Reactor 6.0

Table of contents

1. Reactor 6.0 Documentation

Installation Guide
Step by step instructions for deploying Reactor in various environments and configuring Reactor services.

Administration Guide
Information on operating Reactor, including starting and stopping the server, migrating from previous versions and managing process data.

Design Guide
How to build and deploy Reactor process workflows, including guidelines for good process design. Includes a reference for Reactor Process Component Library objects.

User Guide
How to execute, monitor and participate in Reactor process workflows.

Developer Guide
How to use Reactor APIs to program custom process actions, customize Reactor applications and integrate Reactor with other systems.

Send us your feedback (http://www.oakgrovesystems.com/aboutus/contactus.htm) or contact our Support Group (http://www.oakgrovesystems.com/support/main_support.htm) if you cannot find what you need here.

2. Reactor 6.0

2.1. Reactor Overview

2.1.1. Introduction

The purpose of this guide is to:

- Introduce the features of the Reactor technology that are frequently requested by users.
- Provide step-by-step instructions of how to use the Reactor Product Suite to automate business processes.
- Provide a reference guide to assist users in the design and implementation of automated business processes.

Business process management systems like Reactor offer the possibility to define, automate, monitor, audit, and control business operations by leveraging the Web, middleware, and standards to more efficiently and effectively manage by process. The Reactor Product Suite

provides the tools to rapidly integrate sophisticated process management into business operations and applications.

2.1.2. Product Suite

Oak Grove's Reactor is a suite of products that work together to model, automate, integrate and streamline business processes, providing a platform for more efficient and productive business. This section describes the Reactor Product Suite, consisting of four primary components:

- Reactor Engine
- Reactor Studio
- Reactor Process Manager
- Reactor Application Framework

2.1.2.1. Reactor Engine

The Reactor Engine is a high-performance J2EE-based process engine for rapid deployment and flexible enactment of workflow or process integration applications. The Reactor Engine can be programmed entirely from Reactor Studio, or Reactor services can be leveraged by the EJB/Servlet developer as a process layer, embedded within an application.

This multi-platform, application Engine neutral J2EE-based process engine is central to the Reactor Product Suite. Developers and system administrators can quickly deploy the Reactor Engine to their J2EE application Engine. Reactor has been verified with BEA WebLogic, IBM WebSphere, and JBoss for rapid deployment of tightly integrated workflow or process integration applications on virtually any operating system, hardware platform, or environment. For customers who do not already have an Application Engine, Reactor comes bundled with the JBoss Application Engine for one-click deployment of the entire platform for any operating system or hardware platform for which the Sun Java SDK (version 1.4.1_02 or higher) is available.

Reactor Engine integrates seamlessly with existing J2EE applications. This makes Reactor the best choice for J2EE application developers who want to leverage a process layer for abstraction of business processes with their environment, providing the flexibility to quickly modify applications according to business demands. This process layer can be integrated with any number of existing applications and solutions, including CRM, manufacturing, Web Services, workflow, and so on.

Once installed and configured, the Reactor Engine generally requires no additional configuration or action by business users.

2.1.2.2. Reactor Studio

Reactor Studio is a user-friendly, graphical, business process-modeling tool that is ideal for the rapid development of complex process definitions. This tool gives application developers an easy way to build, organize, and reorganize the complex business processes that lay at the heart of their workflow or process integration application. Studio increases application developer productivity by giving them the power to quickly build and deploy process intensive systems using point and click process authoring and powerful scripting language-based programming capabilities.

Once the Reactor Engine is deployed, the Reactor Studio tool can be used to quickly build a process map that accurately reflects the business logic of the application at hand. Studio is a Java application, also requiring the Sun Java SDK (version 1.4.1_02 or higher) runtime. Reactor can handle arbitrarily complex business or application process logic, as well as all known workflow and process modeling patterns. The process designer can use the Studio features that provide both point and click process authoring, and more powerful scripting language programming capabilities. This enables automation of the most sophisticated business processes. Process Definitions from Studio are represented in XML, which can be uploaded to the Reactor Engine for immediate execution.

Business rules for processes are defined and implemented in Studio. Activities are displayed graphically as well as in the familiar tree-view in Studio. Easy to use dialog boxes capture the Process Definition and create the XML templates that Reactor Engine uses to run the Processes and display information to Users through the Process Manager application.

Reactor Studio provides the flexibility to write scripted actions in preferred scripting languages, including but not limited to JavaScript, Tcl, Python or Netrexx. Scripts within the process can perform arbitrarily complex programmed actions, such as accessing a legacy database, executing an operating system program like running a CRM report, or sending a wireless notification (WAP). The scripts can also refer to Operands stored in Reactor to add, change, delete, or update specific Processes during execution (for example, updating a Process with the completion date and time of a Subprocess).

2.1.2.3. Reactor Process Manager

Reactor Process Manager is a complete JSP/Servlet framework for rapid development of sophisticated web based user interfaces to workflow applications, as portlets in existing Process Manager interfaces or for use with existing web-based systems for seamless integration with the applications. Reactor ships with an example Process Manager application called Active Checklist that is suitable for implementation as a web interface for workflow applications developed using Reactor.

Process participants, process owners, managers, and process administrators can interact with Reactor during a business process from either the web based applications they're already

familiar with, or using the Reactor Process Manager application called Active Checklist. The Reactor Application Framework provides the Task User Interface (TUI) for the Reactor Engine. It can be modified, used as is, or Reactor can use a Process Manager developed in-house, using the Java, XML, or SOAP interfaces. End users can effectively manage work assigned to them and coordinate automated or combined manual-automated tasks using this web-based system.

2.2. Site Linkmap

2.2.1. Table of Contents

- Other Design Topics
  - Good Design Practices
  - Customize Process Manager
- User Guide
- Reactor 6.0 User Guide
  - Getting Started
  - Launching Processes
  - Monitoring Progress
  - Completing Activities
  - Analyzing Results
  - Workflow Examples
- Developer Guide
- Object Model
  - Introduction
  - Label Paths
  - Processes
  - Operands
  - Statuses
  - Policies
  - Actors
- Events And Policy Execution
  - Events
  - Writing Policies
- Reactor Clients
  - Introduction
  - Creating Reactor Clients
  - Creating Web Applications
    - Framework
    - Writing Servlet Code
    - Writing JSP Pages
    - Configuration
  - SOAP And WebServices
    - Introduction
    - XML Schemas
    - XML DTD
    - WSDL
- Reactor Requests
  - Request Types
  - Java Code Examples
  - Reactor Java API

2.3. Reactor Legal Issues

2.3.1. License

REDISTRIBUTION NOT PERMITTED

BEFORE YOU USE THE PRODUCT, CAREFULLY READ THE TERMS AND CONDITIONS OF THIS AGREEMENT. BY USING THE PRODUCT, YOU ARE CONSENTING TO BE BOUND BY AND ARE BECOMING A PARTY TO THIS AGREEMENT. IF YOU DO NOT AGREE TO ALL OF THE TERMS OF THIS AGREEMENT, PLEASE RETURN THE PRODUCT IMMEDIATELY.

Oak Grove Systems, Inc. Software and Documentation, hereinafter "the Software", are subject to the following license agreement. By installing the Software, you hereby accept these terms and conditions:

1. The Software is protected by copyright laws of the United States and international copyright treaty provisions. The Software, or any portion thereof, may not be reproduced, rented, leased or in any other manner redistributed without the prior written consent of Oak Grove Systems, Inc. You may make backup copies for archival purposes only. All printed materials are also protected by copyright law and may not be copied, distributed, or reproduced in whole or in part without the prior written consent of Oak Grove Systems, Inc.

2. The Software is licensed for use on a single computer. You may delete the Software from one computer and reinstall it on another, but you may not install the Software on more than one computer at any given time without prior written permission of Oak Grove Systems, Inc.

3. You acknowledge that the Software in source code form remains a confidential trade secret of Oak Grove Systems, Inc. and therefore you agree not to modify the Software or attempt to decipher, decompile, disassemble or reverse engineer the Software, except to the extent applicable laws specifically prohibit such restriction. You may not (i) permit other individuals to use the Software except under the terms of this Agreement, (ii) modify, translate or create derivative works based on the Software, or (iii) remove any proprietary notices or labels on the Software.

4. Title, ownership rights, and intellectual property rights in and to the Software shall remain in Oak Grove Systems, Inc. and/or its suppliers. Title, ownership rights, and intellectual property rights in and to the content accessed through the Software is the property of the applicable content owner and may be protected by applicable copyright or other law. This License gives you no rights to such content.

5. You may not download and install or otherwise export or reexport the Software or any

underlying information or technology except in full compliance with all United States and other applicable laws and regulations. In particular, but without limitation, none of the Software or underlying information or technology may be downloaded and installed or otherwise exported or reexported (i) into (or to a national or resident of) Cuba, Iraq, Libya, North Korea, Iran, or Syria or (ii) to anyone on the US Treasury Department's List of Specially Designated Nationals or the US Commerce Department's Table of Denial Orders. By downloading and installing the Software, you are agreeing to the foregoing and you are representing and warranting that you are not located in, under control of, or a national or resident of any such country or on any such list.

6. THE SOFTWARE IS LICENSED TO YOU "AS IS". TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, OAK GROVE SYSTEMS, INC. EXPRESSLY DISCLAIMS ALL OTHER WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, AND NONINFRINGEMENT, AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION. IF APPLICABLE LAW IMPLIES ANY WARRANTIES OR CONDITIONS WITH RESPECT TO THE SOFTWARE, ALL SUCH WARRANTIES OR CONDITIONS ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF DELIVERY.

OAK GROVE SYSTEMS, INC. DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET YOUR REQUIREMENTS, OR THAT THE OPERATION OF THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT DEFECTS IN THE SOFTWARE WILL BE CORRECTED. FURTHERMORE, OAK GROVE SYSTEMS, INC. DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE SOFTWARE IN TERMS OF THEIR CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE. SHOULD THE SOFTWARE PROVE DEFECTIVE, YOU (AND NOT OAK GROVE SYSTEMS, INC., ITS DISTRIBUTORS, DEALERS OR EMPLOYEES) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR, OR CORRECTION. SOME STATES AND JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF EXPRESS OR IMPLIED WARRANTIES OR CONDITIONS, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY ALSO HAVE OTHER LEGAL RIGHTS WHICH VARY FROM STATE TO STATE AND JURISDICTION TO JURISDICTION.

7. WHILE EVERY EFFORT IS MADE TO ENSURE THAT THE SOFTWARE AND ITS DOCUMENTATION ARE FREE FROM DEFECT, UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, TORT, CONTRACT, OR OTHERWISE, SHALL OAK GROVE SYSTEMS, INC. OR ITS DISTRIBUTORS, DEALERS OR EMPLOYEES

BE LIABLE TO YOU OR ANY OTHER PERSON FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFIT OR GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES OCCASIONED BY THE USE OF THE SOFTWARE. IN NO EVENT WILL OAK GROVE SYSTEMS, INC. BE LIABLE FOR ANY DAMAGES IN EXCESS OF OAK GROVE SYSTEMS, INC. LIST PRICE FOR A LICENSE TO THE SOFTWARE, EVEN IF OAK GROVE SYSTEMS, INC. SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY OTHER PARTY. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO LIABILITY FOR DEATH OR PERSONAL INJURY TO THE EXTENT APPLICABLE LAW PROHIBITS SUCH LIMITATION. FURTHERMORE, SOME STATES AND JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS LIMITATION AND EXCLUSION MAY NOT APPLY TO YOU.

8. If you are a unit or agency of the United States Government or are acquiring the Software for any such unit or agency, the following apply:

(a) If the unit or agency is the Department of Defense ("DOD"), the Software and Documentation are classified as "commercial computer software" and "commercial computer software documentation", respectively, and, pursuant to DFAR Section 227.7202, the Government is acquiring the Software and Documentation in accordance with the terms of this Agreement.

(b) If the unit or agency is other than DOD, the Software and Documentation are classified as "commercial computer software" and "commercial computer software documentation", respectively, and pursuant to FAR Section 12.212, the Government is acquiring the Software and Documentation in accordance with the terms of this Agreement.

9. This agreement is effective until it terminates. You may terminate this Agreement at any time by destroying the Software and all backup copies thereof. This Agreement will terminate immediately without notice from Oak Grove Systems, Inc. if you fail to comply with any provision of this Agreement. Upon termination you must destroy the Software and all backup copies thereof.

10. You shall comply with all laws and regulations applicable to your activities under this Agreement, including but not limited to, all Department of Commerce and other United States export controls. This Agreement constitutes the entire agreement between the parties and supersedes any other communications or advertising with respect to the Software. Notwithstanding the foregoing, to the extent that any court, tribunal, or other governmental authority of competent jurisdiction determines that any terms of this Agreement are in

conflict with the terms of any statute, treaty, or regulation, the conflicting terms of this Agreement shall be superseded only to the extent necessary by the terms required by such law, statute, treaty, or regulation. If any provision of this Agreement shall be otherwise unlawful, void, or for any reason unenforceable, then that provision shall be enforced to the maximum extent permitted so as to effect the parties' intent. In either case, the remaining provisions of this Agreement shall remain in full force and effect. This Agreement is governed by the laws of the State of California, U.S.A., excluding conflict of laws provisions and the 1980 United Nations Convention on Contracts for the International Sale of Goods. Oak Grove Systems, Inc. failure to enforce any provision of this Agreement shall not be deemed a waiver of such provision.

Other product and company names appearing in Oak Grove Systems, Inc. products and materials are used for identification purposes only and may be trademarks or registered trademarks of their respective companies. Registered and unregistered trademarks used in any Oak Grove Systems, Inc. products and materials are the exclusive property of their respective owners.

2.3.2. Legal Notices

This publication, including any associated materials, guides or portions thereof, may not be reproduced or transmitted in any form or by any means, electronic, mechanical, or otherwise, without the prior written permission of Oak Grove Systems.

This documentation is provided without any warranty whatsoever, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose.

Reactor, Reactor Studio, Reactor Engine, Reactor Process Manager, Reactor 5 and Reactor 6 are trademarks of Oak Grove Systems.

Sun, Sun Microsystems, Java, J2EE, JSP, EJB, JDBC, JVM are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

2.4. Reactor 6.0.1 Release Notes

2.4.1. Known Issues

The following table lists the known issues with Reactor and provides a suggested workaround for each issue.

Issue	Workaround
Map viewer may not work correctly unless you clear web browser's Java cache.	Restart web browser, or otherwise clear its Java cache.

OR nodes may not work as expected with repeated executions (looping)	Do not use OR nodes contained within a process loop. Connect all inputs of the OR node directly to the output of the proposed OR node.
Microsoft Windows freezes when Reactor Studio exits. Sun's SDK 1.4.1_01 for Microsoft Windows is known to cause some video drivers to freeze when any Java Swing application exits.	Upgrade to SDK 1.4.1_02 or higher.
Reports do not work on UNIX systems where there is no X server installed.	The reports module requires the DISPLAY environment variable to be set. Set the DISPLAY environment variable to a server with a working X Server installation before starting the Reactor engine.
Processes don't retain assigned images after saving and reloading.	Place your image files into the `studio/images` directory and reference them from there.
5.5.3 clip libraries don't retain drawing information, Form information, or other meta data information.	An upgrade utility is in progress. Wait for the utility or create new library components using Reactor Studio 6.0.
Renaming a Process Definition through the API may cause drawing information to not display correctly.	Don't rename process definitions through the API. Use Studio instead.

2.4.2. Product History

This section provides a history of the Reactor Software development. The major changes implemented in each release of Reactor since 5.5 are outlined below.

2.4.2.1. 6.0.1 Changes

Changes from release 6.0 to release 6.0.1.

Reactor Engine

- Re-organized the hibernate descriptors for easier DB deployments.
- Applied scrolling changes from worklist query to query criteria.
- Query criteria now support some scrolling and sorting.
- Engine now support MS SQL Server.
- Engine now supports DB2.
- Included the Reactor Web Services Module in bundles.
- Increased the default max heap size on Enterprise to be 512 MB.
- Tweaked the caching to be more reasonable.
- Upgraded Hibernate to 2.1.7c.

- Fixed problem with Create API not returning the correct IDs.
- Fixed issue with dynamic creation of Process Objects.
- Modified Configuration Tool from Professional. Only limited configuration is supported.
- Fixed a NullPointerException on some Parameterized Policies.

Reactor Studio

- Fixed the parameter types on `SendNotifcationByRole` Policy Action.
- Fixed the parameter types on `CopyRemoteOperandWithID` Policy Action.
- Fixed the parameter types on `CopyLocalOperandWithID` Policy Action.
- Added `commons-codec-1.3.jar` for serialzing binary data into processes.
- Field Set *New* and *Add* buttons have been resized/moved slightly.
- Fixed some error dialogs to have correct messages.
- Fixed issues with making Field Sets available to other Activities.
- Studio now stamps Studio version to exported clip libraries.
- Fixed an error message when pasting clipboard contents.
- Fixed a NullPointerException when pasting Decisions.
- Fixed problem with *NotifyParticipant* feature of Forms.
- Fixed problem with wrong message dialog showing when editing an Actor.
- Added a new Role, *Process Participant*, to support notifications.

Reactor Process Manager

- Slight change to the legend on MAP view.
- Fixed problem with finished transitions not displaying correctly in MAP View.
- Fixed problem with sort indicator not displaying on some columns.
- Fixed the title of the `ProcessManager.html` document in the bundles.
- Added functionality to use the new sorting / scrolling capability for Query API.
- Replaced the O'Reilly servlets package with commons-fileupload.
- Notes can now be longer than 4000 characters.
- Removed the source for the Map viewer from the distributions.
- Enhanced navigation to other Instance views from MAP view, TAB view, XML view, and audit trail view.
- Process properties now show in the audit trail view.
- An error page is now displayed when someone tries to acquire an Activity that has already been acquired.
- Fixed some potential issues with result sets and statements not being closed in Reports.
- Durations should now be correct in Reports.
- Fixed a NullPointerException on some Reports.
- Fixed issues with Reports on Oracle Platforms.
- Reports now have an auto-refresh capability
- Fixed problem with StartProcess for parsing Operand names and values.

Documentation

- Updated JavaDocs on many classes.
- JavaDocs now include Directory Module and user for creating custom Directory Modules.

Converter

- Converter now converts clip library items also.

Samples

- *New Task* and *New Task Request* now use the Start Form for the correct Process.
- Updated the documentation on several sample Processes.

2.4.2.2. 6.0 Changes

Changes from release 5.5.3 to release 6.0.

Reactor Studio

- Richer form-building with field sets.
- New Timer Tool for graphically showing timers.
- New Stop Node for multiple exit points on the Process palette.
- Additional features for customizing display on the Process Palette.
- Internationalized and made ready for localizations.

Reactor Process Manager

- Renamed from "Reactor Portal" to clear up potential confusion.
- Now uses Struts web application framework.
- Internationalized and made ready for localizations.

Documentation

- Migrated documentation from PDF document to Apache Forrest

Converter

- Created Converter to migrate Reactor 5 Processes to Reactor 6.

Deployment

- Evaluation bundle now uses JBoss 3.2.5 with HSQLDB.

2.4.2.3. 5.5.3 Changes

Changes from release 5.5.2 to release 5.5.3.

Reactor Engine

- Modified the way Quartz persists job information.
- Modified the `isDefinition` column on the process table for Sybase. Sybase installations will need to migrate their tables from `bit` to `int`.

Reactor Studio

- Several changes to prevent invalid Processes from being created.

Reactor Process Manager

- Performance enhancements for the Activity List.
- Modified the durations on the Process Manager Reports to be more useful.

2.4.2.4. 5.5.2 Changes

Changes from release 5.5.1 to release 5.5.2.

Reactor Engine

- Fixed bug where Timers were lost in Engine shutdown.
- Fixed bug with Audit Trail handling of Operand updates when deployed to Oracle database.

Reactor Studio

- Updated Decision Activity handling to fix race condition and improve performance.
- Fixed bug where pasting Join/Split bars would corrupt the Process Definition in some circumstances.
- Fixed bug where adding a `Custom` Policy would corrupt the Process Definition in some circumstances.
- Fixed bug where Studio did not log out from the Engine when done.
- Fixed bug where Process definition was not created correctly if it had a Join/Split bar connected to the Stop icon with no inbound connections to the Join/Split bar.
- Fixed bug where editing a timer-related Policy did not show the correct schedule expression.
- Fixed bug where Policy Action parameters were not shown in the order defined in the Policy Action definition XML file.
- Added support for Group type to Swimlanes.
- Added Swimlane documentation to user guide.
- Updated documentation.

Reactor Process Manager

- Fixed bugs involving traversal coloring in graphical view of Process Map.
- Fixed bug where Process Manager did not log out from the Engine when done.
- Fixed bug where `Email` Operands were not rendered with mailto: link.
- Improved handling of attachment file types by adding Content-Disposition HTTP header to downloaded attachments.
- Fixed bug in `DelegateePage.jsp` where form was not rendered properly.
- Fixed bug where button was shown without correct style on the tranferrer's page for declined transfer.
- Fixed bug where WebLogic 8.1 complained about authentication object not being serializable.
- Resolved usability issue where selected sort field did not carry over to subsequent pages.

Command Line Tool

- Added capability for Command Line Tool to take Process Definition files created by Studio and import them into the Engine.
- Updated documentation of Command Line Tool.

Deployment

- Fixed problems deploying Quartz scheduler to MS SQL Server.
- Fixed problems deploying Quartz scheduler to Sybase.
- Added Quartz calendar class to JavaDoc documentation.
- Fixed bug where `DBObjectPolicy` did not have required classes in classpath.
- Updated documentation on deploying Quartz scheduler to databases.

2.4.2.5. 5.5.1 Changes

Changes from release 5.5 to release 5.5.1

Reactor Engine

- Updated to use Quartz as default scheduler, which includes support for calendars.
- Added support for Sybase database.
- Upgraded to Pointbase 4.7 database in evaluation bundle.
- Fixed bugs where Timers were not behaving properly.
- Fixed message bundle for configuration tool to work with SDK 1.3.1.

Reactor Studio

- Added Hand toolbar button for panning capability.
- Added keyboard shortcuts for `CUT`, `COPY` and `PASTE` in drawing panel.
- Added capability for Swimlanes to handle Titles and Groups.
- Added capability for Picklist Operands to have default values.

- Added the option to suppress popup notification of successful login.
- Added `IncrementOperand` Policy Action.
- Added `TimerCalendars` Policy Action.
- Modified `NOT` logic node to prohibit multiple inputs.
- Updated `SetACLofProcess` Policy Action to handle Titles and Groups.
- Updated example processes.
- Fixed bugs in Decision Activity expression handling.
- Fixed bug in handling of `Otherwise` condition in Decision Activity.
- Fixed bug in ordering of Policy parameters in `Edit Policy` dialog window.
- Fixed bug with `Close All` where User was asked to confirming closing of empty Processes.

Reactor Process Manager

- Added more information to display of Audit Trail.
- Improved handling of delegation.
- Fixed intermittent bug with acquiring Activities.

3. Complete Documentation

4. Reactor Installation And Configuration

This section contains installation and deployment information for each of the three versions of Reactor.

Install Reactor Professional
Reactor Professional is distributed as a standalone application with no configuration required on most systems. This page contains step-by-step installation instructions.

Install Reactor Enterprise
Reactor Enterprise is a high performance, fully scalable engine designed to fully leverage your J2EE computing infrastructure. This page contains step-by-step instructions for deploying Reactor Enterprise to your application server and configuring database, directory, authentication, and scheduling services.

Install Reactor Embedded
Reactor Embedded is optimized for embedding and distribution with your Enterprise Application product. Reactor Embedded includes complete source code for the Engine, Studio, and Process Manager web application to help get you started. This page contains complete setup instructions.

5. Reactor 6.0 Professional

5.1. Installing Reactor 6.0 - Professional

5.1.1. Overview

Reactor 6.0 Professional is distributed as a standalone application and requires little or no configuration to get started. It's almost as easy as download, unpack, and run!

The following sections walk through the process of installing and running Reactor 6.0 Professional step-by-step:

5.1.2. Prerequisities

1. Ensure your machine is running the Sun Microsystems J2SE SDK version 1.4.1_02 or higher.
   Help Me
   - If necessary, you can download the latest version from the Sun Microsystems website.
     Help Me
   - If you downloaded a new version of the SDK, install the new J2SE SDK into a root level directory such as `C:\j2sdk1.4.2\`.
2. Check the `JAVA_HOME` variable in the operating system. It must be set to point at the J2SE SDK directory.
   Help Me

Note:

Sun's Java SDK 1.4.1_01 for Microsoft Windows is known to cause some video drivers to freeze when a Java Swing application exits. Versions after 1.4.1_01 do not have this problem.

5.1.3. Download File

Download Reactor from the Oak Grove Systems (http://www.oakgrovesystems.com) website.

5.1.4. Install Reactor

Unzip the Reactor bundle with file compression software such as WinZip. The extraction options should already be set (as shown in the following diagram on a Windows machine) to create the proper directories and to place the files in their proper locations. If not, modify the Extract window to match these settings.

Warning:

Reactor should be installed in a directory at the root level and the directory name must not contain any spaces.

Now display the contents of the directory `C:\ReactorProfessional-6.0\`. using Windows Explorer or a terminal window command like `ls` in UNIX. You should see something like the picture below containing eight files and five directories.

5.1.5. Start Reactor

To start the Reactor Engine, execute the `StartReactorEngine.bat` (Windows) file or the `StartReactorEngine.sh` script (UNIX) in the `C:\ReactorProfessional-6.0\` directory. It will open a terminal window and begin a roughly 30 second startup process.

When you see something similar to the picture below in the window, the Engine has been started.

Note:

The terminal window will continue to periodically add lines as the Engine begins to execute instructions.
Reactor Studio may be used independently of the Reactor Engine and the Reactor Process Manager. It is not required to have the Engine running to design workflows in Studio. However, the Engine must be running in order to upload, download, execute or monitor the status of a workflow Process.

5.1.6. Get Started!

After the Engine is running, the Process Manager application may be used. To use the Process Manager, open your web browser and type `http://localhost:8080/ReactorPortal` in the Address bar. Alternatively, you may execute the `ReactorProcessManager.html` file located at `C:\ReactorProfessional-6.0\`. Both methods should take you to the Reactor Process Manager login page. You may login using a Username of `admin` and a Password of `admin`.

That's it! You have installed and confirmed the operation of the entire Reactor Product Suite. Now you're ready to conquer the world.

Java Help

Java Version

To check your version of Java, simply type the following command at the command prompt.

```
java -version
```

You should see something like the following picture. If your version is not at least 1.4.1_02,

then you need to upgrade.

> **Note:**
> This will not tell you if you have the Java SDK. In most cases the SDK is not part of the standard installation on a machine. If you have not specifically downloaded the SDK, then you probably only have the run-time VM.

back

Java SDK Download

The latest version of Sun's J2SE SDK is 1.5 and can be downloaded from the Sun Microsystems website (http://java.sun.com/downloads/) . Although the download is free, Sun may require that you register with them prior to downloading any files.

> **Warning:**
> Sun's Java SDK 1.4.1_01 for Microsoft Windows is known to cause some video drivers to freeze when a Java Swing application exits. Versions after 1.4.1_01 do not have this problem.

back

Setting The JAVA_HOME Variable

For Windows Users:

Environment variables are accessed via the Advanced tab of System settings in the Windows Control Panel. If there is no `JAVA_HOME` variable set, click on the `New` button to add the variable. When you are done it should look something like the picture below.

Restart your computer for the new setting to take effect.

For UNIX Users:

Although the the various flavors of UNIX are a bit different, executing a comamnd along the lines of the following example should work in most cases:

```
export JAVA_HOME=/usr/bin/java1.4
```

back

5.2. Reactor Professional Bundle

5.2.1. Overview

The following table lists the directories included in the Reactor Professional distribution package. The Reactor WAR file is already deployed to Tomcat in `C:\ReactorProfessional-6.0\tomcat-4.1.30\webapps\`.

5.2.2. Structure

This is the structure of the directories created by Reactor Professional.

Directory	Contents
\commandline	Reactor command line interface files.
\lib	Reactor configuration files.
\samples	Examples that illustrate client code and web applications that interact with the Reactor Engine.
\studio	Reactor Studio Application - a graphical design tool for authoring Processes.
\tomcat-4.1.30	Reactor Engine and Process Manager deployed and preconfigured in a distribution of the Apache Tomcat 4.1.30 servlet container and JSP engine.

6. Reactor 6.0 Enterprise

6.1. Deploying Reactor 6.0 - Enterprise

6.1.1. Overview

There are six parts to installing and configuring Reactor Enterprise for use. Reactor comes "ready-to-run" and if you plan to use the Enterprise bundle without integrating it into your particular environment, then you may skip the sections about setting up the Application Server, the Database and Directory Services. Simply install Reactor, verify the key configuration settings (such as e-mail host) and you're ready to go.

If you plan to use Reactor with another Application Server and/or Database or plan to use it with an LDAP, then simply follow the applicable instructions in each of the relevant sections. The order in which the setup task are completed matters, so complete each section in the sequence shown here. Once the last section is complete, start up the Application Server and

the Database Server and then each piece of the Reactor Product Suite. Login to the Process Manager using a Username of `admin` and a Password of `admin` to verify the correct operation of Reactor.

6.1.2. Installation

The first step in getting Reactor setup to use is installing the software. The installation instructions for Reactor Professional will guide you through the process of installing Reactor in five easy steps. The directory and file structure will be slightly different in the Enterprise version and the main directory will be called `C:\ReactorEnterprise6.0-JBoss3.2.5\`, but the installation steps are identical.

If you are using the Enterprise bundle and plan to use the included JBoss 3.2.5 Application Server and HSQL Database, then you can skip directly to the Advanced Topics section.

6.1.3. App Servers

This section explains how to deploy The Reactor Product Suite on some of the most popular application servers.

J2EE applications such as Reactor don't necessarily need any details about the database they are to access. For example, an application may not need to know whether it's communicating with an Oracle server running on the same machine or a PostgreSQL server running on a machine on another continent. Instead, the application only needs the JNDI name of a `javax.sql.DataSource`. Data sources are configured in the J2EE server's configuration, not the application's configuration. The J2EE Server, not the application, manages data sources and the JNDI namespace. Applications lookup the data source in the J2EE server's JNDI namespace and call the `getConnection()` method of the data source to acquire a JDBC `java.sql.Connection` to the underlying database. One can configure a J2EE server to manage multiple data sources, each with a different JNDI name. Any number of applications can access the same data source by lookup it up by its JNDI name in the J2EE server's JNDI namespace. Also, a single application can access multiple data sources by looking them up separately by their respective JNDI names. To determine how to configure a Data Source in a particular application server, see the documentation for that particular application server below.

If you are going to use an Application Server other than the bundled JBoss 3.2.5 product, go to this section and follow the appropriate instructions. Reactor has been tested and verified with the following Application Servers.

- BEA
  - WebLogic 8.1

    - WebLogic 7.0.0
- IBM
    - WebSphere 5.0
    - WebSphere 4.0
- JBOSS
    - 3.2.5 *(included in the Reactor evaluation bundle)*
    - 3.2.1
    - 3.0.6

6.1.4. Databases

Reactor manages the workflow transactions through the use of a database. The Reactor bundles are set up to use an integrated HSQL database for this purpose. However, Reactor can be used with the following databases.

- DB2
- HSQL *(included in the Reactor evaluation bundle)*
- InstantDB
- MS SQL Server
- MySQL
- Oracle
- Pointbase
- PostgreSQL
- Sybase

6.1.5. Directory Service

There are two mechanisms available in Reactor to authenticate Users, CSV Files containing Usernames and Passwords or an LDAP. Reactor is distributed with a set of simple CSV Files that define the key attributes of each permitted User and uses these files by default to perform authentication.

6.1.6. Advanced Topics

The Advanced Topics section covers issues such as setting up the Audit Trail feature, using the Quartz timer scheduling feature, clustering and load balancing.

6.1.7. Configuration

Reactor has several configuration files containing various settings that determine how Reactor functions, how it displays data and how it interacts with external applications and Users. Before running Reactor, you should familiarize yourself with the Configuration

information in the Administration Guide and verify that the default settings are correct for your system.

6.2. Reactor Enterprise Bundle

6.2.1. Overview

The following table lists the directories included in the Reactor Enterprise distribution package. The Reactor EAR file is already deployed to the JBoss application server located in the `C:\ReactorEnterprise-6.0\jboss-3.2.5\server\default\deploy` directory.

6.2.2. Structure

This is the structure of the directories created by Reactor Enterprise.

Directory	Contents
\commandline	Reactor command line interface files.
\docs	Reactor documentation files.
\jboss-3.2.5	Reactor Engine and Process Manager deployed and preconfigured in a distribution of the JBoss 3.2.5 Application Server.
\lib	Reactor configuration files.
\samples	Examples that illustrate client code and web applications that interact with the Reactor Engine.
\SQL	Database schema files that enable Reactor to use a broad range of open source and commercial databases.
\studio	Reactor Studio Application - a graphical design tool for authoring Processes.

6.3. Application Server Deployment

6.3.1. Using BEA Apllication Servers

6.3.1.1. WebLogic 7.0

1. If you are using an Oracle data source, uncomment both of the lines in the Reactor configuration file named `ProcessObjectService.properties` that begin with `database.storageFactory` or `database.clobUtils`.
2. WebLogic binds to TCP port 7001 by default. Use this to create your FrontDesk URL as described in the general deploy instructions.
3. Choose the internal authentication approach outlined in the JAAS section in the general deployment instructions.
4. WebLogic needs the Xerces parser in the system classpath. Copy `xerces.jar` from `ReactorServer-5.5.3-RC3.ear` to the filesystem and add it to WebLogic's system classpath by editing the WebLogic startup script.
5. WebLogic must be run with the `-Djava.security.manager` option. Prepare for this by editing the WebLogic startup script.
6. WebLogic requires access to the `ReactorAuthentication.jar` libraries in the system classpath. Extract `ReactorAuthentication.jar` from the `ReactorServer-5.5.3-RC3.ear` file and place it in a directory of your choosing. Edit the WebLogic startup script to include this JAR in the startup classpath.
7. Grant java.security.AllPermission to the Reactor code by editing the `WEBLOGIC_HOME/server/lib/weblogic.policy` file.

Note:

For more information, go to Configuring a Data Source in BEA WebLogic 7.0 (http://edocs.bea.com/wls/docs70/adminguide/jdbc.html-1074927)

6.3.1.2. WebLogic 8.1

1. If you are using an Oracle data source, uncomment both of the lines in the Reactor configuration file named `ProcessObjectService.properties` that begin with `database.storageFactory` or `database.clobUtils`.
2. WebLogic binds to TCP port 7001 by default. Use this to create your FrontDesk URL as described in the general deployment instructions.
3. After creating a DataSource, under *Configuration:Connections:Advanced Options*:
   - Check `Keep XA Connection Till Transaction Complete`.
   - Check `Supports Local Transaction`.

Note:

For more information, go to Configuring a Data Source in BEA WebLogic 8.1 (http://edocs.bea.com/wls/docs81/ConsoleHelp/jdbc.html)

6.3.2. Using IBM Application Servers

6.3.2.1. WebSphere 4.0

Reactor's WebSphere distribution contains J2EE 1.2 deployment descriptors, and therefore does not use the JMS version of the Policy Execution Service. Follow the instructions for configuring the session bean version of the Policy Execution Service.

R5 has been tested on WebSphere 4.0 using WebSphere's APPLICATION classloading scheme, which uses a single classloader for each J2EE Application. The instructions below are written for use with this classloading scheme.

WebSphere's APPLICATION classloading scheme requires `jdom.jar` to be in WebSphere's system classpath rather than within the EAR file. Otherwise an error message about violation of classloading constraints results. Remove `jdom.jar` from the EAR file and put it in WebSphere's system classpath. If, after placing `jdom.jar` in WebSphere's classpath, you still receive the error, try putting `jdom.jar` in the `/lib/ext` directory of your JRE instead (`/AppServer/java/jre/lib/ext` by default).

Due to an unusual implementation of JNDI in WebSphere 4, it is necessary to set the value of the `use.websphere.workaround` property to `TRUE` in the `TimerService.properties` Reactor configuration file.

WebSphere's deployment tool offers to precompile JSPs. Set this option to *No*. During the deployment process, WebSphere creates temporary files and directories, which it uses for various steps of the deployment (such as calling rmic to create the EJB stub classes. In fact, WebSphere creates so many directories and nests them so deeply that the path to these temporary files may exceed the length limit imposed by some operating systems, such as Windows 2000. If this occurs, there are a couple of ways one can go about shortening the pathnames to get them under the length limit. One is to change the filename of `ReactorEJB.jar` to `R5.jar`. This requires modifying the `manifest.mf` files in the EJB, JAR and WAR files as well. Another option is to use the windows command *SUBST*. For example, `SUBST X: C:\WebSphere\AppServer` chops ~20 characters of the front of the path. If SUBST is used, it may also be necessary to modify the values of the variables set in WebSphere's startup scripts to use the new drive letter.

After deploying any application to WebSphere, it is necessary to regenerate WebSphere's Apache plug-in configuration to access the web modules(s) of the newly installed application. Reactor is not an exception to this rule. Choose the internal authentication approach outlined in the JAAS section in the general deploy instructions.

Note:

For more information, go to Configuring a Data Source in IBM WebSphere 4.0 (http://www-3.ibm.com/software/webservers/appserv/doc/v40/aes/infocenter/index.html)

6.3.2.2. WebSphere 5.0

Before deploying the R5 ear to WAS, a number of resources must be prepared. These steps assume you are using the Administrative Console for configuration. Note that it may be necessary to save changes to the master configuration between these steps.

1. Configure a JMS Topic Connection Factory and Topic Destination. Use the WebSphere JMS Provider unless you have reason to do otherwise. Select the QUEUED port and enable XA transactions for the connection factory.
2. Configure a JMS Listener Port under Servers > Application Servers > YourServer > Message Listener Service > Listener Ports. Sample settings:
   - Name: ReactorListenerPort
   - Initial State: Started
   - Connection factory JNDI Name: Should match the JNDI name configured above
   - Destination JNDI Name: Should match the JNDI name configured above
   - Maximum Sessions: 5
   - Maximum Retries: 10
   - Maximum Messages: 5

You will need to configure the Application using the name of this listener later.

Configure Java 2 Connector security for database access. Navigate to Security > JAAS Configuration > J2C Authentication Data and create a new Alias. Configure the alias with the database user's credentials.

If using the WebSphere MSSQL 2000 JDBC provider, you will need to install the Microsoft JDBC driver for SQL Server 2000 and also run the scripts to build the extended stored procedures. The .dll files provided with the driver must be copied into SQL Server's `binn` directory for the extended sprocs to work.

WebSphere 5.0 deploys applications to `http://host:80 and http://host:9080` by default. Be sure to update the FrontDesk URLs in the EAR file before deploying.

Install the application. Please take note of the following settings:

- Use Binary Configuration: checked
- Deploy EJBs: checked
- Use Metadata From Binaries: checked
- For the ReactorServerEJB module, provide the listener port name you created above.
- For each EJB requiring a JNDI name, set the JNDI name to be the same as the EJB name. For example: reactor/ReactorEJBFrontDesk.

Note:

For more information, go to Configuring a Data Source in IBM WebSphere 5.0 (http://publib7b.boulder.ibm.com/wasinfo1/en/info/aes/ae/cdat_datasor.html)

6.3.3. Using JBoss Application Servers

6.3.3.1. JBoss 3.0.4

The following steps are necessary to properly deploy Reactor to work with the JBoss 3.0.4 Application Server.

1. Find or create a connection-manager descriptor XML for the database you wish to deploy. Examples can be found in the `C:\ReactorEnterprise6-JBoss3.2.5\jboss\docs\examples\jca\` directory. See the section on Configuring a Datasource for details.
2. Reference the appropriate security module for your data source inside the `login-config.xml` file, if appropriate.
3. Optionally, configure the login module(s) for the security domain of your choosing (CSV is the default) in `login-config.xml` as well. See the section about JAAS for details.
4. In `jbossmq-service.xml`, create a new connection factory with the OILServerILService mbean for the Policy Execution Service JMS.
5. Likewise, in `jbossmq-destinations-service.xml`, create a new topic. Note that the name attribute of the JMS Topic in this file should not include the `topic/` prefix. For example, to create a Topic named `topic/ReactorEvents`, use a name attribute of `name="jboss.mq.destination:service=Topic,name=ReactorEvents"`.
6. Deploy the EAR to the deploy directory.

6.3.3.2. JBoss 3.2.1

The deployment instructions for JBoss 3.2.1 are almost the same as for JBoss 3.0.4. However, for version 3.2.1 it is necessary to set the RecursiveSearch attribute to `TRUE` in the JBOSS configuration file `conf/jboss-service.xml`.

6.3.3.3. Datasources

To configure a Datasource in JBoss, follow the steps below.

1. Add the JAR file(s) for the JDBC driver you want to use to the JBoss classpath. Put them in the `\jboss321\server\default\lib\` directory to accomplish this.
2. In the `\jboss321\docs\examples\jca\` directory, find the sample XML file corresponding to the database you want to use.
3. Edit the XML file:
   - Change the content of the amp#60;attribute name="JndiName"amp#62; tag to change the JNDI name of the Datasource if desired. If you are using this Datasource to store Process data for Reactor, the JNDI name must match the value of the `ds.name`

property in Reactor's `ProcessObjectService.properties` configuration file.
- Change the content of the amp#60;config-property name="ConnectionURL" type="java.lang.string"> tag to a URL that can be used to connect to the database.
- Change the content of the amp#60;config-property name="DriverClass" type="java.lang.string"> tag to the classname of the JDBC driver you are using to connect to the database.
- Change the content of the amp#60;config-property name="Username" type="java.lang.string"> tag to the username that should be used to connect to the database.
- Change the content of the amp#60;config-property name="Password" type="java.lang.string"> tag to the password that should be used to connect to the database.

4. Remove any other data source XML files from the JBoss deploy directory (`\jboss321\server\default\deploy\`) that have the same JNDI name as the one specified for the new data source. For example, in Reactor's JBoss bundle distribution, the deploy directory contains a `pointbase-service.xml` file that defines a data source for Pointbase. It has a JNDI name of *ReactorDS*, so it would have to be removed before any new data source could be added with a JNDI name of *ReactorDS*.
5. Put the modified XML file for the new data source in the JBoss deploy directory (`\jboss321\server\default\deploy\`) and start JBoss.

6.4. Database Deployment

6.4.1. Using A DB2 Database With Reactor

The following steps are necessary to properly deploy Reactor to work with a DB2 database. These steps have been tested with version 8.2 of DB2.

1. Start the *Control Center* tool that comes with DB2 (choose the *Advanced* view if you're given the choice).
2. Select the *Instance* where you're going to establish the Reactor Database. In the navigation pane, traverse the *All Systems* node, the appropriate host, the *Instances* node, and finally the instance node.... (let's call it *db2user1* for now).

> **Note:**
> If you're starting from scratch, you probably set up the *db2user1* instance as part of the DB2 setup.

3. Start the instance *db2user1* by executing a right click and selecting the *Start* option. If the instance is already started, you'll get a warning to that effect.... (it's already "active"). That's fine.
4. Create a new database within the instance. Select the *Databases* node under the instance

and right click on it. Choose the *Create Database* | *Standard* option.

1. Set the name of the new database (for example: REACTOR). Note that the name is limited to 8 characters.
2. Accept the *creation defaults* by clicking the *Finish* button.
3. If all goes well, the creation will be successful and you'll be asked whether or not you want to launch the *Configuration Advisor*. Click the *No* button - if you want to reconfigure the database then you can do it later.

5. Create a *Buffer Pool* with a 16K page size (this is necessary for some of the Reactor schema tables). Traverse to the REACTOR/Buffer Pools node, right click, and choose the *Create* option.
   1. Set the name of the new *Buffer Pool* (for example: REACTOR_BP).
   2. Set the *Page Size* to 16 (instead of the default which is 4).
   3. Accept the other *creation defaults* by clicking the *OK* button. You may be warned that the *Buffer Pool* cannot be created until the next startup... (possibly due to insufficient memory). That's okay. If you do get this warning, you'll have to click the *Cancel* button to get out of the dialog.
6. Create a *Table Space* that uses the new *Buffer Pool* (this is also necessary for some of the Reactor schema tables). Traverse to the REACTOR/Table Spaces node, right click and choose the *Create* option.
   1. Set the name of the new *Table Space* (for example: REACTOR_TS).
   2. Accept the default for the *type* of table space (i.e. Regular).
   3. Select the new *Buffer Pool* for the new *Table Space* (i.e. select the REACTOR_BP buffer pool).
   4. Select the *Database-managed space (high performance)* option for the *space management system*.
   5. Add a container for the *Table Space*. Click the *Add* button and identify the directory and filename where you wish the *Table Space* data to be stored (for example: /apps/db2/tablespace/reactor_ts.db2).

   Note:
   If you're going to have a lot of Reactor workflow Instances in the database, you'll probably want to increase the size of the file to something like 100MB (instead of the default 20MB).

   6. Accept the other *creation defaults* by clicking the *Finish* button.
7. Load the Reactor schema into the REACTOR database.
   1. Open the Command Editor by Selecting *Tools* | *Command Editor* in the main menu section.
   2. Add the REACTOR database target by clicking the *Add* button and selecting the REACTOR database.
   3. Open the `C:\ReactorEnterprise6.0-Jboss3.2.5\SQL\DB2\Create-Processes.sql`

file by clicking the folder open icon, clicking the *Selected* | *Open* option from the main menu, or executing the *Ctrl-O* keyboard shortcut.

4. Execute the `Create-Processes.sql` file by clicking the execute icon, clicking the *Selected* | *Execute* option from the main menu, or executing the *Ctrl-Enter* keyboard shortcut.
5. Repeat steps 7b and 7c for the `Create-Attachments.sql`, `Create-Audit.sql`, and `Create-Timers.sql` files in the `C:\ReactorEnterprise6.0-Jboss3.2.5\SQL\DB2` directory. You'll be asked whether or not you want to replace or append the subsequent SQL statement sets to the previous SQL statement sets. Click the *Yes* button to replace the current input (to avoid re-executing the previous SQL statement sets).

Note:

It's a good idea to clear the results after each execution - however, it's not necessary.

The Reactor schema is now in place within the REACTOR database. Once you configure your Application Server to use it as your Reactor datasource, you will be able to use Reactor.

6.4.2. Using The Bundled HSQL Database Server With Reactor

If you are using one of the Reactor bundles, the HSQL database is already included and configured to run with Reactor.

You do not need to do anything further.

6.4.3. Using An InstantDB Database With Reactor

FIXME (To Be Completed):

This database is currently being tested with the latest release. Instructions for using it with Reactor will be updated soon.

6.4.4. Using An MS SQL Server Database With Reactor

These steps have been tested with Microsoft SQL Server 2000 database and the JDBC driver developed by the jTDS Project.

1. Install SQL Server, if you haven't already.
2. Create a user that will access Reactor data.
3. Create a new database for Reactor data or use an existing one.
4. Grant the Reactor user privilege to read, write, and delete records from the Reactor database. This does not need to be done explicitly if the database was created by the Reactor user.

5. Load the following four MS SQL schemas from the `C:\ReactorEnterprise6.0-JBoss3.2.5\SQL\MSSQLServer` directory into the database using the osql utility:
   `Create-Processes.sql`
   `Create-Attachments.sql`
   `Create-Audit.sql`
   `Create-Timers.sql`
6. Get the JTDS SQL Server JDBC driver (v0.9.1) (http://jtds.sourceforge.net/index.html) from SourceForge.
7. Put the JDBC driver in your application server's classpath. For JBoss, copy the JAR file to the jboss321\server\default\lib directory.
8. Create a DataSource in the application server configuration. See the instructions above for the application server being used.
9. Use the Reactor Configuration Tool to edit the `hibernate.cfg.xml` file in the Reactor EAR file. Modify the *dialect* property to:

```
 <property
name="dialect">net.sf.hibernate.dialect.SQLServerDialect</property>
```

10. Use the Reactor Configuration Tool to edit the `Portal.properties` file in the Reactor EAR file.
    - Change the *frontdesk.url* property to be the actual IP address. It fixes a map viewer issue.
    - Near the bottom ,there is a section that specifies the class that will be used to generate information for the reports. Comment out the line that reads
      `com.oakgrove.reports.queryReportClass=com.oakgrove.reports.StdReport`
      and uncomment the line that reads
      `com.oakgrove.reports.queryReportClass=com.oakgrove.reports.MSSQLRepo`
11. Use the Reactor Configuration tool to modify the `Quartz.properties` file in the Reactor EAR file. Modify the `org.quartz.jobStore.selectWithLockSQL` parameter to `SELECT * FROM {0}LOCKS UPDLOCK WHERE LOCK_NAME = ?`. Now add the following line and comment out the similar line referencing HSQL.

```
 org.quartz.jobStore.driverDelegateClass =
org.quartz.impl.jdbcjobstore.MSSQLDelegate
```

Note:

You may have to add the following two lines to the `java.policy` file in the `/jre/lib/security` subdirectory of the Java 2 Platform installation directory.

```
permission java.util.PropertyPermission "java.naming.*", "read,write";
permission java.net.SocketPermission "*.microsoft.com:0-65535", "connect";
```

6.4.5. Using A MySQL Database Server With Reactor

FIXME (To Be Completed):

This database is currently being tested with the latest release. Instructions for using it with Reactor will be updated soon.

6.4.6. Using An Oracle Database With Reactor

These steps have been tested with Oracle versions 8 and 9.2.

1. Install Oracle, if you haven't already.
2. Create a user that will access Reactor data.
3. Create a new database for Reactor data or use an existing one.
4. Grant the Reactor user privilege to read, write, and delete records from the Reactor database. This does not need to be done explicitly if the database was created by the Reactor user.
5. Load the schema into the database (by using SQL*Plus, for example) from the schema SQL files located in the `%ReactorHome%\SQL\Oracle` directory.
6. Get the Oracle JDBC driver (http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/index.html) that matches your version of Oracle database.
7. Put the Oracle JDBC driver in your application server's classpath. For the JBoss, copy the JAR file to the `jboss/server/default/lib` directory.

Note:

Many application servers come with JDBC classes and Oracle JDBC drivers already in the classpath. Unfortunately, there are often conflicts between incompatible versions of these classes/drivers. It may be necessary to find and remove older versions of the Oracle JDBC driver from your application server's classpath.

8. Create a dataSource in the application server configuration. See the instructions above for the application server being used.
9. Locate (or build) the `JdbcOraWrapper.jar` file. You can find it in the `...\lib` directory of the JDBC distribution, or you can build it from the source if you have it. The source is located in the `...\resources\thirdparty\orawrap` directory.
10. Place the orawrapper driver (`JdbcOraWrapper.jar`) and the oracle driver in `%ReactorHome%\jboss-3.2.5\server\default\lib` directory.
11. Copy `oracle-ds.xml` from the `%ReactorHome%\jboss-3.2.5\docs\examples\jca` directory into the `%ReactorHome\jboss-3.2.5\server\default\deploy` directory.

```
Example:
```

```
 <jndi-name>DefaultDS</jndi-name>
<connection-url>jdbc:oracle_clob:thin:@[hostname_oracle_server]:1521:[SID]</connecti
 <driver-class>JdbcOraWrapperDriver</driver-class>
 <user-name>[username]</user-name>
 <password>[password]</password>

NOTE: "JdbcOraWrapperDriver" for driver-class and "oracle_clob" for
connection-url properties.
```

12. Delete the `hsqldb-ds.xml` file.
13. Use the Reactor Configuration Tool to edit the `hibernate.cfg.xml` file in the Reactor EAR file.
    - Modify the *dialect* property to:

      ```
       <property
      name="dialect">net.sf.hibernate.dialect.Oracle9Dialect</property>
      ```

    - Add the following line in the properties section:

      ```
       <property name="hibernate.jdbc.batch_size">0</property>
      ```

    - Modify the path for the mapping files to be *Oracle/** (for example: `Oracle/HibProcess.hbm.xml`).
14. Use the Reactor Configuration Tool to edit the `quartz.properties` file in the Reactor EAR file.
    - In the Configure JobStore section use the following setup:

      ```
       org.quartz.jobStore.driverDelegateClass =
      org.quartz.impl.jdbcjobstore.oracle.OracleDelegate
      ```

    - Comment out the following line:

      ```
       org.quartz.jobStore.selectWithLockSQL
      ```

15. Use the Reactor Configuration Tool to edit the `portal.properties` file in the Reactor EAR file.
    - Uncomment the following line:

      ```
      #com.oakgrove.reports.queryReportClass=com.oakgrove.reports.OracleReportQuery
      ```

    - Comment out the following line:

      ```
      com.oakgrove.reports.queryReportClass=com.oakgrove.reports.StdReportQuery
      ```

16. Use the Reactor Configuration Tool to edit the `audit-trail.properties` file in the Reactor EAR file. Uncomment the following line:

```
#audit.storage.class: com.oakgrovesystems.audit.OracleAuditStorage
```

6.4.7. Using A Pointbase Database With Reactor

FIXME (To Be Completed):

This database is currently being tested with the latest release. Instructions for using it with Reactor will be updated soon.

6.4.8. Using A PostgreSQL Database With Reactor

FIXME (To Be Completed):

This database is currently being tested with the latest release. Instructions for using it with Reactor will be updated soon.

6.4.9. Using A Sybase Database With Reactor

FIXME (To Be Completed):

This database is currently being tested with the latest release. Instructions for using it with Reactor will be updated soon.

6.5. Directory Service Deployment

6.5.1. User Access

There are two mechanisms available in Reactor to authenticate Users, CSV Files containing Usernames and Passwords or an LDAP. Reactor is distributed with a set of simple CSV Files that define the key attributes of each permitted User. This section outlines how to use the CSV files that are included with Reactor.

6.5.1.1. CSV Files

Use the Reactor Configuration Tool to modify the evaluation CSV Files by editing these files in the `ReactorEnterprise6.ear` file. Users may be added, deleted or modified with this tool. The Title and Group properties for each User may also be set with this tool. The table below shows the files affected by any modifications.

File Name	Data Format

EvalUsers.csv	username, password, first name, last name, email
EvalTitles.properties	title: comma-separated list of usernames
EvalGroups.properties	group: comma-separated list of usernames

The Server must be restarted to make the changes effective.

Reactor comes with four users already configured. The following table shows the default Users in this distribution, with their respective Title and Group settings. When adding new Titles/Groups, be aware that spaces are not allowed in the Title or Group designators.

Username	Password	Title	Groups
admin	admin		
alice	alice	CEO	Management
bob	bob	CTO	Management, Engineering
charlie	charlie	CFO	Management, Accounting
frank	frank	CIO	Management, Engineering
judy	judy	Accountant	Accounting, Payroll
susan	susan	Developer	Engineering
linda	linda	CMO	Management, Marketing
jose	jose	CSR	Customer_Service
ruth	ruth	CSR_Manager	Management, Customer Service
tom	tom	Shipping_Clerk	Operations
henry	henry	HR_Manager	Management, HR

6.5.2. Using an LDAP With Reactor

There are two mechanisms available in Reactor to authenticate Users, CSV Files containing Usernames and Passwords or an LDAP. This section covers how to configure Reactor to work with an LDAP.

6.5.2.1. Using LDAP For Authentication And Directory Service

Authentication

To authenticate usernames and passwords against an LDAP directory instead, edit the `AuthenticationService.properties` configuration file. Modify the `login.module` value to be *LDAP* instead of *CSV*. The line specifying the csvFile is no longer needed, so you can delete it or comment it out.

Edit the `ldap.url` property to specify the URL of the LDAP directory. For example:

```
ldap.url: ldap://localhost:389
```

Edit the `ldap.dn.mapping` property to specify how Reactor should map a Username string to a fully qualified LDAP dn. The special string ${username} will be replaced by the Username of the User attempting to authenticate. For example, if the LDAP dn for a User with Username bsmith is uid=bsmith,ou=People,dc=mycompany,dc=com then the appropriate configuration would be:

```
ldap.dn.mapping: uid=${username},ou=People,dc=mycompany,dc=com
```

Edit the ldap.mechanism property to specify the LDAP authentication mechanism, which is optional if the mechanism is simple.

Directory Service

Reactor also integrates with a directory service, which provides Policies with access to information about the people and groups in an organization. For example, a Policy can send an email to the assignee of a Process by looking up the assignee's email address from the directory service by calling the `lookupEmail()` method of a `com.oakgrovesystems.reactor.util.Navigator` object. Programs within the Reactor Engine assume that Users in the directory service have associated with them a unique string (Username), which can be used to retrieve information specific to that User. To configure the Reactor Engine to use an LDAP directory service, make the following changes to the Reactor Engine configuration files:

1. Modify the `DirectoryService.properties` file. The `moduleClass` property should be:
   `com.oakgrovesystems.reactor.services.directory.LdapDirectoryModule`
   The `directory.config` property should be:

`conf/ldap.config`

2. Modify the `LdapDirectoryModule.properties` file. Set the url property to the URL of your LDAP server. For example:
   `ldap://ldap.mycompany.com:389`
   If your server allows directory lookup or search requests from anonymous Users, set the authMechanism property to none. Otherwise, set the authXxx properties to allow the Reactor Engine to authenticate to your LDAP server to lookup or search for Users, Groups and Titles.

See the comments in the `LdapDirectoryModule.properties` file for more details about the other configuration properties required.

Configuring Reactor To Authenticate To Your LDAP Server

If your LDAP server does not allow directory lookup or search requests from anonymous users, you will have to configure the Reactor Engine to authenticate to your LDAP server. Different LDAP servers support different authentication mechanisms, but a mechanism supported by most servers is the use of a clear text Username and Password. To use this mechanism, edit the `LdapDirectoryModule.properties` file:

1. Set the value of the authMechanism property to simple.
2. Set the value of the authUsername property to the Username you would like the Reactor Engine to use for authentication.
3. Set the value of the authMechanism property to the Password associated with that Username.

Other values for the authMechanism property can be used, but may require additional system properties to be set in the application server's JVM, or may require additional configuration of the LDAP server. The value of the authMechanism property becomes the value of the Context.SECURITY_AUTHENTICATION used to create a JNDI initial context. For more information on other authentication mechanisms, see Sun's documentation on JNDI and LDAP security.

Search vs. Lookup

The Reactor Engine supports two different methods of retrieving User and Group information from an LDAP server: lookup and search. The lookup method involves retrieving the attributes of a User or Group object by its LDAP name. The search method involves searching an LDAP context (specified by the startingContextName property) and all of its sub contexts (recursively) for objects matching a given search filter.

Each method has advantages over the other and they are appropriate for different LDAP configurations. For example, Users may be stored in the LDAP directory in such a manner that it is impossible to write a lookupString that will allow access all the users. On the other

hand, if such a lookupString can be written, it is likely to be faster to lookup users by name than to search for them (although actual performance will differ between LDAP servers).

Groups Omitted

For performance reasons, the current implementation of the LdapDirectoryModule does not include the Groups in User objects returned by methods of the DirectoryModule interface. If Group information for a User is needed, a separate call to getGroupsForUser() can be made.

6.6. Advanced Topics

6.6.1. Audit Trail Setup

6.6.1.1. EAR File Source Bundle

Warning:

The following Audit Trail set up instructions **DO NOT** apply to the evaluation bundle. The evaluation bundle is already set up with the Audit Trail feature installed and enabled. This section only applies to the EAR file source bundle.

If you're setting up the Audit Trail feature from the EAR file source bundle, you will need to complete the following steps (MS SQL Server is used for the example).

1. Create the database tables for use of audit trail. The schema can be found in the directory `C:\ReactorEnterprise6-JBoss3.2.5\docs\sql\MSSQLServerAuditTrail.sql`.
2. In AuditTrail.properties, please provide the datasource value which contains the audit trail tables in the property audit.ds.name. Also, change the value of all the events to be logged to true.
3. In FrontDesk.properties, please provide the correct url for the property frontdesk.url.
4. Extract the following jar files to a directory (e.g. c:\audittrail). These files are located inside the ReactorPolicies6.jar file.
   - jdom.jar
   - ReactorCommon6.jar
   - ReactorConfig.jar
   - ReactorEJB.jar
   - ReactorPolicies6.jar.
5. At the `C:\audittrail\>` command prompt, execute the command

```
 java -classpath jdom.jar;ReactorCommon6.jar;ReactorConfig.jar;
 ReactorEJB.jar;ReactorPolicies6.jar com.oakgrovesystems.audit.Audit
admin admin
```

Note:

The Reactor Engine must be running when executing this command. Do not leave spaces after the semi-colons separating the JAR file names. The above example is split onto two lines to facilitate screen viewing.

6.6.1.2. Installation And Configuration With Other Databases

In addition to the included HSQL database, Reactor supports the use of the following databases to handle the Audit Trail function.

- DB2
- MS SQL Server
- MySQL
- Oracle
- PostgreSQL
- Sybase

To use one of these databases with the Audit Trail feature, complete the following steps:

1. Prepare a database for Audit Trail. The Audit Trail Policies write their Events to several tables within your database. For the Reactor supported databases, SQL source code to generate the appropriate tables is provided within the `C:\ReactorEnterprise6-JBoss3.2.5\docs\sql\` directory of your Reactor distribution. For example, if PostgreSQL will house your Audit Trail database, use your data import tool to import the file: `C:\Reactor5.5.3-JBoss3.2.1\docs\sql\PostgresSQLAuditTrail.schema`
2. Configure the data source with your application server. Per your application server's documentation, configure an appropriate JNDI-accessible data source for the Audit Trail Policies to access, following the example: `java:/MyAuditTrailDS`
3. Configure the `AuditTrail.properties` file. The file `AuditTrail.properties` contains all of the configuration parameters for Reactor Audit Trail. This file is located inside `ReactorConfig.jar`, which you just extracted. Manually extract and edit the `AuditTrail.properties` file. Modify the property `audit.ds.name` to correspond to the JNDI name to which you've bound the data source you created in step 2 (i.e. `java:/MyAuditTrailDS`). Additionally, while editing this file you can enable or disable any particular Policies that you wish. By default, all seven of the Events will be recorded. Set any of the Policies to `FALSE` to disable them.
4. Tell the Audit Trail application how to get to Reactor. The Audit Trail installer communicates with Reactor via Reactor's XML-over-HTTP interface. If you are executing the installer on the same machine, it is likely that the appropriate Front Desk

URL for Reactor already is configured inside `ReactorConfig.jar`, which you extracted above.

If you intend to execute the installer from a remote machine, or it does not seem to connect, you should verify that you're using the appropriate URL. Open the Reactor Configuration Tool and check `frontDesk.properties`. If necessary, adjust the `frontdesk.url` property to point to your Reactor Engine installation. As an example, within the evaluation bundle, the Front Desk URL is:

(http://localhost:8080/ReactorServer/FrontDesk)

The following table details the available AuditTrail database schemas.

Database	Schema File
DB2	DB2AuditTrail.schema
HSQL	HSQLAuditTrail.schema
Oracle	OracleAuditTrail.schema
PostgreSQL	PostgreSQLAuditTrail.schema
MS SQL Server	MSSQLServerAuditTrail.sql
Sybase	SybaseAuditTrail.schema

6.6.2. Configuring The Quartz Scheduler

6.6.2.1. Reactor Bundles

The Reactor bundles come pre-configured to use Quartz. No additional configuration is necessary to start using the scheduling functionality within Reactor.

6.6.2.2. DB Schema

The Quartz scheduler requires database tables to persist scheduling information. There are scripts contained in the `\SQL` directory to create the Quartz schema.

6.6.2.3. Property File

The Quartz scheduler is configured through a property file called `Quartz.properties` stored within the Reactor EAR file. The Reactor bundle includes a tool for editing the configuration files without manually extracting the contents of the EAR and JAR files. To use this tool, run either the `configure.bat` script on Windows or the `Configure.sh` script on UNIX. This will open the configuration files in the Reactor EAR file.

Open the configuration tool and select the `quartz.properties` from the list of configuration files in the left side of the window.

Setting the Instance Information

Set the `org.quartz.scheduler.instanceName` to a meaningful name. Every Quartz instance used by different applications should have a unique name. The JBoss evaluation bundle uses *ReactorScheduler*.

Set the `org.quartz.scheduler.instanceId`. This instance id should be unique within the application. If you have an application that uses more than one instance of the Quartz scheduler (i.e. in a clustered environment), each instance must be unique. The JBoss bundle uses *1*.

Set the `org.quartz.scheduler.rmi.export` and `org.quartz.scheduler.rmi.proxy` to both be `FALSE`. These settings are only used when quartz is running in stand-alone mode.

Setting The Thread Pool

Quartz uses a thread pool for executing jobs. In Reactor, these jobs consist of sending a policy execution message to the Policy Execution Service, and as such, the jobs are extremely short-lived.

Set the `org.quartz.threadPool.class` to `org.quartz.simpl.SimpleThreadPool`.

Set the number of threads by setting `org.quartz.threadPool.threadCount` to an integer value. A rule of thumb for setting this value is the number of concurrent timer expirations divided by 30. It is not recommended to set this value less than 2.

Set the thread priority by configuring `org.quartz.threadPool.threadPriority` to an integer between 1 and 9. These values correspond to the thread priority constants in `java.lang.Thread`. It is recommended you leave this value at 5 (Normal priority).

Setting the job store

Quartz can be used with a number of databases in a number of configurations. This section describes how to set up Quartz to work with the Reactor database.

Quartz has two transaction mechanisms. Containter managed transactions and application managed transactions. Quartz should be set up to use it's own transaction, so `org.quartz.jobStore.class` to

`org.quartz.impl.jdbcjobstore.JobStoreTX`.

Configure the appropriate JDBC delegate for your database. Different JDBC drivers have different ways of dealing with BLOBS and CLOBS. Set `org.quartz.jobStore.driverDelegateClass` to one of:

```
 com.oakgrovesystems.reactor.scheduling.quartz.PointbaseJDBCDelegate
(Pointbase)
 org.quartz.impl.jdbcjobstore.StdJDBCDelegate (Many JDBC-compliant drivers)
 org.quartz.impl.jdbcjobstore.MSSQLDelegate (Microsoft SQL Server,Sybase)
 org.quartz.impl.jdbcjobstore.PostgreSQLDelegate (PostgreSQL)
 org.quartz.impl.jdbcjobstore.WebLogicDelegate (WebLogic)
 org.quartz.impl.jdbcjobstore.oracle.OracleDelegate (Oracle)
```

Quartz names a dataSource in its configuration file. This dataSource is not to be confused with a J2EE DataSource. The Quartz data source is merely a set of configuration settings for a particular Quartz data store.

Choose a name for the Quartz dataSource by setting `org.quartz.jobStore.dataSource`. The JBoss evaluation bundle uses R5DS. This value is case sensitive and will be used extensively throughout the rest of the configuration file.

Specifiy a prefix for the Quartz tables. Unless you have modified the SQL scripts with this bundle, you should set `org.quartz.jobStore.tablePrefix` to *qrtz_*. Quartz prepends this value to table names, such as `TRIGGERS` to end up with `qrtz_TRIGGERS`.

Warning:

You may get errors if you are using a case-sensitive database, and do not have the case set correctly.

Specifiy if Quartz is in a clustered app server environment by setting `org.quartz.jobStore.isClustered` to `TRUE` or `FALSE`. For evaluation bundles, you will want to set this to `FALSE`.

Setting the Database

Quartz needs at least 2 connections to the database to function properly. One connection is used for polling for the next job, and the other connection is used for job execution. You must configure these connections.

The recommended way of providing a DB connection to Quartz is to provide a J2EE datasource in the JNDI tree. You can specify this by setting `org.quartz.dataSource.R5DS.jndiURL` where `R5DS` is your Quartz dataSource

configured above. It is recommended that you use the same J2EE datasource as you provide for Reactor, as long as both the Reactor tables and the Quartz tables are in the same schema. The JBoss evaluation bundle uses `java:/ReactorDS` but it may be different depending upon your application server.

On some application servers, Quartz is able to create a J2EE dataSource dynamically. If you would like Quartz to dynamically create your J2EE DataSource, configure the following parameters:

```
org.quartz.dataSource.R5DS.driver
org.quartz.dataSource.R5DS.URL
org.quartz.dataSource.R5DS.user
org.quartz.dataSource.R5DS.password
org.quartz.dataSource.R5DS.maxConnections
```

Where `R5DS` is the quartz dataSource configured above.

Warning:

Do not configure both a JNDI datasource and a dynamic datasource!

6.6.2.4. More Info

More information on the Quartz Scheduler can be found at the OpenSymphony (http://www.opensymphony.com) web site. Additional documentation and user forums can be found there.

6.6.3. Reactor In a Clustered Environment

Here are the steps required to set up Reactor in a clustering environment.

Note:

n1 refers to node 1
n2 refers to node 2

1. Install the product in multiple independent directories. The installations can be on the same machine or on different machines.
2. Decide on the database that the systems will share. An example is to have an HSQL database run within one of the nodes (TCP mode on **n1**) and to have the other node(s) utilize it, much as they'd use Oracle (**n2**).

```
n1/jboss-3.2.5/server/default/deploy/hsqldb-ds.xml

n2/jboss-3.2.5/server/default/deploy/hsqldb-ds.xml
```

3. Update `ReactorConfig.jar` in the product installations with a revised `ReactorServices.properties` file.

```
n1/jboss-3.2.5/server/default/deploy/Reactor-6.01B.ear/ReactorConfig.jar
                /ReactorServices.properties

n2/jboss-3.2.5/server/default/deploy/Reactor-6.01B.ear/ReactorConfig.jar
                /ReactorServices.properties
```

4. Configure the jndi properties and the jgroups properties for each of the nodes. These properties files are referenced by the `ReactorServices.properties` file. The jndi properties provide the information that any node in the cluster should use to interact with the node. The jgroups properties define the *protocol stack* that jgroups will use to maintain the *cluster group*.

Warning:

The jgroups configuration files are not really straightforward and should be manipulated with care. Note that the jgroups files need to identify a *bind_addr* for the UDP protocol which matches the host running jgroups. It's an optional parameter but jgroups may not work without it. Also note the multicast address and port identifier as it relates to the next step.

```
n1/nfs/data/adam-node.jgroups.xml

n1/nfs/data/adam-node.jndi.properties

n2/nfs-adam/data/eve-node.jndi.properties

n2/nfs-adam/data/eve-node.jgroups.xml
```

5. Make sure that your system(s) support multicasting and that the firewalls on each machine allow the multicast traffic through. Unfortunately, jgroups cannot complain when it doesn't get the traffic because it doesn't know "who" is supposed to join its group. JGroups provides a little utility to help you ensure that multicasting can be sent from one

host and received on another.

Note:
If you're using more than one machine then you should make sure all paths support it.

6. Set the *BindAddress* values within JBoss. This will allow the JBoss services to be accessible outside of the machine. Most, if not all, of the *BindAddresses* are in the `jboss-service.xml` file.

Note:
The default address is 127.0.0.1 and it is not publicly accessible.

```
n1/jboss-3.2.5/server/default/conf/jboss-service.xml
n2/jboss-3.2.5/server/default/conf/jboss-service.xml
```

7. Set the RMI related properties in system properties at JBoss startup so that *remoteable* objects will be bound to the correct address. Otherwise, they will be bound to 127.0.0.1 and be inaccessible, even though the RMI service is accessible. These changes should be made in the `run.sh` script.

```
n1/jboss-3.2.5/bin/run.sh
n2/jboss-3.2.5/bin/run.sh
```

Warning:
The change to use the file Datasource (or an alternative shared Datasource) for JMS should also be done to avoid rendering the server useless (connection pool/transaction problem when hibernate and JMS are using the same Datasource).

6.6.4. Load Balancing With Reactor

FIXME (To Be Completed):
This topic will completed and updated shortly.

7. Reactor 6.0 Embedded

7.1. Deploying Reactor 6.0 - Embedded

7.1.1. Overview

FIXME (To Be Completed):

This topic will completed and updated shortly.

8. Reactor 6.0 Administration

This tab covers topics that are important to those charged with the responsibility to administer Reactor.

Starting And Stopping Reactor
Instructions for starting and stopping the Reactor Engine.

Reactor Configuration Utility
How to configure various Reactor settings for your particular system.

Reactor Logs
Where to look for information on the operation and performance of Reactor.

9. Reactor 6.0 Administration Guide

9.1. Starting And Stopping The Reactor Engine

9.1.1. Start Reactor

The Reactor Engine is the core piece of the Reactor Product Suite. The Engine controls the execution of the workflow and interfaces with a database to maintain the Process data.

To start the Reactor Engine, execute the batch file `run.bat` (Windows) or `run.sh` (UNIX) in the `C:\ReactorEnterprise6.0-JBoss3.2.5\` directory. A command window will open and Reactor will begin a startup routine lasting anywhere from 30 seconds to 4 minutes. During this time, Reactor is writing to the log all of the actions it is taking to start the Engine.

Note:

The exact time for Reactor to complete the startup routine depends on several factors, with the two most important being the

current load on the machine housing the Reactor software and the activities of any virus checking software on the machine. Oak Grove Systems does not recommend diluting the effectiveness of virus protection software, but setting the virus protection software to ignore the Reactor directories will speed up the startup routine dramatically.

When the Reactor Engine has completed the startup routine, the command window will look something like the picture below.

Note:

The terminal window will continue to periodically add lines as the Engine begins to execute instructions. The command window can be minimized, but must remain open for Reactor to work properly.

If Reactor does not start properly, the first step is to review the Boot Log. The Boot Log keeps track of all the actions performed by Reactor during the startup phase. Any errors encountered by Reactor will be written to the log.

9.1.2. Stop Reactor

The Reactor Engine can be stopped by simply closing the command window.

9.2. Reactor Configuration

In some cases, you may need to modify Reactor's configuration files. Several of the Reactor Engine internal services have properties files with configuration options. Some optional capabilities also have separate configuration files. These files are stored in the `ReactorConfig.jar`, `ReactorPortalConfig.jar` and `ReactorAuthentication.jar` files, which are stored inside the `Reactor-6.0.ear` file.

The Reactor bundle includes a tool for editing the configuration files without manually extracting the contents of the EAR and JAR files. To use this tool, execute the `Configure.bat` file on Windows or the `Configure.sh` script on UNIX, located in the `C:\%REACTOR_HOME%\` directory. This will open the configuration property files in the `Reactor-6.0.ear` file, which is located in the `C:\%REACTOR_HOME%\server\default\deploy\` directory. If the EAR file has been moved, the explicit path/filename must be specified. The filename can be specified as an argument on the command line, or selected through a menu option once the tool is running.

There are several properties files within the `Reactor-6.0.ear` file that can be custom configured to suit a wide range of deployment requirements. These properties files are described in the following table.

File Name	Property Contents

AdminService.properties	Contains the license key for Reactor. Do **NOT** modify this value or your installation of Reactor may not work.
AuditTrail.properties	Sets which kinds of Events and Operands are tracked by the audit log.
AuthenticationService.properties	Tells Reactor whether to use the evaluation CSV File or to use a specified LDAP directory file for User authentication purposes.
DirectoryService.properties	Tells Reactor whether to use the evaluation CSV File or to use a specified LDAP directory file for directory services purposes.
EvalDirectoryModule.properties	Specifies the files holding the Users, Titles and Groups data for the Reactor evaluation CSV File.
EvalGroups.properties	The Reactor evaluation CSV File that assigns Users to Groups.
EvalTitles.properties	The Reactor evaluation CSV File that assigns Users to Titles.
EvalUsers.csv	The Reactor evaluation CSV File that holds the following User data: • Username • Password • First Name • Last Name • Email Address
LdapDirectoryModule.properties	Provides information to Reactor regarding the LDAP schema.
PolicyExecutionService.properties	Defines JMS topic where Events are published that trigger Policy execution.
Portal.properties	Sets various Process Manager display properties. For detailed information on the setting in this file, see the section on the `Portal.properties` file in Customizing Process Manager In order to send notifications via email, the email host must be properly configured in this section.
PortalVersion.properties	Contains the version number and build number of the Portal client that is currently being used.

ProcessCommandService.properties	Contains the jndi name of Hibernate's SessionFactory.
ProcessObjectService.properties	Used to configure connections for databases. Bundled HSQL product is already configured.
ReactorServices.properties	Used to set whether Reactor will be run in EJB mode or Non-EJB mode.
ReactorVersion.properties	Contains the version number and build number for the release of Reactor that is currently being used.
TimerService.properties	Defines the scheduling module used with Reactor.
com/oakgrovesystems/reactor/services/authentication/AuthToken.hbm.xml	Sets the hibernate mapping DTD.
ehcache.xml	Sets cache management configuration properties.
frontdesk.properties	Contains the URL for the Reactor Engine Front Desk. This URL is used by Portal to establish communication with the Server.
hibernate.cfg.xml	
log4j.properties	Defines the level and format of Reactor log messages.
login.config	Defines the various JAAS login configuration modules available.
quartz.properties	Defines the various properties required by the Quartz Scheduler. For detailed information on the setting in this file, see the section on the `Quartz.properties` file under the Scheduler topic.

9.3. Reactor Logs

There are two log files in Reactor to assist in the administration of the Reactor Product Suite. They are the Boot Log and the Server Log. All actions performed by Reactor are written to these two log files, located in the `C:\ReactorEnterprise6.0-JBoss3.2.5\jboss-3.2.5\server\default\log` directory.

9.3.1. Boot Log

The Reactor Boot Log contains a written record of actions performed by the Reactor software during the startup routine. The focus of this log is the startup of the application server and the start of the Reactor Engine. Once the Engine is started, all actions are written to the Server Log.

> **Note:**
> This log file is overwritten every time the Reactor Engine is started.

9.3.2. Server Log

The Server Log contains a written record of all the actions performed by the Reactor Engine once it has been started. In this file will be information on the execution of the various workflow Processes being run and all connections via the Process Manager application.

> **Note:**
> This log file is overwritten every time the Reactor Engine is started.

9.3.3. Result Codes

On every interaction between the Engine and Process Manager, a result code is generated. These result codes can be helpful in tracking down errors. For a complete list of Reactor generated result codes, see the Result Codes table located in the Developer Guide.

10. Reactor 6.0 Process Design

This tab is aimed at assisting designers of Reactor workflow Processes who are using Studio. It covers the basics of how to build a workflow as well as providing detailed information about the configuration of the standard Process Components included with Reactor. For more detailed programming information, see the Programming tab.

Using Studio
This section covers the basic use of the Studio Process design tool to create and upload Process Definitions.

Defining Process Flow
This section outlines some of the basic concepts behind the design of workflows and explains how to represent those concepts in a Reactor Process. It also contains visual examples of how these concepts are generated in Studio.

Adding Data: Using Operands
This section explains how to capture and manipulate data in Reactor. Operands are used by Reactor to refer to a piece of data and are one of the key Process Components used by Reactor to execute workflow Processes. There is also an excellent reference section that explains how to configure each type of Operand.

Taking Action: Using Policies
This section covers the topic of how to get Reactor to perform an Action. Policies tie an Event to a specific Action. They describe what is going to happen within a particular Activity and under what circumstances it will happen. Probably one of the most useful parts of the entire site is the reference section that explains how to configure each of the standard Policy Actions included in Reactor.

Involving Users: Actors and Forms
This section covers Actors and Activity Forms. Actors are the participants in the Process. Activity Forms are used to define the User interface displayed when an Activity is started or completed.

11. Using Studio

11.1. Designing A Workflow Process

The process designer can program Process Definitions using XML or can use Reactor Studio to create the XML Process Definitions. These Process Definitions can be exported to other applications and can be reused by Reactor either as a whole Process or implemented as a component within a different Process.

The Studio tool saves Process Definitions as XML files. These XML files are human readable and can be manipulated and customized as desired by the Process designer.

11.1.1. Starting Studio

For Microsoft Windows operating systems either one of the two following methods will start the Studio program.

- From an MS-DOS window command prompt change the directory to the `C:\Reactor6.0-JBoss3.2.5\` directory using the cd command and run the `Studio.bat` script.
- From Windows Explorer double-click on the Studio batch file icon in the `C:\Reactor6.0-JBoss3.2.5\` directory.

Note:
The script must be run from the directory where it is located, so it may be necessary to run the script from the command

prompt on some Windows operating systems.

For UNIX operating systems use the following method to start the Studio program.

- From a shell prompt change the directory to the `C:\Reactor6.0-JBoss3.2.5\` directory using the cd command and run the `Studio.sh` script.

When Studio has started, a blank Process Diagram titled Untitled1 will appear as shown in the following screen shot.

The Studio screen is divided into two (see above picture) or three sections (see next picture) depending on the menu options selected. Assuming all three sections are visible, the left section is called the Process Tree, the middle section is called the Process Diagram and the right section is called the Formatting Pane.

The Process Tree is a hierarchical representation of the Process being created which shows all of the Process Components within the Process. The Process Diagram is a graphical representation of the Process being created which shows the workflow steps and any special notes added by the designer. The Formatting Pane is a panel used strictly for formatting objects on the Process Diagram. The only section that must be displayed is the Process Diagram. The Process Tree and Formatting Pane can be hidden if desired using the *View* command on the menu bar.

11.1.2. Studio Toolbars

There are two toolbars that can be displayed at the top of the Studio screen underneath the menu bar. The main Studio toolbar is always visible just underneath the menu bar. The formatting toolbar can be toggled on/off via the menu or by pressing *F10*. The function of each icon outlined in the tables below. The icons are grouped and presented here according to their purpose.

- File Handling
- Engine Upload/Download
- Process Editing
- Other
- Formatting
- More Formatting

File Handling Icons

Toolbar File Commands

Toolbar Icon	Function

	Creates a new workflow Process and opens it a new Process window.
	Opens an existing workflow Process in a new Process window.
	Saves the file on the Process Diagram that currently has the focus.
	Closes the file on the Process Diagram that currently has the focus.
	Prints the workflow Process that currently has the focus.

back to top

Engine Upload/Download Icons

Toolbar Upload/Download Commands

Toolbar Icon	Function
	Opens a window showing all of the workflow Processes that have been uploaded to the Reactor Engine. • Designer must be able to login to the Engine in order to view the Processes.
	Uploads the Process that currently has the focus to the Reactor Engine. The designer has the option of giving the uploaded Process a different name than the identifier used by Studio. • Designer must be able to login to the Engine in order to view the Processes.
	Opens a window showing the Processes that are currently loaded into the Reactor Engine. The designer can then select one of the Processes to download into Studio for editing. **Warning:** Any Instances of the Process that are already started will not be affected by edits to a downloaded Process. Once the Process has been uploaded to the Engine again, all subsequent Instances will reflect the edits.

back to top

Process Editing Icons

Toolbar Editing Commands

Toolbar Icon	Function
	Allows the designer to select an object for editing.
	Allows the designer to quickly grab an object and move it.
	Adds a new Activity box to the Process Diagram.
	Turns on the anchor points of screen objects so that the designer can add Connecting Lines between Process Components.
	Adds a Logic Node to the Process Diagram. Logic options are *AND*, *OR* and *NOT*. **Note:** The default is to always add an *AND* Logic Node.
	Allows the designer to draw a box around all Activities to be performed by a particular participant.
	Adds a note to the Process Diagram, allowing the designer to add Process documentation right on the flow diagram. Notes have no effect on the execution of a workflow Process.
	Adds a bar that allows the Process to split off in multiple directions simultaneously or combine and progress once all of the paths have been traversed. For more detailed information see the section on Join/Split Bars.
	Adds a configurable decision point to the Process Diagram called a Decision Activity. For more detailed information see the section on

	Decision Activities.
	Adds a timer to an Activity.
	Adds another Stop Node to the Process Diagram. Processes may have multiple stopping points.

back to top

Other Icons

Other Toolbar Commands

Toolbar Icon	Function
	Allows the designer to zoom in or zoom out on the process diagram.
	Opens the Clip Library feature of Reactor. The Clip Library can be used to hold Process patterns that will be used again.
	Makes the formatting toolbar visible underneath the main toolbar. The formatting toolbar allows the designer to customize the look and feel of the Process Flow diagram. • The formatting toolbar can be also be toggled between visible and invisible by using F10. • The formatting here has no effect on what web clients see via the Process Manager interface.
	Opens the Help feature of Reactor.

back to top

Formatting Icons

Toolbar Formatting Commands

Toolbar Icon	Function
	Allows the designer to change the fill color of an

	object such as an Activity box or a Decision Activity.
	Allows the designer to change the outline color of an object.
	Allows the designer to change the color of the selected text.
	Allows designer to change the font style for the selected text. The default font is *Dialog*.
	Allows designer to change the font size for the selected text. The default font size is 11.

back to top

More Formatting Icons

More Toolbar Formatting Commands

Toolbar Icon	Function
	Toggles the selected text between **bold** and normal.
	Toggles the selected text between *italic* and normal.
	Toggles the selected text between underlined and normal.
	Allows the designer to set the width of the selected line(s).
Select...	Allows the designer to choose an alternate image to replace the image that is selected.
Remove	Returns an object to it's default Studio image.
	Toggles the formatting tools between being visible as a toolbar underneath the main Studio toolbar and being visible as a panel on the right side of the Process Diagram.

back to top

11.1.3. Process Naming

The first thing to do when designing a new Process is to change the Process Label. The Process Label can be edited by right clicking on the *Untitled1* Process Component shown on the Process Tree. Once the Edit Process window has been opened, the new Process can be given a Label. The designer also has the option to provide a description of the Process.

11.2. Navigating Around In Studio

11.2.1. Editing

All Process Components can be edited using the Process Tree. Most of them can also be edited right on the Process Diagram. Operands, Policies and Actors cannot be edited via the Process Diagram because they are not depicted on the Process Diagram.

To edit a Process Component via the Process Tree, open the Process Tree until the desired component becomes visible and then right click on it to open the Component Menu. Click on the *Edit* button to open the Edit Window. Upon completing the edit, click on OK to close the window and save the changes.

There are two options available for editing Process Components via the Process Diagram. Every Process Component can be edited by right clicking on the Process Component to open the Process Component Menu. Then click on the *Edit* button to open the Edit Window. For Activities, Notes, Swimlanes and Decisions, the Edit Window can also be opened by simply double clicking the Process Component. Upon completing the edit, click on OK to close the window and save the changes.

11.2.2. Zoom

This is placeholder text.

11.2.3. Pan

This is placeholder text.

11.2.4. Find Figure Feature

This is placeholder text.

11.3. Opening And Saving Workflow Processes

11.3.1. Open A Process

To open an existing Process, expand Studio's *File* menu on the menu bar, click Open, and then select the directory and/or file you wish to open. Studio only understands XML files. Alterntively, you may click on the *Open File* icon on the toolbar to accomplish the same task.

> **Note:**
> Reactor comes with several example Processes. They are located in the `...\Studio\examples` directory.

11.3.2. Save A Process

There are two alternatives for saving a Process. The Save command on the menu bar or in the icon tray will both save the Process to the already existing file without prompting the designer for a file name and location. If no saved version exists, Studio will prompt with a default file name (identical to the Process Label on the tab at the top of the Process Diagram) and location. The file name and location can be modified by the designer as necessary.

The Save As command is only available from the menu bar and is handy for times when saving alternative versions of the same Process. In this case, Studio will prompt with a default file name (again identical to the Process Label on the tab at the top of the Process Diagram) and location. The file name and location should be modified by the designer to avoid writing over existing Process Definition files.

11.4. Uploading And Downloading Processes

11.4.1. Uploading A Process

Processes must be uploaded before they can be executed. The procedure for uploading a Process is very simple. Click on the Upload icon. If you are not already logged into the Reactor Engine, you will be prompted for your login Username and Password.

> **Note:**
> If you are just getting started after downloading and installing Reactor and you are prompted to login, use `admin` for the Username and `admin` for the password.

Once logged in, or if you are already logged in, a window will open displaying the Processes already residing on the Server. The Process Label (as shown on the tab at the top of the drawing) of the currently active Process Diagram will be shown in the upload box. Simply click on OK to upload the Process to the Server. Once a Process is uploaded to the Reactor Engine it is available for execution via Process Manager and will be displayed on the Process

Definitions tab in Process Manager.

A Process can be uploaded using a different name. However, it is important to note that doing this will change the Process Label shown on the tab at the top of the drawing. To upload a Process using a different name, use the *Upload As* command on the menu bar and enter the new name in the box when the upload window pops up.

11.4.2. Downloading A Process

FIXME (MJ Glen):

This section isn't written yet.

11.4.3. Browsing Processes

FIXME (MJ Glen):

This section isn't written yet.

11.5. Documenting A Process

FIXME (MJ Glen):

This section has not been written yet.

11.6. Building Reusable Process Components

Studio comes with a feature called Clip Explorer that allows designers to create reusable Process Components. Clip Explorer is library of Process Components that have been saved for future use. By creating these reusable Process Components, designers can dramatically reduce the time required to implement a new Process.

To add a Process Component to the Clip Explorer library, right click on the component to be saved and select Add To Clip Explorer from the Process Component Menu. Once saved, these Process Components can be copied elsewhere in the same Process or copied to a new and entirely unrelated Process. The screen shot below shows the Clip Explorer window that is available in Studio.

A Clip Library can be shared via the ability to import and export files, however it is not currently possible to read or access other individual Clip Libraries across a network.

Note:

Clip Explorer toggles open and closed with the F11 key.

12. Defining Process Flow

12.1. Reactor Process Design Concepts

This section describes the syntax and semantics of the Reactor Process Flow Design Model.

12.1.1. Process Diagrams

The Process Diagram below shows a typical **Review** Process Defintion. In this example, the User identified in the *Reviewer* Operand is asked to "Review" and either *Approve* or *Reject* an item. (The "item" is defined in the configuration of the Review activity, and is not visible on the Process Diagram.)

If the reviewer rejects the item, then the *Process Invoker* gets an opportunity to revise and resubmit the item or cancel the process. The process is completed immediately if the reviewer approves the item.

back to top

12.1.2. Elements

The following is a summary of the symbols that are used to create Process Diagrams that describe a Process Definition from which Process Instances are created and executed.

Symbol	Name	Description
	Start	The start point of a Process Definition. Activated when the Instance is started. Each Process Diagram has exactly one Start symbol.
	Activity	Activities represent the work to be performed during the execution of an Instance. Activities may be manual, automated, or a combination of both. For more information, see the section on Configuring Activities.

	Transition (Connecting Line)	Transitions show the flow one node to the next. Transitions are called Connecting Lines in Studio.
	Stop	Stop nodes show the exit points of a Process Definition. Each map may have one or more Stop symbols. The Instance stops when this node is reached.
	Subprocess	A nested process. Each Subprocess has its own Process Diagram. Subprocesses may be nested to any level. Use a Qualified Stop to set outcomes of a Subprocess.
	Swimlane	Identifies the Actor who will perform the Activities that are inside the Swimlane.
	Decision	Conditionally follow one or more Transitions based on evaluating one or more expressions.
	Junction	All outgoing Transitions from this node are traversed when, and not before, all incoming Transitions have been traversed.
	AND	The outgoing Transition from this node is traversed when, and not before, all incoming Transitions have been traversed.
	OR	The Transition flowing out from this node is traversed exactly once when the first of the incoming Transitions has been traversed.
	Timer Event	Represents the expiration of a timer; may repeat. The timer is

		initialized when the innermost Process touching the Timer symbol is started. The outgoing transition from a Timer symbol is traversed each time the Timer expires.
	Qualified Stop	Execution stops when this node is reached and the associated Status *Okay* is applied to the Instance. Used to set the outcome of a Subprocess.
	Guarded Transition (Connecting Line)	The Transition is traversed only if the source Activity has the named Status when it completes. These Transitions are also called Connecting Lines in Studio. For more information, see the section on Configuring Statuses.
	Annotation	Used to add documentation to Process Diagrams. Annotations have no effect on Process execution.
	Association	Used to associate Annotations to specific nodes on a Process Diagram. Associations have no effect on process execution.

back to top

12.1.3. Transitions

Activities can have any number of incoming and outgoing Transitions. The possible outcomes of an Activity are determined by the Statuses that are associated with the Transitions that are directly connected to that Activity. The Activity is started when any one incoming Transition is traversed. When the Activity stops, each outgoing Transition is considered for traversal: If the Transition has no associated Status, then that Transition is automatically traversed. If the Transition has an associated Status, then that Transition is traversed only if the associated Status has been applied to its source Activity. Consider the following examples:

	C is started when either A or B stops.
	Both A and B are started when C stops.
	B is started when C stops and, if S1 has been applied to C, then A is also started when C stops.

back to top

12.1.4. Swimlanes

Swimlanes identify the Actors who will perform any Activities that are contained within the Swimlane. Swimlanes may identify individuals or groups by name or by organizational role. The name can be associated directly with the Swimlane or taken from an Operand at runtime.

	User *bob* is designated as the Participant of Activity A. When started, Activity A will appear on *bob's* Work List.
	The User who is identified by the title *CFO* in the Directory Service is designated as the Participant of Activity B. When started, Activity B will appear on the CFO's work list.
	Activity C is assigned to the Accounting group. Each member of the group *Accounting* as defined in the Directory Service is designated as an *Eligible Participant* for Activity C, and will see Activity C on their work list. The first Eligible Participant to "acquire" activity C will become the *Participant* responsible for completing the activity, and every other Eligible Participant will become a *Standy Participant*.
	Activity D is assigned to the User whose name is stored in the Operand *Reviewer*.

back to top

12.1.5. Flow Controls

Decisions and Junctions provide a means to control the flow of Reactor Processes. Decisions enable conditional routing based on the evaluation of one or more rules. Junctions enable

multiple threads to be synchronized at a specific point, and allow one thread to spawn multiple other threads. Logic nodes *AND* and *OR* provide a means to visually construct complex boolean conditions. Each of these is defined in the following table.

	Decide which Activities to start based on the value of the Operand *C*. In this example, Activity A will be started if *C* is less than 10, Activity B will be started if *C* is in the range 10 to 100, and Activity C will be started if *C* is greater than 100. Decisions can start multiple Activities; each Transition whose condition is satisfied will be traversed. The *(Otherwise)* Transition will be traversed only if no other Transition has a condition that is satisfied.
	Activities B and C will both be started when Activity A stops.
	Activity C will not start until both Activities A and B have stopped.
	Activity D will be started only if Activity C stops with status S3 and either Activity A stops with status S1 or Activity B stops with status S2.

12.1.5.1. Decisions

The Decision Process Component is used to progress a Process along one or more paths based on the evaluation of a set of specified logical expressions. Upon reaching the Decision, the Process will evaluate the logic expressions defined within the Decision to determine which path(s) to follow.

The Decision always has at least two Status conditions available, but may have many more. Each logical expression that is defined is associated with a Status. That Status is added to the Decision if the associated logical expression evaluates to true. If none of the logical expressions evaluate to true, then the Decision is given an Otherwise Status and the Process progresses along the path dictated by the Otherwise Status. The Otherwise status requires no logical expression from the designer. It is only applied in a situation where none of the other defined logical expressions evaluate to true. The following diagram shows demonstrates how to edit a Decision, including the use of the Boolean expression builder, as well as how this flow control Process Component might be used in a Process.

The diagram above shows an example Process, the Decision Edit window and the Boolean expression builder. Decisions define branches in the process flow that are based the values of

operands during execution. Each potential outcome of a Decision is defined with an expression. These expressions can have relational, arithmetic, and boolean operations.

Decision Expression Examples

In the following examples, Operands are defined with the names X, Y, Cost, Color, and Time Limit. These are all valid expressions for Decision Activity conditions:

- Cost < 1000
- (X + Y) < 10 and Y > 2
- Color = "Yellow"
- Color != 'Blue'
- !((X < Time Limit) && (Time Limit > Y / 2))

Operators Allowed In Decision Activity Expressions

The following operations are allowed in Decision Activity expressions.

Symbols	Operation
OR, or, ||	Logic: disjunction
AND, and, &&	Logic: conjunction
NOT, not, !	Logic: inversion
=, ==	Comparison: equal to
!=	Comparison: not equal to
<	Comparison: less than
<=	Comparison: less than or equal to
>	Comparison: greater than
>=	Comparison: greater than or equal to
+	Arithmetic: addition
-	Arithmetic: subtraction
*	Arithmetic: multiplication
/	Arithmetic: division
%	Arithmetic: modulo
(,)	Grouping: parentheses

Data Types And Conversions

Values in expressions can be text, integers, floating point numbers, or booleans (TRUE/FALSE). Values are interpreted based on their context in an expression. Values are converted to numbers for arithmetic operations, and converted to booleans for logical comparisons. A text value is interpreted as true in a boolean context only if it is true or yes, in any combination of upper case and lower case. A numeric value is interpreted as true in a boolean context only if it is non-zero. A boolean value is interpreted in a numeric context as 0 for false, 1 for true. In a string context, a boolean value is converted to true or false.

Selecting Operands In A Decision Activity

Operands in Decision Activity expressions must be associated with the Process containing the Decision Activity, or with one of its parent Processes. There is no way to refer to Operands defined in other scopes, such as Operands associated with Activities that are siblings of the Decision Activity. Operand labels with spaces between words are allowed in expressions. Numbers, underscores, and dashes are also acceptable in Operand labels, but the labels must start with a letter. Operand labels in expressions are case-sensitive. Letters, numbers, underscores, dashes and spaces are acceptable in the labels of Operands that appear in Decision Activity expressions.

Warning:
Symbols that are valid in expressions (> or * for example) may not be recognized as being part of Operand labels.

back to top

12.1.6. Timer Events

Timer Events are used to advance processes at specified times or after specified intervals. Timers may be configured to reset and expire again any number of times. The Timer Event symbol is used to visually show the impact time dependencies on process flow. Timers are initialized when the innermost process that is touching the corresponding Timer Event symbol is started. The outgoing transition from a Timer Event symbol is traversed each time the timer expires. If a Timer Event symbol is placed over the border of an Activity, then expiration of that time will also cause the activity to stop.

	Activity A stops 10 seconds after it starts. A timer is set to expire 10 seconds after Activity A starts. When the timer expires, Activity A is stopped (because the Timer Event symbol is located on the

	border of Activity A,) and the Transition to the Stop node is traversed, causing this Instance to be stopped with the Status *Timeout*.
	Activity A spawns Activity B 10 seconds after Activity A starts. A timer is set to expire 10 seconds after Activity A starts. Since the Timer Event symbol is located inside the border of Activity A, Activity A is not stopped when the timer expires. Activity B is started in parallel with Activity A when the timer expires.
	The instance times out 10 seconds after it starts. A timer is set to expire 10 seconds after this instance starts. When the timer expires, the Transition to the Stop node with associated Status *Timeout* is traversed, causing the Instance to be stopped with Status *Timeout*.

back to top

12.1.7. Subprocesses

Reactor Process Diagrams may contain Subprocesses. Subprocesses have all of the capabilities and limitations of top level Processes, except that they can be referenced symbolically on the map of the parent process. Since Subprocesses are used in constructing larger flows, a mechanism is needed to set the "outcome" of a Subprocess. Qualified Stop nodes provide this capability. When a Subprocess is stopped by reaching a particular Qualified Stop node, the Status (if any) associated with that Qualified Stop node is passed back up as the "outcome" of the Subprocess as a whole, providing a visual means to indicate the flow of activity proceeding from a Subprocess.

	This definition contains a Subprocess named *Review*, indicated by the stacked activity figure. Subprocess *Review* encapsulates a complex set of conditional reviews described in more detail below. If Subprocess *Review* stops with Status *Approved*, then Activity *Distribute* is started, otherwise the Instance is stopped with no further action. Note that the qualifiers on the outgoing Transitions

	from Subprocess *Review* correspond to the Statuses associated with the Stop nodes on Subprocess *Review's* map, shown below.
	Subprocess Review This is the Process Diagram for Subprocess *Review* used in the definition above. In this example, Activity *Supervisory Review* is always performed, but Activity *Manager Review* is performed only if required. If all required reviews result in approval, then the Subprocess stops with Status *Approved*. If any required review results in a rejection, then the Subprocess stops with Status *Rejected*. Depending on which Stop node is reached, either status *Approved* or Status *Rejected* is passed back up as the completion Status of this Subprocess. These Statuses can then be used as qualifiers on the outgoing Transitions from this Subprocess symbol on the parent Process Diagram.

back to top

12.1.8. Annotations

Annotations may be added to a Process Diagram to provide more information to help readers and maintainers comprehend the Process. Annotations may be free floating or may be associated with a particular symbol using an Association link. Annotations are for documentation only and have no impact on Process execution.

	Free-Floating Annotations can be added to any Process Diagram.
	Annotations can be linked to any symbol on a Process Diagram. Annotations are for documentation purposes only, they have no effect on Process execution.

back to top

12.2. Activities And Subprocesses

Activity objects are the primary building blocks of a model. They contain information about the current State of the Process, the Status of the various Activities within the Process, the conditions under which Activities should start, stop, and change State, and the relationships the Process has with other objects. Non-trivial business processes consist of multiple steps that can often themselves be broken down into smaller, simpler steps.

Activities can really be divided into two classes, Activities and Subprocesses. Activities are just as described above, the basic unit of work in a business process. Subprocesses are Activities that often don't perform any basic unit of work, rather they are containers for a new group of Activities at a sublevel that are executed before returning to the higher level once more. Subprocesses are created by editing an existing Activity via the Process Component Menu and clicking on Create Subprocess. A Subprocess acts very similar to the Root Process with the primary difference that there are no Start Forms associated with a Subprocess.

When a Subprocess is created in Studio, the Process Diagram representation is shown as a set of stacked Activity boxes. Creating a Subprocess opens an entirely new Process Diagram for depicting the Process flow on this new sub-level. One of the primary advantages of using a Subprocess is organization. A complex Process is often easier to understand, represent graphically and even configure if it is broken down into multiple levels of Activities.

There are really three things that can happen to an Activity. It can be completed, initiating the next step in the Process. It can be Delegated, which assigns the Activity to another party for completion and returns the Activity to the delegator for approval before moving on to the next Process step. And finally, it can be Transferred, which assigns the Activity to another party and then moves to the next Process step upon completion by that party. When an Activity is Transferred, the delegator does not get an opportunity to approve the completed Activity before the Process is advanced. Whether an Activity can be Delegated or Transferred is determined by the process designer when setting up the Work Item Form for each Activity.

The following sections provide detailed information regarding how to configure the general attributes of an Activity and a Subprocess. In each case it shows a screenshot of the Edit window and describes how to complete each entry related to the Activity or Subprocess.

12.2.1. Activity

12.2.1.1. Label

Enter a name that represents this Activity on the Studio Process Tree and in Process Manager

displays.

12.2.1.2. Description

Enter any desired text information about this Activity. This field is not required.

12.2.1.3. Process Manager Forms Box

Edit Work Item..

Select this button to configure a Work Item Form that will be displayed to participants via Process Manager when this Activity is active.

Edit Completion Form..

Select this button to configure a Completion Form that will be displayed to participants via Process Manager when this Activity has been completed.

Use Activity JSP

Select this button to use a custom JSP Form in lieu of a Work Item Form. The JSP Form will be displayed to participants via Process Manager when this Activity is active.

12.2.1.4. Process Manager Notifications Box

Notify participant when activity starts

Checking this box creates a SendNotification Policy which will send an email to all Initiative Participants when the Activity has been started.

Notify owner when activity ends

Checking this box creates a SendNotification Policy which will send an email to the Process Owner that the Activity has ended.

12.2.1.5. Delegation & Transfer Box

Allow Delegation

Checking this box will allow the Initiative Participant to delegate the Activity to another

person. The original Initiative Participant will still have to approve the result before the Process can be progressed.

Allow Transfer

Checking this box will allow the Initiative Participant to transfer complete responsibility for the Activity to another person. The original Initiative Participant will not have to approve the result before the Process can be progressed.

12.2.2. Subprocess

12.2.2.1. Label

Enter a name that represents this Subprocess on the Studio Process Tree and in Process Manager displays.

12.2.2.2. Description

Enter any desired text information about this Subprocess. This field is not required.

12.2.2.3. Process Manager Forms Box

Edit Work Item..

Not Available for a Subprocess.

Edit Completion Form..

Not Available for a Subprocess.

Use Activity JSP

Not Available for a Subprocess.

12.2.2.4. Process Manager Notifications Box

Notify participant when activity starts

Checking this box creates a SendNotification Policy which will send an email to all Initiative Participants when the Subprocess has been started.

Notify owner when activity ends

Checking this box creates a SendNotification Policy which will send an email to the Process Owner that the Subprocess has ended.

12.2.2.5. Delegation & Transfer Box

Allow Delegation

Not Available for a Subprocess.

Allow Transfer

Not Available for a Subprocess.

12.3. Statuses

Status Process Components are very simple objects used to trigger conditions in Activities or cause the execution of Policy Actions, as well as to provide information about how a Process is progressing. Status objects can be added at the Process level, where they can be made available to all Activities within the Process or they can be added at a Subprocess/Activity level where they will only be available to that Activity or any Activities associated with the Subprocess depending on the configuration of the Status.

Statuses do not need to be added to every Process Component. Studio automatically assigns a `(Finished)` Status to Connecting Lines when they are added to the Process Diagram. `(Finished)` is not technically a Status, but a State. So by default, if no Status is set for a Connecting Line, Reactor assumes that the Process will progress along the path labeled `(Finished)` once the Process Component in question achieves a finished State.

Below is a screenshot of an Edit Status window with guidance on how to complete each of the attributes of a Status.

12.3.1. Name

Enter a name that represents this Status on the Studio Process Tree and in Process Manager displays. Statuses within the same hierarchy level must have unique Names.

12.3.2. Description

Enter any desired information about this Status as documentation. *This field is not required*.

12.3.3. Scope Box

If the `Visible in entire subtree` box is not checked, this Status will **NOT** be available for selection when editing the Process Diagram.

13. Adding Data: Using Operands

13.1. Operands

Operand Process Components encapsulate arbitrary data that is relevant to the business process. Examples include the name of the mail host, a logged-in user's name, comments entered in a text box at the Web browser interface, a document URL, or a purchase order number.

Placeholder for Diagram (cka to provide).

13.2. Operand Reference Guide

13.2.1. Introduction

This section provides detailed information regarding the use of the standard Operand Data Types included in Reactor. Each Operand description displays a screenshot of the Edit Operand window and explains how to complete each attribute for proper configuration.

13.2.2. Checkbox

13.2.2.1. General

Item	Configuration
Operand Label	Enter a label that represents this Operand on the Studio Process Tree and in Process Manager displays. Operands within the same hierarchy level must have unique Operand Labels.
Data Type	Select the type of Operand to be configured. This is a **Checkbox** Operand. The *Refresh* button updates the set of Data Types that are available for designer selection.

	Warning: Clicking the *Refresh* button also resets the *Default Box* to a checked (TRUE) state.
Description	Enter any desired information about this field as documentation. *This field is not required*.

13.2.2.2. Properties

Item	Configuration
Default Box	If the `Set default value to TRUE` box is checked, this Operand will start with a value of TRUE or 1.

13.2.2.3. Visibility

Item	Configuration
Create In	Shows the available Processes/Activities under which the Operand can be created.
Scope Box	If the `Make visible in entire subtree` box is not checked, this Operand will **NOT** be available to other Activities in the Process.

back to top

13.2.3. Currency

13.2.3.1. General

Item	Configuration
Operand Label	Enter a label that represents this Operand on the Studio Process Tree and in Process Manager displays. Operands within the same hierarchy level must have unique Operand Labels.
Data Type	Select the type of Operand to be configured. This is a **Currency** Operand.

	The *Refresh* button updates the set of Data Types that are available for designer selection. **Warning:** Clicking the *Refresh* button also resets the *Amount($)* property to the default value of 0.00.
Description	Enter any desired information about this field as documentation. *This field is not required*.

13.2.3.2. Properties

Item	Configuration
Amount($)	Enter the currency amount. Entries in this box ARE validated to ensure they match a currency format.

13.2.3.3. Visibility

Item	Configuration
Create In	Shows the available Processes/Activities under which the Operand can be created.
Scope Box	If the `Make visible in entire subtree` box is not checked, this Operand will **NOT** be available to other Activities in the Process.

back to top

13.2.4. DatabaseObject

13.2.4.1. General

Item	Configuration
Operand Label	Enter a label that represents this Operand on the Studio Process Tree and in Process Manager displays. Operands within the same hierarchy level must have unique Operand Labels.

Data Type	Select the type of Operand to be configured. This is a **DatabaseObject** Operand. It is used to interact with a database and can either retrieve information from the database or write information to the database. The *Refresh* button updates the set of Data Types that are available for designer selection. **Warning:** Clicking the *Refresh* button also clears all of the values in the *DB Object Parameters* section.
Description	Enter any desired information about this field as documentation. *This field is not required*.

13.2.4.2. Properties

Item	Configuration
JDBC Driver	Enter the full path location and name of the JDBC driver that should be used. Example: *com.mysql.jdbc.Driver*
Username	Enter the Username required (if any) to gain access to the database.
Password	Enter the password required (if any) to gain access to the database.
Retype Password	Enter the password required again (if any) to gain access to the database.
Database URL	Enter the full path location and name of the database to be accessed. Example: *jdbc:mysql://www.yoursite.com:3306/DatabaseName*
SQL Statement Type	Choose the type of SQL statement to execute. • Choose SELECT to retrieve information from a database. • Choose GET to save information to a database.
SQL Statement	Enter the SQL string to retrieve/save the desired

	value from/to the database. If you want to use an Operand value as one of the query parameters in your SQL statement, use the following format: *${ProcessName:ActivityName:OperandName}*

13.2.4.3. Visibility

Item	Configuration
Create In	Shows the available Processes/Activities under which the Operand can be created.
Scope Box	If the `Make visible in entire subtree` box is not checked, this Operand will **NOT** be available to other Activities in the Process.

back to top

13.2.5. Date

13.2.5.1. General

Item	Configuration
Operand Label	Enter a label that represents this Operand on the Studio Process Tree and in Process Manager displays. Operands within the same hierarchy level must have unique Operand Labels.
Data Type	Select the type of Operand to be configured. This is a **Date** Operand. The *Refresh* button updates the set of Data Types that are available for designer selection. **Warning:** Clicking the *Refresh* button also clears the *Value* property.
Description	Enter any desired information about this field as

	documentation. *This field is not required*.

13.2.5.2. Properties

Item	Configuration
Value	Enter the date. Entries in this box are NOT validated.

13.2.5.3. Visibility

Item	Configuration
Create In	Shows the available Processes/Activities under which the Operand can be created.
Scope Box	If the `Make visible in entire subtree` box is not checked, this Operand will **NOT** be available to other Activities in the Process.

back to top

13.2.6. Email

13.2.6.1. General

Item	Configuration
Operand Label	Enter a label that represents this Operand on the Studio Process Tree and in Process Manager displays. Operands within the same hierarchy level must have unique Operand Labels.
Data Type	Select the type of Operand to be configured. This is a **Email** Operand. The *Refresh* button updates the set of Data Types that are available for designer selection. **Warning:** Clicking the *Refresh* button also resets the *Address* property to the default of `user@domain`.

Description	Enter any desired information about this field as documentation. *This field is not required*.

13.2.6.2. Properties

Item	Configuration
Address	Enter the Email address. Entries in this box are NOT validated.

13.2.6.3. Visibility

Item	Configuration
Create In	Shows the available Processes/Activities under which the Operand can be created.
Scope Box	If the `Make visible in entire subtree` box is not checked, this Operand will **NOT** be available to other Activities in the Process.

back to top

13.2.7. File

13.2.7.1. General

Item	Configuration
Operand Label	Enter a label that represents this Operand on the Studio Process Tree and in Process Manager displays. Operands within the same hierarchy level must have unique Operand Labels.
Data Type	Select the type of Operand to be configured. This is a **File** Operand. It is designed for use in situations where a participant will attach a file to an Activity Form. To attach a file to an Activity Form at design time, use the URL Operand. The *Refresh* button updates the set of Data Types that are available for designer selection.

Description	Enter any desired information about this field as documentation. *This field is not required*.

13.2.7.2. Properties

Item	Configuration
--	There are no configurable properties for this Operand Type.

13.2.7.3. Visibility

Item	Configuration
Create In	Shows the available Processes/Activities under which the Operand can be created.
Scope Box	If the `Make visible in entire subtree` box is not checked, this Operand will **NOT** be available to other Activities in the Process.

back to top

13.2.8. LongText

13.2.8.1. General

Item	Configuration
Operand Label	Enter a label that represents this Operand on the Studio Process Tree and in Process Manager displays. Operands within the same hierarchy level must have unique Operand Labels.
Data Type	Select the type of Operand to be configured. This is a **LongText** Operand. This is a text Operand that is displayed as a multi-line input field on Activity Forms. The *Refresh* button updates the set of Data Types that are available for designer selection. **Warning:**

	Clicking the *Refresh* button also clears the *Value* property.
Description	Enter any desired information about this field as documentation. *This field is not required*.

13.2.8.2. Properties

Item	Configuration
Value	Enter the text string. Used for multiple line entry fields.

13.2.8.3. Visibility

Item	Configuration
Create In	Shows the available Processes/Activities under which the Operand can be created.
Scope Box	If the `Make visible in entire subtree` box is not checked, this Operand will **NOT** be available to other Activities in the Process.

back to top

13.2.9. Number

13.2.9.1. General

Item	Configuration
Operand Label	Enter a label that represents this Operand on the Studio Process Tree and in Process Manager displays. Operands within the same hierarchy level must have unique Operand Labels.
Data Type	Select the type of Operand to be configured. This is a **Number** Operand. The *Refresh* button updates the set of Data Types that are available for designer selection.

	Warning: Clicking the *Refresh* button also clears the *Value* property.
Description	Enter any desired information about this field as documentation. *This field is not required*.

13.2.9.2. Properties

Item	Configuration
Value	Enter the numerical value. Entries in this box ARE validated to ensure they match a number format.

13.2.9.3. Visibility

Item	Configuration
Create In	Shows the available Processes/Activities under which the Operand can be created.
Scope Box	If the `Make visible in entire subtree` box is not checked, this Operand will **NOT** be available to other Activities in the Process.

back to top

13.2.10. Organizational Role

13.2.10.1. General

Item	Configuration
Operand Label	Enter a label that represents this Operand on the Studio Process Tree and in Process Manager displays. Operands within the same hierarchy level must have unique Operand Labels.
Data Type	Select the type of Operand to be configured. This is an **Organizational Role** Operand. It is designed to be

	used with the Organizational Roles specified in the evaluation CSV file or in an LDAP. The *Refresh* button updates the set of Data Types that are available for designer selection. **Warning:** Clicking the *Refresh* button also clears the *Value* property.
Description	Enter any desired information about this field as documentation. *This field is not required*.

13.2.10.2. Properties

Item	Configuration
Value	Enter the Organizational Role in text format. Reactor does **NOT** currently validate these entries against the CSV file or an LDAP. Organizational Roles **ARE** case-sensitive.

13.2.10.3. Visibility

Item	Configuration
Create In	Shows the available Processes/Activities under which the Operand can be created.
Scope Box	If the `Make visible in entire subtree` box is not checked, this Operand will **NOT** be available to other Activities in the Process.

back to top

13.2.11. Picklist

13.2.11.1. General

Item	Configuration
Operand Label	Enter a label that represents this Operand on the Studio Process Tree and in Process Manager displays. Operands within the same

	hierarchy level must have unique Operand Labels.
Data Type	Select the type of Operand to be configured. This is a **Picklist** Operand. It is used to provide a drop down box on an Activity Form that offers multiple selections for the participant. The *Refresh* button updates the set of Data Types that are available for designer selection. **Warning:** Clicking the *Refresh* button also clears the *Default Value* property.
Description	Enter any desired information about this field as documentation. *This field is not required*.

13.2.11.2. Properties

Item	Configuration
Default Value	Enter the default value for the list. This will be the the value that is visible in the drop down box on the Activity Form.

13.2.11.3. List Values

Item	Configuration
List Box	Displays the list of values that will be made available in the Activity Form drop down box. The list is shown in the order that the values will be presented. The three button s to right of this box can be used to adjust the order of presentation and to remove values from the list. • *Delete* - Removes the highlighted value from the list. • *Move Up* - Moves a value one position higher in the list. • *Move Down* - Moves a value one position lower in the list.
New List Item	Click on the *Add* button to add additional values to the list box on the left.

13.2.11.4. Visibility

Item	Configuration
Create In	Shows the available Processes/Activities under which the Operand can be created.
Scope Box	If the `Make visible in entire subtree` box is not checked, this Operand will **NOT** be available to other Activities in the Process.

back to top

13.2.12. ShortText

13.2.12.1. General

Item	Configuration
Operand Label	Enter a label that represents this Operand on the Studio Process Tree and in Process Manager displays. Operands within the same hierarchy level must have unique Operand Labels.
Data Type	Select the type of Operand to be configured. This is a **ShortText** Operand. This is a text Operand that is displayed as a single-line input field on Activity Forms. The *Refresh* button updates the set of Data Types that are available for designer selection. **Warning:** Clicking the *Refresh* button also clears the *Value* property.
Description	Enter any desired information about this field as documentation. *This field is not required*.

13.2.12.2. Properties

Item	Configuration

Value	Enter the text string. Used for single-line input fields.

13.2.12.3. Visibility

Item	Configuration
Create In	Shows the available Processes/Activities under which the Operand can be created.
Scope Box	If the `Make visible in entire subtree` box is not checked, this Operand will **NOT** be available to other Activities in the Process.

back to top

13.2.13. URL

13.2.13.1. General

Item	Configuration
Operand Label	Enter a label that represents this Operand on the Studio Process Tree and in Process Manager displays. Operands within the same hierarchy level must have unique Operand Labels.
Data Type	Select the type of Operand to be configured. This is a **URL** Operand. This Operand creates a hyperlink on an Activity Form. When the participant clicks on the link, they are taken to that site or the file is opened. Use this Operand type to attach a file to an Activity Form at design time. The *Refresh* button updates the set of Data Types that are available for designer selection. **Warning:** Clicking the *Refresh* button also resets the URL property to *http://*.
Description	Enter any desired information about this field as

	documentation. *This field is not required*.

13.2.13.2. Properties

Item	Configuration
URL	Enter the complete URL or the complete path location and filename. Examples: *http://www.oakgrovesystems.com* *C:\My Documents\Adobe\document1.pdf*

13.2.13.3. Visibility

Item	Configuration
Create In	Shows the available Processes/Activities under which the Operand can be created.
Scope Box	If the `Make visible in entire subtree` box is not checked, this Operand will **NOT** be available to other Activities in the Process.

back to top

13.2.14. User

13.2.14.1. General

Item	Configuration
Operand Label	Enter a label that represents this Operand on the Studio Process Tree and in Process Manager displays. Operands within the same hierarchy level must have unique Operand Labels.
Data Type	Select the type of Operand to be configured. This is a **User** Operand. It is designed to be used with the Usernames specified in the evaluation CSV file or in an LDAP. The *Refresh* button updates the set of Data Types that

	are available for designer selection. **Warning:** Clicking the *Refresh* button also clears the *Value* property.
Description	Enter any desired information about this field as documentation. *This field is not required*.

13.2.14.2. Properties

Item	Configuration
Value	Enter the Username in text format. Reactor does **NOT** currently validate these entries against the CSV file or an LDAP. Usernames **ARE** case-sensitive.

13.2.14.3. Visibility

Item	Configuration
Create In	Shows the available Processes/Activities under which the Operand can be created.
Scope Box	If the `Make visible in entire subtree` box is not checked, this Operand will **NOT** be available to other Activities in the Process.

back to top

13.3. Creating Custom Operands

Designers can also create a custom Operand type (called a Parameterized Operand) and add it to the list of Operand types Studio can utilize. Operand types are defined in a single XML file.

Note:
Only process objects can be displayed in a property editor dialog box.

To create custom Operand types, open and edit `OperandTypes.xml` found in the `C:\Reactor6.0-JBoss3.2.5\studio\conf\` directory.

Below is the complete XML format for describing a new Operand type for accessing a database.

```
<operand_type name="DB Access Object">
  <value_control hidden="true">
    <chooser class=""/>
    <value></value>
  </value_control>
  <parameter_groups>
    <group hidden="false" title="DB Security Object Parameters">
      <parameter name="Login ID"
          title="Login ID"
          hidden="true">
        <chooser class="textfield"/>
        <default_value>admin</default_value>
      </parameter>
      <parameter name="Password"
          title="Password"
          hidden="true">
        <chooser class="textfield"/>
        <default_value></default_value>
      </parameter>
    </group>
  </parameter_groups>
</operand_type>
```

First, the Operand has several high-level attributes and elements that must be specified.

- The *operand_type name* attribute is where the Operand type name is specified.
- The *value_control hidden* element specifies whether the GUI for taking the Operand value is hidden or not. Setting this to false will show an input control in the Edit Policy dialogue box.
- The *chooser* element specifies what GUI control will be used for taking in the Operand value. The value within this control is ignored if the value_control hidden element is set to true.
- The *value* element specifies the initial value of the Operand which will be displayed by the GUI control.

Next, the designer defines the Parameters that will be part of the Operand. Operand parameters have several attributes that can be specified by the designer.

- The *parameter name* attribute specifies the name of the parameter. This will be the id of this parameter.
- The *title* attribute specifies the text which will appear as label beside the GUI input control for this parameter.
- The *hidden* attribute tells whether the input control for this parameter will be shown on the Edit Operand dialogue box.

- The *chooser* element specifies what kind of GUI control will be shown as this parameter's input control. This value is ignored when the value of the hidden attribute is set to true.
- The *default_value* element specifies the initial value of this parameter.

Note:
It is possible for an Operand to have multiple parameters or no parameters at all.

14. Taking Action: Using Policies

14.1. Policies

Policy Process Components encapsulate arbitrary logic that is relevant to the Process being modeled. They allow Processes to be custom scripted and extremely flexible. Policies associate an Action with an Event that triggers its execution. Policies accomplish these Actions through the use of a Java class or a BSF script.

There are two elements to a Policy. The first thing to determine is what sort of Event will trigger the Policy. The second element to configure is the Action that the Policy will execute. The Process designer can use any of the existing standard Policy Actions within Reactor or can create their own Parameterized Policy Action if desired.

Note:
Policies within an Activity are executed in alphabetical order.

14.1.1. Policy Events

There are eight ways in which an Event can trigger an Action within a Policy Process Component, using seven different types of Events. Because the Event types exist in the Reactor source code, the designer may only use an Event type from the drop down list in the Edit Policy window. There is no provision for creating new Event types like there is for creating new Operand types or new Policy Actions.

Policy Event	Description
ACLUpdated	Policy Action is triggered when any Actor in the Actor List associated with the current Activity is updated.
AllEvents	Policy Action is triggered when any of the other seven Events occurs.

OperandUpdated	Policy Action is triggered when the selected Operand is updated.
ProcessStarted	Policy Action is triggered when the selected Activity is started.
ProcessFinished	Policy Action is triggered when the selected Activity has reached the finished State.
StatusAddition	Policy Action is triggered when the selected Status has been added to an Activity.
StatusRemoval	Policy Action is triggered when the selected Status has been removed from an Activity.
TimerExpired	Policy Action is triggered when a timer with a designer-configured schedule has expired. For more information on the TimerExpired Event, see the Using Timers section.

14.1.2. Policy Actions

Following is a brief overview of the standard Policy Actions included with Reactor. These policies are ready to use and available to the designer in Studio. The XML code for these policies is located in the `C:\Reactor6.0-JBoss3.2.5\studio\conf\actions\` directory.

The Policy Action Reference provides detailed information regarding each of the standard Policy Actions included in Reactor. For each Policy Action it shows a screenshot of the Policy configuration window and describes how to complete each entry for proper configuration.

Policy Action	Description
AddStatusToProcess	Use this Policy Action to add a Status to some Activity within the currently active Process.
AddStatusToProcessWithID	Use this Policy Action to add a Status to some Activity within another Process. The destination Process for this Action can be either another Reactor Process or a Process that is entirely external to Reactor. The ID for the destination Process is retrieved from an Operand.
CopyACLtoProcess	Use this Policy Action to copy an Actor List to a Process/Activity within the same Process.
CopyLocalOperandWithID	Use this Policy Action to copy an existing

	Operand within a Reactor Process to another Process that is external to Reactor.
CopyOperand	Use this Policy Action to copy an existing Operand to some Activity within the currently active Process.
CopyRemoteOperandWithID	Use this Policy Action to copy a value from another Process that is external to Reactor into an Operand within the currently active Reactor Process.
CreateApprovalInstances	Use this Policy Action to create an approval step in the Process. This Action generates a form in Process Manager and routes it to the approving authority. The "approver" can have multiple options depending on the design of the process, including the option to Transfer and Delegate. Transfer routes the approval instance to another User. The Process will resume according to the selection made by that User without reverting back to the original participant. Delegate routes the approval instance to another User. After that User has made a selection, the approval instance is sent back to the original approver. The Process will resume according to the selection made at this point by the original approver.
Custom	Use this Policy Action when implementing a custom Action designed in source code. Reactor can currently work with custom Policy Actions written in java, jacl, javascript, jpython and netrexx.
DBObjectPolicy	Use this Policy Action when retrieving or updating a value in an external database. Requires the use of the Database Object Operand to provide information required to access and query the database.
ExecuteShellCommand	Use this Policy Action to execute a command line function in a UNIX environment.
IncrementOperand	Use this Policy Action to increment an Operand value by a selected amount.
RemoveStatusFromProcess	Use this Policy Action to remove a Status from some Activity within the currently active Process.

SendNotification	Use this Policy Action to send a notification message to users when the data is to be "hard coded" into the Policy itself. This Policy works best when the user being notified is always a particular user.
SendNotificationByOperand	Use this Policy Action to send a notification message according to some Operand.
SendNotificationByRole	Use this Policy Action to send a notification message to Users according to their defined Role.
SetACLofProcess	Use this Policy Action to add an Actor for a Process/Activity and set their role.
SetMultipleProcessACEs	Use this Policy Action to add multiple Actors for a Process/Activity and set their roles. All the added Actors must have the same role.
SetPicklistItems	Use this Policy Action to dynamically set the items for a Picklist Operand.
StartProcess	Use this Policy Action when the objective is to have an event in one Process start an entirely different Process.
StartProcessWithLabel	Use this Policy Action to start another Process. The Process for this Action can be either another Reactor Process or a Process that is entirely external to Reactor.
StopProcess	Use this Policy Action when the objective is to have an event in one Process stop an entirely different Process.
StopProcessWithID	Use this Policy Action to stop another Process. The Process for this Action can be either another Reactor Process or a Process that is entirely external to Reactor.

14.2. Policy Action Reference Guide

14.2.1. Introduction

This section provides detailed information regarding the use of the standard Policy Actions included in Reactor. Each Policy Action description displays a screenshot of the Edit Policy

window and explains how to complete each attribute for proper configuration.

14.2.2. AddStatusToProcess

14.2.2.1. Event/Action

Policy Item	Configuration
Event	Select the Event that will trigger this Policy Action from the drop down box. For detailed information see the Events section of the Policy Overview page.
Action	Select the Action that will execute when this Policy is triggered by the Event. When the Event is triggered, **AddStatusToProcess** will add the Status selected below in the Parameters Box to the Process/Activity selected below in the Parameters box. The *Refresh* button updates the set of Policy Actions that are available for designer selection. **Warning:** Clicking the *Refresh* button also erases any values in the Parameters box.
Description	Enter any desired information about this Policy as documentation. *This field is not required*.

14.2.2.2. Parameters

Policy Item	Configuration
Process	Select or enter the Process that will have a Status added to it. The button to the right opens an object picker window listing available Processes/Activities for configuring this portion of the Parameters box.
Status	Select or enter the Status that will be added to the selected Process/Activity below. The button to the right opens an object picker window listing available Statuses for configuring this portion of the Parameters box.

14.2.2.3. Label

Policy Item	Configuration
Policy Label	Enter a name that represents this Policy on the Studio Process Tree and in Process Manager displays. Policies within the same hierarchy level must have unique Labels. The default Policy Label will always be a combination of the Event And the Action.

back to top

14.2.3. AddStatusToProcessWithID

14.2.3.1. Event/Action

Policy Item	Configuration
Event	Select the Event that will trigger this Policy Action from the drop down box. For detailed information see the Events section of the Policy Overview page.
Action	Select the Action that will execute when this Policy is triggered by the Event. When the Event is triggered, **AddStatusToProcessWithID** will add the Status selected below to an external Process that has its ID stored in an Operand. The *Refresh* button updates the set of Policy Actions that are available for designer selection. **Warning:** Clicking the *Refresh* button also erases any values in the Parameters box.
Description	Enter any desired information about this Policy as documentation. *This field is not required*.

14.2.3.2. Parameters

Policy Item	Configuration

Process ID	Select the Operand containing the Process ID that will have the Status selected below added to it. The button on the right opens an object picker window to select the Operand containing the desired information. The ID is the long alphanumeric GUID assigned by Reactor, not the Process Label.
Status Label	Select or enter the Status Label that will be added to the external Process with the ID above. The button to the right opens an editor window so that the parameter may be loaded from a file if desired.

14.2.3.3. Label

Policy Item	Configuration
Policy Label	Enter a name that represents this Policy on the Studio Process Tree and in Process Manager displays. Policies within the same hierarchy level must have unique Labels. The default Policy Label will always be a combination of the Event And the Action.

back to top

14.2.4. CopyACLtoProcess

14.2.4.1. Event/Action

Policy Item	Configuration
Event	Select the Event that will trigger this Policy Action from the drop down box. For detailed information see the Events section of the Policy Overview page.
Action	Select the Action that will execute when this Policy is triggered by the Event. When the Event is triggered, **CopyACLtoProcess** will copy an Actor list from one Process/Activity to another. The designer has the option to select a different Role for the copied Actor

	list than the original Actor list had. The *Refresh* button updates the set of Policy Actions that are available for designer selection. **Warning:** Clicking the *Refresh* button also erases any values in the Parameters box.
Description	Enter any desired information about this Policy as documentation. *This field is not required*.

14.2.4.2. Parameters

Policy Item	Configuration
Source Process	Select the Process/Activity that will be the source of the copied Actor list. The button to the right opens an object picker window listing available Processes/Activities for configuring the source of the copied Actor List.
Source Role	Select the Role for the Actor List used by the source Process/Activity from the list.
Destination Process	Select the Process/Activity that will be the destination for the copied Actor list. The button to the right opens an object picker window listing available Processes/Activities for configuring the destination of the copied Actor List.
Destination Role	Select the Role for the Actor List used by the destination Process/Activity from the list.

14.2.4.3. Label

Policy Item	Configuration
Policy Label	Enter a name that represents this Policy on the Studio Process Tree and in Process Manager displays. Policies within the same hierarchy level must have unique Labels. The default Policy Label will always be a combination of the Event And the Action.

back to top

14.2.5. CopyLocalOperandWithID

14.2.5.1. Event/Action

Policy Item	Configuration
Event	Select the Event that will trigger this Policy Action from the drop down box. For detailed information see the Events section of the Policy Overview page.
Action	Select the Action that will execute when this Policy is triggered by the Event. When the Event is triggered, **CopyLocalOperandWithID** will copy the value of an Operand from this Process to an external Process. The *Refresh* button updates the set of Policy Actions that are available for designer selection. **Warning:** Clicking the *Refresh* button also erases any values in the Parameters box.
Description	Enter any desired information about this Policy as documentation. *This field is not required*.

14.2.5.2. Parameters

Policy Item	Configuration
Destination Operand Label	Select or enter the Operand Label from the external Process that will have the value of the Source Operand copied to it. The button to the right opens an Editor window for the user to either enter a long Operand Label or load the Operand Label from a file.
Source Operand	Select or enter the Operand from this Process that will be copied to the external Process. The button to the right opens an object picker window listing available Operands for configuring the Source Operand.
Destination Process ID	Select the Operand that contains the external Process

	ID. The button to the right opens an Operand selection window. The button to the right opens an object picker window listing available Operands for configuring the Destination Process ID. The ID is the long alphanumeric GUID assigned by Reactor, not the Process Label.

14.2.5.3. Label

Policy Item	Configuration
Policy Label	Enter a name that represents this Policy on the Studio Process Tree and in Process Manager displays. Policies within the same hierarchy level must have unique Labels. The default Policy Label will always be a combination of the Event And the Action.

back to top

14.2.6. CopyOperand

14.2.6.1. Event/Action

Policy Item	Configuration
Event	Select the Event that will trigger this Policy Action from the drop down box. For detailed information see the Events section of the Policy Overview page.
Action	Select the Action that will execute when this Policy is triggered by the Event. When the Event is triggered, **CopyOperand** will copy the value of the Source Operand to the Destination Operand within this Process. The *Refresh* button updates the set of Policy Actions that are available for designer selection. **Warning:** Clicking the *Refresh* button also erases any values in the Parameters box.

Description	Enter any desired information about this Policy as documentation. *This field is not required*.

14.2.6.2. Parameters

Policy Item	Configuration
Destination Operand	Select the Operand that will be updated with the new value. The button to the right opens an object picker window listing available Operands for configuring the Destination Operand.
Source Operand	Select the Operand that will be copied. The button to the right opens an object picker window listing available Operands for configuring the Source Operand.

14.2.6.3. Label

Policy Item	Configuration
Policy Label	Enter a name that represents this Policy on the Studio Process Tree and in Process Manager displays. Policies within the same hierarchy level must have unique Labels. The default Policy Label will always be a combination of the Event And the Action.

back to top

14.2.7. CopyRemoteOperandWithID

14.2.7.1. Event/Action

Policy Item	Configuration
Event	Select the Event that will trigger this Policy Action from the drop down box. For detailed information see the Events section of the Policy Overview page.
Action	Select the Action that will execute when this Policy is triggered by the Event. When the Event is triggered,

	CopyRemoteOperandWithID will copy the value of an Operand from an external Process to this Process. The *Refresh* button updates the set of Policy Actions that are available for designer selection. **Warning:** Clicking the *Refresh* button also erases any values in the Parameters box.
Description	Enter any desired information about this Policy as documentation. *This field is not required*.

14.2.7.2. Parameters

Policy Item	Configuration
Destination Operand	Select or enter the Operand from the this Process that will have the value of the Source Operand copied to it. The button to the right opens an object picker window listing available Operands for configuring the Destination Operand.
Source Process ID	Select the Operand that contains the external Process ID. The button to the right opens an object picker window listing available Operands for configuring the Source Process ID.
Source Operand Label	Select or enter the Operand Label from the external Process. The button to the right opens an Editor window for the user to either enter a long Operand Label or load the Operand Label from a file.

14.2.7.3. Label

Policy Item	Configuration
Policy Label	Enter a name that represents this Policy on the Studio Process Tree and in Process Manager displays. Policies within the same hierarchy level must have unique Labels. The default Policy Label will always be a combination of the Event And the Action.

back to top

14.2.8. CreateApprovalInstances

14.2.8.1. Event/Action

Policy Item	Configuration
Event	Select the Event that will trigger this Policy Action from the drop down box. For detailed information see the Events section of the Policy Overview page.
Action	Select the Action that will execute when this Policy is triggered by the Event. When the Event is triggered, **CreateApprovalInstances** will require a decision from a set of Actors before allowing the Process to progress. If all Actors approve, the Process will add the Approved Status to the Approval Activity. If any Actor denies, the Process will add the Denied Status to the Approval Activity. The *Refresh* button updates the set of Policy Actions that are available for designer selection. **Warning:** Clicking the *Refresh* button also erases any values in the Parameters box.
Description	Enter any desired information about this Policy as documentation. *This field is not required*.

14.2.8.2. Parameters

Policy Item	Configuration
Approval Activity	Select or enter the Activity that will have the Approval or Denied Status added to it. The button to the right opens an object picker window to select the Process/Activity to which the Approved Status or the Denied Status will be added.
Approved Status	Select or enter the Status that will be added to

	the Approval Activity if all Actors approve. The button to the right opens an object picker window to select the Status that will be added.
Denied Status	Select or enter the Status that will be added to the Approval Activity if any of the Actors deny. The button to the right opens an object picker window to select the Status that will be added.
ACL Names	Select or enter the Operand containing the Name(s) of the approval process participants. The button to the right opens an object picker window to select. the desired Operand.
ACL Type	Select or enter the Type of participant (user, title, group). The button to the right opens an Editor window for the user to either enter text manually or load text from a file.
ACL Role	Enter the Role of the task participants. The default is *initiative participant* and in most cases it will not require modification. The button to the right opens an Editor window for the user to either enter text manually or load text from a file.

14.2.8.3. Label

Policy Item	Configuration
Policy Label	Enter a name that represents this Policy on the Studio Process Tree and in Process Manager displays. Policies within the same hierarchy level must have unique Labels. The default Policy Label will always be a combination of the Event And the Action.

back to top

14.2.9. Custom

14.2.9.1. Event/Action

Policy Item	Configuration
Event	Select the Event that will trigger this Policy

	Action from the drop down box. For detailed information see the Events section of the Policy Overview page.
Action	Select the Action that will execute when this Policy is triggered by the Event. When the Event is triggered, **Custom** will execute custom code. The *Refresh* button updates the set of Policy Actions that are available for designer selection. **Warning:** Clicking the *Refresh* button also erases any values in the Parameters box.
Description	Enter any desired information about this Policy as documentation. *This field is not required*.

14.2.9.2. Parameters

Policy Item	Configuration
Language	Select the language of the custom code that that will be executed. *Java* is the default value. Other options include *jacl*, *javascript*, *jpython* and *netrexx*.
Source Type	Select the source type for the custom code that will be executed. *Source code* is the default value. Other options include *classname* and *URL*.
Source	Enter the source code, classname or URL. The *Edit Source..* button opens an Editor window for the user to either type in the source code, classname or URL to execute or gives the designer an option to load it from a file.

14.2.9.3. Label

Policy Item	Configuration
Policy Label	Enter a name that represents this Policy on the Studio Process Tree and in Process Manager displays. Policies within the same hierarchy level must have unique Labels. The default Policy Label will always be a combination of the Event

	And the Action.

back to top

14.2.10. DBObjectPolicy

14.2.10.1. Event/Action

Policy Item	Configuration
Event	Select the Event that will trigger this Policy Action from the drop down box. For detailed information see the Events section of the Policy Overview page.
Action	Select the Action that will execute when this Policy is triggered by the Event. When the Event is triggered, **DBObjectPolicy** will either get a value from a database or update an existing database field. This Policy requires a particular Operand Data Type called Database Object in order to work properly. See the Database Object Operand page for more configuration information. The *Refresh* button updates the set of Policy Actions that are available for designer selection. **Warning:** Clicking the *Refresh* button also erases any values in the Parameters box.
Description	Enter any desired information about this Policy as documentation. *This field is not required*.

14.2.10.2. Parameters

Policy Item	Configuration
DBObjectOperand	Enter the Database Object Operand containing the parameters for determining whether this Policy will pull data from or push data to the database. Operand also has information required to access the database. The button to

	the right opens an object picker window listing available Operands for configuring the DB Object Operand.

14.2.10.3. Label

Policy Item	Configuration
Policy Label	Enter a name that represents this Policy on the Studio Process Tree and in Process Manager displays. Policies within the same hierarchy level must have unique Labels. The default Policy Label will always be a combination of the Event And the Action.

back to top

14.2.11. IncrementOperand

14.2.11.1. Event/Action

Policy Item	Configuration
Event	Select the Event that will trigger this Policy Action from the drop down box. For detailed information see the Events section of the Policy Overview page.
Action	Select the Action that will execute when this Policy is triggered by the Event. When the Event is triggered, **IncrementOperand** will cause the selected Operand to be incremented by the amount specified. The *Refresh* button updates the set of Policy Actions that are available for designer selection. **Warning:** Clicking the *Refresh* button also erases any values in the Parameters box.
Description	Enter any desired information about this Policy as documentation. *This field is not required*.

14.2.11.2. Parameters

Policy Item	Configuration
Operand	Select or enter the Operand that will be incremented. The button to the right opens an object picker window listing available Operands.
Increment Value	Enter the value by which the Operand should be incremented. Opens an Editor window for the user to either enter a value or load it from a file. The default increment value is 1.

14.2.11.3. Label

Policy Item	Configuration
Policy Label	Enter a name that represents this Policy on the Studio Process Tree and in Process Manager displays. Policies within the same hierarchy level must have unique Labels. The default Policy Label will always be a combination of the Event And the Action.

back to top

14.2.12. RemoveStatusFromProcess

14.2.12.1. Event/Action

Policy Item	Configuration
Event	Select the Event that will trigger this Policy Action from the drop down box. For detailed information see the Events section of the Policy Overview page.
Action	Select the Action that will execute when this Policy is triggered by the Event. When the Event is triggered, **RemoveStatusFromProcess** will remove the Status selected below in the Parameters box from the Process/Activity selected below in the Parameters box.

	The *Refresh* button updates the set of Policy Actions that are available for designer selection. **Warning:** Clicking the *Refresh* button also erases any values in the Parameters box.
Description	Enter any desired information about this Policy as documentation. *This field is not required*.

14.2.12.2. Parameters

Policy Item	Configuration
Status	Select or enter the Status that will be removed from the selected Process/Activity below. The button to the right Opens an object picker window listing available Statuses for configuring this portion of the Parameters box.
Process	Select or enter the Process/Activity that will have a Status removed from it. The button to the right Opens an object picker window listing available Statuses for configuring this portion of the Parameters box.

14.2.12.3. Label

Policy Item	Configuration
Policy Label	Enter a name that represents this Policy on the Studio Process Tree and in Process Manager displays. Policies within the same hierarchy level must have unique Labels. The default Policy Label will always be a combination of the Event And the Action.

back to top

14.2.13. SendNotification

14.2.13.1. Event/Action

Policy Item	Configuration
Event	Select the Event that will trigger this Policy Action from the drop down box. For detailed information see the Events section of the Policy Overview page.
Action	Select the Action that will execute when this Policy is triggered by the Event. When the Event is triggered, **SendNotification** will send an email to the recipient specified in the *To* parameter. The *Refresh* button updates the set of Policy Actions that are available for designer selection. **Warning:** Clicking the *Refresh* button also erases any values in the Parameters box.
Description	Enter any desired information about this Policy as documentation. *This field is not required*.

14.2.13.2. Parameters

Policy Item	Configuration
From	Enter the email address of the message originator. The button to the right opens an Editor window for the user to either manually enter the required information if it is lengthy or load the required information from a file. **Warning:** Any spaces will cause an error.
Host	Enter SMTP. The email host name must be properly configured in the `Process Manager.properties` section of the Reactor Configuration Tool. See the section Configuring Reactor for more information. The button to the right opens an Editor window for the user to either manually enter the required information if it is lengthy or load the required information from a file.
Subject	Enter the subject line text for the email. The

	button to the right opens an Editor window for the user to either enter text manually or load text from a file.
Message	Enter the text for the body of the email message. The button to the right opens an Editor window for the user to either enter text manually or load text from a file.
To	Enter the e-mail address of the recipient. The button to the right opens an Editor window for the user to either enter text manually or load text from a file. **Warning:** This Policy Action only works for a single recipient.

14.2.13.3. Label

Policy Item	Configuration
Policy Label	Enter a name that represents this Policy on the Studio Process Tree and in Process Manager displays. Policies within the same hierarchy level must have unique Labels. The default Policy Label will always be a combination of the Event And the Action.

back to top

14.2.14. SendNotificationByOperand

14.2.14.1. Event/Action

Policy Item	Configuration
Event	Select the Event that will trigger this Policy Action from the drop down box. For detailed information see the Events section of the Policy Overview page.
Action	Select the Action that will execute when this Policy is triggered by the Event. When the Event is triggered,

	SendNotificationByOperand will send an email to the recipient or recipients specified in an Operand. This Policy also uses Operands to set mail server host and message body. The *Refresh* button updates the set of Policy Actions that are available for designer selection. **Warning:** Clicking the *Refresh* button also erases any values in the Parameters box.
Description	Enter any desired information about this Policy as documentation. *This field is not required*.

14.2.14.2. Parameters

Policy Item	Configuration
Host	Select the Operand containing the text SMTP. The email host name must be properly configured in the `Process Manager.properties` section of the Reactor Configuration Tool. See the section Configuring Reactor for more information. The button to the right opens an object picker window, displaying the available Operands for selection.
Subject	Enter the subject line text for the email. The button to the right opens an Editor window for the user to either enter text manually or load text from a file.
Message	Select the Operand containing the body of the message. The button to the right opens an object picker window, displaying the available Operands for selection.
User	Select the Operand containing the email recipient name or names. The button to the right opens an object picker window, displaying the available Operands for selection.

14.2.14.3. Label

Policy Item	Configuration

Policy Label	Enter a name that represents this Policy on the Studio Process Tree and in Process Manager displays. Policies within the same hierarchy level must have unique Labels. The default Policy Label will always be a combination of the Event And the Action.

back to top

14.2.15. SendNotificationByRole

14.2.15.1. Event/Action

Policy Item	Configuration
Event	Select the Event that will trigger this Policy Action from the drop down box. For detailed information see the Events section of the Policy Overview page.
Action	Select the Action that will execute when this Policy is triggered by the Event. When the Event is triggered, **SendNotificationByRole** will send an email to the recipient or recipients based on their Role. This Policy also uses Operands to set mail server host and message body. The *Refresh* button updates the set of Policy Actions that are available for designer selection. **Warning:** Clicking the *Refresh* button also erases any values in the Parameters box.
Description	Enter any desired information about this Policy as documentation. *This field is not required*.

14.2.15.2. Parameters

Policy Item	Configuration
Host	Select the Operand containing the text SMTP. The email host name must be properly

	configured in the `Process Manager.properties` section of the Reactor Configuration Tool. See the section Configuring Reactor for more information. The button to the right opens an object picker window, displaying the available Operands for selection.
Subject	Enter the subject line text for the email. The button to the right opens an Editor window for the user to either enter text manually or load text from a file.
Message	Select the Operand containing the body of the message. The button to the right opens an object picker window, displaying the available Operands for selection.
Role	Select the Operand containing the Role of the participants to which the email will be sent. The button to the right opens an object picker window, displaying the available Operands for selection.

14.2.15.3. Label

Policy Item	Configuration
Policy Label	Enter a name that represents this Policy on the Studio Process Tree and in Process Manager displays. Policies within the same hierarchy level must have unique Labels. The default Policy Label will always be a combination of the Event And the Action.

back to top

14.2.16. SetACLofProcess

14.2.16.1. Event/Action

Policy Item	Configuration
Event	Select the Event that will trigger this Policy Action from the drop down box. For detailed information see the Events section of the Policy

	Overview page.
Action	Select the Action that will execute when this Policy is triggered by the Event. When the Event is triggered, **SetACLofProcess** will add an Actor List to a Process/Activity and set their Role in that Process/Activity. The *Refresh* button updates the set of Policy Actions that are available for designer selection. **Warning:** Clicking the *Refresh* button also erases any values in the Parameters box.
Description	Enter any desired information about this Policy as documentation. *This field is not required*.

14.2.16.2. Parameters

Policy Item	Configuration
Process	Enter the Process/Activity to which an Actor List will be added. The button to the right opens an object picker window to select the Process/Activity to which a new Actor will be added.
Name	Select the Operand containing the name of the Group, User or Organizational Role to be added. The button to the right opens an object picker window, displaying the available Operands for selection.
Role	Select one of the seven Roles from the drop down list.

14.2.16.3. Label

Policy Item	Configuration
Policy Label	Enter a name that represents this Policy on the Studio Process Tree and in Process Manager displays. Policies within the same hierarchy level must have unique Labels. The default Policy Label will always be a combination of the Event And the Action.

back to top

14.2.17. SetMultipleProcessACEs

14.2.17.1. Event/Action

Policy Item	Configuration
Event	Select the Event that will trigger this Policy Action from the drop down box. For detailed information see the Events section of the Policy Overview page.
Action	Select the Action that will execute when this Policy is triggered by the Event. When the Event is triggered, **SetMultipleProcessACEs** will add multiple Actors to a Process/Activity and set their Role in that Process/Activity. The *Refresh* button updates the set of Policy Actions that are available for designer selection. **Warning:** Clicking the *Refresh* button also erases any values in the Parameters box.
Description	Enter any desired information about this Policy as documentation. *This field is not required*.

14.2.17.2. Parameters

Policy Item	Configuration
Delimiter	Enter the delimiter used to separate the Actor Names. The default is a comma. The button to the right opens an Editor window for the user to either enter the delimiter manually or load the information from a file.
Process	Enter the Process/Activity to which the Actors will be added. The button to the right opens an object picker window to select the Process/Activity to which multiple new Actors will be added.

Name	Select the Operand containing the name of the new Actors to be added. The button to the right opens an object picker window, displaying the available Operands for selection.
Role	Select one of the seven Roles from the drop down list.

14.2.17.3. Label

Policy Item	Configuration
Policy Label	Enter a name that represents this Policy on the Studio Process Tree and in Process Manager displays. Policies within the same hierarchy level must have unique Labels. The default Policy Label will always be a combination of the Event And the Action.

back to top

14.2.18. SetPicklistItems

14.2.18.1. Event/Action

Policy Item	Configuration
Event	Select the Event that will trigger this Policy Action from the drop down box. For detailed information see the Events section of the Policy Overview page.
Action	Select the Action that will execute when this Policy is triggered by the Event. When the Event is triggered, **SetPicklistItems** will set the Picklist Items for a Picklist Operand. The *Refresh* button updates the set of Policy Actions that are available for designer selection. **Warning:** Clicking the *Refresh* button also erases any values in the Parameters box.

Description	Enter any desired information about this Policy as documentation. *This field is not required*.

14.2.18.2. Parameters

Policy Item	Configuration
Picklist Operand	Enter or select the Picklist Operand. The button to the right opens an object picker window, displaying the available Operands for selection.
Picklist Values	Enter the list of values to populate the Picklist Operand. Use a comma delimiter. The button to the right opens an Editor window for the user to either manually enter a long list or to load the information from a file.

14.2.18.3. Label

Policy Item	Configuration
Policy Label	Enter a name that represents this Policy on the Studio Process Tree and in Process Manager displays. Policies within the same hierarchy level must have unique Labels. The default Policy Label will always be a combination of the Event And the Action.

back to top

14.2.19. StartProcess

14.2.19.1. Event/Action

Policy Item	Configuration
Event	Select the Event that will trigger this Policy Action from the drop down box. For detailed information see the Events section of the Policy Overview page.
Action	Select the Action that will execute when this Policy is triggered by the Event. When the Event is triggered,

	StartProcess will start an Activity or Subprocess. The *Refresh* button updates the set of Policy Actions that are available for designer selection. **Warning:** Clicking the *Refresh* button also erases any values in the Parameters box.
Description	Enter any desired information about this Policy as documentation. *This field is not required*.

14.2.19.2. Parameters

Policy Item	Configuration
ProcessToStart	Enter the Activity or Subprocess that will be started. The button to the right opens an object picker window listing available Activities and Subprocesses.

14.2.19.3. Label

Policy Item	Configuration
Policy Label	Enter a name that represents this Policy on the Studio Process Tree and in Process Manager displays. Policies within the same hierarchy level must have unique Labels. The default Policy Label will always be a combination of the Event And the Action.

back to top

14.2.20. StartProcessWithLabel

14.2.20.1. Event/Action

Policy Item	Configuration
Event	Select the Event that will trigger this Policy Action from the drop down box. For detailed information see the Events section of the Policy

	Overview page.
Action	Select the Action that will execute when this Policy is triggered by the Event. When the Event is triggered, **StartProcessWithLabel** will start a separate external Process. The *Refresh* button updates the set of Policy Actions that are available for designer selection. **Warning:** Clicking the *Refresh* button also erases any values in the Parameters box.
Description	Enter any desired information about this Policy as documentation. *This field is not required*.

14.2.20.2. Parameters

Policy Item	Configuration
Return Address Operand	Enter the name of the Operand that contains the Return Address. The button to the right opens an Editor window for the user to either enter the name of the Return Address Operand manually or load it from a file.
Started Process Address	Select the Operand that contains the address of the external Process to be started using the button to the right. The button to the right opens an object picker window, displaying the available Operands for selection.
Process To Start	Select The Operand containing the name of the external Process to be started using the button to the right. The button to the right opens an object picker window, displaying the available Operands for selection.

14.2.20.3. Label

Policy Item	Configuration
Policy Label	Enter a name that represents this Policy on the Studio Process Tree and in Process Manager displays. Policies within the same hierarchy level must have unique Labels. The default Policy

	Label will always be a combination of the Event And the Action.

back to top

14.2.21. StopProcess

14.2.21.1. Event/Action

Policy Item	Configuration
Event	Select the Event that will trigger this Policy Action from the drop down box. For detailed information see the Events section of the Policy Overview page.
Action	Select the Action that will execute when this Policy is triggered by the Event. When the Event is triggered, **StopProcess** will cause an Activity or Subprocess to finish. All Policies contained by the Activity/Subprocess will execute and any Statuses generated by those Policies will be added to the Activity/Subprocess. This Policy is most commonly used to drive an automated Activity to completion within a Reactor Process. The *Refresh* button updates the set of Policy Actions that are available for designer selection. **Warning:** Clicking the *Refresh* button also erases any values in the Parameters box.
Description	Enter any desired information about this Policy as documentation. *This field is not required*.

14.2.21.2. Parameters

Policy Item	Configuration
ProcessToStop	Enter the Activity or Subprocess that will be driven to a (`finished`) state. The button to the right opens an object picker window listing

	available Activities and Subprocesses.

14.2.21.3. Label

Policy Item	Configuration
Policy Label	Enter a name that represents this Policy on the Studio Process Tree and in Process Manager displays. Policies within the same hierarchy level must have unique Labels. The default Policy Label will always be a combination of the Event And the Action.

back to top

14.2.22. StopProcessWithID

14.2.22.1. Event/Action

Policy Item	Configuration
Event	Select the Event that will trigger this Policy Action from the drop down box. For detailed information see the Events section of the Policy Overview page.
Action	Select the Action that will execute when this Policy is triggered by the Event. When the Event is triggered, **StopProcessWithID** will cause an Activity or Subprocess to finish. All Policies contained by the Activity/Subprocess will execute and any Statuses generated by those Policies will be added to the Activity/Subprocess. The *Refresh* button updates the set of Policy Actions that are available for designer selection. **Warning:** Clicking the *Refresh* button also erases any values in the Parameters box.
Description	Enter any desired information about this Policy as documentation. *This field is not required*.

14.2.22.2. Parameters

Policy Item	Configuration
ID of Process to Stop	Select the Operand that contains the ID of the external Process to be stopped using the button to the right. The button to the right opens an object picker window, displaying the available Operands for selection. The ID is the long alphanumeric GUID assigned by Reactor and stored in the database, not the Process Label shown by Process Manager or Studio.

14.2.22.3. Label

Policy Item	Configuration
Policy Label	Enter a name that represents this Policy on the Studio Process Tree and in Process Manager displays. Policies within the same hierarchy level must have unique Labels. The default Policy Label will always be a combination of the Event And the Action.

back to top

14.3. Creating Custom Policies

Process designers should not have to write custom code to define common Policy Actions. Reactor Studio provides a mechanism for process designers to select a predefined Policy Action and define parameters to create a new instance of that Policy Action. The Edit Policy dialog box in Reactor Studio provides a drop-down list of Actions. One of these is called Custom, which refers to a policy that requires custom code to be typed or loaded from a file. The other Policy Actions are loaded from XML files found in the `C:\Reactor6.0-JBoss3.2.5\studio\actions\` directory installed with Reactor Studio. These other Policy Actions are Parameterized Policies, because they can require parameters to create specific Policies. The Reactor Studio application loads the XML files every time it is started. Process designers can create custom Policy Actions by creating new XML files that define Parameterized Policies. Reactor Studio incorporates the new Policy Actions the next time it is started.

14.3.1. Initial Steps

Decide on a name for the Policy Action template. Write a brief description of what the Policy does. The name and description should provide enough information for process designers to understand the purpose of the Policy. Decide what parameters the process designer should be able to set. Parameters can be plain text values. Parameters can be label paths to Reactor objects (Process, Operand, Status, or Policy) relative to the policy location. Parameters can be Roles with respect to the Process, where the value would be something like Initiative Participant or Process Invoker.

The value for a parameter can be set statically or dynamically. When it is set statically, the process designer explicitly defines a value for the parameter. When the value is set dynamically, the process designer specifies an Operand that will contain the value when the Policy is executed. The string representation of the relative label path to the Operand will be substituted into the Policy script template when an Operand is chosen by the process designer. The Policy script must explicitly include code to create a label path object from the label path string. Decide which parameters should be set dynamically and which should be set explicitly by the process designer.

14.3.2. Writing the Policy

Write the Policy script in the same way a custom script is written. In places where a parameter should be inserted into the text of the policy, use ${Name} (where Name is the name of the parameter.)

For example, consider a Parameterized Policy with a parameter named Process that is given a value of `./ApprovePurchase`. This line appears in the parameterized script shown below.

```
LabelPath lp = navigator.getRelativeLabelPath("${Process}");
```

The line will be changed to the following in the Policy sent to Reactor Engine.

```
LabelPath lp = navigator.getRelativeLabelPath("./ApprovePurchase");
```

14.3.3. Interacting with Reactor Engine

The Policy can make requests to Reactor Engine to get and set objects. This is done by using a Navigator object. The Policy script can get a Navigator object by including the following

line of code.

```
NavigatorProxy navigator = (NavigatorProxy) bsf.lookupBean("proxy");
```

This Navigator object is created by Reactor Engine when the Policy is executed. It has a valid authentication token, which is required for all interactions with Reactor Engine. For more details, see the Javadoc describing the `com.oakgrovesystems.util.Navigator` class.

14.3.4. Getting Parameter Values from Operands

In the case of a parameter that is set dynamically, the value set in the template is actually just the relative label path to the Operand. The Policy script must get that Operand explicitly. The following code illustrates how a dynamic parameter value might be accessed by a Policy script.

```
// Get role from operand
LabelPath operandLabelPath =navigator.getRelativeLabelPath("${Role}");
Operand operand = navigator.getOperand(operandLabelPath);
if (operand == null) {
    navigator.logError("Could not get Role operand");
    return null;
}
String role = operand.getValue();
```

14.3.5. Creating the XML

First, take a look at the XML files in the `C:\Reactor6.0-JBoss3.2.5\studio\actions\` directory. These are the Parameterized Policies delivered with Reactor Studio, and they might serve as useful examples.

Start the new policy XML file with a standard XML declaration.

```
<?xml version="1.0" encoding="UTF-8" ?>
```

Start the root element, named policy.

```
<policy>
```

Set the name of the new Parameterized Policy Action.

```
<policy_type>StartProcessWithAssignee</policy_type>
```

Set the source type of the Policy, which will generally be *Source Code*.

```
<source_type>Source Code</source_type>
```

Add the Policy script source code. (Be sure to exit with *return null*)

```
<source>
 <![CDATA[
 try {
  NavigatorProxy navigator = (NavigatorProxy) bsf.lookupBean("proxy");
  LabelPath processPath = navigator.getRelativeLabelPath("${Process}");
  LabelPath assigneePath = navigator.getRelativeLabelPath("${Assignee}");
  navigator.setACLofProcess(processPath, "Initiative Participant",
assigneePath);
  navigator.startProcess(processPath);
 }
 catch (com.oakgrovesystems.reactor.client.ReactorProxyException e)
 {
    e.printStackTrace();
 }
 return null;
 ]]>
</source>
```

Add the Policy description.

```
<description>
       Starts another process, after assigning a user to be the
       Initiative Participant.
</description>
```

Specify the programming language used by the script source code.

```
<language>java</language>
```

Add the parameters. The parameter element has an Operand attribute which can be `TRUE` or `False`. The parameter element contains three elements: *name*, *data_type*, and *default_value*. The name must contain a value, but the other elements can be empty. If the data_type

element is empty, the parameter will be assumed to contain plain text.

Below are two parameters, one set dynamically from an Operand, and another set statically by the process designer.

```
<parameters>
  <parameter operand="true">
    <name>Assignee</name>
    <data_type/>
    <default_value/>
  </parameter>
  <parameter operand="false">
    <name>Process</name>
    <data_type>Process</data_type>
    <default_value/>
  </parameter>
</parameters>
```

Finally, close the Policy element.

```
</policy>
```

14.3.6. Using the New Parameterized Policy

Put the XML file into the `C:\Reactor6.0-JBoss3.2.5\studio\actions\` directory. Reactor Studio must be restarted before the new Policy Action will appear on the drop down list of available Actions.

Below is a complete sample XML file for a new Parameterized Policy Action.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<policy>
 <policy_type>StartProcessWithAssignee</policy_type>
 <source_type>Source Code</source_type>
 <source>
 <![CDATA[
 try {
  NavigatorProxy navigator = (NavigatorProxy) bsf.lookupBean("proxy");
  LabelPath processPath = navigator.getRelativeLabelPath("${Process}");
  LabelPath assigneePath = navigator.getRelativeLabelPath("${Assignee}");
  navigator.setACLofProcess(processPath, "Initiative Participant",
assigneePath);
  navigator.startProcess(processPath);
 }
 catch (com.oakgrovesystems.reactor.client.ReactorProxyException e)
 {
```

```
        e.printStackTrace();
    }
    return null;
    ]]>
    </source>
    <description>
        Starts another process, after assigning a user to be the
        Initiative Participant.
    </description>
    <language>java</language>
    <parameters>
        <parameter operand="true">
            <name>Assignee</name>
            <data_type/>
            <default_value/>
        </parameter>
        <parameter operand="false">
            <name>Process</name>
            <data_type>Process</data_type>
            <default_value/>
        </parameter>
    </parameters>
</policy>
```

15. Involving Users: Actors And Forms

15.1. Actors

The Actor Process Component is used to specify the responsibilities of the various participants in a Process. This page displays an Edit Actor Properties window and provides guidance on how to complete each of the attributes of an Actor.

There are two attributes that can be specified about any Actor, the Role and the Identity. These attributes determine how that Actor will participate in the Process.

15.1.1. Actor's Role in Process

First the Actor is assigned one of the seven Roles listed below.

Role	Description
Initiative Participant	Assigning this role to an Actor means that the Actor will be directly responsible for completing an Activity.
Eligible Participant	Assigning this role to an Actor means that the Actor will receive a particular Activity request on their Activity List in Process Manager. Multiple Users, Groups or Titles can become *Eligible*

	Participants. The first participant that accepts the request will become the *Initiative Participant*, with all others becoming *Standby Participants*. This role is generally given to several participants when the task will be completed by the first available person in the group.
Standby Participant	This role is not generally directly assigned to an Actor. Any Actors who are *Eligible Participants* become *Standby Participants* once an Activity has been accepted by another *Eligible Participant*. If that request should time out due to a configured Policy, then all of the *Standby Participants* become *Eligible Participants* once again.
Process Engineer	Reactor automatically assigns this designation to the person who created and uploaded the Process to the Reactor Engine.
Process Invoker	Reactor automatically assigns this Role to anyone who initiates a Process Instance.
Process Owner	This role is generally assigned to the person who has overall responsibility for the successful execution of the Process.
Spectator	This role can be assigned to anyone who may have an interest in viewing the current state of a Process.

15.1.2. Identity

Select the Identity of Actor using the Radio buttons. The Identity value is used to determine who will participate in the various Activities of the Process. Valid Identities are read from the `Reactor-6.0.ear` file, which can be edited by running `configure.bat` (Windows) or `configure.sh` (Unix) in the `C:\Reactor6.0-JBoss3.2.5\` directory. There are three methods for specifying the Identity of an Actor in Reactor.

Method	Description
User Name	This Identity refers to a specific individual. A **User Name** may have multiple Organizational Roles and belong to multiple Groups.
Group Name	This Identity refers to any Group of Users, as specified in the CSV file. A **Group Name** may be comprised of multiple

	Users with multiple Organizational Roles.
Organizational Role Name	This Identity generally refers to a position title, as specified in the CSV file. It can be used for formal titles and/or informal titles. An **Organizational Role Name** may have multiple Users and belong to multiple Groups.

15.2. Activity Form Reference

There are three types of Forms available for use within Reactor.

- Start Form
- Work Item
- Completion Form

There is only one Start Form for each Process, which is what the Process Invoker sees when initiating the Process from Process Manager. There can be as many Work Item Forms as Activities in a Process. Each Work Item Form is associated with a particular Activity and is only accessible by participants in that Activity. There can also be as many Completion Forms as Activities in a Process. Like Work Item Forms, Completion Forms are associated with a particular Activity and are only accessible by Activity participants. Work Item Forms will be the most commonly used Forms in Reactor. Completion Forms are not used in many situations, but are most useful in situations where feedback is required after completing an Activity. A sample Work Item Form design window from Studio is shown below.

15.2.1. Creating A Form

To create a Form that will be displayed with a particular Activity (we'll use a Work Item Form in this example), right-click on the desired Activity box and select the edit option. Click the button that is labeled *Edit Work Item* and a new window will open allowing you to specify the necessary information for the Form. The upper box is where you can enter any instructions or other pertinent information that should be conveyed to the Form recipient. The lower left box displays all of the Operands that are available to be displayed on the Form. The lower right box displays the Operands that will be displayed on the Form, along with several configurable properties for each of the displayed Operands:

- whether an entry is required for the displayed Operand.
- whether the Operand value can be edited by the individual viewing the Form.
- What text (if any) should be shown next to the Operand box on the Form to provide guidance in entering a value for the Operand.

The designer has the option to add new Operands to the Process while editing the Form by clicking the *New Operand* button. This will open an Operand configuration window in the

same way as if the Operand was being added via the Process Tree.

For more information on how to configure the various Forms in Studio, see the Forms section of the Process Component Reference in Chapter 5.

The following screen shot shows all of the possible parts of a Form displayed in Process Manager. Depending on how the Form has been configured in Studio, only some of the sections may be visible.

15.2.2. Start Form

The Start Form is a web interface that is deployed when a Process is first initiated. It is generally used by the Process Invoker to provide information necessary to carry out the various Activities within the Process. Designers can configure Start Forms to contain text instructions, Operand values, Process Attributes and Operand input elements. The configured elements of a custom Start Form are inserted into the Form Header, Activity and Operand sections of this web page.

15.2.3. Work Item

Process Manager Forms have four basic parts called the Form Header, the Activity section, the Operand section and the Annotations.

A Work Item is a form that is used within the Process to accomplish tasks and/or communicate information. The Work Item can be configured to display up to four sets of information.

- **Process Attributes**
  information about the particular Process Instance, which may include things like the Process Title, the Process Label, the time started and the Process Invoker. Process Attributes are displayed in the Form Header.
- **Instructions**
  text information generally entered by the Process designer to provide guidance regarding the completion of the assigned Activity. Instructions are displayed in the Activity section of the Form.
- **Action Buttons**
  for instances where a decision or an acknowledgment is required in order to determine the next step in the Process. The Action Buttons that will be visible on the form are determined by the Statuses on the connecting lines exiting an Activity. So if there is only one connecting line named Done exiting an Activity, then the Form will display only one button with the caption Done as in the picture above. Depending on how the Activity is configured, two additional buttons Delegate and Transfer may be available to the Form recipient. See the Activities and Subprocesses section in the Process Component

Reference for more information.
Action buttons are displayed in the Activities section of the Form. To suppress Action Buttons on a Work Item Form use Statuses starting with an underscore (i.e. _Done). See Statuses for more details.

- **Operands**
  relevant pieces of Process data that are communicated as part of the Process or are required to progress the Process. Operands are displayed in the Operand section of the Form.
- **Annotations**
  Process execution notes added by participants that may include relevant information for participants further along in the Process. Annotations are displayed in their own section at the bottom of the Form. They only exist as part of an executing Process Instance and cannot be configured in Studio.

15.2.4. Completion Form

The Completion Form is generally used in a situation where feedback is desired upon the completion of an Activity. Nearly identical to the Work Item Form, the Completion Form can also be made to display Operands as well as performing its primary function of providing feedback.

15.2.5. Form Types

There are three types of Activity Forms available in Reactor. The configuration of each type of form is shown in detail below.

15.2.5.1. Start Form

Instructions

15.2.5.2. Work Item Form

Instructions

15.2.5.3. Completion Form

Instructions

16. Other Design Topics

16.1. Good Design Practices

Here are some guidelines that will help you design clean and effective workflow Processes for Reactor.

- Use verbs for Activity and Status names whenever possible.
- Scope Activities to contain as much work as can be completed before a handoff must occur.
- Encapsulate related Activities that produce an intermediate result into a Subprocess.

16.2. Customizing Process Manager

Process Manager is Reactor's web based application that executes workflows uploaded and created through Reactor Studio. The source code comes free with the bundle and it is found in the directory `C:\ReactorEnterprise6.0B1-JBoss3.2.5\samples\portal`.

The source code can be used as a reference to develop the web application using Reactor's API and also to customize the Process Manager.

16.2.1. Building The Process Manager Application

To recompile and rebuild the Process Manager application, use the Ant build file named `build.xml`, which contains a list of targets that build specific areas of the application.

Target	Purpose
clean	cleans up by deleting output from previous builds.
compile	compiles the Process Manager source codes.
config	creates `ReactorPortalConfig.jar` that contains the property file specific to Process Manager.
framework	creates `ReactorPortalFramework.jar` that contains java classes specific to Process Manager.
MapViewer Jar	creates `mapViewer.jar` that contains classes specific to Process Manager's map viewer.
reports	compiles the jasper report files.

webapp	creates `ReactorPortal.war` and `ReactorPortal.ear` that is web application itself.

16.2.2. Portal.properties File

`Portal.properties` is the configuration file that enables customization of certain areas of the Process Manager without coding. This is done by running the `configure.bat` file (Windows) or the `configure.sh` file (UNIX).

The following table lists each property and its purpose.

Property	Purpose
frontdesk.url	contains the location of the Reactor Engine.
attachments.module	contains the module/package that handles the File Operand.
attachments.file.size.limit	sets the maximum file size that can be stored in the File Operand.
attachments.temp.dir	temporary directory where the uploaded files are initially stored.
attachments.files.dir	directory where the uploaded files are permanently stored.
attachments.ds.name	sets the datasource used by the attachment module to obtain the jdbc connection.
date.format	sets the date format that will be applied when displaying dates.
date.initial	sets the initial date that matches the format in "date.format" property. **Warning:** This value must be "Monday January 1, 1900 01:00 am".
locale.language	sets the language parameter for creating the Local object used for configuring the date format.
locale.country	sets the country parameter for creating the Local object used for configuring the date format.

proxy.type	identifier used to created an XML or EJB proxy.
email.hostname	sets the hostname required when sending email notifications.
list.displaySize	sets the maximum number of items displayed in a list. This size is utilized when displaying pages such as the Instance Manager or Activity Manager.
jasper.report.dir	directory where the jasper report files are located.
com.oakgrove.reports.queryReportClass	identifies the module that will be used to generate the information for reports.
image.dir	directory where the images used by the web application are located.

16.2.3. Internationalization

Process Manager implements I18N (internationalization) that enables the customization of labels, texts, and titles. The property files are `ReactorPortalErrors.properties` and `ReactorPortalResources.properties`. These files are found in `ReactorPortalFramework.jar`, which is located inside `Reactor6.0.ear` under the package `com.oakgrove.i18n`.

16.2.4. Process Manager JARs and Directory Structure

The following three files are the repository for all of the Process Manager JAR files. They are located in the Reactor6.0.ear file. Their contents and directory structure are outlined below.

`ReactorPortal.war` is the web application of the Process Manager. This is where the application related files (JSP, HTML, images, etc.) are found.

Directory	Contents
/	where the JSP, CSS, HTML, include and Javascript files are located.
/images	where the images used are located.
/reports	where the jasper report files are located.
/web-inf	where the struts and deployment descriptors are located.

`ReactorPortalFramework.jar` is the package containing the web applications action classes.

`ReactorPortalConfig.jar` is the package containing the Process Manager property files.

16.2.5. Customizing Forms In Process Manager

Reactor Process Manager creates web pages for Process Start Forms, Activity Work Item Forms, and Activity Completion Forms. There may be times when it is useful to customize the pages for a particular Process. Reactor Process Manager provides mechanisms for customizing pages without writing new servlets or JSP pages. Before using the default conventions for creating pages, Reactor Process Manager checks for the existence of custom content associated with the process. Metadata can be set by using the Studio Edit Activity window. This metadata can specify custom content for the Start Form, a Work Item, or a Completion Form. It can also specify a JSP to use for displaying activities.

To display existing Operand values, include content with the following format:

```
${operand:OperandLabel}
```

Replace `OperandLabel` with the label of the Operand. The Operand must be associated with the current Activity, its parent, or an "ancestor" (parent of a parent, etc.). These values can be inserted anywhere in the HTML, including attributes of HTML elements and JavaScript function calls.

The following Process Attributes can be displayed:

```
${process:id}
${process:label}
${process:description}
${process:parent:id}
```

Like Operand values, Process Attributes can be inserted anywhere in the HTML, including

attributes of HTML elements and JavaScript function calls.

The Start Form and Completion Form can include HTML form input elements. These inputs are used by Process Manager Framework servlets to set Operand values. Only Operands directly associated with the Process can be set with this mechanism. The input name is the prefix operand: followed by the label of the Operand. For example:

```
<input name="operand:OperandLabel">
```

Other input types can be used as well, such as text areas and option lists.

The Start Form is the web page displayed when the user starts a top-level Process. The contents of a custom Start Form are inserted into the middle of this web page. The custom start form can contain Operand values, Process Attributes and Operand input elements:

```
${operand:OperandLabel}
${process:description}
<input name="operand:OperandLabel">
```

The Work Item is the web page that displays the details of an Activity. The contents of a Work Item are inserted into the middle of this web page. The custom Work Item can contain Operand values and Process Attributes:

```
${operand:OperandLabel}
${process:description}
```

The Completion Form is the web page displayed when a User completes an Activity. The contents of a Completion Form are inserted into the middle of this web page. The custom Completion Form can contain Operand values, Process Attributes and Operand input elements:

```
${operand:OperandLabel}
```

```
${process:description}

<input name="operand:OperandLabel">
```

The custom Activity JSP provides an option for more extensive customization than the custom Activity template allows, without writing a completely new web application. However, it requires having the JSP defined in the web application's web.xml deployment descriptor file. To specify a custom JSP page, put the name of the JSP page in the Activity JSP field of Studio's Edit Activity window, as it appears in the <servlet-name> element.

17. Using Reactor 6.0

This tab covers important information for Users who want to start, monitor, participate in and analyze results of Reactor Process workflows.

Getting Started
Information on connecting to Process Manager via your web browser.

Launching Processes
Information on Process Definitions and how to initiate a new Process Instance.

Completing Activities
Information on viewing and completing Activities.

Monitoring Progress
Information on monitoring the state of a Process.

Analyzing Results
How to use the reports to analyze the results in Reactor.

Workflow Examples
Stepping through an example Process workflow.

18. Reactor 6.0 User Guide

18.1. Getting Started

18.1.1. Starting Process Manager

The Reactor Process Manager web application is deployed when the Reactor Engine is started. It can be accessed in two ways:

1. By opening the `ProcessManager-WebClient.html` file located in

`C:\ReactorEnterprise6.0-JBoss3.2.5\` directory.
2. By opening your browser and entering `http://<hostname>:8080/ReactorPM`

Warning:
The Reactor Engine must be running in order to use Process Manager.

The <hostname> refers to the location of the Reactor Engine. If Reactor has been installed on a standalone PC, this value would typically be *localhost*. The page may also be bookmarked to make it easier to return to the Process Manager start page in the future.

18.1.2. Logging In

The first screen shown when Process Manager is activated is a typical login page.

Users need to type in their Username and Password to gain access to the Process Manager pages that control the initiation and execution of Processes within Reactor. If using the default CSV Files distributed with Reactor, any one of the following Username/Password combinations will work.

- admin/admin
- alice/alice
- bob/bob
- charlie/charlie
- frank/frank
- judy/judy
- susan/susan
- linda/linda
- jose/jose
- ruth/ruth
- tom/tom
- henry/henry

Note:
Username and Password entries are case-sensitive, so be sure to note their exact spelling prior to attempting to Login.

18.2. Launching Processes

18.2.1. Definition Manager

The Definition Manager displays all of the available Processes loaded into the Reactor Engine that can be initiated by a User. The diagram below shows the tabular display of the

Process Definitions and highlights the key functionality embedded within each listing.

The Definition Manager display is sorted alphabetically (a to z based on the Name column) when it first appears. However, it can be sorted in alternative ways by clicking on the desired column to sort by that column. Clicking on a column again will reverse the order of the sort. Double clicking on a specific Process Name will open a screen displaying the details about that Process Definition.

If the number of Process Definitions available is greater than the `list.displaySize` property in the `Portal.properties` file, then a blue *next* will be visible in the lower right corner of the page. Click on *next* to view the additional pages of Process Definitions.

The following table provides details about each of the columns displayed by Definition Manager.

Column	Description
Action	Allows the User to view details of, start or delete the Process Definition.
Name	The Process Label given to the Process in Studio.
Description	The Process Description attribute (if any) that was provided in Studio.
Created By	The User that uploaded the Process Definition to the Reactor Engine.
Last Modified	The time and date of the last change to the Process Definition.

There are three basic actions that can be accomplished from the Definition Manager page:

- Open a Process and view/edit the details (the folder icon)
- Start new Instances of a Process (the green arrow icon)
- Delete a Process from the Engine (the trashcan icon)

18.2.2. Starting An Instance

A Process Instance is a single execution of a particular Process. Generally, Users are given an opportunity to name each execution of a Process Instance so that it can be uniquely identified. The Process Instance is identified in Process Manager as Instance and saved in the Process as the Process Title Operand.

A new Instance of a Process workflow is started by clicking on the green arrow icon to the left of the Process Name. When a new Instance is started, a Start Form will become visible.

The most important field on this Form is the Instance Name. The Instance Name is how Users are able to keep track of multiple Instances of the same Process.

> **Note:**
> In most cases Instance Names should be meaningful, so that other Users can understand the purpose of that Instance at a glance.

If no Instance name is provided or duplicate Names are used, it is quite likely that there will be confusion over the purpose of each Instance. The Instance Name is also an important aspect of report writing. Meaningless or nonsensical Names will make any reports about running Instances or completed Instances much less useful. There are situations where the Process Name itself describes the workflow sufficiently and each instance is simply a sequential invocation of a highly specialized Process (i.e. a customer order entry Process). In such cases there may be no practical value to assigning a meaningful Instance Name.

The Instance is officially started once all of the required fields on the Start Form have been filled in and the *Start* button is clicked.

18.3. Instance Manager Screen

18.3.1. Process Instances

The Instances tab displays all of the Process Instances currently being executed by the Reactor Engine. The diagram below shows the tabular display of the Instances and highlights the key functionality embedded within each listing. The Instances display is sorted alphabetically (a to z based on the Name column) when it first appears. The Instances can be sorted in alternative ways by clicking on the desired column to sort by that column. Clicking on a column again will reverse the order of the sort. Double clicking on a specific Process Instance will open a screen showing the details of that Instance. The following table provides details about each of the columns displayed on the Instances tab.

Column	Description
Action	Allows User to view details of or stop a running Process Instance.
Name	The Process Title entered by the Process Invoker when the Process was initiated or that was designated by the custom Start Form.
Process	The Process Label of the Process that was used to generate this Process Instance, as shown on the Definitions tab.

Started	The time and date this Process Instance was started.
Started By	The Username of the User who started the Process Instance.

18.3.2. Viewing The State Of A Process Instance

A User can view the current State of a Process Instance by selecting the Instance tab in Process Manager. When a Process Instance is selected for review, the browser displays the following four tabs covering all aspects of that particular Process Instance.

18.3.2.1. Table View

The initial view upon selecting a Process Instance is called the Table View. This view displays all of the Actors, Operands, Policies and Activities that comprise the Process Instance and any data that has been entered, retrieved or otherwise captured for use in the Process.

18.3.2.2. Map View

The Map View is a graphical representation of the State of the Process Instance that appears similar to the Studio rendering. Through the use of color and labels, the Process map highlights the Process Components that are unstarted, started and finished. It also highlights the actual path followed by the Process and the current Statuses associated with each Connecting Line.

18.3.2.3. XML View

The XML View shows the actual XML code that is being executed. This can be useful for technical debugging purposes.

18.3.2.4. Audit Log

The Audit Log is a simple tabular log of all the Events related to a particular Process Instance with a description of the instructions executed by the Reactor Engine. The Audit Log includes the date, time, User, Event, Process Attributes and Process Parameters affected by Process actions.

18.4. Activity Manager Screen

18.4.1. Activities

The Activities tab displays all of the Activities currently assigned to the User. The diagram below shows a tabular display of an Activities page and highlights the key functionality embedded within each listing. The Activities display is sorted alphabetically (a to z based on the Name column) when it first appears. The Activities can be sorted in alternative ways by clicking on the desired column to sort by that column. Clicking on a column again will reverse the order of the sort. Double clicking on an Activity will open the Activity for the User to allow viewing and/or participation in the Activity. The following table provides details about each of the columns displayed on the Activities tab.

Column	Description
Action	Allows the User to view and participate in the Activity.
Name	The Activity Label given to the Activity in Studio.
Instance	The Process Title of the Process this Activity belongs to, as shown on the Instances tab.
Definition	The Process Label of the Process this Activity belongs to, as shown on the Definitions tab.
Started	The time and date this Activity was started.

18.4.2. Placeholder

This is placeholder text.

18.5. Reactor Process Reports

18.5.1. Overview

Reactor contains a reporting feature based on JasperReports. For more information on how to generate custom reports for Reactor visit the JasperReports (http://jasperreports.sourceforge.net/) site at SourceForge.

18.5.2. Standard Reports

There are five standard reports that come with the Reactor Process Manager.

Report	Purpose/Features
System Activity Summary	Summary of total number of Processes that were started, currently running, completed and the average running duration.

System Activity by Process Report	Summary of total number of Processes that were started, completed and the average running duration by a particular Process Owner.
System Activity by User Report	Summary of total number of Processes that were started, currently running, completed and the average running duration by a particular Process Owner.
System Activity by Date Report	Chart showing the total number of Processes that were started, currently running and completed and the date interval it was started.
Process Aging Report	Chart showing the total number of Processes that are currently running and the date interval it was started.

18.6. Workflow Process Examples

18.6.1. Studio Tutorial

The Reactor Studio tool comes with a step-by-step online tutorial that can be accessed via the Help menu. The tutorial walks you through the process of creating a simple process called Sample Process. This process is started by a human user from the Process Manager in order to ask another user to do something. The assignee can accept or reject the request. At any time, either user can check the progress of the Process in Process Manager.

Use the tutorial to build your own version of Sample Process, and check the completed example as you go. The tutorial teaches you how to:

- Start Studio, edit the Process Label, and add one Activity -- *Request*.
- Save the Process and upload it to the Reactor Engine.
- Start Process Manager in your browser and log in.
- Locate the Process you created and start it.
- See Request on the Activity List and accept it.

The tutorial continues with steps to create more Activities, assign Roles, define Statuses and even demonstrates how to use Operands to collect text that a user will enter in a text box on a Form shown in Process Manager.

The tutorial also explains how to set up Operands and Policies to implement automatic email notifications, so that the Reactor Engine sends the requester and the assignee email notices when certain conditions are met.

18.6.2. Sample Processes

Reactor comes with a collection of sample Processes. These sample Processes can be found in the `C:\Reactor5.5.3-JBoss3.2.1/studio/examples` directory. They can be easily accessed via the File Open command on the Studio menu bar.

19. Reactor 6.0 Programming

This tab covers important information for Developers who want to embed Reactor into their software or simply want to extend the capabilities of Reactor.

Object Model
Explains and provides examples of how the four different types of Process Objects are used within Reactor to model a business Process workflow.

Events And Policy Execution
This section covers the events that trigger an action by the Reactor Engine. It explains how to create and compile Policies, which represent actions taken by Reactor.

Reactor Clients
This section covers the development of Reactor client applications. Topics include how to use the Process Manager Framework, writing servlet code and JSP pages and building a web application.

Reactor Requests
This section covers the 16 different types of requests that can be made to the Reactor Engine in detail. Also provided are Java code examples of these requests. This section also covers the Reactor Java API in detail.

20. Object Model

20.1. Reactor Object Model

Business processes are modeled in Reactor with four different types of objects: Process, Operand, Status, and Policy.

Process objects are the primary building blocks of a model. They contain information about the current state and status of the process, the conditions under which it should start, stop, and change state, and the relationships the Process has with other objects. Most non-trivial business processes consist of multiple steps (Subprocesses) that can often themselves be broken down into smaller, simpler steps. To model such processes, Process objects are associated with an arbitrary number of Subprocesses, which are themselves Process objects,

and can be broken down recursively into other Subprocesses. Process objects represent both process definitions ("definitions") and enactments of a process definition ("instances"). The only difference between a Process object that is an instance and one that is a definition is that the attribute "definition" is true for definitions and false for instances. Also, some attributes are only relevant for instances (current statuses, start time, and end time.) Instances are cloned from definitions by issuing a CloneInstance command to the Reactor system.

Operand objects (called Process Data in the Studio tool) encapsulate arbitrary data that is relevant to the business process modeled by a Process. Examples include a document URL or a purchase order number.

Status objects are very simple objects used to trigger conditions in Process objects or cause the execution of Policy objects, as well as to generally provide information about how a process is progressing.

Policy objects encapsulate arbitrary logic that is relevant to the process being modeled. They allow processes to be custom scripted and are extremely flexible. Policies associate a Java class or policy execution service script with an event that should trigger its execution.

Note the careful separation between the process data and logic and the business data and logic. Process objects represent only the abstract process data and logic. Operands represent the business process data and policies represent the business process logic. Statuses are the connection between the process logic and business logic.

These are some common tasks in the lifecycle of a process:

Author
- creates a process definition
- queries process definitions
- updates a process definition

Invoker
- clones a process definition, creating an instance
- sets operands values in a process instance
- starts a process instance

Participant
- queries processes
- queries a process tree
- sets operand values in a process instance
- adds current status to a process instance
- stops a process instance

20.2. Label Paths

20.2.1. Referring to Specific Objects Using The Label Path

Each object stored by Reactor has a globally unique identifier which is a string of characters that combines hexadecimal digits (0-9, a-f), colons (:), and dashes (-). These IDs are used by an object to identify other objects that are associated with it. For example, Process uses object IDs to list its associated Operands. Requests to Reactor can use object IDs to specify objects.

In some cases, the exact ID of the object might not be known by something that needs to refer to the object. A Policy may need to set the value of an Operand, but the ID of that Operand is different for each instance of the associated Process. That Policy cannot identify the Operand using an ID. When an application creates a new object, Reactor assigns it a unique ID. The application needs to use its own object IDs internally before sending the creation request to Reactor, but those internal Ids are not guaranteed to be globally unique. Applications creating new objects may need a different way to refer to specific objects.

The "label path" is an alternate way to refer to an object when using the object ID is not feasible. A label path contains a sequence of strings, where each string is the label of a parent or ancestor of the object, and the last label belongs to the object itself. A label path can include the ID of an object from which to start the path. If no ID is included, then the first label must refer to an object that has no parent Process or associated Process.

20.2.2. Syntax Examples

The string representation of a label path looks the example below.

```
operand://-2f4d386d:2c7568%3Ae8571b9182:-7cf4/Process3/Operand7
```

The EBNF definition of the string label path syntax is shown below.

```
label_path      := type ':' path
path            := absolute_path | relative_path
absolute_path   := [ '//' ] '/' labels
relative_path   := '//' id '/' relative_labels
```

```
relative_labels := dots | labels | dots '/' labels

dots            := '.' | dotdots | '.' '/' dotdots

dotdots         := '..' | '..' '/' dotdots

type            := 'def' | 'inst' | 'process' | 'operand' | 'status' |
'policy' | 'object'

labels          :=  "[-,:,%,a-z,A-Z,0-9]+"
```

Paths starting with an object that has no parent or associated process can have either of these formats.

```
<TYPE>:/<ABSOLUTE_PATH>

<TYPE>:///<ABSOLUTE_PATH>
```

Paths relative to an object ID can have either of these formats.

```
<TYPE>://<ID>/<RELATIVE_PATH>

<TYPE>://<ID>/
```

20.2.3. Object Type Codes

The following table shows the valid <TYPE> values and their meanings.

Value	Meaning
def	object must be definition or associated with definition
inst	object must be instance or associated with instance
process	object must be either definition or instance
operand	object must be an operand
status	object must be a status

policy	object must be a policy
object	object can be any type

The <ID> is an object's unique ID. Note that this can not contain "/".

An <ABSOLUTE_PATH> is a list of labels, separated by "/". The first label specifies an object that has no parent or associated process. The last label is the label of the object being specified by the whole path.

A <RELATIVE_PATH> is like the absolute path, except that the labels are optional, and may be preceded by an optional list of "." and ".." elements. There can be at most one "." element at the beginning. There can be any number of ".." elements immediately preceding the labels. The "." makes the labels relative to the process or associated process of the object with the specified ID. If the ID specifies a process, then the "." refers to that process. Otherwise, the "." refers to the process associated with that object. The first ".." makes the labels relative to the parent of the process specified or associated with the ID. Each following ".." makes the labels relative to the parent one more level up.

20.2.4. XML Representation Of A Label Path

The XML representation of a label path looks like the following example.

```
<reactor_common:label_path type="operand">
                <reactor_common:root>
                                -2f4d386d%3A2c7568%3Ae8571b9182%3A-7cf4
                </reactor_common:root>
                <reactor_common:label>
                                Process3
                </reactor_common:label>
                <reactor_common:label>
                                Operand7
                </reactor_common:label>
</reactor_common:label_path>
```

This is the DTD fragment for the label path format, including "xmlns" for namespaces.

```
<!ELEMENT reactor_common:label (#PCDATA)>
<!ATTLIST reactor_common:label
       xmlns CDATA #FIXED "reactor_common.dtd">
               <!--
               The root is used when a label path starts from a particular
               object. It contains the ID of a Reactor object.
               -->
<!ELEMENT reactor_common:root (#PCDATA)>
<!ATTLIST reactor_common:root
                 xmlns CDATA #FIXED "reactor_common.dtd">
               <!--
               The <label> elements in a <label_path> are ordered.  The
order of
               <label> elements represents the order of traversal down the
object
               tree from parent to child. If the object is at the root
level,
               meaning that it has no parents, then the label path
contains only
               the label of the object.  Otherwise, the label path starts
with a
               top-level process and follows the chain of child processes
down to
               the process containing the object.  The last <label>
element in a
               <label_path> is the <label> of the object to which the
<label_path>
               resolves. If specified, the type must be process, operand,
```

```
status,
                policy, or object.
                -->
 <!ELEMENT reactor_common:label_path
                ((reactor_common:root,reactor_common:label*)
                | (reactor_common:label+))>
 <!ATTLIST reactor_common:label_path
                type (process | operand | status | policy | object)
#IMPLIED
                xmlns CDATA #FIXED "reactor_common.dtd"
                xmlns:reactor_common CDATA #FIXED "reactor_common.dtd">
```

20.3. Object Model - Processes

20.3.1. Overview

Each Process object is associated with an arbitrary number of other Process objects, which are Subprocesses. Thus, the objects form a tree where each Process object is associated with a node in the tree and no two Process objects are associated with the same node. The nodes associated with the Subprocesses are child nodes of the node associated with the parent process. Each Operand, Status, and Policy object is also associated with a node in the tree, but multiple Operands, Statuses or Policies can be associated with the same node. There is no Process associated with the root node of the Reactor data, but the root node may be associated with "global" Operands, Statuses, and Policies.

Each Process has the following attributes and associations:

Attribute	Description
ID	A String that uniquely identifies an object within the Reactor system.
label	Labels are short, human-readable strings that can be used as an alternative to ids to identify a Process. All the objects associated with a Process should have unique labels.
description	Descriptions are human readable text describing a Process. They are not used by internal

	Reactor functionality, but are included for use by Reactor clients.
ACL (set of ACEs)	The relationships between the Reactor objects and the users of the system are captured using ACLs, which are composed of ACEs (Access Control Elements). **User** *A single person known to the directory and authentication services.* **Group** *A collection of users defined in the directory service.* **Title** *A string associated with one or more users in the directory service (functionally similar to a group). E.g. CEO, CTO, VP Sales, Director of Marketing.* **Role** *A string associated with a collection of R5 permissions. This string describes the relationship between a user and an object. There is one set of roles for the Reactor system. This set is completely customizable.* **ACL (Access Control List)** *A set of mappings (ACEs- Access Control Elements) that determines which sets of users, groups, and titles have which roles with respect to a given object.*
is definition	A boolean (true or false) indicating whether the object represents a Process definition (true) or a Process instance (false).
state	State can be unstarted, started or finished. See the next section 'states and current statuses' for details.
start date	If a process restarts (iterates), then the start date holds the most recent one.
end date	If a process restarts (iterates), then the end date hold the most recent one.
preconditions	The preconditions of a Process dictate the circumstances under which it should be started automatically. Specifically, one of the conditions specified by the preconditions attribute must be false and become true in order for the Process

	to be started automatically.
change conditions	The change conditions of a Process dictate the circumstances under which the Process's state or current statuses should change. A change condition for a Process is composed of a condition that must be met and a change that should be applied to the Process when the condition is met. The condition specified by a change condition must be false and become true for the change to be applied automatically. changeConditions is a Set of instances of ProcessChangeCondition.
timers	An ordered list of timers (TimerSpec objects) that will be started in the Reactor Timer Service when the Process starts and deleted when the Process stops. The ordering of the List is used to match up TimerSpecs with the timer IDs of the started timers. The ids for the timers are stored in the timerIds List such that the first id in the TimerIds list corresponds to the first Timer in the timers list, and so on. The timerIds list is populated at the time the Process is started. Thereafter, the lengths of the timers and timerIds Lists should always be equal.

Associations	Description
parent process	The ReactorObjectId of the Superprocess of this Process.
Subprocesses	The set of ReactorObjectIds of the Subprocesses of this Process.
operands	The set of ReactorObjectIds of the Operands associated with this Process.
statuses	The set of ReactorObjectIds of the Statuses associated with this Process.
current statuses	The current statuses attribute of a Process object indicates the Statuses that currently apply to the Process. It is not used by Process definitions, only by Process instances. A Process can have any number of current statuses at a given time. Reactor adds or removes Statuses to the Process's set of current

	statuses in response to an 'add status' or 'remove status' command or by application of the Process's change conditions.
policies	The set of ReactorObjectIds of the Policies associated with this Process.

20.3.2. States And Current Statuses

The state attribute of a Process object holds the information on the current state of the process. It is not used by Process definitions, only by instances. Process instances can be in one of three states: unstarted, started, or finished. When an instance is cloned from a definition, it is placed in the unstarted state. It remains in the unstarted state until either a Start command is issued to start it, or until its precondition becomes true and it is started automatically. It then transitions to the started state. It remains in the started state until either a Stop command is issued to stop it or one of its change conditions becomes true and the change dictated by that change condition indicates that the process should be stopped. It then transitions to the finished state. From the finished state, an instance can be restarted (transition back to the started state) in the same way that it transitions from the unstarted state to the started state, by a Start command or by its precondition becoming true. A Process will also transition to the finished state from either of the other states if its parent process transitions to the finished state. State transitions can also occur as a result of application of the Process's change conditions.

The "current statuses" attribute of a Process object indicates the statuses that currently apply to the Process. It is not used by Process definitions, only by instances. A Process can have any number of current statuses at a given time. Statuses can be added or removed to the Process's set of current statuses by issuing an AddStatus or RemoveStatus command to the Reactor system or by application of the Process's change conditions.

20.3.3. Pre-Conditions And Change Conditions

The basic relationships between Processes are captured in their precondition and change conditions attributes. As implied by the name, the precondition of a Process dictates the circumstances under which it should be started automatically. Specifically, the condition specified by the precondition attribute must be false and become true in order for the Process to be started automatically. The change conditions of a Process dictate the circumstances under which the Process's state or current statuses should change. A change condition for a Process is composed of a condition that must be met and a change that should be applied to the Process when the condition is met. The condition specified by a change condition must be false and become true for the change to be applied automatically.

There are two types of simple conditions that can be used in preconditions and change conditions: ProcessStateEquals and ProcessHasStatus. A ProcessStateEquals condition specifies a Process object and one of the three possible states of that Process (unstarted, started, or finished). It evaluates to true if the state of a Process specified is equal to the state specified. A ProcessHasStatus condition specifies a Process object and a Status object. It evaluates to true if the Status specified appears in the set of current Statuses of the Process specified.

The conditions used in preconditions and change conditions may also be compound conditions. There are three types of compound conditions: conjunctions, disjunctions, and inversions. A conjunction specifies a set of other conditions and evaluates to true if and only if all of the conditions in the specified set evaluate to true. A disjunction specifies a set of other conditions and evaluates to true if and only if one or more of the conditions in the specified set evaluate to true. The conditions in the set of a conjunction or disjunction may either be simple conditions or compound conditions. An inversion specifies another condition, and evaluates to true if and only if the condition specified evaluates to false. The condition specified by an inversion may be either a simple condition or a compound condition.

There are three types of changes that can be used in change conditions: ProcessStateChange, ProcessStatusAddition, and ProcessStatusRemoval. A ProcessStateChange specifies the new state that a Process object will have after the change has been applied. ProcessStatusAddition and ProcessStatusRemoval changes specify the status that should be added or removed respectively from the Process's set of current statuses when the change is applied.

When the state or set of current statuses of a Process instance changes, the preconditions or change conditions of the changed Process or other Processes may become true, which triggers further changes, which may make other preconditions or change conditions true. This cascade of changes continues recursively. This is how Processes are automated by the Reactor system. Changing one Process automatically causes other Processes to change and fire events that can trigger Policy execution or external actions.

Specifically, when a Process's state or current status changes, the following conditions are evaluated (in order) and if they have been made true by the change, the appropriate resulting change is applied.

1. The preconditions of the Subprocesses of the changee.
2. The preconditions of the peer processes of the changee (processes with the same superprocess as the changee).
3. The change conditions of the changee.
4. The preconditions of the changee.
5. The change conditions of the superprocess of the changee.

20.3.4. Timers

Reactor provides the ability to schedule actions for some point in the future. This is accomplished by setting timers that expire at some point in the future. When a timer expires, it fires an event, which can trigger policies.

A common use for timers is to manage due dates for tasks. For example, suppose that if a Process object has not finished after two days, some action should be taken. A good way to accomplish this is to associate a policy with the "ProcessStarted" event for the Process that sets the timer with a specific event type. Then associate a policy with the timer expiration event that takes the necessary action.

Each Timer has the attributes shown in the table below.

Attribute	Description
schedule date	protected java.util.Date scheduleDate
schedule expression	protected java.lang.String scheduleExpression
repeat expression	protected java.lang.String repeatExpression
repeat count	protected int repeatCount
end date	protected java.util.Date endDate
end expression	protected java.lang.String endExpression
calendar	protected java.lang.String calendar
description	protected java.lang.String description
event	ReactorEvents are used by the Reactor System to trigger Reactor Policy execution. A single Event can trigger the execution of multiple Reactor Policy objects. And a single Reactor Policy can be triggered by different kinds of events. ReactorEvents are characterized by their type (a.k.a. name) (e.g. "ProcessStateChange"), their source (e.g. "ProcessCommandService", "TopLevelProcessX.SubProc1", etc.), and their attributes, which are stored in a Map. ReactorEvents are sent from various parts of the Reactor system to the Policy Execution Service, which decides which Policy objects should be executed. Events can also optionally be scoped to the ancestor nodes of a given node, called the

	scopeNodeId. In this case, only policies associated with the ancestor nodes of the scopeNodeId are eligible to be started as a result of this event. ReactorEvent objects are immutable.

The event can contain arbitrary key/value pairs.

20.4. Object Model - Operands

Each Operand has the following attributes and associations:

Attribute	Description
ID	A String that uniquely identifies an object within the Reactor system.
label	Labels are short, human-readable strings that can be used as an alternative to ids to identify an object. All the objects associated with a Process should have unique labels.
description	Descriptions are human readable text describing an object. They are not used by internal Reactor functionality, but are included for use by Reactor clients.
ACL (set of ACEs)	An ACL (Access Control List) defines the relationships between a Reactor object and the users of the system. An ACE is composed of ACEs (Access Control Elements).
visible in subtree	A boolean (true or false) that defines the scope of the Operand.
value	The data stored in the Operand.

Association	Description
process	The id of the Process with which this Operand is associated.

If an Operand has its "visible in subtree" attribute set to "true", then the Operand's value can be accessed by Subprocesses of the process associated with the Operand. Otherwise, only the Process associated with the Operand can access the Operand's value.

Operands can store any value that can be expressed as a string. Objects or complicated data

structures can be stored by converting to an XML representation or using Java serialization. However, this is not recommended. Instead, the recommended approach is to store the data in another system, such as a database, and store a handle or reference ID in the operand value itself.

20.5. Object Model - Statuses

Each Status has the following attributes and associations:

Attribute	Description
ID	A String that uniquely identifies an object within the Reactor system.
label	Labels are short, human-readable strings that can be used as an alternative to IDs to identify an object. All the objects associated with a Process should have unique labels.
description	Descriptions are human readable text describing an object. They are not used by internal Reactor functionality, but are included for use by Reactor clients.
ACL	An ACL (Access Control List) defines the relationships between a Reactor object and the users of the system. An ACE is composed of ACEs (Access Control Elements).
available in subtree	A boolean (true, False) that defines the scope of the Status.

Association	Description
process	The id of the Process with which this Status is associated.

If a Status has its "available in subtree" attribute set to "true", then the Status can be added to Subprocesses of the process associated with the Status. Otherwise, the Status can only be added to its associated Process. Statuses can be used for a variety of purposes. The most common usage is to use statuses in preconditions, for specifying transitions between activities. Statuses can also be used to trigger policy execution. This can be independent of any process transition. Finally, statuses can be used as markers that provide meaning when they are queried or displayed by client applications.

20.6. Object Model - Policies

Each Policy has the following attributes and associations:

Attribute	Description
ID	A String that uniquely identifies an object within the Reactor system.
label	Labels are short, human-readable strings that can be used as an alternative to IDs to identify an object. All the objects associated with a Process should have unique labels.
description	Descriptions are human readable text describing an object. They are not used by internal Reactor functionality, but are included for use by Reactor clients.
ACL	An ACL (Access Control List) defines the relationships between a Reactor object and the users of the system. An ACE is composed of ACEs (Access Control Elements).
event source	The event source a ReactorEvent must have to trigger the execution of this Policy. A null indicates that any event source may trigger the execution of this Policy.
event name	The event name (a.k.a. event type) a ReactorEvent must have to trigger the execution of this Policy. A null indicates that an event of any type may trigger the execution of this Policy.
event attributes	The event name (a.k.a. event type) a ReactorEvent must have to trigger the execution of this Policy.
language	The language of this Policy (only used with BSF Policies).
source code type	Indicates whether the String sourceCode attribute contains the classname, url or raw source for the executable code of this Policy.
source code	Specifies the classname, URL, or raw source for the executable code of this Policy, as specified by the sourceCodeType attribute.

security policy	Specifies the name of the security policy to enforce on the execution of this Policy.

Association	Description
process	The id of the Process with which this Status is associated.

The event source, event name, and event attributes combine to describe the event that triggers the policy execution. The language, source code type, and source code specify the action to be taken by the policy.

Each Policy consists of one of two types of executable instructions that are executed when the Policy is triggered: Java classes and BSF scripts. The source code type attribute of a Policy attribute indicates the type of the Policy and how to access the executable instructions so that they can be executed. The source code type must be one of the following three values:

SourceCode
indicates that the executable instruction is a Policy execution script.
Classname
indicates that the executable instruction is a Java class.
URL
indicates that the executable instruction is a Policy execution script.

If the source code type is **SourceCode**, the contents of the source code attribute of the Policy are interpreted as raw source code of the language specified by the language attribute of the Policy. This source code is sent to the Policy execution service.

If the source code type is **Classname**, the contents of the source code attribute of the Policy are assumed to be a fully qualified Java class name. It attempts to load the class from the class path of the application server and execute an instance of it. The class must implement the `java.lang.Runnable` interface to be executed correctly.

If the source code type is **URL**, the contents of the source code attribute of the Policy is assumed to be a URL that can be used to retrieve the raw source code of the language specified by the language attribute of the policy. If no language is specified by the language attribute of the policy, the Reactor system attempts to determine the language from the file extension of the URL. The source code is sent to the Policy execution service.

20.7. Object Model - Actors

Each Reactor object of the four primary types has an Access Control List (ACL) containing Actors. An Actor represents a person or group that has some role pertaining to the Reactor object. The person or group can be defined explicitly by providing a name, or implicitly by

specifying the title of a role to be resolved by the enterprise directory service.

Each Actor has the attributes shown in the table below.

Attribute	Description
Name	The name of the user, organizational title, or group.
Type	This indicates whether the Actor being referred to is a user, title, or group.
Role	This defines the Actor's relationship relative to the Reactor object.

For example, suppose a Reactor system is configured with two roles: "owner" and "assignee". The "assignee" role may only be associated with the permissions to add or remove a current status or stop Process objects. The "owner" role may be associated with all permissions. In this situation, one can edit the ACL of a particular object to specify which users, groups, and titles have "owner" permissions and which users, groups, and titles have "assignee" permissions with respect to that particular object.

21. Events And Policy Execution

21.1. Events

The execution of a Policy is triggered in response to the firing of an event. There are three basic sources of events. Some events are triggered by changes to a process, either a change in state or a change in current status. Other events are triggered by a Process's timers. External systems can also send events to Reactor.

Each event consists of an event type, an event source, a table of event attributes, and optionally a Process ID that determines the scope of the event. If the scope Process ID is included, only Policies associated with the Process specified or its ancestor Processes or Policies in the root node may be triggered by the event. The attributes in the table are specific to the type of event. For example, the attributes of a "ProcessFinished" event contains values for the following keys: "ProcessID" and "ProcessLabelPath"; the attributes of a "StatusAddition" event contains values for the following keys: "ProcessID", "ProcessLabelPath", "StatusID", and "StatusLabelPath".

Future, external systems will be able to register with Reactor to be notified of events.

Events only trigger the execution of Policies that match the event. The attributes of a Policy determine which events the Policy matches. A Policy can specify the type, source, and/or

attributes of the events that should trigger it. For example, if a Policy specifies a type of "ProcessStarted" and an attribute value of "foo.bar" for the key "ProcessLabelPath", any event of type "ProcessStarted" with that attribute in its list of attributes will trigger the execution of that Policy, regardless of the source of the event or the other attributes of the event.

21.1.1. Process Change Events

The source of process change events is "ProcessCommandService", an internal Reactor service. These are the different event names sent by the Process Command Service:

21.1.1.1. ProcessStarted

Property	Description
ProcessID	The ID of the started process.
ProcessLabelPath	The label path of the started process.

21.1.1.2. ProcessFinished

Property	Description
ProcessID	The ID of the stopped process.
ProcessLabelPath	The label path of the stopped process.

21.1.1.3. StatusAddition

Property	Description
ProcessID	The ID of the process.
ProcessLabelPath	The label path of the process.
StatusID	The ID of the Status to be added.
StatusLabelPath	The label path of the Status to be added.

21.1.1.4. StatusRemoval

Property	Description
ProcessID	The ID of the process.

ProcessLabelPath	The label path of the process.
StatusID	The ID of the Status to be removed.
StatusLabelPath	The label path of the Status to be removed.

21.1.2. Timer Events

The source of timer events is "TimerService", an internal Reactor service. Every timer event has the name "TimerExpired". Properties are specified by the process author who creates the timers, but every timer has a property named "ProcessID" which identifies the process containing the timer.

21.2. Policies

Reactor allows policy code to be written in a variety of languages, including but not limited to Java, JavaScript, Tcl, and Python.

21.2.1. Using Precompiled Java Classes

Any Java class that implements the java.lang.Runnable interface or the `com.oakgrovesystems.reactor.PolicyAction` interface can be invoked by a Policy. Specify the class name in the Policy's "source code" attribute. The class must be accessible to the classloader that loads the PolicyExecutionMessageBean in the application server. A good way to make it accessible to this classloader is to place it in the ReactorPolicies.jar file within the Reactor EAR file. Alternatively, some application servers have alternate mechanisms to "hot deploy" classes into their runtime classpaths, which may be preferable to modifying the EAR file.

The NavigatorProxy class is useful for accessing Reactor objects from within a Policy. It can also be helpful to have access to the Policy object and the ReactorEvent that triggered the Policy execution. These objects are made available to classes that implement the PolicyAction interface rather than just the Runnable interface. Prior to calling the run() method of a PolicyAction, Reactor calls its setTriggeringEvent(), setPolicy(), and setNavigator() methods, which allows the class to store references to those three objects in instance variables for subsequent use by the run() method. For more details, see the Javadoc API for the `com.oakgrovesystems.reactor.PolicyAction` class.

21.2.2. Using Uncompiled Source Code via BSF

Reactor can use IBM's Bean Scripting Framework (BSF) to execute policies. This allows policy code to be written in a variety of languages, including but not limited to Java,

JavaScript, Tcl, and Python.

21.2.3. Using Java Source Code

Set the Policy's source type to "Source Code", and its source language to "Java". Put valid Java source code into the Policy's source code attribute. This code will be inserted into a Java method. The return value from this method will be ignored, so just return null. The method type is not void, so always include the return value.

Instead of making the source code an attribute of the Policy object, one could instead place the source code elsewhere (on the network or file system, for example) and simply provide the URL of the source code to the Policy. In this case, set the Policy's source type to "URL" and set the Policy source code attribute to be the URL. The code will be retrieved from the URL and placed into a Java method. The code should return null; as described above.

The NavigatorProxy class is useful for accessing Reactor objects from within a Policy. Use this code to get a NavigatorProxy object from a Java Policy:

```
NavigatorProxy proxy = (NavigatorProxy) bsf.lookupBean("proxy");
```

For more details, see the Javadoc API for the NavigatorProxy class in the `com.oakgrovesystems.reactor.client` package.

21.2.4. Using Other Languages

BSF allows other languages to access the NavigatorProxy object, even though it is a Java object. See the BSF documentation for more information. BSF was originally developed at IBM.

Note:

For more info see the IBM BSF web site (http://www-124.ibm.com/developerworks/projects/bsf/) .
Future versions of BSF will be released through the Apache Jakarta BSF project (http://jakarta.apache.org/bsf/)

21.2.5. Writing Reusable Policies

The first step to making a Policy reusable is to avoid hard-coding any data into the script. Put data in Operands. The Reactor Studio provides a mechanism to add new Policy templates. A Policy template (called a "parameterized Policy") is made available to Studio by creating an

XML file in the ..\studio\actions directory of the Studio distribution. See that directory for examples of XML files with the correct format.

22. Reactor Clients

22.1. Reactor Clients

22.1.1. Overview

There are currently three interfaces to communicate with the Reactor Server: calling EJB Session Bean methods, sending SOAP messages via HTTP, and sending Reactor XML via HTTP. There will soon be facilities for sending JMS messages to the Reactor Engine. Until then, application developers can create custom message-driven EJBs to receive JMS messages and send them to the Reactor Engine through existing interfaces.

Regardless of the underlying mechanism, the concepts used by all the interfaces are the same. The application creates a request with a certain type, adds an authentication token, possibly adds some parameters, and then sends the request to the Reactor Engine. Reactor then sends back a response containing a result code and possibly some return values. The result code contains a numeric code indicating success or failure, a corresponding text message meant to be read by users, and possibly some information that could help with debugging if something went wrong. The authentication token is obtained at the beginning of a session by providing the proper credentials in a Login request (which does not require a valid authentication token.) The result codes are shown in the next section covering authentication.

Java developers can also use the client API which provides shortcut methods for sending requests and managing objects.

22.1.2. Authentication

Except for the Login request, each request requires a valid authentication token. This authentication token is acquired by sending a Login request with a valid username and password. The response, if successful, contains an authentication token must be used by subsequent requests. The token serves to identify the person or entity making the request. Requests with an invalid token are denied.

An explanation of the result codes generated by Reactor is provided in the table shown in the next section.

22.1.3. Result Codes

Code	Explanation

200	OK
400	Bad Request
404	Bad Parameters
405	Bad Request Type
406	Cannot Deliver Format
407	Could Not Parse Request
408	Bad Response
410	Unauthorized
411	Login Failed
412	Could Not Authenticate Due To Server Configuration
413	Rejected By ACL
414	Duplicate Object
420	Not Found
421	Invalid License
500	Internal Error
501	Not Implemented
502	Service Not Initialized
503	Service Temporarily Unavailable
504	Service Not Found
505	Failed Assertion
506	Version Not Supported

22.2. Creating A Reactor Clients

Programmers can create their own Reactor Client for interface with the Engine.

22.2.1. Writing Client Code

The com.oakgrovesystems.reactor.client.ReactorProxy interface provides methods for interacting with Reactor Engine. The com.oakgrovesystems.reactor.client package also includes two concrete implementations of the ReactorProxy interface, XMLReactorProxy and EJBReactorProxy. The XMLReactorProxy class uses XML/HTTP to communicate with Reactor Engine. The EJBReactorProxy class communicates with an EJB stateless session bean to provide the same functionality. The EJBReactorProxy in most cases has better performance, but the XMLReactorProxy class makes it easier to communicate through a firewall.

The ReactorObjectId and LabelPath classes are used to identify objects. They are found in the com.oakgroveystems.reactor package. The ReactorObjects class, also found in com.oakgroveystems.reactor, provides methods for storing and accessing objects that have been queried from Reactor Engine. The QueryCriteria class, found in the com.oakgrovesystems.reactor.client package, is used to create queries for getting objects from Reactor Engine. The objects themselves are described in the com.oakgrovesystems.reactor.processMediation package. The four main classes are Process, Operand, Status, and Policy. The com.oakgrovesystems.processMediation.xml package contains builder and outputter classes which convert objects to XML representations and back to objects. For details, see section Chapter 0 "Reactor Java API" in this document, or the JavaDoc HTML in the Reactor distribution in this location:

```
docs/api/index.html
```

22.2.2. Building A Client Application

The file Reactor-5.0.jar contains classes necessary to compile an application that connects to the Reactor Engine. It is located in the share directory in the Reactor Engine distribution.

22.2.3. Running A Client Application

In addition to Reactor-5.0.jar, other jar files may be required to run a client application.

If the application connects to Reactor Engine using XMLReactorProxy, then the following jar, distributed with Reactor Engine, must be included in the classpath:

- `share/jdom.jar`

If the application connects to Reactor Engine using EJBReactorProxy, then the EJB and JNDI jar files must be present. Each application server may have its own EJB and JNDI client jar files. The following jar files will work with the distribution for the JBoss application server:

- `share/ejb.jar`
- `share/jndi.jar`

- jboss321/client/jnp-client.jar
- jboss321/client/jboss-client.jar
- jboss321/client/jbosssx-client.jar

In addition to having the EJB and JNDI jar files, clients using the EJB interface will need to configure JNDI properly. In general, this will be done by having a jndi.properties file in the runtime classpath of the application. The file jboss321/bin/jndi.properties, in the distribution for the JBoss application server, can be copied for use in client applications.

22.2.4. Sample Client Applications

The Reactor Engine distribution contains examples of client applications. See the samples/client directory for code, build scripts, and launch scripts.

Windows
use `build.bat` and `run.bat`.
UNIX
type `/bin/sh build.sh` and `/bin/sh run.sh` to use the build and launch Bourne shell scripts.

The com.oakgrovesystems.Process Manager.action package in the Reactor Process Manager Framework is code that is intended to be executed in a servlet, but it uses the ReactorProxy interface like a client application. The source code is available in the Reactor Engine distribution under the Process Manager directory.

22.3. Creating Web Applications

22.3.1. Creating Reactor Web Applications

22.3.1.1. Application Framework

The Reactor Application Framework is a collection of patterns, conventions, and Java classes for building web applications that communicate with Reactor Engine. A web application built with the Reactor Application Framework consists of JSP pages, Java servlets, and a `web.xml` configuration file. Some web applications may include additional Java classes, HTML files, and images.

The JSP pages use HTML, JSP tags, and Apache Struts tags to display data passed to the JSP pages through Java Beans. The JSP tags provide access to data that is passed through the session or through the request, forwarded from the servlet. The Apache Struts tags provide the capability to iterate over lists of beans. The Apache Struts tags also provide the capability to conditionally display data, based on the value of bean properties. This makes it possible to have JSP pages with no Java code, suitable for graphic artists to edit the displays with

third-party HTML editing software.

> **Note:**
> More information about Apache Struts is available at the Apache Struts Website. (http://jakarta.apache.org/struts/)

Since the JSP pages do not contain any Java code, all the behavior takes place in servlets. Each servlet delegates most of the work to one or more "action" objects. The servlet passes a ReactorProxy object to the action object, which allows the action object to communicate with the Reactor 5 Server.

When an action object executes, it creates content beans with text properties that are ready to be displayed. The servlet then gets the content beans from the action object and uses the servlet request context to pass them to the JSP page for display.

The source code to the Reactor Process Manager Framework is bundled with the Reactor Engine distribution, in the `\samples\Process Manager\src\framework` directory.

22.3.1.2. Content Beans

The table below displays the content beans and their associated properties from the com.oakgrovesystems.Process Manager.content package.

Content Bean	Properties
OperandContentBean	• label • value • dataType
OperandListBean	• operands (returns a List) • size
ProcessContentBean	• id • idURLEncoded (useful for including in URLs) • labelPath • labelPathURLEncoded • label • description • startDate • state ("unstarted", "started", "finished") • parentLabel • title • lockState ("", "eligible", "acquired", "standby")
ProcessListBean	• processList (returns a List) • size

StatusContentBean	• id • idURLEncoded (useful for including in URLs) • label • description
StatusListBean	• statusList (returns a List) • size
ValueBean	• value (used for storing simple strings)

The `com.oakgrovesystems.Process Manager.content` package also includes the *ProcessTemplate* class. This has a static method *toString* (String, Process, ReactorObjects). This takes a template string and fills in values for the process and associated objects.

22.3.1.3. Actions

The `com.oakgrovesystems.Process Manager.action` package contains the following action objects:

- AcquireActivity
- ApplyLogin
- ApplyStartForm
- ReleaseActivity
- ShowActivity
- ShowActivityList
- ShowAttachment
- ShowCompletionForm
- ShowProcessList
- ShowStartForm

Each action must be initialized before it can be executed. First, the setServerProxy() method passes a ReactorProxy object to the action, so it can communicate with the Reactor Engine. The setAuthentication() method passes an Authentication object to the action, which is used to get the identity of the user who initiated the action. The setParameters() method passes a Map object with key/value pairs obtained by calling the Servlet Facade's parseParameters() method.

After initialization, one calls the execute() method which does all the work. This can throw a Process ManagerException object if anything goes wrong. If the action succeeds, it might set properties that can be accessed with "get" methods particular to a particular action class. An appropriate use of these properties would be to indicate which type of display should be used to present the contents generated by the action. The contents themselves are accessed through the getContent() method of the action object. The Process ManagerServlet base class

provides a convenience method, setContent(), for copying beans from the action's content to the servlet's request context.

Once the content beans are placed in the servlet's request context, the servlet should forward the display to the appropriate JSP page. In cases where the servlet is processing a submitted form, often it is appropriate to redirect to some other page rather than forward to a JSP for display. One such example is a servlet that processes the form to complete an activity. Once the activity is complete, the servlet redirects to a servlet that shows the list of available activities that remain.

AcquireActivity

Use the `AcquireActivity` action when a user wants to lock an activity, so that other users can no longer participate. The activity's ACL should have an entry where the user holds the "Eligible Participant" role. After executing the AcquireActivity action, all entries with the "Eligible Participant" role will be modified to have the "Standby Participant" role, and the user will be modified to have the "Activity Participant" role. Use the ReleaseActivity action to reverse the effects of an AcquireActivity action. This action takes the ID of a process instance, along with the ID and label of an optional status, and makes an addStatusToProcess() request to the Reactor Engine. If no status is specified, this action makes a stopProcess() request instead of an addStatusToProcess() request.

	Parameters
id	String with the ID of the process instance.
	ReactorProxy Methods Called
Lock()	Attempts to add the Initiative Participant role for the user, and also move Eligible Participant ACEs to have the Standby Participant role, for the selected process instance.
	Content
	None.
	Side Effects
1	The User gains the Initiative Participant role in the Process Instance.

ApplyCompletionForm

Use the ApplyCompletionForm to stop an activity. The status and operands of an activity can be updated as part of completing the activity.

This action takes the ID of a process instance, along with the ID and label of an optional status, and makes an addStatusToProcess() request to Reactor Engine. If no status is specified, this action makes a stopProcess() request instead of an addStatusToProcess() request.

Parameters	
ReactorProxy Methods Called	
Content	
	None.
Side Effects	
1	The process instance is stopped. If a status parameter is present, the specified status is added to the process instance.

ApplyLogin

Use the ApplyLogin action to verify a user's credentials and get an authentication token from the Reactor Engine.

This action takes a username and password, makes an authentication request to Reactor Engine, then saves the authentication token in the servlet session. This is the only action that does not require a valid authentication token to already be in the servlet session.

Parameters	
ReactorProxy Methods Called	
Content	
	None.
Side Effects	

ApplyStartForm

Use the ApplyStartForm action when the user has submitted a form to initiate a process. The start form is typically created by the ShowStartForm action. This form can contain fields with values for operands to be initialized before starting the process. Each field with an operand value must be named "operand:" followed by the label of the operand.

This action takes the "id" parameter and "operand:${LABEL}" parameters and makes cloneInstance(), query(), setObjects(), and startProcess() requests to Reactor Engine.

Parameters	
ReactorProxy Methods Called	
Content	
	None.
Side Effects	
1	A Process Instance is created, configured and started.

ReleaseActivity

The ReleaseActivity action is the opposite of the AcquireRelease action. It modifies an activity's ACL to remove the entry where the user has the "Initiative Participant" role. It also modifies all entries with the "Standby Participant" role to have the "Eligible Participant" role.

This action takes the ID of a process instance and makes an unlock() request to Reactor Engine, using the username from the Authentication object.

Parameters	
ReactorProxy Methods Called	
Content	
	None.
Side Effects	
1	The Process Instance no longer has the User as an active participant.

ShowActivity

Use the ShowActivity action to display the details of an activity that has been assigned to the user.

This action takes the ID of a process instance, makes a query() request to Reactor Engine, then stores a ProcessContentBean with the activity details, an OperandListBean with the

operand values, and a StatusListBean with the possible statuses that could be used to complete the activity. If the process metadata has a custom activity template, then the template is filled in and stored in a ValueBean.

Parameters
ReactorProxy Methods Called
Content
None.
Side Effects

ShowActivityList

Use the ShowActivityList action to display all activities either that are assigned to the user, or can be acquired by the user. To show the details of a particular activity, use the ShowActivity action.

This action uses the username from the Authentication object to create a ProcessListBean with all the activities in which the user has "Eligible Participant" or "Initiative Participant" roles.

Parameters
None.
ReactorProxy Methods Called
Content
None.
Side Effects

ShowAttachment

Use the ShowAttachment action to retrieve the contents of a "File" operand which has been stored in a document repository using the AttachmentModule.

This action takes the ID of a document which has been stored as an attachment, uses the configured AttachmentModule to get metadata and contents of the document, then sends the contents of the document to through the servlet's output stream to the web browser. Since this

class uses the servlet's output stream directly, setServlet() must be called before execute().

The ServletFacade class provides the parseParameters() method, which creates a Map object with key/value pairs for each parameter. This method also handles forms with an encoding type of "multipart/form-data". Files that are uploaded as part of multipart forms are converted into Attachment objects, which can then be stored in a document repository using classes in the `com.oakgrovesystems.attachments` package. Attachments can be retrieved by users through servlets that use the ShowAttachment action.

Parameters
ReactorProxy Methods Called
Content
None.
Side Effects

ShowCompletionForm

Use the ShowCompletionForm action to create a form to complete an activity. This provides the content of a form, but does not modify the activity. If the form is empty, discovered by calling the action's isEmptyForm() method after execution, then the activity can be completed immediately using the ApplyCompletionForm action, or by using the ReactorProxy API directly.

This action takes the ID of a process instance, the name of the selected completion status, and the IDs of possible completion statuses. It makes a query() request to Reactor Engine, gets the process instance and its associated operands, and stores a ProcessContentBean and the selected completion status ID and label. If the process instance has a custom completion form defined in its metadata, then a ValueBean is set with the completion form. Otherwise, an OperandListBean is added with some of the operands. Operands will not be included unless their metadata specifies that they appear in completion forms.

Parameters
ReactorProxy Methods Called
Content
None.

Side Effects

ShowProcessList

Use the ShowProcessList action to display a list of process definitions. To create and start an instance of a process definition, use the ShowStartForm action.

This action takes no parameters, makes a query() request to the Reactor Engine, then stores a ProcessListBean named "processListBean".

Parameters
None.
ReactorProxy Methods Called
Content
None.
Side Effects

ShowStartForm

Use the ShowStartForm to display a form to configure a process to be started. The ApplyStartForm action will start the process after the form is submitted.

This action takes the "id" parameter, makes a query() request to Reactor Engine, then stores a ProcessContentBean named "processBean", a OperandListBean named "operandListBean". If the process has a custom start form in its "Process Manager: Start Form" metadata, then the a ValueBean named "formContents" is also stored in the content.

Parameters
ReactorProxy Methods Called
Content
None.
Side Effects

22.3.2. Writing Servlet Code

Process Manager servlets should be very simple. Action objects are responsible for business logic, and content beans are responsible for formatting data for display.

Each Process Manager servlet extends the Process ManagerServlet class and implements the service(HttpServletRequest, HttpServletResponse) method. The following import statements are required.

```
import com.oakgrovesystems.Process Manager.Authentication;
import com.oakgrovesystems.Process Manager.Process ManagerException;
import com.oakgrovesystems.Process Manager.Process ManagerServlet;
import com.oakgrovesystems.Process Manager.ServletFacade;
import com.oakgrovesystems.Process Manager.action.*; // select appropriate
actions
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
```

The service() method should create a ServletFacade object for accessing the Java Servlet API. This makes it easier to unit test the Process Manager servlet in isolation, and can also be more convenient.

```
ServletFacade servlet;
servlet = new ServletFacade(request, response, getServletContext());
```

Use the ServletFacade object to get an authentication object from the session. This assumes that there is a login servlet that puts the authentication in a standard location.

```
```

```
String authKey = "reactor.authentication";

Authentication auth;

auth = (Authentication) servlet.getSessionAttribute(authKey);
```

Check that the authentication object exists and is valid. If no valid authentication object is found, then display the login page or an appropriate error. Data in the session may expire if not used for some time, so a missing authentication object is not necessarily an unusual event.

```
String loginFormJSP = "/Login";

if ((auth == null) || (!auth.isValid())) {

servlet.forward(loginFormJSP);

}
```

Once the authentication object is found, get the authentication token and pass it to the ReactorProxy object named "reactorProxy", defined in the Process ManagerServlet class. This ReactorProxy object will be used by the action object to communicate with the Reactor Engine.

```
reactorProxy.setAuthToken(auth.getToken());
```

Create an action object to process parameters and generate content. See Section Chapter 65536 "Actions" for descriptions of action classes that are included with Reactor Process Manager Framework, or create your own action classes.

```
ApplyCompletionForm action;

action = new ApplyCompletionForm();
```

Initialize the action object. This includes passing the ReactorProxy object and the authentication object. If there are parameters, the parameters should be parsed and passed to the action object. It is important that a servlet that parses parameters should not forward to another servlet that also parses parameters! This could result in the second servlet waiting forever to read data from an input stream that has already been completely read.

```
action.setServerProxy(reactorProxy);
action.setAuthentication(auth);
action.setParameters(servlet.parseParameters());
```

Execute the action, and handle any Process ManagerException object which might be thrown.

```
String errorKey = "reactor.error";
String errorJSP = "/Error";
try {
        action.execute();
} catch (Process ManagerException e) {
       servlet.setRequestAttribute(errorKey, e);
       servlet.forward(errorJSP);
       return;
}
```

Choose a JSP page to forward or redirect to for display. Forward to display pages which depend on content beans being passed in the request context. Redirect to servlets or displays that do not depend on any parameters, especially after processing a submitted form. Be careful not to forward to another servlet that processes the input stream from a servlet that also processes the input stream.

```
String activityListRedirect = "ActivityList";
servlet.sendRedirect(activityListRedirect);
```

22.3.3. Writing JSP Pages

Each display should have its own JSP page. The JSP page might use the JSP tag "<%@ include file="..." %>" to include shared elements, such as headers or footers. The beginning of the JSP page should import the appropriate Java classes and set the page's "session" value to "true". The JSP page must also define any tag libraries and declare any beans used in the page.

```
<%@ page language="java"
                session="true"
                import="com.oakgrovesystems.Process
Manager.content.*,java.util.*"
                errorPage="Error.jsp" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld"  prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<jsp:useBean id="processBean" scope="request"
                class="com.oakgrovesystems.Process
Manager.content.ProcessContentBean"/>
<jsp:useBean id="operandListBean" scope="request"
                class="com.oakgrovesystems.Process
Manager.content.OperandListBean"/>
```

To display a value stored in a simple content bean, use the <jsp:getProperty> tag.

```
<jsp:getProperty name="processBean" property="label"/>
```

To iterate through a list of beans, use the Apache Struts tags.

```
    <logic:iterate name="operandListBean"
property="operands"
id="operandBean"
scope="request"
type="com.oakgrovesystems.Process Manager.content.OperandContentBean">
                <li> <bean:write name="operandBean" property="label"/>
        </logic:iterate>
```

To conditionally include HTML depending on the value of a particular bean's property, use the Apache Struts <logic:equal> and <logic:notEqual> tags.

```
<logic:notEqual name="operandBean" property="dataType" value="File">
                <bean:write name="operandBean" property="value"/>
</logic:notEqual>

<logic:equal name="operandBean" property="dataType" value="File">
                <a href="GetAttachment?id=<bean:write name='operandBean'
                        property='value'/>">Attachment</a>
</logic:equal>
```

When the contents of a form might include uploaded files, remember to use the multipart form encoding type.

```
<form action="ApplyForm" method="POST" enctype="multipart/form-data">
```

22.3.4. Configuring And Building A Web Application

22.3.4.1. Configuring A Web Application

Each servlet and JSP page needs to be configured in the servlet's deployment descriptor. The deployment descriptor is an XML document named "web.xml" located in the WEB-INF directory of the web application. This section describes the minimal subset of the deployment descriptor required to deploy a Process Manager web application.

The format of the deployment descriptor is described by a DTD published by Sun Microsystems. A valid servlet deployment descriptor begins like the sample code shown below.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app PUBLIC
               "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
               "http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">
```

The root element of the document is "web-app".

```
<;web-app>;
```

The next group of elements is "servlet" elements. Each one describes either a servlet or a JSP page. Both use the "servlet-name" element to assign a unique name to the servlet. The elements for JSP pages contain a "jsp-file" element, but "servlet-class" elements are used for servlets.

```

```

```
<servlet>
       <servlet-name>ProcessList<servlet-name>
       <servlet-class>activechecklist.ProcessList<servlet-class>
</servlet>

<servlet>
       <servlet-name>ProcessListJSP<servlet-name>
       <jsp-file>/ProcessList.jsp<jsp-file>
</servlet>
```

After adding all of the "servlet" elements, add a "servlet-mapping" element for each "servlet" element. While it might be more convenient to place each "servlet-mapping" element immediately after its corresponding "servlet" element in the file, that would violate the file format described in the DTD. Each "servlet-mapping" element contains a "servlet-name" element using the name assigned previously in the file. Then it contains a "url-pattern" element with a relative path. This assigns a URL to the servlet or JSP page, relative to the root URL of the web application.

```
<servlet-mapping>
       <servlet-name>ProcessList</servlet-name>
       <url-pattern>/ProcessList</url-pattern>
</servlet-mapping>

<servlet-mapping>
       <servlet-name>ProcessListJSP</servlet-name>
       <url-pattern>/ProcessListJSP</url-pattern>
</servlet-mapping>
```

After all the servlet mapping elements, declare any custom tag libraries used by the JSP

pages.

```
<taglib>
       <taglib-uri>/WEB-INF/struts.tld</taglib-uri>
       <taglib-location>/WEB-INF/struts.tld</taglib-location>
</taglib>
```

Close the tag of the root element, and the document is finished.

```
</web-app>
```

22.3.4.2. Building A Web Application

A Process Manager web application consists of HTML pages, images, JSP pages, Servlet classes, JAR files, and a deployment descriptor. To build a web application, create a directory to store all these things.

Copy the HTML pages, images, and JSP pages into the directory. Create a subdirectory named "WEB-INF". Create a "classes" subdirectory of "WEB-INF". Compile the servlet classes, possibly with ancillary supporting classes, and copy them into the `WEB-INF/classes` directory. Make sure that the directory structure under "classes" mirrors the package structure of the compiled classes. For example, the `com.oakgrovesystems.example.SampleServlet` class file would go into the `WEB-INF/classes/com/oakgrovesystems/example/SampleServlet.class` file. If there are JAR files required by the web application that are not in the application server's default classpath, then create a "lib" subdirectory in "WEB-INF". Copy the required JAR files into the "WEB-INF/lib" directory. Copy the "web.xml" deployment descriptor file into the "WEB-INF" directory. Create a WAR file using the same tool that you use to create JAR files, like the *jar* tool that is part of Sun's SDK. Use the `.war` filename extension instead of `.jar`. The parent of the *WEB-INF* directory should not be included in the WAR file. The JSP pages and *WEB-INF* directories should be at the top level of the WAR file's contents.

See your application server's documentation for instructions to deploy a WAR file. To deploy

a WAR file in the JBoss application server that is bundled with some evaluation copies of the Reactor Product Suite, just copy the WAR file into the `jboss321/server/default/deploy` directory.

22.3.4.3. Sample Web Application

The Reactor Engine distribution includes a sample web application in the `..\samples\Process Manager\src\Process Manager\activechecklist` directory. The servlet source code is located under the `src` directory. The JSP pages, HTML, images, and deployment descriptor are under the `webapp` directory. The build scripts use Ant, from the Apache Jakarta Project. The `build.xml` file contains the build configuration, and the `build.bat` and `build.sh` scripts execute the build on Microsoft Windows and UNIX respectively.

> **Note:**
> More information about Ant is available at the Apache Ant web site (http://jakarta.apache.org/ant/)

The build scripts create an `ActiveChecklist.war` file in the `dist` directory.

22.4. SOAP And WebServices

22.4.1. SOAP Messages And Web Services

Some configurations of Reactor Engine require additional actions to deploy the SOAP interface. Administrators should refer to `ReadmeSOAP.html` in the Reactor Engine distribution for more details.

The request and response SOAP messages provide wrappers for the same XML used in the XML/HTTP interface. The format of the requests and responses is described in XML schemas and a DTD that can be used to construct and validate the XML messages.

The XML schemas can be viewed in this documentation or they can be downloaded from the Oak Grove web site.

> **Note:**
> The XML schemas can be downloaded from the Oak Grove Systems web site using the following links:
> R5.xsd (http://www.oakgrovesystems.com/xml/R5.xsd)
> reactor_common.xsd (http://www.oakgrovesystems.com/xml/reactor_common.xsd)
> process.xsd (http://www.oakgrovesystems.com/xml/process.xsd)
> operand.xsd (http://www.oakgrovesystems.com/xml/operand.xsd)
> status.xsd (http://www.oakgrovesystems.com/xml/status.xsd)
> policy.xsd (http://www.oakgrovesystems.com/xml/policy.xsd)

The DTD is listed in Appendix B and can be found at this URL:

> **Note:**
> The DTD can also be downloaded from the Oak Grove Systems web site using the following link:
> R5.dtd (http://www.oakgrovesystems.com/xml/R5.dtd)

Reactor is also distributed with a WSDL file which can be used to deploy Reactor 5 Server as a web service. Future versions of Reactor are expected to also make it possible to expose individual processes and activities as web services using WSFL. The SOAP interface is deployed using Apache SOAP, which makes it possible to use the SOAP interface on a wide variety of application servers. The WSDL file is listed in Appendix D and included in the webservices directory of the Reactor Engine distribution. The WSDL file can be accessed on the deployed server through the following URL, though the host name and port number of the server may vary for different configurations.

```
http://localhost:9090/ReactorServer/wsdl/ReactorServer.wsdl
```

The following shows a sample SOAP request and response, including the HTTP headers. The examples from Section 4.3 can be placed in the request and response elements at the appropriate parts of the SOAP messages. The XML request is passed as a string argument, and the XML response is also returned as a string argument.

This is the request SOAP message.

```
POST /soap/servlet/rpcrouter HTTP/1.0
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: 828
SOAPAction: "(empty soap action)"
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
```

```
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
<ns1:handleRequest xmlns:ns1="urn:reactor-service"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<request xsi:type="xsd:string"><?xml version="1.0"
 encoding="UTF-8"?>
<reactor_request>
   <request_type>authentication_login</request_type>
   <parameters>
       <login>
           <username>admin</username>
           <password>admin</password>
       </login>
   </parameters>
</reactor_request>
</request>
</ns1:handleRequest>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

This is the response SOAP message.

```


HTTP/1.0 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 899
Set-Cookie: JSESSIONID=b41ogxyzm1;Path=/soap
```

```
 Servlet-Engine: Tomcat Web Server/3.2.2 (JSP 1.1; Servlet 2.2; Java 1.3.0;
  java.vendor=Sun Microsystems Inc.)
 <?xml version='1.0' encoding='UTF-8'?>
 <SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
 <SOAP-ENV:Body>
 <ns1:handleRequestResponse xmlns:ns1="urn:reactor-service"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
 <return xsi:type="xsd:string"><?xml version="1.0"
  encoding="UTF-8"?>
 <response><result_code><code_number>200</code_number><
  user_message>OK</user_message><debug_info /></result_code><
  return_values><authentication_token>-2f4d386d:394894:e8ed3c0e36:-77d1<
  /authentication_token><token_expiration>never</token_expiration><
  /return_values></response>
 </return>
 </ns1:handleRequestResponse>
 </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>
```

22.4.2. Reactor XML Schemas

22.4.2.1. R5.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
```

```
       xmlns:xsd='http://www.w3.org/2001/XMLSchema'
       xmlns:reactor_common='reactor_common.dtd'
       xmlns:process='process.dtd'
       xmlns:operand='operand.dtd'
       xmlns:status='status.dtd'
       xmlns:policy='policy.dtd'>
 <xsd:import namespace='reactor_common.dtd'
schemaLocation='reactor_common.xsd'/>
 <xsd:import namespace='process.dtd' schemaLocation='process.xsd'/>
 <xsd:import namespace='operand.dtd' schemaLocation='operand.xsd'/>
 <xsd:import namespace='status.dtd' schemaLocation='status.xsd'/>
 <xsd:import namespace='policy.dtd' schemaLocation='policy.xsd'/>
 <xsd:complexType name='reactor_request' mixed='true'>
  <xsd:sequence>
   <xsd:element ref='request_type'/>
   <xsd:element ref='authentication_token' minOccurs='0'/>
   <xsd:element ref='parameters' minOccurs='0'/>
  </xsd:sequence>
 </xsd:complexType>
 <xsd:element name='reactor_request' type='reactor_request'/>
 <xsd:element name='request_type' type='xsd:string'/>
 <xsd:element name='authentication_token' type='xsd:string'/>
 <xsd:element name='parameters'>
  <xsd:complexType>
   <xsd:sequence>
    <xsd:any minOccurs='0' maxOccurs='unbounded'/>
   </xsd:sequence>
```

```
 </xsd:complexType>
</xsd:element>
<xsd:complexType name='response' mixed='true'>
 <xsd:sequence>
  <xsd:element ref='result_code'/>
  <xsd:element ref='return_values' minOccurs='0'/>
 </xsd:sequence>
</xsd:complexType>
<xsd:element name='response' type='response'/>
<xsd:element name='result_code'>
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref='code_number'/>
   <xsd:element ref='user_message'/>
   <xsd:element ref='debug_info'/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
<xsd:element name='code_number' type='xsd:int'/>
<xsd:element name='user_message' type='xsd:string'/>
<xsd:element name='debug_info' type='xsd:string'/>
<xsd:element name='return_values'>
 <xsd:complexType>
  <xsd:sequence>
   <xsd:any minOccurs='0' maxOccurs='unbounded'/>
  </xsd:sequence>
 </xsd:complexType>
```

```
</xsd:element>
<xsd:element name='login'>
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref='username'/>
   <xsd:element ref='password'/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
<xsd:element name='username' type='xsd:string'/>
<xsd:element name='password' type='xsd:string'/>
<xsd:element name='token_expiration' type='xsd:string'/>
<!-- For process_command_* and process_object_* requests and responses:
-->
<!-- This is used in a 'process_object_get' request
to specify the type of object being retrieved. -->
<xsd:element name='type' type='xsd:string'/>
<xsd:element name='query'>
 <xsd:complexType>
  <xsd:choice>
   <xsd:element ref='all_objects'/>
   <xsd:element ref='process_tree'/>
   <xsd:element ref='processes'/>
  </xsd:choice>
 </xsd:complexType>
</xsd:element>
<xsd:element name='all_objects'/>
<xsd:element name='process_tree'>
```

```
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref='depth' minOccurs='0'/>
   <xsd:element ref='reactor_common:object_reference'/>
   <xsd:element ref='getSuperprocess' minOccurs='0'/>
   <xsd:element ref='getOperands' minOccurs='0'/>
   <xsd:element ref='getStatuses' minOccurs='0'/>
   <xsd:element ref='getPolicies' minOccurs='0'/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
<xsd:element name='depth' type='xsd:int'/>
<xsd:element name='getSuperprocess'/>
<xsd:element name='getOperands'/>
<xsd:element name='getStatuses'/>
<xsd:element name='getPolicies'/>
<xsd:element name='processes'>
 <xsd:complexType>
  <xsd:sequence>
   <xsd:choice minOccurs='0'>
    <xsd:element ref='definitions'/>
    <xsd:choice>
     <xsd:element ref='instances'/>
     <xsd:element ref='started_instances'/>
    </xsd:choice>
   </xsd:choice>
   <xsd:element ref='ace_pattern' minOccurs='0'/>
```

```
   <xsd:element ref='references_only' minOccurs='0'/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
<xsd:element name='definitions'/>
<xsd:element name='instances'/>
<xsd:element name='started_instances'/>
<xsd:element name='references_only'/>
<xsd:element name='ace_pattern'>
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref='principal_type' minOccurs='0'/>
   <xsd:element ref='principal' minOccurs='0'/>
   <xsd:element ref='role' minOccurs='0'/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
<!-- user|group|title -->
<xsd:element name='principal_type' type='xsd:string'/>
<xsd:element name='principal' type='xsd:string'/>
<xsd:element name='role' type='xsd:string'/>
<xsd:element name='process'>
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref='reactor_common:object_reference'/>
  </xsd:sequence>
 </xsd:complexType>
```

```
</xsd:element>
<xsd:element name='status'>
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref='reactor_common:object_reference'/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
<xsd:element name='objects'>
 <xsd:complexType>
  <xsd:sequence>
   <!-- set of process|operand|status|policy objects -->
   <xsd:any minOccurs='0' maxOccurs='unbounded'/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
<xsd:element name='use_id_object_references'/>
</xsd:schema>
```

22.4.2.2. reactor_common.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
       xmlns='http://www.w3.org/2001/XMLSchema'
       xmlns:reactor_common='reactor_common.dtd'
       targetNamespace='reactor_common.dtd'>
 <element name='label' type='string'/>
```

```
<element name='description' type='string'/>
<element name='acl'>
 <complexType>
  <sequence>
   <element ref='reactor_common:ace' minOccurs='0'
        maxOccurs='unbounded'/>
  </sequence>
 </complexType>
</element>
<element name='ace'>
 <complexType>
  <sequence>
   <element ref='reactor_common:profile' maxOccurs='unbounded'/>
   <element ref='reactor_common:role' maxOccurs='unbounded'/>
  </sequence>
 </complexType>
</element>
<element name='profile'>
 <complexType>
  <sequence>
   <choice>
    <element ref='reactor_common:user'/>
    <element ref='reactor_common:title'/>
    <element ref='reactor_common:group'/>
   </choice>
  </sequence>
 </complexType>
```

```
</element>
<element name='user' type='string'/>
<element name='title' type='string'/>
<element name='group' type='string'/>
<element name='role' type='string'/>
<element name='associated_process'>
 <complexType>
  <sequence>
   <element ref='reactor_common:object_reference' minOccurs='0'/>
  </sequence>
 </complexType>
</element>
<element name='object_reference'>
 <complexType>
  <sequence>
   <choice>
     <element ref='reactor_common:id'/>
     <element ref='reactor_common:label_path'/>
   </choice>
  </sequence>
 </complexType>
</element>
<element name='id' type='string'/>
<element name='root' type='string'/>
<element name='label_path'>
 <complexType>
  <sequence>
```

```
   <choice>
    <sequence>
     <element ref='reactor_common:root'/>
     <element ref='reactor_common:label' minOccurs='0'
          maxOccurs='unbounded'/>
    </sequence>
    <sequence>
     <element ref='reactor_common:label' maxOccurs='unbounded'/>
    </sequence>
   </choice>
  </sequence>
  <attribute name='type' use='optional'>
   <simpleType>
    <restriction base='string'>
     <enumeration value='process'/>
     <enumeration value='operand'/>
     <enumeration value='status'/>
     <enumeration value='policy'/>
     <enumeration value='object'/>
    </restriction>
   </simpleType>
  </attribute>
 </complexType>
</element>
</schema>
```

22.4.2.3. process.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
        xmlns='http://www.w3.org/2001/XMLSchema'
        xmlns:reactor_common='reactor_common.dtd'
        xmlns:process='process.dtd'
        targetNamespace='process.dtd'>
 <import namespace='reactor_common.dtd'
schemaLocation='reactor_common.xsd'/>
 <element name='process'>
  <complexType>
   <sequence>
    <element ref='reactor_common:id' minOccurs='0'/>
    <element ref='reactor_common:label'/>
    <element ref='reactor_common:description'/>
    <element ref='reactor_common:acl'/>
    <element ref='process:start_date'/>
    <element ref='process:end_date'/>
    <element ref='process:timers'/>
    <element ref='process:timer_ids'/>
    <element ref='process:state'/>
    <element ref='process:current_statuses'/>
    <element ref='process:precondition' minOccurs='0'/>
    <element ref='process:change_condition' minOccurs='0'/>
    <element ref='process:operands'/>
    <element ref='process:statuses'/>
    <element ref='process:policies'/>
    <element ref='process:superprocess'/>
```

```
   <element ref='process:Subprocesses'/>
  </sequence>
  <attribute name='definition' use='required' type='boolean'/>
 </complexType>
</element>
<element name='start_date' type='string'/>
<element name='timers'>
 <complexType>
  <sequence>
   <element ref='process:timer_spec' minOccurs='0'
        maxOccurs='unbounded'/>
  </sequence>
 </complexType>
</element>
<element name='timer_spec'>
 <complexType>
  <sequence>
   <element ref='process:timer_description'/>
   <element ref='process:event'/>
   <element ref='process:calendar' minOccurs='0'/>
   <choice minOccurs='0'>
    <element ref='process:schedule_expression'/>
    <element ref='process:schedule_date'/>
   </choice>
   <element ref='process:repeat_expression' minOccurs='0'/>
   <choice>
    <element ref='process:end_expression'/>
```

```
    <element ref='process:end_date'/>
    <element ref='process:repeat_count'/>
   </choice>
  </sequence>
 </complexType>
</element>
<element name='timer_description' type='string'/>
<element name='event'>
 <complexType>
  <sequence>
   <element ref='process:event_name'/>
   <element ref='process:event_attribute' minOccurs='0'
        maxOccurs='unbounded'/>
  </sequence>
 </complexType>
</element>
<element name='event_name' type='string'/>
<element name='event_attribute'>
 <complexType>
  <sequence>
   <element ref='process:att_key'/>
   <element ref='process:att_value'/>
  </sequence>
 </complexType>
</element>
<element name='att_key' type='string'/>
<element name='att_value' type='string'/>
```

```
<element name='calendar' type='string'/>
<element name='schedule_expression' type='string'/>
<element name='schedule_date' type='string'/>
<element name='repeat_expression' type='string'/>
<element name='end_expression' type='string'/>
<element name='end_date' type='string'/>
<element name='repeat_count' type='int'/>
<element name='timer_ids'>
 <complexType>
  <sequence>
   <element ref='process:timer_id' minOccurs='0'
        maxOccurs='unbounded'/>
  </sequence>
 </complexType>
</element>
<element name='timer_id' type='string'/>
<element name='state'>
 <complexType>
  <attribute name='state_name' use='required'>
   <simpleType>
    <restriction base='string'>
     <enumeration value='unstarted'/>
     <enumeration value='started'/>
     <enumeration value='finished'/>
    </restriction>
   </simpleType>
  </attribute>
```

```
 </complexType>
</element>
<element name='current_statuses'>
 <complexType>
  <sequence>
   <element ref='reactor_common:object_reference' minOccurs='0'
        maxOccurs='unbounded'/>
  </sequence>
 </complexType>
</element>
<element name='precondition'>
 <complexType>
  <sequence>
   <element ref='process:condition' minOccurs='0'/>
  </sequence>
 </complexType>
</element>
<element name='change_condition'>
 <complexType>
  <sequence>
   <element ref='process:condition' minOccurs='0'/>
   <element ref='process:change' maxOccurs='unbounded'/>
  </sequence>
 </complexType>
</element>
<element name='condition'>
 <complexType>
```

```
  <sequence>
   <choice>
    <element ref='process:conjunction'/>
    <element ref='process:disjunction'/>
    <element ref='process:inversion'/>
    <element ref='process:process_state_equals'/>
    <element ref='process:process_has_status'/>
   </choice>
  </sequence>
 </complexType>
</element>
<element name='conjunction'>
 <complexType>
  <sequence>
   <element ref='process:condition' maxOccurs='unbounded'/>
  </sequence>
 </complexType>
</element>
<element name='disjunction'>
 <complexType>
  <sequence>
   <element ref='process:condition' maxOccurs='unbounded'/>
  </sequence>
 </complexType>
</element>
<element name='inversion'>
 <complexType>
```

```
  <sequence>
   <element ref='process:condition'/>
  </sequence>
 </complexType>
</element>
<element name='process_state_equals'>
 <complexType>
  <sequence>
   <element ref='reactor_common:object_reference'/>
   <element ref='process:state'/>
  </sequence>
 </complexType>
</element>
<element name='process_has_status'>
 <complexType>
  <sequence>
   <element ref='reactor_common:object_reference' minOccurs='2'
        maxOccurs='2'/>
  </sequence>
 </complexType>
</element>
<element name='change'>
 <complexType>
  <sequence>
   <choice>
    <element ref='process:state_change'/>
    <element ref='process:status_addition'/>
```

```
    <element ref='process:status_removal'/>
   </choice>
  </sequence>
 </complexType>
</element>
<element name='state_change'>
 <complexType>
  <sequence>
   <element ref='process:state'/>
  </sequence>
 </complexType>
</element>
<element name='status_addition'>
 <complexType>
  <sequence>
   <element ref='reactor_common:object_reference'/>
  </sequence>
 </complexType>
</element>
<element name='status_removal'>
 <complexType>
  <sequence>
   <element ref='reactor_common:object_reference'/>
  </sequence>
 </complexType>
</element>
<element name='operands'>
```

```
 <complexType>
  <sequence>
   <element ref='reactor_common:object_reference' minOccurs='0'
        maxOccurs='unbounded'/>
  </sequence>
 </complexType>
</element>
<element name='statuses'>
 <complexType>
  <sequence>
   <element ref='reactor_common:object_reference' minOccurs='0'
        maxOccurs='unbounded'/>
  </sequence>
 </complexType>
</element>
<element name='policies'>
 <complexType>
  <sequence>
   <element ref='reactor_common:object_reference' minOccurs='0'
        maxOccurs='unbounded'/>
  </sequence>
 </complexType>
</element>
<element name='superprocess'>
 <complexType>
  <sequence>
   <element ref='reactor_common:object_reference' minOccurs='0'/>
```

```
   </sequence>
  </complexType>
 </element>
 <element name='Subprocesses'>
  <complexType>
   <sequence>
    <element ref='reactor_common:object_reference' minOccurs='0'
         maxOccurs='unbounded'/>
   </sequence>
  </complexType>
 </element>
</schema>
```

22.4.2.4. operand.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
        xmlns='http://www.w3.org/2001/XMLSchema'
        xmlns:reactor_common='reactor_common.dtd'
        xmlns:operand='operand.dtd'
        targetNamespace='operand.dtd'>
 <import namespace='reactor_common.dtd'
schemaLocation='reactor_common.xsd'/>
 <element name='operand'>
  <complexType>
   <sequence>
    <element ref='reactor_common:id' minOccurs='0'/>
```

```
    <element ref='reactor_common:label'/>
    <element ref='reactor_common:description'/>
    <element ref='reactor_common:acl'/>
    <element ref='operand:operand_value'/>
    <element ref='reactor_common:associated_process'/>
   </sequence>
   <attribute name='visible_in_entire_subtree' use='required'
type='boolean'/>
  </complexType>
 </element>
 <element name='operand_value' type='string'/>
</schema>
status.xsd
<?xml version="1.0" encoding="UTF-8"?>
<schema
        xmlns='http://www.w3.org/2001/XMLSchema'
        xmlns:reactor_common='reactor_common.dtd'
        xmlns:status='status.dtd'
        targetNamespace='status.dtd'>
 <import namespace='reactor_common.dtd'
schemaLocation='reactor_common.xsd'/>
 <element name='status'>
  <complexType>
   <sequence>
    <element ref='reactor_common:id' minOccurs='0'/>
    <element ref='reactor_common:label'/>
    <element ref='reactor_common:description'/>
    <element ref='reactor_common:acl'/>
```

```
    <element ref='reactor_common:associated_process'/>
   </sequence>
   <attribute name='applicable_to_entire_subtree' use='required'
type='boolean'/>
  </complexType>
 </element>
</schema>
```

22.4.2.5. policy.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
        xmlns='http://www.w3.org/2001/XMLSchema'
        xmlns:reactor_common='reactor_common.dtd'
        xmlns:policy='policy.dtd'
        targetNamespace='policy.dtd'>
 <import namespace='reactor_common.dtd'
schemaLocation='reactor_common.xsd'/>
 <element name='policy'>
  <complexType>
   <sequence>
    <element ref='reactor_common:id' minOccurs='0'/>
    <element ref='reactor_common:label'/>
    <element ref='reactor_common:description'/>
    <element ref='reactor_common:acl'/>
    <element ref='policy:event_name'/>
    <element ref='policy:event_attributes'/>
    <element ref='policy:policy_type'/>
```

```
   <element ref='policy:language' minOccurs='0'/>
   <element ref='policy:policy_definition'/>
   <element ref='policy:security_policy'/>
   <element ref='reactor_common:associated_process'/>
  </sequence>
 </complexType>
</element>
<element name='event_name' type='string'/>
<element name='event_attributes'>
 <complexType>
  <sequence>
   <any minOccurs='0' maxOccurs='unbounded'/>
  </sequence>
 </complexType>
</element>
<element name='event_source'>
 <complexType>
  <sequence>
   <choice>
    <element ref='reactor_common:object_reference'/>
    <element ref='policy:service_name'/>
   </choice>
  </sequence>
 </complexType>
</element>
<element name='service_name' type='string'/>
<element name='policy_type' type='string'/>
```

```
 <element name='language' type='string'/>
 <element name='policy_definition'>
  <complexType>
   <sequence>
    <choice>
     <element ref='policy:classname'/>
     <element ref='policy:URL'/>
     <element ref='policy:source_code'/>
    </choice>
   </sequence>
  </complexType>
 </element>
 <element name='security_policy' type='string'/>
 <element name='classname' type='string'/>
 <element name='URL' type='string'/>
 <element name='source_code' type='string'/>
</schema>
```

22.4.3. XML DTD

22.4.3.1. Placeholder

```


<!-- R5.dtd (Reactor 5.0.2, 2001 August 17) -->
<!-- Oak Grove Systems, Inc. -->
<!-- http://www.oakgrovesystems.com/ -->
<!--
```

```
        A note about XML Namespaces:
        Namespaces are not officially part of the XML 1.0 specification,
        so a DTD can not completely express the requirements for properly
        constructing XML documents with namespaces.
        In order to allow the XML to be properly validated, each element
        is defined to have an optional xmlns attribute. This is not an
        accurate expression of how the xmlns attribute should be used.
        Every element with a namespace in its name must be in a context
        where that namespace is defined. There must be an xmlns attribute
        in either that element or some element containing it. That xmlns
        attribute must specify the proper location of the DTD. It's not
        necessary for the location to actually contain the DTD, or even
        be reachable, but it is necessary for the location to be the
        same as used by the parser that will be reading the XML.
        Each namespace has a separate section in this DTD. The comments
        at the beginning of each section contain the xmlns attribute
        required for that namespace.
        See http://www.w3.org/TR/REC-xml-names/ for the official document
        describing XML Namespaces.
-->
<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
DEFAULT NAMESPACE (no xmlns required)
These elements are used for requests and responses. The parameters and
return values will sometimes contain elements from the other namespaces.
-->
<!ELEMENT reactor_request
        (request_type,
```

```
          authentication_token?,
          parameters?)>
 <!ELEMENT request_type (#PCDATA)>
 <!ELEMENT authentication_token (#PCDATA)> <!-- empty for login requests
-->
 <!ELEMENT parameters ANY>


 <!ELEMENT response
          (result_code,
          return_values?)>
 <!ELEMENT result_code
          (code_number,
          user_message,
          debug_info)>
 <!ELEMENT code_number (#PCDATA)>
 <!ELEMENT user_message (#PCDATA)>
 <!ELEMENT debug_info (#PCDATA)>
 <!ELEMENT return_values ANY>

 <!-- For authentication_login requests and responses: -->
 <!ELEMENT login
          (username,
          password)>
 <!ELEMENT username (#PCDATA)>
 <!ELEMENT password (#PCDATA)>

 <!ELEMENT token_expiration (#PCDATA)>
 <!-- For process_command_* and process_object_* requests and responses:
```

```
-->
          <!-- This is used in a 'process_object_get' request
          to specify the type of object being retrieved. -->
 <!ELEMENT type (#PCDATA)>
          <!-- The 'query' tag goes inside the 'parameters' tag of a
          'reactor_request' document of type 'process_object_query' -->
 <!ELEMENT query
          (all_objects
          | process_tree
          | processes)>
          <!-- This returns all stored objects (useful for debugging) -->
 <!ELEMENT all_objects EMPTY>
          <!-- This returns a process and specified associated objects. -->
 <!ELEMENT process_tree
          (depth?,
          reactor_common:object_reference,
          getSuperprocess?,
          getOperands?,
          getStatuses?,
          getPolicies?)>
          <!-- The integer depth of the object tree to return.
          0 means just the root, a negative number means the entire tree.
          Default is 0. -->
 <!ELEMENT depth (#PCDATA)>
          <!-- If the following elements are included and have the string
          "true" as the content (without quotes), then the process tree
will
          contain the superprocess of the root process, operands, statuses,
```

```
          and policies (respectively) -->
 <!ELEMENT getSuperprocess (#PCDATA)> <!-- (true|false) -->
 <!ELEMENT getOperands (#PCDATA)> <!-- (true|false) -->
 <!ELEMENT getStatuses (#PCDATA)> <!-- (true|false) -->
 <!ELEMENT getPolicies (#PCDATA)> <!-- (true|false) -->
          <!-- This returns a collection of processes matching the
          specified criteria. -->
 <!ELEMENT processes
          ((definitions | (instances|started_instances) )?,
          ace_pattern?,
          references_only?)>
          <!-- If this element is included, the query only returns
processes
          that are definitions, not instantiated processes. -->
 <!ELEMENT definitions EMPTY>
          <!-- If this element is included, the query only returns
          instantiated processes, not definitions. The instantiated
processes
          could be in any state (unstarted, started, finished) -->
 <!ELEMENT instances EMPTY>
          <!-- If this element is included, the query only returns
          instantiated that are in the "started" state. -->
 <!ELEMENT started_instances EMPTY>
          <!-- If this element is included, references will be returned
          instead of whole processes. -->
 <!ELEMENT references_only EMPTY>
          <!-- If included, the query only returns processes with one or
          more ACEs in their ACLs that match the specified pattern. -->
```

```
<!ELEMENT ace_pattern
         (principal_type?,
         principal?,
         role?)>
<!ELEMENT principal_type (#PCDATA)> <!-- (user|group|title) -->
<!ELEMENT principal (#PCDATA)> <!-- name of the user, group or title -->
<!ELEMENT role (#PCDATA)> <!-- eg: Initiative participant -->
         <!-- This is used to specify the process in a
         'process_command_add_status' request. -->
<!ELEMENT process (reactor_common:object_reference)>
         <!-- This is used to specify the status in a
         'process_command_add_status' request. -->
<!ELEMENT status (reactor_common:object_reference)>
         <!-- This is used when including a set of objects in both
         requests and responses. -->
<!ELEMENT objects ANY>
         <!-- To get object_reference elements that use 'id' instead of
         'label_path' in a response, include the
'use_id_object_references'
         element in the 'parameters' of a request. -->
<!ELEMENT use_id_object_references EMPTY>


<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
NAMESPACE reactor_common (xmlns:reactor_common="reactor_common.dtd")
These elements are general, and are used in the other namespaces.
-->
<!ELEMENT reactor_common:label (#PCDATA)>
```

```
<!ATTLIST reactor_common:label
         xmlns CDATA #FIXED "reactor_common.dtd">
         <!-- Wrap the contents in CDATA before adding to this element!
-->
         <!-- e.g. <![CDATA[...contents...]]> -->
<!ELEMENT reactor_common:description (#PCDATA)>
<!ATTLIST reactor_common:description
         xmlns CDATA #FIXED "reactor_common.dtd"
         xmlns:reactor_common CDATA #FIXED "reactor_common.dtd">
<!ELEMENT reactor_common:acl
         (reactor_common:ace)*>
<!ATTLIST reactor_common:acl
         xmlns CDATA #FIXED "reactor_common.dtd"
         xmlns:reactor_common CDATA #FIXED "reactor_common.dtd">
<!ELEMENT reactor_common:ace
         (reactor_common:profile+,
         reactor_common:role+)>
<!ATTLIST reactor_common:ace
         xmlns CDATA #FIXED "reactor_common.dtd"
         xmlns:reactor_common CDATA #FIXED "reactor_common.dtd">
<!ELEMENT reactor_common:profile
         (reactor_common:user
         | reactor_common:title
         | reactor_common:group)>
<!ATTLIST reactor_common:profile
         xmlns CDATA #FIXED "reactor_common.dtd"
         xmlns:reactor_common CDATA #FIXED "reactor_common.dtd">
<!ELEMENT reactor_common:user (#PCDATA)>
```

```
<!ATTLIST reactor_common:user
         xmlns CDATA #FIXED "reactor_common.dtd"
         xmlns:reactor_common CDATA #FIXED "reactor_common.dtd">
<!ELEMENT reactor_common:title (#PCDATA)>
<!ATTLIST reactor_common:title
         xmlns CDATA #FIXED "reactor_common.dtd"
         xmlns:reactor_common CDATA #FIXED "reactor_common.dtd">
<!ELEMENT reactor_common:group (#PCDATA)>
<!ATTLIST reactor_common:group
         xmlns CDATA #FIXED "reactor_common.dtd"
         xmlns:reactor_common CDATA #FIXED "reactor_common.dtd">
<!ELEMENT reactor_common:role (#PCDATA)>
<!ATTLIST reactor_common:role
         xmlns CDATA #FIXED "reactor_common.dtd"
         xmlns:reactor_common CDATA #FIXED "reactor_common.dtd">
<!ELEMENT reactor_common:associated_process
         (reactor_common:object_reference)?>
<!ATTLIST reactor_common:associated_process
         xmlns CDATA #FIXED "reactor_common.dtd"
         xmlns:reactor_common CDATA #FIXED "reactor_common.dtd">
<!ELEMENT reactor_common:object_reference
         (reactor_common:id
         | reactor_common:label_path)>
<!ATTLIST reactor_common:object_reference
         xmlns CDATA #FIXED "reactor_common.dtd"
         xmlns:reactor_common CDATA #FIXED "reactor_common.dtd">
<!ELEMENT reactor_common:id (#PCDATA)>
```

```
<!ATTLIST reactor_common:id
          xmlns CDATA #FIXED "reactor_common.dtd"
          xmlns:reactor_common CDATA #FIXED "reactor_common.dtd">
          <!--
          The root is used when a label path starts from a particular
          object. It contains the ID of a Reactor object.
          -->
<!ELEMENT reactor_common:root (#PCDATA)>
<!ATTLIST reactor_common:root
          xmlns CDATA #FIXED "reactor_common.dtd">
          <!--
          The <label> elements in a <label_path> are ordered.  The order of
          <label> elements represents the order of traversal down the
object
          tree from parent to child. If the object is at the root level,
          meaning that it has no parents, then the label path contains only
          the label of the object.  Otherwise, the label path starts with a
          top-level process and follows the chain of child processes down
to
          the process containing the object.  The last <label> element in a
          <label_path> is the <label> of the object to which the
<label_path>
          resolves. If specified, the type must be process, operand,
status,
          policy, or object.
          -->
<!ELEMENT reactor_common:label_path
          ((reactor_common:root,reactor_common:label*)
          | (reactor_common:label+))>
```

```
<!ATTLIST reactor_common:label_path
         type (process | operand | status | policy | object) #IMPLIED
         xmlns CDATA #FIXED "reactor_common.dtd"
         xmlns:reactor_common CDATA #FIXED "reactor_common.dtd">

<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
NAMESPACE process (xmlns:process="process.dtd")
These elements define processes, with their attributes and associations.
Attributes are contained in the process element. The process element also
contains references to associated objects.
-->
<!ELEMENT process:process
         (reactor_common:id?,
         reactor_common:label,
         reactor_common:description,
         reactor_common:acl,
         process:start_date,
         process:end_date,
         process:timers,
         process:timer_ids,
         process:state,
         process:current_statuses,
         process:precondition*,
         process:change_condition*,
         process:operands,
         process:statuses,
         process:policies,
```

```
          process:superprocess,
          process:Subprocesses)>
 <!ATTLIST process:process
          definition (true|false) #REQUIRED
          xmlns CDATA #FIXED "process.dtd"
          xmlns:process CDATA #FIXED "process.dtd"
          xmlns:reactor_common CDATA #FIXED "reactor_common.dtd">
 <!ELEMENT process:start_date (#PCDATA)>
 <!ATTLIST process:start_date
          xmlns CDATA #FIXED "process.dtd"
          xmlns:process CDATA #FIXED "process.dtd">
 <!ELEMENT process:timers
          (process:timer_spec)*>
 <!ATTLIST process:timers
          xmlns CDATA #FIXED "process.dtd"
          xmlns:process CDATA #FIXED "process.dtd">
 <!ELEMENT process:timer_spec
          ((process:timer_description),
          (process:event),
          (process:calendar)?,
          (process:schedule_expression | process:schedule_date)?,
          (process:repeat_expression)?,
          (process:end_expression | process:end_date |
process:repeat_count))>
 <!ATTLIST process:timer_spec
          xmlns CDATA #FIXED "process.dtd"
          xmlns:process CDATA #FIXED "process.dtd">
```

```
<!ELEMENT process:timer_description (#PCDATA)>
<!ATTLIST process:timer_description
         xmlns CDATA #FIXED "process.dtd"
         xmlns:process CDATA #FIXED "process.dtd">
<!ELEMENT process:event
         (process:event_name,
         (process:event_attribute*))>
<!ATTLIST process:event
         xmlns CDATA #FIXED "process.dtd"
         xmlns:process CDATA #FIXED "process.dtd">
<!ELEMENT process:event_name (#PCDATA)>
<!ATTLIST process:event_name
         xmlns CDATA #FIXED "process.dtd"
         xmlns:process CDATA #FIXED "process.dtd">
<!ELEMENT process:event_attribute
         (process:att_key,
         process:att_value)>
<!ATTLIST process:event_name
         xmlns CDATA #FIXED "process.dtd"
         xmlns:process CDATA #FIXED "process.dtd">
<!ELEMENT process:att_key (#PCDATA)>
<!ATTLIST process:att_key
         xmlns CDATA #FIXED "process.dtd"
         xmlns:process CDATA #FIXED "process.dtd">
<!ELEMENT process:att_value (#PCDATA)>
<!ATTLIST process:att_value
         xmlns CDATA #FIXED "process.dtd"
```

```
          xmlns:process CDATA #FIXED "process.dtd">
<!ELEMENT process:calendar (#PCDATA)>
<!ATTLIST process:calendar
          xmlns CDATA #FIXED "process.dtd"
          xmlns:process CDATA #FIXED "process.dtd">
<!ELEMENT process:schedule_expression (#PCDATA)>
<!ATTLIST process:schedule_expression
          xmlns CDATA #FIXED "process.dtd"
          xmlns:process CDATA #FIXED "process.dtd">
<!ELEMENT process:schedule_date (#PCDATA)>
<!ATTLIST process:schedule_date
          xmlns CDATA #FIXED "process.dtd"
          xmlns:process CDATA #FIXED "process.dtd">
<!ELEMENT process:repeat_expression (#PCDATA)>
<!ATTLIST process:repeat_expression
          xmlns CDATA #FIXED "process.dtd"
          xmlns:process CDATA #FIXED "process.dtd">
<!ELEMENT process:end_expression (#PCDATA)>
<!ATTLIST process:end_expression
          xmlns CDATA #FIXED "process.dtd"
          xmlns:process CDATA #FIXED "process.dtd">
<!ELEMENT process:end_date (#PCDATA)>
<!ATTLIST process:end_date
          xmlns CDATA #FIXED "process.dtd"
          xmlns:process CDATA #FIXED "process.dtd">
<!ELEMENT process:repeat_count (#PCDATA)>
<!ATTLIST process:repeat_count
```

```
          xmlns CDATA #FIXED "process.dtd"
          xmlns:process CDATA #FIXED "process.dtd">
<!ELEMENT process:timer_ids
          (process:timer_id)*>
<!ATTLIST process:timer_ids
          xmlns CDATA #FIXED "process.dtd"
          xmlns:process CDATA #FIXED "process.dtd">
<!ELEMENT process:timer_id (#PCDATA)>
<!ATTLIST process:timer_id
          xmlns CDATA #FIXED "process.dtd"
          xmlns:process CDATA #FIXED "process.dtd">
<!ELEMENT process:state EMPTY>
<!ATTLIST process:state
          state_name (unstarted|started|finished) #REQUIRED>
<!ELEMENT process:current_statuses
          (reactor_common:object_reference)*>
<!ATTLIST process:current_statuses
          xmlns CDATA #FIXED "process.dtd"
          xmlns:process CDATA #FIXED "process.dtd">
<!ELEMENT process:precondition
          (process:condition)?>
<!ATTLIST process:precondition
          xmlns CDATA #FIXED "process.dtd"
          xmlns:process CDATA #FIXED "process.dtd">
<!ELEMENT process:change_condition
          (process:condition,
          process:change+)>
```

```
<!ATTLIST process:change_condition
         xmlns CDATA #FIXED "process.dtd"
         xmlns:process CDATA #FIXED "process.dtd">
<!ELEMENT process:condition
         (process:conjunction
         | process:disjunction
         | process:inversion
         | process:process_state_equals
         | process:process_has_status)>
<!ATTLIST process:condition
         xmlns CDATA #FIXED "process.dtd"
         xmlns:process CDATA #FIXED "process.dtd">
<!ELEMENT process:conjunction
         (process:condition)+>
<!ATTLIST process:conjunction
         xmlns CDATA #FIXED "process.dtd"
         xmlns:process CDATA #FIXED "process.dtd">
<!ELEMENT process:disjunction
         (process:condition)+>
<!ATTLIST process:disjunction
         xmlns CDATA #FIXED "process.dtd"
         xmlns:process CDATA #FIXED "process.dtd">
<!ELEMENT process:inversion
         (process:condition)>
<!ATTLIST process:inversion
         xmlns CDATA #FIXED "process.dtd"
         xmlns:process CDATA #FIXED "process.dtd">
```

```
<!ELEMENT process:process_state_equals
         (reactor_common:object_reference,
         process:state)>
<!ATTLIST process:process_state_equals
         xmlns CDATA #FIXED "process.dtd"
         xmlns:process CDATA #FIXED "process.dtd">
<!ELEMENT process:process_has_status
         (reactor_common:object_reference,
         reactor_common:object_reference)>
<!ATTLIST process:process_has_status
         xmlns CDATA #FIXED "process.dtd"
         xmlns:process CDATA #FIXED "process.dtd">
<!ELEMENT process:change
         (process:state_change
         | process:status_addition
         | process:status_removal)>
<!ATTLIST process:change
         xmlns CDATA #FIXED "process.dtd"
         xmlns:process CDATA #FIXED "process.dtd">
<!ELEMENT process:state_change
         (process:state)>
<!ATTLIST process:state_change
         xmlns CDATA #FIXED "process.dtd"
         xmlns:process CDATA #FIXED "process.dtd">
<!ELEMENT process:status_addition
         (reactor_common:object_reference)>
<!ATTLIST process:status_addition
```

```
          xmlns CDATA #FIXED "process.dtd"
          xmlns:process CDATA #FIXED "process.dtd">
 <!ELEMENT process:status_removal
          (reactor_common:object_reference)>
 <!ATTLIST process:status_removal
          xmlns CDATA #FIXED "process.dtd"
          xmlns:process CDATA #FIXED "process.dtd">
          <!-- According to MOF/XMI, the five elements below would be
          "associations", whereas the elements above would be "attributes"
-->
 <!ELEMENT process:operands
          (reactor_common:object_reference)*>
 <!ATTLIST process:operands
          xmlns CDATA #FIXED "process.dtd"
          xmlns:process CDATA #FIXED "process.dtd">
 <!ELEMENT process:statuses
          (reactor_common:object_reference)*>
 <!ATTLIST process:statuses
          xmlns CDATA #FIXED "process.dtd"
          xmlns:process CDATA #FIXED "process.dtd">
 <!ELEMENT process:policies
          (reactor_common:object_reference)*>
 <!ATTLIST process:policies
          xmlns CDATA #FIXED "process.dtd"
          xmlns:process CDATA #FIXED "process.dtd">
          <!-- superprocess is empty if process is at the top level -->
 <!ELEMENT process:superprocess
          (reactor_common:object_reference)?>
```

```
<!ATTLIST process:superprocess
         xmlns CDATA #FIXED "process.dtd"
         xmlns:process CDATA #FIXED "process.dtd">
<!ELEMENT process:Subprocesses
         (reactor_common:object_reference)*>
<!ATTLIST process:Subprocesses
         xmlns CDATA #FIXED "process.dtd"
         xmlns:process CDATA #FIXED "process.dtd">

<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
NAMESPACE operand (xmlns:operand="operand.dtd")
Note that the associated process for an operand is associated by using
an object_reference element.
-->
<!ELEMENT operand:operand
         (reactor_common:id?,
         reactor_common:label,
         reactor_common:description,
         reactor_common:acl,
         operand:operand_value,
         reactor_common:associated_process)>
<!ATTLIST operand:operand
         visible_in_entire_subtree (true|false) #REQUIRED
         xmlns CDATA #FIXED "operand.dtd"
         xmlns:operand CDATA #FIXED "operand.dtd"
         xmlns:reactor_common CDATA #FIXED "reactor_common.dtd">
         <!-- Wrap the contents in CDATA before adding to this element!
```

```
-->
         <!-- e.g. <![CDATA[...contents...]]> -->
<!ELEMENT operand:operand_value (#PCDATA)>
<!ATTLIST operand:operand_value
         xmlns CDATA #FIXED "operand.dtd"
         xmlns:operand CDATA #FIXED "operand.dtd">

<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
NAMESPACE status (xmlns:status="status.dtd")
Note that the associated process for a status is associated by using
an object_reference element.
-->
<!ELEMENT status:status
         (reactor_common:id?,
         reactor_common:label,
         reactor_common:description,
         reactor_common:acl,
         reactor_common:associated_process)>
<!ATTLIST status:status
         applicable_to_entire_subtree (true|false) #REQUIRED
         xmlns CDATA #FIXED "status.dtd"
         xmlns:status CDATA #FIXED "status.dtd"
         xmlns:reactor_common CDATA #FIXED "reactor_common.dtd">

<!-- ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
NAMESPACE policy (xmlns:policy="policy.dtd")
Note that the associated process for a policy is associated by using
```

```
an object_reference element.
-->
<!ELEMENT policy:policy
         (reactor_common:id?,
         reactor_common:label,
         reactor_common:description,
         reactor_common:acl,
         policy:event_name,
         policy:event_attributes,
         policy:policy_type,
         policy:language?,
         policy:policy_definition,
         policy:security_policy,
         reactor_common:associated_process)>
<!ATTLIST policy:policy
         xmlns CDATA #FIXED "policy.dtd"
         xmlns:policy CDATA #FIXED "policy.dtd"
         xmlns:reactor_common CDATA #FIXED "reactor_common.dtd">
<!ELEMENT policy:event_name (#PCDATA)>
         <!-- e.g. "ProcessStateChange", "ProcessLabelChange",
              "StatusCreation", "OperandValueChanged" -->
<!ATTLIST policy:event_name
         xmlns CDATA #FIXED "policy.dtd"
         xmlns:policy CDATA #FIXED "policy.dtd">
<!ELEMENT policy:event_attributes ANY>
         <!-- information about the event (what happened?, why did it
happen?,
              who triggered it?, when did it happen?, etc.) -->
```

```
<!ATTLIST policy:event_attributes
         xmlns CDATA #FIXED "policy.dtd"
         xmlns:policy CDATA #FIXED "policy.dtd">
         <!-- for future use -->
<!ELEMENT policy:event_source
         (reactor_common:object_reference
         | policy:service_name)>
         <!-- e.g. ID of the process object involved -->
<!ATTLIST policy:event_source
         xmlns CDATA #FIXED "policy.dtd"
         xmlns:policy CDATA #FIXED "policy.dtd">
         <!-- for future use -->
<!ELEMENT policy:service_name (#PCDATA)>
<!ATTLIST policy:service_name
         xmlns CDATA #FIXED "policy.dtd"
         xmlns:policy CDATA #FIXED "policy.dtd">
<!ELEMENT policy:policy_type (#PCDATA)>
         <!-- "JavaClass", "BSF", etc. -->
<!ATTLIST policy:policy_type
         xmlns CDATA #FIXED "policy.dtd"
         xmlns:policy CDATA #FIXED "policy.dtd">
<!ELEMENT policy:language (#PCDATA)>
         <!-- java -->
<!ATTLIST policy:language
         xmlns CDATA #FIXED "policy.dtd"
         xmlns:policy CDATA #FIXED "policy.dtd">
<!ELEMENT policy:policy_definition
```

```
          (policy:classname
          | policy:URL
          | policy:source_code)>
          <!-- code or reference to code -->
<!ATTLIST policy:policy_definition
          xmlns CDATA #FIXED "policy.dtd"
          xmlns:policy CDATA #FIXED "policy.dtd">
<!ELEMENT policy:security_policy (#PCDATA)>
          <!-- the Java Security policy enforced during the execution
               of this policy -->
<!ATTLIST policy:security_policy
          xmlns CDATA #FIXED "policy.dtd"
          xmlns:policy CDATA #FIXED "policy.dtd">
<!ELEMENT policy:classname (#PCDATA)>
<!ATTLIST policy:classname
          xmlns CDATA #FIXED "policy.dtd"
          xmlns:policy CDATA #FIXED "policy.dtd">
<!ELEMENT policy:URL (#PCDATA)>
<!ATTLIST policy:URL
          xmlns CDATA #FIXED "policy.dtd"
          xmlns:policy CDATA #FIXED "policy.dtd">
          <!-- Wrap the contents in CDATA before adding to this element!
-->
          <!-- e.g. <![CDATA[...contents...]]> -->
<!ELEMENT policy:source_code (#PCDATA)>
<!ATTLIST policy:source_code
          xmlns CDATA #FIXED "policy.dtd"
```

```
          xmlns:policy CDATA #FIXED "policy.dtd">
```

22.4.4. WSDL

These files were generated by IBM WebSphere Toolkit, version 2.4.

22.4.4.1. ReactorService.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="ReactorService"
  targetNamespace="http://www.oakgrovesystems.com/ReactorService"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:interface="http://www.oakgrovesystems.com/ReactorService-interface"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:types="http://www.oakgrovesystems.com/ReactorService"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<import
location="http://localhost:8080/ReactorServer/wsdl/ReactorService-interface.wsdl"
 namespace="http://www.oakgrovesystems.com/ReactorService-interface">
</import>
<service name="ReactorService">
  <documentation>IBM WSTK V2.4 generated service definition
file</documentation>
  <port binding="interface:ReactorServiceBinding"
        name="ReactorServicePort">
    <soap:address
location="http://localhost:8080/soap/servlet/rpcrouter"/>
  </port>
</service>
```

```
</definitions>
```

22.4.4.2. ReactorService-interface.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="ReactorService"
targetNamespace="http://www.oakgrovesystems.com/ReactorService-interface"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.oakgrovesystems.com/ReactorService-interface"
xmlns:types="http://www.oakgrovesystems.com/ReactorService-interface/types/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<message name="InhandleRequestRequest">
  <part name="meth1_inType1" type="xsd:string"/>
</message>
<message name="OuthandleRequestResponse">
  <part name="meth1_outType" type="xsd:string"/>
</message>

<portType name="ReactorService">
  <operation name="handleRequest">
    <input message="tns:InhandleRequestRequest"/>
    <output message="tns:OuthandleRequestResponse"/>
  </operation>
</portType>

<binding name="ReactorServiceBinding"
```

```
        type="tns:ReactorService">
  <soap:binding style="rpc"
        transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="handleRequest">
    <soap:operation soapAction="urn:reactor-service"/>
    <input>
      <soap:body
         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
         namespace="urn:reactor-service"
         use="encoded"/>
    </input>
    <output>
      <soap:body
         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
         namespace="urn:reactor-service" use="encoded"/>
    </output>
  </operation>
</binding>
</definitions>
```

23. Reactor Requests

23.1. Request Types

The Reactor Engine can handle 16 different request types.

- AddStatus
- CloneInstance
- Create
- Delete

- Get
- Lock
- Login
- Logout
- QueryAllObjects
- QueryProcessTree
- QueryProcesses
- RemoveStatus
- SetObjects
- Start
- Stop
- Unlock

23.1.1. AddStatus

The AddStatus request adds a Status object to the "current statuses" attribute of a Process. This is useful for triggering preconditions, change conditions, and policy execution. The Status change could also serve just to provide information, and not cause any action.

Parameter	Description
Process ID	The process acquiring the new status.
Process Label Path	The process acquiring the new status.
Status ID	The status being added.
Status Label Path	The status being added.

Warning:

Either the Process ID or Process Label Path may be specified, but not both. Setting one causes the other to become unset.
Either the Status ID or Status Label Path may be specified, but not both. Setting one causes the other to become unset.

The response to this request contains a result code, but does not contain any return values.

```
Sample XML request:
        <?xml version="1.0" encoding="UTF-8"?>
        <reactor_request>
        <request_type>process_command_add_status</request_type>
```

```
<authentication_token>${authentication_token}</authentication_token>
        <parameters>
          <process>
            <reactor_common:object_reference
                      xmlns:reactor_common="reactor_common.dtd">
              <reactor_common:id>${process_id}</reactor_common:id>
            </reactor_common:object_reference>
          </process>
          <status>
            <reactor_common:object_reference
                      xmlns:reactor_common="reactor_common.dtd">
              <reactor_common:id>${status_id}</reactor_common:id>
            </reactor_common:object_reference>
          </status>
        </parameters>
        </reactor_request>



Sample XML response:
        <?xml version="1.0" encoding="UTF-8"?>
        <response>
          <result_code>
            <code_number>200</code_number>
            <user_message>OK</user_message>
            <debug_info />
          </result_code>
```

```
          <return_values />
        </response>
```

23.1.2. CloneInstance

The CloneInstance request makes a copy of a process definition and changes it to be a process instance. The process instance begins in an unstarted state, to allow for Operand values to be set before starting. Use the Start request to start the process instance.

Parameter	Description
Process ID	The process acquiring the new status.
Process Label Path	The process acquiring the new status.
Clone Label	Specifies a new label to give to the cloned process.

Warning:

Either the Process ID or Process Label Path may be specified, but not both. Setting one causes the other to become unset.

If the Clone Label is not specified, the cloned process keeps the same label as the original. The Clone Label parameter is useful for giving each process instance a label path that does not need to be relative to any object ID.

The response to this request contains a result code. If successful, the response also contains a return value.

Return Value	Description
Clone ID	The ID of the new object resulting from the clone.

```
Sample XML request:
        <?xml version="1.0" encoding="UTF-8"?>
        <reactor_request>
          <request_type>process_command_clone_instance</request_type>
<authentication_token>${authentication_token}</authentication_token>
```

```
          <parameters>
            <reactor_common:object_reference
                      xmlns:reactor_common="reactor_common.dtd">
              <reactor_common:id>${process_id}</reactor_common:id>
            </reactor_common:object_reference>
          </parameters>
        </reactor_request>



Sample XML response:
        <?xml version="1.0" encoding="UTF-8"?>
        <response>
          <result_code>
            <code_number>200</code_number>
            <user_message>OK</user_message>
            <debug_info />
          </result_code>
          <return_values>
            <reactor_common:object_reference
                      xmlns:reactor_common="reactor_common.dtd">
              <reactor_common:id>${instance_id}</reactor_common:id>
            </reactor_common:object_reference>
          </return_values>
        </response>
```

23.1.3. Create

Use the Create request to add objects that do not already exist. Use the SetObjects request to modify existing objects or to set values of objects that might not exist.

Parameter	Description
Objects	A set of Reactor objects.
Use Label Paths	Whether request should specify objects with label paths instead of object IDs. If not specified, the request will not use label paths.

An application may need the request to use label paths instead of object IDs because the Reactor Engine will assign new IDs to created objects. This only an issue if the application stores and continues to use the IDs of new objects after sending the request.

The response to this request contains a result code, but does not contain any return values.

```
Sample XML request:
        <?xml version="1.0" encoding="UTF-8"?>
        <reactor_request>
          <request_type>process_command_create</request_type>
<authentication_token>${authentication_token}</authentication_token>
          <parameters>
            <objects>
              <operand:operand xmlns:operand="operand.dtd">
                ...
              </operand:operand>
              <operand:operand xmlns:operand="operand.dtd">
                ...
              </operand:operand>
              <operand:operand xmlns:operand="operand.dtd">
                ...
              </operand:operand>
```

```
            </objects>
          </parameters>
        </reactor_request>



Sample XML response:
        <?xml version="1.0" encoding="UTF-8"?>
        <response>
          <result_code>
            <code_number>200</code_number>
            <user_message>OK</user_message>
            <debug_info />
          </result_code>
          <return_values />
        </response>
```

23.1.4. Delete

The Delete request removes an object from Reactor 5. Deleting a process also deletes all associated objects, with the exception of the parent process, but including all levels of Subprocesses and their associated objects.

Parameter	Description
ID	Specifies the object to delete.
Label Path	Specifies the object to delete.
Type	Specifies the type of the object to delete.
Delete Definitions Only	Whether to only delete Process definitions.

Warning:

Either the ID or Label Path may be specified, but not both. Setting one causes the other to become unset.

The Type should specify a Process, Operand, Status, Policy, or Object.

The "Delete Definitions Only" parameter is used in the case where a label path is used to identify the object to be deleted and the label path specified resolves to exactly one Process definition and at least one other Process instance. Without setting this parameter's value to "true", the default is for Reactor to fail if a label path matches multiple objects.

The response to this request contains a result code, but does not contain any return values.

```
Sample XML request:
        <?xml version="1.0" encoding="UTF-8"?>
        <reactor_request>
          <request_type>process_command_delete</request_type>
<authentication_token>${authentication_token}</authentication_token>
          <parameters>
            <type>process</type>
            <reactor_common:object_reference
                      xmlns:reactor_common="reactor_common.dtd">
              <reactor_common:id>${operand_id}</reactor_common:id>
            </reactor_common:object_reference>
          </parameters>
        </reactor_request>



Sample XML response:
        <?xml version="1.0" encoding="UTF-8"?>
        <response>
```

```
          <result_code>
            <code_number>200</code_number>
            <user_message>OK</user_message>
            <debug_info />
          </result_code>
          <return_values />
        </response>
```

23.1.5. Get

The Get request retrieves one object from Reactor. Use the query requests to retrieve multiple objects.

Parameter	Description
ID	Specifies the object to acquire.
Label Path	Specifies the object to acquire.
Type	Specifies the type of the object to acquire.

Warning:
Either the ID or Label Path may be specified, but not both. Setting one causes the other to become unset.

The Type should specify a Process, Operand, Status, Policy, or Object.

The response to this request contains a result code. If successful, the response also contains the following return value.

Return Value	Description
Object	The requested Reactor object.

To get more than one object, use one of the query requests.

```
Sample XML request:
        <?xml version="1.0" encoding="UTF-8"?>
```

```
        <reactor_request>
          <request_type>process_object_get</request_type>
<authentication_token>${authentication_token}</authentication_token>
          <parameters>
            <type>operand</type>
            <reactor_common:object_reference
                      xmlns:reactor_common="reactor_common.dtd">
              <reactor_common:id>${operand_id}</reactor_common:id>
            </reactor_common:object_reference>
          </parameters>
        </reactor_request>



Sample XML response:
        <?xml version="1.0" encoding="UTF-8"?>
        <response>
          <result_code>
            <code_number>200</code_number>
            <user_message>OK</user_message>
            <debug_info />
          </result_code>
          <return_values>
            <object>
              <operand:operand xmlns:operand="operand.dtd">
                ...
              </operand:operand>
```

```
            </object>
          </return_values>
        </response>
```

23.1.6. Lock

The Lock request acquires a role in the ACL of an element. If the role is already present in the ACL, the request is denied. The request can optionally take the name of an "eligible" role and the name of a "standby" role. If present, elements with the "eligible" role are modified to have the "standby" role. The request can also take a status. This will add a status to the process being locked, or to the associated process if the object is not a process.

Parameter	Description
Object ID	Specifies the object to lock.
Object Label Path	Specifies the object to lock.
ACE	The ACE to create.
Eligible Role	The role for ACEs which could have acquired the lock.
Standby Role	The role for ACEs which can no longer acquire the lock.
Status ID	Specifies the status to add.
Status Label Path	Specifies the status to add.

Warning:

The Object ID and Object Label Path cannot both be specified. Similarly, the Status ID and Status Label Path cannot both be specified.

```
Sample XML request:
        <?xml version="1.0" encoding="UTF-8"?>
        <reactor_request xmlns:reactor_common="reactor_common.dtd">
          <request_type>
```

```
            process_command_lock
          </request_type>
          <authentication_token>
            ${authentication_token}
          </authentication_token>
          <parameters>
            <reactor_common:object_reference>
              <reactor_common:id>${id}</reactor_common:id>
            </reactor_common:object_reference>
            <reactor_common:ace>
              <reactor_common:profile>
                <reactor_common:user>${user}</reactor_common:user>
              </reactor_common:profile>
              <reactor_common:role>${lock_role}</reactor_common:role>
            </reactor_common:ace>
            <eligible_role>${eligible_role}</eligible_role>
            <standby_role>${standby_role}</standby_role>
          </parameters>
        </reactor_request>
```

Sample XML response:

```
        <?xml version="1.0" encoding="UTF-8"?>
        <response>
          <result_code>
            <code_number>200</code_number>
```

```
            <user_message>OK</user_message>
            <debug_info />
          </result_code>
          <return_values />
        </response>
```

23.1.7. Login

The Login request gets an authentication token which will be required for all subsequent requests in a session.

Parameter	Description
Username	Username of the person or entity logging in.
Password	Password of the person or entity logging in.

Both the username and password must be specified.

The response to this request contains a result code. If successful, the response also contains the following return values.

Return Value	Description
Authentication Token	Used for authenticating in subsequent requests.
Token Expiration	Contains specification of when the token expires (if ever).

```
Sample XML request:
        <?xml version="1.0" encoding="UTF-8"?>
        <reactor_request>
          <request_type>authentication_login</request_type>
          <parameters>
            <login>
              <username>${username}</username>
```

```
            <password>${password}</password>
          </login>
        </parameters>
      </reactor_request>
```

Warning:

Note that Reactor does not trim leading and trailing whitespace around the password, since whitespace may be significant in the password. This is an exception, since whitespace around element content is usually trimmed.

```
Sample XML response:
      <?xml version="1.0" encoding="UTF-8"?>
      <response>
        <result_code>
          <code_number>200</code_number>
          <user_message>OK</user_message>
          <debug_info />
        </result_code>
        <return_values>
          <authentication_token>${auth_token}</authentication_token>
          <token_expiration>never</token_expiration>
        </return_values>
      </response>
```

23.1.8. Logout

The Logout request relinquishes an authentication token, ending a session.

Parameter	Description
Token to Expire	The authentication token to log out.

Note that the Logout request does not take the authentication token from the request itself, but from the parameter.

The response to this request contains a result code, but does not contain any return values.

```
Sample XML request:
        <?xml version="1.0" encoding="UTF-8"?>
        <reactor_request>
          <request_type>authentication_logout</request_type>
<authentication_token>${authentication_token}</authentication_token>
          <parameters>
            <authentication_token>
              ${authentication_token}
            </authentication_token>
          </parameters>
        </reactor_request>



Sample XML response:
        <?xml version="1.0" encoding="UTF-8"?>
        <response>
          <result_code>
            <code_number>200</code_number>
            <user_message>OK</user_message>
            <debug_info />
          </result_code>
          <return_values />
```

```
        </response>
```

23.1.9. QueryAllObjects

The QueryAllObjects request is useful for archiving or exporting the entire Reactor data set.

Parameter	Description
Use Label Paths	XML response will use label paths instead of IDs for object references. (optional)

The response to this request contains a result code. If successful, the response also contains the following return value:

Parameter	Description
Objects	The set of all Reactor objects.

```
Sample XML request:
        <?xml version="1.0" encoding="UTF-8"?>
        <reactor_request>
          <request_type>process_object_query</request_type>
<authentication_token>${authentication_token}</authentication_token>
          <parameters>
            <query>
              <all_objects />
            </query>
          </parameters>
        </reactor_request>



Sample XML response:
```

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
  <result_code>
    <code_number>200</code_number>
    <user_message>OK</user_message>
    <debug_info />
  </result_code>
  <return_values>
    <objects>
      <process:process xmlns:process="process.dtd">
        ...
      </process:process>
      ...
    </objects>
  </return_values>
</response>
```

23.1.10. QueryProcessTree

The QueryProcessTree request is useful for getting details about a specific process and its associated objects.

Parameter	Description
ID	Object ID of the Process
Label Path	The label path of the Process.
Definition Tree Only	True if label path applies only to definitions.
Depth	How many levels of Subprocesses to include. *(-1 for all levels, 0 for no Subprocesses)*
Include Superprocess	Specifies whether parent process should be returned.

Include Operands	Specifies whether operands should be returned.
Include Statuses	Specifies whether statuses should be returned.
Include Policies	Specifies whether policies should be returned.

Warning:
Either the ID or Label Path may be specified, but not both. Setting one causes the other to become unset.

The "Definition Tree Only" parameter is optional. It is used in the case where a label path is used to identify the process and the label path resolves to exactly one Process definition and at least one other Process instance. Without setting this parameter's value to "true", the default is for Reactor to fail if a label path matches multiple objects.

The response to this request contains a result code. If successful, the response also contains the following return value:

Return Value	Description
Objects	Specified Reactor objects.

```
Sample XML request:
      <?xml version="1.0" encoding="UTF-8"?>
        <reactor_request>
          <request_type>process_object_query</request_type>
<authentication_token>${authentication_token}</authentication_token>
          <parameters>
            <query>
              <process_tree>
                <depth>-1</depth>
                <reactor_common:object_reference
                          xmlns:reactor_common="reactor_common.dtd">
                  <reactor_common:id
xmlns:reactor_common="reactor_common.dtd">
```

```
                    ...
                  </reactor_common:id>
                </reactor_common:object_reference>
                <getOperands/>
                <getStatuses/>
                <getPolicies/>
              </process_tree>
            </query>
          </parameters>
        </reactor_request>
```

Sample XML response:

```
        <?xml version="1.0" encoding="UTF-8"?>
        <response>
          <result_code>
            <code_number>200</code_number>
            <user_message>OK</user_message>
            <debug_info />
          </result_code>
          <return_values>
            <objects>
              <process:process xmlns:process="process.dtd">
                ...
              </process:process>
              ...
            </objects>
```

```
        </return_values>
      </response>
```

23.1.11. QueryProcesses

The QueryProcesses request is useful for getting a set of processes where a user or group is assigned a particular role.

Parameter	Description
ACE Profile	Describes element in ACL of every Process to be returned.
Must Be Definition	Specifies whether to return only process definitions.
Must Be Instance	Specifies whether to return only process instances.
Must Be Started	Specifies whether to return only started process instances.
Only IDs	Specifies whether to return the IDs of the found processes instead of the objects.

The parameters specifying the type of process must be consistently set. For example, "Must Be Definition" cannot be true when either "Must Be Instance" or "Must Be Started" is true.

To find every process where a user holds a certain role: set the ACE Profile type to user, set the user name, and set the role name. Similar approaches allow queries for processes where a title or group holds a certain role.

To find all process definitions, leave the ACE Profile unspecified and set "Must Be Definition" to true.

The response to this request contains a result code. If successful, the response also contains the following return value:

Return Value	Description
Objects	Specified Reactor objects.

```
Sample XML request:
        <?xml version="1.0" encoding="UTF-8"?>
        <reactor_request>
          <request_type>process_object_query</request_type>
<authentication_token>${authentication_token}</authentication_token>
          <parameters>
            <query>
              <processes>
                <started_instances />
                <ace_pattern>
                  <principal_type>user</principal_type>
                  <principal>bob</principal>
                  <role>Initiative Participant</role>
                </ace_pattern>
              </processes>
            </query>
          </parameters>
        </reactor_request>



Sample XML response:
        <?xml version="1.0" encoding="UTF-8"?>
        <response>
          <result_code>
            <code_number>200</code_number>
            <user_message>OK</user_message>
            <debug_info />
```

```
        </result_code>
        <return_values>
          <objects>
            <process:process xmlns:process="process.dtd">
              ...
            </process:process>
            ...
          </objects>
        </return_values>
      </response>
```

23.1.12. RemoveStatus

The RemoveStatus request removes a Status object from the "current statuses" attribute of a Process. This is useful for triggering preconditions, change conditions, and policy execution. The Status change could also serve just to provide information, and not cause any action.

Parameter	Description
Process ID	The process acquiring the new status.
Process Label Path	The process acquiring the new status.
Status ID	The status being added.
Status Label Path	The status being added.

Warning:

Either the Process ID or Process Label Path may be specified, but not both. Setting one causes the other to become unset.
Either the Status ID or Status Label Path may be specified, but not both. Setting one causes the other to become unset.

The response to this request contains a result code, but does not contain any return values.

```


Sample XML request:
```

```
        <?xml version="1.0" encoding="UTF-8"?>
        <reactor_request>
        <request_type>process_command_remove_status</request_type>
<authentication_token>${authentication_token}</authentication_token>
        <parameters>
          <process>
            <reactor_common:object_reference
                      xmlns:reactor_common="reactor_common.dtd">
              <reactor_common:id>${process_id}</reactor_common:id>
            </reactor_common:object_reference>
          </process>
          <status>
            <reactor_common:object_reference
                      xmlns:reactor_common="reactor_common.dtd">
              <reactor_common:id>${status_id}</reactor_common:id>
            </reactor_common:object_reference>
          </status>
        </parameters>
        </reactor_request>



Sample XML response:
        <?xml version="1.0" encoding="UTF-8"?>
        <response>
          <result_code>
            <code_number>200</code_number>
```

```
            <user_message>OK</user_message>
            <debug_info />
          </result_code>
          <return_values />
        </response>
```

23.1.13. SetObjects

The SetObjects request either updates objects if they already exist or modifies them if they do not. It is the client's responsibility to ensure that they have a "recent" copy of the objects to be updated (so as to not overwrite other client changes).

Parameter	Description
Objects To Update	Specifies a set of Reactor objects.
Use Label Paths	This parameter is optional. It specifies whether the request is identifying objects with label paths rather than object IDs. By default, the Reactor Engine assumes the request is identifying objects with object IDs.

For updates, a client application should use label paths instead of object IDs because the Reactor Engine may assign new IDs to created objects.

The response to this request contains a result code, but does not contain any return values.

```
Sample XML request:
<?xml version="1.0" encoding="UTF-8"?>
<reactor_request>
  <request_type>process_command_set</request_type>
  <authentication_token>authToken</authentication_token>
  <parameters>
    <use_id_object_references />
    <objects>
```

```
      <operand:operand
xmlns:operand="operand.dtd"visible_in_entire_subtree="false">
        <reactor_common:idxmlns:reactor_common="reactor_common.dtd">
                 402881e6:78a212:fb9d230b1f:-8000</reactor_common:id>
        <reactor_common:labelxmlns:reactor_common="reactor_common.dtd">
                 Process Title</reactor_common:label>
<reactor_common:descriptionxmlns:reactor_common="reactor_common.dtd"><
                 ![CDATA[Description]]
         ></reactor_common:description>
        <reactor_common:acl xmlns:reactor_common="reactor_common.dtd" />
        <operand:operand_value><![CDATA[New
Value]]></operand:operand_value>
<reactor_common:associated_processxmlns:reactor_common="reactor_common.dtd">
          <reactor_common:object_reference>
            <reactor_common:id>OperandObjectId</reactor_common:id>
          </reactor_common:object_reference>
        </reactor_common:associated_process>
        <reactor_common:metadata xmlns:reactor_common="reactor_common.dtd"
/>
      </operand:operand>
    </objects>
  </parameters>
</reactor_request>


Sample XML response:
        <?xml version="1.0" encoding="UTF-8"?>
        <response>
          <result_code>
```

```
            <code_number>200</code_number>
            <user_message>OK</user_message>
            <debug_info />
          </result_code>
          <return_values />
        </response>
```

23.1.14. Start

The Start request changes the state of a process instance to be "started".

Parameter	Description
Process ID	The process acquiring the new status
Process Label Path	The process acquiring the new status.

Warning:

Either the Process ID or Process Label Path may be specified, but not both. Setting one causes the other to become unset.

The response to this request contains a result code, but does not contain any return values.

```
Sample XML request:
        <?xml version="1.0" encoding="UTF-8"?>
        <reactor_request>
          <request_type>process_command_start</request_type>
<authentication_token>${authentication_token}</authentication_token>
          <parameters>
            <reactor_common:object_reference
                      xmlns:reactor_common="reactor_common.dtd">
              <reactor_common:id>${process_id}</reactor_common:id>
```

```
            </reactor_common:object_reference>
          </parameters>
        </reactor_request>



Sample XML response:
        <?xml version="1.0" encoding="UTF-8"?>
        <response>
          <result_code>
            <code_number>200</code_number>
            <user_message>OK</user_message>
            <debug_info />
          </result_code>
          <return_values />
        </response>
```

23.1.15. Stop

The Stop request changes the state of a process instance to be "finished".

Parameter	Description
Process ID	The process acquiring the new status.
Process Label Path	The process acquiring the new status.

Warning:
Either the Process ID or Process Label Path may be specified, but not both. Setting one causes the other to become unset.

The response to this request contains a result code, but does not contain any return values.

```
Sample XML request:
        <?xml version="1.0" encoding="UTF-8"?>
        <reactor_request>
          <request_type>process_command_stop</request_type>
<authentication_token>${authentication_token}</authentication_token>
          <parameters>
            <reactor_common:object_reference
                      xmlns:reactor_common="reactor_common.dtd">
              <reactor_common:id>${process_id}</reactor_common:id>
            </reactor_common:object_reference>
          </parameters>
        </reactor_request>


Sample XML response:
        <?xml version="1.0" encoding="UTF-8"?>
        <response>
          <result_code>
            <code_number>200</code_number>
            <user_message>OK</user_message>
            <debug_info />
          </result_code>
          <return_values />
        </response>
```

23.1.16. Unlock

The Unlock request release a role from the ACL of an element. If the role is not already present in the ACL, the request is denied. The request can optionally take the name of an

"eligible" role and the name of a "standby" role. If present, elements with the "standby" role are modified to have the "role" role. The request can also take a status. This will remove a status from the process being unlocked, or from the associated process if the object is not a process.

Parameter	Description
Object ID	Specifies the object to unlock.
Object Label Path	Specifies the object to unlock.
ACE	Specifies the ACE to remove.
Eligible Role	Specifies the role for ACEs which will be able to acquire a lock.
Standby Role	Specifies the role for ACEs which can not currently acquire a lock.
Status ID	Specifies the status to remove.
Status Label Path	Specifies the status to remove.

Warning:

The Object ID and Object Label Path cannot both be specified. Similarly, the Status ID and Status Label Path cannot both be specified.

```
Sample XML request:
        <?xml version="1.0" encoding="UTF-8"?>
        <reactor_request xmlns:reactor_common="reactor_common.dtd">
          <request_type>
            process_command_unlock
          </request_type>
          <authentication_token>
            ${authentication_token}
          </authentication_token>
          <parameters>
```

```
            <reactor_common:object_reference>
              <reactor_common:id>${id}</reactor_common:id>
            </reactor_common:object_reference>
            <reactor_common:ace>
              <reactor_common:profile>
                <reactor_common:user>${user}</reactor_common:user>
              </reactor_common:profile>
              <reactor_common:role>${lock_role}</reactor_common:role>
            </reactor_common:ace>
            <eligible_role>${eligible_role}</eligible_role>
            <standby_role>${standby_role}</standby_role>
          </parameters>
        </reactor_request>


Sample XML response:
        <?xml version="1.0" encoding="UTF-8"?>
        <response>
          <result_code>
            <code_number>200</code_number>
            <user_message>OK</user_message>
            <debug_info />
          </result_code>
          <return_values />
        </response>
```

23.2. Java Code Examples

This page provides some Java code samples for various Reactor requests.

23.2.1. AddStatus

```
ReactorObjectId processId = ...
ReactorObjectId statusId = ...
String authToken = ...
AddStatus request = new AddStatus();
request.setProcessId(processId);
request.setStatusId(statusId);
request.setAuthToken(authToken);
ReactorProxy reactor = ...
ReactorResponse response = reactor.handleRequest(request);
if (response.failed()) {
...
}
```

23.2.2. CloneInstance

```
ReactorObjectId processId = ...
String authToken = ...
CloneInstance request = new CloneInstance();
request.setProcessId(processId);
request.setAuthToken(authToken);
ReactorProxy reactor = ...
```

```
ReactorResponse baseResponse = reactor.handleRequest(request);
if (baseResponse.failed()) {
...
}
CloneInstanceResponse response = (CloneInstanceResponse) baseResponse;
String instanceId = response.getInstanceId();
```

23.2.3. Create

```
Operand operand1 = ...
Operand operand2 = ...
Operand operand3 = ...
Set objects = new HashSet();
objects.add(operand1);
objects.add(operand2);
objects.add(operand3);
String authToken = ...
Create request = new Create();
request.setObjects(objects);
request.setAuthToken(authToken);
ReactorProxy reactor = ...
ReactorResponse baseResponse = reactor.handleRequest(request);
if (baseResponse.failed()) {
...
}
```

23.2.4. Delete

```
ReactorObjectId id = ...
String authToken = ...
Delete request = new Delete();
request.setId(id);
request.setType(Operand.class);
request.setAuthToken(authToken);
ReactorProxy reactor = ...
ReactorResponse response = reactor.handleRequest(request);
if (response.failed()) {
...
}
```

23.2.5. Get

```
ReactorObjectId id = ...
String authToken = ...
Get request = new Get();
request.setId(id);
request.setType(Operand.class);
request.setAuthToken(authToken);
ReactorProxy reactor = ...
ReactorResponse baseResponse = reactor.handleRequest(request);
```

```
if (baseResponse.failed()) {
...
}
GetResponse response (GetResponse) baseResponse;
Operand operand = (Operand) response.getObject();
```

23.2.6. Lock

```
ReactorObjectId objectId = ...
ACE lockAce = ...
String authToken = ...

Lock request = new Lock();
request.setId(objectId);
request.setLockACE(ace);
request.setEligibleRole("Eligible Participant");
request.setStandbyRole("Standby Participant");

ReactorProxy reactor = ...
ReactorResponse response = reactor.handleRequest(request);
if (response.failed()) {
...
}
```

23.2.7. Login

```
String username = ...
String password = ...
Login request = new Login();
request.setUsername(username);
request.setPassword(password);
ReactorProxy reactor = ...
ReactorResponse baseResponse = reactor.handleRequest(request);
if (baseResponse.failed()) {
...
}
LoginResponse response (LoginResponse) baseResponse;
String authToken = response.getAuthToken();
```

23.2.8. Logout

```
String authToken = ...
Logout request = new Logout();
request.setTokenToExpire(authToken);
request.setAuthToken(authToken);
ReactorProxy reactor = ...
ReactorResponse response = reactor.handleRequest(request);
if (response.failed()) {
...
}
```

23.2.9. QueryAllObjects

```
String authToken = ...
QueryAllObjects request = new QueryAllObjects();
request.setAuthToken(authToken);
ReactorProxy reactor = ...
ReactorResponse baseResponse = reactor.handleRequest(request);
if (baseResponse.failed()) {
...
}
QueryResponse response (QueryResponse) baseResponse;
Set objects = response.getObjects();
```

23.2.10. QueryProcessTree

```
ReactorObjectId processId = ...
String authToken = ...
QueryProcessTree request = new QueryProcessTree();
request.setId(processId);
request.setDepth(-1);
request.setIncludeSuperprocess(false);
request.setIncludeOperands(true);
request.setIncludeStatuses(true);
request.setIncludePolicies(true);
```

```
request.setAuthToken(authToken);
ReactorProxy reactor = ...
ReactorResponse baseResponse = reactor.handleRequest(request);
if (baseResponse.failed()) {
...
}
QueryResponse response (QueryResponse) baseResponse;
Set objects = response.getObjects();
```

23.2.11. QueryProcesses

```
ACE profile = new ACE(ACE.USER_TYPE, "bob", "Initiative Participant");
String authToken = ...
QueryProcesses request = new QueryProcesses();
request.setACE(profile);
request.setMustBeStarted(true);
request.setAuthToken(authToken);
ReactorProxy reactor = ...
ReactorResponse baseResponse = reactor.handleRequest(request);
if (baseResponse.failed()) {
...
}
QueryResponse response (QueryResponse) baseResponse;
Set objects = response.getObjects();
```

23.2.12. Remove Status

```
ReactorObjectId processId = ...
ReactorObjectId statusId = ...
String authToken = ...
RemoveStatus request = new RemoveStatus();
request.setProcessId(processId);
request.setStatusId(processId);
request.setAuthToken(authToken);
ReactorProxy reactor = ...
ReactorResponse response = reactor.handleRequest(request);
if (response.failed()) {
...
}
```

23.2.13. SetObjects

```
Operand operand1 = ...
Operand operand2 = ...
Operand operand3 = ...
Set objects = new HashSet();
objects.add(operand1);
objects.add(operand2);
objects.add(operand3);
```

```
String authToken = ...
SetObjects request = new SetObjects();
request.setObjectsToUpdate(objects);
request.setAuthToken(authToken);
ReactorProxy reactor = ...
ReactorResponse baseResponse = reactor.handleRequest(request);
if (baseResponse.failed()) {
...
}
```

23.2.14. Start

```
ReactorObjectId processId = ...
String authToken = ...
Start request = new Start();
request.setProcessId(processId);
request.setAuthToken(authToken);
ReactorProxy reactor = ...
ReactorResponse response = reactor.handleRequest(request);
if (response.failed()) {
...
}
```

23.2.15. Stop

```
ReactorObjectId processId = ...
String authToken = ...
Stop request = new Stop();
request.setProcessId(processId);
request.setAuthToken(authToken);
ReactorProxy reactor = ...
ReactorResponse response = reactor.handleRequest(request);
if (response.failed()) {
...
}
```

23.2.16. Unlock

```


ReactorObjectId objectId = ...
ACE lockAce = ...
String authToken = ...

Unlock request = new Lock();
request.setId(objectId);
request.setLockACE(ace);
request.setEligibleRole("Eligible Participant");
request.setStandbyRole("Standby Participant");

ReactorProxy reactor = ...
ReactorResponse response = reactor.handleRequest(request);
```

```
if (response.failed()) {
...
}
```

23.3. Reactor Java API

This section describes the highlights of the Java API for Reactor. Refer to the Javadoc HTML pages for specific details about arguments, return values, and exceptions. The Javadoc HTML pages are distributed with the Reactor Engine, in the following directory.

`docs/javadocs/index.html`

The ReactorProxy, ReactorQuery, and ReactorObjects classes are the most important classes for accessing process mediation objects and communicating with the Reactor Engine.

23.3.1. ReactorProxy

The ReactorProxy interface is found in the `com.oakgrovesystems.reactor.client` package. It is implemented by two classes in the same package, EJBReactorProxy and XMLReactorProxy. They provide the following methods:

- login()
- logout()
- setAuthToken()
- getAuthToken()
- createObjects()
- setObjects()
- delete()
- cloneInstance()
- startProcess()
- stopProcess()
- get()
- query()
- lock()
- unlock()
- addStatusToProcess()
- removeStatusFromProcess()

23.3.1.1. ReactorProxy login()

The login() method takes a username and password as arguments, and returns an authentication token. This authentication token is also stored by the ReactorProxy object for use in subsequent method calls. Either login() or setAuthToken() must be called successfully before other methods will be able to succeed. Reactor Engine requires a valid authentication token with every request, except login requests.

23.3.1.2. ReactorProxy logout()

The logout() method invalidates the authentication token set by a previous call to login() or setAuthToken().

23.3.1.3. ReactorProxy setAuthToken()

The setAuthToken() method provides a way to pass an authentication token directly to the ReactorProxy object. This is useful when the token has been acquired previously and stored as session data.

23.3.1.4. ReactorProxy getAuthToken()

The getAuthToken() method returns the authentication token set by a previous call to login() or setAuthToken(). This is useful for getting the token to be stored as session data.

23.3.1.5. ReactorProxy createObjects()

The createObjects() method sends a set of objects to Reactor Engine to be added to the workflow engine. It will fail if an object in the set has the same ID as an object already known to the workflow engine. To modify existing objects, use the setObjects() method.

23.3.1.6. ReactorProxy setObjects()

The setObjects() method sends a set of objects to Reactor Engine to be updated in the workflow engine. The objects can be either new or modified; they do not have to be previously known to the workflow engine.

23.3.1.7. ReactorProxy delete()

The delete() method removes objects from the workflow engine. When a process is deleted, all the associated objects are also deleted.

23.3.1.8. ReactorProxy cloneInstance()

Before a process can be started, it must be cloned. This means that a copy is made of a

process definition, along with all its Subprocesses and associated objects. The only difference between the process definition and the process instance is the value of the isDefinition property in the process. The cloned instance begins in an unstarted state. This provides the opportunity to call setObjects() to configure the process instance. This might include adding operands, setting operand values, and setting ACLs.

23.3.1.9. ReactorProxy startProcess()

Once a process instance has been created, the startProcess() method starts the process instance. This event may trigger the execution of policies. The new state of the process may also trigger changes by causing some conditions to become true.

23.3.1.10. ReactorProxy stopProcess()

The stopProcess() method causes a running process instance to stop. This event may trigger the execution of policies. The new state of the process may also trigger changes by causing some conditions to become true.

23.3.1.11. ReactorProxy get()

The get() method retrieves one object from Reactor Engine. Either the label path or object ID must be known. Use the query() method to retrieve multiple objects, or when an object's exact identity is not known.

23.3.1.12. ReactorProxy query()

The query() method retrieves objects from the Reactor Engine based on some criteria. A QueryCriteria object defines the specific criteria for which objects to return, as well as what associated objects to return. The results are stored in a ReactorObjects object.

23.3.1.13. ReactorProxy lock()

The lock() method attempts to acquire a role in an Access Control Element (ACE) that is not held by any other user or group. The method fails if the role is already held. The lock() method optionally takes arguments for eligible and standby roles. All elements with the eligible role are modified to have the standby role. Another optional argument is a status that to be added to the process if the lock succeeds. If the specified object is not a process, then the status will be added to the object's associated process.

The lock() method is useful in situations where only one person should be able to edit a process definition or work on an activity in a running process instance.

23.3.1.14. ReactorProxy unlock()

The unlock() method reverses the effects of a lock() method. It attempts to remove a role in an Access Control Element. It can also modify entries with a specified standby role to change to an eligible role. If a status is specified, it will remove the status from the process. If the object is not a process, it will attempt to remove the status from the object's associated process.

23.3.1.15. ReactorProxy addStatusToProcess()

The addStatusToProcess() method adds a status to the set of current statuses for a process. This event may trigger the execution of policies. The change in status of the process may also trigger changes by causing some conditions to become true.

23.3.1.16. ReactorProxy removeStatusFromProcess()

The removeStatusFromProcess() method removes a status from the set of current statuses for a process. This event may trigger the execution of policies. The change in status of the process may also trigger changes by causing some conditions to become true.

23.3.2. QueryCriteria

The QueryCriteria class is found in the `com.oakgrovesystems.reactor.client` package. It is used to retrieve processes and their associated objects. Some properties of the criteria identify characteristics that must be matched by retrieved processes. Other properties specify which associated objects to retrieve, and how many levels of Subprocesses to retrieve. Each property has accessor methods to set and get the property. The methods to set properties are listed below.

- setProcessId()
- setProcessLabelPath()
- setACE()
- setStateEquals()
- setMustBeDefinition()
- setMustBeInstance()
- setMustBeRoot()
- setHasStatusId()
- setHasStatusLabel()
- setHasParentId()
- setHasParentLabel()
- setHasAssociatedSubprocessId()

- setHasAssociatedSubprocessLabel()
- setHasAssociatedOperandId()
- setHasAssociatedOperandLabel()
- setHasAssociatedStatusId()
- setHasAssociatedStatusLabel()
- setHasAssociatedPolicyId()
- setHasAssociatedPolicyLabel()
- setDepth()
- setIncludeOperands()
- setIncludeParent()
- setIncludePolicies()
- setIncludeStatuses()

23.3.2.1. QueryCriteria setProcessId()

This method sets the ID of a particular process. The query will retrieve that process, along with whatever Subprocesses and associated objects are specified in further properties. The setProcessLabelPath() method can be used when the ID of the process is not known.

23.3.2.2.

23.3.2.3. QueryCriteria setProcessLabelPath()

This method sets the label path identifying of a particular process or group of processes. The query will retrieve those processes, along with whatever Subprocesses and associated objects are specified in further properties. The setProcessId() method can be used when the ID of the process is known.

23.3.2.4. QueryCriteria setACE()

This method restricts the set of retrieved processes to those with an ACL containing the specified ACE. For example, this is useful for querying all the process instances where the user is a participant.

23.3.2.5. QueryCriteria setStateEquals()

This method sets the state that retrieved processes must be in. This is useful for restricting the query to processes that are currently running.

23.3.2.6. QueryCriteria setMustBeDefinition()

This method restricts the query to retrieve only process definitions, and not process instances.

23.3.2.7. QueryCriteria setMustBeInstance()

This method restricts the query to retrieve only process instances, and not process definitions.

23.3.2.8. QueryCriteria setMustBeRoot()

This method indicates that the retrieved processes must not have any parents. This is useful when querying process definitions where the Subprocesses should be filtered out.

23.3.2.9. QueryCriteria setHasStatusId()

This method specifies a status that must be in the set of current statuses for retrieved processes. The setHasStatusLabel() method can be used when the ID of the status is not known, or when any status with a particular label is sufficient. Use setHasAssociatedStatusId() to specify an associated status rather than a current status.

23.3.2.10. QueryCriteria setHasStatusLabel()

This method specifies a status that must be in the set of current statuses for retrieved processes. The setHasStatusId() method can be used when only a specific status with a known ID will satisfy the criteria. Use setHasAssociatedStatusLabel() to specify an associated status rather than a current status.

23.3.2.11. QueryCriteria setHasParentId()

This method specifies a process that must be the parent of every retrieved process. The setHasParentLabel() method can be used when the ID of the parent process is not known, or when any process with a certain label is sufficient.

23.3.2.12. QueryCriteria setHasParentLabel()

This method specifies a process that must be the parent of every retrieved process. The setHasParentId() method can be used when only a specific process with a known ID will satisfy the criteria.

23.3.2.13. QueryCriteria setHasAssociatedSubprocessId()

This method indicates that the retrieved processes must all have a Subprocess with the specified ID. The setHasAssociatedSubprocessLabel() method can be used when the ID of the Subprocess is not known, or when any Subprocess with a certain label is sufficient.

23.3.2.14. QueryCriteria setHasAssociatedSubprocessLabel()

This method indicates that the retrieved processes must all have a Subprocess with the specified label. The setHasAssociatedSubprocessId() method can be used when only a specific process with a known ID will satisfy the criteria.

23.3.2.15. QueryCriteria setHasAssociatedOperandId()

This method indicates that the retrieved processes must all have an operand with the specified ID. The setHasAssociatedOperandLabel() method can be used when the ID of the operand is not known, or when any operand with a certain label is sufficient.

23.3.2.16. QueryCriteria setHasAssociatedOperandLabel()

This method indicates that the retrieved processes must all have an operand with the specified label. The setHasAssociatedOperandId() method can be used when only a specific operand with a known ID will satisfy the criteria.

23.3.2.17. QueryCriteria setHasAssociatedStatusId()

This method indicates that the retrieved processes must all have an operand with the specified ID. The setHasAssociatedOperandLabel() method can be used when the ID of the operand is not known, or when any operand with a certain label is sufficient. Use setHasStatusId() to specify a status in the set of current statuses for a process, rather than just an associated status.

23.3.2.18. QueryCriteria setHasAssociatedStatusLabel()

This method indicates that the retrieved processes must all have a status with the specified label. The setHasAssociatedStatusId() method can be used when only a specific operand with a known ID will satisfy the criteria. Use setHasStatusLabel() to specify a status in the set of current statuses for a process, rather than just an associated status.

23.3.2.19. QueryCriteria setHasAssociatedPolicyId()

This method indicates that the retrieved processes must all have a policy with the specified ID. The setHasAssociatedPolicyLabel() method can be used when the ID of the policy is not known, or when any policy with a certain label is sufficient.

23.3.2.20. QueryCriteria setHasAssociatedPolicyLabel()

This method indicates that the retrieved processes must all have a policy with the specified

label. The setHasAssociatedPolicyId() method can be used when only a specific policy with a known ID will satisfy the criteria.

23.3.2.21. QueryCriteria setDepth()

This method specifies the number of levels of Subprocesses to include with each retrieved process that meets the query criteria. If a depth of 0 is specified, then no Subprocesses will be retrieved. If a depth of -1 is specified, then all levels of Subprocesses will be retrieved. The default is for no Subprocesses to be retrieved, if the setDepth() method for a QueryCriteria object is not called.

23.3.2.22. QueryCriteria setIncludeParent()

This method specifies whether a query should return the parent process of each process meeting the query criteria.

23.3.2.23. QueryCriteria setIncludeOperands()

This method specifies whether the operands associated with each retrieved process should also be retrieved. Note that this does not include operands that are visible to a process, but associated with a parent or ancestor of the process.

23.3.2.24. QueryCriteria setIncludePolicies()

This method specifies whether the policies associated with each retrieved process should also be retrieved.

23.3.2.25. QueryCriteria setIncludeStatuses()

This method specifies whether the associated statuses and current statuses for each retrieved process should also be retrieved.

23.3.3. ReactorObjects

The ReactorObjects class is found in the `com.oakgrovesystems.reactor` package. It manages a collection of reactor objects that have been either retrieved from Reactor Engine or created directly by the application. Relationships between Reactor process mediation objects are maintained by indirect references, so the relationship can be preserved without retrieving the object itself from Reactor Engine.

The methods covered in this section manage the collection and provide access to process mediation objects.

- add()
- remove()
- clear()
- size()
- getObjects()
- getProcess()
- getProcesses()
- getParent()
- getSubprocess()
- getSubprocesses()
- getRootProcesses()
- getOperand()
- getOperands()
- getCurrentStatus()
- getCurrentStatuses()
- getStatus()
- getStatuses()
- getPolicies()
- getPolicy()

23.3.3.1. ReactorObjects add()

This method adds an object to the collection.

23.3.3.2. ReactorObjects remove()

This method returns an object from the collection.

23.3.3.3. ReactorObjects clear()

This method removes all objects from the collection.

23.3.3.4. ReactorObjects size()

This method returns the number of objects in the collection.

23.3.3.5. ReactorObjects getObjects()

This method returns a Collection object containing all the objects.

23.3.3.6. ReactorObjects getProcess()

Depending on its arguments, this method returns a process associated with the specified object, or identified by the specified ID or label path.

23.3.3.7. ReactorObjects getProcesses()

This method returns a Set object holding all processes managed by the ReactorObjects object.

23.3.3.8. ReactorObjects getParent()

This method returns the parent process of the specified process object.

23.3.3.9. ReactorObjects getSubprocess()

When passed a Process object, this method returns a Subprocess with the specified label.

23.3.3.10. ReactorObjects getSubprocesses()

This method returns a Set object containing all the Subprocesses of a specified process.

23.3.3.11. ReactorObjects getRootProcesses()

This method returns a Set object containing all the processes that have no parent process.

23.3.3.12. ReactorObjects getOperand()

Depending on its arguments, this method returns an operand associated with the specified process, or identified by the specified ID or label path.

23.3.3.13. ReactorObjects getOperands()

This method returns a Set object containing the operands associated with the specified process.

23.3.3.14. ReactorObjects getCurrentStatus()

This method takes a Process object and the label of a Status object. If one of the current statuses of the process has the specified label, then that Status object is returned. Otherwise, the method returns null.

23.3.3.15. ReactorObjects getCurrentStatuses()

This method returns a Set object containing the current statuses for the specified process. Use

the getStatuses() method to get the set of associated statuses.

23.3.3.16. ReactorObjects getStatus()

Depending on its arguments, this method returns a status associated with the specified process, or identified by the specified ID or label path.

23.3.3.17. ReactorObjects getStatuses()

This method returns a Set object containing the statuses associated with the specified process. Use getCurrentStatuses() to get the set of current statuses.

23.3.3.18. ReactorObjects getPolicy()

Depending on its arguments, this method returns a policy associated with the specified process, or identified by the specified ID or label path.

23.3.3.19. ReactorObjects getPolicies()

This method returns a Set object containing the policies associated with the specified process.