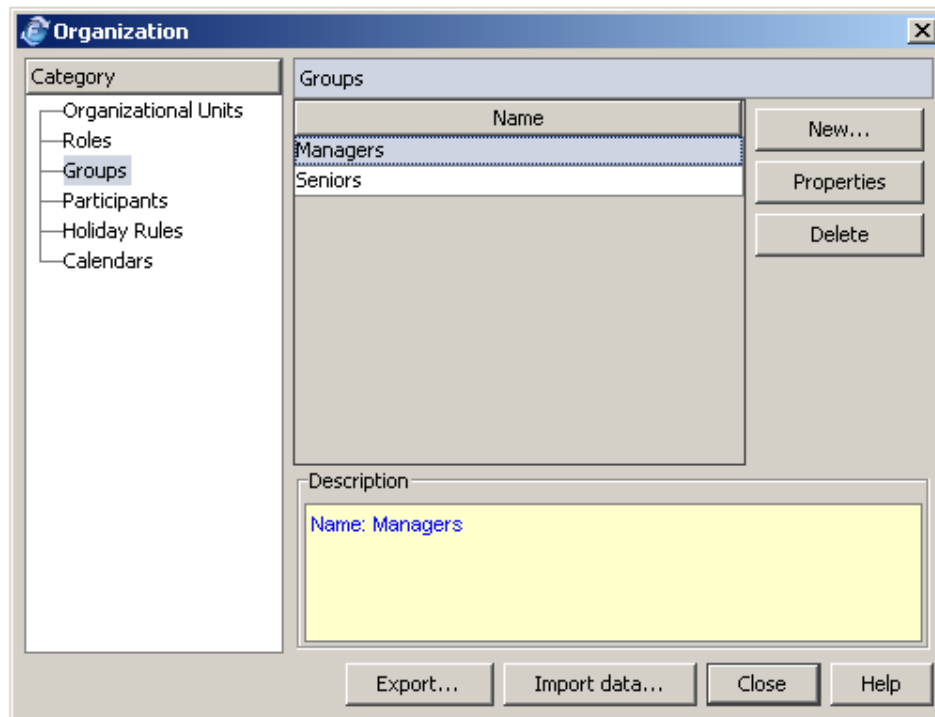
If you set a next participant to the instance within the activity's BP-Method, this participant is selected and not the default participant for role.

See the component documentation in FuegoBPM Studio component section for further information.

Organizational Groups

A group is a profile. You can include Participants in a group to provide them with the abilities defined for the group. A set of roles is assigned to the group. When participants log in to Work Portal, the groups to which the participants belong are checked in order to determine the final set of roles they play within the organization. This means that the participant inherits all the roles defined for the groups he or she belongs to.

Group information can be viewed by selecting the **Groups** category in the Organization window's left panel. The Groups view in the Organization window allows you to create, modify, and delete groups in the organization. Take into account any changes made in the **Organization** window while the FuegoBPM Enterprise Server is running, it requires you to **Refresh Server Data**. This option is located on the **Run** menu. It makes the changes available when executing processes. Refer to Refreshing Server Data detailed documentation for further information on this topic.



Creating groups

To create a group

1. Open the Organization window and select the **Groups** category in the left panel. The groups view, listing all the organizational groups that have been defined, appears in the right panel.
2. Click the **New** button. A blank Group profile is displayed in a new window.

Group

• Name

Description

Groups
Groups conforming this group to extend its functionality

Name

Add
Delete

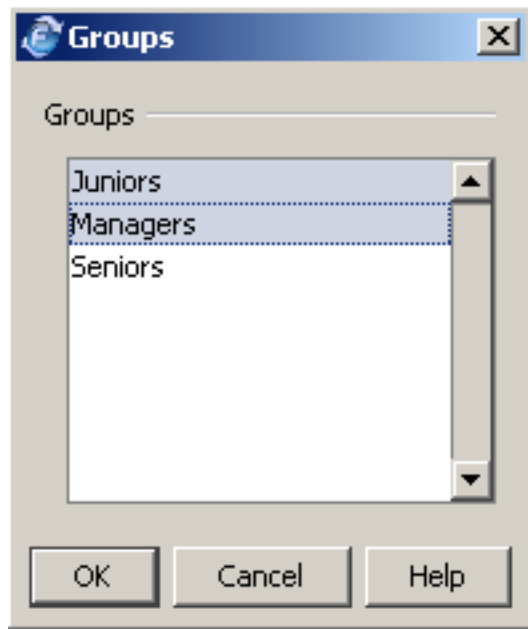
Roles
Roles inherited by all the members of this group

Role	Parameter

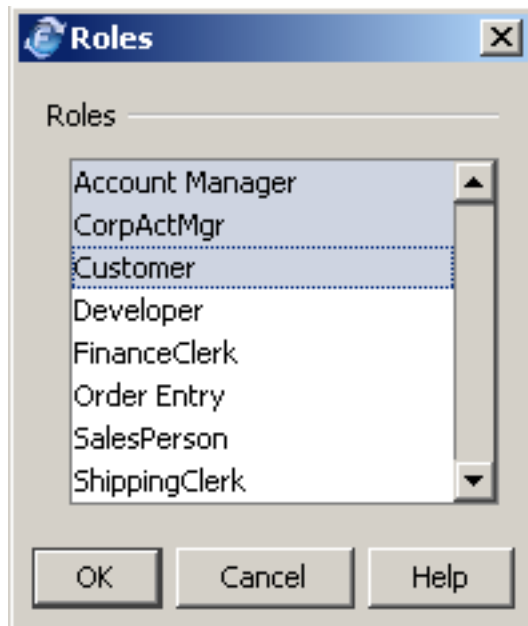
Add
Delete

OK Cancel Help

3. If needed, add any nested groups to the Groups list. Adding other groups to this list means that all the members of the group will inherit not only the roles defined in the Roles list, but also the roles of the groups in the Groups list. In order to add a group, click the **Add** button next to the **Groups** list. Select one of the groups displayed in the drop-down list.



4. Add the organizational roles that are inherited by all the members of the group being created. To do this, click the **Add** button next to Roles assignment list. Select the role from the drop-down list and click **OK**.



5. If the role is parametric, you must choose one of the available parameters in the drop-down list displayed in the **Parameter**

column.

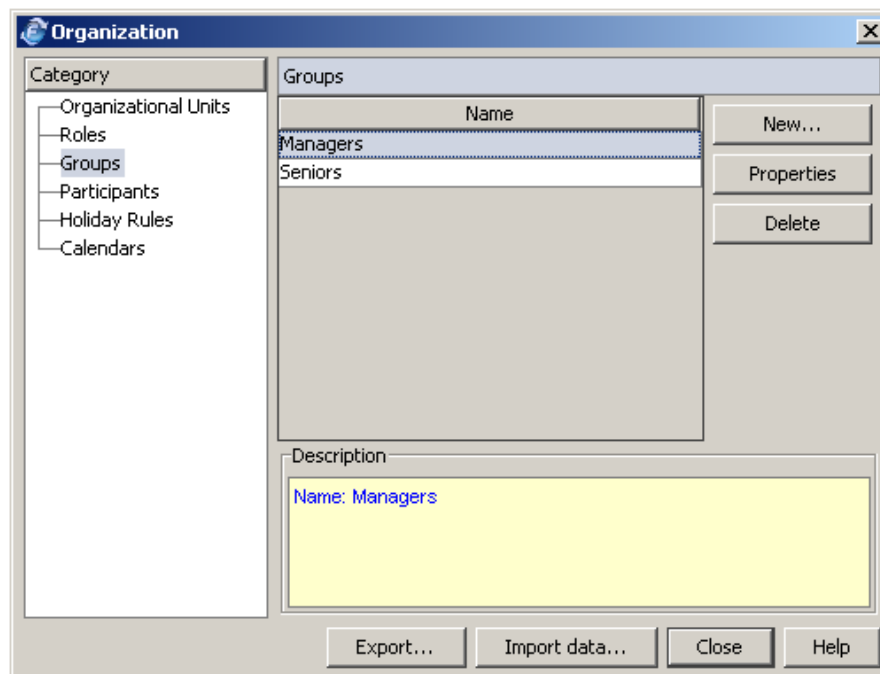
6. Click **Save**.

Editing Groups

Once the group has been created, you can change any of the settings initially assigned.

To edit a group

1. Open the Organization window.
2. Click **Groups** category in the left panel.
3. Click on the group you want to edit in the groups list displayed in the right panel.



4. Click the **Properties** button.

5. Modify all the settings you need by using the **General** tab in the edit Group window.

Group

General | References

Name: Managers

Description: Managers

Groups

Groups conforming this group to extend its functionality

Name

Add Delete

Roles

Roles inherited by all the members of this group

Role	Parameter
Account Manager	commercial
Account Manager	individual

Add Delete

OK Cancel Help

6. Click the **References** tab to view the details on how other objects reference the group being edited. This is useful if you want to check references before modifying or deleting the group.



7. Click **Save**.

Deleting Groups

To delete a group

1. Open the Organization window.
2. Click on the **Groups** category in the left panel.
3. Click on the group you want to delete in the groups list displayed

in the right panel.

4. Click the **Delete** button.

Note



When deleting a group all the references to the group deleted will also be removed, including group assignments to participants and other groups.

Participants

Participants defined in the organization are all the people enabled to track and perform tasks of business processes designed and developed with FuegoBPM Studio.

Participants might belong to an Organizational Unit. If so, he or she can only perform tasks on processes deployed in that organizational unit or any of the lower levels within the organizational unit's hierarchy.

You can assign a set of roles to a participant. When a participant logs on to Work Portal, he or she is able to perform all the tasks defined for the roles assigned to the participant.

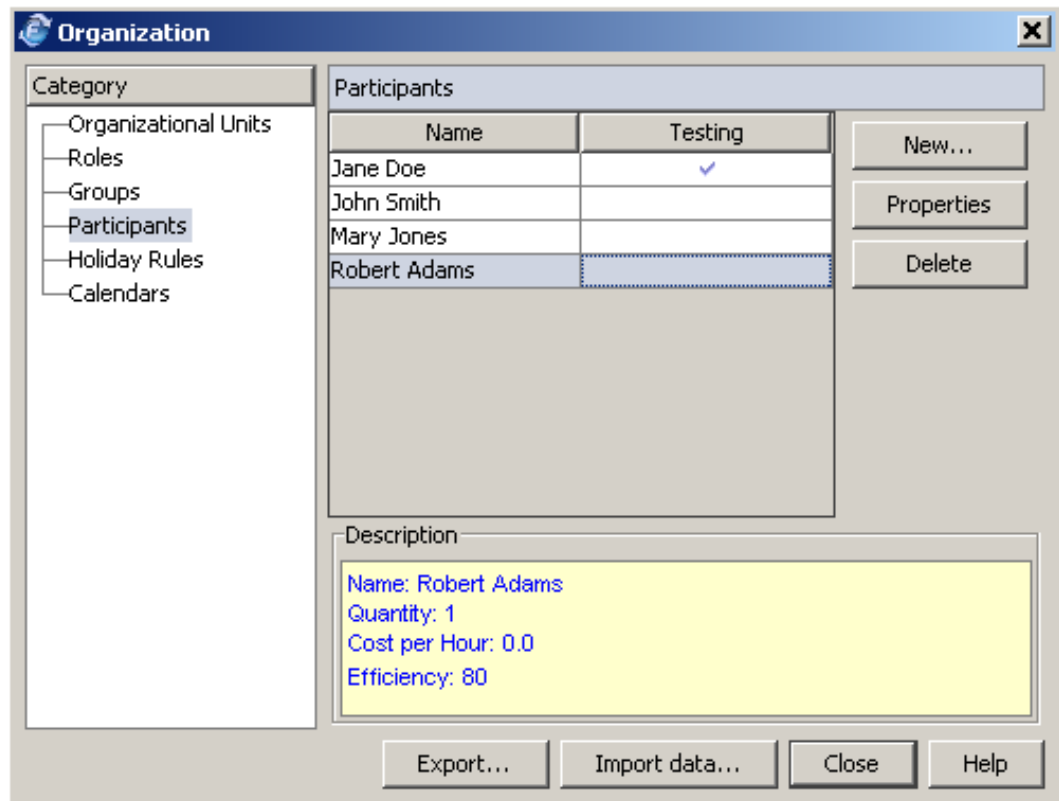
The Participant view in the **Organization** window allows you to create, modify, and delete participants in the organization. You can create both participants exist once the project is installed in a production environment and you can also create participants and participant profiles for testing purposes while designing the project.

When you make any changes to the Organization window while the FuegoBPM Enterprise Server is running, you must **Refresh Server Data**. This option is located on FuegoBPM Studio's **Run** menu. It makes the changes available when executing processes. Refer to Refreshing Server Data for further information on this topic.

Creating Participants

To create a participant

1. Open the Organization window. Select the **Participants** category displayed in the left panel.



2. Click the **New** button. The blank participant profile appears in a new window with two tabs.

Participant

General | Advanced

Name

Organizational Unit

E-mail Address

Groups

Groups to which the participant belongs

Name

Add
Delete

Roles

Roles that the participant carries out

Role	Parameter

Add
Delete

OK Cancel Help

3. Click on the **General** tab. Type the user Name in the **Name** field.
4. Optionally, select the organizational unit from the drop-down list to assign the participant to an organizational unit. If you do not select an organizational unit, the participant is a member of all the organizational units.

Participant

General | Advanced

Name:

Organizational Unit: **Fuego**

E-mail Address:

Groups:

Groups to which this participant belongs:

Roles:

Roles that the participant carries out

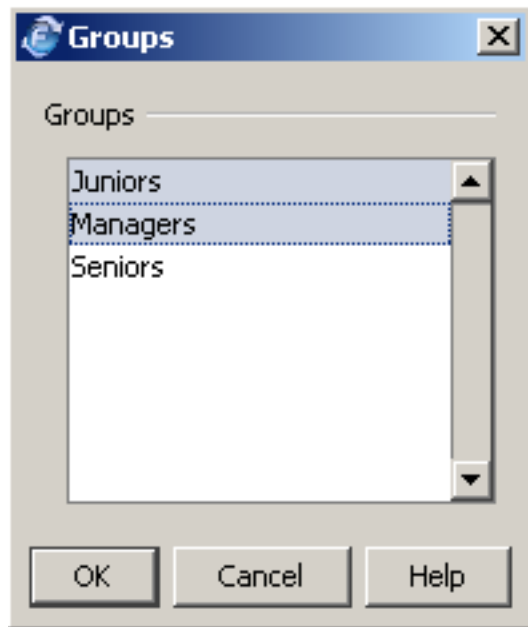
Role	Parameter

Add

Delete

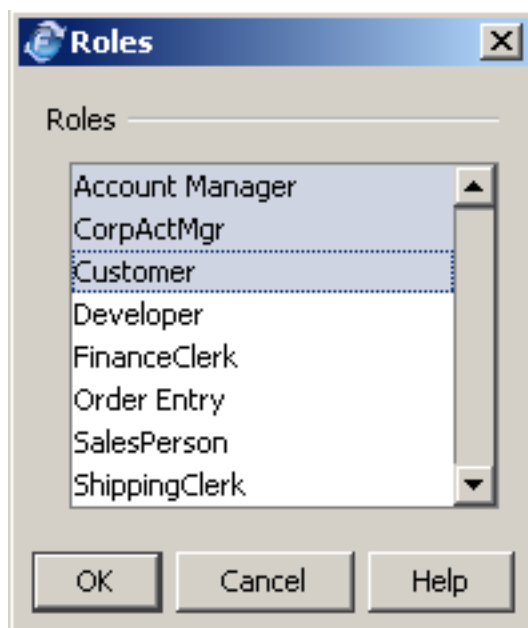
OK Cancel Help

5. Optionally, type the user's e-mail address.
6. If you want this participant to belong to one or more groups, click the **Add** button next to the Groups' list.



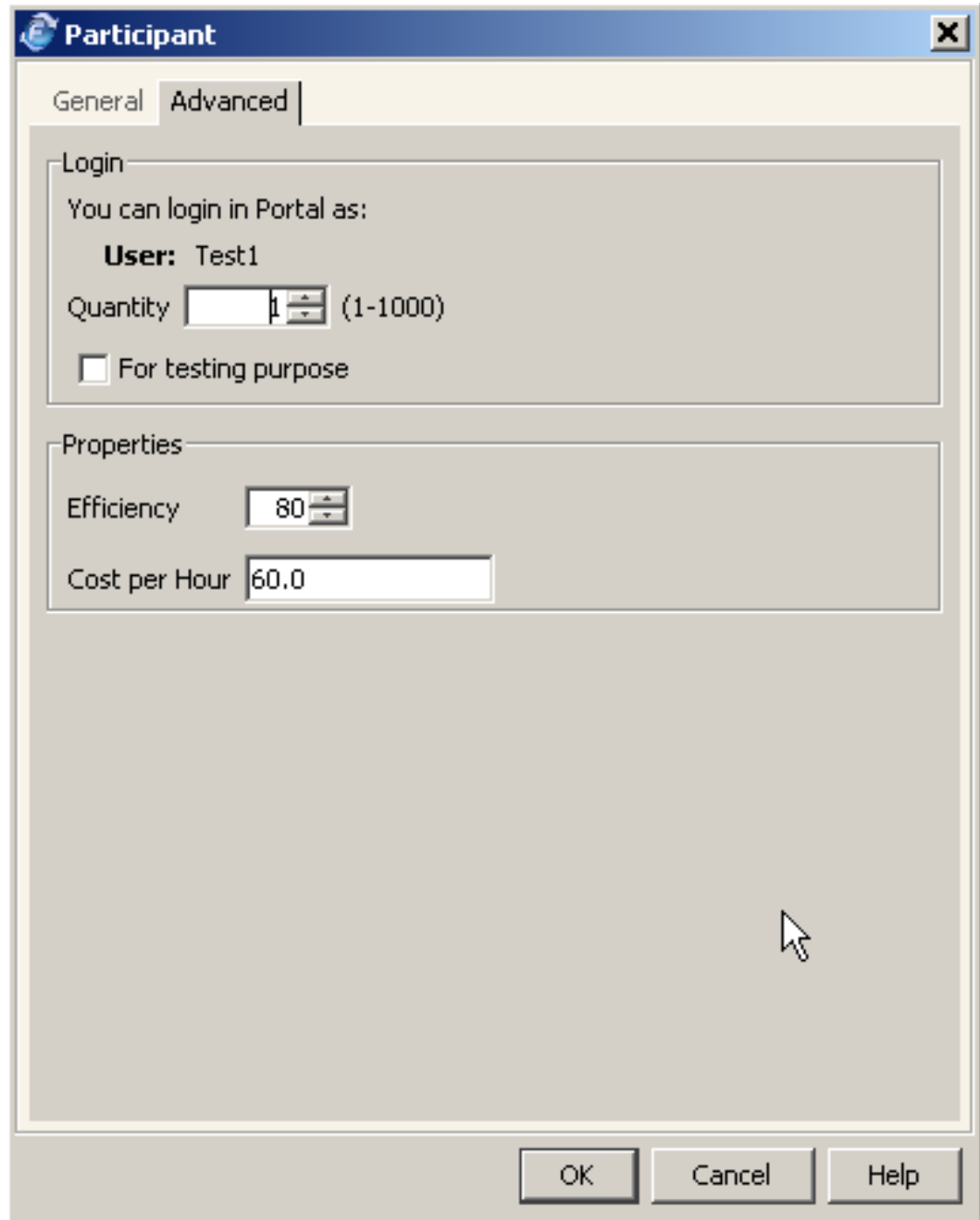
Select the list of groups from the drop-down list and click **OK**.

7. If you want to assign a role to this participant, click **Add** in the Role Assignment area.



8. Select the list of roles from the drop-down list and click **OK**.
9. If you are creating a participant for testing purposes only, open

the tab **Advanced** and check the **For testing purposes** checkbox. This checkbox is not checked by default, meaning that the participant will be created in the production environment where you install the project. Participants created to test the process design should be enabled and set as for testing purposes from here.



The screenshot shows a Windows-style dialog box titled "Participant" with a close button (X) in the top right corner. The dialog has two tabs: "General" and "Advanced", with "Advanced" currently selected. The "Advanced" tab contains two main sections: "Login" and "Properties".

Login Section:

- Text: "You can login in Portal as:"
- Text: **User:** Test1
- Text: Quantity (1-1000)
- Text: ☐ For testing purpose

Properties Section:

- Text: Efficiency
- Text: Cost per Hour

At the bottom of the dialog are three buttons: "OK", "Cancel", and "Help". A mouse cursor is visible over the "Cancel" button.

10. Click **Save**.

Creating multiple participants for testing and simulation

FuegoBPM Studio allows you to easily test how a process works with a large number of users connecting to it.

On the **Advanced** tab in the **Participants** view, you can simulate the creation of large numbers of participants without creating them one-by-one.

In order to provide this ability, when advanced preferences are set, for each participant that you configure in **General** tab, you will have as many participants available to connect to Work Portal as defined in the **Quantity** field. However, the participants created here are not real participants and will only be available for testing purposes while working on development environments. This means that once the project is deployed in a production environment, if the **For testing purposes** check box is unchecked, only one participant will be able to log in to Work Portal. The one with the ID defined in the **General** tab and not the others that are automatically created for simulation purposes. If the check box **For testing purposes** is checked, this user will not be able to connect to Work Portal when the project is deployed in a production environment.

FuegoBPM Studio provides the ability to simulate execution of a process. Simulation **does not** execute the activity tasks but only simulates their execution by waiting the time the tasks **would** take to execute. In the simulation of the process, some properties of the participant are considered when calculating statistics.

By entering these properties in the **Advanced** tab, you can test your process with the number of participants you specify and simulate the execution of the process when participants have these characteristics.

To create a participant profile

The same set of steps mentioned to create a participant apply to create a participant profile. Participants created here will all have the roles and groups defined on the **General** tab.

After entering data in **General** tab:

1. Click on the **Advanced** tab. The **Advanced** properties tab is displayed.

The screenshot shows a Windows-style dialog box titled "Participant". It has two tabs: "General" and "Advanced", with "Advanced" currently selected. The dialog is divided into two main sections: "Login" and "Properties".

Login Section:

- Text: "You can login in Portal as:"
- Text: **User:** Test1, Test12,..., Test14
- Text: Quantity (1-1000)
- Text: ☒ For testing purpose

Properties Section:

- Text: Efficiency
- Text: Cost per Hour

At the bottom of the dialog are three buttons: "OK", "Cancel", and "Help". A mouse cursor is visible over the "OK" button.

2. In the *Login* section, information referring to user names that you can use to login to Work Portal is displayed. Enter the number of participants in the **Quantity** field. If, for instance, you enter 100, then 100 participants are created. The participants' names are composed of the **Name** you entered in the **General** tab and a sequential number from 2 to 100 as a suffix. After that, you

can connect to Work Portal with 100 different participants without having to create them one-by-one.

3. In the *Properties* section, in the **Cost** field, enter how much each human resource with this profile will cost the company.
4. In the **Efficiency** field, enter the efficiency percentage estimated for participants with this profile. The higher the percentage, the more efficient the participant is.

Note



While testing your processes in a development environment, Work Portal does not require any password to log in. Once FuegoBPM Express/Enterprise is installed in a production environment, users defined through the FuegoBPM Studio Organization window will be able to log in to Work Portal using the same id and password. The first time every user logs in, Work Portal prompts the user to change the password.

Default Participant for a Role

You can bind an instance with a participant for a specific role. This means that whenever the instance gets to a specific role, it is automatically assigned to the associated participant.

To dynamically set the default participant for a Role you have to use the FuegoBPM's **Role component**.

When an instance arrives to an activity that belongs to a role, and this role has a default participant set (previously set in an activity's BP-Method using the component), the instance is automatically selected by that participant.

For example if the instance is in **activity A** and at that point you know that when the instance arrives to any activity that can be processed by the Role *Credit Rep*, this instance has to be processed by *John Smith* that is the supervisor, then you have to set in the **activity A BP-Method**, the default participant for Role *Credit Rep* as *John Smith*.

Warning



The Participant.next has higher precedence over the Participant for Role in case someone in the Work Portal uses the "Send To". If you set a next participant to the instance within the activity's BP-Method, this participant is selected and not the default participant for role.

Example:

To set the default participant for a role :

```
r1 as Role
r1 = Activity.role.find(name : "role1")
r1.defaultParticipantForCurrentInstance =
    Participant.find(name : "test1")
```

See the component documentation in FuegoBPM Studio component section for further information.

Editing Participants

To edit a participant

1. Open the Organization window.
2. Click on **Participants** in the left panel.
3. Click on the participant you want to edit in the participants list in the right panel.
4. Click the **Properties** button to edit the participant settings.
5. Change the information you need in the **General** and **Advanced** tabs. Remember that all the changes will not be updated to the runtime environment unless a **Refresh Server Data** option is performed.

6. Click **Save** to save the changes.

Note



Participant window does not have **References** tab because it is an organizational object that is not referenced by other objects in the organization.

Deleting Participants

To delete a participant

1. Open the Organization window.
2. Click the **Participants** category in the left panel.
3. Click the participant you want to delete in the participants list in the right panel.
4. Click **Delete**.

Participant's permission for instance assignment

A participant can be configured to be able to assign an instance in an interactive activity to another participant in the same role.

Assigning an instance is only available for a participant if the interactive activity is defined as **Assignable**, and depending on the participant's permissions.

Participant category in the role

Participants are assigned to a role with a category. This category, represents the hierarchical level of the participant within the role. Possible category values are from 0-9. The higher the category, the


higher the hierarchical level of the participant in the role.

The category is used to determine to which participant the instance can be assigned to. The category is assigned in the Web Console.

[Participants](#) > [Edit Participant jsmith](#) > [Assigned roles](#) > **Role Assignment**

Properties	
Role Id	Account Manager
Parameter	No Apply
Category	2
Permissions	<input checked="" type="checkbox"/> eXecute <input checked="" type="checkbox"/> Route <input checked="" type="checkbox"/> Select <input checked="" type="checkbox"/> Abort <input checked="" type="checkbox"/> Delegate <input checked="" type="checkbox"/> Grab <input checked="" type="checkbox"/> Escalate <input checked="" type="checkbox"/> Peer Assignment
<input type="button" value="Save"/> <input type="button" value="Cancel"/> <input type="button" value="Reset"/>	

Instance assignment permissions (Delegation, Escalation & Peer Assignment)

The participant will be able to assign an instance, only if he has first selected it to himself. If after that he does not see the  button in the instance line, then, he has not enough permissions to assign an instance to another participant.

1. **Delegate:** this permission enables the participant to assign instances to participants with a lower category in the Role.
2. **Escalate:** this permission enables the participant to assign instances to participants with a higher category in the Role.
3. **Peer Assignment:** this permission enables the participant to assign instances to participants with the same category in the Role.

Grab permission

Grab permission enables the participant to assign an instance to any participant in the role no matter which category the participants have or if the instance is already assigned to a participant.

If the instance is already assigned to one participant, and another participant that has Grab permissions wants to reassign it, then this operation can be done by performing a search to find the instance and later proceed to reassign it.

Examples of Assigning Instances permission

Having the role, participants, categories and assigned permissions shown in the table below.

Role : Account Manager

Participant Name	Category	Assigned Permissions
John Smith	3	Grab
Peter Drayfus	2	Peer Assignment, Delegate
Tom Ryan	2	Peer Assignment
Dan Austin	1	Peer Assignment , Escalate

Case 1: Grab permission

John Smith is the participant with higher hierarchy level. Remember that the higher the category, the higher the hierarchical level. This participant has grab permissions. As shown in the image below, he can assign the instance to any participant in the role, without the need of selecting the instance to himself.

FUEGO Work Portal Welcome, John Smith Search - Options - Help - Logout

Inbox First Showing 1-4 of 4 Last

✓	Description	Activity	Priority	State	Received	Deadline	Participant	Order Amount
<input type="checkbox"/>	Diving Supply OrderFill6	Review Order	Normal	Activity completed	Nov 2 02:41:31 PM			80.00
<input type="checkbox"/>	Diving Supply OrderFill7	Review Order	Normal	Activity completed	Nov 2 03:49:04 PM			20.00
<input type="checkbox"/>	Flipper Scuba OrderFill8	Review Order	Normal	Activity completed	Nov 2 03:49:26 PM			9,000
<input type="checkbox"/>	Scubapro Dive Shops OrderFill9	Review Order	Normal	Running	Nov 2 03:49:43 PM			80.00

Fuego™ - Work Portal

Assigned button enabled, without having the instance selected to the participant. Grab Permission.


The list of participants to whom J.Smith can assign the instance are all the participants of the role. As he has the higher hierarchy in the role, all the listed participants have an arrow down ▼ beside their name.

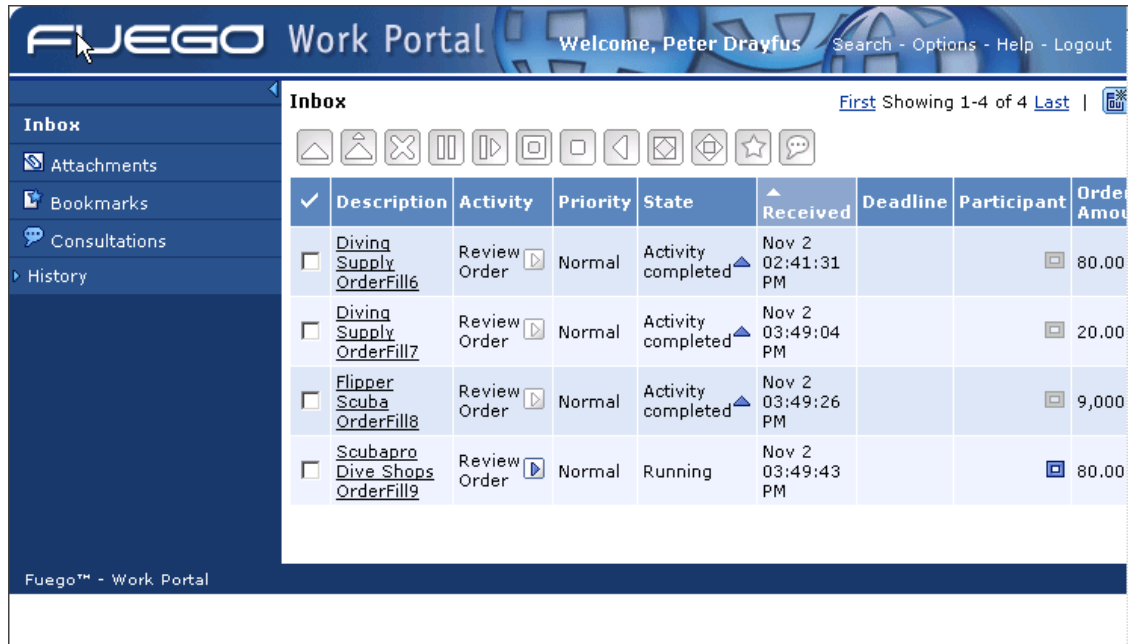
Select a Participant - Microsoft Internet Explorer

Name	Id	E-mail
▼ Dan Austin	daustin	
▼ Peter Drayfus	pdrayfus	
▼ Tom Ryan	tryan	

Cancel

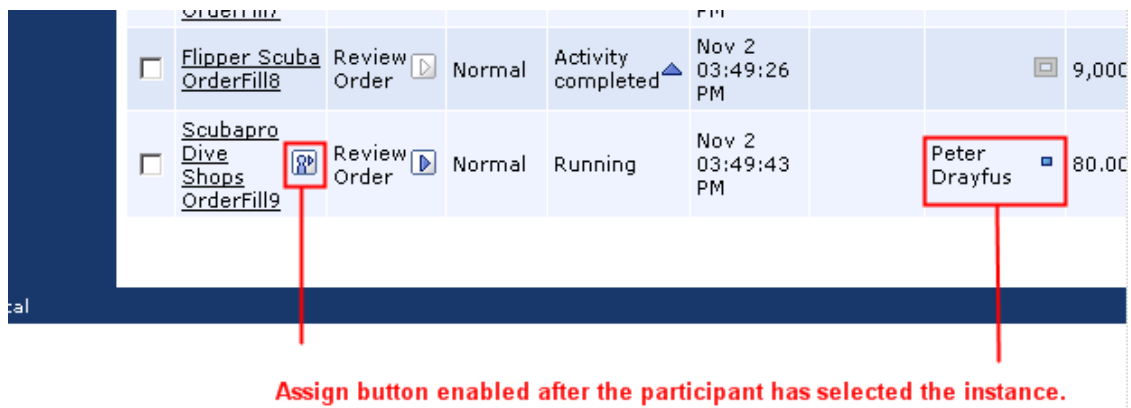
Case 2: Peer Assignment / Delegate

The participant Peter Drayfus has no grab permissions for assigning instances. That is why when he executes his Work Portal, the **Assign Participant** button  does not appear in the first column.





✓	Description	Activity	Priority	State	Received	Deadline	Participant	Order Amount
<input type="checkbox"/>	Diving Supply OrderFill6	Review Order	Normal	Activity completed	Nov 2 02:41:31 PM			80.00
<input type="checkbox"/>	Diving Supply OrderFill7	Review Order	Normal	Activity completed	Nov 2 03:49:04 PM			20.00
<input type="checkbox"/>	Flipper Scuba OrderFill8	Review Order	Normal	Activity completed	Nov 2 03:49:26 PM			9,000
<input type="checkbox"/>	Scubapro Dive Shops OrderFill9	Review Order	Normal	Running	Nov 2 03:49:43 PM			80.00

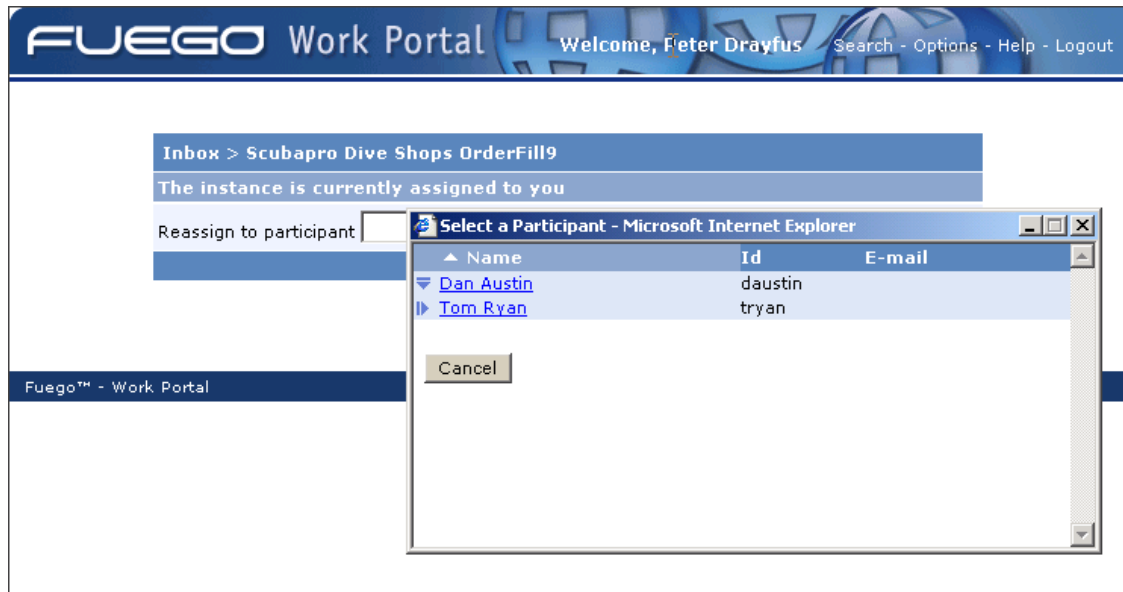
He can assign a participant after selecting the instance to himself.



<input type="checkbox"/>	Flipper Scuba OrderFill8	Review Order	Normal	Activity completed	Nov 2 03:49:26 PM			9,000
<input type="checkbox"/>	Scubapro Dive Shops OrderFill9	Review Order	Normal	Running	Nov 2 03:49:43 PM		Peter Drayfus	80.00

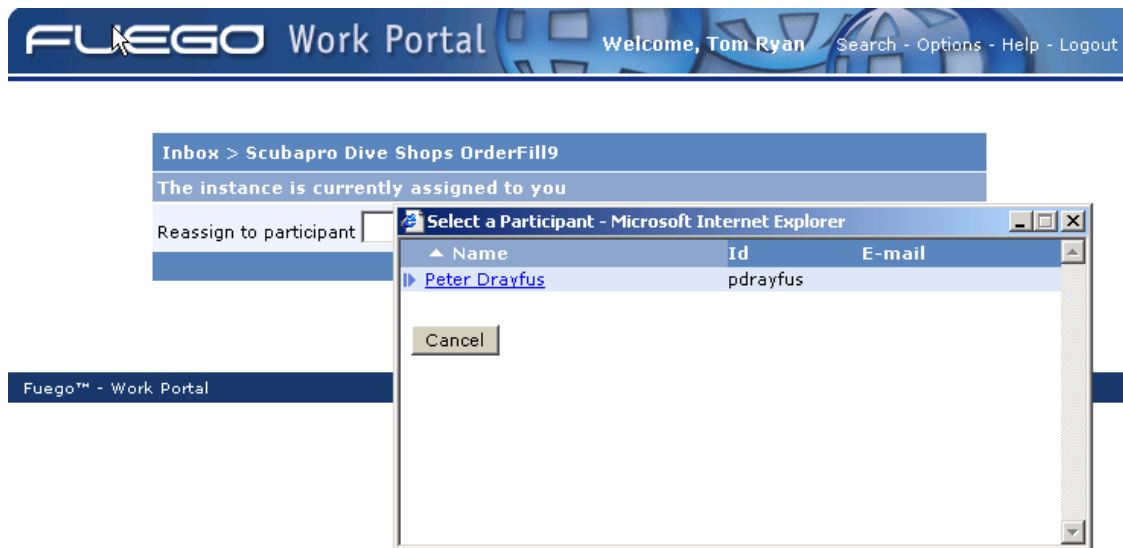
Assign button enabled after the participant has selected the instance.

As he has **Peer Assignment** and **Delegate** permissions, he can only assign the instance to participants with his same category or a lower one. The hierarchy is indicated with an arrow to the right  and an arrow down  beside the respective participants' name.




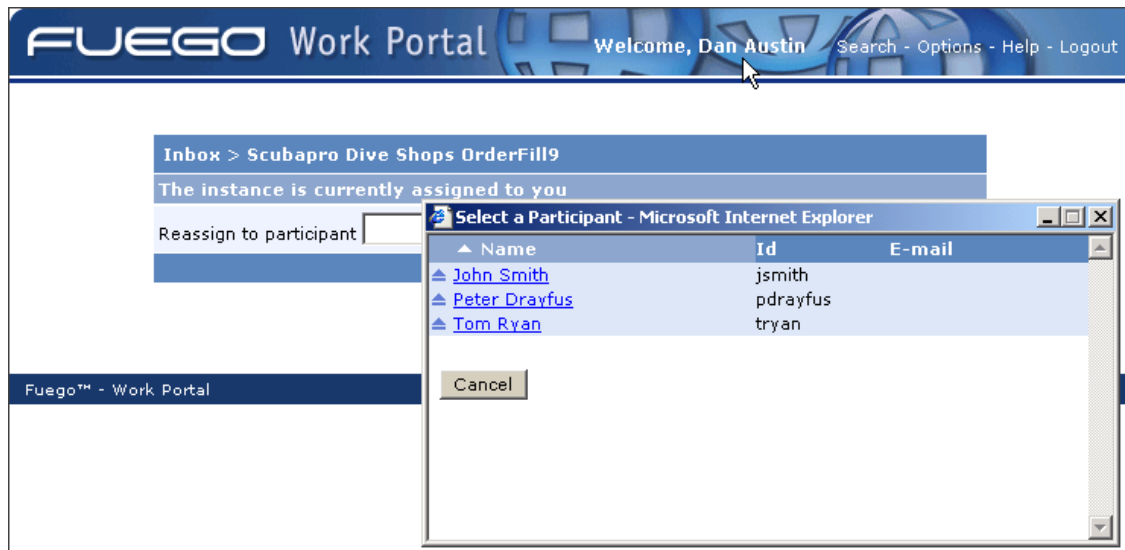
Case 3: Peer Assignment

The participant Tom Ryan has only the **Peer Assignment** permission. That is the reason why, after selecting the instance to himself, he only sees *Peter Drayfus* as the possible participant to select, who is the only one with his same category, 2. The hierarchy is indicated with an arrow to the right ▶ beside the participant name.




Case 4: Peer Assignment / Escalate

The participant Dan Austin has the **Peer Assignment** and **Escalate** permissions. That is the reason why, after selecting the instance to himself, he sees all the other participants, who, in this example have higher categories. The hierarchy is indicated with an arrow up  beside all participants' name.



Note

 All changes on assigned **Roles** and **Permissions** (add, delete or updates) are effective once the participant connected to the Work Portal, logs out and logs in again.

Holiday Rules

Holiday rules are composed of a set of non-working days. Holidays rules must be associated with calendar rules.

FuegoBPM Enterprise Server takes note of the holidays defined in the holiday rule when calculating activity deadlines. It considers them as exceptions to the normal calendar rules on certain days of the year.

Note



Holiday rules apply only if they are associated to a calendar rule defined for the organization or organizational unit.

You can define two different types of holidays:

- Fixed - dates for a fixed year (yyyy/mm/dd)
- Common - dates that are always non-working days (mm/dd) irrespective of the year

Fixed holiday rules apply to those holidays that vary from year to year. For example, Easter varies every year so you need to define a fixed holiday each year for Easter.

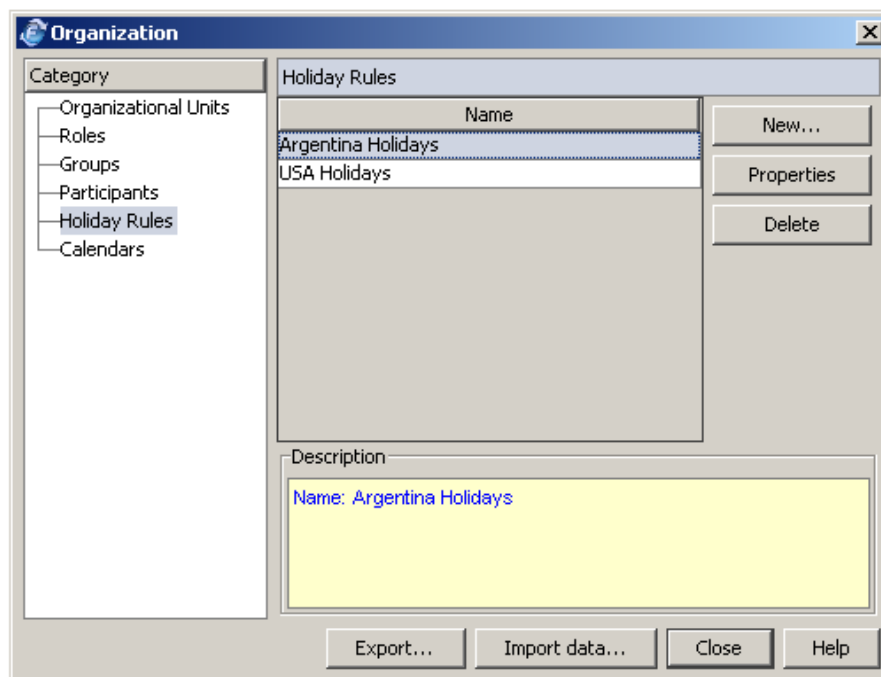
A **common** holiday is typically a holiday that occurs every year on the same day of the month. New Year's day is an example of a common holiday.

The **Holiday Rules** view in the **Organization** window allows you to create, modify, and delete holiday rules for the organization. Any changes made to the **Organization** window while the Server is running require you to **Refresh Server Data**. This option is located on FuegoBPM Studio's **Run** menu. It makes the changes available when executing processes. Refer to Refreshing Server Data for further information on this topic.

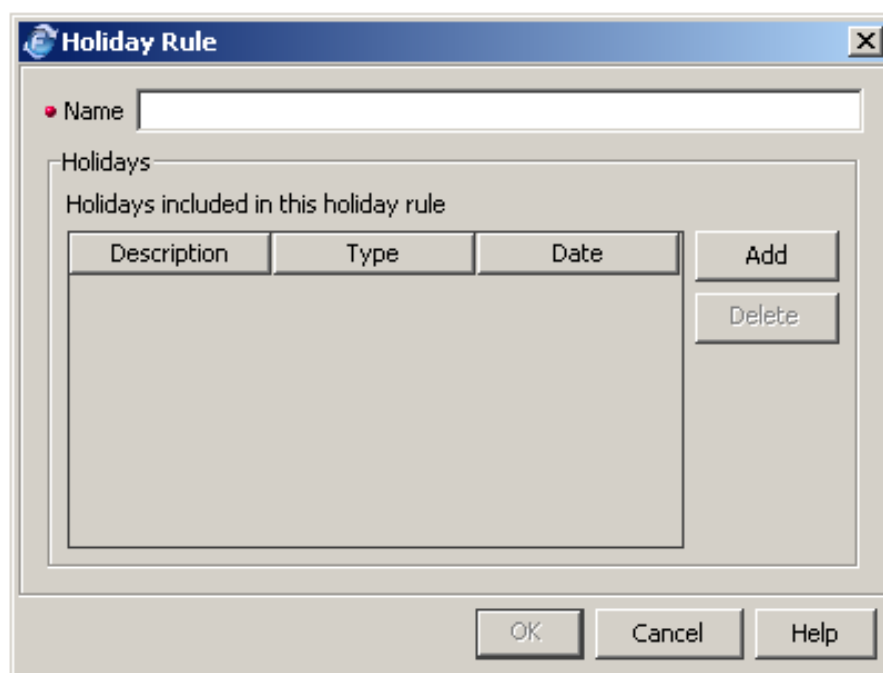
Creating Holiday Rules

To add a Holiday Rule

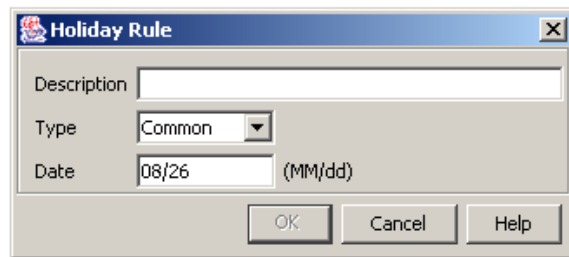
1. Open the **Organization** window.
2. Select **Holidays** category on the categories list in the left panel.



3. Click the **New** button. A blank Holiday Rule profile displays in a new window.



4. Type a Holiday Rule name.
5. Click the **Add** button next to **Holidays** table to add a new non-working day.



The image shows a 'Holiday Rule' dialog box. It has a title bar with a small icon and the text 'Holiday Rule'. Inside, there is a 'Description' label followed by a text input field. Below that is a 'Type' label followed by a dropdown menu currently showing 'Common'. Underneath is a 'Date' label followed by a text input field containing '08/26' and a label '(MM/dd)' to its right. At the bottom of the dialog are three buttons: 'OK', 'Cancel', and 'Help'.

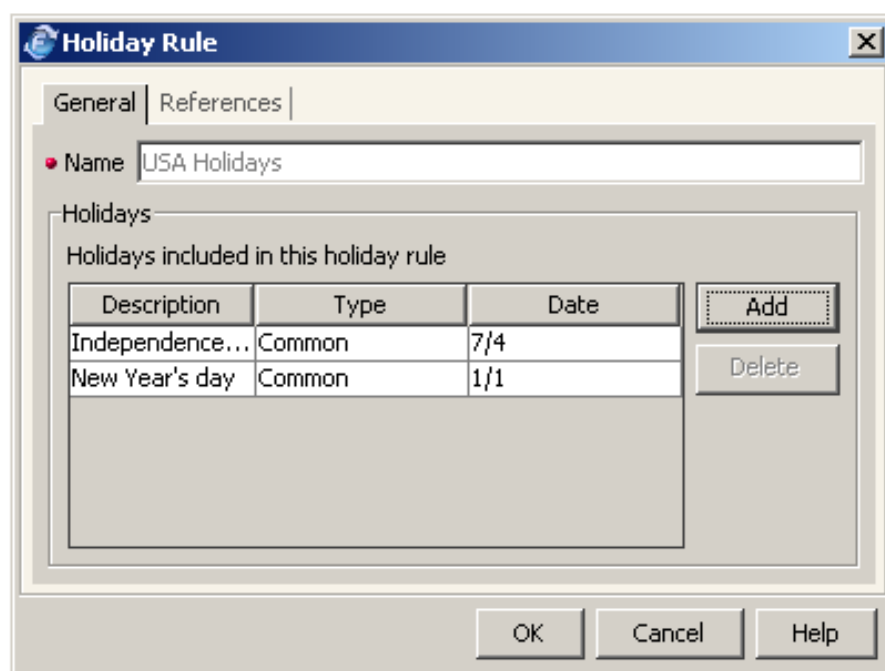
6. Enter the holiday's Description.
7. Choose one of available holiday types: **Common** or **Fixed**.
8. Enter the date in the suitable format. Then click **Ok**.
9. Once you have entered all the holidays for the rule, click **Save**.

Editing Holiday Rules

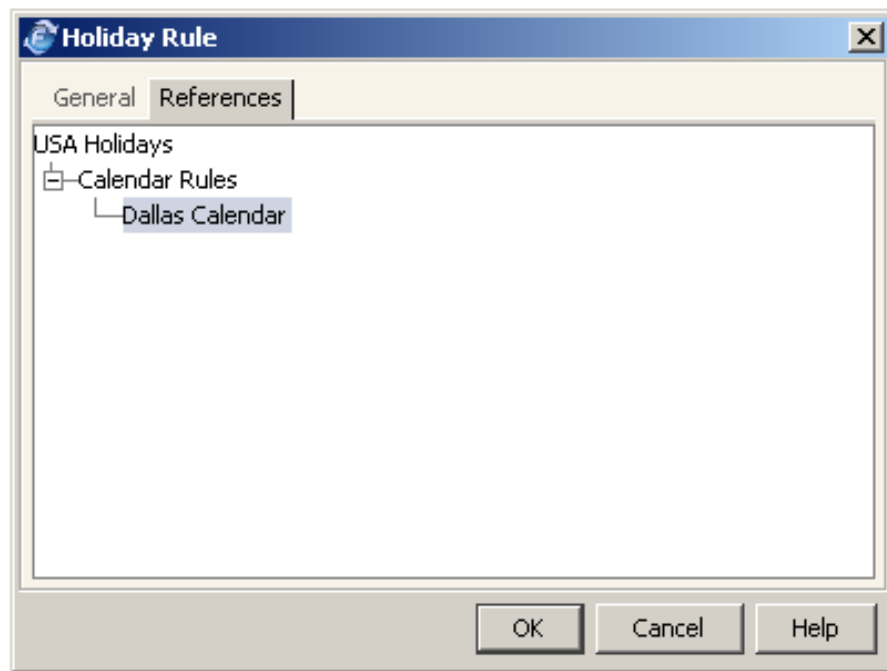
To edit a Holiday Rule

1. Open the Organization window.
2. Click on **Holiday Rules** in the left panel.
3. Click on the Holiday Rule you want to edit in the holiday rules list displayed in the right panel.
4. Click the **Properties** button.
5. Change settings at will using the **General** tab. You can create new holidays or remove existing ones by clicking on the **Add** and

Delete buttons.



6. Click **Save** to save the changes.
7. If you want to check how other objects use the holiday rule being edited, click the **References** tab. The references are displayed.




Deleting Holiday Rules

To delete a Holiday Rule

1. Open the Organization window.
2. Click on the Holiday Rules category in the left panel.
3. Click on the Holiday Rule you want to delete in the holiday rules list displayed in the right panel.
4. Click the **Delete** button.

Note

 Take into account that all the references to the deleted holiday rule will also be removed.

Calendar Rules

It is a common practice to model business processes using deadlines or displaying date and times information. FuegoBPM Studio allows you to define calendar rules. Calendar rules are assigned with a time zone, a holiday rule, and a work schedule.

FuegoBPM Enterprise Server calculates deadlines by making use of calendar rules, if any have been defined.

Once you have defined all the calendar rules needed in the organization, you can assign each Organizational Unit a different calendar rule.

When the Server finds it necessary to calculate an activity deadline for a process instance, it looks for a calendar rule since it defines the time zone, the working hours and the holidays conforming the exceptions to the normal working days.

Take into account that while working on Organization window updating information, if the Server is running, you must use the **Refresh Server Data** option of FuegoBPM Studio's **Run** menu to make the changes available when executing processes. Refer to Refreshing Server Data detailed documentation for further information on this topic.

How the Server decides which calendar rule to use

When the Server needs to calculate a deadline, it first looks for the organizational unit where the process of the instance being executed has been deployed. Once the Server has the information of the organizational unit, it looks for the calendar rule defined for that organizational unit. If no rule has been assigned to the organizational unit, it looks for the calendar rule defined for the parent organizational unit of the organizational unit where the process is deployed. If no rule has been defined for the parent organizational unit, it keeps looking in the upper levels of the organization hierarchy until it finds an organizational unit with a calendar rule defined. If no calendar rule is found, it simply doesn't use any rule and assumes that all days are working days.

FuegoBPM Enterprise version allows for setting calendar rules at role level. If FuegoBPM Enterprise is installed, the Server will also take into account the calendar rule set for the organizational unit where the process is deployed **and the activity role where the instance is running**. The calendar rule set at role level is first evaluated by the Server and overrides the one defined for the organizational unit, if defined.

How calendar rules are used to calculate dates

This example illustrates how the Server uses calendar rules to calculate deadlines properly:

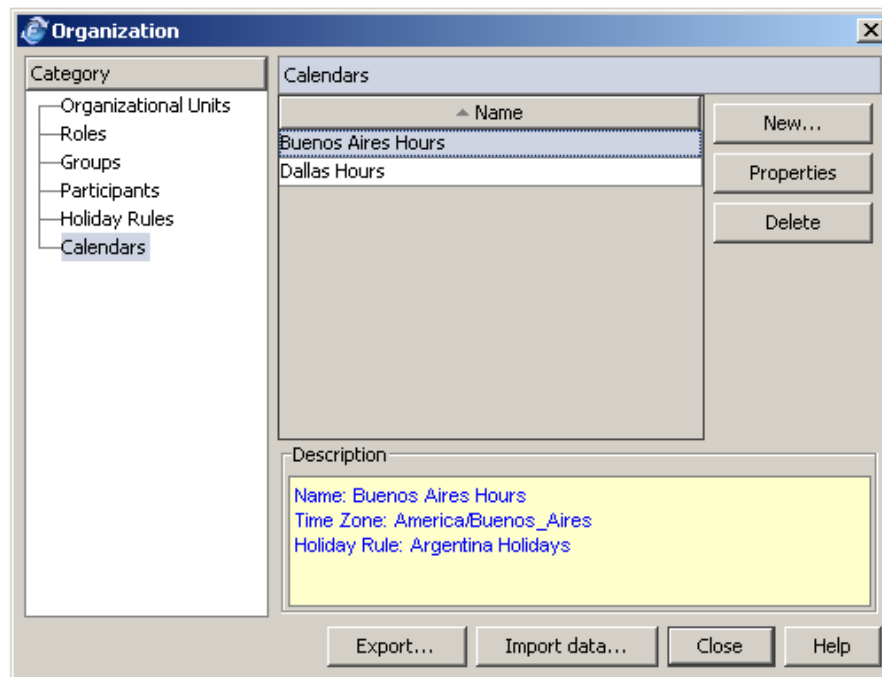
Let's suppose it's Friday 6:00 PM when the due date is calculated. The calendar rule found has set the working hours from Monday to Friday, 9:00 AM to 7:00 PM. The process has a due transition with the interval defined as 2 hours later. Therefore, the deadline for this instance in that activity will be next Monday at 10:00 AM.

Now, let's suppose that, in addition, the calendar rule found has a holiday rule associated setting next Monday as a non working day; then, the deadline would be next Tuesday 10:00 AM instead.

Creating calendar rules

To create a calendar rule

1. Open the Organization window. Select the Calendar Rules category in the left panel, the calendar view listing all the calendar rules defined up to now appears in the right panel.



2. Click on **New** button: A blank **Calendar Rule** profile displays in a new window.

Calendar Rule

Name:

Time Zone:

Holidays:

Working Days

	Starting Time	Finishing Time	Starting Time	Finishing Time
<input type="checkbox"/> Monday	08:00 AM	12:00 PM	<input type="checkbox"/> 01:00 PM	05:00 PM
<input type="checkbox"/> Tuesday	08:00 AM	12:00 PM	<input type="checkbox"/> 01:00 PM	05:00 PM
<input type="checkbox"/> Wednesday	08:00 AM	12:00 PM	<input type="checkbox"/> 01:00 PM	05:00 PM
<input type="checkbox"/> Thursday	08:00 AM	12:00 PM	<input type="checkbox"/> 01:00 PM	05:00 PM
<input type="checkbox"/> Friday	08:00 AM	12:00 PM	<input type="checkbox"/> 01:00 PM	05:00 PM
<input type="checkbox"/> Saturday	08:00 AM	12:00 PM	<input type="checkbox"/> 01:00 PM	05:00 PM
<input type="checkbox"/> Sunday	08:00 AM	12:00 PM	<input type="checkbox"/> 01:00 PM	05:00 PM

OK Cancel Help

- Enter a name for the calendar rule in the **Name** field.
- Select the time zone from the Time Zone drop-down menu. The Server calculates deadlines based on the time zone defined here. Note that a process might be instantiated by users in different locations. Users may be aware that the time zone used to calculate deadlines differ from the time zones they are currently working in. In such cases, deadlines might appear confusing. For example, suppose that a process is deployed in Spain. The process has been designed to set activity expiration one hour after the instance arrives. The calendar rule settings are the following: **Working days** Monday to Friday 9:00 AM to 6:00 PM, **Time Zone** (GMT+1), the **holiday rule** sets January 1 as a non working day. It's December 31 when a user in US creates an instance at 5:00 PM, US time (GMT-3). When the Server receives

the request, it uses the calendar rule to calculate dates. As the difference is 7 hours between these 2 locations, it registers the creation time as 12:00 AM. Following the calendar rule's working days and holiday rule, the Server schedules the deadline for January 2, 10:00 AM Spain time (GMT+1). However, when the end user in USA checks the deadline through Work Portal, he/she will see January 2, 3:00 AM.

5. Select a **Holiday Rule** from the drop-down menu, if any applies.
6. In the **Working Days area** of the window, select the appropriate days of week by clicking in the box to the left of the day and then enter or select the **Starting Time** and **Finishing Time** for each day. If there is a standard work break on any given day, select the corresponding check box and enter the Starting and Finishing Times surrounding the break.
7. Click **Save** to save the new rule.

Editing Calendar Rules

To edit a Calendar Rule

1. Open the Organization window.
2. Click on the Calendars category in the left panel.
3. Click on the Calendar Rule you want to edit in the calendar rules list displayed in the right panel.
4. Click on the **Properties** button.
5. Change the calendar settings at will in the **General** tab.

Calendar Rule

General | References

Name: Dallas Calendar

Time Zone: (GMT-6:00) US/Central

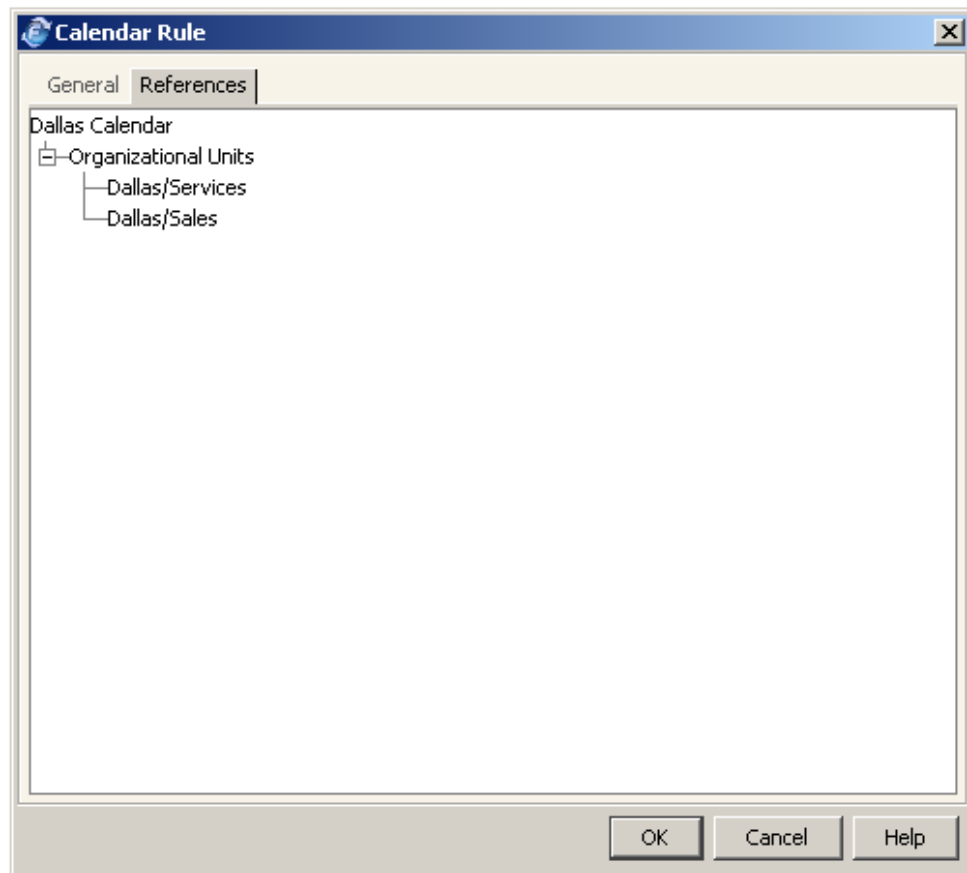
Holidays: USA Holidays

Working Days

	Starting Time	Finishing Time	Starting Time	Finishing Time
<input checked="" type="checkbox"/> Monday	08:00 AM	12:00 PM	<input checked="" type="checkbox"/> 01:00 PM	05:00 PM
<input checked="" type="checkbox"/> Tuesday	08:00 AM	12:00 PM	<input checked="" type="checkbox"/> 01:00 PM	05:00 PM
<input checked="" type="checkbox"/> Wednesday	08:00 AM	12:00 PM	<input checked="" type="checkbox"/> 01:00 PM	05:00 PM
<input checked="" type="checkbox"/> Thursday	08:00 AM	12:00 PM	<input checked="" type="checkbox"/> 01:00 PM	05:00 PM
<input checked="" type="checkbox"/> Friday	08:00 AM	12:00 PM	<input checked="" type="checkbox"/> 01:00 PM	05:00 PM
<input type="checkbox"/> Saturday	08:00 AM	12:00 PM	<input type="checkbox"/> 01:00 PM	05:00 PM
<input type="checkbox"/> Sunday	08:00 AM	12:00 PM	<input type="checkbox"/> 01:00 PM	05:00 PM

OK Cancel Help

6. Click **Save** to save the changes.
7. If you want to check how other objects in the organization use the calendar rule being edited, click **References** tab and the references are displayed.



Deleting Calendar Rules

To delete a CalendarRule

1. Open the Organization window.
2. Click on the Calendars category in the left panel.
3. Click on the Calendar Rule you want to delete in the calendar rules list displayed in the right panel.
4. Click on the **Delete** button.

Note



Take into account that all the references to the deleted calendar rule will also be removed.

Chapter 21. Defining the Server and Runtime Properties

Server

FuegoBPM Enterprise Server orchestrates people, applications, and business partners into executable end-to-end business processes, improving business visibility and reducing business complexity.

The Server actively executes and manages the orchestration of business services according to the rules defined in a published process model. Orchestrations may be performed on services within the company or across the firewall in a B2B environment. Orchestrated processes can also be exposed as Web Services. The Server can execute Web Services transparently across internal and external processes.

FuegoBPM Enterprise Server is responsible for:

- Accepting requests generated from Work Portal
- Executing required tasks
- Maintaining the state of all instances flowing through the deployed processes
- Providing extensive version control that allows process models to be modified, published, and deployed with zero-latency since multiple versions can be run simultaneously

The Server is an essential element of the FuegoBPM Suite. Without the Server, you are not able to:

- Access tasks defined in a process

- Publish, deploy, or activate a process

FuegoBPM Studio automatically creates and configures a Server for each new project. With no configuration required, you can design your processes and immediately put them into work after starting the server. However, some properties can be changed according to your business requirements.

The **Server preferences window** allows you to set and modify a few properties related to the Server that can modify the execution behavior.

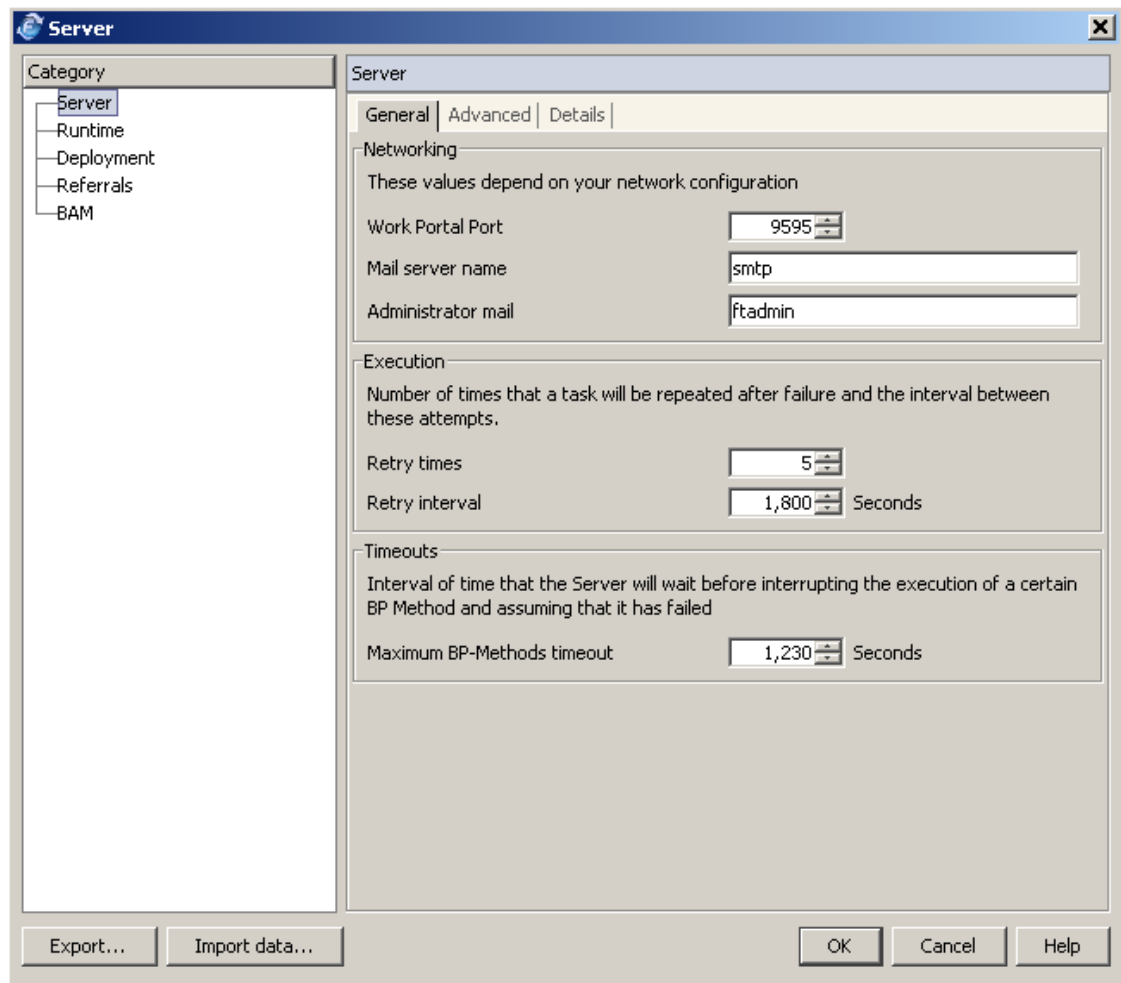
When the Server is started, processes start to be executed. This execution takes place on a runtime environment isolated from changes made in the development environment of FuegoBPM Studio. The **Server preferences** window also allows you to view the deployment settings currently applied to that runtime environment.

Changes made to server preferences through this window while the server is running will be automatically or manually applied to the runtime environment depending on the values set for the **Runtime** preferences.

To view and change Server Preferences

In FuegoBPM Studio, go to **Run** menu, select **Server preferences**. The **Server preferences** window appears:

Defining the Server and Runtime Properties



The following categories are managed by the Server properties window.

Category	Description
Server	Enables you to change some server properties as well as to view the detailed configuration information that FuegoBPM Studio automatically sets.
Runtime	Enables you to change runtime properties.
Deployment	Allows for changing deploy preferences information of all your

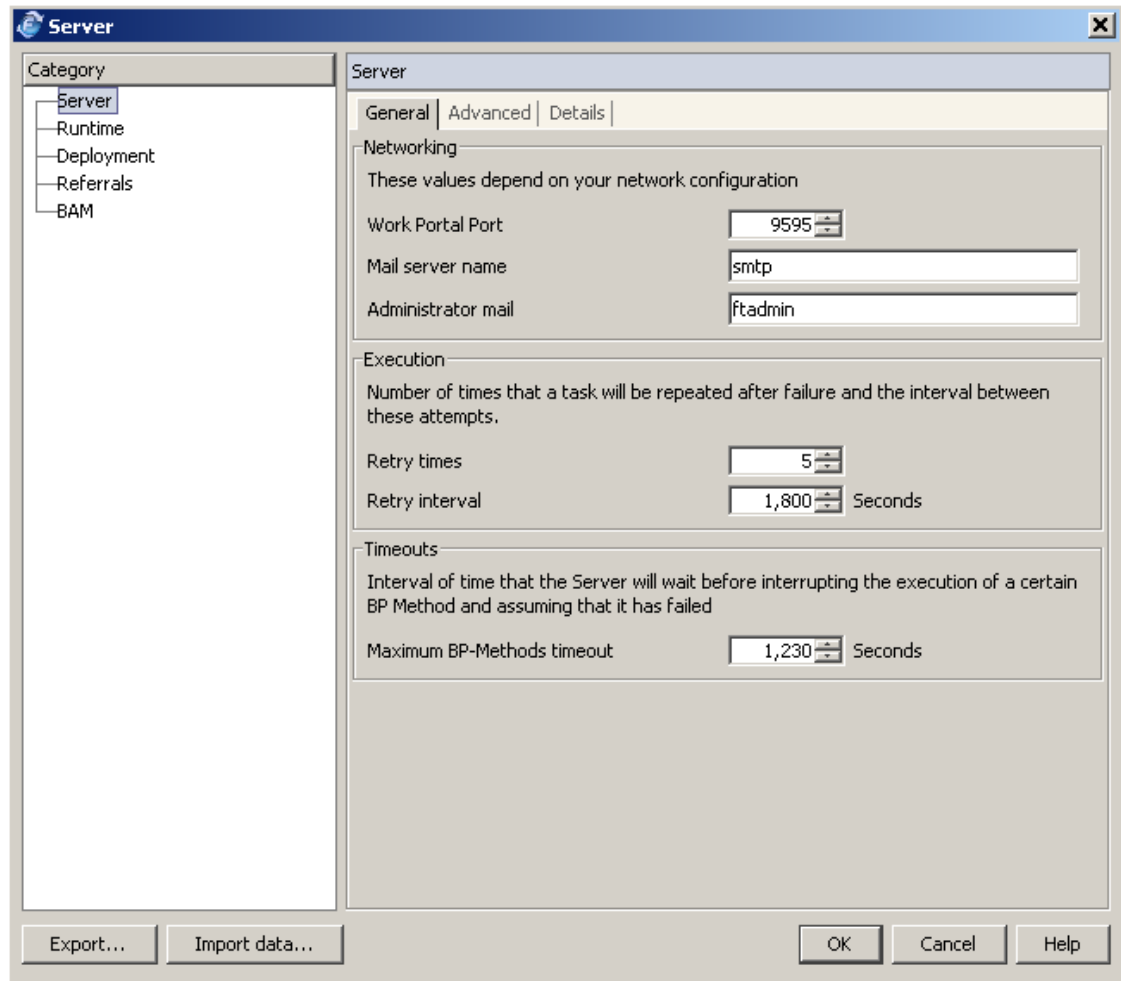
Category	Description
	processes.
Deployed Processes	Only available when the runtime environment is initiated. Displays the information of the currently deployed processes in the runtime environment.
Referrals	Provides the interface to define the information required to enable process to process communication.
BAM	Enables the BAM configuration

Server Properties

Typically, FuegoBPM Enterprise Servers should be created by the system administrator who is familiar with network connections, proxies, ports, and so on.

FuegoBPM Express eases this task by automatically creating and setting up the Server when you create a project.

There are some properties that can be configured to change some server behavior and enhance execution performance. To set these configurable properties, go to **FuegoBPM Studio Run** menu and select **Server preferences** menu item: the Server properties window appears. Choose **Server** category from the list of categories displayed in the left panel. The right panel presents three tabs shown as follows.



The General tab

The **General** tab lets you configure the basic server properties, that is, those properties that are usually modified by the user in order to customize basic options.

All properties have a default value assigned to them. However, you can change some of the following values when needed.

Networking properties

In the following group of general properties, you will be able to configure certain values that depend on the network configuration.

- Select the port that defines the location where Work Portal and Work Portal Views Administrator applications are running.
- Type the **Mail Server Name**. This is the host that the Server will use when sending an e-mail. If your smtp server is configured using simple authentication, you can set this field using the url format user:password@host. If not you can set the server name and it will be enough.
- Type the **Administrator mail name**. This is the Administrator's e-mail address. This address will be used when the Server sends an e-mail as the 'from' field. Any reply to the e-mail that the Server sends will go to this account. This address will also receive any log message sent by mail (if that option is enabled).

Execution properties

- The **Retry Times** indicates the number of times that the Server will retry an automatic operation that is not initially successful.
- The **Retry Interval** sets the interval between each attempt or retry.

Timeout properties

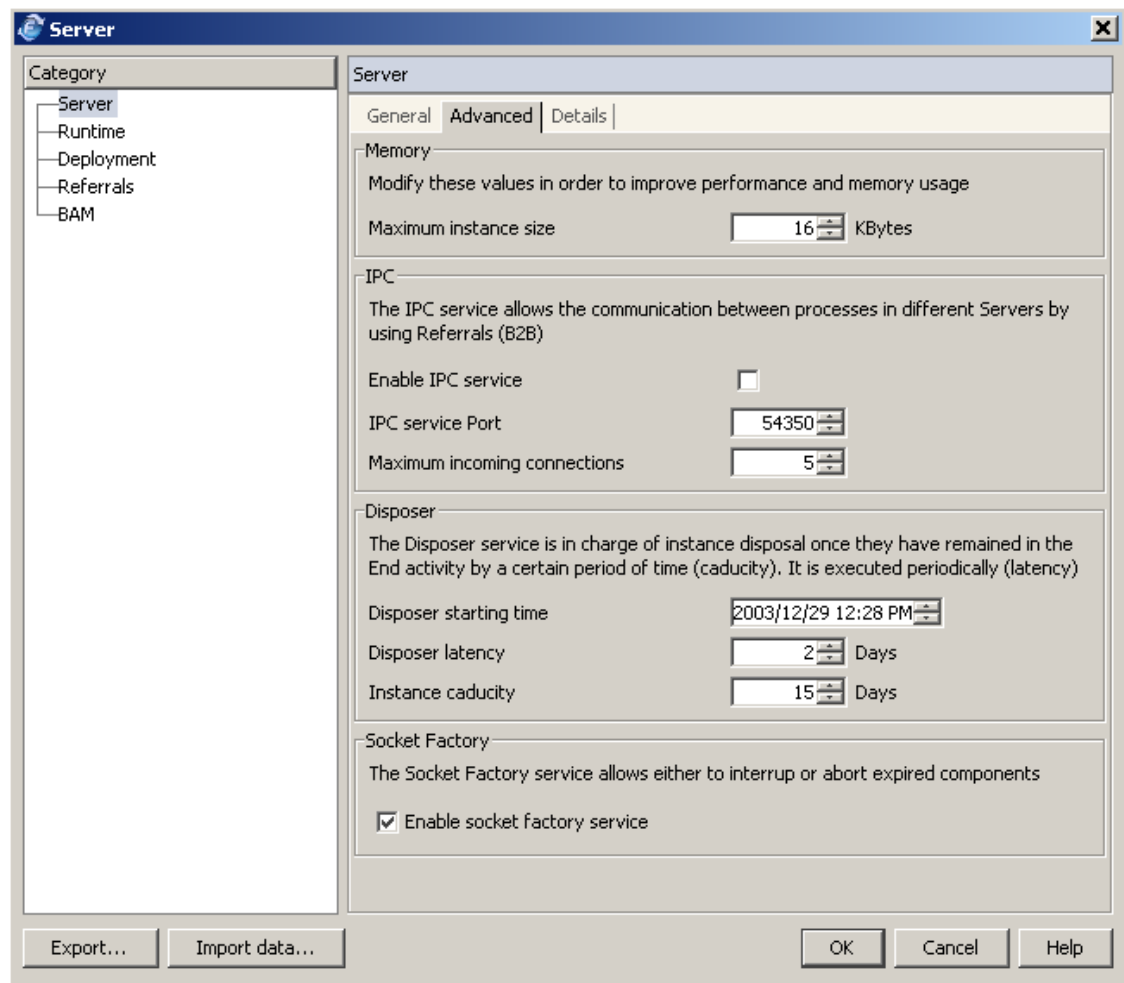
- By setting the **Maximum BP-Methods Timeout**, you limit the maximum time a BP-Method will run before the Server times out; it will interrupt the execution of the BP-Method and assume that it has failed. The default value is set to 1200 seconds (20 minutes). When defining the method, you can re-define the default timeout for that method (by default set to 5 minutes) and this server property is used to state the Maximum timeout for any BP-Method execution of any process within the server. The new set timeout cannot exceed the Maximum BP- Method timeout set in this field.

If you set a greater value a runtime exception is thrown and the BP-Method execution is aborted.

See Execution timeout for detailed information.

The Advanced tab

The **Advanced** tab lets you configure some advanced features.



Memory properties

The first group of advanced properties lets you adjust some values related to the amount of memory that the Server may consume and

its usage in caching instances. Appropriate settings may improve the overall execution performance.

- The **Maximum instance size** is the maximum amount of memory an instance can consume. Attachments and notes added to an instance in Work Portal do not count against this size limit. This value is used to control the size of process instances and avoid date usage of instance variables. In most cases, when this limit is exceeded, it is recommended that you review how process instances are used. In some cases, it makes more sense only to store the key of some information and retrieve the whole content later than having the whole data stored in process instance variables. Another disadvantage to overly large instance variables is that they require a large amount of time and effort when serializing this information into the Server's database.

Inter-process Communication properties

Inter-process Communication (IPC) is the exchange of data between one process and another, either within the same computer or over a network.

- Select the checkbox **Enable IPC Service** to set whether you want to or not to enable the service.
- In the **IPC service Port** field, type the IPC service port number.
- In the **Maximum incoming connections** field, configure the maximum number of external connections that an server will accept at the same time. Any request for connection received after this maximum number will be denied. The default value is 5.

Disposer properties

The Disposer service is in charge of instance disposal. Once an

instance finishes the process execution, it remains in the 'End' activity. After a certain period, they are usually not needed and would just consume space; hence, they can be deleted. This task is performed by the Disposer service. It runs periodically, looking for instances that have remained in the End activity longer than a given number of days and deletes them.

- Specify when you want this service to be executed by selecting the **Disposer starting time**. The date will be used when starting the service for the first time. In following executions, only the time will be taken into account. It is recommended to select a period when the Server is not supposed to be performing other tasks (at night, for instance, in usual business hours.)
- The period between executions of this service is specified by the **Disposer latency** property.
- Finally, you can modify how much time the instances remain in the 'End' activity before being deleted by editing the **Instance caducity** property.

Note



When the Server is running, some of the properties cannot be changed. This is so because the changes on those properties can take effect only if the Server is restarted. You need to stop the Server in order to be able to modify their values. Changes made to properties that allow to be changed when the Server is running are applied to the current execution provided that a **Refresh Server Data** operation is performed.

Socket Factory properties

It is highly recommended to keep "Enable socket factory service" checkbox selected.

FuegoBPM activity's **tasks** have a **timeout**. See Timeout for further information.

When the task is executing, and such timeout expires, the task has to be interrupted by the Server. Furthermore, all the resources that are locked by this task, are released by the Server.

On the other hand, **FuegoBPM socket factory** creates sockets that are set with a **timeout**.

The relation between both of the above features is that when the task is executing, any **socket** created by the components used in such task, is set with the **task timeout**. Consequently, the Server is able to interrupt tasks when required.

If **FuegoBPM socket factory** is not used (checkbox not selected), it may be impossible for the Server to

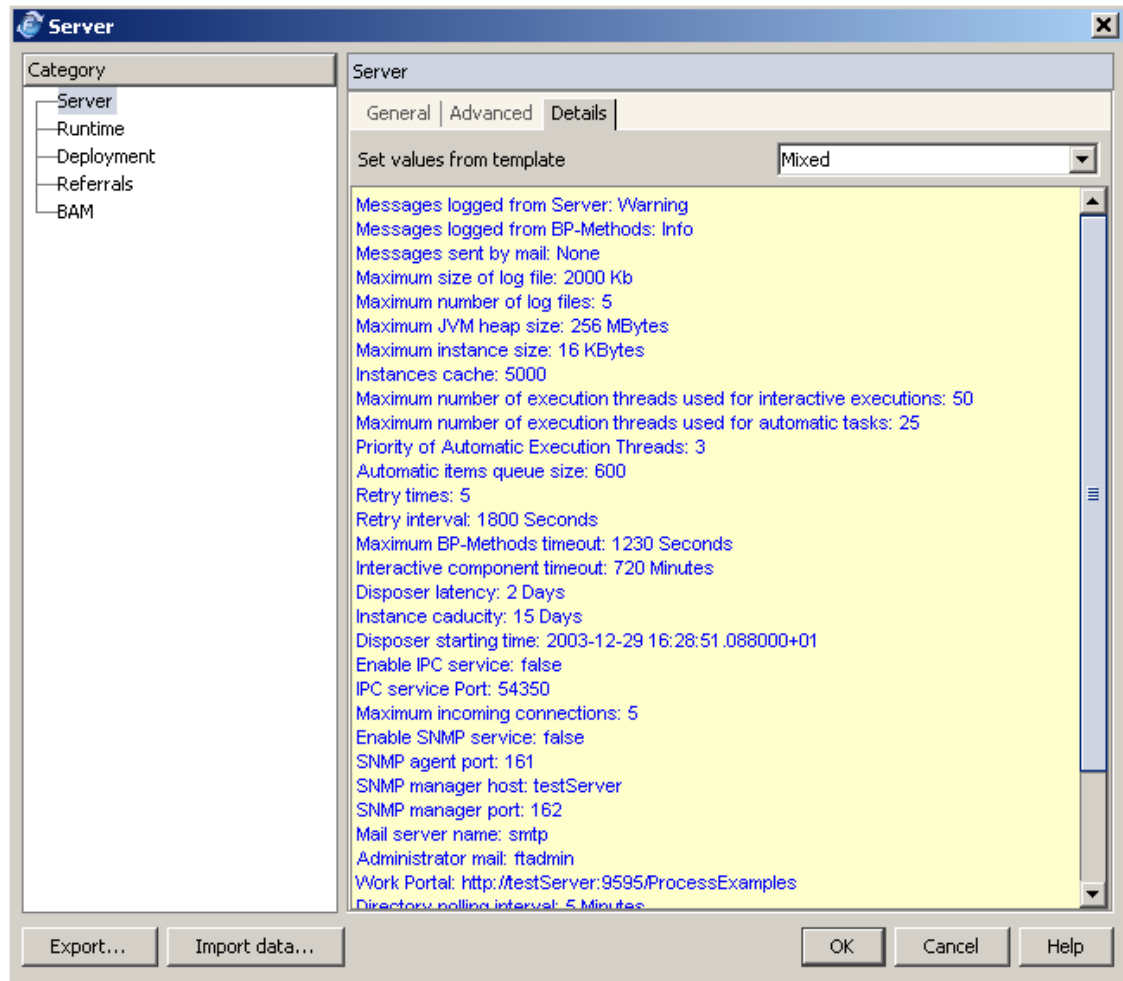
interrupt a task that is using a component that is reading a socket. If so, some specific resources used by such component might be not released.

However, the FuegoBPM Administrator may prefer to use the default Java Socket factory or anyone else that was set to the JVM somehow but the above describes some consequences that might occur.

The Details tab

The **Details** tab displays the complete list of properties settings. The list includes not only all the settings you edited in Basic and Advanced Properties tabs but also properties that cannot be edited individually.

Defining the Server and Runtime Properties

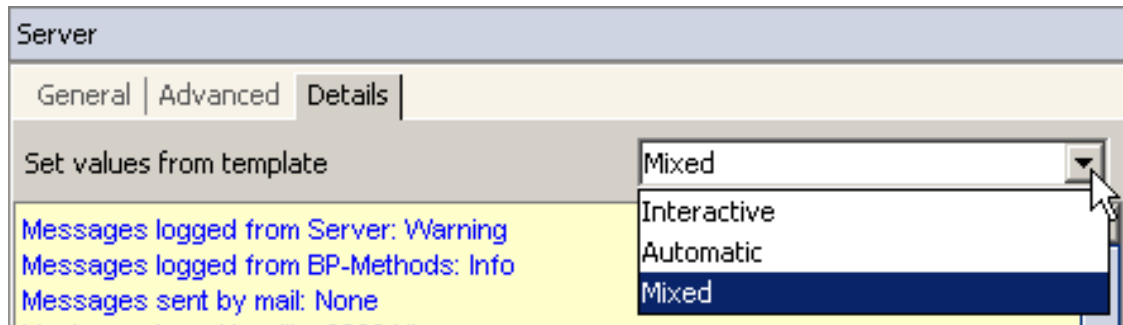


FuegoBPM Enterprise edition allows you to change the values of all settings shown here. In FuegoBPM Express, you can only choose between certain Templates, depending on the kinds of processes that will be executed by the Server.

Currently, you can choose among the following options from the templates drop-down list on the top right of **Details** panel:

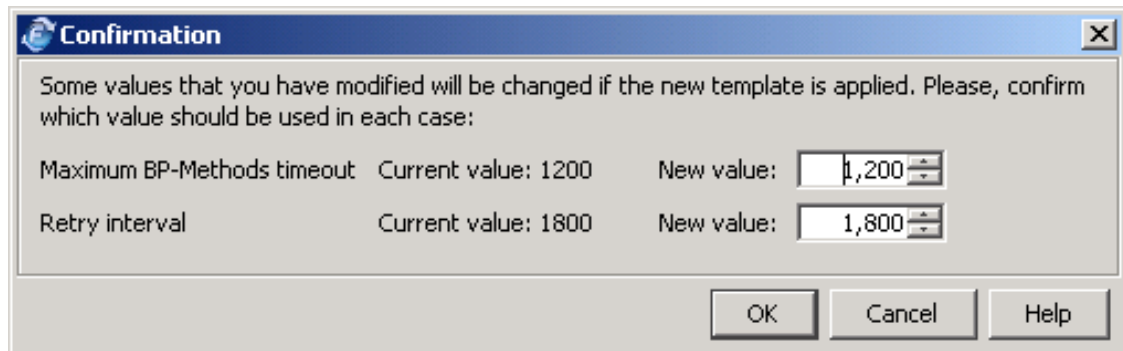
- Interactive: if your project executes interactive processes (default)
- Automatic: if your project executes automatic processes
- Mixed: if your project has a mix of automatic and interactive

processes



By selecting one of the templates, the non-editable properties are set with the optimum values for the kind of project you have. Some editable properties values are suggested by the templates as well: **Maximum BP-Methods timeout** suggested values are 1800 secs for Automatic projects, 600 secs for Interactive, and 1200 secs for Mixed. **Retry Interval** is suggested to be set to 1800 secs for Automatic and interactive projects and 600 secs for Interactive and Mixed projects.

If you have edited any of these properties and after that you decide to apply one of the templates, a confirmation message appears warning you that the template values will be applied unless you enter a different one. Later, you can go to Basic properties panel to edit and choose the appropriate values.



The non-editable properties are the following:

- **Messages logged from Server:** These are the messages that are sent to the log from the Server. The severity is set to *Warning*. Only messages with that severity or higher will appear in the log.
- **Messages logged from BP-Methods:** These are messages that are sent to the log when events are encountered in BP-Methods. The severity is set to *Info*. Only messages with that severity or higher will appear in the log.
- **Messages sent by mail:** this option is set to *None*. No messages are sent by mail in FuegoBPM Studio Server. If a severity was set, only messages with that severity or higher will be emailed.
- **Maximum size of log file:** represents the maximum disk space that will be used for each log file. Log files will grow until they reach this maximum disk space size. It is set to 2000 Kb.
- **Maximum number of logs** indicates the maximum number of log files stored at a given time. The Server log files will move on a round robin basis. Once the last one has been completed, logs will start to be created on the first Server in a circular way. The value set to this property is 5.
- **Maximum JVM heap size** (Mbytes): The default value is 256MB. This is the maximum memory size for the Server. The value associated with this field depends on how many processes are

deployed in this Server and how much memory each process consumes. It's also related to the instance cache and the maximum size of them.

- **Maximum instance size** (Kbytes): The default value is 16KB. This is the maximum amount of memory an instance can consume. Attachments and notes added to an instance in the Work Portal are not included in this size limit. This value is used to control the size of process instances. Instance variables defined at design time are the objects that mainly affect this parameter.
- **Instances cache:** This is the maximum amount of instances to be saved in memory. It is set in 5000.
- **Maximum number of execution threads used for interactive execution:** the maximum number of execution threads available to be used. Execution threads are responsible for the execution of Interactive BP-Methods. Therefore, more interactive BP-Methods can be processed if the pool size is a high number. FuegoBPM Express sets this number to 50 in the Interactive template, 25 in the Automatic, and 50 in the Mixed.
- **Maximum number of execution threads used for automatic tasks:** the maximum number of threads that will process items. These threads are responsible for the execution of automatic BP-Methods (methods that require no end user intervention.) The higher the thread count number, more automatic BP-Methods can be processed. This number is set to 5 in the Interactive template and 25 in both the Automatic and Mixed.

- **Priority of Automatic Execution Threads:** the priority of the thread that processes automatic tasks. It is set to 5 (lowest) in the Interactive template, 1 (highest) in the Automatic, and 3 in the Mixed.
- **Automatic items queue size:** the size of the automatic items queue in memory. It is set to 200 in the Interactive template, 1000 in the Automatic, and 600 in the Mixed.
- **Retry times:** This number indicates the number of times that the Server will retry an automatic operation that is not initially successful.
- **Retry interval:** This is the interval between each attempt or retry (seconds).
- **Maximum BP-Methods timeout** (seconds): this property defines the **limit** of time a BP-Method will run before the task times out. See Execution timeout for detailed information.
- **Interactive component timeout** (minutes): This value is the time limit that the Server will wait for a component that runs on the client to be completed. See Execution timeout for detailed information.

- **Disposer latency:** This field indicates the frequency (number of days) of the disposer execution. For example, if this field is set to 2 days, then every 2 days the server executes the Disposer. The Disposer deletes all the process instances that are suitable for deletion depending on the *Instance caducity* indicated below.
- **Instance caducity:** Indicates how long ago the instance was aborted or completed in order to be deleted when the Disposer is executed.
- **Disposer starting time:** The time of start for the Disposer process.
- **Enable IPC service:** Enables IPC.
- **IPC service Port:** IPC service port number.
- **Maximum incoming connections:** The maximum number of external connections that a FuegoBPM Enterprise Server will accept at the same time. Any connection request received after this maximum number is denied. The default value is 5.
- **Enable SNMP service:** determines if Simple Network Management Protocol (SNMP) service is enabled to allow for server control from a SNMP console. Set to false.

- **SNMP agent port:** Defines the port in which the SNMP agent will be started.
- **SNMP manager host:** Host of the remote console that will receive the SNMP information.
- **SNMP manager port:** Port of the remote console that will receive the SNMP information.
- **Mail server name:** The mail server that the FuegoBPM Enterprise Server will use when sending e-mail.
- **Administrator mail:** The Administrator's e-mail address. Replies to the e-mail that are sent by the Server will go to this account.
- **Web Work Portal URL:** URL address for logging into the Work Portal in the Web Work Portal URL field. This location is where the Work Portal application is running. The URL for the Work Portal is generally *http://host:port/WorkPortal/*. For example, when the Server notifies users who have new instances by e-mail, this URL is part of the message. That way, they can open the application directly.
- **Directory polling interval:** allows you to specify how often the information stored in the directory service will be re-read by the server. Set to 15m in Automatic template, 1m in Interactive

template, and 5m in Mixed.

- **Instance retrieval size:** A number that indicates the maximum instance retrieval size.
- **Notify thread priority:** The priority of the threads that notify the clients about modifications to the Servers. Under normal circumstances, the Server will notify the Work Portal client Server of new states so that the Work Portal can reflect this state when the client refreshes the Portal in their browser.
- **Latency between notifications:** The time that has to pass between notifications to clients. For example, the amount of time that passes between when an end user clicks the **Send** button in the Work Portal and the time when the instance actually appears ready the following activity.
- **Store events:** The frequency the server should store events. You can select it from the pull-down list.

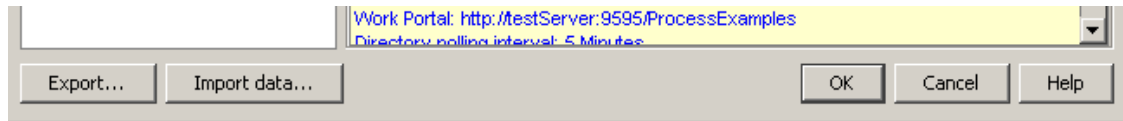
Exporting and Importing the Server Preferences

Export Server Preferences

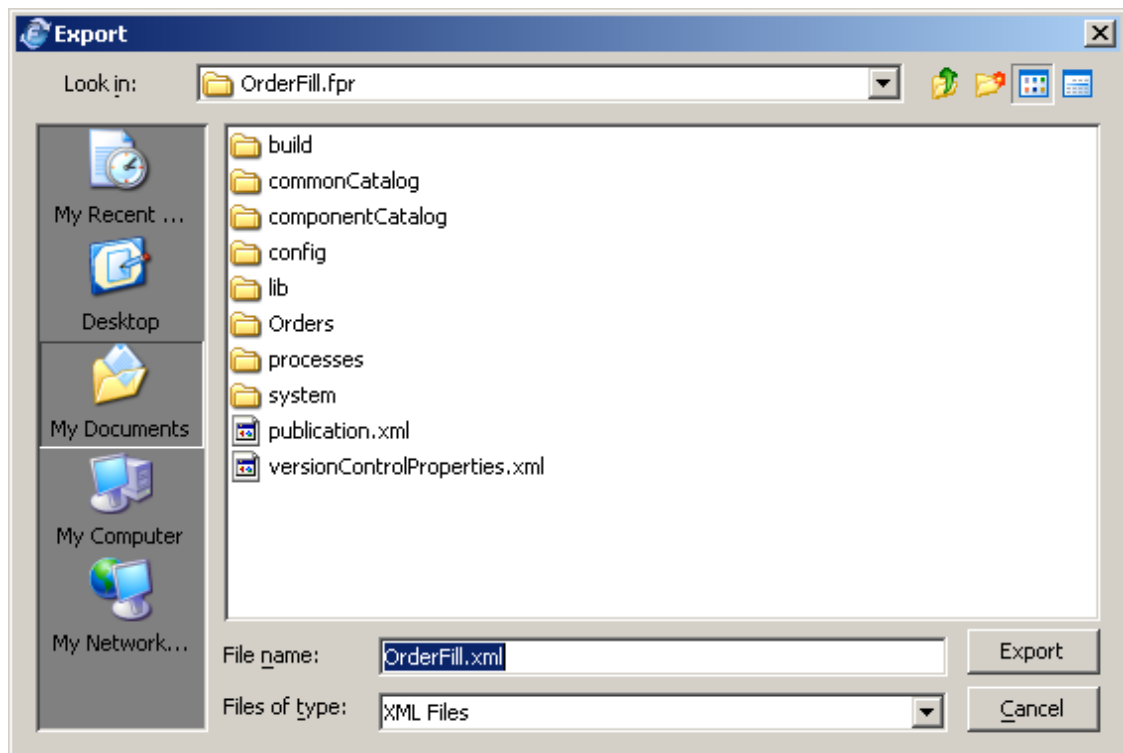
If you need to reuse the Server configuration parameters in any of the projects, you can export the **Server preferences** to import them into another project later.

Select the **Export** button at the bottom of the window

Defining the Server and Runtime Properties



It is recommended that you write the configuration file to the same directory as the project as well as to use the project name as the file name ending with *.xml*. Change the file or directory as desired.



Import Server Preferences

Once you have exported the Server configuration from one project, you can import it to any other project.

First of all, be sure that the Server is not running (if so, the **Import data** option is disabled).

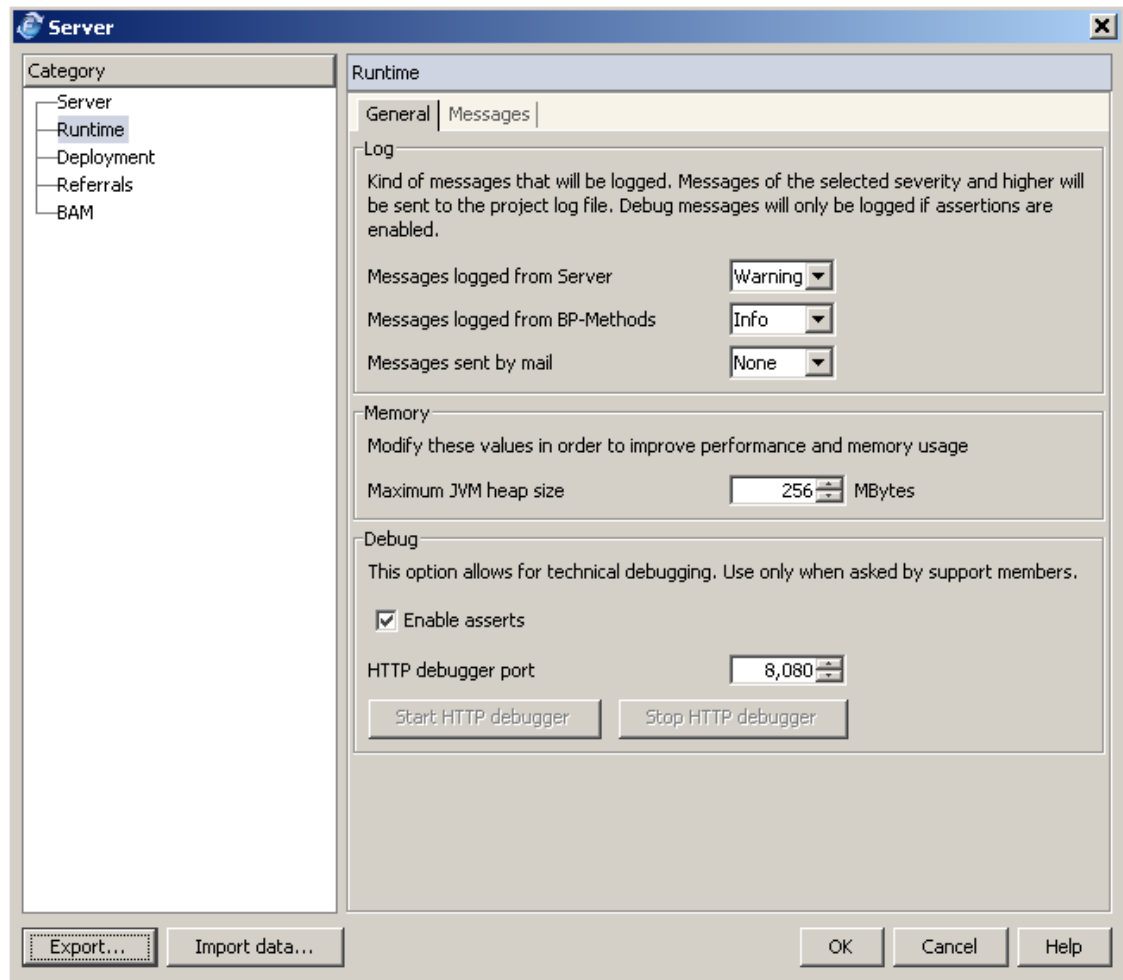
Next, select **Import data** at the bottom of the window. Browse for the exported *.xml* file and proceed to import it.

Runtime Properties

This category allows you to change more properties related to the runtime execution.

To set these configurable properties, go to **FuegoBPM Studio Run** menu and select **Server preferences** menu item: the Server properties window appears. Choose **Runtime** category on the list of categories displayed in the left panel. The right panel presents the following information.

General Preferences



Log properties

The first group of properties refer to logging functions. FuegoBPM Enterprise Server logs all the events that take place during the

execution to a log file. The events logged in that log file are assigned with a severity that goes from Debug to Fatal. You can change the type of information being logged by changing these properties.

- Select an server message severity from the **Messages logged from Server** drop-down list. These are the messages that are sent to the log from the server. The types of messages in severity order from the greatest to the least severe are: Fatal, Severe, Warning, Info, and Debug. This field lets you select the lowest level of message that you want to appear in the log. For example, if you select the severity Warning, then only messages with that severity or higher will appear in the log.
- Select a BP-Methods message severity from the **Messages logged from BP-Methods** drop-down list. These are messages that are sent to the log from BP-Methods.
- Besides, you may indicate if you want to receive some log messages via e-mail. This is configured by selecting one of the options in the **Messages sent by mail** drop-down list. These messages will be sent to the Administrator's e-mail address.

Memory

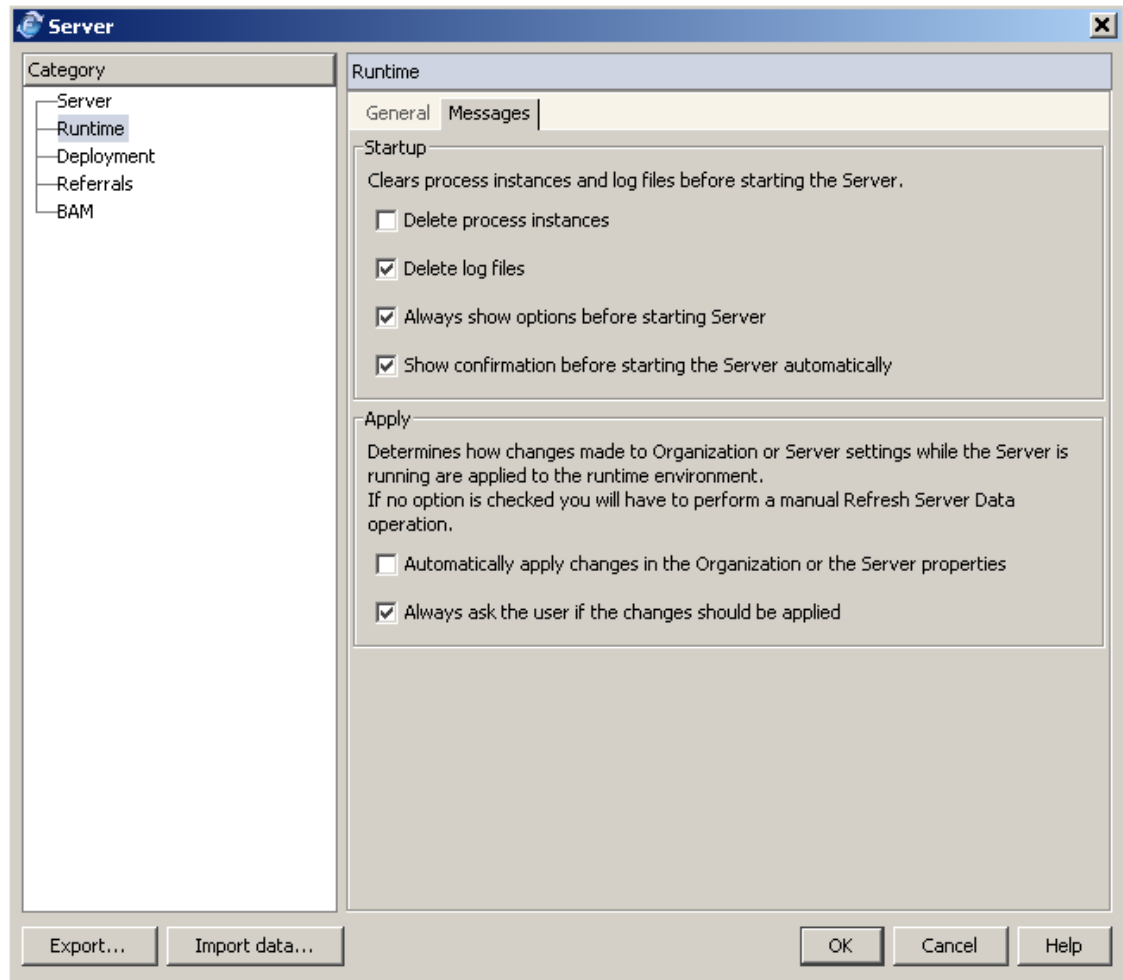
Setting this property lets you adjust the amount of memory that the Server may consume and its usage in caching instances. Appropriate settings may improve the overall execution performance.

- The **Maximum JVM heap size** is the maximum amount of memory that the Server may consume as an independent Java process. Of course, it cannot be greater than the amount of physical memory (RAM) of the host in which it is being executed.

Enable asserts property

This option allows technical debugging. It should be enabled only when support members require it. Otherwise, it must remain unchecked.

Messages



Startup

These options are related to the start server operation:

- Delete process instances: Check this option if you want all process instances to be removed every time the server is started.

- Delete log files: Check this option if you want the log files to be cleaned every time the server is started.
- Always show options before starting the Server: check this option to enable a dialog that asks for the values of the above described Startup options in each **Start server** operation.
- Show confirmation before starting the server automatically.

Apply

These settings determine how changes are applied to the Server runtime environment when the server is currently running. Any changes in the Organization or modifications to the Server's properties are applied as follows:

- If you select **Automatically apply changes in the Organization or the Server properties**, then each time you modify any of them, changes are immediately applied and no confirmation is required.
- If you select **Always ask the user if the changes should be applied**, each time a change is performed, you are required to confirm whether the changes should have immediate effect or not.
- If none of the above options is selected, you must apply changes manually by running from the **Run** menu the Refresh Server Data option in order to have the changes refreshed in the runtime environment.

The update process only takes place when the Server is running. Otherwise, changes are automatically applied the next time you start the server.

Note



When the Server is running, some of the properties cannot be changed. This is so because the changes on those properties can take effect only if the Server is restarted. You need to stop the Server to be able to modify their values. Changes made to properties that allow to be changed when the Server is running are applied to the current execution provided that a **Refresh Server Data** operation is performed.

Deployment

Once you have created a project, you can start modeling your business processes for the project.

After finishing the design of the processes, you can put them to start working by using the **Publish & Deploy** option in the **Run** menu.

The **Deployment** category in the **Server preferences** window allows for changing deploy settings of all the processes included in your project.

Each new process added to the project is initially added to the **Organizational Units Deploy Preferences** list to be deployed in the entire Organization. However, you have the possibility of changing these default deploy preferences so that some of the processes are deployed in specific Organizational Units or in several at the same time.

When a **Publish & Deploy** operation is performed, a set of default Work Portal views is generated in order to provide a way of presenting processes information such as instances, attachments and global applications. By changing the Views generation preferences you can quickly customize the way in which the views appear in user's Work Portals.

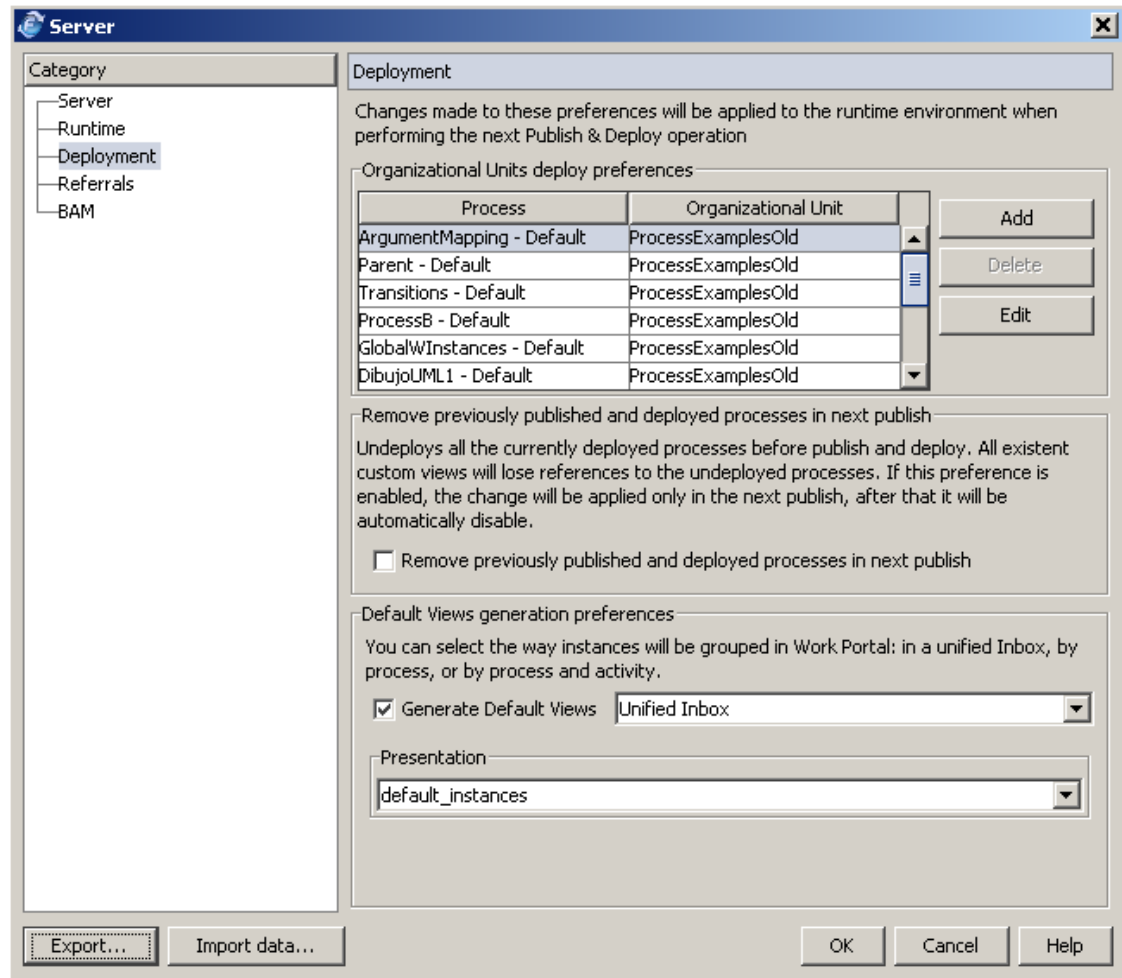
Note



Important Note: All changes introduced through this **Deploy Preferences** panel will not be updated to **FuegoBPM Enterprise Server** until a **Publish & Deploy** is performed.

To view deployment preferences, go to FuegoBPM Studio **Run** menu.

Select **Server preferences** option. The Server preferences window pops up. Then, select **Deployment** category. The **Deployment Preferences** are displayed in the right panel.



Organizational Units deploy preferences

The information displayed includes a list containing one row for each process deployment. Initially, you will see one row for each process. The table includes the following information:

- The process identification. This information cannot be edited.
- The organizational unit where the process is or will be deployed. The default value is the Organization, which means that the

process will be visible in all Organizational Units.

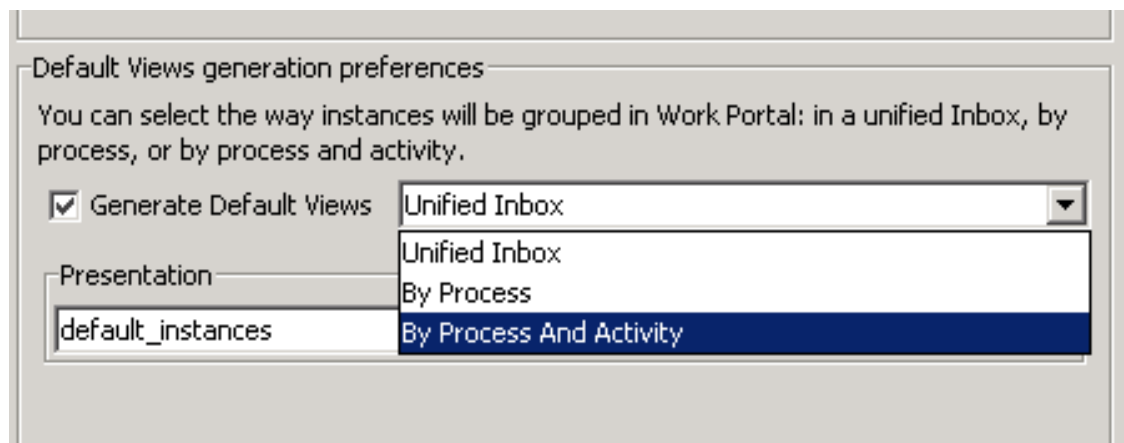
Remove previously published and deployed processes in next publish

This option can be checked to force the undeployment of all previously published and deployed processes. Notice that the undeployment will be performed only for the next publish operation. Once the **Publish & Deploy** has been performed, this preference value is automatically reset to be unchecked. Subsequent publish operations will not remove previous processes unless this checkbox is checked manually before publishing.

Take into account that when this option is checked, all the custom views previously created will lose references to the processes since the processes are recreated.

Default Views generation preferences

At the bottom of the window you must configure whether or not Work Portal default views will be automatically generated for the project and the way in which those views will be visible for Work Portal users.



Views are the format in which instances, applications and attachments appear grouped in Work Portal.

If the checkbox is checked, you have to select one of the following options that determines how views are generated.

The default views will be visible from Work Portal users with the selected format. FuegoBPM Studio automatically creates a set of default views for the project.

Select one of the possibilities according to your needs.

- **Unified Inbox:** If this option is selected, only the following views are generated: A single instances view (Inbox) that will display the complete list of instances of all the processes in the project and a single applications view that displays the applications for all the processes in the project.
- **By process:** It generates an instances view and an applications view for each process of the project.
- **By process and activity:** It generates an instances view for each process and an instances view for each interactive activity within the process. It also generates an applications view for each process.

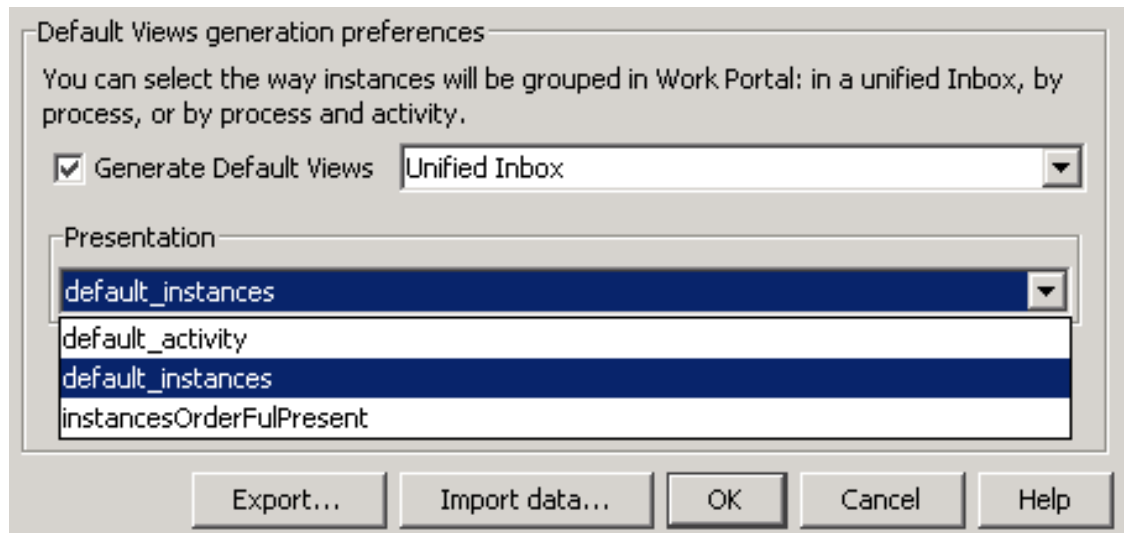
For any of the options selected, a single attachments view that groups the attachments of all the instances of the project is generated.

If the process has grab activities, the user will only see the grab activity view, if views are created *By process and activity*. Otherwise, the user will only be able to grab instances, from the *Search* results panel or if you manually create a custom view for the grab activity of type *Activity Instances*.

If the checkbox is not selected, default views will not be generated. You need to use **Work Portal Views Administration** to create the custom views that better display your business processes information.

Presentation

Select from the combo options the presentation to be used when creating the views. The presentations defined in the project are included in the list.



Deploying processes on Organizational Units

If you want any of the processes to be deployed on a specific Organizational Unit, you must consider the visibility of a process. When a process is deployed in one of the organizational units, it becomes visible for the participants belonging to that organizational unit and for those belonging to any of the organizational units located in lower levels of the same branch within the organization hierarchy.

FuegoBPM Studio validates the fact that a single process is deployed only once in the same branch of the organization tree. This is why a unique version of the process should be visible from any organizational unit. If the processes were deployed in two or more organizational units of the same branch, the Server would not be able to resolve which valid deployment is visible for the lowest level in the organization.

To ensure this, FuegoBPM Studio automatically varies the **Organizational Units** drop-down list according to the currently deploy preferences already set in the **Organizational Units deploy preferences** table.

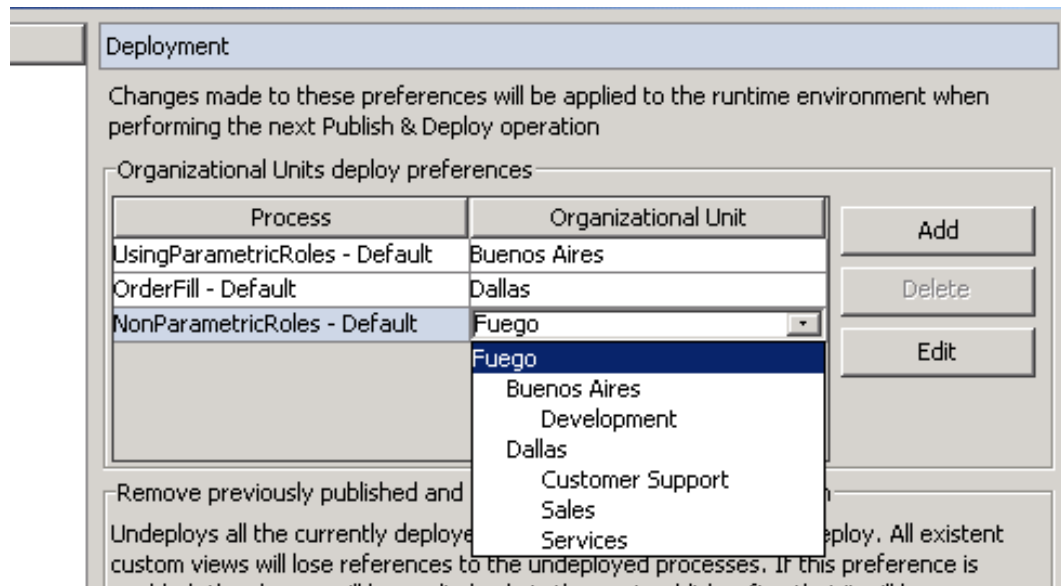
For every new process added to your project, FuegoBPM Studio adds a row in the Organizational Unit deploy preferences table and sets the Organizational Unit with the root Organization as default value.

When a process is deployed in the root Organization, it becomes visible for all the organizational units. If you let the row as it is set by default, you will not be able to deploy the same process in any other organizational unit.

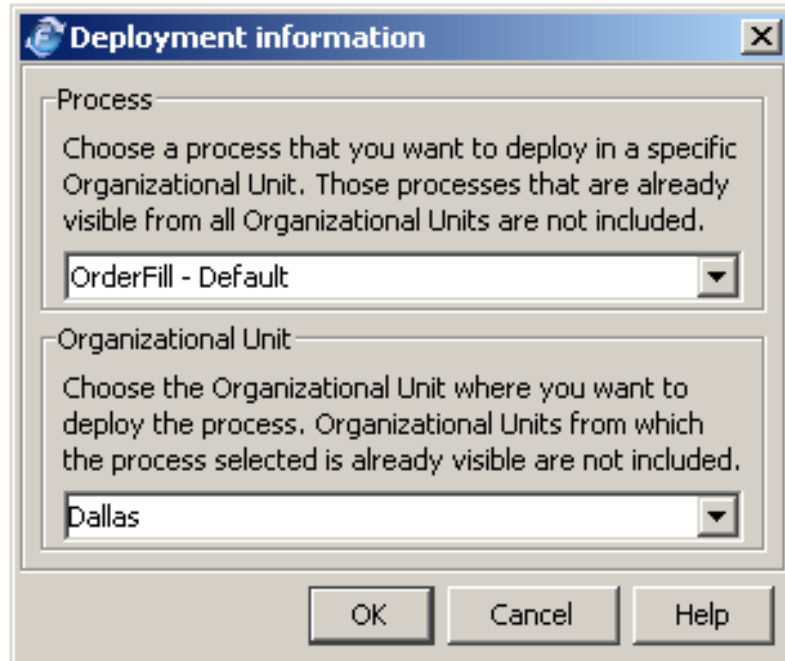
If you decide to deploy a process for one or more Organizational Units, edit first the default row automatically set by FuegoBPM Studio and select the Organizational Unit at will. Then, you can add rows for the other remaining Organizational Units according to the visibility criteria.

To deploy a process in a specific Organizational Unit

1. If the process has been already included in the **Organizational Units deploy preferences** list for a specific Organizational Unit and you want to change it, you can edit the Organizational Unit either by clicking on the **Organizational Units** drop-down list, by double-clicking on the row or by selecting the row and clicking on **Edit** button.

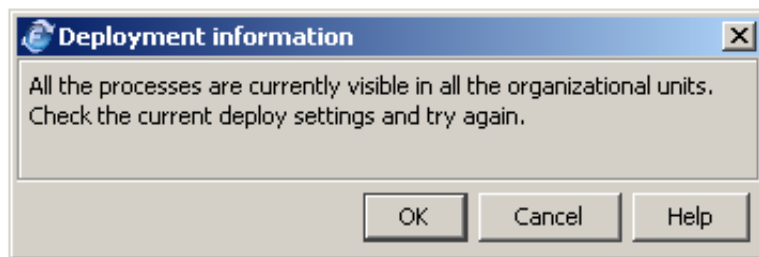


2. If you want to deploy a process in an additional organizational unit, click **Add** button next to the **Deploy preferences** list.



3. Select the process and the Organizational Unit from the drop-down lists. The processes and organizational units displayed in the lists vary depending on the current selection.

For example, if you have one of the processes deployment preferences set to the root organization, the **Add** Deployment information window will not display that process in the processes combo since such process can not be deployed in the root organization and in children organizations at the same time. In the cases that there is no other possibility to deploy any of the processes in your project but the currently selected, the following message notifies you of the situation:




If the process you are looking for does not appear in the processes list, check the current deploy preferences and edit them first, if so needed.

Deleting a process from Organizational Units Deploy preferences

If you decide that one of the processes in the project is no longer needed in the specified Organizational Unit, select it in the **Deploy Preferences** list and click **Delete** button next to the list. The process deployment preference is deleted. In the next **Publish & Deploy** operation, the process will be undeployed for that Organizational Unit.

Every process in the project must always be present in this list at least once. This means that every process will always have a deployment preference set.

Note

 FuegoBPM Studio ensures that every process has at least one deploy preference set: Notice that the **Delete** button changes from enable to disable depending on the process selected. Furthermore, it is enabled provided that more than one deploy preference remain in the selected process. If you want the process not to be deployed at all, you must delete it from the project.

For further details, see Publish and Deploy.

Deployed Processes

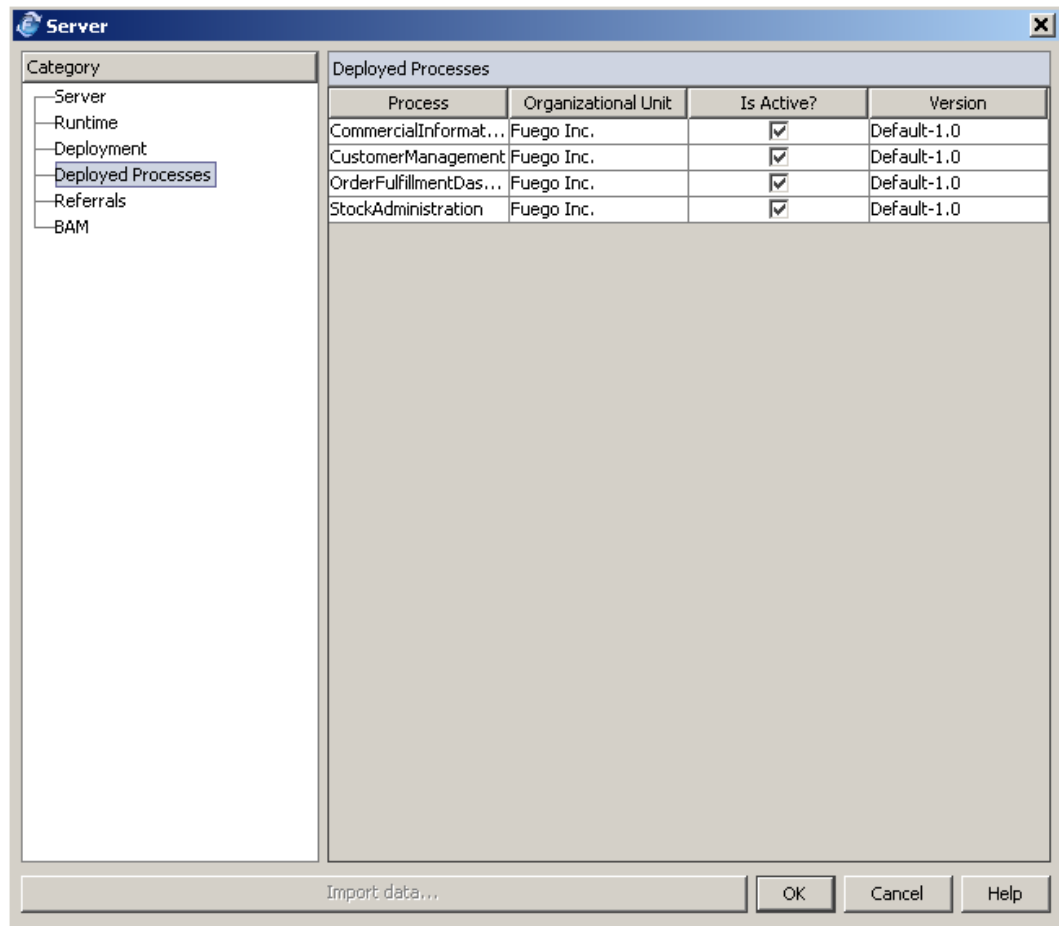
To have a thorough view of the processes that are currently deployed in the runtime environment, **FuegoBPM Studio** provides the **Deployed Processes** category in the **Server** window.

The category is available as long as the runtime environment has been initiated. The runtime environment is initiated either after you perform a **Process & Deploy** operation or you start the server.

The panel displays a row for each different combination of Process/Organizational Unit. It also displays whether the process deployed in the shown Organizational Unit is active or not. If the checkbox **Is Active** is unchecked, it means that the process is currently deprecated. The last column shows the version of the deployed process.

To view the **Deployed Processes** panel:

1. Go to **Run** menu, select **Server Preferences**.
2. Click on **Deployed Processes** category in the left panel. The information is displayed showing a list of all currently deployed processes and their active status.



Referrals

The creation of a referral is a step that must be performed in order to enable the process to process communication.

Process referrals are used by FuegoBPM Enterprise Server when one process needs to call a sub-process that resides in a different server. For example, a warehouse company may have a process that calls a sub-process that resides in a shipping company. Both processes are published and deployed to different directory servers in different networks which are typically behind firewalls. The warehouse process must be able to find a referral to the shipping process to send the instance to the shipping process.

This scenario could also occur where internal processes are published and deployed on different directory servers.

When the Server needs to use Process Referrals

When business analysts design a process that has to communicate to another process, the target process information is included in the model.

The name of the process should always be provided. The Organization might also be present, but this is optional. If no organization is available, the Server assumes that the target process is deployed in the same organization where the process being executed is deployed. The same is valid when the Organizational Unit is not present.

With the above-mentioned information, the Server tries to find the target process. If target process Organization is different from the organization where the process being executed is deployed or when it is the same Organization but the process is not found in any of the organization servers, the Server uses **Process Referrals** to reach the target process.

Creating a Process Referral

You must define **Process Referrals** to provide the Server with the needed connection information that makes the access to a remote process possible .

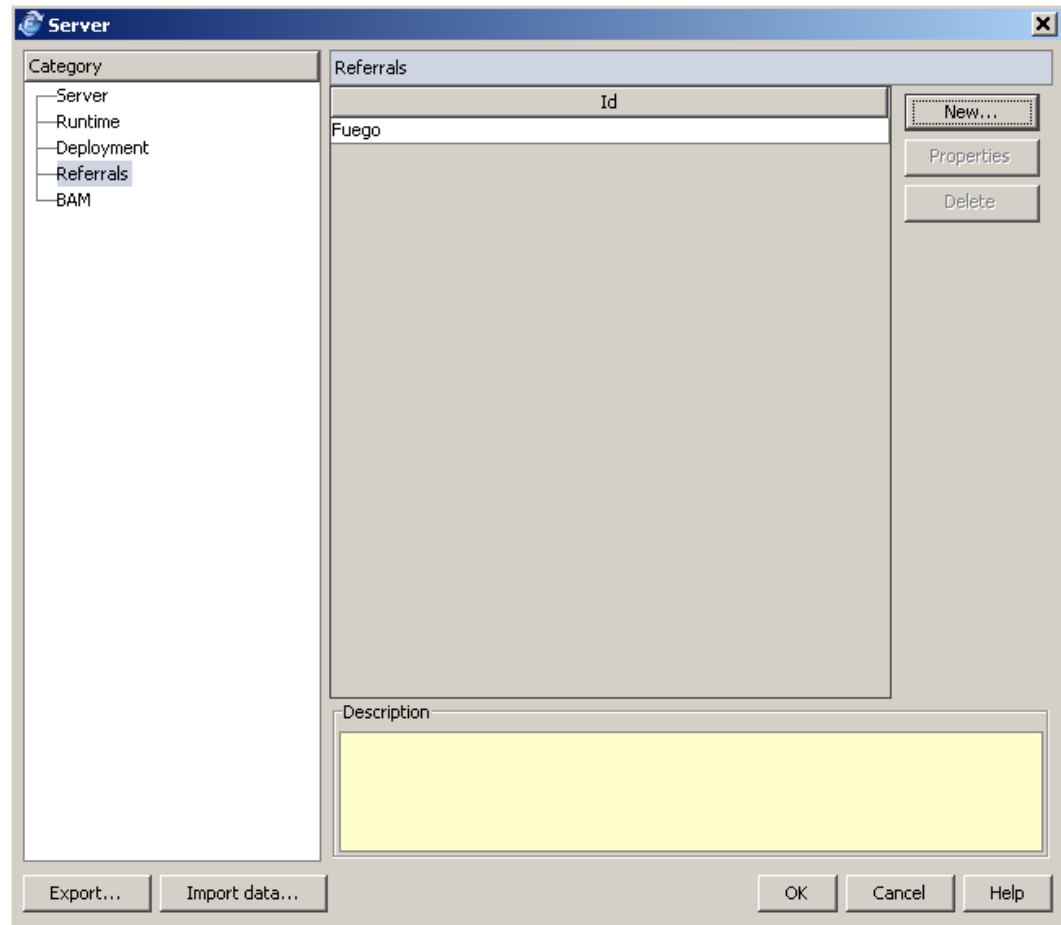
More than one referral can be defined for the same target organization.

FuegoBPM Studio provides the possibility to define a different referral for every single process you need to connect to. However, it is also posible to define a referral that determines one unique address to conect to all the processes deployed in one of the organizational units of the target organization. Furthermore, it is also possible for you to define a referral for the entire target organization.

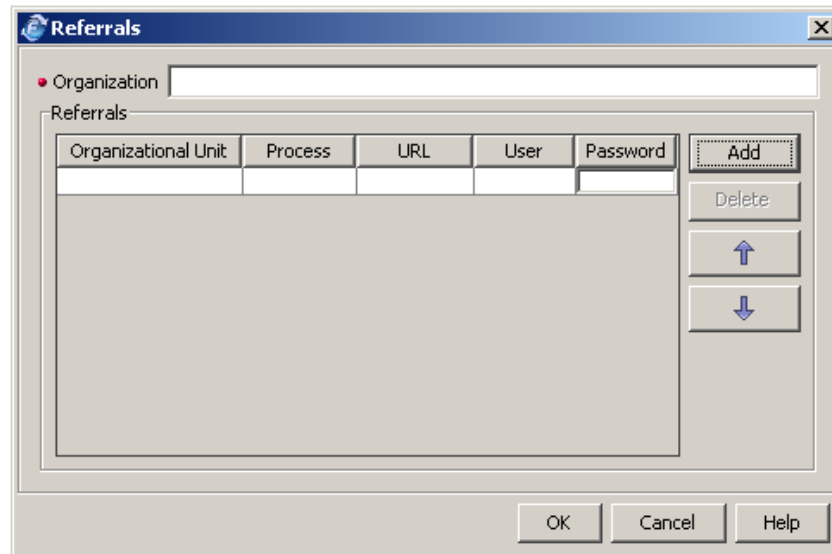
Process Referrals for the same target organization must be defined in descending order from the most specific to the least specific.

To add a process referral

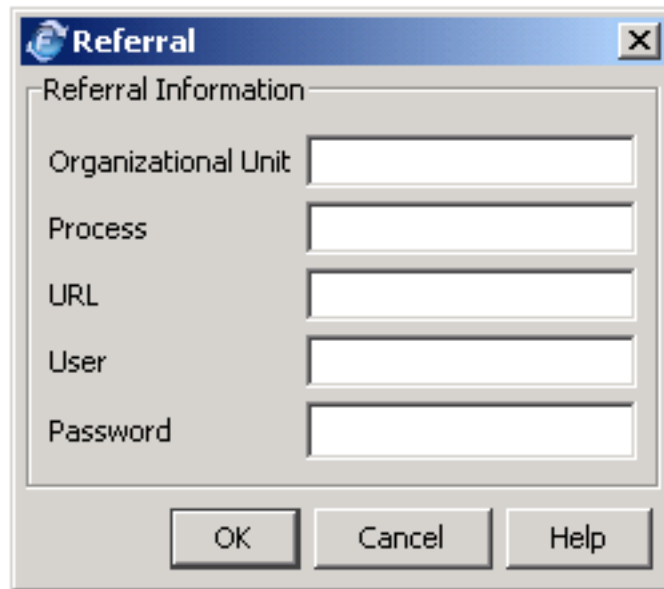
1. Open the Server window by going to **Run** menu, then select **Server**. Select **Referrals** category in the list of categories displayed in the left panel.



2. Click on **New** button next to the **Referrals** list. The **Adding process referral** dialog appears.



3. Enter the name of the referral organization in the **Organization** field.
4. Then, add all the needed referrals. When the Server finds it necessary to communicate with a process deployed in an Organization with this name, it will look for the referrals list set here. Then, for each referral that you want to add in this organization, do the following:
5. Click on **Add** button. The dialog to enter referral information appears:

A screenshot of a Windows-style dialog box titled "Referral". The dialog box has a blue title bar with a close button (X) in the top right corner. Below the title bar, there is a tab labeled "Referral Information". Inside the tab, there are five text input fields, each with a label to its left: "Organizational Unit", "Process", "URL", "User", and "Password". At the bottom of the dialog box, there are three buttons: "OK", "Cancel", and "Help".

6. Enter the following information:

- a. **Organizational Unit** name the referral process has been deployed to. You can leave this field empty, meaning that if you need to communicate with a process in this organization, no matter the OU where it is deployed, the URL defined in the **URL** field will reach the process.
- b. **Process** the name of the process you are referring to. If you leave this field blank, it means that all processes deployed in the **Organizational Unit**, no matter their name, will be found using the URL defined in the **URL** field.
- c. **URL** enter the complete URL address for the Server where the process has been deployed. Include any servlet-mapping information here as well. (Servlet creation and mapping is described in the System Administration Guide.)
- d. **User** enter a user name that the company you are trying to connect to has authorized to be used to connect to the remote web server.

- e. **Password** Enter a password.
7. Once you have finished entering the referrals, click **Save** . The referral is added to the Referrals list.
8. If the remote process must reconnect to your process after it has finished processing, such as through Subflow, Termination Wait, or Process Notification activities, the remote company must also add a referral. Otherwise, only IPC (Inter Process Communication) must be enabled in the remote company.
9. You must now restart both servers so that the referrals are reflected to the servers.

How the Server selects the referral

When the Server finds it necessary to communicate with a process located in an Organization other than the one where the process being executed is deployed, it starts looking for a referral to get the information to establish the connection and reach the process.

The referrals defined in the **Referrals List** for the remote organization are evaluated in descending order until a referral matching the process information is found.

The Server starts evaluating the first referral of the list. The selected referral will be the one that:

- is exactly defined for the Organizational Unit and Process name of the target process, or
- is not defined for a specific process name but the information of the organizational unit matches the ou of the target process, or
- is not defined for a specific process name but it is defined for an

organizational unit in the upper levels of the ou of the target process in the organization hierarchy, or

- the referral does not specify any organizational unit nor process name.

If any of the above-mentioned conditions is true, the Server takes the RUL set for that referral to establish the connection. Otherwise, it keeps on evaluating the next referral item of the list with the same criteria.

Deleting a Referral

To delete a referral

1. Open the Server window by going to **Run** menu, then select **Server**
2. Click on **Referrals** category in the left panel.
3. Click on the referral you want to delete in the referrals list displayed in the right panel.
4. Click on the **Delete** button.

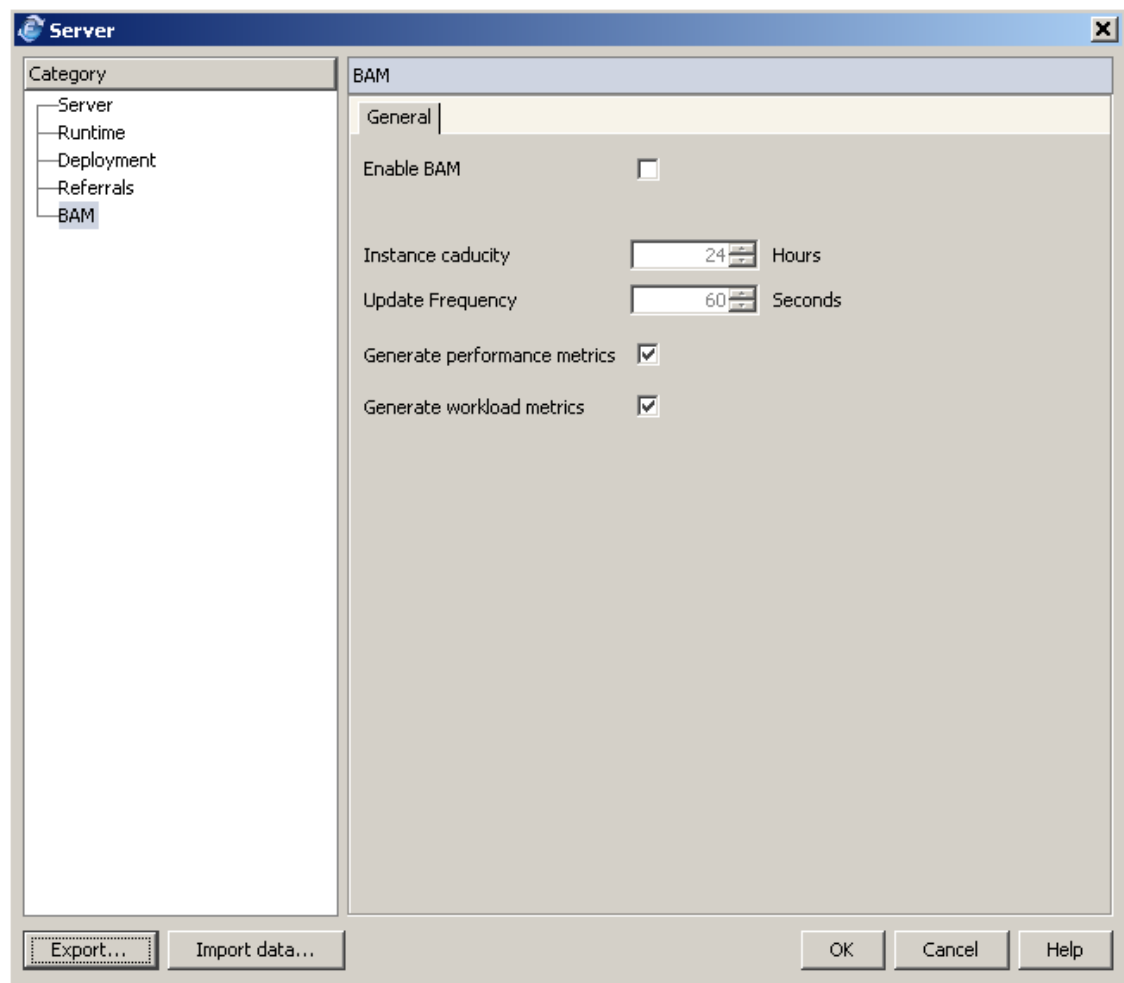
BAM Properties

Enable BAM automatic update by selecting the **Enable BAM** checkbox.

The BAM database is generated in the same embedded Cloudscape than the project server.

BAM Properties

- **Update Frequency** (in minutes): The amount of time allowed to update the BAM data.
- **Instance caducity** (in hours): The period of time that the BAM information will remain active. After that time, the next update process execution will delete it.
- **Generate performance Metrics**: Select this checkbox if you want to store performance data.
- **Generate workload Metrics**: Select this checkbox if you want to store workload data.



Refreshing the Server Data

When the FuegoBPM Enterprise Server is running and you need to introduce changes to the organization definitions or to the server preferences, you must use the **Refresh Server Data** option in order to make the changes available to the **Server** current execution.

Optionally, you can use this function to apply changes made to Work Portal Views immediately. When the project is changed and re-deployed or when some changes are made to Views from Portal Administrator, these changes are applied automatically to the **runtime environment** and therefore applied to all Work Portal sessions. However, there is a delay caused by a queue of notifications that the server has to process. **Refresh Server Data** option can be used in these cases in order to avoid waiting for the time it takes for the server to process notifications of changes.

Some of the changes made in FuegoBPM Studio are updated to the currently opened FuegoBPM Studio users **Work Portal** sessions. Other changes will take effect provided that the users that are currently logged in, later logout and relogin.

To refresh the server data

1. Go to **Run** menu from **Process Studio**.
2. Select **Refresh Server Data** option.
3. A progress bar displays showing the percentage of the total time of the refreshing operation that has already elapsed so that you know how long the refreshing process will take to finish.




After updating the data, changes that are automatically applied to Work Portal sessions and server execution include:

- Changes in the set of default views caused by a **Publish & Deploy** operation. Work Portal Views might have been changed after project deployment due to changes in the processes or due to changes made to deploy preferences. See Deploy Preferences for further details.
- Changes to Custom Views and Presentations made through Portal Administrator, New Custom Views and Presentations created.
- Changes made to the editable server preferences.

Changes that will be applied provided that the currently logged in users re-login include:

- Changes in the set of roles assigned to users.
- Changes made to the groups the participants belong to.
- Changes to the Organizational Units the participants are assigned to.
- Remove operations on organization objects such as participants, roles, groups, etc.

Note

 Users that are logged in to Work Portal when the Refresh Server Data is executed will remain working as if no changes had been made to Organization properties. Changes made to views will be automatically updated. Other changes will not be visible until a logout and relogin is performed.

The **Refresh Server Data** option is only available when the server is currently running. If the server is not running, this option is not

necessary since all the changes to the organization are included in the Server execution when the Server is started.

Inter-process Communication

Inter-process Communication (IPC) is the exchange of data between one process and another, either in the same computer or over a network.

When the runtime environment is provided through **FuegoBPM Express** or FuegoBPM Enterprise runtime server edition, IPC might be needed when two or more departments within your company need to exchange information of processes designed in each department's projects.

Using IPC, a process can pass instance information to a child process (subprocess) or it can send a

notification event to processes located in the same FuegoBPM Enterprise Server or in different Servers.

Companies could be in need not only of communicating processes from different departments within the organization but also of processes working independently in a different company.

To achieve a process for communication, FuegoBPM Studio provides an interface composed by a set of specific activities that allows for sending and receiving notifications between processes. See IPC activities - Organizations interaction for a detailed description on how to use IPC interface in you process model.

IPC might be included in a project model to achieve:

- **Modularity:** A process might be split into subprocesses to avoid high complexity levels in the design.
- **Reusability:** The same process can be called from more than one process, even twice or more times from the same process.

- Load balancing: the processes can be set in different servers to balance the work load.

Possible IPC scenarios in FuegoBPM Express or FuegoBPM Enterprise

Depending on where the processes to communicate are deployed, IPC will take place in one of the following scenarios.

- Same Server: In order to achieve modularity, a process is split into several processes that need to connect to each other. In such cases the processes will reside in the same server.
- Business to Business (B2B) - server to server: It is useful when two or more companies or different departments within the same organization using different projects need to interact in the same process.

FuegoBPM Studio responds to all above-mentioned scenarios in the same way. However, IPC is internally implemented in a different way for each case.

For processes that reside in the same server, the communication is quickly solved by using internal mechanisms.

When the processes reside in different organizations or in different projects within the same organization, FuegoBPM Studio makes use of Process Referrals in order to find the location of the target process.

Process Referral allows integration with other organizations that also use FuegoBPM Studio.

In all cases where a company decides to include IPC in its processes, it will need to know the process to communicate with.

In some cases, it might be useful to use the **process interface** instead of using the complete process design as the target process. For instance, in B2B scenarios it is possible that a company reveals only what is needed to send or receive notifications that generate the process interface.

The process interface doesn't contain any business logic detail but only the minimum information required to be called as a subprocess. See Generating a process interface for further details on how to create a process interface.

Server-to-Server configurations

Configuration variations may exist when processes are distributed across multiple servers. The Servers may run on different machines, inside the same private network or in different private networks.

The following diagram represents possible configurations of Server-to-Server communications.

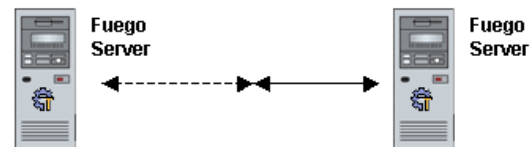
Configuration 1

Two servers connected on a public network



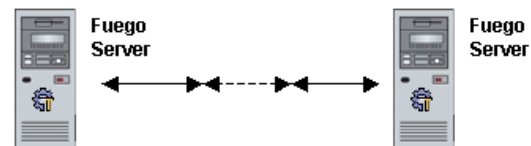
Configuration 2

One server runs on a machine connected to the public network and the other server runs inside a private network. This configuration avoids installing additional software to forward notifications to the other server inside the private network.



Configuration 3

Most common configuration. Both servers are behind firewalls connected to private networks. The servers connect to each other via the public IP network involving multiple IP addresses. Involves network to network communication across firewalls. A port is generally defined through the firewall. The common port 80 that accommodates HTTP traffic is also used



Enabling B2B

To configure FuegoBPM Servers in order to enable B2B communication, both organizations involved must be provided with the IPC abilities. In order to enable IPC service, some properties must be configured. See IPC Server Properties for further details on how to enable IPC communication.

In addition to enabling IPC service, you must configure **Process Referrals** to provide FuegoBPM Studio with the information needed to reach a process in a remote location. It is also needed to create a participant in order to connect to the remote organization.

See Also

- How to configure Process Referrals
- How to configure participants

Chapter 22. Executing Processes

Publish and Deploy

Publish and Deploy activates the process design into a real-time situation where the activities and roles can be fulfilled automatically or by human users.

When you are satisfied with the project design meeting your business's requirements, the **Publish & Deploy** option allows you to publish the entire project and deploy it in a FuegoBPM Enterprise Server.

Although it is absolutely transparent for you as a **FuegoBPM Studio** user, when clicking on **Publish & Deploy** option, two main steps are performed:

- Publishing a project has two main purposes:
 1. Each process is prepared for deployment to the end users' real-time environment.
 2. BP-method (business logic) is used to build Java source code, which is compiled to Java classes.

- During deployment:
 1. The process is associated to a certain Organizational Unit
 2. The Server is notified that a new process (or a new version of an already deployed process) is available to users associated to that

Organizational Unit, so that they can begin working with it. If the end users are currently working with an older copy of the business process in the Work Portal, they will not lose any of the instances in their queues.

Publish and deploy a project

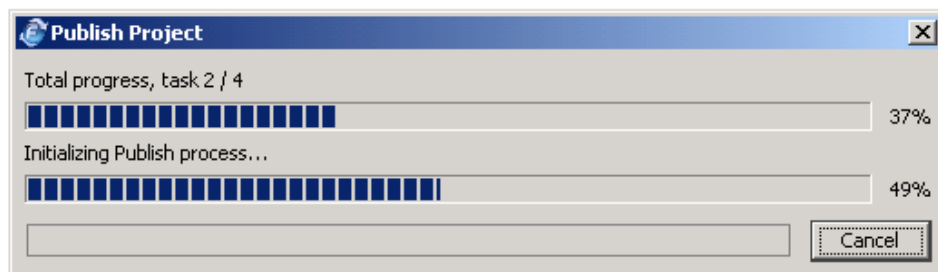
To publish and deploy a project

1. From the FuegoBPM Studio menu options, select **Run**, and then **Publish & Deploy** or click on **Publish & Deploy** icon



in the toolbar.

2. The **Publish Project** progress bar appears showing the percentage of time already elapsed over the total time the publish operation will take.



When publishing and deploying the project, the settings configured in Server Preferences->Deployment are checked. Every process in the project will be published and deployed in the Organizational Units specified in this dialog. If a certain process does not appear here (because the user removed every reference), the default Organizational Unit (root) will be assumed. The publishing also detects if there were processes currently deployed that have been removed from the project. If any old process is found, it is

undeployed. This means that end-users will not be able to work in that process anymore. Views will be generated as indicated in deployment server preferences too.

If the **Remove previously published and deployed processes in next publish** checkbox is checked, **all** currently deployed processes are undeployed first and then, the deployment takes place for all the processes in the project.

Undeployment of processes implies that all the instances created and running in that process will be removed.

It also makes all custom views lose references to the processes they have been assigned since the processes are removed and recreated.

Note



It is recommended that you always check the **Deploy Preferences** settings before publishing the project. The **Deploy Preferences** window allows you to change the Organizational Units where processes are deployed and decide whether **Work Portal Default Views** are automatically generated for end users or not. See Server Preferences->Deployment for further detailed information.

Starting the Server

FuegoBPM Enterprise Server is an essential element of the FuegoBPM Suite. It is responsible for:

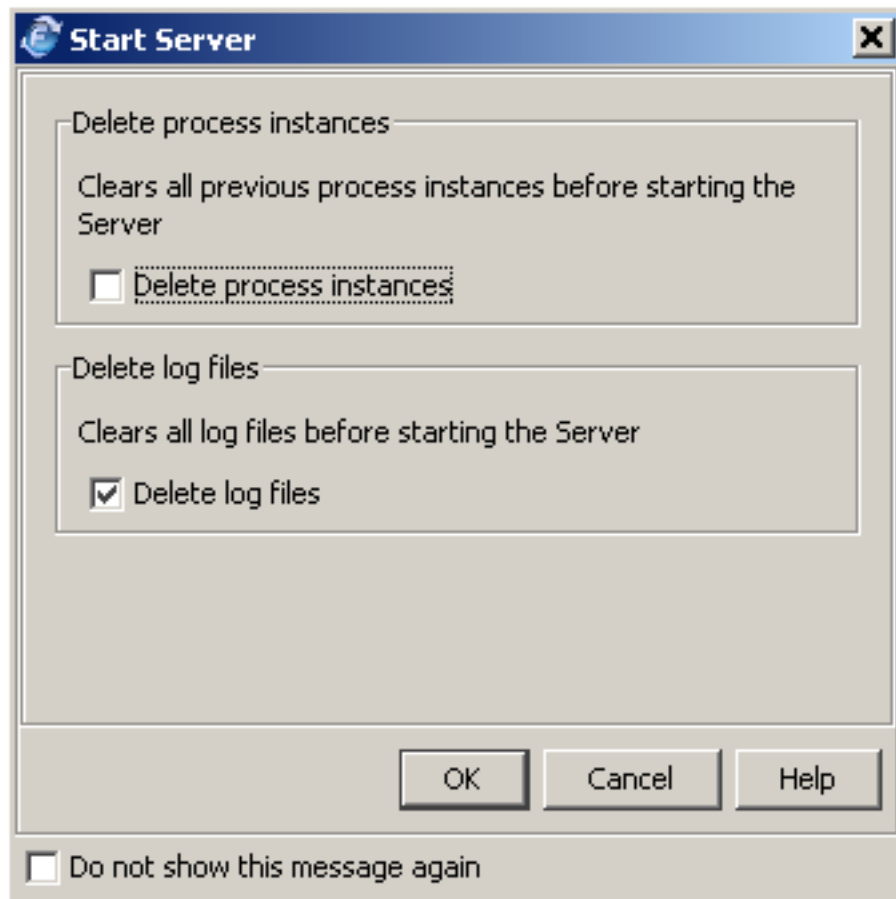
- Accepting requests generated from the Work Portal.
- Executing required tasks.
- Maintaining the state of all instances flowing through the deployed processes.
- Providing extensive version control that allows process models to be modified, published and deployed with zero-latency, since multiple versions can be run simultaneously.

FuegoBPM Studio automatically creates and configures the Server for each new project. It is not necessary to perform any configuration to design your processes and publish them to the created server.

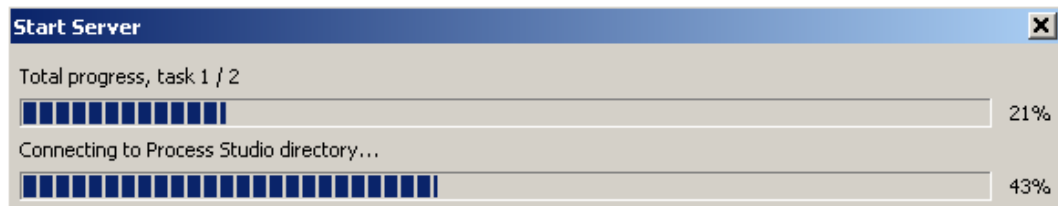
The published and deployed processes start to execute in a **runtime environment** after starting the server.

To start the Server

1. Go to **Run** menu
2. Select **Start server** option. The option is only available when the Server is currently stopped.
3. If the preference **Always show options before starting server** set in Server Preferences->Runtime is checked, the following dialog appears:



4. Check the **Delete process instances** to clear all previous processes instances before starting.
5. Check the **Delete log files** if you want to clear all log files before starting the Server.
6. A progress bar shows the percentage of lapsed time over the total time that the starting process will take.




Once the Server is started, the deployed processes start running.

Furthermore, administrator users can connect to **Portal Administrator** to administer the way the information is presented to users **Work Portals** and **FuegoBPM Work Portal Users** can connect to Work Portal to process interactive tasks.

To start Work Portal

1. From Process Studio, go to **Run** menu.
2. Select **Launch Work Portal** option. An Internet browser connects to the URL where Wb Portal web application for the currently opened project is deployed. You will be prompted to enter login information.

Note

 **Note** : Remember that Server runs in a **runtime environment** completely isolated from the environment where **FuegoBPM Studio** runs. This is why once it has been started, all the changes set to the **Organization** data will not be updated to the server execution unless the Refresh Server Data function is performed or the project is re-published using Publish & Deploy option.

Stopping the Server

To shut the Server down

1. Click **Run** menu from **FuegoBPM Studio**.
2. Select **Stop Server** option. This option is only enabled when the server is running.
3. The Stop server progress bar appears and shows the percentage of elapsed time over the total time the stopping process will take.



When the FuegoBPM Enterprise Server is stopped, all the processes stop executing. This means that no task is executed. Besides, FuegoBPM Studio users will not be able to connect to their **Work Portals** . Administrator users will not be able to connect to **Portal Administrator** either.

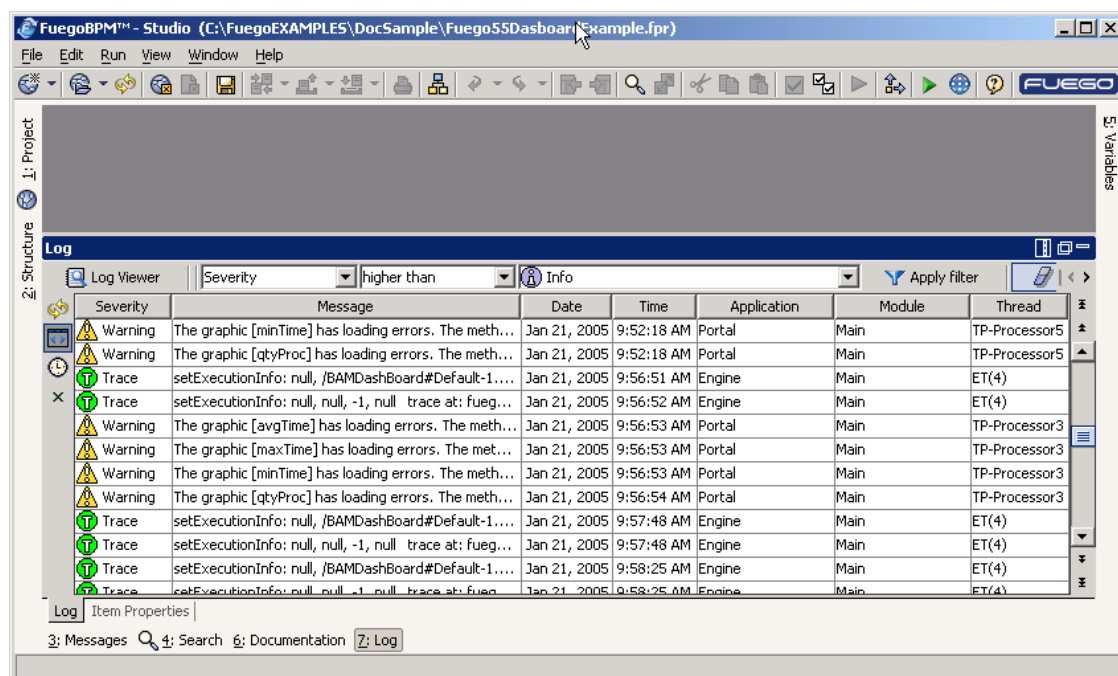
Chapter 23. Log Viewer

FuegoBPM Log Viewer

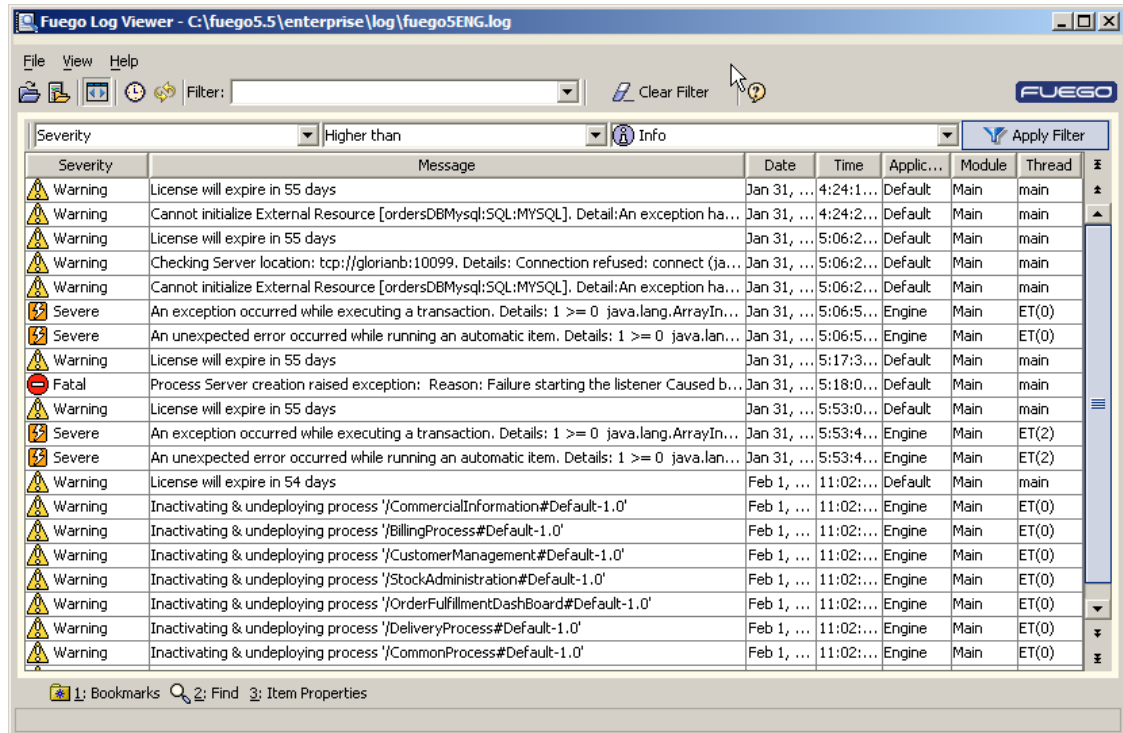
The FuegoBPM Log Viewer enables you to read information logged by the FuegoBPM Server. A set of log files is created for each project you define. FuegoBPM Log Viewer reads the files and displays them to help you monitor and trace Server execution.

To view the log


Click the Log option at the bottom of FuegoBPM Studio. A new panel appears and displays a log viewer with restricted functionality.



Click the **FuegoBPM Log Viewer** button on the top-left corner of the Log Viewer panel to enable complete functionality of the Log Viewer.




Note

 Log files are not generated and, therefore, cannot be read by **Log Viewer** until the project is published or the Server is started for the first time.

The Work Environment

Click **Log** at the bottom of **FuegoBPM Studio** or launch the **FuegoBPM Logviewer stand alone (Enterprise version)** to open the Log Viewer panel. This panel enables you to view the messages logged by the FuegoBPM Server. In FuegoBPM Studio, the panel is a limited version of the Log Viewer.

If you are working with FuegoBPM Studio, click the **FuegoBPM Log Viewer** button  within the Log panel to launch the Log Viewer in a new window.

The work environment changes a little from one version to another. The description of all the functions of **FuegoBPM Studio Log**

Viewer Panel or the **Log Viewer window** are provided below.

Log Viewer Menus

The **File**, **View**, and **Help** menus are only available from the **Log Viewer** window. However, some of the functions included in these menus are available from toolbar icons in the **Process Studio Log Viewer Panel**.

File Menu

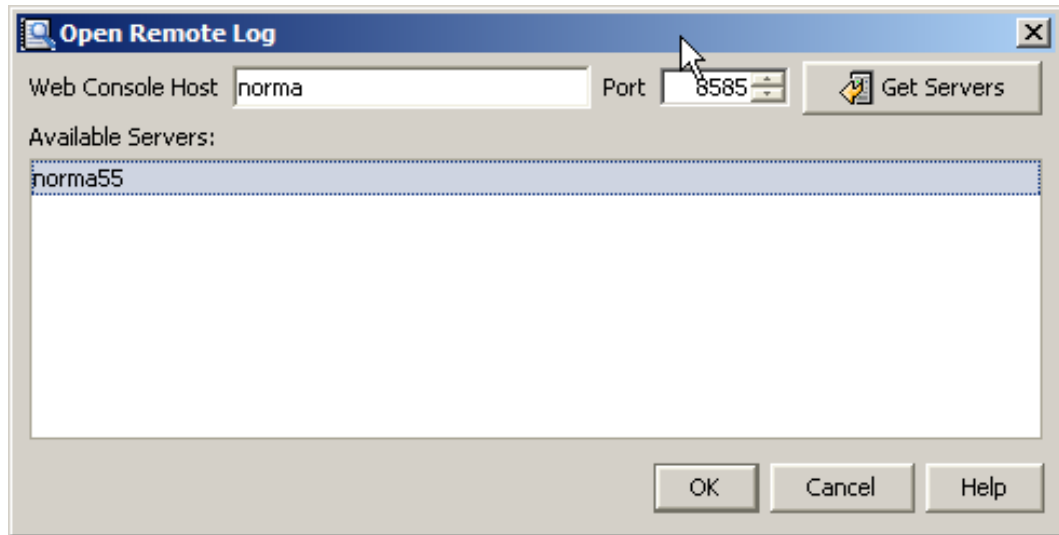
The following table describes the options available from the **File** menu.

Option	Description
Open	Opens a local log file
Open remote log ...	Opens a remote log file. Valid for the FuegoBPM Logviewer Standalone (FuegoBPM Enterprise edition)
Save As	Saves a log file as a .txt file.
Close	Exits the Log Viewer.
Preferences	You can define some preferences when executing the logviewer

Open remote log ...

This option is present when you are working with the FuegoBPM Logviewer Standalone version. It allows to open a log localized in a remote host.

1. When you select this option, the *Open remote log* dialog opens.



2. Write the host name where the FuegoBPM Web Console runs and its port. Then select the **Get Servers** button and type the Fuego Administrator credentials in the *Username* and *Password* of the dialog opened. The list of FuegoBPM Servers is displayed in the *Available Servers* low portion of the dialog.
3. Double click in the server you are interested, and its log will be displayed in the FuegoBPM Logviewer main pane.

PREFERENCES

General:

- **LogViewer Size:** Indicates the number of items to display by page.
- **Update Frequency:** period of time (seconds) to refresh the Lowviewer with new log information.

Time Zone:

The time within each log message from the log file, is represented as

GMT 0. Therefore if you want to view the logs and you belong to a different GMT then you can see the time of each log message in real time by setting the logviewer time zone based on the server that generated that log file.

- **Time Zone:** select the time zone in which the log file was generated
- **Engine Time zone:** it indicates the server's time zone. The server that generated this log file. It will indicate *not available* if there is no opened log file or the log file corresponds to an old version of FuegoBPM.
- **Select the Engine time zone:** the logviewer timezone is set automatically with the server's time zone (the server that created that log file). It is enabled once you have opened a log file .

View menu

The following table describes the options available from the **View** menu.

Option	Description
Language	Set the language to see the FuegoBPM LogViewer
Clear Filter	Clears the currently applied filter.
Automatic adjustment to window	Sets the table to auto-resize mode when a column is resized. This option allows you to see all the columns in the window or panel with no need to scroll through them. If you disable this option, you will need to use the scroll bars to see all the columns. The Log Viewer window

Option	Description
	becomes scrollable from right to left.
Automatic Refresh	Automatically refreshes the Log Viewer main workspace based on the setting of the Update frequency seconds, as defined in the Log Viewer preferences. Every n seconds, the Log Viewer changes focus to the end of the file to show the latest logged items.
Refresh	Shows the latest logged items.





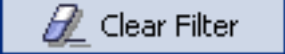



Help Menu

The following table describes the options available from the **Help** menu.

Option	Description
Contents	Launches the Log Viewer online help.
About	Displays information about the Log Viewer version and virtual machine information.



Log Viewer Toolbars

The Log Viewer toolbars display shortcuts for the most frequently used menu options. The following table lists the tools and their functions. The tools are listed in the order they appear on the toolbar from left to right.

Icon	Description	Where It Is Available
	Opens remote log	Log Viewer
	Shows the latest logged items.	FuegoBPM Studio Log Viewer Panel & Log Viewer Window
	Drop-down list containing saved filters.	Log Viewer window
	Applies the defined filter.	FuegoBPM Studio Log Viewer Panel & Log Viewer Window
	Clears currently applied filter.	FuegoBPM Studio Log Viewer Panel & Log Viewer Window
	Sets the logs table to auto-resize mode when a column is resized.	FuegoBPM Studio Log Viewer Panel
	Automatically refreshes the Log Viewer based on the setting of the Update frequency seconds, as defined in the Log Viewer preferences.	FuegoBPM Studio Log Viewer Panel
	Launches the Log Viewer online help.	Log Viewer window

Log Viewer Tabs

The Log Viewer has three tabs that control additional functionality. Tabs can be located on the top, right, bottom, or left of the window depending on your selections.

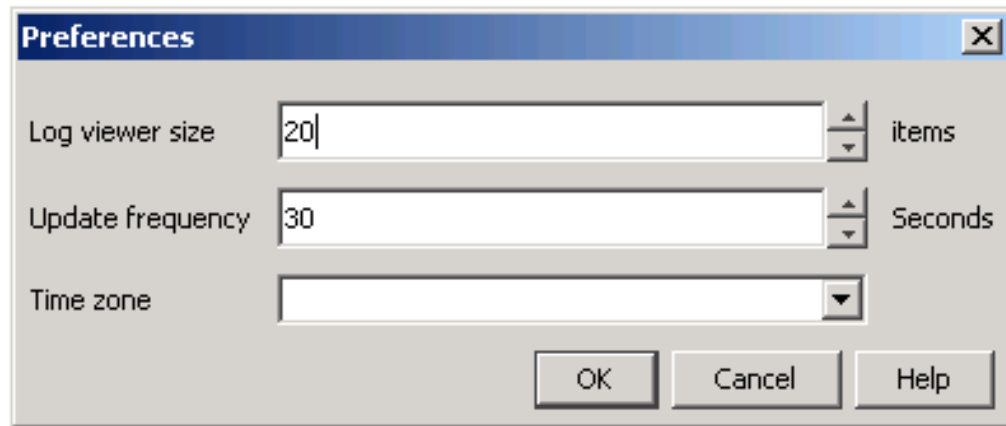
Tab	Description
 <u>2</u> : Find...	Opens the find panel and creates a log file filter or selects a log file filter.
 <u>1</u> : Bookmarks	Opens the Bookmarks panel.
<u>3</u> : Item Properties	Displays complete information for a logged item. Opens a window with the entire logged message and the properties of the log item, such as Severity, Time, Application, and so on.

Setting Preferences

Some preferences can be set when using the **Log Viewer** window.

To set the Log Viewer preferences

1. If the Log Viewer window version is in use, select **File** then **Preferences** from the menu options. If you are using the Log Viewer panel from FuegoBPM Studio, select the FuegoBPM Studio **File** menu option, the **Preferences**, then **Log** in the preferences window. In either case, the following preferences can be configured:



2. In the **Log viewer size** field, type the number of items in the log viewer size field. This indicates the number of items or rows that are displayed in the viewer.
3. Type the **Update frequency** rate. This number indicates the time that must pass before the viewer is updated. The Server is constantly writing information to the log file. The viewer is updated automatically at the interval that is specified in this field.
4. In the **Time zone** field, choose the time zone where you will view the log files. Time zone impacts how you see the date and time of log items. If you don't specify a time zone, Log Viewer gets the default time zone for the host where FuegoBPM Studio is running.

The Logged Information

When the Log Viewer panel is displayed, log items (or log messages) are displayed in date and time order from oldest to newest. The following information is displayed:

Columns	Description
Severity	Indicates the kind of message (FATAL, SEVERE, WARNING, INFO, DEBUG).
Messages	The message that the Server sends to the log.
Time	The time that the message was logged.
Date	The date the message was logged.
Application	Application that sent the message. All FuegoBPM Suite applications can send log messages to the log files.
Module	Module that sent the message.
Thread	Thread that sent the message.

Note



Each column can be resized as needed.

Severity

The severity of the logged messages depends on the FuegoBPM Server properties.

Time and Date

When applying filters to time and date attributes, the time can be matched as *absolute* or as *relative*.

- **Absolute** means a fixed time.
- **Relative** has a value made up of the current date and time plus a value you can set. Relative times are calculated each time the

search is performed and are based on the actual date/time (now). The value for this field is a number of minutes, hours, days, weeks, months, and years to be added to the current date and time. To define a past date or time, select a negative number as the value.

Date Format

The date is saved in the GMT+0 format.

Note



If the log is generated in a country where GMT+6 format is used and the log is read in a country using the GMT+0 format, the time stamp within the file will not match the actual time the log was saved. For example, if a log item was registered at 8:00 A.M. in Mexico GMT+6, it will be displayed as 2:00 P.M. in a country located at GMT+0.

The date is converted to the local time zone of the computer where the Log Viewer is running.

Selecting the Columns to be Displayed

You can choose to display or hide specific columns in the Log Viewer.

To select columns to display

1. Right-click on a column heading in the Log Viewer. The complete list of columns appears in a menu.
2. Select the check box next to the column headings you wish to see in the Log Viewer. Clear the check boxes next to the column headings that you want to leave out.

Message	Severity	Date
No efect with the new runLe	✓	c 2, 2003
Executing item: IMMEDIATE	✓	c 2, 2003
Executing item: IMMEDIATE	✓	c 2, 2003
- ToDoService switching to ru	✓	c 2, 2003
Executing item: IMMEDIATE	✓	c 2, 2003
- NewsDispatcher switching b	✓	c 2, 2003
Executing item: IMMEDIATE	✓	c 2, 2003
Executing item: IMMEDIATE	✓	c 2, 2003
Executing item: IMMEDIATE	✓	c 2, 2003




Changing the Column Order

You can change the columns' order using drag and drop. Click on a column heading and while holding the mouse button, drag the column to the new location.

Scroll Bar Functions

The number of lines displayed in the Log Viewer window is determined by the number of lines defined in the Log Viewer preferences dialog box. Use the extended scroll bar functions to view all of the logged items.

The extended scroll bar functions are as follows:

-  *Up/down arrow* - moves the page one log item up or down while keeping your selected item highlighted.
-  *Previous/Next page arrow* - moves the main panel to the previous or next page. The page size is defined in the Log Viewer preferences.
-  *Begin/End arrow* - moves to the first or last log item within the log file.

For example, assume that the number of lines defined in the Preferences dialog box is 1,000. The complete log file has 5,000 log items divided into 5 logs. You have selected items from 1001 to 2000.

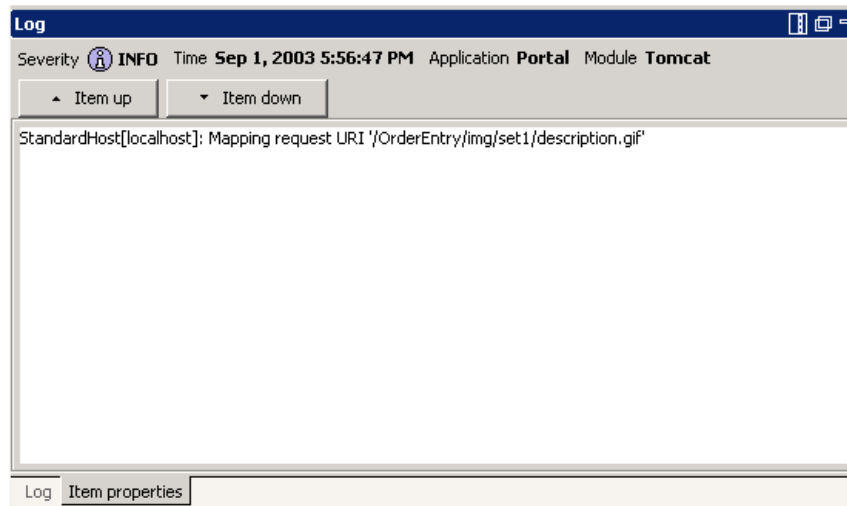
- Item up arrow shows item 1000 (window contains items 1000 to 1999)
- Item down arrow shows item 2001 (window contains items 1002 to 2001)
- Page up/down arrows show items 1 to 1000 or 2001-3000
- Begin/End log arrows show the first log item (1) or the last log item (5000)

Auto-resizing the Log Viewer Window

You can automatically adjust the Log Viewer window to set the table to auto-resize mode when a column is resized. This option enables you to see all of the columns in the main panel with no need to scroll to see columns that do not fit in the window. If you disable this option, you must use the scroll bars in order to see all the columns.

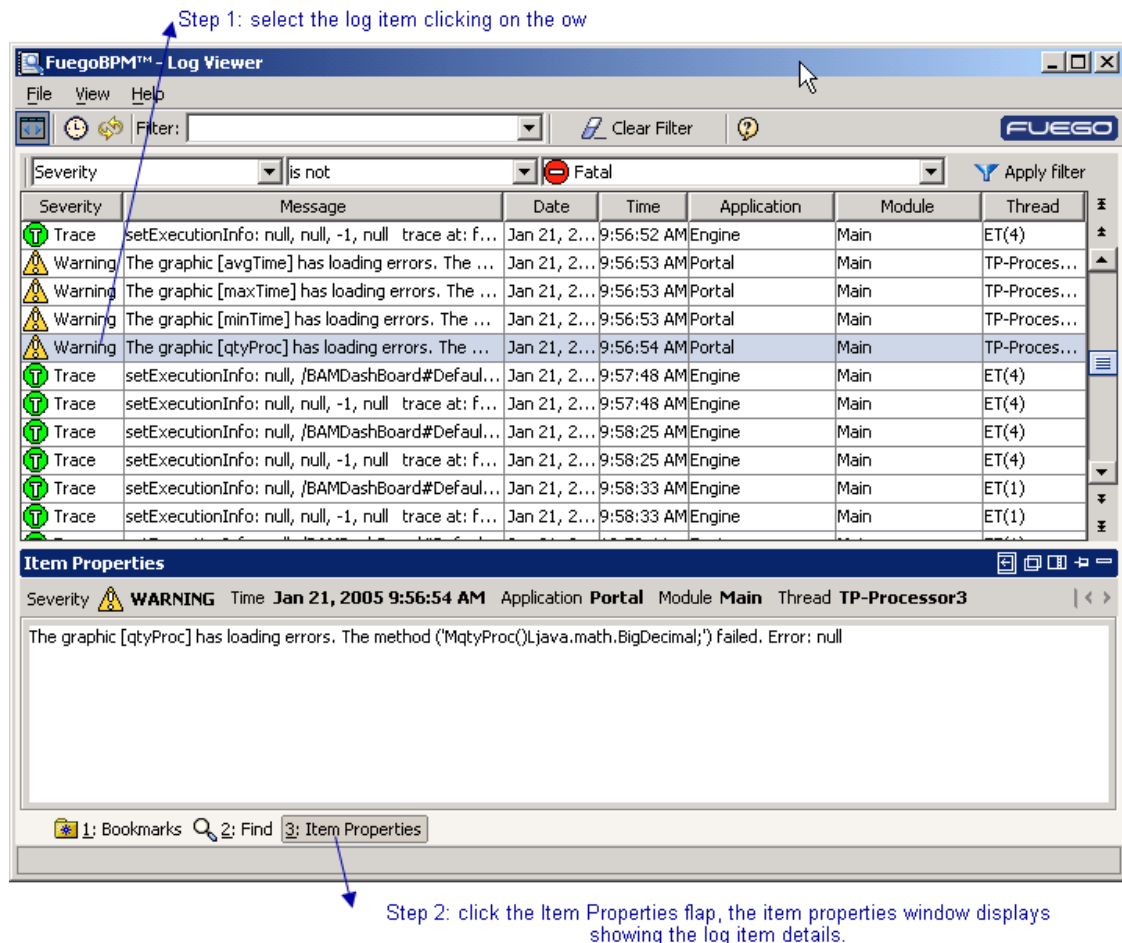
Log Items Window

Double-click on a log item and the item properties window is displayed:



Use the **Item up** and/or **Item down** buttons to see the previous or next item details. Click the **Log** tab at the bottom of the panel to return to the complete list of log items.

From the **Log Viewer** window, select the log item row, then click the **Item Properties** tab. The **Item Properties** panel is displayed:



Automatically Refreshing the Log Viewer

If configured, the Log Viewer will change focus to the end of the log file every *n* seconds to show the latest log items. Disable the auto-refresh option when you are applying filters or watching bookmarked items because the refresh operation changes the view to the end of the log file.

Filters

Log Viewer provides several ways to find log messages in a log file. You can use the **Quick filter toolbar** to filter the log messages matching a single filter condition. This option is available from both the **Process Studio Log Viewer Panel** and the **Log Viewer**

window.

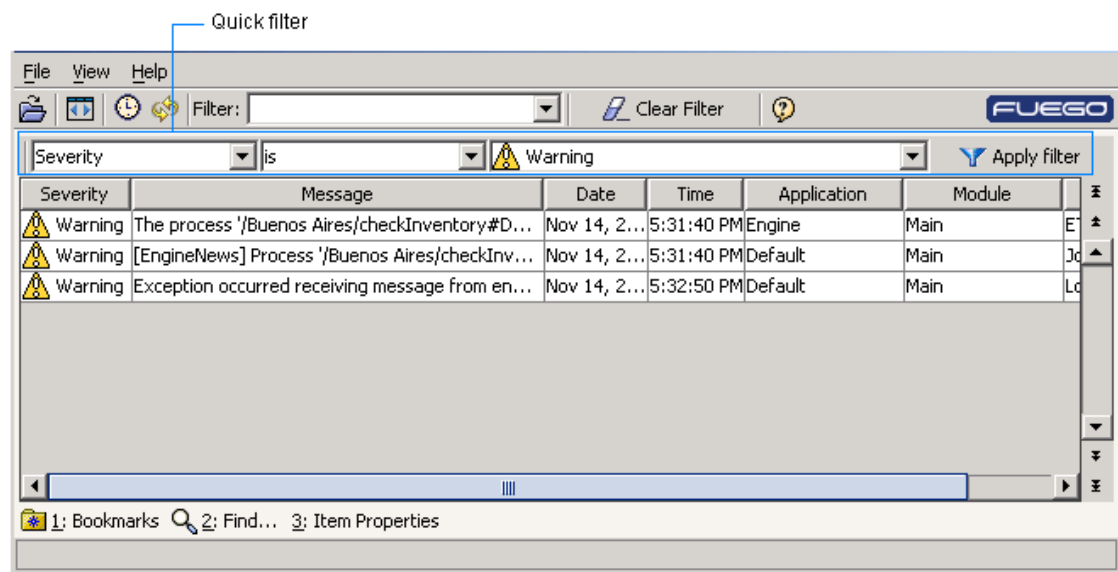
You can also define multi-condition filters to find a specific set of log items. These filters can be saved for use at a later time. There is no limit to the number of filters you can define. Filters are only available from the **Log Viewer window**.

Finding Log Items


If a single condition provides enough information to find the log items, you can use the quick filter toolbar. For example, if you choose the filter condition "Severity is Warning," the Log Viewer displays all logged items with the severity level of *Warning*.

Applying Quick Filters

The **Quick filter** option is available in the **Process Studio Log Viewer** panel and in the Log Viewer window. To apply a quick filter, select the filter condition and click the **Apply filter** button. The results are displayed in the Log Viewer main window:

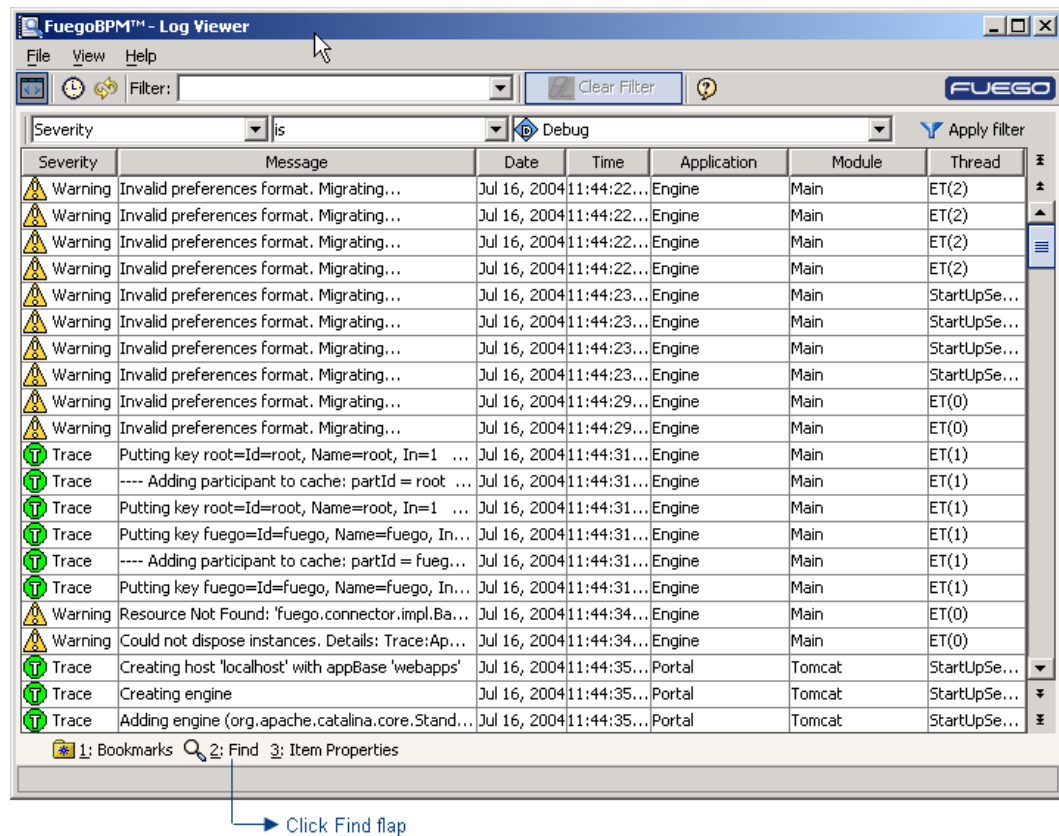


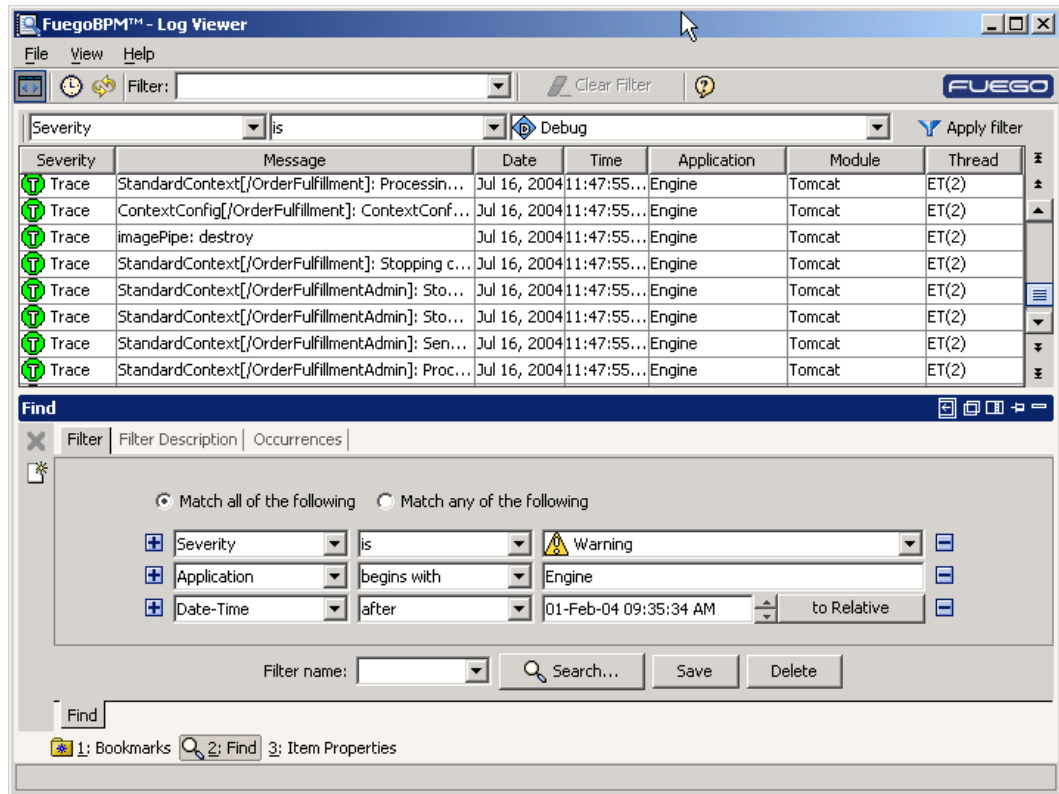
Note

 When using the quick filter, you can apply only one filter condition at a time. If you search for items using a more complex search condition, you need to use the **Filter** button.

Using Multi-condition Filters

1. Click the **Find** tab in the Log Viewer window. The **Find** panel is displayed in the bottom section of the Log Viewer:



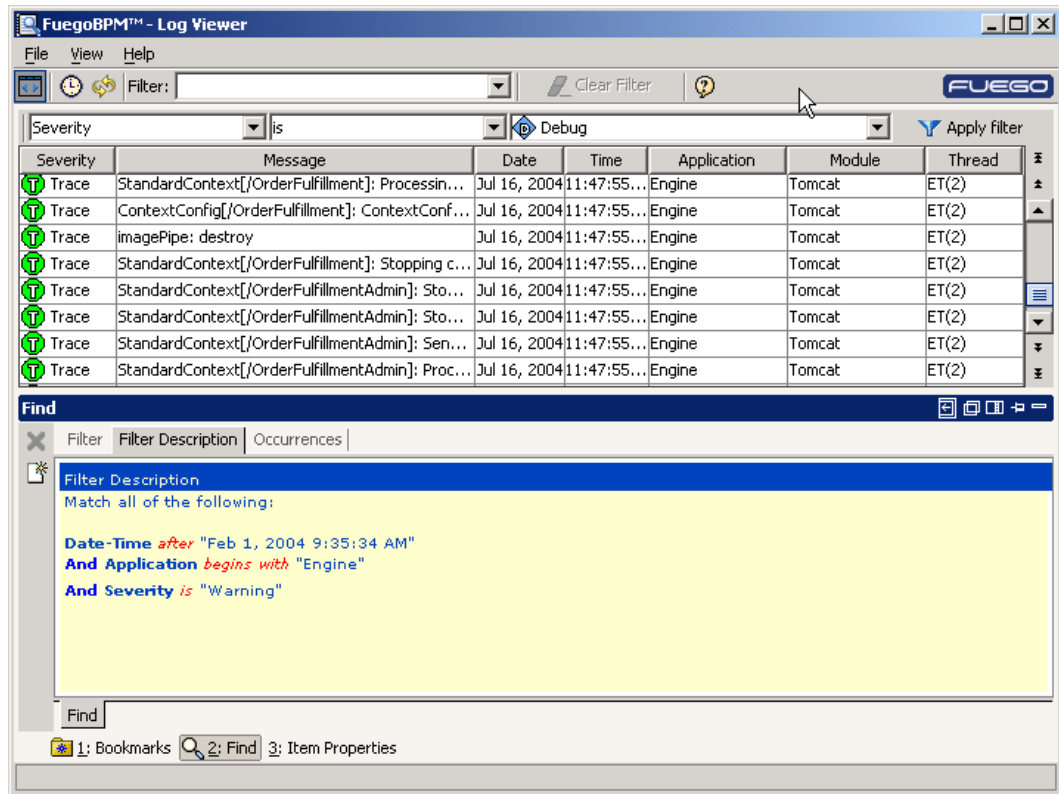


- On the Filter tab of the **Find** panel, enter all of the conditions that the logged items should match using the **plus** sign icon. The **Match all of the following** check box indicates that the log items to recover must match all the defined conditions. The **Match any of the following** check box indicates that the log item to recover must match at least one of the conditions. For further information on how to combine conditions, see Connector Rules. Take into account the following options:

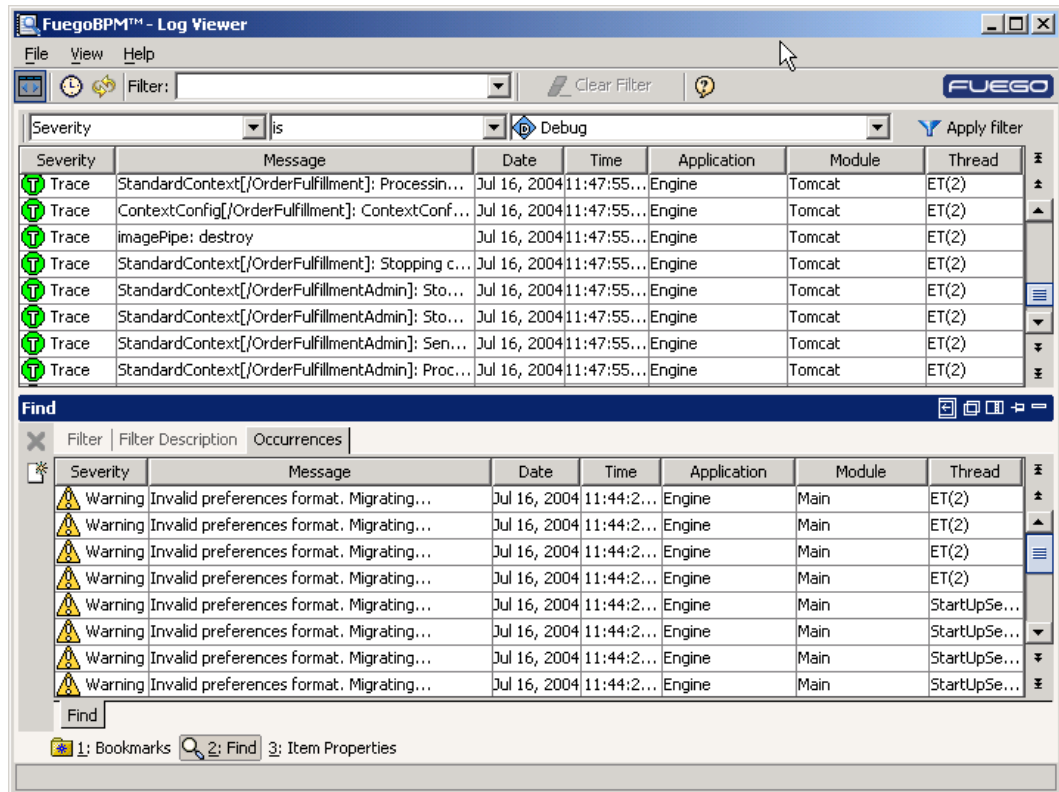
Condition	Connector	Options
Severity	is, is not, higher than, lower than	Debug, Info, Warning, Severe, Fatal
Message	begins with, ends with, is, is not, contains, not contains	Type a search string.

Condition	Connector	Options
Date-Time	is, is not, before, after	Absolute- a fixed date, such as 3/3/2001 23:43:56.Relative- the date is calculated each time the search is performed and is based on the current date and time. The result of the query is a point in time before or after the specified value.
Application	begins with, ends with, is, is not, contains, not contains	Type in a search string.
Module	begins with, ends with, is, is not, contains, not contains	Type in a search string.
Thread	begins with, ends with, is, is not, contains, not contains	Type in a search string.
Level	is, is not, higher than, lower than	Select the level from drop-down list.

The Filter Description tab displays a detailed description of the search filter and all applicable conditions.

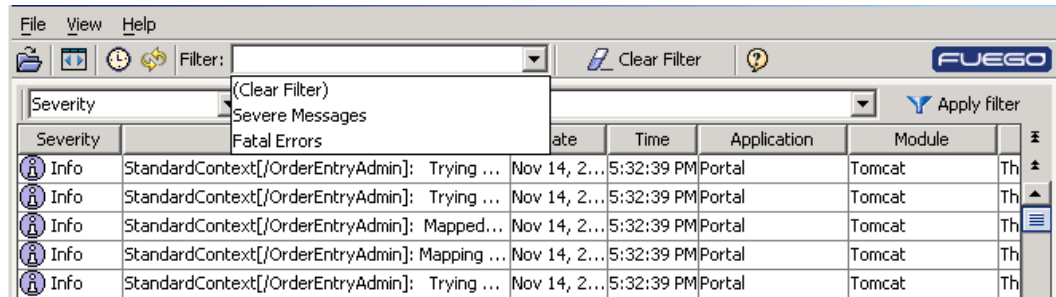


3. Click **Search** to search the log for all messages matching your filter conditions. The items matching the criteria are listed in the Occurrences tab on the Find panel.




The number of items displayed here is determined by **Log Viewer preferences** configuration. These items are ordered from the oldest to the newest. If the number of log items exceeds the defined lines to display, a warning message is displayed at the bottom of the **Find** panel.

4. To save the filter, type a name in the **Filter name** field (using any combination of alpha or numeric characters) and click **Save**. Once the filter is saved, it is included in the filters drop-down menu in the Log Viewer toolbar. You can select one of the stored filters to find the log items matching the filter conditions at any time while using the Log Viewer.



Note

 When the Find panel is open, you can create new filters by clicking the **New** button. You can have multiple filters open at the same time. Each filter is displayed in a separate Find panel. Also, if you double-click on a log item in the Occurrences tab of the Find panel, it is highlighted and displayed within its context in the main Log Viewer panel.

Deleting Multi-condition Filters

To delete a filter

1. From the Log Viewer menu **Find** tab, select the filter name from the drop-down list.
2. Click **Delete** to delete the filter.

Clearing Filters

After applying a quick filter, the Log Viewer main panel displays only those log items that match the condition of the filter.

To clear the filter from the currently viewed log

- From the Log Viewer Menu, select **View** then **Clear Filter**.

or

- From the **Process Studio Log Viewer Panel**, click the **Clear filter** button.

After clearing the filter, the Log Viewer displays the complete list of all log messages.

Connector Rules

If more than a condition for the same attribute (for example, Severity) is defined within a Filter, they will be connected to resolve the complete condition. This is accomplished using the logical operators OR and AND. Each connector is classified into a type and based on the combination of the same or different type, the **OR** or **AND** applies.

Classification I

Description	Connector
Is	+
Is not	-
Contains	+
Not contains	-

Using the above connectors, the following rules apply:

Connector	Connector	Operator	Description
+	+	OR	The combination of two types of + will be connected by an OR
+	-	AND	The combination

Connector	Connector	Operator	Description
			of one type + and another type - will be connected by an AND
-	+	AND	The combination of one type "-" and another type + will be connected by an AND
-	-	AND	The combination of two types of "-" will be connected by an AND

Examples

Conditions	explanation
Severity is Debug / Severity is Info	Both conditions use the "is" connector, therefore results displayed in the Log Viewer will contain a severity of DebugORInfo
Severity is not Debug/ Severity is not Info	Both conditions use the "is not" connector, therefore results displayed in the Log Viewer will contain a severity of DebugANDInfo

Classification II

- Begin with: B

- End with: **E**
- Before: *lower than* sign.
- After: *greater than* sign.
- Lower than: *lower than* sign.
- Higher than: *greater than* sign.

Using the above connectors, the following rules apply:

- **B, B: OR** - The combination of two types of **B** will be connected by an OR
- **B, E: AND** - The combination of one type **B** and another type **E** will be connected by an AND
- **E, B: AND** - The combination of one type **E** and another type **B** will be connected by an AND
- **E, E: OR** - The combination of two types of **E** will be connected by an OR
- *lower than, lower than: AND* - The combination of two types of *lower than* will be connected by an AND
- *lower than, greater than: OR (OI)* - The combination of one type *lower than* and another type *greater than* will be connected by an OR
- *greater than, lower than: AND (CI)* - The combination of one type *greater than* and another type *lower than* will be connected by an AND
- *greater than, greater than: AND* - The combination of two types of *greater than* will be connected by an AND

Examples

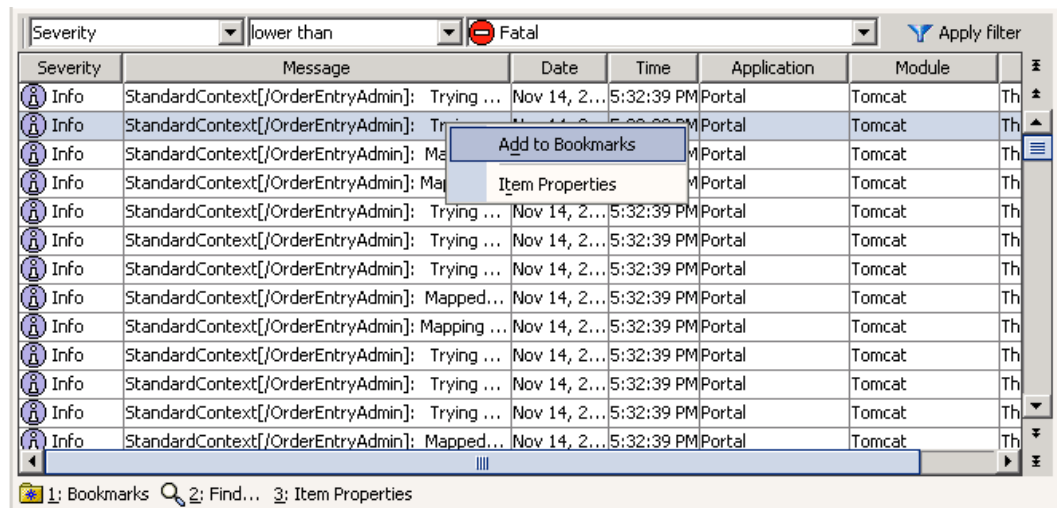
Conditions	explanation
Message Begins with "The server" / Message Ends with "successfully"	The conditions will be combined using AND.
Level Higher than 2 / Level Lower than 5	The conditions will be combined using AND (CI).

Bookmarks

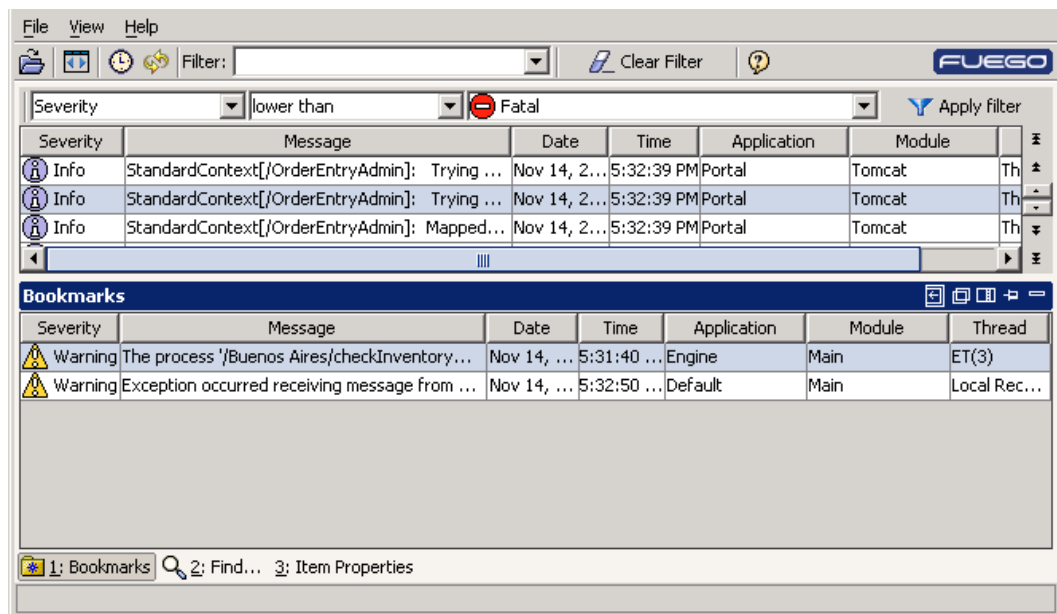
Log Viewer enables you to bookmark a log item so that you can easily find it among all of your logged messages. More than one item can be bookmarked at the same time. In the current Log Viewer session, you can double-click on a bookmarked item to view it within the context in which the logged event occurred. This function is only available in the **Log Viewer window**.

To bookmark a log item

1. Open a log file in the main Log Viewer window.
2. Right-click on a log file item and select **Add Bookmark** from the pop-up menu. The selected item is added to the Bookmarks panel.



- Optionally, click the **Bookmarks** tab to view all your bookmarked items in the **Bookmark panel** at the bottom of the Log Viewer window.



If you double-click a log item in the Bookmark panel, it is displayed in the top panel of the Log Viewer window. Furthermore, if you have applied a quick filter so that the main (top) panel contains ONLY the log items that match the unique condition, then you double-click a log item that you saved as a bookmark, the filter is cleaned and the

bookmarked item is displayed in the main panel.

Warning



Bookmarks are preserved even when the Bookmarks panel is not visible; however, remember that Bookmarks are **temporary** and will disappear when you close the Log Viewer session.

Working with Log Files

When you create a project in FuegoBPM Studio, a directory named for your project is created. The log files are saved in this directory under the sub-directory called *system*. For example, log files for the *YourProject* project are stored in the *YourProject.fpr/system* directory.

You will find up to five log files named **YourProject.log.x**, where x is 0-4.

These log files work in a rotating fashion. When the first file reaches maximum capacity, a new file is created. This continues until all five files reach capacity. Then, the data in the first file is overwritten and the file is reused. The files continue to be reused in a circular manner until the FuegoBPM Server is stopped and no other actions are taken that result in log generation.

Logs are used by all components of FuegoBPM Studio to record actions.

Chapter 24. Appendixes

Quick Definitions

The following are some quick definitions to be considered in order to understand the following:

- Groups Handling
- Exception Handling
- Compensate Handling

Activity

Represents the execution of some specific tasks.

Flow

Is a set of activities or *Groups* in some order, linked by transitions.

Group

Is an *Activity* that contains a *Flow* where *Inner Activities* have common behaviors, like timeout, exception handler, etc.

Inner Activities

Are *activities* contained in the *Group*.

Parent Group

Is the *Group* which contains the *Activity*.

Process Flow

Is a *Flow* that represents the normal behavior of the business instance. If the *Process Flow* encounters an exceptional condition, it should *back-out* or *compensate* for what it is doing before aborting.

Exception Handler Flow

Is a *Flow* defined as part of the same *Group* or *Activity* and it is executed when the handle *Group* or *Activity* fails or throws an *exception*.

Compensation Handler Flow

Is a *Flow* defined as part of the same *Group* or *Activity*. The *compensation* is used to release resources and reverse already executed activities.

Handled Group or Handled Activity

Is the *Group* or *Activity* being handled by the *Exception* or *Compensation Handler Flow*.

Appendix A - Implement FuegoBPM Studio VCS using VSS

Overview

The purpose of this document is to outline how to use Microsoft Visual SourceSafe (VSS from now on) within FuegoBPM Studio.

Source elements

Refer to Version Control System, to understand the Project structure and files that have to be taken into account to be stored into VSS.

How to use it

Processes

To add a process, procedure or screen flow,

1. Open VSS
2. Get the latest version of the project

3. Open FuegoBPM Studio
4. Create a process(es), procedure(s) or screen flow(s)
5. Check that your design is correct
6. Close FuegoBPM Studio
7. Add into VSS the new file(s) located under the *processes* folder
8. Close VSS

To modify a process, procedure or screen flow,

1. Open VSS
2. Get the latest version of the project
3. Check out the files that you want to modify
4. Open FuegoBPM Studio
5. Apply changes to selected process(es), procedure(s) or screen flow(s)
6. Check that your design is correct
7. Close FuegoBPM Studio
8. Check into VSS the file(s) that you have modified
9. Close VSS

To delete a process, procedure or screen flow,

1. Open VSS

2. Get the latest version of the project
3. Open FuegoBPM Studio
4. Verify that the processes, procedures and screen flows that you are trying to delete are not used by any other process, procedure or screen flow (You can do this by right clicking over the element and selecting the *Find Usages* option)
5. Delete the process(es), procedure(s) and screen flow(s)
6. Check that your design is correct
7. Close FuegoBPM Studio
8. Remove from VSS the deleted file(s)
9. Close VSS

Catalog

To add a module,

1. Open VSS
2. Get the latest version of the project
3. Open FuegoBPM Studio
4. Create the module(s)
5. Close FuegoBPM Studio
6. Add into VSS the file(s) located under the *componentCatalog* folder that has both the same name as the new module, and the *XCDL* extension
7. Close VSS

To add a component,

1. Open VSS
2. Get the latest version of the project
3. Open FuegoBPM Studio
4. Create or catalog the new component(s)
5. Check that your design is correct
6. Close FuegoBPM Studio
7. Add into VSS the new file(s) located under the *componentCatalog* folder
8. Close VSS

To modify a component,

1. Open VSS
2. Get the latest version of the project
3. Check out the file(s) that you want to modify
4. Open FuegoBPM Studio
5. Apply changes to selected component(s)
6. Check that your design is correct
7. Close FuegoBPM Studio
8. Check into VSS the file(s) that you have modified
9. Close VSS

To delete a module,

1. Open VSS
2. Get the latest version of the project
3. Open FuegoBPM Studio
4. Verify that no other process(es), procedure(s), or screenflow(s) uses the components inside the module(s) that you are trying to delete
5. Delete the selected module(s)
6. Check that your design is correct
7. Close FuegoBPM Studio
8. Remove from repository the deleted file(s)
9. Close VSS

To deleting a component,

1. Open VSS
2. Get the latest version of the project
3. Open FuegoBPM Studio
4. Verify that no other process(es), procedure(s), or screen flow(s) uses the component(s) that you are trying to delete
5. Delete the selected component(s)
6. Check that your design is correct
7. Close FuegoBPM Studio

8. Remove from repository the deleted file(s)
9. Close VSS

External Resources

To add the first external resource,

1. Open VSS
2. Get the latest version of the project
3. Open FuegoBPM Studio
4. Add the new external resource
5. Close FuegoBPM Studio
6. Add into VSS the *configurations.xml* file located under the *config* folder
7. Close VSS

To add an external resource,

1. Open VSS
2. Get the latest version of the project
3. Check out the *configurations.xml* file located under the *config* folder
4. Open FuegoBPM Studio
5. Add the new external resource(s)
6. Close FuegoBPM Studio

7. Check into VSS the *configurations.xml* file located under the *config* folder
8. Close VSS

To modify an external resource,

1. Open VSS
2. Get the latest version of the project
3. Check out the *configurations.xml* file located under the *config* folder
4. Open FuegoBPM Studio
5. Add the new external resource
6. Close FuegoBPM Studio
7. Check into VSS the *configurations.xml* file located under the *config* folder
8. Close VSS

To delete an external resource,

1. Open VSS
2. Get the latest version of the project
3. Check out the *configurations.xml* file located under the *config* folder
4. Open FuegoBPM Studio
5. Verify that no other process(es), procedure(s), or screen flow(s) uses the external resource(s) that you are trying to delete

6. Delete the selected external resource(s)
7. Close FuegoBPM Studio
8. Check into VSS the *configurations.xml* file located under the *config* folder
9. Close VSS

To delete the last external resource,

1. Open VSS
2. Get the latest version of the project
3. Open FuegoBPM Studio
4. Verify that no other process(es), procedure(s), or screen flow(s) uses the external resource(s) that you are trying to delete
5. Delete the selected external resource
6. Check that your design is correct
7. Close FuegoBPM Studio
8. Remove from VSS the *configurations.xml* file located under the *config* folder
9. Close VSS

Note



Please, refer to the section Version Control System to find information on how FuegoBPM Studio implements it.

Appendix B - IBM WebSphere 5.1 EJB

integration with Fuego

Introduction

This document will describe how FuegoBPM will integrate with Enterprise Java Beans (EJB) deployed on an IBM WebSphere 5.1 Application Server.

Configuring IBM WebSphere 5.1 to receive Fuego EJB Calls

In order for any of the Fuego products to invoke an EJB deployed in IBM WebSphere 5.1, we will need to setup and deploy a Web Application that will act as a

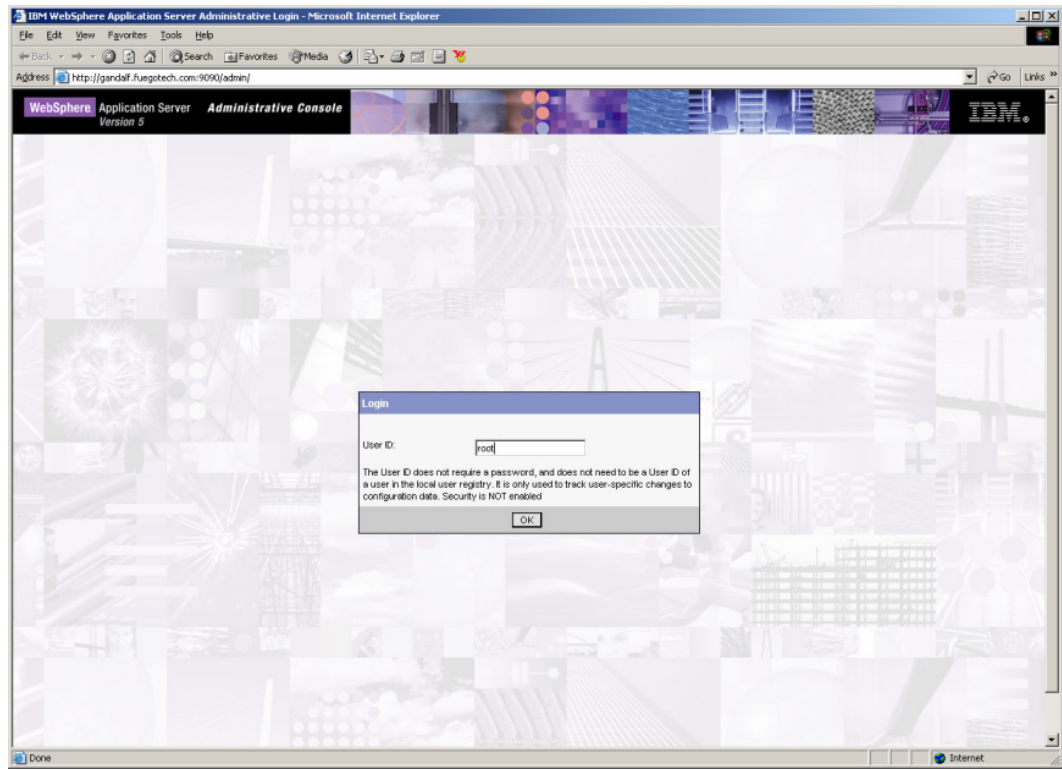
bridge to execute the EJB calls.

The Web Application to deploy comes in a form of a WAR file and is named *bsf-remote-server.war*. This WAR file can be obtained or downloaded from <http://forge.objectweb.org/projects/bsframework>. Fuego will also start distributing this jar file in Fuego distribution in the short term.

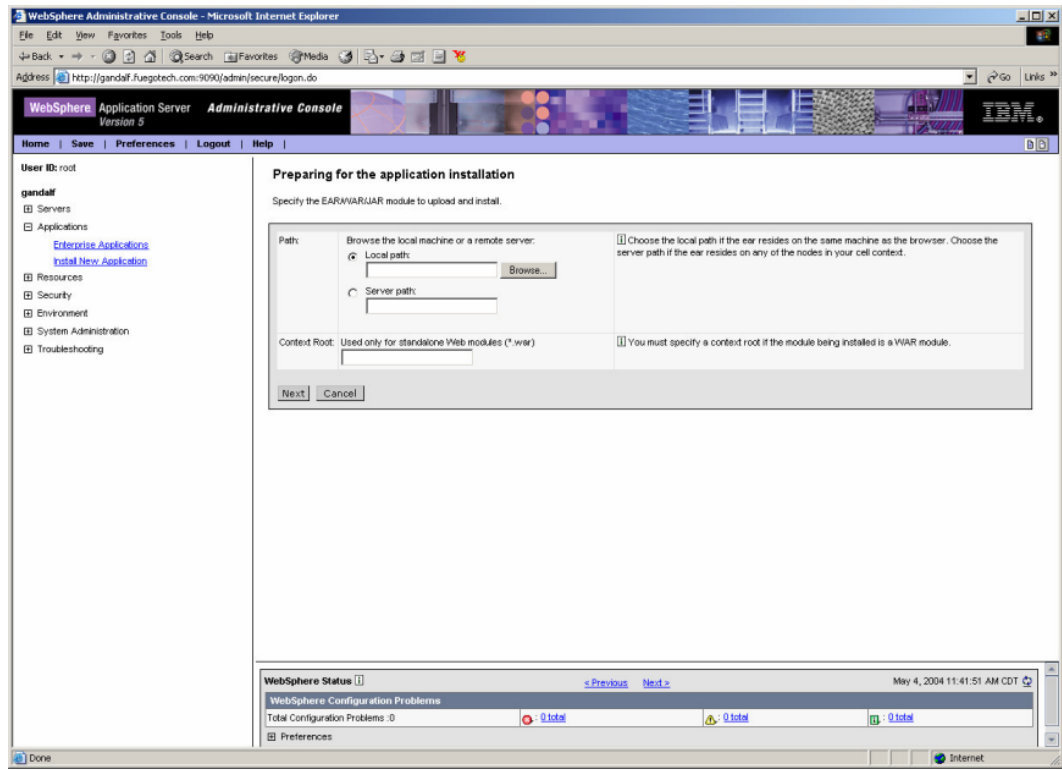
Deploying the Web Application in WebSphere 5.1

In order to deploy this Web Application in WebSphere 5.1, follow these instructions:

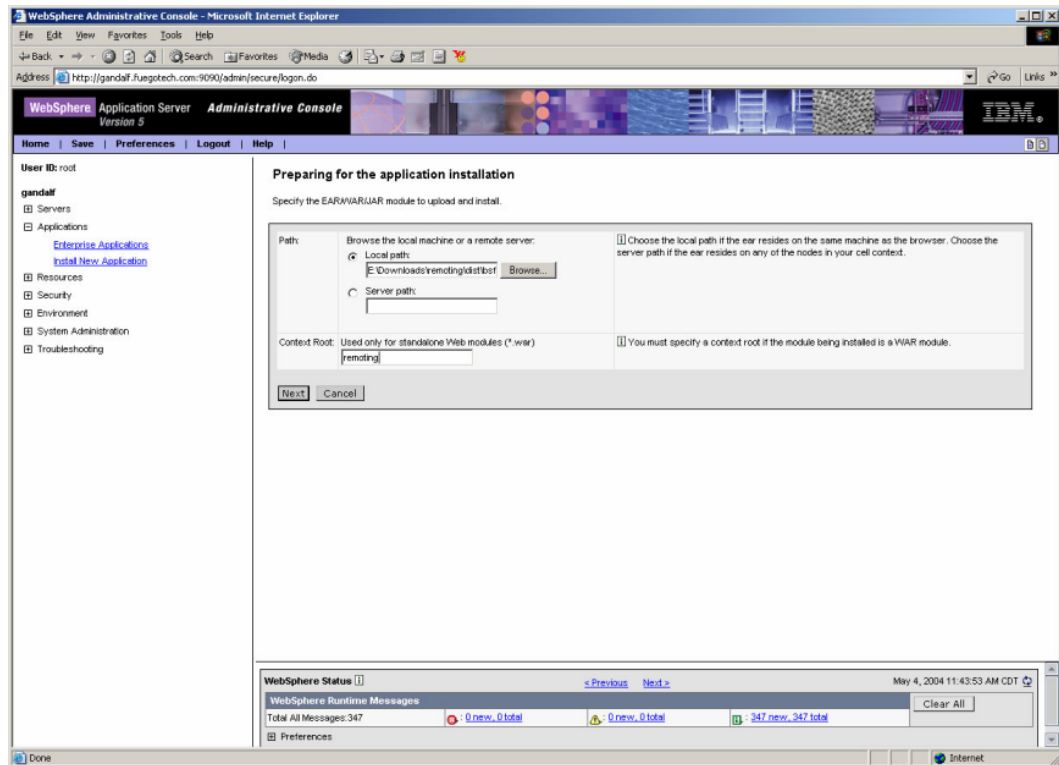
1. Log into the WebSphere Console



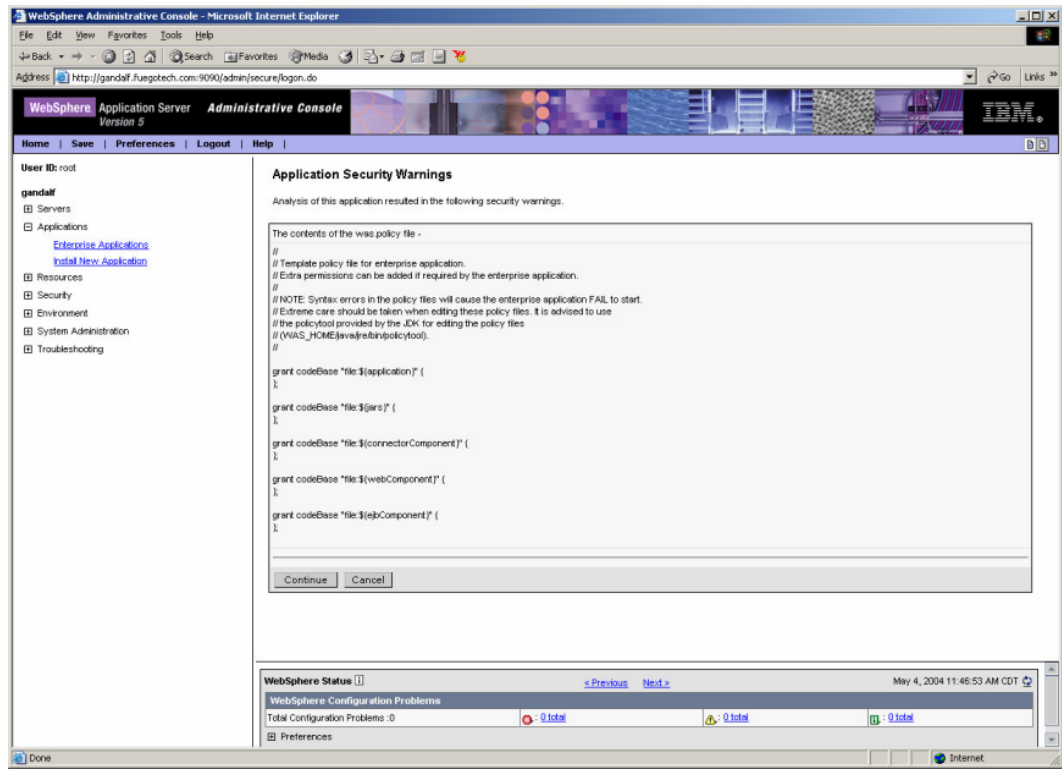
2. After successfully logging into WebSphere Console, click on the *Applications* link on the left panel and then on the *Install New Applications* link as shown below.



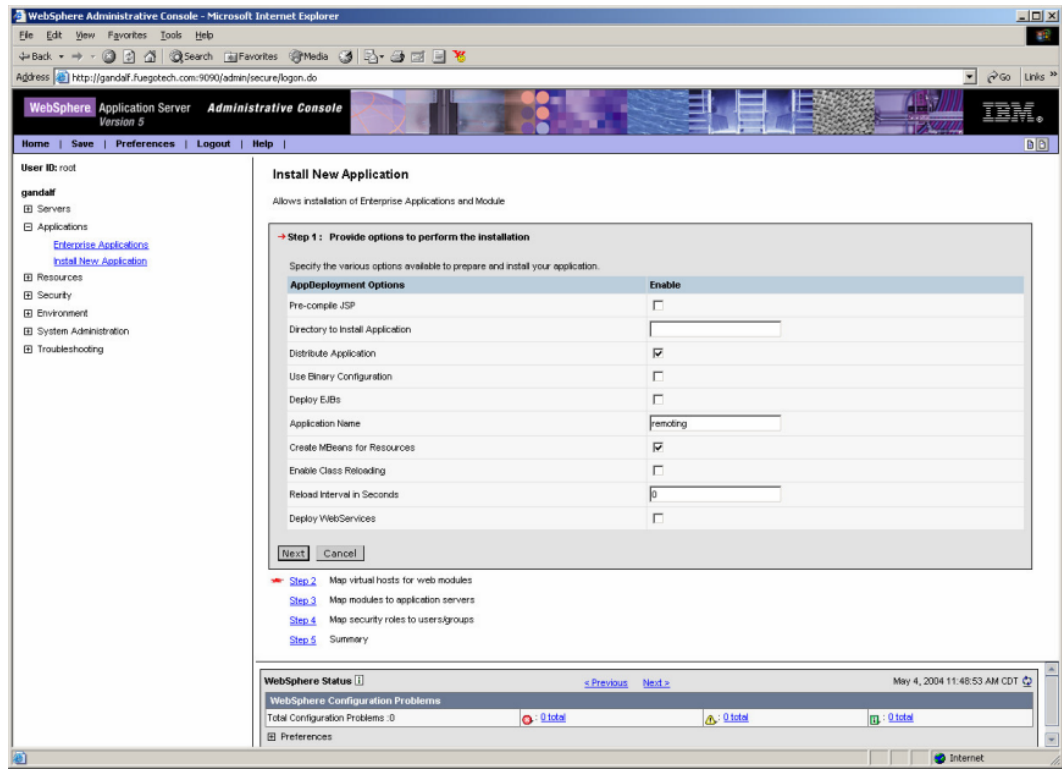
3. Click on the *Browse* button and search for the war file mentioned below. The name of the Web Application file name is: *bsf-remote-server.war*. Then you will need to enter the string literal *remoting* on the *Context Root* text field as shown below. Then click on the *Next* button to proceed.



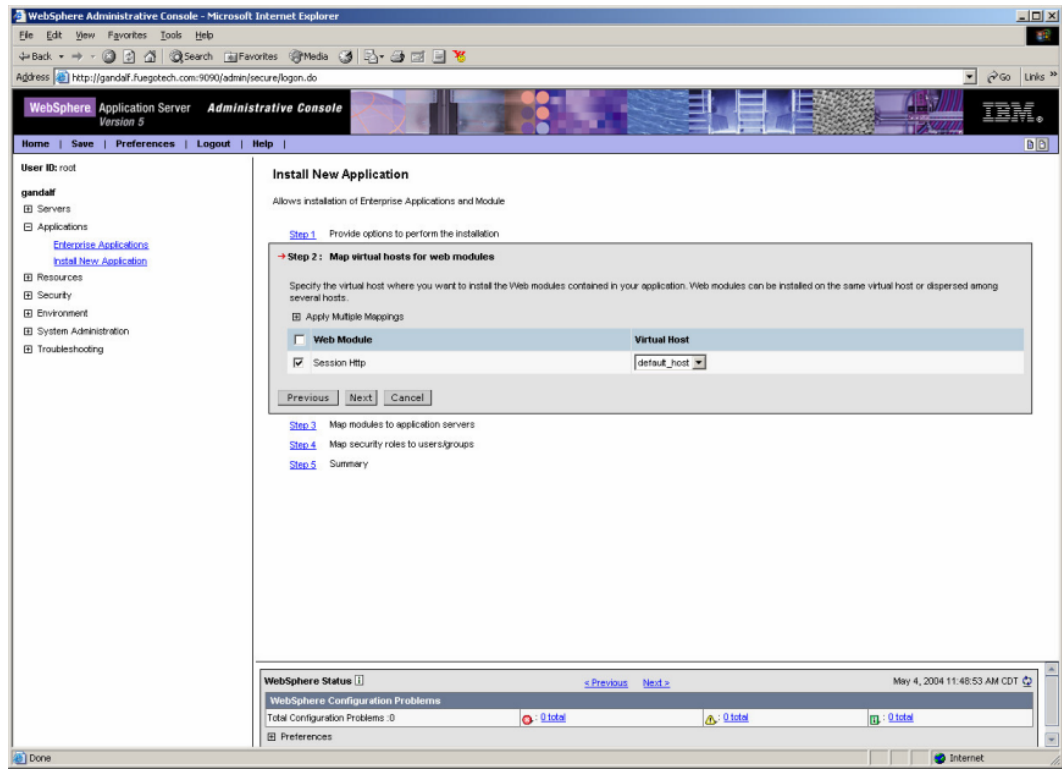
4. Next, we will need to provide values for the application bindings. Unless something specific is requested by the Application Server Administrator, we will follow the proposed default values as shown below. To proceed click on the *Next* button.



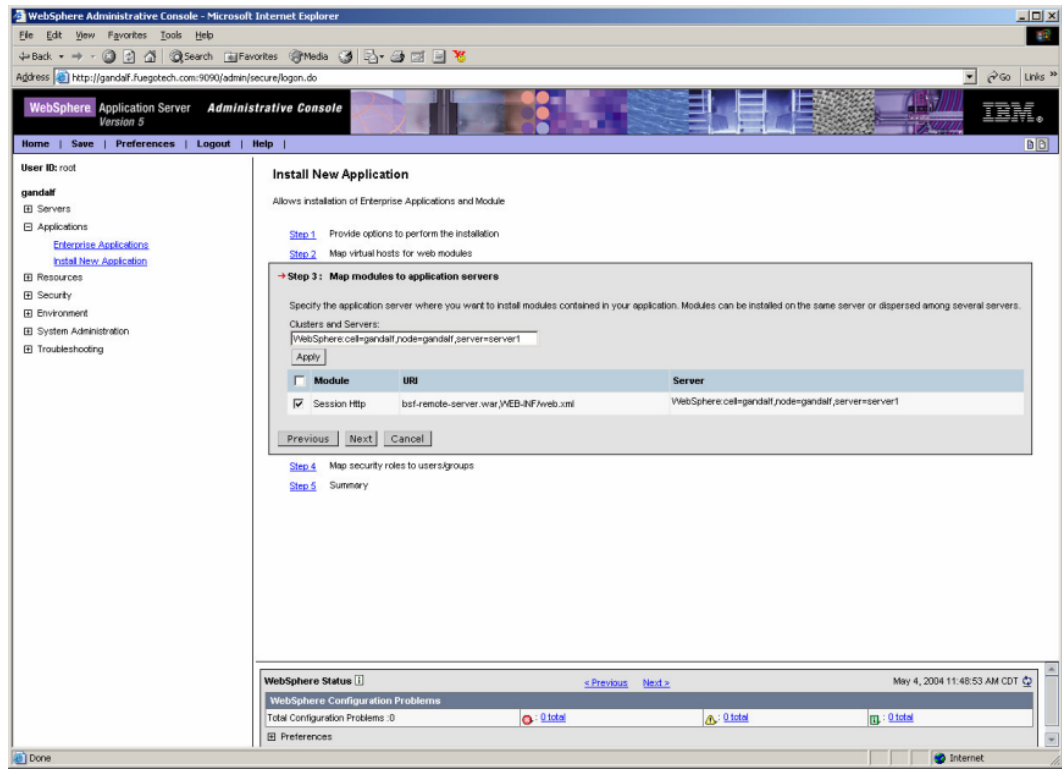
- The panel above shows the Application security. Unless something specific is required by the Application Server Administrator, we will accept these defaults and click on the *Continue* button to complete the Web Application deployment.



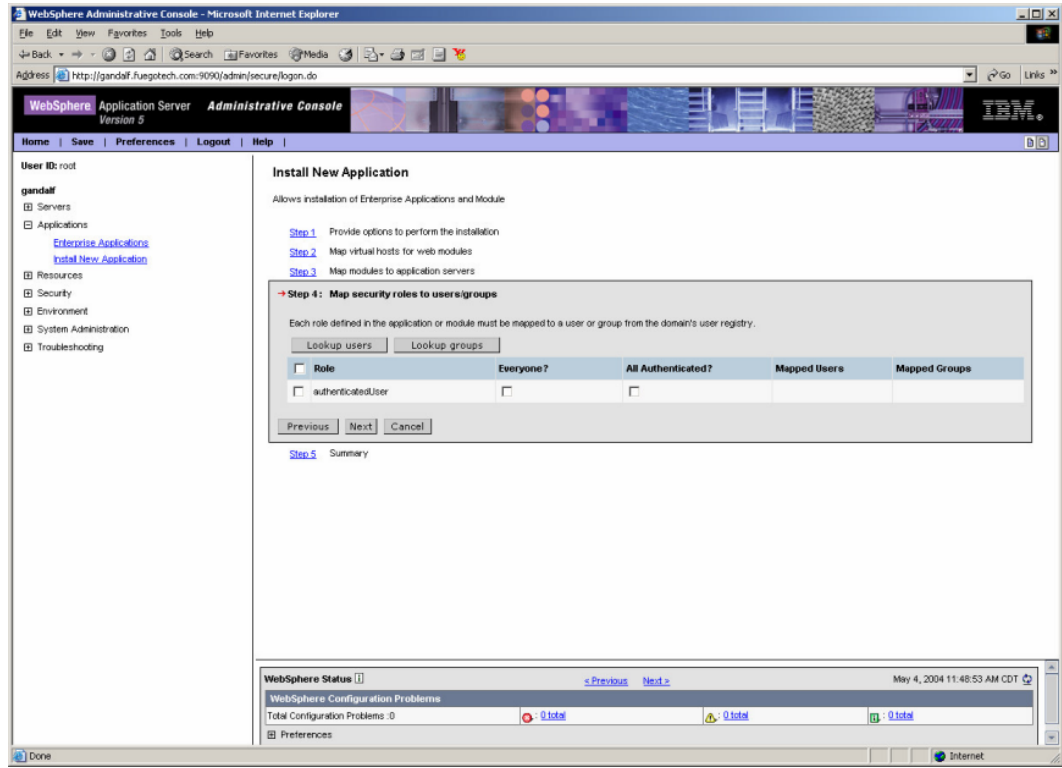
- Accept all defaults in this panel and enter *remoting* as the Application Name. Then click on the *Next* button.



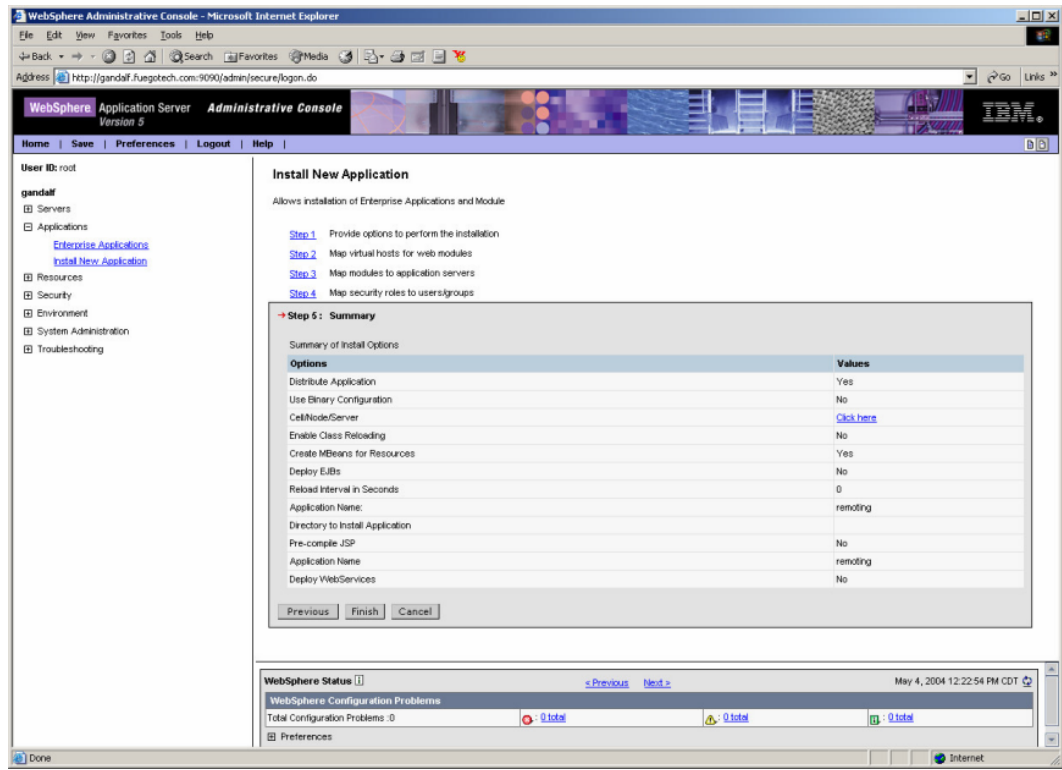
7. Select the *Session Http* checkbox and click on the *Next* button to continue.



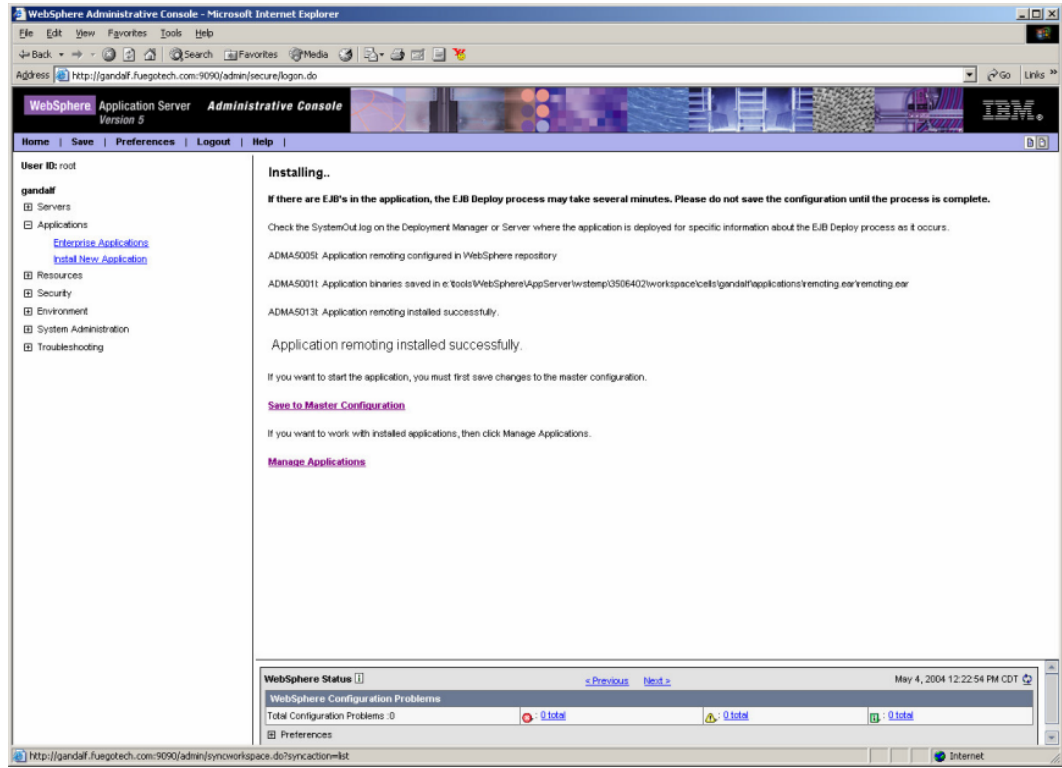
8. Select the *Session Http* checkbox and click *Next* to continue.



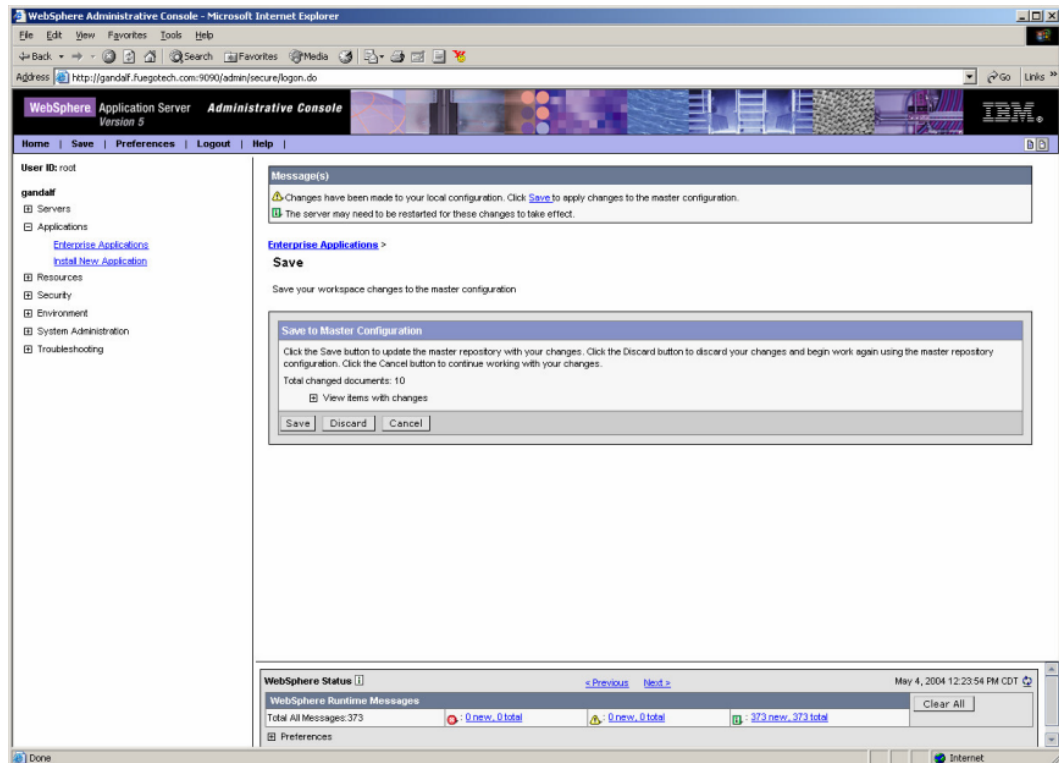
9. Unless the Web Application requires authentication, you can click on the *Next* button. For our deployment case, we will assume no authentication is required for this Web Application.



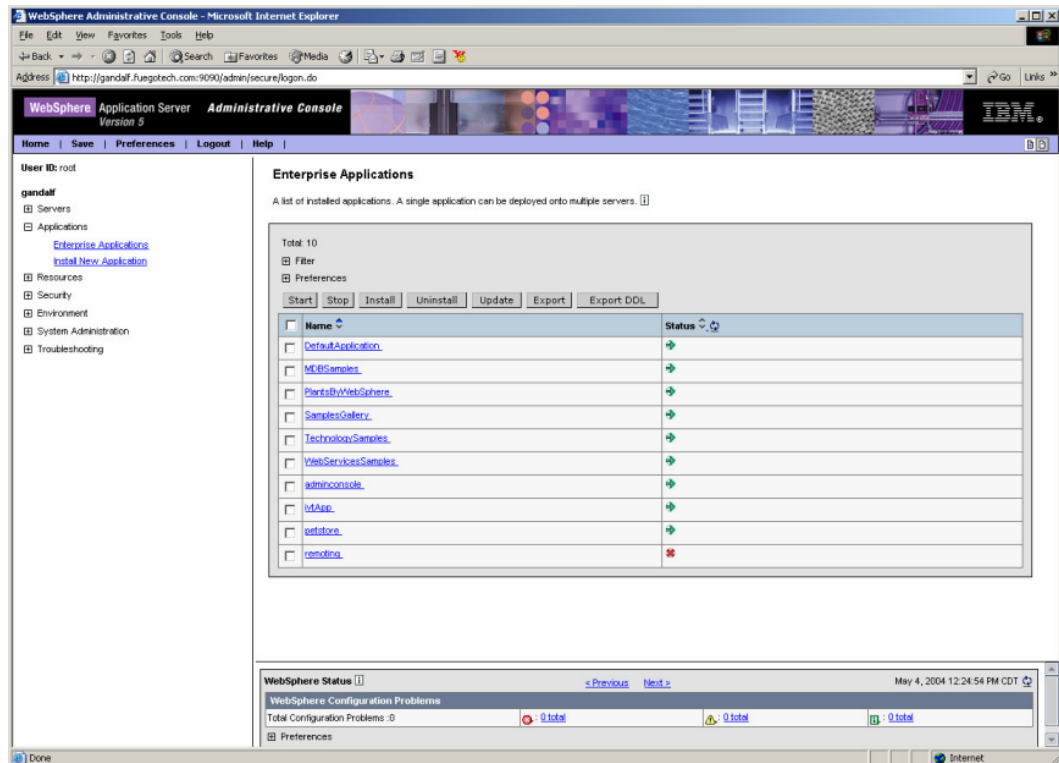
- Now, you will need to click on the *Finish* button to complete the Web Application deployment.



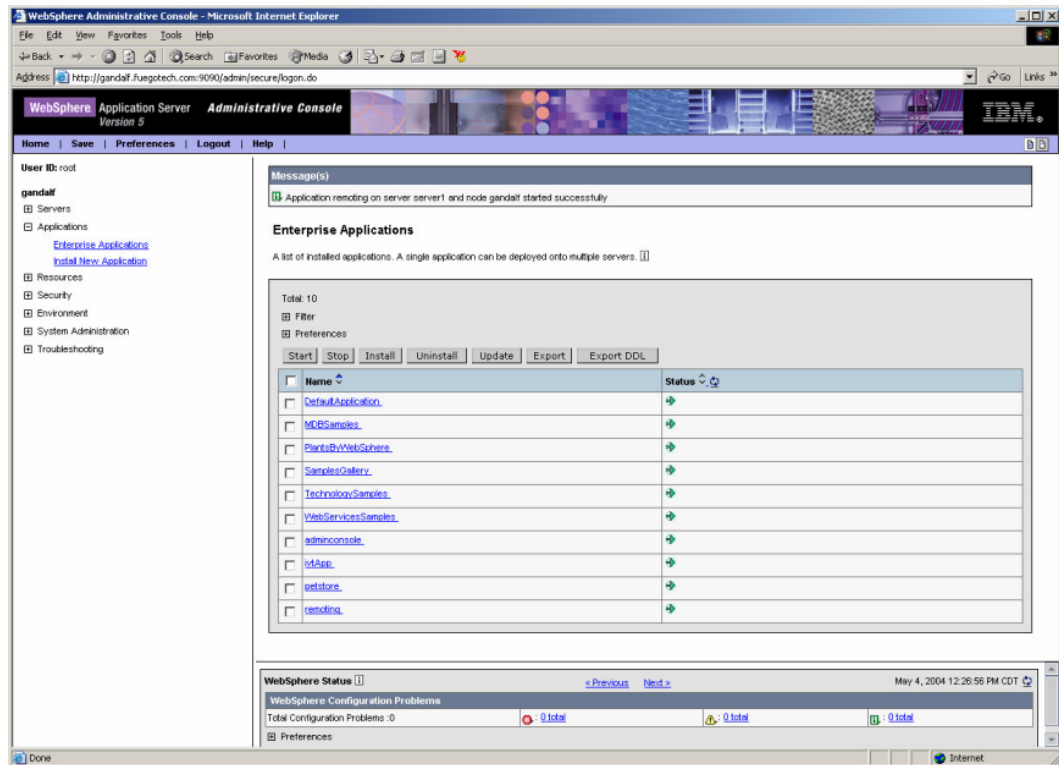
11. Once the Web Application installation has completed successfully, you will need to click on the *Save to Master Configuration* link to save the configuration.



12. Click on the *Save* button to continue. Once the save operation finished, you will need to click on the *Enterprise Applications* link on the left panel. Once there you will see the *remoting* Enterprise Application listed among all deployed applications as shown below.



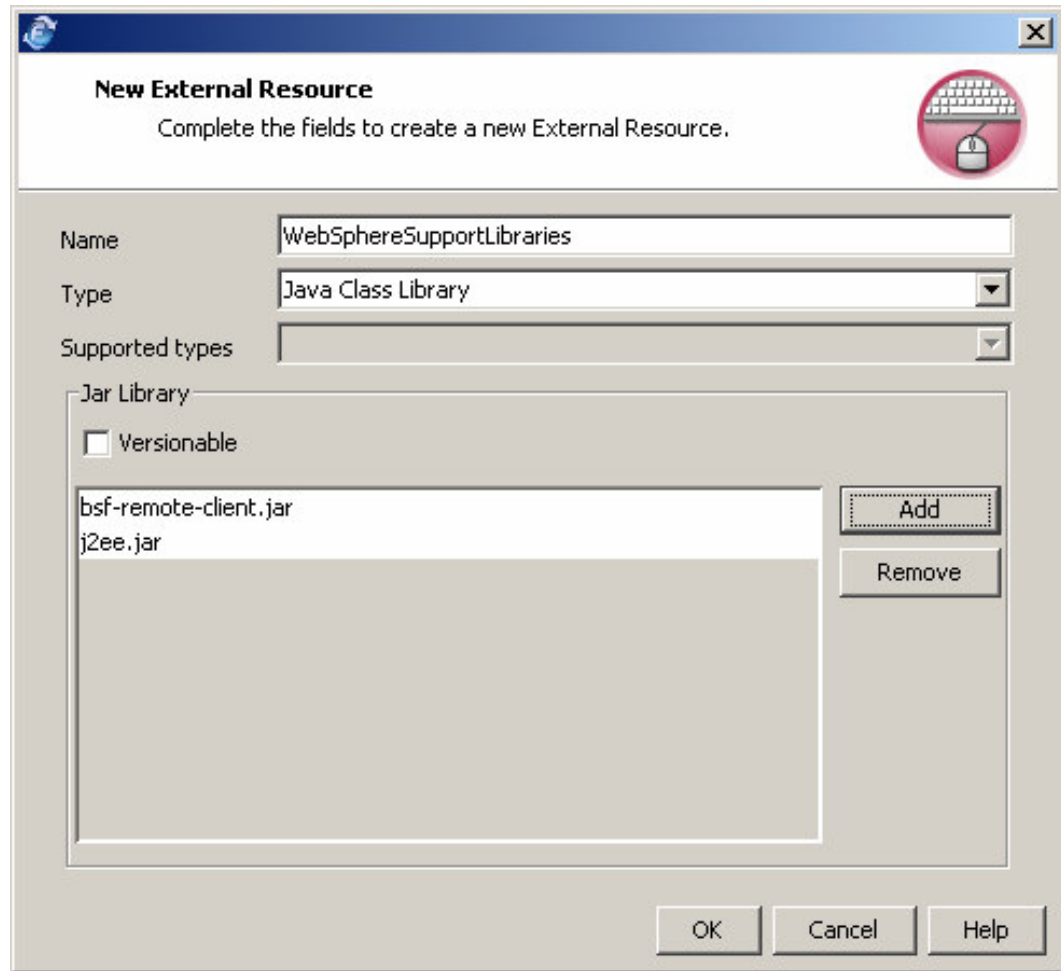
13. Now, we will need to start the *remoting* Web Application. For this purpose, you will need to select the *remoting* checkbox and then click on the *Start* button on the panel toolbar. After a successful start of the Web Application, you should see a green arrow as with the rest of the Web Applications as shown below.



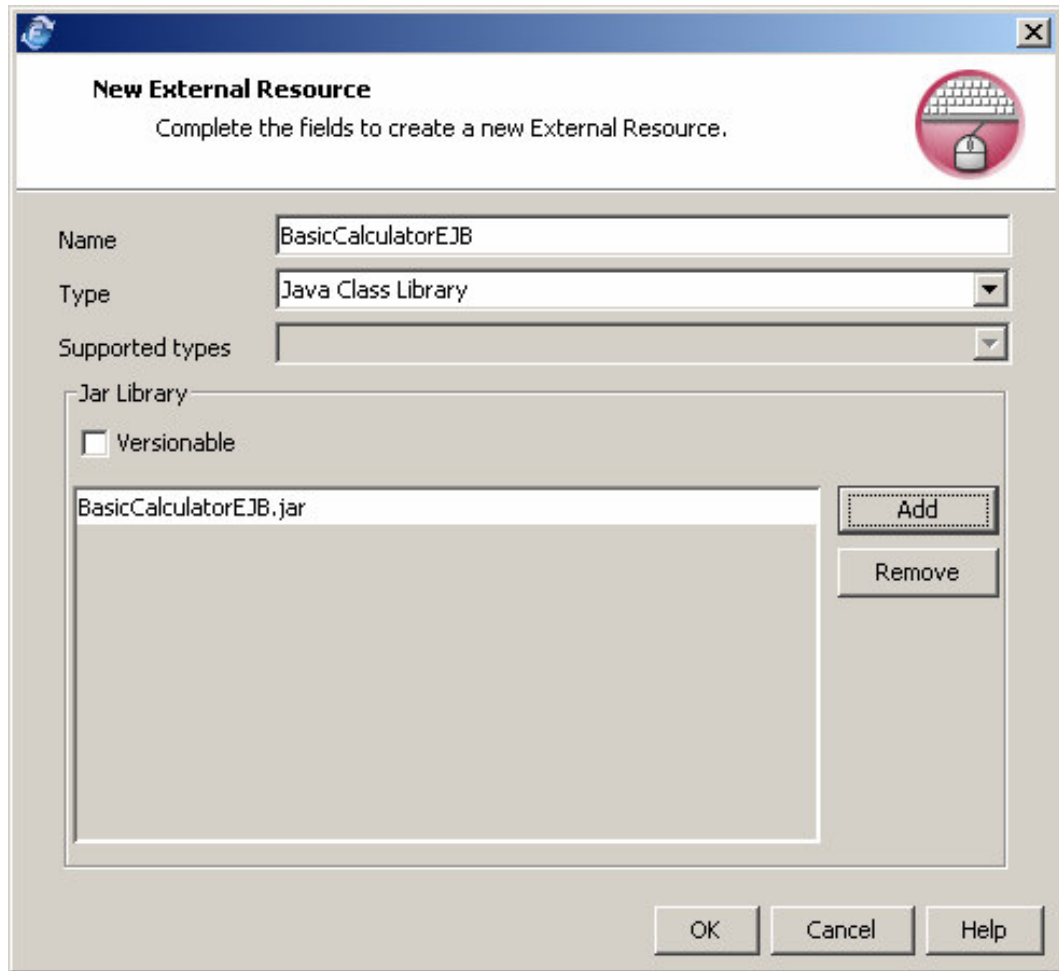
Setting up FuegoBPM Studio

In order to catalogue the EJBs in the FuegoBPM Studio, you will need to follow these instructions as directed below:

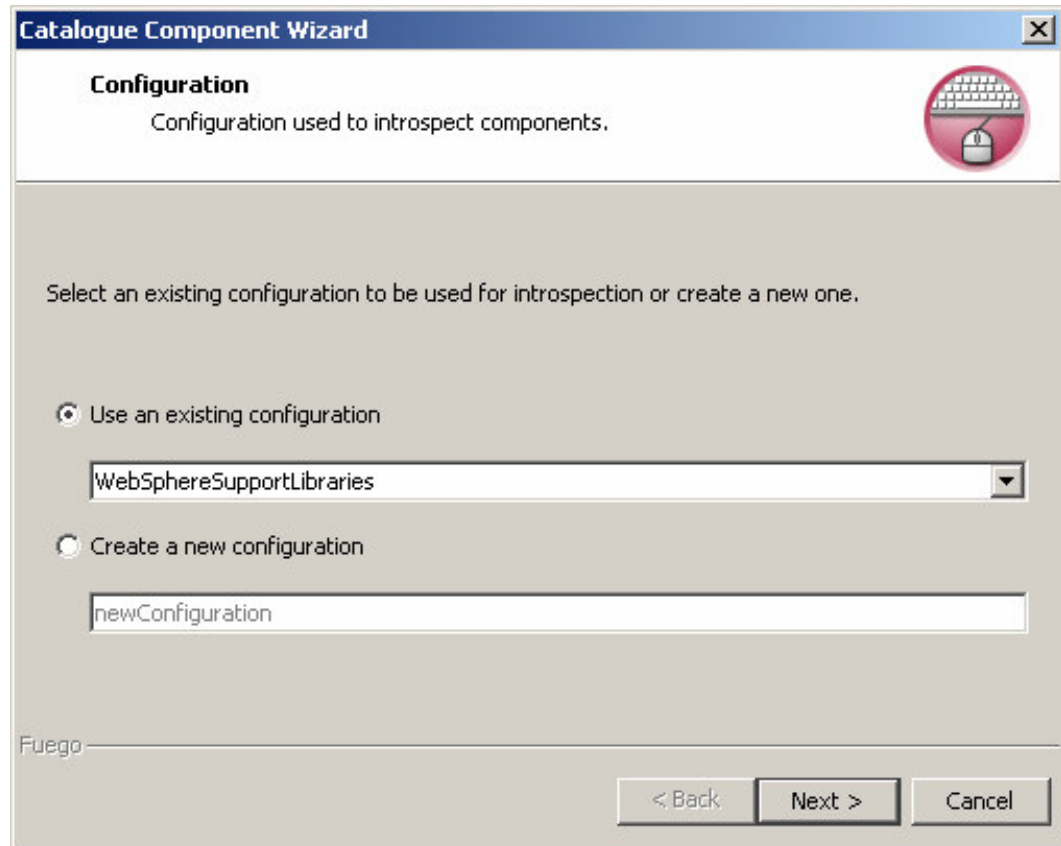
1. Launch the FuegoBPM Studio and move to the External Resources to catalogue a new Java Library. The dialog below shows what jar files need to be included in for the FuegoBPM Studio to successfully connect to the IBM WebSphere 5.1 Application Server to invoke EJBs.



2. The *j2ee.jar* can be obtained from the IBM WebSphere 5.1 Application Server distribution. The *bsf-remote-client.jar* jar file can be obtained and downloaded from <http://forge.objectweb.org/projects/bsframework>. Fuego will also start distributing this jar file in Fuego distribution in the short term. Once finished, you will need to click on the *Ok* button.
3. The next step is to catalogue the jar file that contains the EJB Bean. This class needs to be included in the jar file for the invocation to work correctly. The dialog below shows how to catalogue the *BasicCalculator* Enterprise Java Bean Example that comes out of the box with the IBM WebSphere 5.1 Application Server distribution. This jar can be found in the samples folder.



4. Although you can create a different Java Class Library external resource for each Enterprise Java Bean, it may be convenient to define one Java Class Library External resource where all the needed JARs containing the Enterprise Java Beans are catalogued. If you are going to catalogue more than one Enterprise Java Bean JAR file, you may want to change the *Name* of the Java Class Library External Resource.
5. Once we have finished cataloguing the Java Libraries cataloguing, we will need to catalogue the needed Java Classes from these libraries. For this purpose, we will first introspect the Java Classes to connect to the IBM WebSphere 5.1 Application Server as described by the screenshot sequence shown below:



The screenshot shows a Windows-style dialog box titled "Catalogue Component Wizard". The "Configuration" tab is active, with a sub-header "Configuration" and a description "Configuration used to introspect components." in the top section. A keyboard and mouse icon is in the top right corner. The main area contains the instruction "Select an existing configuration to be used for introspection or create a new one." Below this are two radio button options: "Use an existing configuration" (which is selected) and "Create a new configuration". Under the first option is a dropdown menu showing "WebSphereSupportLibraries". Under the second option is a text input field containing "newConfiguration". At the bottom left is the "Fuego" logo. At the bottom right are three buttons: "< Back", "Next >", and "Cancel".

Catalogue Component Wizard

Configuration
Configuration used to introspect components.

Select an existing configuration to be used for introspection or create a new one.

☒ Use an existing configuration

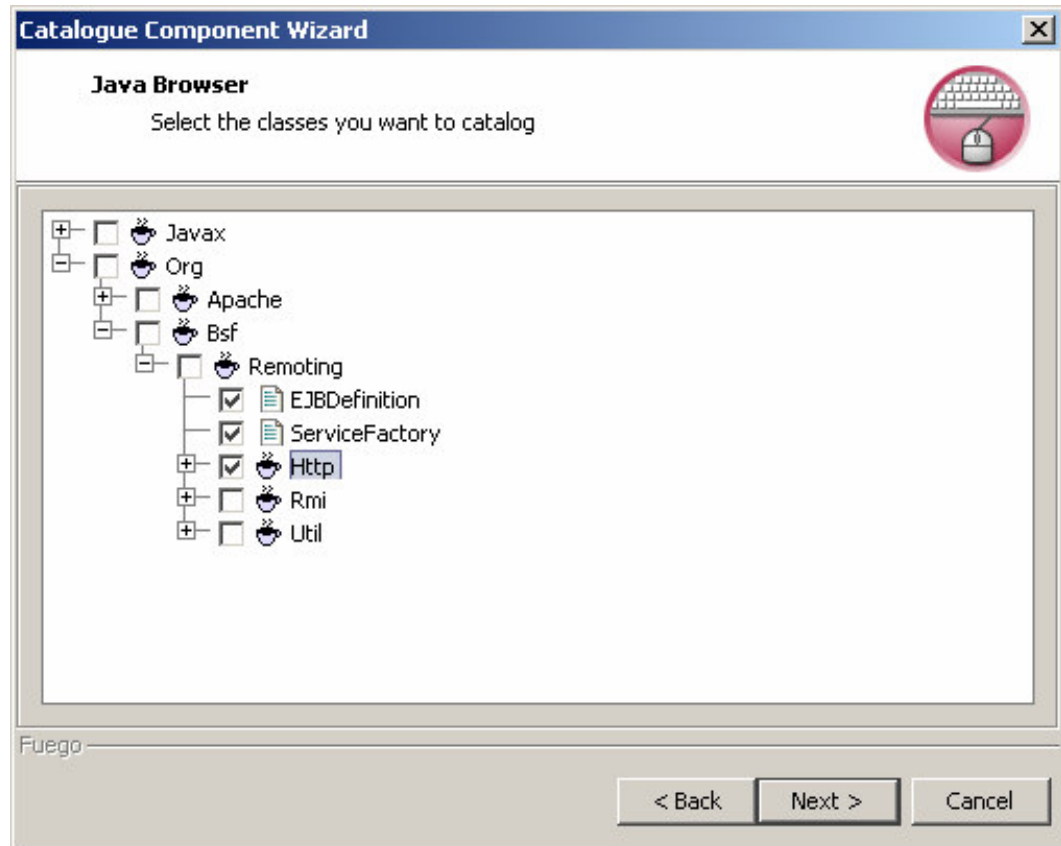
WebSphereSupportLibraries

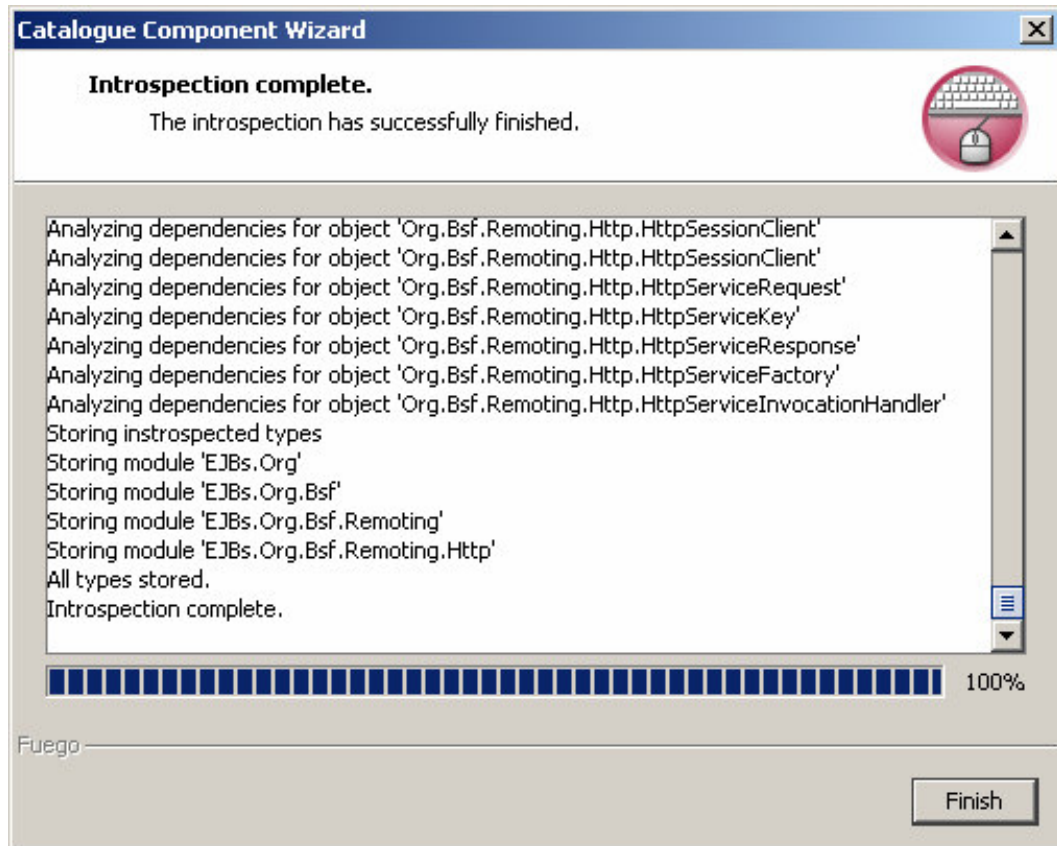
☐ Create a new configuration

newConfiguration

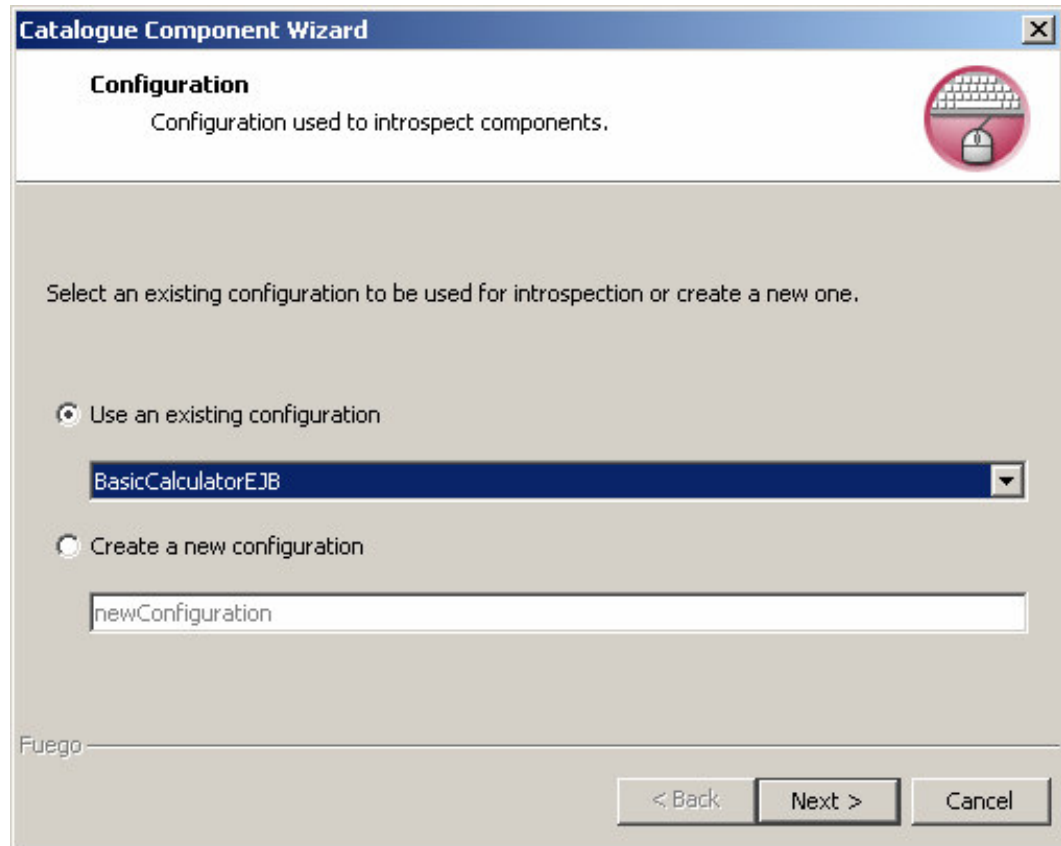
Fuego

< Back Next > Cancel





6. Then, we will need to catalogue the Enterprise Java Bean class that is present in the Other Java Class Library. Follow this sequence of screenshots to perform the Java Class introspection:



The image shows a Windows-style dialog box titled "Catalogue Component Wizard". It has a blue title bar with a close button (X) in the top right corner. The main area is divided into two sections. The top section, titled "Configuration", has a subtitle "Configuration used to introspect components." and a red circular icon with a keyboard and mouse. The bottom section contains the instruction "Select an existing configuration to be used for introspection or create a new one." Below this, there are two radio buttons. The first, "Use an existing configuration", is selected. Below it is a dropdown menu showing "BasicCalculatorEJB". The second radio button, "Create a new configuration", is unselected. Below it is a text input field containing "newConfiguration". At the bottom left, the text "Fuego" is visible. At the bottom right, there are three buttons: "< Back", "Next >", and "Cancel".

Catalogue Component Wizard

Configuration
Configuration used to introspect components.

Select an existing configuration to be used for introspection or create a new one.

☒ Use an existing configuration

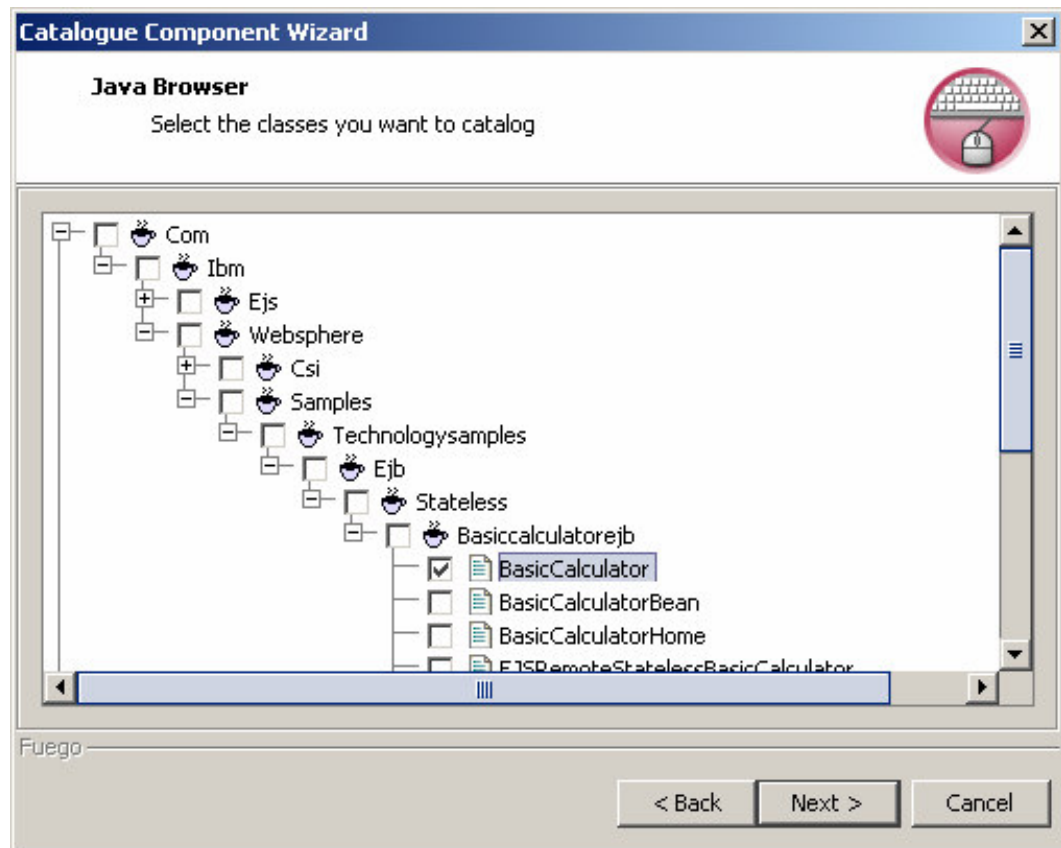
BasicCalculatorEJB

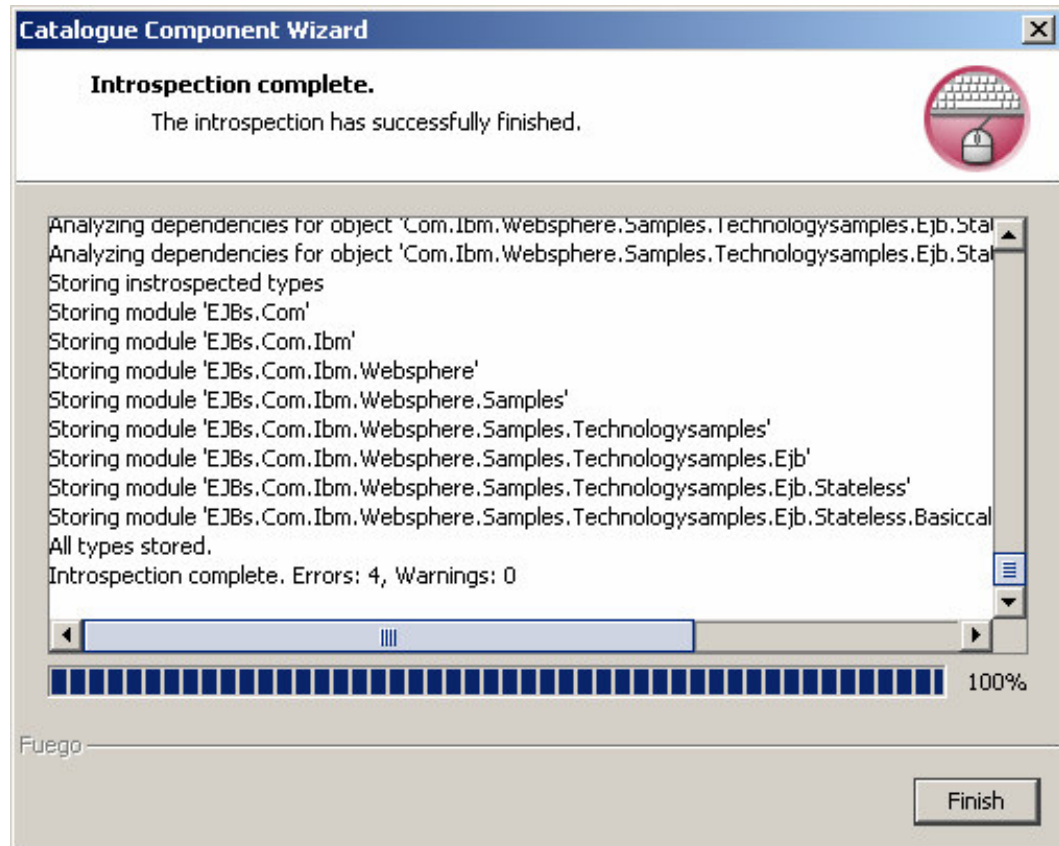
☐ Create a new configuration

newConfiguration

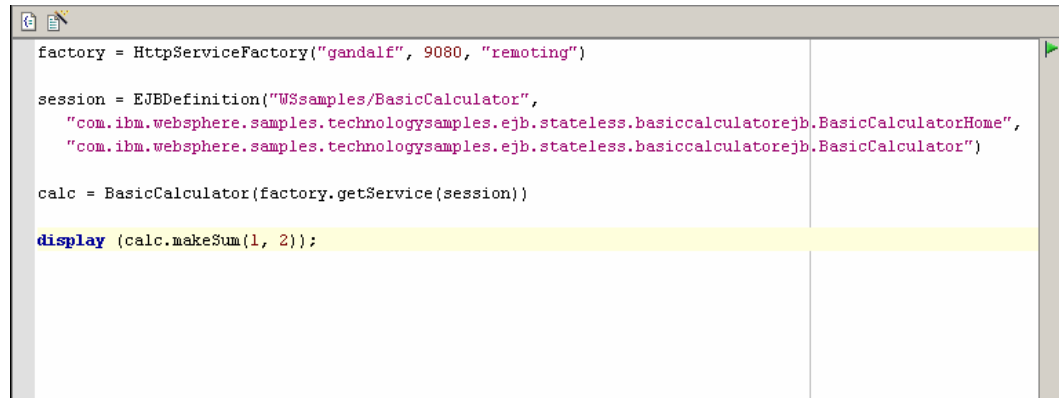
Fuego

< Back Next > Cancel





7. You can ignore the error messages shown at the end of this introspection since they are not going to impact the invocation of Enterprise Java Beans. Now that we finished the introspection of needed EJB and invocation framework class, we will create a Fuego Object that will show what FBL needs to be implemented to do the actual Enterprise Java Bean invocation call. The Fuego Object is named *WSEJB* and it will have a *test* method. The implementation for this method is depicted below:



```
factory = HttpServiceFactory("gandalf", 9080, "remoting")

session = EJBDefinition("WSsamples/BasicCalculator",
    "com.ibm.websphere.samples.technologysamples.ejb.stateless.basiccalculatorejb.BasicCalculatorHome",
    "com.ibm.websphere.samples.technologysamples.ejb.stateless.basiccalculatorejb.BasicCalculator")

calc = BasicCalculator(factory.getService(session))

display {calc.makeSum(1, 2)};
```

8. Now, we will describe what the arguments for each Object means:

a. HttpServiceFactory Object.

- i. 1st Argument: It is the host where the IBM WebSphere 5.1 Application Server resides.
- ii. 2nd Argument: It is the IBM WebSphere 5.1 Application Server incoming HTTP Port. By default it is *9080*.
- iii. 3rd Argument: It is the previously installed Web Application name. The instructions above, installed the Web Application under the *remoting* name, so in this case it will be this String literal.

b. EJBDefinition Object

- i. 1st Argument: It is the Enterprise Java Bean Deployment Name. This name can be found on the WebSphere 5.1 Application Server Web Console. For the example we are using, it is *WSsamples/BasicCalculator*.
- ii. 2nd Argument: It is the name of the Enterprise Java Bean Home class name qualified with the java package

name. See in the example screenshot above.

- iii. 3rd Argument: It is the name of the Enterprise Java Bean class name qualified with the java package name. See in the example screenshot above.

9. The third FBL instruction is the actual creation of the Enterprise Java Bean reference with the EJB on the Application Server. This reference is created by casting to the Enterprise Java Bean class, the returning value of the method invocation *factory.getService(session)*.
10. The fourth FBL instruction is invoking the Enterprise Java Bean method exposed through the Enterprise Java Bean Interface and displaying the resulting value.
11. After this we are ready to do the Enterprise Java Bean Invocation.

IBM WebSphere 5.1 Application Server Classloader

As the *remoting* Web Application is deployed not as part of the Enterprise Java Bean Enterprise Application Package, it will not share the same classloader as these deployed applications. As they are not sharing the classloader, the *remoting* Web Application will not be able to successfully invoke the Enterprise Java Bean when the request comes from Fuego. For this reason, there are 2 alternatives proposed to make the Enterprise Java Bean Classes available to the *remoting* Web Application. These are exposed below.

1. Copying the Enterprise Java Bean JAR file into the Application Server *lib/ext* directory.

2. Before deploying the `bsf-remote-server.jar` you can:
 - a. Open the WAR file into a temporary directory.
 - b. Copy the Enterprise Java Bean JAR files into the Web Application WEBINF/lib directory.
 - c. Bundle the WAR file again. This can be done using WinZip and then renaming the file with the *war* extension.
 - d. Deploy the WAR file again.

Appendix C - JMS Integration Examples

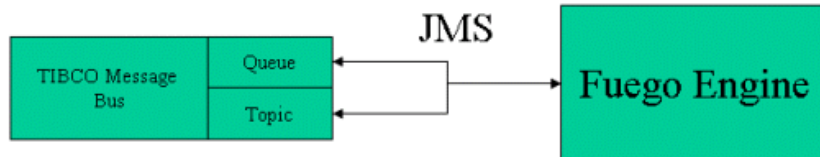
Integrating FuegoBPM and TIBCO

This appendix illustrates the integration between TIBCO and FuegoBPM. It describes the underlying architecture for connecting both products as well as describing the integration efforts to integrate TIBCO to participate in FuegoBPM Business Processes managed and orchestrated by a FuegoBPM Engine using TIBCO's JMS Implementation.

Architecture of TIBCO – Fuego Integration

The following screenshot describe the underlying architecture that is being used to integrate Fuego with TIBCO consuming and sending messages from and to a TIBCO Queue or Topic using the standard Java API for Messaging JMS (Java Messaging Service).

TIBCO Fuego Integration Architecture Diagram



Business Processes deployed in a Fuego Engine can consume and send messages from and to TIBCO Queues or Topics.

Creating the TIBCO Queues in TIBCO EMS

This section shows how to create a Queue, its ConnectionFactory and JNDI Name in TIBCO so that FuegoBPM can connect and integrate with it. This reflects the usage of TIBCO's *tibjmsadmin* command distributed with TIBCO's EMS distribution.

```
tcp://localhost:7222> create factory
    FuegoSampleXAConnectionFactory
        xaqueue url=tcp://192.168.0.91:7222
QueueConnectionFactory 'FuegoSampleXAConnectionFactory'
has been created
tcp://localhost:7222> create queue FuegoSampleQueue
Queue 'FuegoSampleQueue' has been created
tcp://localhost:7222> create jndiname FuegoSampleQueue
    queue FuegoSampleQueue
JNDI name 'FuegoSampleQueue' has been created
```

Integrating TIBCO APIs in Fuego Studio

After the TIBCO Queue, Connection Factory and JNDI Reference Name have been defined in TIBCO, we can start working on the effort within FuegoBPM to integrate and communicate with TIBCO's

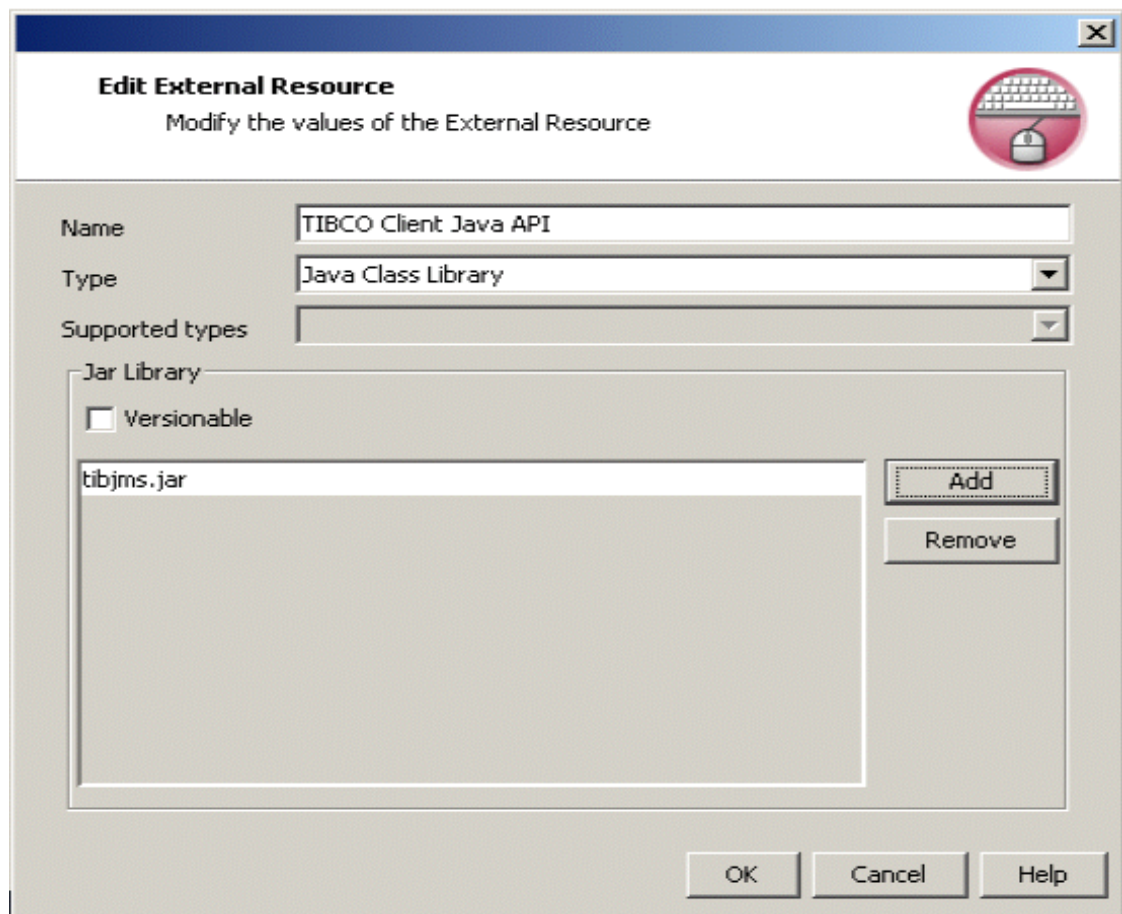
Queue to exchange any kind of messages.

Defining the External Resources

The first thing to do in FuegoBPM is to define external resources that will help us to integrate with TIBCO EMS infrastructure.

Java Class Libraries

The first external resource to add is a *Java Class Library* that includes the client side JMS Implementation for TIBCO's Messaging Infrastructure. This **tibjms.jar** jar file can be found on TIBCO's distribution under the *\$TIBCO/clients/java* directory. You have a snapshot of how this resource should look like.



JMS Messaging Service

Once the J2EE Application Server resource has been added, the

developer will need to specify the JNDI properties and parameters to find to connect to TIBCO EMS. The developer will need to specify the J2EE Server that denotes the location of TIBCO and also the kind of resource to connect to (QUEUE or TOPIC). The developer will also need to specify the resource JNDI name (in our case the Queue JNDI Name) as well as the ConnectionFactory specified in TIBCO as shown before.

Edit External Resource
Modify the values of the External Resource

Name: TIBCO JMS Queue

Type: JMS Messaging Service

Supported types:

Details:

J2EE: TIBCO EMS Server

Destination Type: QUEUE

Lookup Name: FuegoSampleQueue

Connection Factory Lookup Name: FuegoSampleXAConnectionF

JMS Listener Port: (only for WebSphere)

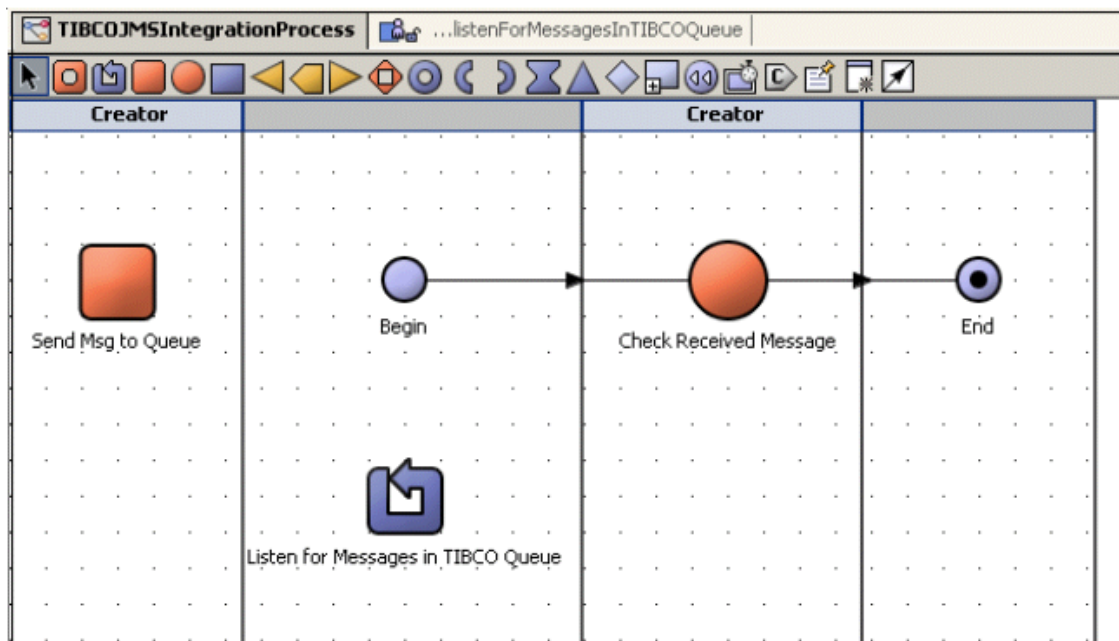
OK Cancel Help

Creating a Business Process that uses the resources in External Resources to connect to TIBCO

After all the external resources have been created, we can define a business process that can consume and send messages from and to a JMS Queue or Topic.

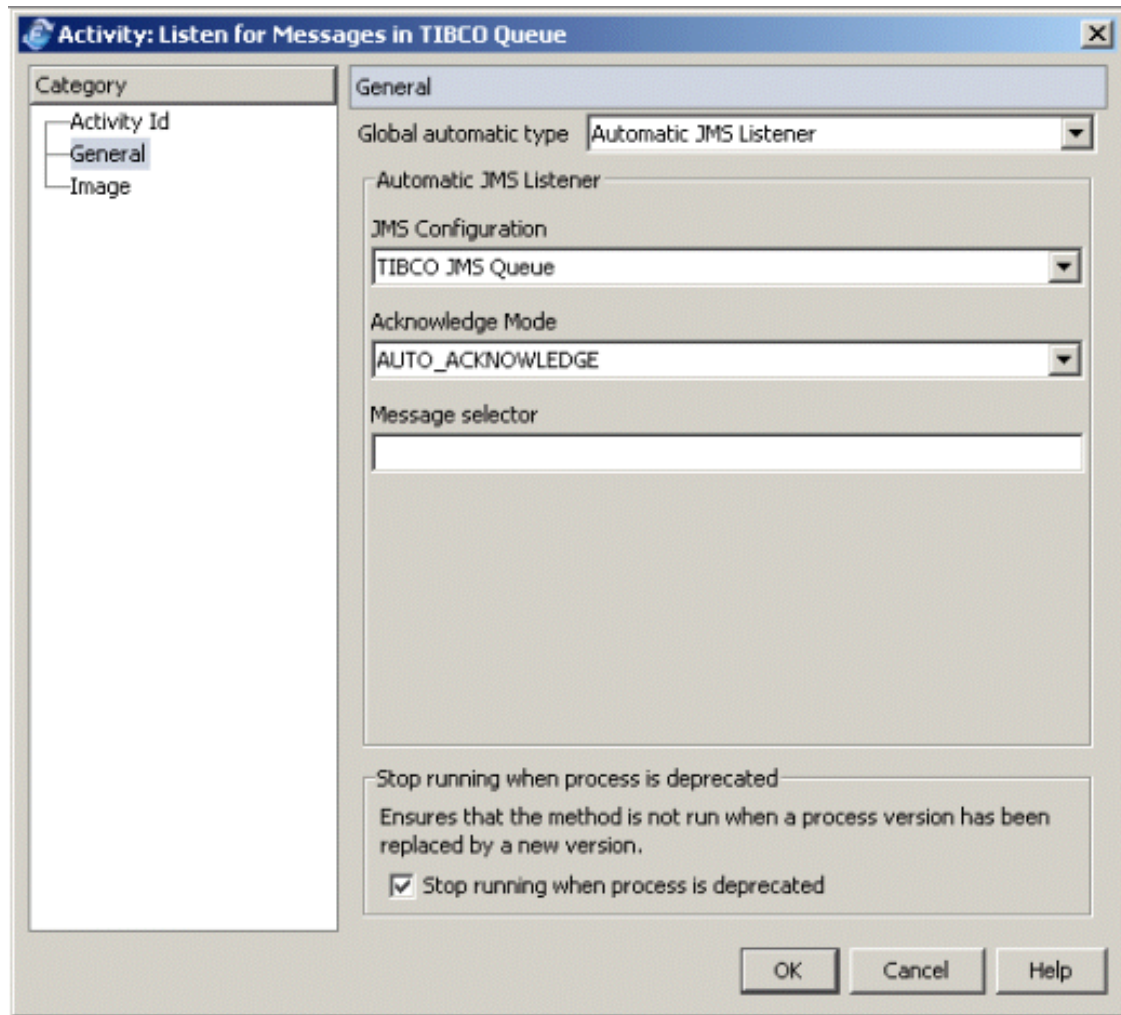
The sample process

This sample process has a Global activity *Send Msg to Queue* that will send a message to the JMS Queue. This message will be picked by the Global Automatic Activity *Listen for Messages in TIBCO Queue* and create an instance into the process. The instance then will be waiting on the *Check Received Message* interactive activity to verify that the message could be send and received from the JMS Queue deployed on TIBCO.



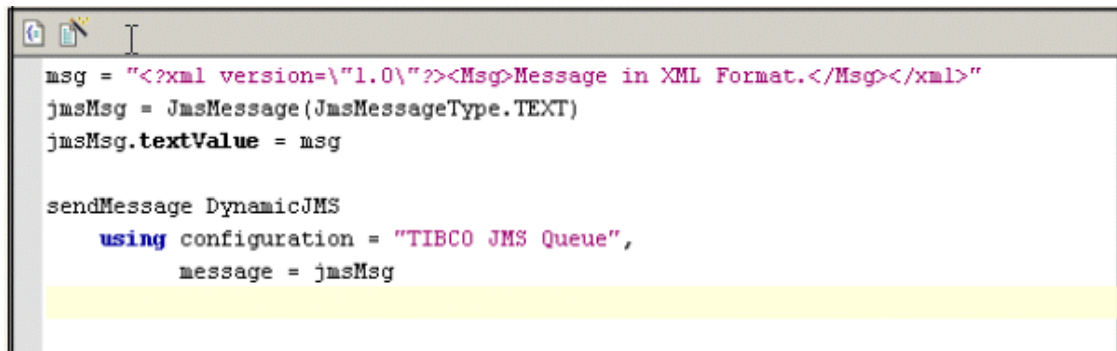
Properties of the Global Automatic Activity listening for JMS Messages

The following screenshot depicts the properties of the Global Automatic activity that was defined of type "Automatic JMS Listener".



Code Snippet for sending Messages to a TIBCO Queue

The following screenshot depicts the code needed to send a message to the JMS Queue deployed in TIBCO. This is the code for the "Send Msg to Queue" Global activity.

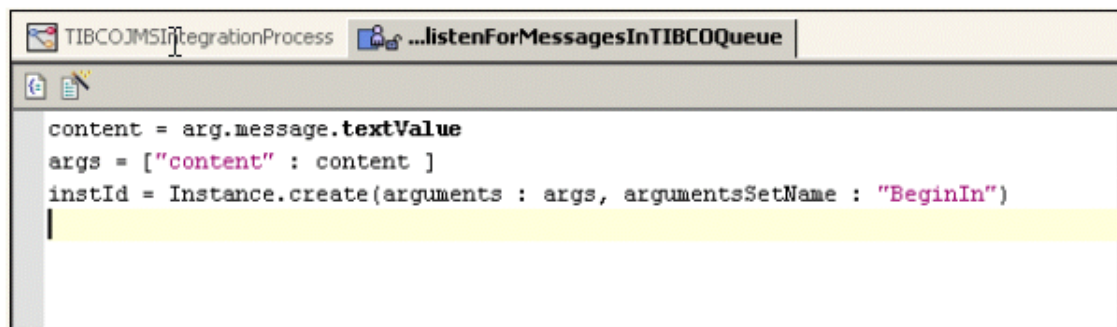


```
msg = "<?xml version='1.0'?'><Msg>Message in XML Format.</Msg></xml>"
jmsMsg = JmsMessage(JmsMessageType.TEXT)
jmsMsg.textValue = msg

sendMessage DynamicJMS
    using configuration = "TIBCO JMS Queue",
        message = jmsMsg
```

Code Snippet for receiving a Message from a TIBCO Queue

The code below is the implementation of the Global Automatic activity that is executed when a message is picked and retrieved from the Queue.



```
TIBCOJMSIntegrationProcess ...listenForMessagesInTIBCOQueue

content = arg.message.textValue
args = ["content" : content ]
instId = Instance.create(arguments : args, argumentsSetName : "BeginIn")
```

Integrating FuegoBPM and IBM MQSeries

This appendix describes the steps to configure the MQ Series Queue used by FuegoBPM as well as the steps for configuring FuegoBPM to integrate with these IBM MQSeries resources. In a nutshell, the integration described in this document is through JMS (Java Messaging Service).

Creating the Queue Manager and Queue objects in MQSeries

IBM MQ Series needs a Queue Manager where MQ Series Queues are deployed. For this reason, we will start with the definition of the Queue Manager and then continue with the definition of the MQ Series

Queues to be deployed on this Queue Manager that will manage them.

The approach taken in this document uses IBM MQ Series command line to create the objects managed by MQ Series. This does not limit the creation of these resources with graphical interfaces provided by Windows versions of MQ Series.

Creating the Queue Manager

In our example, we will create a Queue Manager called "QMFirego". If a Queue Manager has already been defined and you want to utilize this one, you may skip this step. However, this Queue Manager name as a reference through out the document.

The command line to create the Queue Manager follows:

```
# crtmqm QMFirego
```

Following, we will include the commands to start the Queue Manager as well as adding a listener to accept request to the Queues deployed on it.

The command line to start the Queue Manager "QMFirego" follows:

```
# strmqm QMFirego
```

The command line to start the listener for the QMFirego Queue Manager follows:

```
# runmqclsr -m QMFirego -t tcp
```

Once the Queue Manager has been started along with its listener,

the Queue Manager is ready to accept incoming connections forwarded to the deployed MQ Series Queues.

The command line for stopping the Listener for the Queue Manager QMFuego follows:

```
# runmqlsr -m QMFuego
```

The command line for stopping the Queue Manager QMFuego follows:

```
# endmqm QMFuego
```

The command line for checking the status of the Queue Manager and its Queues follows:

```
# dspmq
```

Creating MQ Queues using JMSAdmin

So far, we have configured the basis for deploying MQ Series Queues. When these Queues are created, Fuego will use the existing and well known API JMS to connect to the Queues to push and receive messages. We will also focus on how to define the Objects through JNDI so that Fuego can localize through the JNDI Lookup mechanism.

In order to connect to MQ Series Queues using JMS, the administrator will need to define a "QueueFactory" and "Queue" objects in a JNDI repository. For simplicity, we will use the Sun's FileSystem JNDI Implementation. It is possible to use other JNDI repositories such as IBM WebSphere one but for simplicity and

portability this document will use Sun's FileSystem JNDI Implementation.

To facilitate the creation of these Objects, IBM MQ Series 5.3 already provides a program called "JMSAdmin" bundled and distributed with the installation of this software. This command utilizes a configuration file named "JMSAdmin.config". As we will be using Sun's FileSystem JNDI Implementation we will need to focus on only 2 properties defined in this file. These 2 are listed below:

```
INITIAL_CONTEXT_FACTORY=
    com.sun.jndi.fscontext.RefFSContextFactory
PROVIDER_URL=file:/E:/tmp/JNDI-Directory
```

The location of the Directory Information may be specified by providing another location on the URL referenced by the "PROVIDER_URL" property.

JMSAdmin program requires the following environment variables are defined to work correctly. This information may also be obtained from IBM Documentation but it may be practical to have it at hand in this document. A sample environment shell script is depicted below:

```
#!/bin/bash
export MQ_HOME=" e:\Tools\WebSphere /WebSphere MQ"
export MQ_JAVA_INSTALL_PATH="$WS_HOME/WebSphere MQ/Java"
export FS_LDAP="e:\Tools/JNDIFileSystem/lib"
~np~ #MQ JMS~/np~
export MQ="$MQ;$MQ_JAVA_INSTALL_PATH/lib"
export MQ="$MQ;$MQ_JAVA_INSTALL_PATH/lib/com.ibm.mq.jar"
export MQ="$MQ;$MQ_JAVA_INSTALL_PATH/lib/com.ibm.mqjms.jar"
export MQ="$MQ;$MQ_JAVA_INSTALL_PATH/lib/connector.jar"
export MQ="$MQ;$MQ_JAVA_INSTALL_PATH/lib/jms.jar"
~np~ #File System LDAP~/np~
export MQ="$MQ;$FS_LDAP/fscontext.jar"
export MQ="$MQ;$FS_LDAP/providerutil.jar"
export CLASSPATH="$MQ;$CLASSPATH"
export PATH="$JAVA_HOME/bin:$MQ_JAVA_INSTALL_PATH/lib:$PATH"
```

Directories may need to be adjusted to point to the IBM MQ Series software installation directory.

Once the environment and JMSAdmin configuration file setup, we will proceed to describe how the Queue is defined using JMSAdmin program. The following script may be provided as input to the JMSAdmin. The script follows:

```
delete QCF(QCFuego)
define QCF(QCFuego) desc(Fuego Queue) tran(CLIENT)
               qmgr(QMFuego)
host(gandalf) port(1414) chan(SYSTEM.DEF.SVRCONN)
delete Q(FuegoQ)
define Q(FuegoQ) desc(FuegoQueue) QMGR(QMFuego)
QU(SYSTEM.DEFAULT.LOCAL.QUEUE)
end
```

The above script may be placed in a file named "queuesetup.sc" and executed as described in the following command line:

```
# ./JMSAdmin.sh < queuesetup.sc
```

Using JMS to integrate with created Queues

Once the Queue Manager has been defined, started, a listener is up and running for it and the JNDI settings have been defined with the JMSAdmin shell script, we are ready to create some Java components that using JMS can connect to MQ Series 5.3.

Find an example in the last section of this topic named *JMS program to send and receive messages into an MQ Series Queue*.

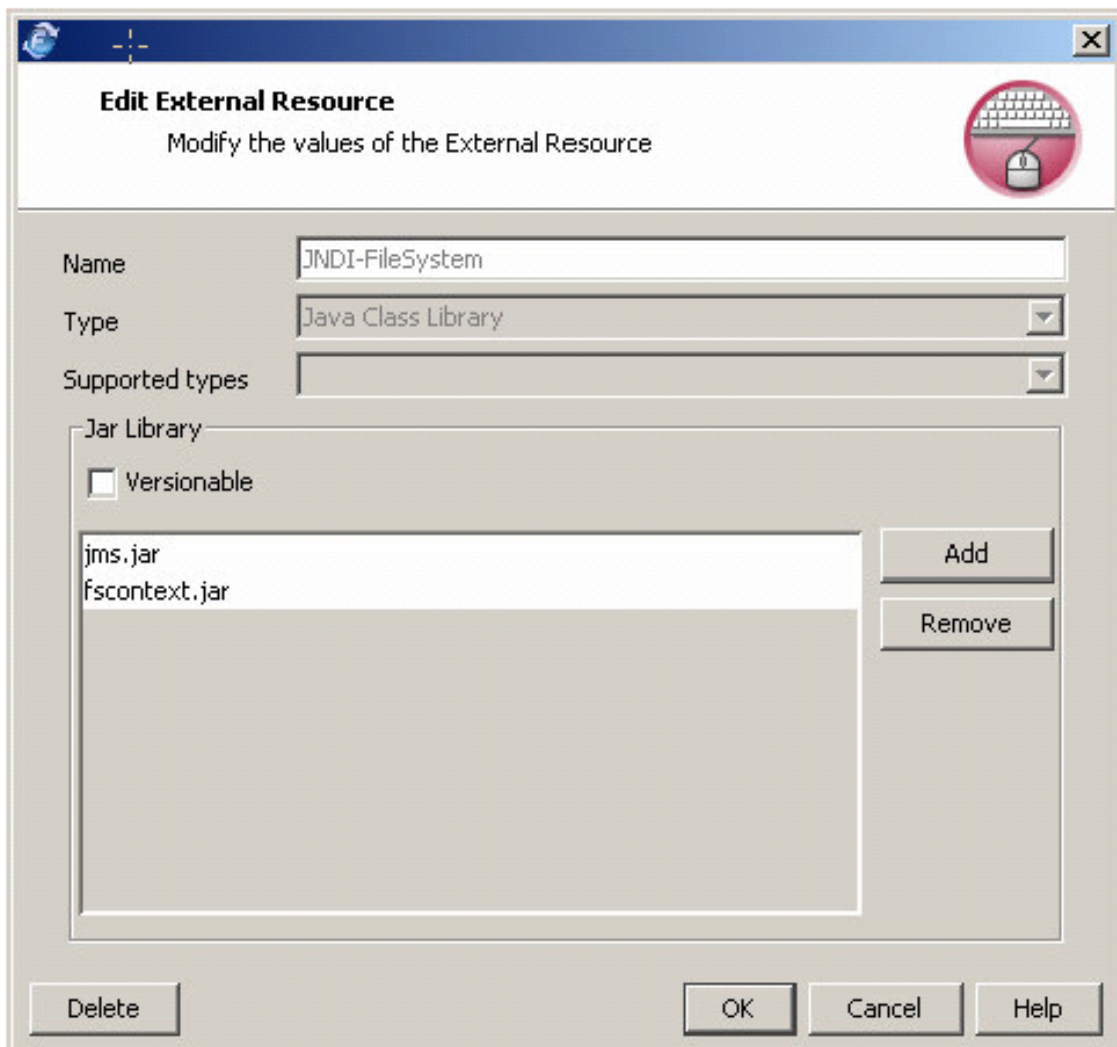
Integrating Fuego with MQ Series

This section of the document will guide you through the configuration of the Fuego Studio to integrate to the recently created MQ Series

Queues using JMS and JNDI.

Cataloguing JNDI and JMS APIs as Java Class Library External Resources

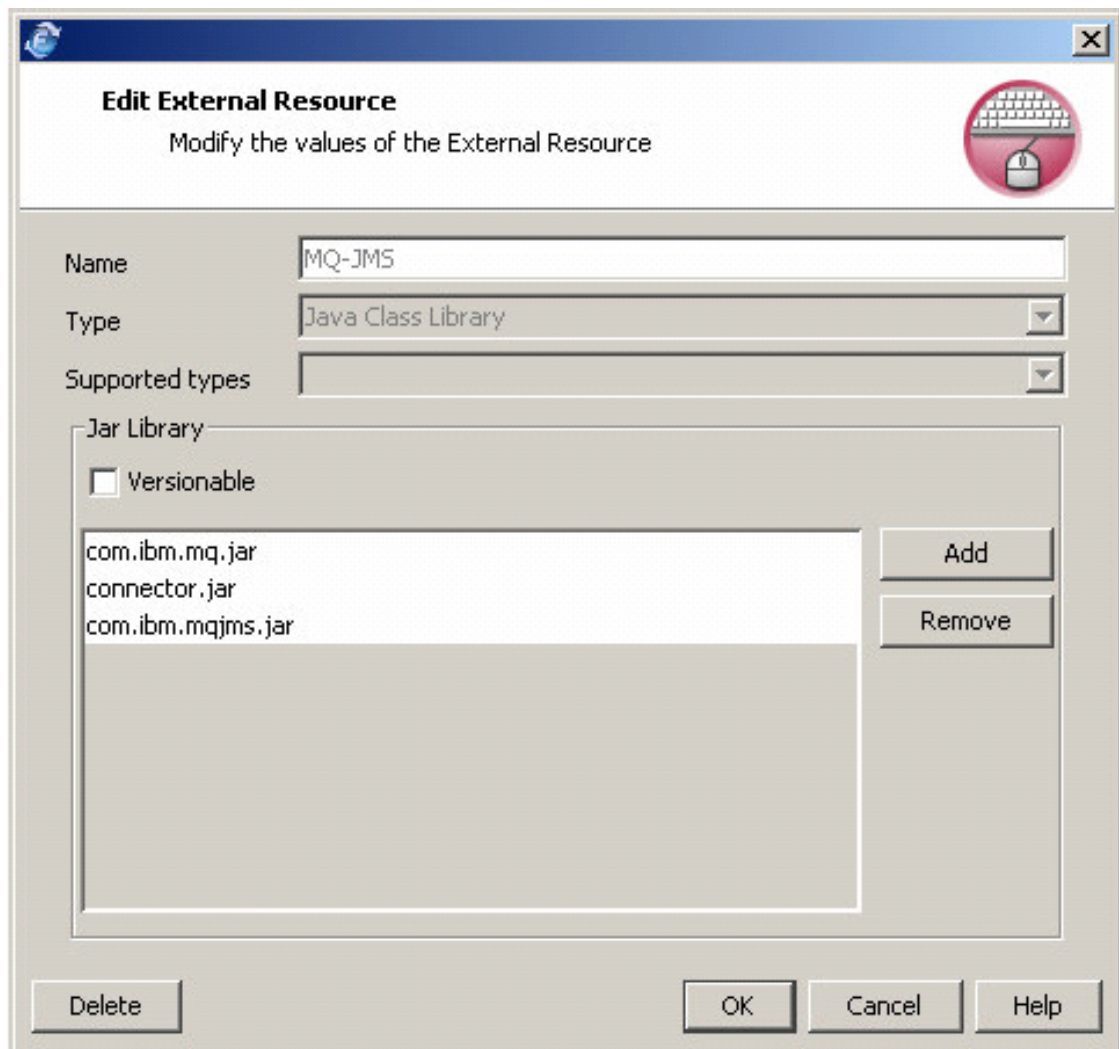
In order to use JNDI's service and JMS, we will first need to catalogue these APIs as Fuego Java Components. For this purpose, we will need to catalogue the API jars as a Java Library External Resource as depicted in the figure below:



jms.jar and **fscontext.jar** are public APIs from Sun and can be downloaded from this vendor.

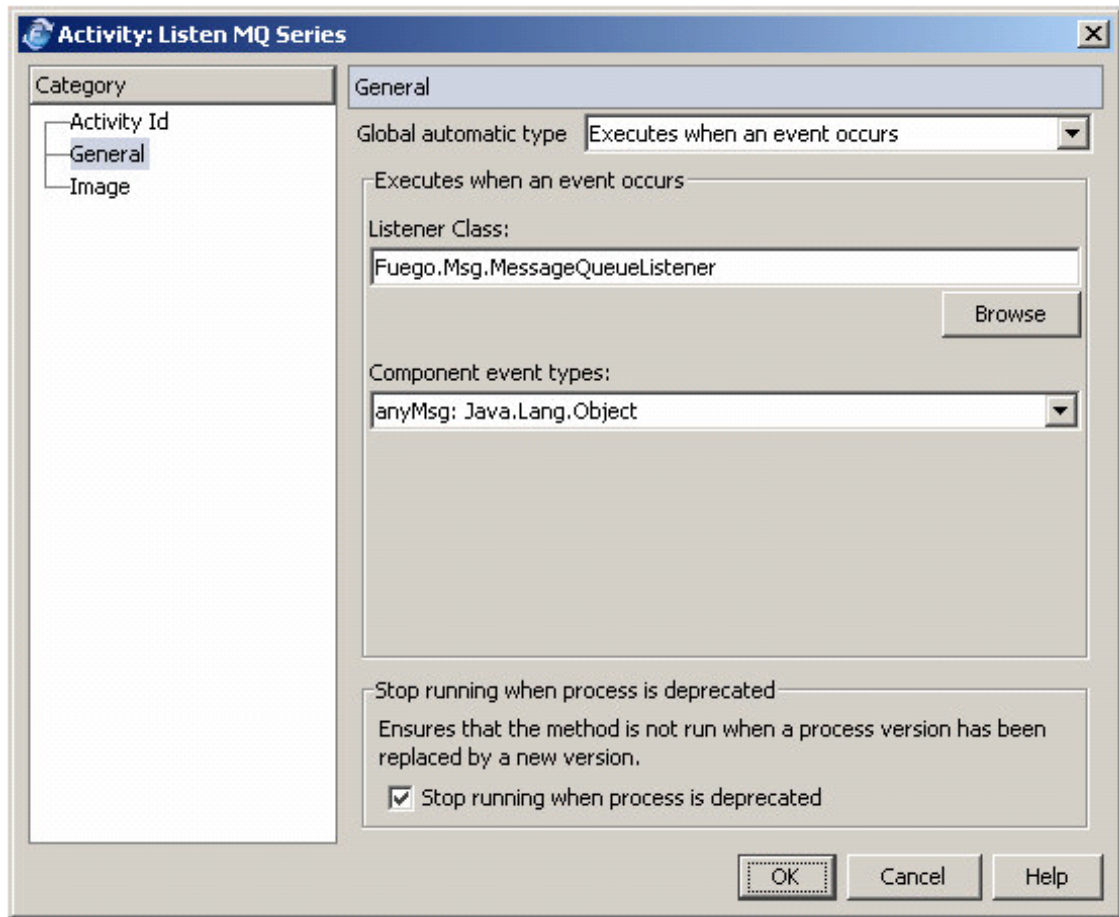
Likewise, we will need to catalogue IBM JMS Implementation Jar files

as a Java Class Library External Resource as shown below:



Adding a Listener type Global Automatic Activity to receive MQ Messages

The integration to receive Queue Messages is through the incorporation of a business process activity of type Global Automatic. Furthermore, this Global Automatic Activity needs to be of Listener type. The properties for this Global Automatic Activity is depicted in the figure below:



As you can see, we are using an existing Fuego Component named "Fuego.Msg.MessageQueueListener" as selected in the "Listener Class" drop down box.

This component internally uses JMS to connect to the MQ Series Queue Manager and Queues to receive Queue Messages. Furthermore, we are configuring the Component to receive any kind of message types as specified in the "Component event types" drop down box.

The following are both the Init and Listener Global Automatic Activity FBLs:

Init Global Automatic FBL

```
componentArguments = [  
  "contextFactory" :  
    "com.sun.jndi.fscontext.RefFSContextFactory",  
  "providerURL" : "file:/e:/tmp/JNDI-Directory",  
  "factoryName" : "QCF",  
  "queueName" : "FuegoQ"
```

In this case, we are defining the Context Factory to locate the Queue which in our case is Sun's FileSystem JNDI context Factory. Additionally, we are providing the location of the JNDI Repository with the Factory and Queue Names.

This FBL will be executed ONCE only. It is either executed when the Engine starts or when the process is first deployed.

Listener Global Automatic FBL

```
args = [ "data" : arg.textMsg ]  
ProcessInstance.create(arguments : args,  
  argumentsSetName : "BeginIn")
```

For this particular example, we are getting the message extracted from the MQ Series Queue and creating an instance into the process with the received argument.

This FBL is executed every time a message gets into the specified Queue of the Init FBL.

Adding JMS Code to send Messages to an MQ Queue

The Java Class added in Appendix A can be used as an abstraction using JMS to send Messages to a JMS Queue. Instead of having to deal with all the JMS details directly in a Fuego Object, this component may be used instead.

To continue with the guide, you will need to compile the Java Class

and package it into a Jar file.

This Jar file needs to be added as a Java Class Library External Resource.

Afterwards, you will need to catalogue the `fuego.jms.JMSMessage` Class as a Java Component in the Fuego Studio Catalogue.

The FBL to send Messages to an MQ Series Queue would be something like this:

```
data = "Data1"
env = [ "contextFactory":
        "com.sun.jndi.fscontext.RefFSContextFactory" ]
env = [ "providerURL": "file:/e:/tmp/JNDI-Directory" ]
env = [ "factoryName": "QCFuego" ]
env = [ "queueName": "FuegoQ" ]
m = JMSMessage(env)
logMessage "Sending Message to " + env["factoryName"] + "-"
+ env["queueName"]
m.send(data)
```

data in this case is of type `String`. *env* is of type `Any[Any]`. *m* is of type `JMSMessage`.

JMS program to send and receive messages into an MQ Series Queue

```
package fuego.jms;
import java.io.PrintStream;
import java.util.HashMap;
import java.util.Hashtable;
import javax.jms.*;
import javax.naming.directory.InitialDirContext;
public class JMSMessage
{
    public JMSMessage(HashMap params)
    {
        contextFactory = (String)params.get("contextFactory");
        providerURL = (String)params.get("providerURL");
        authentication = (String)params.get("authentication");
    }
}
```

```

principal = (String)params.get("principal");
credentials = (String)params.get("credentials");
factoryName = (String)params.get("factoryName");
queueName = (String)params.get("queueName");
if(contextFactory == null || contextFactory.length()==0
    || providerURL == null ||
providerURL.length() == 0 || factoryName == null ||
factoryName.length() == 0 ||
queueName == null || queueName.length() == 0)
{
    String msg = "Some of the required parameters are null
        or Empty.";
    throw new RuntimeException(msg);
}
else
{
    return;
}
}

public static void main(String args[])
    throws Exception
{
    HashMap p = new HashMap();
    p.put("contextFactory",
        "com.sun.jndi.fscontext.RefFSContextFactory");
    p.put("providerURL", "file:/e:/tmp/JNDI-Directory");
    System.out.println(p);
    if(args[0].equals("send"))
    {
        p.put("factoryName", "QCFuego");
        p.put("queueName", "FuegoQ");
        JMSMessage m = new JMSMessage(p);
        System.out.println("Sending to " +
            p.get("factoryName") + "-" +
            p.get("queueName"));
        m.send(args[1]);
    }
    else
    {
        if(args[0].equals("receive"))
        {
            p.put("factoryName", "QCFuego");
            p.put("queueName", "FuegoQ");
            System.out.println("Receiving from " +
                p.get("factoryName") + "-" +
                p.get("queueName"));
            JMSMessage m = new JMSMessage(p);
            boolean consume = args.length > 1 &&
                args[1].equals("consume");
            String msg = m.receive(0, consume);
            if(msg != null)
            {
                System.out.println("Message Received
                    (" + (consume ? "consumed" : "NOT

```

```
        consumed" + "));
        System.out.println("Length: " + msg.length());
        System.out.println("-----");
        System.out.println(msg);
        System.out.println("-----");
    }
    else
    {
        System.out.println("No message received.");
    }
}

public void send(String messageText)
    throws Exception
{
    send(messageText, true, 0L, 4);
}

public void send(String messageText,
    boolean persistent)
    throws Exception
{
    send(messageText, persistent, 0L, 4);
}

public void send(String messageText,
    boolean persistent, long timeToLive)
    throws Exception
{
    send(messageText, persistent, timeToLive, 4);
}

public void send(String messageText,
    boolean persistent, long timeToLive, int priority)
    throws Exception
{
    QueueConnection connection = null;
    try
    {
        InitialDirContext initContext =
            new InitialDirContext(getContextParams());
        QueueConnectionFactory factory =
            (QueueConnectionFactory)
                initContext.lookup(factoryName);
        Queue queue = (Queue)initContext.lookup(queueName);
        connection = factory.createQueueConnection();
        QueueSession session =
            connection.createQueueSession(false, 1);
        QueueSender sender = session.createSender(queue);
        TextMessage message =
            session.createTextMessage(messageText);
        sender.send(message, persistent ? 2 : 1, priority,
```

```
        timeToLive);
    }
    finally
    {
        try
        {
            if(connection != null)
                connection.close();
        }
        catch(JMSEException ee) { }
    }
}

public String receive(int timeout)
    throws Exception
{
    return receive(timeout, true);
}

public String receive(int timeout, boolean consume)
    throws Exception
{
    String textMessage = null;
    QueueConnection connection = null;
    try
    {
        InitialDirContext initContext =
            new InitialDirContext(getContextParams());
        QueueConnectionFactory factory =
            (QueueConnectionFactory)initContext.lookup
                (factoryName);
        Queue queue = (Queue)initContext.lookup(queueName);
        connection = factory.createQueueConnection();
        QueueSession queueSession =
            connection.createQueueSession(false, 2);
        QueueReceiver queueReceiver =
            queueSession.createReceiver(queue);
        connection.start();
        Message m = queueReceiver.receive(timeout);
        if(m != null)
        {
            if(m instanceof TextMessage)
                textMessage = ((TextMessage)m).getText();
            if(consume)
                m.acknowledge();
        }
    }
    finally
    {
        if(connection != null)
            try
            {

```

```
        connection.close();
    }
    catch(JMSEException e) { }
    }
    return textMessage;
}

private Hashtable getContextParams()
{
    Hashtable environment = new Hashtable();
    environment.put("java.naming.factory.initial",
        contextFactory);
    environment.put("java.naming.provider.url",
        providerURL);
    if(authentication == null ||
        authentication.length() == 0)
        authentication = "simple";

    environment.put("java.naming.security.authentication",
        authentication);

    if(principal == null || principal.length() == 0)
        principal = "";

    if(credentials == null || credentials.length() == 0)
        credentials = "";

    environment.put("java.naming.security.principal",
        principal);
    environment.put("java.naming.security.credentials",
        credentials);
    environment.put("java.naming.referral", "throw");
    return environment;
}
String contextFactory;
String providerURL;
String authentication;
String principal;
String credentials;
String factoryName;
String queueName;
}
```

Integrating FuegoBPM and Weblogic

Find the project *fuegoWeblogic* delivered with the FuegoBPM Studio distribution to see an example of integration between FuegoBPM and Weblogic.

The project can be found in the *samples* directory under the FuegoBPM Studio installation directory.

Appendix D - FuegoBPM Portlets

What is a Portlet

A portlet is a Java technology based web component, managed by a portlet container that processes requests and generates dynamic content. Portlets are used by portals as pluggable user interface components that provide a presentation layer to Information Systems.

The content generated by a portlet is also called a fragment. A fragment is a piece of markup (e.g. HTML, XHTML, WML) adhering to certain rules and can be aggregated with other fragments to form a complete document. The content of a portlet is normally aggregated with the content of other portlets to form the portal page. The lifecycle of a portlet is managed by the portlet container.

Normally, users interact with content produced by portlets, for example by following links or submitting forms, resulting in portlet actions being received by the portal, which are forwarded by it to the portlets targeted by the user's interactions.

FuegoBPM JSR-168 compliant Portlet

What is a JSR-168 compliant Portlet?

JSR-168 is the Java Portlet Specification from the Java Community Process. This specification enables standard portlets to be developed, which can be deployed on different portals which adhere to it.

What does FuegoBPM provide?

FuegoBPM provides a custom set of JSR-168 portlets which can be deployed on Portals from different vendors such as IBM, BEA and Apache.

These FuegoBPM *out-of-the-box* Portlets provides a way of interacting with a process in connection to your assigned role or roles within your company and effectively and efficiently manage your tasks with minimal extra training.

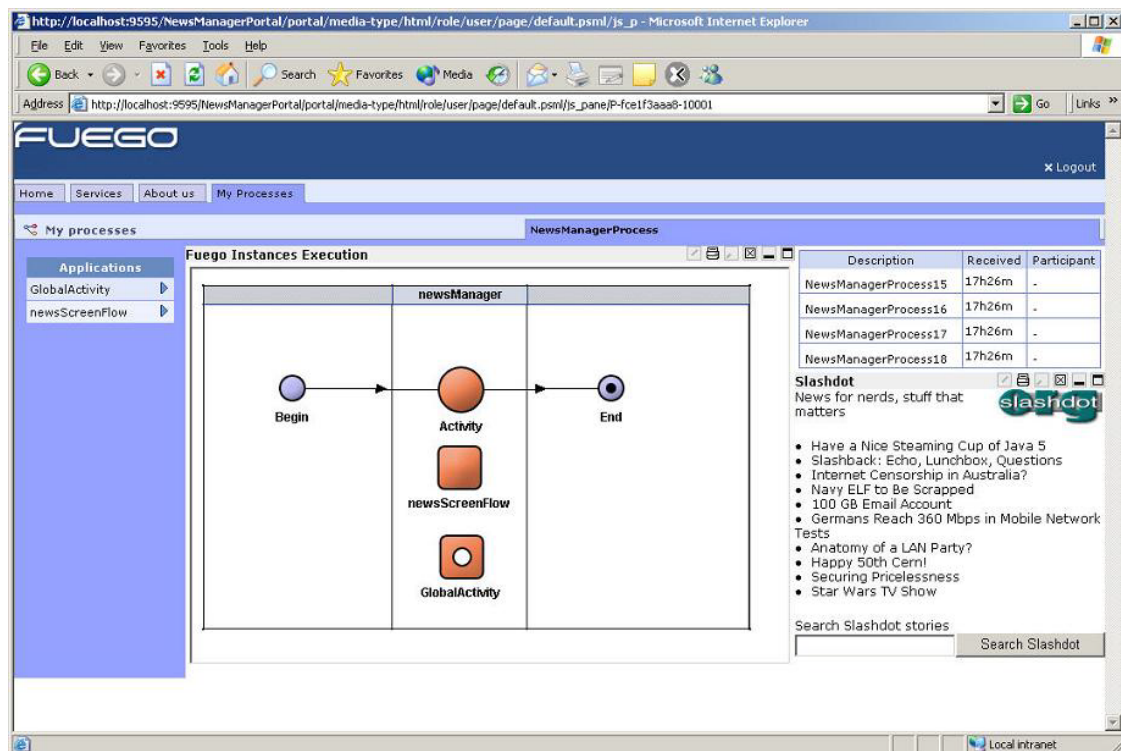
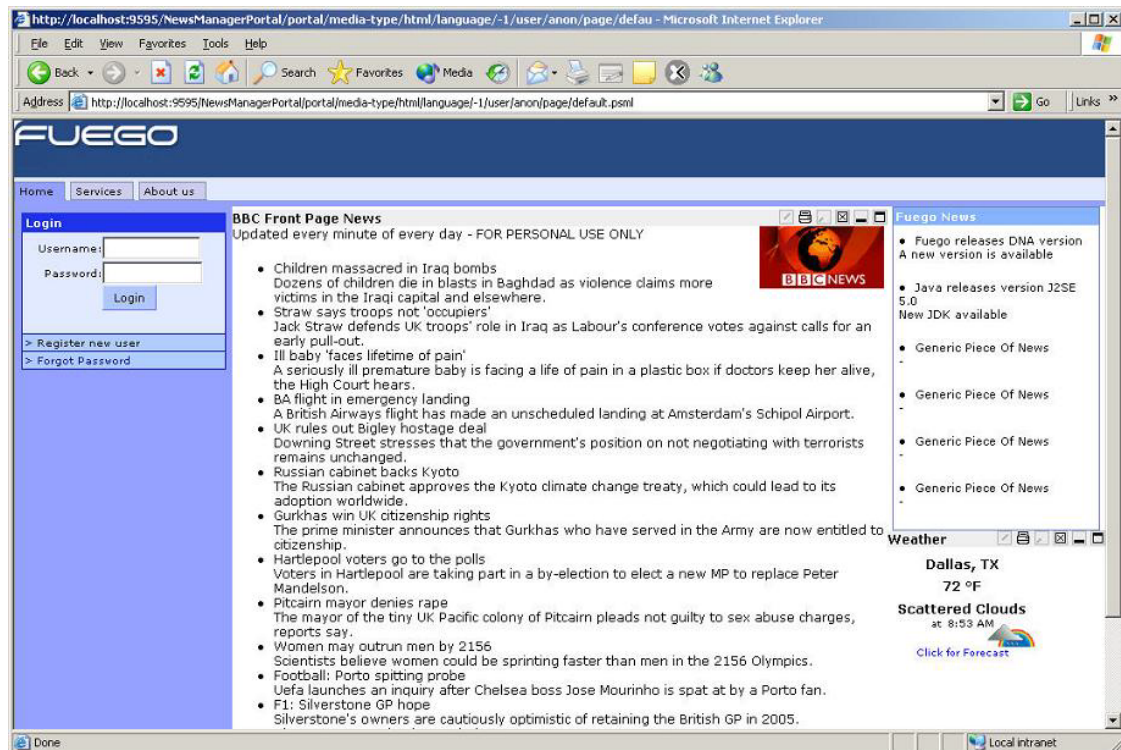
FuegoBPM Portlets allows you to:

- Process activities and tasks in current instances.
- Customize how your instances are organized using the portlet edition mode.
- Perform operations to a group of instances.
- Add notes and attachments to an instance.
- View detailed task descriptions, notes, attachments and audit information.

Examples

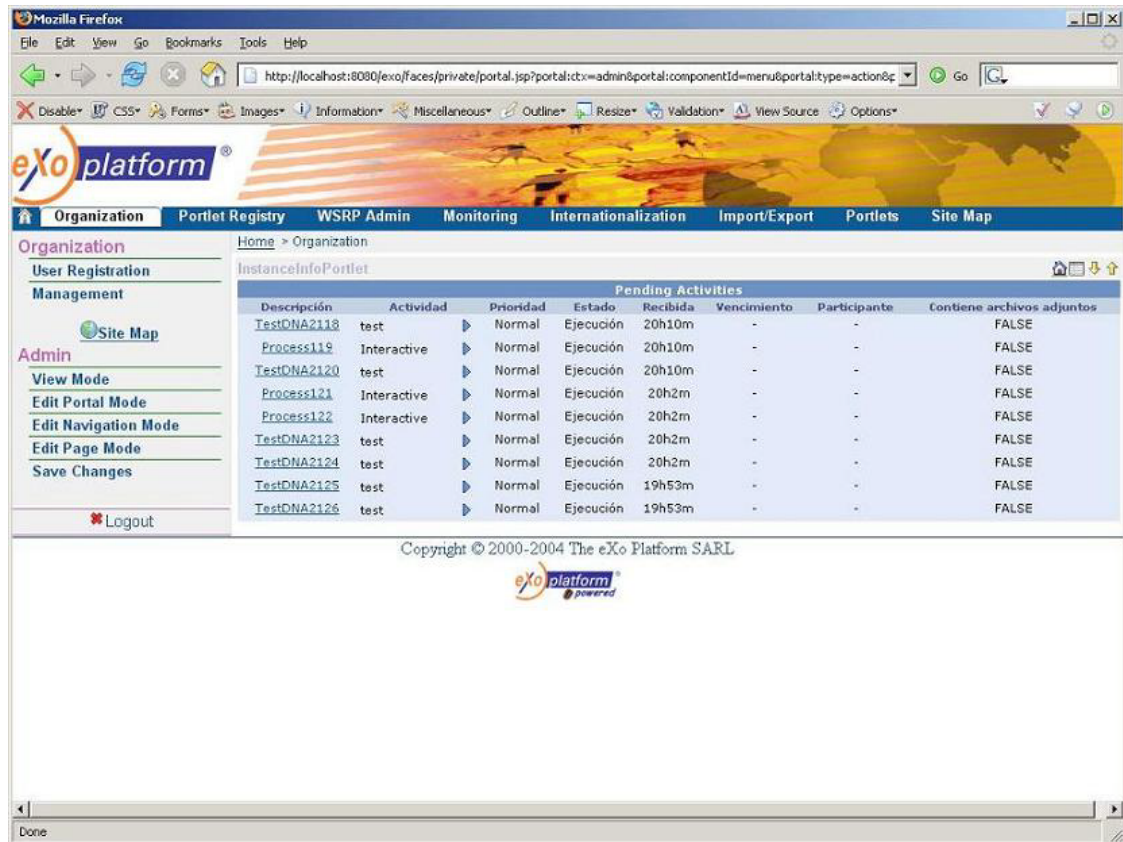
The following images show some FuegoBPM Portlets deployed on the *Jetspeed JSR-168* compliant portal:

Appendixes



The following image shows some FuegoBPM Portlets deployed on the

eXo JSR-168 compliant portal:



Appendix E - Other Documents of Interest

Additionally you can consult the following documentation:

1. PAPI Javadoc: in the FuegoBPM Studio installation directory, in the **help/docs** directory.
2. ANT Javadoc: in the FuegoBPM Studio installation directory, in the **help/docs** directory.
3. FuegoBlocks Components Javadoc: in the FuegoBPM Studio installation directory, in the **help/docs** directory.
4. APIs for integrating with Fuego: in the Fuego support site.