Space Details

Key:	PentahoDoc
Name:	BI Server Documentation - Latest
Description:	Latest version of the Pentaho BI Server
Creator (Creation Date):	admin (Nov 15, 2006)
Last Modifier (Mod. Date):	admin (Nov 27, 2006)

Available Pages

        - Chart Component
        - Chart Tag Reference
            - by-row
            - chart
            - chart-attributes
            - chart-background
            - chart-type
            - color
            - color-palette
            - dataset-type
            - domain-includes-zero
            - domain-label-rotation
            - domain-label-rotation-dir
            - domain-period-type
            - domain-sticky-zero
            - domain-title
            - domain-title-font
            - domain-vertical-tick-labels
            - dot-height
            - dot-width
            - height
            - is-3D
            - is-stacked
            - line-style
            - line-width
            - markers-visible
            - orientation
            - paramName
            - paramName2
            - plot-background
            - range-title
            - range-title-font
            - subtitle
            - title
            - title-font
            - title-position
            - url-template
            - width
        - TimeSeriesCollectionComponent
        - XYSeriesCollectionComponent
- Content Repository Cleaner
- Email
- Hello World
- Jasper Reports
- JavaScript

- JFree Reports
- Kettle
- MDX Query
- Prepared Components - Enabling Subreporting and Connection Sharing
- Printing
- Scheduling
  - Misfires
  - Triggers
- Secure Filter (Prompting)
- SQL Execute
- SQL Query

Creating Pentaho Solutions

This page last changed on Nov 28, 2006 by dmoran.

This guide describes the philosophy and tools for creating solutions to business problems using the Pentaho BI Platform. The Pentaho BI Platform integrates capabilities from the major application areas of the Pentaho BI Suite: Reporting, Analysis, Dashboard, Data Mining and Workflow. The integration includes security, scheduling, auditing, deployment, solution management and workflow capabilities.

This guide is broken up into the following parts:

- I - Solution Oriented Approach — The philosophy behind the Pentaho BI Platform and its solutions.
- II - Building Solutions — Technical description of the BI Platform's major components and how to create, deploy and manage solutions.
- III - Actions and Component Reference — Detailed reference for the components distributed with the Platform.

I - Solution Oriented Approach

In many years of helping customers create reports and analytical systems, we have encountered a similar situation many times. The scenario is always different, but the basic need is always the same: a report is delivered or a particular situation is encountered in the data and something specific needs to happen - a decision must be made, causes must be discovered, or a process must be started. In these cases the information presentation, analysis, and delivery (BI) is a part of a larger process. This process exists to solve the business problem: it is the solution.

To clarify:

- Often the solution to a business problem is a process that includes Business Intelligence.
- Therefore: the Business Intelligence, by itself, is not the solution to the problem.
- If Business Intelligence is part of the process, the Business Intelligence tools are, unavoidably, also part of the process.
- A Business Intelligence tool that does not understand processes, or how to be part of one, will be hard to integrate into the solution.

The Pentaho BI Platform is the first process-centric and solution-oriented Business Intelligence platform.

Sure we can throw in a little bold text and make it look grand but how do we back up a statement like that when other BI providers are claiming to be adding process-centric features?

Workflow at the Core

The Pentaho BI Platform has several options available for executing activities. The default execution engine is a built-in, lightweight, Business Flow Sequencer. This sequencer allows the solution developer to build solutions from collections of Business Flows that are generally linear and success oriented utilizing passed parameters and a minimum of looping. For example: run a query to find out which departments are over budget, run a budget report for each of those departments, and finally email each report to the department manager.

When the business requirements require user interaction, parallel tasks, deadlines and extensive error handling, there is built-in support for utilizing a comprehensive workflow engine. The workflow engine uses a standard language, XML Process Definition Language (XPDL), to execute the Business Flows within the system. An example of this type of solution would be a Human Resources new hire process where multiple departments have to be notified about the new hire, actions would need to be coordinated and a final task to verify all resources have been issued and all paperwork is complete can be verified and marked completed.

In the case where a solution needs to be coordinated externally, any Business Flow defined in the system is available as web services and return their results via SOAP packages. This allows actions to be coordinated via an orchestration technology such a BPEL workflow engine or a remote application.

Components may also be embedded directly into a custom Java application. This can be important when your solution needs to be part of an existing application or you just need complete programmatic control.

No matter what deployment option you choose:

- The platform understands the nature of processes because everything in it is one.
- The processes are defined in a standard process definition language that is externally viewable, editable, and customizable.
- There is no hidden business logic.

The platform is built on processes and process definitions.

Service-Oriented Architecture (SOA)

This is rapidly becoming a meaningless term with every application that accepts URLs claiming to have a SOA. When you design a system with a workflow engine as the conductor and director every activity in the system, every step of each process, every bubble in your process diagram must be implemented as a standalone, re-usable component that can be directed to execute the activity required.
This is not just an SOA, this is a Service-Implemented Architecture (SIA). Every activity in every process can be a web service because all activities only ever execute as services. They know no other invocation. The three rules to web services success are: invocation, invocation, invocation.

Services are the building blocks of automated business processes.

Process Integration

Every process and activity in the Pentaho BI Platform executes as a service. If you want to call a process or activity defined in the platform from a process executing in another system, you can. It's easy.

Every activity in the system understands how to be part of another process.

Rules, Rules, Rules

The definition of the platform processes are externally defined, but what about the rules that govern the workflow? XPDL has built in support for complex routing control, and we have added support for multiple rules engines so business logic can be integrated easily into the processes. Multiple rules engines are supported and required because it is unlikely the logic for every decision in every process can be defined easily by only one rules engine.

For example, the business rule to determine the credit analyst for a customer might be best described three different ways in three implementations:

- A simple piece of script: if (customerNo < 3000) return 'Bob' else return 'Mary'
- A complex chaining algorithm that bases the decision on the customer's current sales pipeline, service level agreement, lifetime value, payment history, industry segment, and location

- A call to a database or ERP system to lookup the analyst for the customer

If the credit analyst for each customer is stored as a record in an ERP system, trying to maintain the rule in a different system will be a redundant effort with additional cost, additional risk, and no added value.

Flexible business rules are a critical part of automated business processes.

Business Intelligence / Business Process Boundary

The line between business intelligence and business processes is flexible in the Pentaho BI Platform. This is because the line between business intelligence and business processes is blurry and should be up to you. If you have a BI platform that clearly defines the boundaries between it and your other systems, you probably have an application silo that is hard to integrate the way you need it to. It is your processes, your data, and your software.

- The default engine executing processes within the platform allows you to easily script a light-weight workflow
- When required all or part of the light-weight flow can be implemented with a full-featured workflow engine
- The Pentaho BI Platform includes multiple rules engines
- The Pentaho BI Platform activities are easily integrated into other processes

Case Study

Problem: When an employee shows up for work at a health care organization with an expired license, there are two outcomes. It may be noticed and a more costly agency worker must replace the employee until their license is renewed, or it is not noticed in which case a patient safety hazard and a liability risk occurs

Business goals: increase patient safety, reduce liability of unlicensed employees, and reduce money spent on agency staff covering for unlicensed employees.

Current process: Each manager maintains a list of license expirations for their department.

Proposed 'solution': Scheduled execution of a report from a central database that lists, by department, licenses held by each employee, and the expiration date of their current licenses.

Solution 1: Give them what they ask for

Create a 50 page report and deliver it to each department once a month.

Resulting Business Process:

- Running of report is not audited. If it does not happen when expected how long before someone realizes?

- Managers in each department are required to read the report and filter the information. Reports get lost, managers take vacations, and dates get misread.
- When managers spot upcoming license expirations they leave a note in the employee's mail box. Notes get lost or placed in wrong boxes.
- Employees try to schedule preparation, application and certification time. Schedule conflicts arise, preparation suffers.
- Employees fail certification with no time for further preparation or certification before license expiration.

This solution is incomplete because it only automates the information delivery, it does not help the real process that has to occur. The business goal is reached at best as a by-product of the reporting solution.

Solution 2: Give them what they need

- Create business rules that determine the lead time required for adequate preparation for each type of license and escalation paths for problem cases.
- Run an audited report every day or week that lists those employees within their lead time. For each employee initiate a defined license renewal process:
  1. Deliver the information electronically to both manager and employee
  2. Require electronic acknowledgment from both
  3. Direct employee to schedule preparation time
  4. Direct manger to approve schedule
  5. Require employee to enter certification test date
  6. Escalate warnings if insufficient re-test time has been allowed
  7. Require manager to validate new license
  8. Deliver notifications on certification failure to manager and scheduling application.
- Provide on-line, real-time reporting on the license renewal process
- Produce audit reports of monthly and quarterly performance

This solution solves the business problem.

Conclusion

In order to deliver this solution you need reporting and analysis tools that:

- Support the business rules needed
- Audit report execution and delivery of information
- Integrate seamlessly with a workflow system

You also need a workflow / business process engine that:

- Handles time-based escalations
- Audits execution of activities within the process
- Integrates with reporting and analysis tools

You also need to provide real-time and historical process performance reports

This is the Pentaho BI Platform.

The Pentaho BI Platform is uniquely process-centric and solution-oriented.

- It is process-centric because it is built ground-up to be process-based.
- It is solution-oriented because the solution for many business problems is a process, and the platform includes all the major components required to implement process-based solutions.

Creating Pentaho Solutions II - Building Solutions

II - Building Solutions

This page last changed on Dec 12, 2006 by dmoran.

I - Solution Oriented Approach | Creating Pentaho Solutions | III - Actions and Component Reference

This document describes the architecture of the Pentaho BI Platform and details the components and tools required to build solutions. It is intended for people interested in building solutions and creating content. It is also valuable for anyone who needs to interface with or develop portions of the Pentaho BI Platform.

To get the most out of this documentation, it is recommended that you have the Pentaho BI Suite (Pre-Configured Installation) and Pentaho Design Studio installed on your local machine. Many of the examples in this document refer to the sample solutions and data that come with the Pentaho BI Suite. Either version of the Pentaho BI Suite, Open and Pro, will work.

Many of the examples are illustrated by using the Pentaho Design Studio. Using the design studio along with this guide will save you from hand editing XML. A task that for some reason is not enjoyable by all people (or - so I have been told - many times.)

If you already have a working Pentaho Server and functioning Design Studio, you can skip the first section.

I - Solution Oriented Approach | Creating Pentaho Solutions | III - Actions and Component Reference

01. Configuring the BI Server and Design Studio

The quickest way to get started with the BI Platform is to download and run the Pentaho BI Platform Pre-Configured Install (PCI). For more information, see Getting Started with the BI Platform.

The Pentaho Design Studio provides a graphical environment for building, managing, and testing your solutions. It provides a collection of templates, editors and wizards to help create and maintain solutions. Many of the examples in this document refer to the Design Studio. For information, see studio: Getting Started with Design Studio.

Both the Design Studio and PCI are available from the Pentaho Downloads page

Setting up Design Studio to use the samples

At this point, you have either installed the standalone Design Studio or installed the Design Studio plug-in into Eclipse, and you have a working install of the Pentaho samples. You should have tested that the samples in **samples/getting-started** work and tried one or two reports in **samples/steel-wheels/reports** within your environment and with your browser. Your Pentaho BI Server is running and waiting for requests.

Tech Tip

The Design Studio is file-system based. All of the content that is being edited exists on or is available via the local machines file-system. This could include shared folders, nfs mounts, etc. This is why we recommend, for following this guide, having both the BI Server and Design Studio running on the same computer.

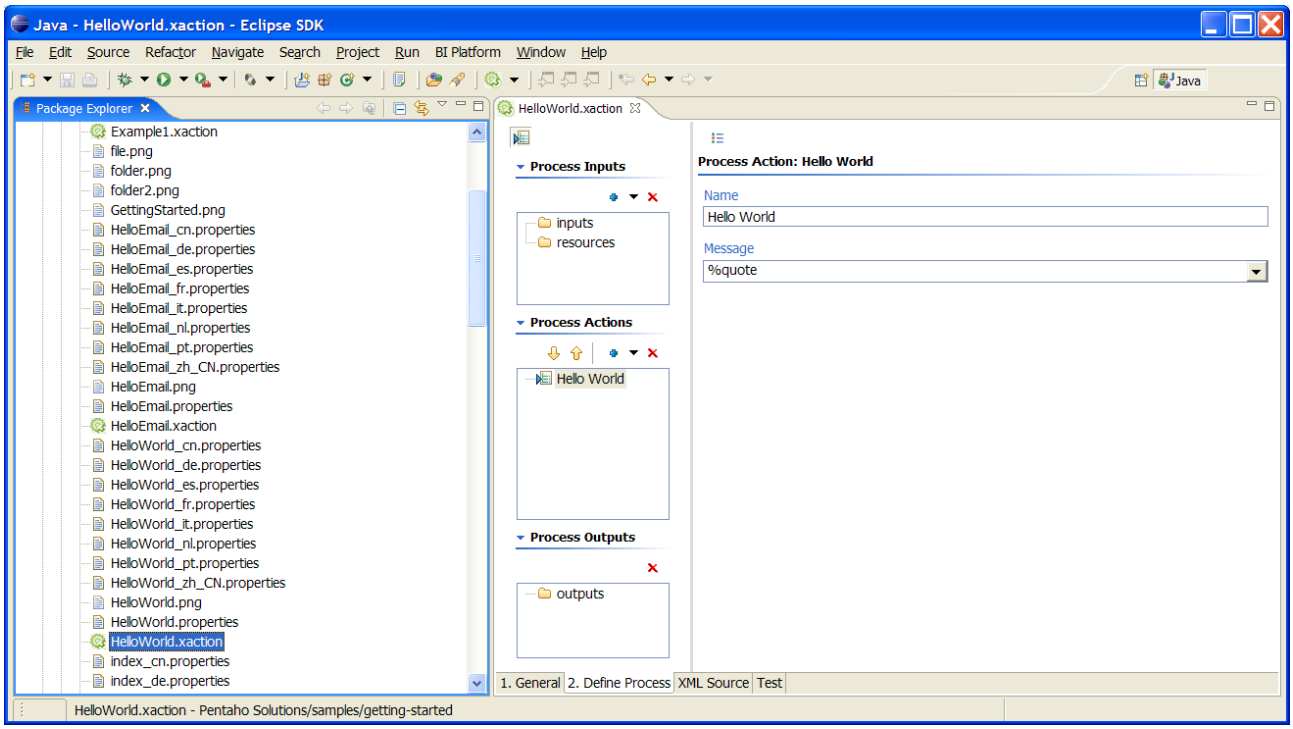
If you haven't done so already, start the Design Studio. If the welcome screen appears, close it by clicking on the X next to Welcome.

- Select **File->New->Project**.
- Select **Simple** from the New Project wizard
- Press the **Next>** button.
- Enter **Pentaho Solutions** as the project name. Although any name is fine, this document will refer to **Pentaho Solutions**
- Uncheck the **Use default** check box
- Browse to the **pentaho-solutions** directory. If you are using the PCI, this will be **/pentaho-demo/pentaho-solutions**.
- Select **Finish**.

You are now have a Design Studio project that is set up to edit and test your samples solution.

Browsing the Solution Repository

You should now see your Pentaho Solutions project displayed in the tree on the left side of the Design Studio. If you expand the solution folder you'll see plenty of files. These are the files that make up your solution and are managed within the Design Studio. Let's take a look at one to get a feel of what the Design Studio can do for us. Go ahead and in the left hand tree, open the **Pentaho Solutions/samples/getting-started** folder. Double-click on the **HelloWorld.xaction** file and the Action Sequence editor will open in the edit pane.



Verify that you can test from within the Design Studio

The **Test** tab at the bottom of the Design Studio is used for generating and testing Action Sequences. At this point, don't worry about what an Action Sequence means, it will be explained later. Make sure you're looking at the **HelloWorld.xaction** from the last section then click the test tab. Now let's test out the action sequence.

> **Tech Tip**
>
> Currently, the Design Studio uses the Pentaho BI Server to execute Action Sequences. When pressing the **Run** button, the Design Studio submits an HTTP request to the server and displays the result in the built in browser. The built in browser is usually the default browser on your computer.
>
> This is exactly the same as if you used your browser to navigate the PCI samples and clicked on that Action Sequence. It is why you are prompted to save changes to your Action Sequence when going to the test tab. And, it is why you see a URL next to the Run button.

- Make sure that the Pentaho Server URL points to your running BI Server.
- Press the Test Server button. If everything is set up properly, the Pentaho Demo home page should

appear.
- Now select the Run button to submit an HTTP request to the BI Server to execute the current action sequence. In the embedded browser window you should see "Hello World. Greetings from the Pentaho BI Platform."

Now let's change the message displayed by this action sequence and test our change to make sure that it works.

- Select the **Define Process** tab.
- Select **Hello World** in the **Process Actions** box.
- Change the text **%quote** in the **Message** text box to something else like - **I did it!**.
- Return to the **Test** tab. Select **Yes** to save if prompted to save.
- If you are using the Pro Pentaho Server...
    - The default configuration of the Pro BI Server doesn't run action sequences from the file system, but instead runs them from the Pro BI Server repository. You will need to tell the server to refresh the database from the file-system.
    - Navigate to the Admin page of your Pentaho BI Server using your browser.
    - Select the **Solution Repository** icon.
    - Select **Yes** when prompted **Are you sure you want to do this?**
- Select Run.

You should now see the new message - **Hello World - I did it!**.

Tech Tip

The original message reads **%quote**. This notation is used to internationalize action sequences. There are HelloWorld_xx.properties files in the same directory as the action sequence we're currently modifying. Each file has the strings used by this action sequence translated into the appropriate language. Within the Design Studio, you can double click the HelloWorld_en.properties file and find the string assigned to **quote** and change it to read **I did it!**. For any strings starting with **%**, the server will first look for a local properties file, walk the path back to the top of the solution tree looking for the correct **.properties** file, and finally, use the text it as is.

If you successfully changed and tested Hello World; congratulations, you are ready to move on.

I had issues

Sorry you are having trouble. The most likely problem is that the **Design Studio** and **BI Server** are not pointing to the same location for the solution.

TODO

- Add additional troubleshooting steps
- Write a set of the steps for each bullet item below

- Use the browser to navigate the solution directly to see if it is a caching issue.
- Verify that the folder path in the **web.xml** matches the solution directory and the Design Studio project directory

- Open the Action Sequence in a text editor make sure change happened
- Others?

II - Building Solutions 02. Terminology

02. Terminology

This page last changed on Jun 05, 2007 by bseyler.

01. Configuring the BI Server and Design Studio | II - Building Solutions | 03. Architecture

---

TODO

I think we should move some or all of this to a master glossary somewhere at the top level and include or link to it. It might be good to break the terms up into categories like products, Design Studio related, BI Server related etc...

The list here is not meant to be inclusive, just relevant to this document. The frivolous text at the top would make more sense in a top level glossary and may not be palatable by marketing but I like acknowledging the difficulty of naming and maybe getting some input from the community on where we can improve.

-Doug

The Pentaho Project has many, many parts and can be used in a many different configurations. A lot of the names are similar or similar in meaning. Terms like *Reporting* can mean different things depending on context. For example: *Pentaho Reporting* is the name of the report server formerly known as jFreeReports. There is a *Reporting Pillar* that represents all of the components, APIs and applications related to creating, deploying, executing and distributing reports. You can design a Report using the *Report Designer* or *Report Design Wizard* and you can deploy a report using the *Pentaho BI Server.* There will soon be a *Web Based Query and Reporting* module. To confuse things a little further, you can deploy BIRT and JasperReports reports via the server too. Then, for our international friends... all those names need to be translated.

Naming is never easy, and unfortunately, bad names are often hard to retire. With all that said, this section will define some of the most important names that will be used throughout this document and hopefully will not conflict with other documents.

Common Terms Used Within Pentaho

Pentaho BI Platform Project	The open source project focused on delivery of platform infrastructure services, that also provides integration of Pentaho's end user and data integration capabilities. The Pentaho BI Platform project includes capabilities like security, integration, APIs, scheduling, and workflow.
Pentaho BI Pillar	Also referred to as a **Pentaho module** A grouping of products related by functionality. The 6 Pillars of the Pentaho BI Project are: Reporting, Analysis, Dashboards, Data Mining, Data Integration and BI Platform.

Pentaho BI Platform	The applications, APIs and components that support Pentaho's end-user reporting, analysis, and dashboard capabilities with back-end security, integration, scheduling, and workflow capabilities.
Pentaho Open BI Suite	This is the all-inclusive, external name for Pentaho's open source capabilities, including Reporting, Analysis, Dashboards, Data Mining, Data Integration, and a BI Platform.
Pre-Configured Install (PCI)	It is a **Pentaho BI Platform** deployed into a pre-installed and configured JBoss Application Server. In addition, it contains ready to use solution repositories including sets of reports demonstrating functionality and is meant to be a quick start demo.
Pentaho BI Server	This is the J2EE application that runs within a host application server and services user requests. The term is used to refer to the server portion of the platform without regard to how or where it is deployed.
XMI	XML Metadata Interchange: The XML Metadata Interchange (XMI) is an OMG standard for exchanging metadata information via Extensible Markup Language (XML).
Pro	"**Pro"** is a legacy terms that used to describe closed-source capabilities that were only available to paying customers. When referencing a software build, Pentaho now uses the term **"Subscription Edition"** to describe the build. And the additional closed-source features included in that build are now known as Pentaho Management Services.
Design Studio	A stand alone application that hosts a set of Pentaho plug-ins used for creating, testing and administering content for the Pentaho Project. It is an Eclipse http://eclipse.org workbench, pre-configured and customized for Pentaho. Currently there is only one plug-in; the **Action Sequence Editor.** In many cases people use the **Design Studio** to refer to the this plug-in. Future releases will provide more plug-ins and more capabilities. The goal is to have it be the client UI for as much of the administration and content creation as possible. This is considered to be an Administrator, Developer or Content Creator tool. Not a typical end user tool.
Action Sequence Editor	The Eclipse Plug-in that allows people to generate Action Sequences (a script that run within the **Pentaho BI Platform.**)
WAQR	Pronounce "wacker". Web Ad hoc Query and Reporting

Pimper	a.k.a. The PME, the Pentaho Metadata Editor
SWAG	Scientific Wild Assed Guess
CWM	Common Warehouse Metamodel

BI Platform Specific Terms

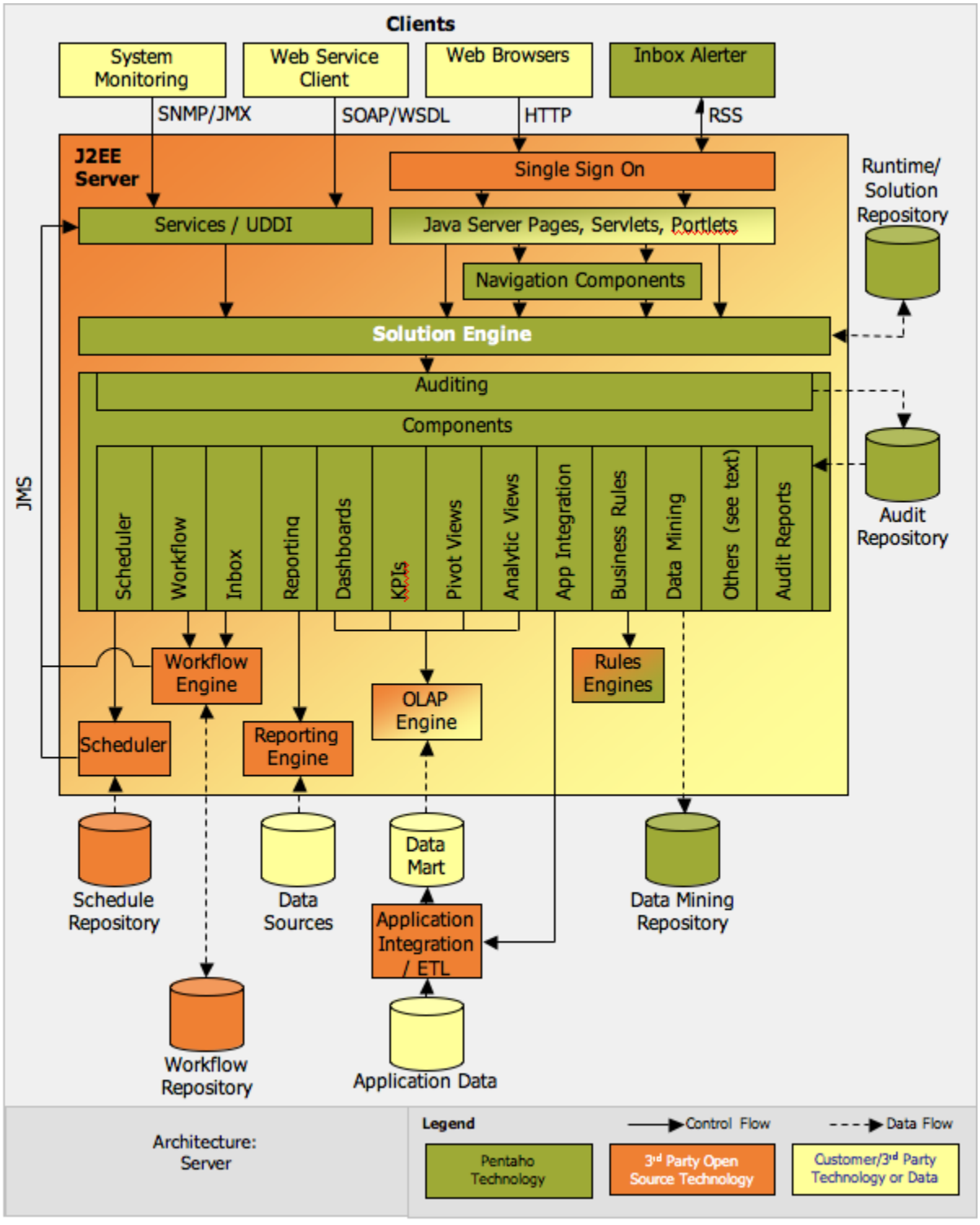
Solution	A **solution** consists of a collection of documents (files) that collectively define the processes and activities that are the system's part in implementing a solution to a business problem. These documents include **Action Sequences**, workflow process definitions, report definitions, images, rules, queries etc. A **solution** is represented in the file system as the top level folder.
Solutions Folder	Refers to the top level folder containing all of the solutions available to the BI Server. In the **DB Based Repository**, this is the top level of the solution hierarchy
Solution Repository	The location where **solutions** and the metadata they rely on is stored and maintained. Requests made to the platform to have actions executed rely on the action being defined in the Solution Repository. There are two implementations of the **solution repository** - The **file-based solution repository** and the **DB based solution repository** The **DB based solution repository** is only available in **Pro**.
Solution Engine	The software that retrieves the definition of an action from the Solution Repository and directs its execution.
Component	The component layer is an API that provides a standard interface between the solution engine and the application that executes business logic. A component may contain all of the code required to perform a task or may just be an interface to another application or system. Data and instructions to the component are provided via an Action Definition.
Action Definition	An XML definition specifying the parameters, resources and settings required for the execution of a task within a single component. The Action Definition defines which component to call, what data to pass into and receive from the component and any component specific information required. An action definition is not a stand alone document; it is a part of an Action Sequence.
Action Sequence	An XML document that defines the interaction between one or more Action Definitions. It defines

	the smallest complete task that the Solution Engine can perform. When the Solution Engine is told to execute - it is given an Action Sequence document to execute. The execution of the Action Sequence can be completed autonomously or may execute as part of another Action Sequence. Action Sequence Definitions are stored in the Solution Repository.
Runtime Context	Action Sequences are transformed from XML by the solution engine into objects that are interpreted by the Runtime Context. The Runtime Context maintains a contract between the Solution Engine and the Action Sequence and enforces a contract between the Action Sequence and the components.
PMD	Pentaho MetaData
PME	Pentaho Metadata Editor
WAQR	Web Ad hoc Query and Reporting
MQL	MetaData Query Language
MDR	MetaData Repository

01. Configuring the BI Server and Design Studio | II - Building Solutions | 03. Architecture

03. Architecture

The architecture diagram below shows the relationship between the major components of the BI Server and it's interfaces with the outside world. The heart of the server is the Solution Engine. The Solution Engine is the focal point for activity within the Pentaho BI Platform. It "sits" between the outside world - Web Client, Services, System Monitor etc. and the Component Layer. Requests to do work come into the solution engine and are routed to the appropriate component or components for execution.



Action Sequences

An Action Sequence is an XML document that defines the smallest complete task that the solution engine can perform. It is executed by a very lightweight process flow engine and defines the order of execution of one or more the components of the Pentaho BI Platform. We avoid calling this a process flow because it is missing many of the capabilities of a true process flow engine. It is good for sequencing small, linear, success oriented tasks like reporting and bursting. It has the ability to loop through a result set, call

another Action Sequence and conditionally execute components. The Action Sequence document should have a ".xaction" suffix.

Solutions and the Solution Repository

A solution is not a single document; it's a collection of documents. It's a logical grouping of Action Sequences and the resources they require. The grouping is maintained by the Solution Repository. You can see the structure of the solution repository by navigating to the pentaho-solution directory in the top level PCI install directory. The default location is: /pentaho-demo/pentaho-solutions. All folders within this directory with the exception of **system** are solution folders. Solution folders may contain any numbers of folders, those folders may contain any number of folders and so on. This is also true for the DB based Repository which actually models a file-system.

!SolutionFilesystem.png|align=left!In this view of the solutions folder, there are 3 solutions defined, **admin**, **samples**, and **test.** The **system** folder is not a solution. It is where system configuration information, component specific settings and resources are located.

The HelloWorld Action Sequence is located in the **samples** solution with a path of **getting-started** and an Action Sequence name of **HelloWorld.xaction**. This illustrates the three part address that is used to locate an Action Sequence within the repository: ***solution id***, ***path*** and ***Action Sequence name***. With this kind of naming, it is possible to group Action Sequences in any manner desired; by department or role etc. Use whatever structure makes sense for that solution.

The system Folder

As mentioned earlier, the **system** folder is a special case. One of the most important files it contains is the **pentaho.xml** configuration file. The **pentaho.xml** file contains system wide configuration settings for the Pentaho BI Platform. See the pentaho.xml Reference for more information.

Most of the directories under **system** are used by individual components.

Directory	Description
BIRT	Settings, Configuration files and pluins used by the BIRT Reporting component.
content	The storage directory for content generated by Action Sequences like Reports, PDF's, HTML pages etc. This is also known as the content repository.
custom	Location where the UI templates are stored for both the PCI navigation and default and custom parameter pages. The XSL's used to generate the PCI navigation UI is also stored here.
dtd	The DTD's used by the Platform are located here.
google	Contains settings like license key for Google maps integration.
hibernate	Location of the hibernate-jboss-managed.xml.

jasperreports	JasperReports Configuration file. Compiled JasperReports are also stored here.
kettle	Kettle settings.xml.
logs	Components can store any log files they generate here.
olap	Mondrian stores it's data source information here.
quartz	Location for the Quartz properties file.
shark	Enhydra Shark workflow engine working directory containing configuration, logs and repository.
simple-jndi	JNDI settings for the client tools.
smtp-email	Server email configuration.
test-suite	Configuration settings for the Test Manager interface.
tmp	Temporary files that need to be available via URL call like images or generated charts.

The test solution

The **test** solution contains Action Sequences and JUnit tests used to test functionality of parts of the BI Server. The JUnit tests are used during the nightly builds to perform regression testing.

TODO

Document the JUnit tests and the Test Manager stuff.

The Runtime Context

When requested to run, the solution, path and name of an Action Sequence is passed to the Solution Engine for execution. The Solution Engine retrieves the Action Sequence from the Solution Repository and creates a runtime environment. This runtime environment, called the Runtime Context, is where the Action Sequence is executed step by step.

TODO

Finish this section

Putting it all together

04. Action Sequences

This page last changed on Dec 13, 2006 by dmoran.

03. Architecture	II - Building Solutions	05. Integrating Pentaho Reports (JFreeReports)

The Action Sequence is an XML document that defines the smallest complete task that the solution engine can perform. It is executed by a very lightweight process flow engine and defines the order of execution of one or more the components of the Pentaho BI Platform. We avoid calling this a process flow because it is missing many of the capabilities of a true process flow engine. It is good for sequencing small, linear, success oriented tasks like reporting and bursting. It has the ability to loop through a result set, call another Action Sequence and conditionally execute components. The Action Sequence document should have a ".xaction" suffix.

03. Architecture	II - Building Solutions	05. Integrating Pentaho Reports (JFreeReports)

01 - Anatomy of an Action Sequence

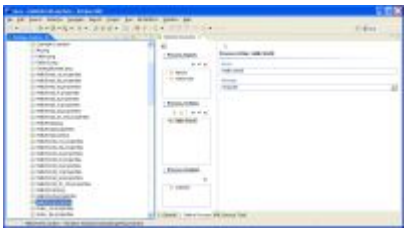
This page last changed on Jun 05, 2007 by bseyler.

04. Action Sequences 02 - Creating and Editing an Action Sequence

The easiest way to explore an Action Sequence is via the Action Sequence Editor plugin in the Pentaho Design Studio. The Action Sequence editor has four tabs along the bottom: General, Define Process, XML Source and Test. The function of each tab will be discussed in more detail later, their basic functions are:

- **1. General** - Basic properties like title, help etc.
- **2. Define Process** - Defines the inputs, outputs, resources required by the Action Sequence and allows you to program the interactions between the Action Sequence and the Pentaho Components
- **XML Source** - The raw XML that the editor is generating
- **Test** - Interface for executing the Action Sequence on the Pentaho BI Server

Click through each tab to get familiar with the editor. Check out the XML Source tab to get an idea of what the editor is saving you from. Now let's look a bit more closely at the HelloWorld.xaction.



General Information

As we mentioned earlier the "General" tab contains some general information about the action sequence, such as the title, author, icon, description, and help text to be displayed in the browser window (as shown below). Notice that the design studio shows the title for action sequence to be "%title". The "%" indicates that this is the name of a string whose value is defined in a properties file with the same name is the same as the action sequence. In this case the property file is named HelloWorld.properties. This is how action sequences accommodate internationalization. . Additionally you can indicate the logging level you would like to use for this action sequence. Logged messages will appear in the pentaho-demo/jboss/server/default/log/server.log file. If you're having problems getting your action sequences working, the log file is a good place to look for clues as to what the problem might be.

Inputs and Resources

Now press the "Define Process" tab. You should see a section labeled "Process Inputs" which lists the inputs and resources used by the action sequence. The inputs are the pieces of information the action sequence will need from the outside world when it runs. They can come from four sources; runtime, request, session, global and default. Runtime parameters are parameters that are stored in the Runtime Context. Remember, the Runtime Context stores the inputs and outputs from previous instances and makes them available to future executions of the same runtime instance id. Request parameters are the name-value pairs specified on a URL. Session parameters are variables that are stored in the user's session and may contain unique values for each user. Global parameters are similar to session parameters except they have the same values for all users. Default values can be specified for each input

and in the Action Sequence document and are used as a last resort.

Session and Global parameters can be used to provide secure filtering of data within the Action Sequence. A session parameter gets initialized by executing an action sequence when the user logs onto the system. The Action Sequence called upon login can be set up to perform a query using the user's login name in the where clause. The result is stored in the user's session and is available to subsequent Action Sequences. Global parameters are initialized when the system starts up and are available for all users. See the "Securing Data Access with Session and Global Filters" document for information on how to set up the filters and use them.

There are two implicit inputs **instance-id** and **solution-id** that are always available and do not need to be specified as inputs or outputs. They are the... well I'm sure you can guess what they are.

Resources are the files needed by the action sequence to complete its job. For example: if the action sequence is going to run a JFree report, one of the resources would be the location of the JFree report definition file.

Using the Design Studio let's take a look at some examples of inputs and resources. Browse to the samples/reporting directory in your "Pentaho Solutions" project and double-click on the JFree_Quad.xaction. Select each of the process inputs to view the details about each of the inputs and resources used by this action sequence.

Outputs

The action sequence outputs are what the action sequence will leave behind when it's complete. Outputs can have three destinations: runtime, session, or content. The first two destinations correspond to the input sources discussed above. The third destination indicates that the output will be put in the http response header or content.

Flow Control

The Action Sequence is not meant to be a replacement for workflow, that being said, there are two ways to control the sequence of execution; loops and conditions. An Action Sequence can execute a group of actions multiple times. The most common usage is to perform the set of actions once for each row in a query result set. The data types that can be specified for a loop are string-list, result-set and property-map-list. Conditional execution can be specified.

A group of actions can also be executed conditionally. The condition that will be evaluated for true is based on a JavaScript result.

Actions (Components)

Within the design studio open the samples/bursting/BurstActionSequence.xaction. What you see in the Process Actions section is a list of all the actions to be performed by this action sequence. Note that the order is important here. The topmost action will be run first, followed by the one below it, and so on. The second action, the one that starts with "Action Loop" probably deserves special mention. It's a loop action that will perform the actions it contains multiple times, depending on what it's set to loop on. In this case

it looks like there are five actions contained in the loop. Click on the first action in the list.

On the right side you can view the action details. You'll notice that there is a place for entering a brief description for the action. It's not necessary to enter anything here, but it's a good idea, as it makes the action sequence much easier to read. Each component has its own editor. Since this action uses the SQL query component, there is an area to specify the database connection, the query, and the expected contents of the query result. Now lets click on the "+" sign next to this action in the Process Actions tree. Notice that there are four items listed under the action. These are the outputs from this action. The rule-result output tells the system where the results of the query are stored. The remaining three outputs correspond to particular columns within the rule-result output. Other actions that follow can use these outputs as their inputs. So, one action can leave outputs that following actions can use as inputs. Additionally each action has available to it the action sequence inputs we discussed earlier. The idea is that each little action has something it can do really well. It takes in some input does some work and leaves some output for some other action(s) to use. Your job is to tie these individual actions together to do something meaningful.

Let's now take a quick look at each of the actions in this action sequence and see how they're working together to get something useful done. As we go through this don't get tied up in all the little details. The idea is to get a feel for how the actions work together. Later we'll learn about the details of each individual type of action.

Let's start with the first action in the actions tree. It performs a SQL query to extract some region, manager, and email information from a database. As mentioned earlier, it leaves some outputs behind. The query results are saved in an output called rule-result, and the other three outputs tell the world the column names for information in the results. Anytime you run a SQL query action make sure you include the column names in the output. That way other actions know what data is available in the rule result.

Next is the action loop. If you select it you'll notice that there isn't a whole lot to it. One notable point is that whenever you see "<" and ">" around a string it indicates that a parameter is being referenced. In this case the parameter is "rule-result". It's not by coincidence that this happens to be the name of the output from the preceding SQL Query action. This is an example of an action using the outputs from a previous action. So the five actions within the loop will each be performed once for each row that was returned in the outputs of the previous query action.

The next three actions are all similar. They're each string formatting actions. They take some input strings, place them into a formatted message, and leave the formatted string as an output. Basically they get things in order for the last two actions. If you click on the "+" sign next to each of these actions in the Process Actions tree, you'll see the outputs that each leaves behind.

Click on the action titled "Generate the report". This action will be generating a JFree report. In the configuration section you'll find that the JFree report specification and report format are being referenced using parameter named "report-definition" and "output-type" respectively. Notice both of these parameters are defined under the Process Inputs. Additionally the configuration section contains the database connection and query information that will be passed to the JFree report. Note that since these values are not enclosed within "<>" they are not parameter names, but are constant values. Finally you'll notice that the report is being saved in an output called "report-output". So we've generated this report and it's sitting in an action output parameter called "report-output". Now what?

Select the last action in the sequence. The name says it all. This action will email the report to the manager of the region for which the JFree report was generated. Take a look at the configuration details and you'll see how this action ties all the pieces together to send report off as an attachment.

Again don't be too concerned if you don't understand every detail. Each type of action has its own set of inputs and outputs. Once you get familiar with them you'll soon be putting them together to do all kinds of useful stuff.

---

04. Action Sequences

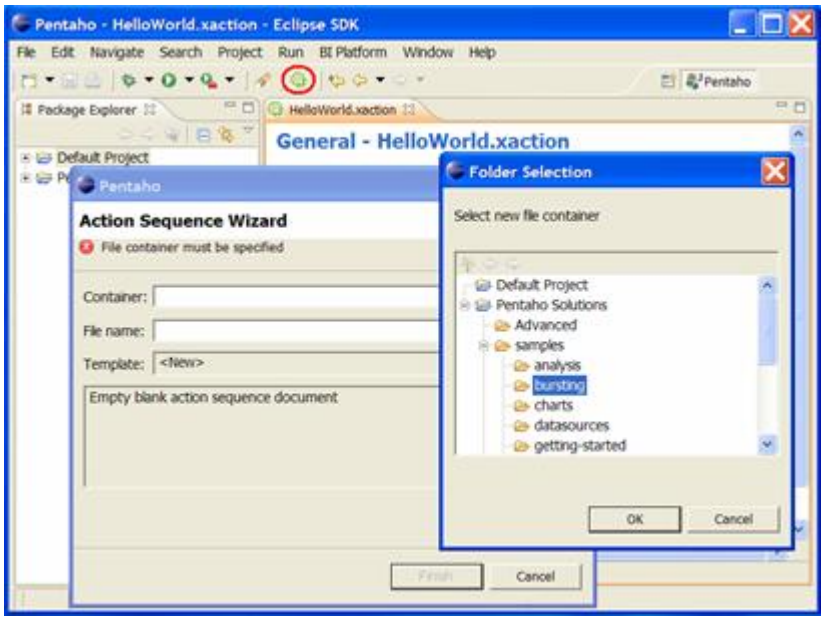
02 - Creating and Editing an Action Sequence

02 - Creating and Editing an Action Sequence

This page last changed on Dec 13, 2006 by dmoran.

01 - Anatomy of an Action Sequence | 04. Action Sequences | 03 - Executing an Action Sequence

Select the Pentaho Design Studio icon (circled in red) and choose "New Action Sequence". Then select the Container Browse button, and choose the "getting-started" folder in the PCI then select OK. Now give your new action sequence a name. Name your action sequence (make sure it ends with ".xaction") and from the "Template" pull-down select "Sample Burst Action" and press the Finish button.



Templates are a great way to get a jump start on building action sequences. A good bit of the information is already filled in for you. All you have to do is fill in the blanks. Feel free to build your own reusable action sequences. When you're done, save them in the PentahoDesign Studio\plugins\org.pentaho.designstudio.editors.actionsequence_x.x.x.x/templates directory. Next time you start the Action Sequence Wizard, your action sequence will be one of the available templates

A burst report is a report that is run multiple times for multiple people. It is most useful if there is a parameter that will filter data specifically for each person. In this example, we do a query to get a list of managers, their email address and the region for which they are responsible. We loop through the result-set and generate a report filtered by the region. A personalized message is created and the report is emailed to the responsible manager.

On the general tab enter a title and description. Notice that the template we used didn't specify an icon. Let's assign one by clicking on the "Browse...", and selecting the samples/bursting/BurstActionSequence.png file. Now choose File->Save and we'll move on.

Now press the "Define Process" tab. In brief this action sequence is going to run a report, then send a report to the each manager in the form of an attached Excel spreadsheet. Let's make a few changes. We'll send a second email to all our managers and let them all know how cool this action sequence stuff is.

In the Process Actions, tree right click on the "Email the report" action. Then select Add->Email. A new email action will be added as the last step in the loop, and the details of the newly created action will appear to the right of the tree. Since we want to send the email to the same managers that received the reports, we'll use the same email address used in the previous email action. Expand the "Lookup the region..." action in the Process Actions tree by clicking on the "+" sign next to the action. Now press and hold down the right mouse button over the "EMAIL" output listed under this action in the tree and drag the mouse over the field labeled "To" on the right hand side, then release the mouse button. A string labeled "<EMAIL>" should appear with the field indicating that the value of the "EMAIL" parameter will be used as the email destination. Let's fill in the "From" address by using the pull-down menu to the right of the "From" field and selecting "<from>". For the sake of example let's CC the email to some hard coded email address. Click in the CC field and type in the email address of your choice. Note that since this is occurring within a loop the person being CC'd will receive the same email for each manager in the loop. So make sure you CC it to someone you really want to annoy. Finally, type in the subject and text message of your choice. When you're done you should end up with something similar to what's shown below. If you run this action sequence your newly created email action should send your email to the appropriate parties.

The point of this little exercise was to give you a taste of how you create your own actions within an action sequence and how to tie the outputs of one action to the inputs of a later action.

---

01 - Anatomy of an Action Sequence	04. Action Sequences	03 - Executing an Action Sequence

03 - Executing an Action Sequence

There are several ways to execute a solution; via Design Studio, URL, Java Code or a Web Service call.

Design Studio

Click on the test tab on the HelloWorld.xaction editor. At the top of the test page, there is a field titled "Pentaho Server URL." If your pentaho server is running, enter the URL to your Pentaho BI Server which is likely to be http://localhost:8080/pentaho if you are running the PCI. Click the "Test Server" button and verify that you see the top level samples page displayed on the test page. Click on "Run" to execute the HelloWorld Action Sequence. You should see the familiar "Hello World. Greetings from the Pentaho BI Platform." message. In the unlikely event that you are not able to not see the Hello World message, make sure the server is running and that you typed the Server URL correctly. Verify that you can run the samples from your browser. If all else fails, try checking the Design Studio forum at www.pentaho.org.

URL

The samples that come with the preconfigured install are launched via URL using the ViewAction (org.pentaho.ui.servlet.ViewAction) servlet. The following URL will launch the HelloWorld Action Sequence:

http://localhost:8080/pentaho/ViewAction?&solution=samples&path=getting-started&action=HelloWorld.xaction

The result returned depends on the Action Sequence Document. You may get a report to view, a text message or just "Action Successful." The following parameters can be entered on the URL:

- solution, path, action* - The location of the Action Sequence document to load.
- instance_id* - The instance Id of a previous Runtime Context
- debug* - set to "true" in order to have debug information written to the execution log.

Web Service Call

In the "Settings and Services" group of the samples that come with the preconfigured install is a Web Service Example. It is still a URL call, this time to the servlet ServiceAction (org.pentaho.ui.servlet.HttpWebService). The following URL will launch the HelloWorld Action Sequence:

http://localhost:8080/pentaho/ServiceAction?solution=samples&path=getting-started&action=HelloWorld.xaction

In this case, the result returned is an XML SOAP Response. The following parameters can be entered on the URL:

- ° solution, path, action* - The location of the Action Sequence document to load.
  ° instance_id* - The instance Id of a previous Runtime Context
  ° debug* - set to "true" in order to have debug information written to the execution log.

Java Call

An Action Sequence can be executed directly from a Java application. For an example of how to do this, open the Java file "org.pentaho.test.RuntimeTest.java" and look at the JUnit test for HelloWorld. This class code can be found by accessing the Pentaho public repository at svn://source.pentaho.org/svnroot.

Action Sequence Recap

The inputs, outputs and resources in the Action Sequence header define a contract between the Action Sequence and the outside world. The Sequence requires the specified inputs and resources to be passed in and will return the specified outputs.

The action-definition defines a contract between each component and the Action Sequence. The action-inputs and action-resources define the parameters that a component requires to execute. The action-outputs define what parameters will be available after the component completes executing. Outputs from one component can be used as inputs to another component. The mapping attribute of the action-inputs allow outputs from one component that have different names to be used as inputs to another component.

Specifying the input/output relationships and their data types allows the system to validate an Action Sequence or set of Action Sequences without actually executing the components. A complete solution can be validated and "locked down" to prevent modification of the Action Sequence documents and eliminate errors due to "broken links" between these documents.

---

02 - Creating and Editing an Action Sequence | 04. Action Sequences | 04 - Action Sequence XML

04 - Action Sequence XML

This section explains the XML that makes up an Action Sequence. Even if you never, ever, ever want to see XML or have anything to do with it (I don't know why anyone would ever think such a thought), there is still some valuable information in this section and it is worth reading or at least skimming.

In most cases the Design Studio will suffice for building Action Sequences. There are times, however, when you may need to work directly with the Action Sequence XML. Some examples of when you might need to edit the XML directly:

- There is usually a time lag between when a new feature is added to the platform and when the Design Studio GUI is ready.
- You may need to work around a Design Studio bug.
- Sometimes the UI for a rarely used feature or special case property would be more confusing than editing the XML
- Custom components or components that you create may not have a corresponding UI
- Copying and pasting pieces from one Acton Sequence to another
- If you are developing your own components you will need to understand the XML

Even when you do need to edit the XML directly, you can still use the Design Studio. Simply click on the **XML Source** tab and go. When switching off the XML Tab, the Action Sequence will be checked for valid XML and Action Sequence structure.

Custom components and components that the Design Studio does not understand will still be visible and editable in the **Process Actions** pane on the **2. Define Process** tab. These components will be displayed with a generic UI that lets you edit inputs, outputs and resources and has a text box for editing the component XML.

1 - Example Action Sequence XML

This is a listing of the Example1.xaction Action Sequence Document.

```
<action-sequence>
   <name>Example1.xaction</name>

   <!-- some header nodes deleted -->

   <inputs>
      <region type="string">
         <default-value>Central</default-value>
         <sources>
            <request>REGION</request>
            <session>aRegion</session>
         </sources>
      </region>

      <from type="string">
         <default-value>joepentaho@pentaho.org</default-value>
      </from>

      <subject type="string">
         <default-value>Pentaho Example1</default-value>
      </subject>

      <message-plain type="string">
         <default-value>
                 This is an email from the Pentaho BI Platform - Example1
         </default-value>
      </message-plain>
   </inputs>

   <outputs/>

   <resources/>

   <actions>
      <action-definition>
         <action-inputs>
            <region type="string"/>
         </action-inputs>

         <action-outputs>
            <rule_result type="string"/>
         </action-outputs>

         <component-name>JavascriptRule</component-name>
         <component-definition>
            <script>
               <![CDATA[
                  if ( "Central".equals( region ) ) {
                     rule_result = "joe@pentaho.org";
                  }
                  else {
                     rule_result = "suzy@pentaho.org";
                  }
              ]]>
            </script>
         </component-definition>
      </action-definition>

      <action-definition>
         <action-inputs>
            <to type="string" mapping="rule_result"/>
            <from type="string"/>
            <subject type="string"/>
```

```
            <message-plain type="string"/>
         </action-inputs>
         <component-name>EmailComponent</component-name>
         <component-definition/>
      </action-definition>

   </actions>
</action-sequence>
```

In this example, the Action Sequence has 4 inputs: **region**, **from**, **subject** and **message-plain**. For region, the type is defined as a string; it has a default value of **Central** and may come from one of two sources; **request** and **session**. When the RuntimeContext resolves the region input at runtime, it will first look in the request (most likely an http request.) If it doesn't find it in the request, it will look in the session (most likely the http session.) If it is not available in the session, the default value will be used. The order that the sources are specified in the XML document is the order that they will be searched. The default is always used as a last resort.

The other inputs only specify a default value. This is analogous to hard coding the parameters to a constant value. Since the output of this Action Sequence is an email, no output parameters will be set.

There are 2 action-definition nodes for this sequence. The first defines a JavaScript rule and requires a region parameter; it will create a new parameter called **rule_result**. This new parameter will be made available to other action-definition nodes in the sequence.

Without getting too deep into the workings of the JavaScript rule, the script defined in the component-definition will be executed and will set the value of **rule_result** to the appropriate email address based on the value of region.

When the first **<action-definition>** completes, the second will execute. It defines an interaction with the Email component. The email component requires 4 action-inputs: **to**, **from**, **subject** and **message-plain**. You may have noticed that the action-inputs: **from**, **subject** and **message-plain** are specified in the inputs section of the Action Sequence header. The RuntimeContext will take the values from there and hand them to the Email Component just as it handed **region** to the JavaScript Component. The source of the **to** action-input isn't directly defined. It is indirectly defined with the **mapping** attribute. This attribute is telling the RuntimeContext to use the value from **rule_result** that was generated by the JavaScript rule and use it as the components **to** input.

---

04 - Action Sequence XML 2 - Defining Inputs

2 - Defining Inputs

There are three types of parameters that Action Sequence documents understand; **inputs**, **outputs** and **resources**. Inputs and outputs are variables of a specific data type like string or property-map (see ADD LINK TO valid data types.) Resources are similar to inputs except they specify a mime type and path. A default value cannot be specified for resources. Typically resources represent large amounts of data like report definitions or images (see ADD LINK TO for valid resource types.)

Parameters can come from four sources; **runtime**, **request**, **session**, **global** and **default**.

- Runtime parameters are parameters that are stored in the Runtime Context. Remember, the Runtime Context stores the inputs and outputs from previous instances and makes them available to future executions of the same runtime instance id.
- Request parameters are the name-value pairs specified on a URL
- Session parameters are variables that are stored in the user's session and may contain unique values for each user.
- Global parameters are similar to session parameters except they have the same values for all users.
- Default values are specified in the Action Sequence document and are used as a last resort.

Session and Global parameters can be used to provide secure filtering of data within the Action Sequence. A session parameter gets initialized by executing an action sequence when the user logs onto the system. The Action Sequence called upon login can be set up to perform a query using the user's login name in the where clause. The result is stored in the user's session and is available to subsequent Action Sequences. Global parameters are initialized when the system starts up and are available for all users. See Using System Actions to Control Data Access for information on how to set up the filters and use them.

Here is an example if the inputs section of an Action Sequence document:

```
<inputs>
    <region type="string">
        <sources>
            <request>REGION</request>
            <runtime>aRegion</runtime>
        </sources>
        <default-value>Central</default-value>
    </region>
</inputs>
```

This example indicates that the Action Sequence document requires a parameter named **region** (case sensitive.) When executed, the Runtime Context will first look to see if there was a parameter named **REGION** in the request. If the Action Sequence was launched from a URL, and there was a parameter **REGION=xxx** specified, than this value (xxx) will be substituted for the **region** input. If it doesn't find the parameter in the request, it will look in its own runtime data for a parameter named **aRegion.** If it doesn't find it in the Runtime Context Data, the value **Central** will be used. The Runtime Context always looks in the sources in the order in which they are specified and takes the default last. If no default was specified, then the Action Sequence would throw an error and return.

There are two implicit parameters **instance-id** and **solution-id** that are always available and do not need to be specified as inputs or outputs. They are the... well I'm sure you can guess what they are.

[1 - Example Action Sequence XML](#) [04 - Action Sequence XML](#) [3 - Data Types](#)

3 - Data Types

The following data types are currently supported by the Pentaho BI Platform.

- content — Content is large chunk of data that is generated within a component.
- long — A Java Long Object.
- propert-map-list — A list of property maps of Java Strings.
- property-map — A property map of Java Strings.
- string — The standard stinky old Java String.
- string-list — A list of Java String Objects.

2 - Defining Inputs | 04 - Action Sequence XML | 4 - Resource Types

content

Content is large chunk of data that is generated within a component.

One example of content is a PDF file generated by the reporting component. You cannot specify a default value for content since it can be of any type and is represented internally as a byte stream.

Example:

This XML defines a content node named **attachment** that is expected to exist in the runtime context under the name **report-output**. In this example, the report component generated a document and stored it as **report-output**. The email component could embed this data as an attachment in an email.

```
<attachment type="content">
    <sources>
        <runtime>report-output</runtime>
    </sources>
</attachment>
```

long

This page last changed on Dec 13, 2006 by dmoran.

A Java Long Object.

Example:

This XML node defines a long with a default value of 25.

```
<amount type="long">
    <default-value>25</default-value>
</amount>
```

propert-map-list

This page last changed on Dec 13, 2006 by dmoran.

A list of property maps of Java Strings.

Example:

This XML node defines a property-map with the name **fruit-data** with 3 property-map sets. Items in the list are contained within <entry key="xxx"> nodes. Property map lists are sometimes used to store the result of a database query. Each property map in the list represents 1 row of data with the keys mapping to column names and the values mapping to data cells.

```
<fruit-data type="property-map-list">
    <default-value type="property-map-list">
        <property-map>
            <entry key="name">orange</entry>
            <entry key="color">orange</entry>
            <entry key="shape">sphere</entry>
            <entry key="texture">dimply</entry>
        </property-map>
        <property-map>
            <entry key="name">grapefruit</entry>
            <entry key="color">Yellow</entry>
            <entry key="shape">sphere</entry>
            <entry key="texture">dimply</entry>
        </property-map>
        <property-map>
            <entry key="name">cucumber</entry>
            <entry key="color">green</entry>
            <entry key="shape">ellipsoid</entry>
            <entry key="texture">smooth</entry>
        </property-map>
    </default-value>
</fruit-data>
```

property-map

This page last changed on Dec 13, 2006 by dmoran.

A property map of Java Strings.

Example:

This XML node defines a property-map with the name **veggie-data** with 4 name value pairs. Items in the list are contained within <entry key="xxx"> nodes. Property maps are sometimes used to represent a single row of data from a database query. The keys map to column names and the value maps to that column's data.

```
<veggie-data type="property-map ">
    <default-value type="property-map">
        <property-map>
            <entry key="name">carrot</entry>
            <entry key="color">orange</entry>
            <entry key="shape">cone</entry>
            <entry key="texture">bumpy</entry>
        </property-map>
    </default-value>
</veggie-data>
```

string

This page last changed on Dec 13, 2006 by dmoran.

The standard stinky old Java String.

Example:

This XML node defines a string with a default value of **Central**. The RuntimeContext will first look for an input parameter named **REGION** in the http request. It will then ask the session for an object named **aRegion.** If neither have a value it will create a string set to **Central**.

```
<region type="string">
    <sources>
        <request>REGION</request>
        <session>aRegion</session>
    </sources>
    <default-value>Central</default-value>
</region>
```

string-list

This page last changed on Dec 13, 2006 by dmoran.

A list of Java String Objects.

Example:

This XML node defines a string-list with the name **to-address** with 4 entries. Items in the list are contained within <list-item> nodes.

```
<to-address type="string-list">
    <default-value type="string-list">
        <list-item>joe.pentaho@pentaho.org</list-item>
        <list-item>admin@pentaho.org</list-item>
        <list-item>sales@pentaho.org</list-item>
        <list-item>noxidj@pentaho.org</list-item>
    </default-value>
</to-address >
```

5 - Actions

The Action Sequence document is the definition, the Runtime Context provides an execution environment and the Components are the business logic. A Component performs a single function, a group of related functions or is a bridge between the Pentaho BI Platform and an external application. The jFree Reports component is an example of a component that interfaces the platform to the jFree Reports reporting engine.

There are two major functions that a Component gets called to do - validate and execute. Validate is called to verify that the inputs are valid and all resources required to execute are available. Execute actually performs the action.

The action-definition node in the Action Sequence document defines how the component should function. It is the place to define and map inputs, outputs and any other metadata or settings the component will require when it is executed. The name of the component that executes the action definition is identified using the component-name node. The name is the name of a Java class that is dynamically loaded at runtime. When referring to the built in Pentaho components, the fully qualified name of the component should not be used, just the class name. For example, use EmailComponent instead of org.pentaho.plugin.email.EmailComponent.

Action-inputs

The action-inputs and action-resources define the parameters that will be passed into the component when it executes. Some components have required inputs that must be available or runtime error will occur. Some inputs may be specified but are optional. For example, the Email Component requires a **to** but the **cc** is optional. There are several ways to satisfy a required input. It can be provided as a passed parameter with the same name to the component. It can be mapped to a different name and passed to the component. It can be hard coded with a constant value or in some cases it could be prompted for. By default the email component should have an action input that looks like:

```
<action-inputs>
    <to type="string" />
        . . .
</action-inputs>
```

Action Input Mapping

What happens if a component requires an input named **x** and I want to pass it a suitable parameter named **y**? You can map any action-input to a different name using the **mapping** attribute. Again, using our trusty Email Component example, let's assume we have a JavaScript rule that performs a query and returns the email address for a user in an output-parameter named **EMAIL**. Let's further assume that I want to use that parameter as the **to** parameter in the Email Component. The XML fragment would look like:

```
<action-inputs>
    <to type="string" mapping="EMAIL"/>
                . . .
</action-inputs>
```

Constant Values as Inputs

If a component requires an input parameter and it is not found in the action-inputs, the component will automatically look for an XML node with the same name, in the component-definition section. If it finds one, it will use the value from there. The XML fragment would look like:

```
<action-inputs>
</action-inputs>

<component-definition>
    <to>joe.pentaho@pentaho.org</to>
</component-definition>
```

Replaceable Parameters

All components have the ability to use the built in template mechanism to perform text replacement using input parameters. This means that a constant value can be modified using the value of an action-input. Hmmm that wasn't very clear either. Let's try with an example:

```
<action-inputs>
    <EMAIL type="string"/>
        . . .
</action-inputs>

<component-definition>
    <to>{EMAIL}@pentaho.org</to>
</component-definition>
```

The text within the curly braces {} is replaced by an input parameter with the same name. In this example, if the value of the action-input EMAIL is **joe.pentaho** then the value of the **to** parameter passed to the email component would be **joe.pentaho@pentaho.org**.

TODO

Explain

Unknown macro: {PREPARE}
and the date parameter.

Prompting

Some components will generate a prompt for certain parameters that they require. The prompt will be generated by the runtime if the calling mechanism permits it. For example, the above mentioned Email Component requires a **to** parameter in order to know where to send the email. If the **to** parameter is specified as an action-input but it has a null value, the Email Component will ask the runtime if prompting is allowed. If the Action Sequence being executed has been

launched from a web browser, the runtime will respond with **true** and the Email Component will ask the runtime to prompt the user for the **to** email address. If the Action Sequence was running via a web service call, the runtime would reply with **false** and the Email Component would generate a parameter not found error and return.

Action-outputs

The **action-outputs** define what parameters will be saved in the Runtime Context and be made available to other Components when that component finishes executing. Like an **action-input**, an **action-output** can be mapped to another variable name using the mapping attribute.

Component-definition

The **component-definition** node provides a place for any Component specific options or parameters.

 **TODO**

Add more about component-definition and action-outputs

4 - Resource Types | 04 - Action Sequence XML | 6 - XML Schema

6 - XML Schema

This page last changed on Dec 13, 2006 by dmoran.

5 - Actions 04 - Action Sequence XML

This is not actually a schema, but it does explain the Action Sequence XML and node requirements.

XML Nodes marked as REQUIRED are only required if their parent node is being used. Attributes shown in square brackets [ ] are optional.

- <**action-sequence**> REQUIRED - Top level node for the Action Sequence Document
  - <**name**> NOT REQUIRED - The name of the Action Sequence, it must match the file name of the document.
  - <**version**> NOT USED - The version of this document
  - <**title**> NOT REQUIRED - Friendly name of the document. Used for display only
  - <**logging-level**> NOT REQUIRED - Sets the logging level for the entire Action Sequence. Valid values are: TRACE, DEBUG, INFO, WARN, ERROR and FATAL. If no logging level is set, ERROR will be used.
  - <**documentation**> NOT REQUIRED - Contains descriptive nodes used for generating documentation.
    - <**author**> - NOT REQUIRED - The author of this Action Sequence
    - <**description**> - NOT REQUIRED - Short (1-3 lines) description of the Action Sequence. This description is used by the solution navigation component to generate its display.
    - <**help**> - NOT REQUIRED - Long Description of the Action Sequence including instructions for it's use by an end user.
    - <**result-type**> - NOT REQUIRED - Type of output this Action Sequence will generate. It is used by the solution navigation component to generate its display. Action Sequences without a result-type will not be displayed by the navigation component. Valid values are: Report, Process, Rule, View and None.
    - <**icon**> - NOT REQUIRED - Thumbnail image that the navigation component will use for generating its display. The path to the image is relative to the directory that the ActionSequence document is in. For example: Example1_image.png
  - <**inputs**> - NOT REQUIRED - Collection of input parameters.
    - <*param-name* **type**="*data-type*" > - NOT REQUIRED - *param-name*is the name of a parameter that the Action Sequence is expecting to be available at run time. The type attribute specifies the data type of this parameter. See below for valid data types.
    - <**default-value**> - NOT REQUIRED - Allows the input parameter to specify a default value if a value has not been supplied. If the default-value node is present but has no value specified, the user will be prompted for the value if possible.
    - <**sources**> - NOT REQUIRED - list of parameter providers in the order they should be queried to obtain a parameter. Valid values are request, session and runtime. Note: if a param-name is set but default-value and sources are both not specified, a validation error will occur.
  - <**outputs**> - NOT REQUIRED - Collection of output parameters.
    - <*param-name* **type**="*data-type*" > - NOT REQUIRED - *param-name*is the name of a parameter that the Action Sequence is expecting will be set by the time all action-definitions have executed. The type attribute specifies the data type of this parameter. See below for valid data types.
  - <**logging-level**> NOT REQUIRED - Sets the logging level during this execution of the action-definition. Valid values are: TRACE, DEBUG, INFO, WARN, ERROR and FATAL. If no logging level is set, ERROR will be used.

- <**resources**> - NOT REQUIRED - Collection of resource parameters.
  - <*resource-name* > - NOT REQUIRED - *resource-name*is the name of a resource that the Action Sequence is expecting to use. The type attribute specifies the data type of this parameter. See below for valid data types.
  - <*resource-type*> - REQUIRED - The name of the type of resource required. Valid values are: solution-file, file and url.
  - <**location**> - REQUIRED - The path to the resource. For a resource-type of "solution-file", the location is a pathname relative to the top level of the current solution. If the resource-type is "file" then the location is assumed to be the a fully qualified path. For resource-type of "url" the location is assumed to be a fully qualified URL.
  - <**mime-type**> - NOT REQUIRED - Gives a hint about the mime type of the resource. **<*actions [loop-on**="*parameter-name*"**]** > - REQUIRED - The actions node contains "action-definition" nodes and optionally more "actions" nodes. The **loop-on** attribute is optional. When it is used, the nodes within "actions" will be executed multiple times. It is necessary to specify a parameter that is of type list (string-list or property-map-list) and the group of nodes that will be executed once for each element in the list. An input parameter will be generated with the same name as the loop-on attribute but it will have the value of one element in the list. For example: if a loop-on attribute named "department" is a string-list with department names, then a parameter named department will be available and be set to a different department name for each iteration.
- <**actions [loop-on**="*parameter-name*"**]** > - NOT REQUIRED - Since a single level of looping is not very fun, actions nodes can be nested within actions nodes to any level desired - no matter how silly it may be to do so.
- <**action-definition**> - REQUIRED (At least 1) - It defines one complete call to a component for execution of a task.
- <**action-inputs**> - NOT REQUIRED - Collection of action-input parameters.
  - <*input-name* **type**="*data-type*" **mapping**=*"param"*> - NOT REQUIRED - *input-name* is the name of a parameter that the Action Definition is expecting to be available at run time. The **type** attribute specifies the data type of this parameter. See 3 - Data Types for valid data types. The **mapping** attribute allows this input to be mapped to an Action Sequence input or a previous action-definition output with a different name.
- <**action-outputs**> - NOT REQUIRED - Collection of action-output parameters.
  - <*output-name* **type**="*data-type*" > - NOT REQUIRED - *output-name*is the name of a parameter that the Component will have set by the time it finishes executing. The type attribute specifies the data type of this parameter. See below for valid data types.
  - <**component-name**> - REQUIRED - The name of the java class that executes the action definition.
  - <**component-definition**> - REQUIRED - The component specific XML definition. See the documentation for the specific component for more information. This node may be empty but it must exist or a validation error will occur.

TODO

- Verify this is up to date
- Add output destinations
- Add resource types xml and string
- come up with a better way to display this - table maybe?

---

5 - Actions 04 - Action Sequence XML

file

An absolute path on the file system.

```
<file>
    <location>D:\samples\reporting\MyReport.rptdesign</location>
    <mime-type>text/xml</mime-type>
</file>
```

solution-file

This page last changed on Dec 13, 2006 by dmoran.

A file on the file system relative to the location of the current Action Sequence document.

```
<solution-file>
    <location>MyReport.rptdesign</location>
    <mime-type>text/xml</mime-type>
</solution-file>
```

url

This page last changed on Dec 13, 2006 by dmoran.

A URL.

```
<file>
    <location>http://www.myserver.com/logo.png</location>
    <mime-type>image/png</mime-type>
</file>
```

05. Integrating Pentaho Reports (JFreeReports)

This page last changed on Dec 15, 2006 by dmoran.

04. Action Sequences II - Building Solutions

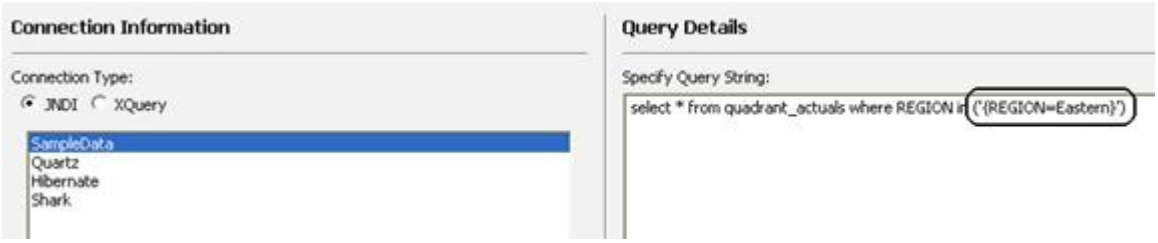
For a more documentation on using JFreeReport with the Pentaho Platform please see Reporting:Report Design Wizard

JFreeReport Report Definitions

JFreeReport report definitions are XML documents typically with a .xml extension, although this is not necessary. The Pentaho JFreeReportComponent (org.pentaho.plugin.jfree.JFReeReportComponent) uses the report definitions along with your data to produce a comprehensive set of output formats. The JFreeReport output formats worth noting are HTML, PDF, CSV, XLS and RTF.

JFreeReport gets its data from a Java TableModel. There is a custom TableModel implementation in the Pentaho Platform which provides for a memory friendly and performance efficient operation. Since JFreeReport operates against a definition and a TableModel there is no way to parameterize the JFreeReport itself. This is accomplished in the action-sequence and in the query itself.

Using the Pentaho Report Design Wizard you can create a JFreeReport definition and a basic action sequence. In the most simple case create a report against a <Blank> template and enter a SQL query against your database.



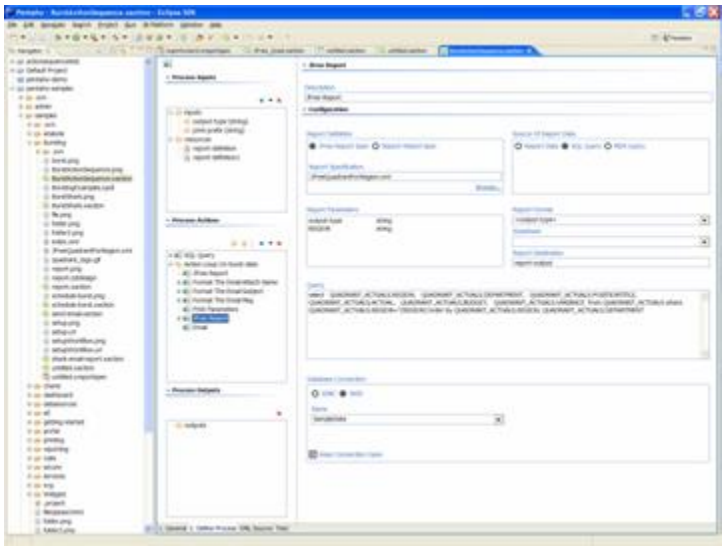
The screenshot above shows a parameterized SQL query for use with the Report Design Wizard. The column 'REGION' is what we are using for our parameter. The name of the parameter is also 'REGION' but this is not necessary. The wizard does not currently have the ability to prompt the user for parameter values, so a default value of 'Eastern' has been supplied. The full parameter is specified as {REGION=Eastern} in the query.

When the report is previewed in the Report Design Wizard the REGION parameter value is replaced with the specified default value (Eastern). When the report is published for use in the Pentaho platform the REGION parameter is setup in the action-sequence and the user will be prompted to enter a value. The action-sequence generated by the Report Design Wizard can be further customized by the Pentaho Design Studio.

Creating the Action Sequence

Once you have verified that the report works in the Report Design Wizard you can rest assured that it will

work in the Pentaho Platform. The reason for this is that the Report Design Wizard runs the report in an embedded standalone version of the Pentaho Platform. There is no need to create an action-sequence since once is generated automatically by the wizard. However, should you have an existing JFreeReport definition that was created by hand or from another tool. To see an example of an action sequence that generates a JFree report using report parameters refer to pentaho_demo/pentaho_solutions/samples/bursting/BurstActionSequence.xaction. Select the Define Process tab and select the JFree Report action in the Process actions tree.



JDBC Driver Setup

If you are using a database driver other than Hypersonic you'll need to configure the JBoss PCI to use that driver.

TODO

add the JDBC setup. We should have an generic JDBC setup doc that the components all like to

Verifying JFreeReport Integration into Pentaho Platform

At this point the report should be plugged in and ready for use. Point your web browser to your PCI (typically http://localhost:8080). Navigate to the report that you created under the Reporting Examples group.

When you hit the URL for the report a default parameter page is generated prompting you for an input to the parameter for REGION.

REGION	DEPARTMENT	POSITIONTITLE
Central	Sales	District Manager
Central	Sales	Senior Sales Rep
Central	Sales	Sales Rep
Central	Sales	Account Executive
Central	Sales	Pre-Sales
Central	Executive Management	CEO
Central	Executive Management	SVP WW Operations
Central	Executive Management	SVP Strategic Development
Central	Executive Management	SVP Partnerships

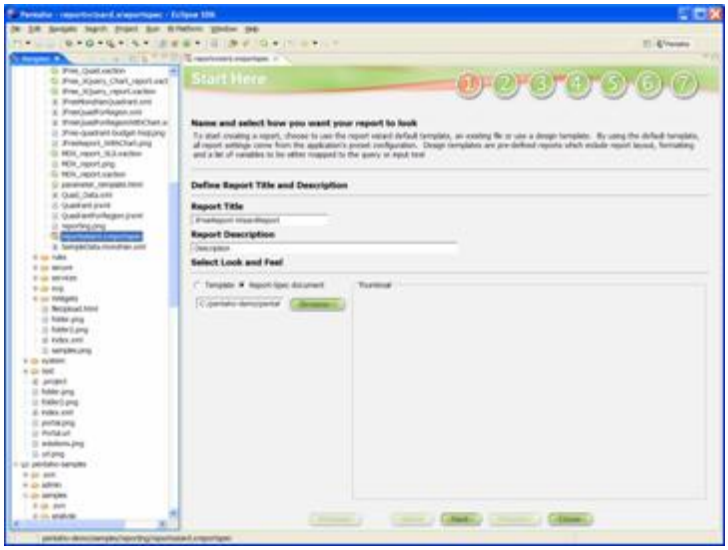
Once you submit the parameter the report will be generated and depending on the output-type in the action-sequence you'll see your report in the format desired. The example above shows an HTML screenshot.

The Report Design Wizard

The Pentaho Report Design Wizard provides for quick and easy creation of JFree reports using a simple step-by-step process, that allows for editing of the most commonly used report formatting features. The design wizard is the easiest way to quickly develop customized JFree reports. The wizard is available both as a stand alone application as well as an editor built into the Pentaho Design Studio. For a detailed description of how to use the wizard refer to the *Report Design Wizard User's Guide.*

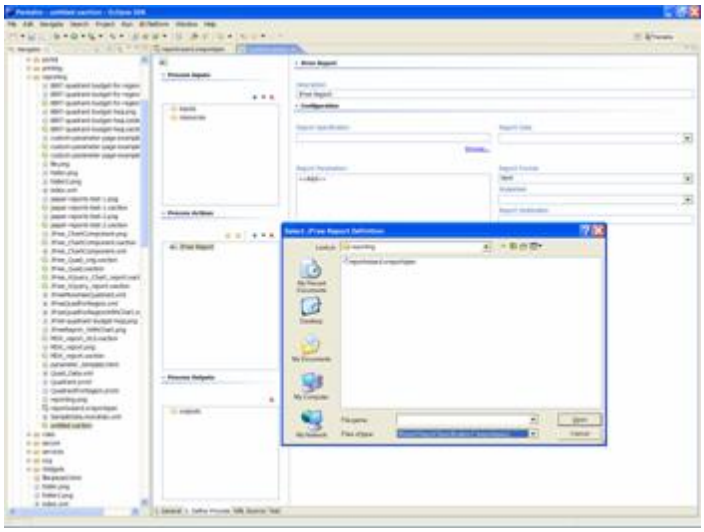
Creating a Design Wizard Specification

Select the Pentaho Design Studio icon and choose "New JFree Report Wizard File". Then select the Container Browse button, and choose the "reporting" folder in the PCI then select OK. Now give your new action sequence a name. You're now ready to design your report. If you're new to the Report Design Wizard, you can work through the example in the *Report Design Wizard User's Guide*. You'll want to skip the step titled "Deploying Reports into Pentaho BI Platform". When your design is complete make sure to save your new report specification.



Using a Design Wizard Specification

Once you've designed your report wizard specification it's time to integrate it into an action sequence that will allow it to run as part of the installed solution. Using the same process we described earlier, create a new blank action sequence in the "reporting" directory of your solution. Assign title, icon, and brief description to your new action sequence to assist you in finding it when we test our action sequence. Select the define process tab then right-click in the Process Actions tree area and select Add->Report->JFree Report. Now select the Browse link on the right side of the page. In the file chooser dialog select the Report Wizard Spec (*.xreportspec) file type then select the wizard specification you created above and select Open. Now save your new action sequence and your ready to go. It's that simple. Use your favorite browser and go to the reporting examples within the PCI. You should see a new entry for the new action sequence you've created. Go ahead and select it and your report should display.



04. Action Sequences II - Building Solutions

III - Actions and Component Reference

This page last changed on Dec 12, 2006 by dmoran.

II - Building Solutions Creating Pentaho Solutions

Actions are the workhorse of any action sequence. Each action in an action sequence is responsible for describing a particular type of task to be performed by the Solution Engine. For example a SQL Query action will describe a SQL query to be performed, the JNDI or JDBC connection to use, and the name of the action output where the query results are to be stored. Behind each action is a component that performs the action. Components are server side Java classes. As the Solution Engine processes each action in an action sequence it executes the component that performs that type of action. In many cases there is a one-to-one correspondence between an action type and the component that performs the action. For example the SQLLookupRule is a component that only processes SLQ query actions. However, that is not always the case. For example the UtilityComponent can perform multiple types of actions.

Like the action sequence itself, each action within the action sequence has a list of inputs and outputs. The action input parameters describe to the component how to perform the action. Each action in the Pentaho BI Platform has unique and specific inputs must be correctly specified. They also have optional inputs and definitions.

The outputs define what parameters will be available in the runtime context when the component has finished executing. Other actions/components that execute later on can use these outputs as inputs. This section describes many of the available actions along with their inputs and outputs.

II - Building Solutions Creating Pentaho Solutions

BIRT Reports

This page last changed on Dec 12, 2006 by dmoran.

Please see Reporting:BIRT

TODO

Should the components settings go here or be imported into this page?

Call External Action Sequence

This page last changed on Dec 12, 2006 by dmoran.

This action is used within an action sequence to allow calling of other action sequences in much the same way as a programming "GoSub". SubActions are executed synchronously.

Component Name: SubActionComponent

Inputs:

REQUIRED
solution -- The solution containing the action sequence being called.
path -- The path to the action sequence within the referenced solution.
action -- The name of the .xaction to be called (must include the .xaction extension).

OPTIONAL
Any input names from the calling .xaction that are desired to be passed to the called .xaction. The called action sequence can get access to the inputs as request inputs.

Outputs:

Any outputs from the called .xaction that need to be consumed by following action definitions or need to be passed out of the action sequence.

Charting

This page last changed on Jan 10, 2007 by bseyler.

The Pentaho BI platform currently employs JFreeChart as its charting engine. Each chart component can create at least 1 type of chart. Several of the chart components can create many chart types. Certain charts (where it makes sense) can be rendered as stacked and three dimensional. The platform charts render themselves as XML and then the XML is transformed to HTML via the use of an .xslt transformation.
Supported chart types are, dial, pie, grid, bar, line, and area charts.

Component Name: ChartComponent

Inputs:

REQUIRED
Chart Data -- The report data. Often this is the output of a SQL Query action.

OPTIONAL
Chart Row Dimension -- Indicates the chart data is to be aggregated along the row dimensions.
Title -- The chart title.
Subtitle -- The chart subtitle.
Title Font Style -- The font style to be used with the title and subtitle.
Title Font Size -- The font point size to be used with the title and subtitle.
Border Color -- The color of the chart border.
Chart Width -- The chart width.
Chart Height -- The chart height.
Title Position - Describes the position of the chart title. Valid positions are TOP, BOTTOM, LEFT, and RIGHT.
Range Title - Optional node that describes the chart range (usually the y-axis).
Chart Background - The color or image to be used as the chart background. This replaces the background of the chart itself and NOT the plot area. So if you set the image here you will probably see your image under the axis labels and scales and not in the plot area.
Plot Background - The color or image to be used as the plot background. This replaces the background of the plot area only.
Orientation -- The chart orientation. This can be either "Horizontal", "Vertical". Defaults to "Vertical".
Is 3D - If true the charting engine does it best to render a 3-D view of the chart.
Is Stacked - If true the charting engine will create a stacked version of this chart type (if possible).
Color Pallette - Singleton that contains a list of colors that make up the series palette.

Outputs:

None

Note:

Title Position, Range Title, Chart Background, Plot Background, Orientation, Is 3D, Is Stacked, URL Template, Prameter Name, and Color Pallette are not currently supported by the design studio.

- Charting XAction Reference

Charting XAction Reference

This page last changed on Jan 10, 2007 by bseyler.

The Pentaho BI platform currently employs JFreeChart as its charting engine. The implementation of the engine currently includes UI components for the following charts: Dial, Pie, Pie Grid, Bar Chart, Line, and Area. Each chart component can create at least 1 type of chart. Several of the chart components can create many chart types. Certain charts (where it makes sense) can be rendered as stacked and three dimensional. The platform charts render themselves as XML and then the XML is transformed to HTML via the use of an .xslt transformation. Currently not all tags are implemented in the design studio so some of these may have to be added by direct editing of the xaction.

- CategoryDatasetComponent
- Chart Component
- Chart Tag Reference
- TimeSeriesCollectionComponent
- XYSeriesCollectionComponent

CategoryDatasetComponent

This page last changed on Jan 10, 2007 by bseyler.

Charting XAction Reference Chart Component

The CategoryDatasetComponent is a UI component that can create a variety of charts including Bar, Line, Pie, Pie Grid, and Area. Where applicable there are several options that can be applied. The creation of the chart is commonly performed by a JSP or indirectly by creating the appropriate portlet object, via the CategoryDatasetChartPortlet. See the Chart Tag Reference for explanations of each node.

```
<chart>
  <chart-type>BarChart</chart-type>
  <title>Sample Chart</title>
  <subtitle>a simple sample</subtitle>
  <chart-background type="color">#FFFFFF</chart-background>
  <chart-background-image>test\charts\ChartBackground.jpg</chart-background-image>
  <plot-background-color>#FFFFFF</plot-background-color>
  <plot-background-image>test\charts\ChartBackground.jpg</plot-background-image>
  <orientation>Horizontal</orientation>
  <height>550</height>
  <width>650</width>
  <is-3D>true</is-3D>
  <is-stacked>true</is-stacked>
  <urlTemplate>
    <![CDATA\[/pentaho/Pivot?solution=samples&path=analysis&action=query1.xaction&department=.[{DEPARTMENT}]
  </urlTemplate>
  <paramName>MEASURES</paramName>
  <paramName2>DEPARTMENT</paramName2>
  <color-palette>
    <color>#336699</color>
    <color>#99CCFF</color>
    <color>#999933</color>
    <color>#666699</color>
    <color>#CC9933</color>
    <color>#006666</color>
    <color>#3399FF</color>
    <color>#993300</color>
    <color>#CCCC99</color>
    <color>#666666</color>
    <color>#FFCC66</color>
    <color>#6699CC</color>
    <color>#663366</color>
    <color>#9999CC</color>
    <color>#CCCCCC</color>
    <color>#669999</color>
    <color>#CCCC66</color>
    <color>#CC6600</color>
    <color>#9999FF</color>
    <color>#0066CC</color>
    <color>#99CCCC</color>
    <color>#999999</color>
    <color>#FFCC00</color>
    <color>#009999</color>
    <color>#99CC33</color>
    <color>#FF9900</color>
    <color>#999966</color>
    <color>#66CCCC</color>
    <color>#339966</color>
    <color>#CCCC33</color>
  </color-palette>
</chart>
```

Chart Component

This page last changed on Jan 10, 2007 by bseyler.

CategoryDatasetComponent Charting XAction Reference Chart Tag Reference

The ChartComponent is a UI component that can create a variety of charts including Bar, Line, Pie, Pie Grid, and Area. Where applicable there are several options that can be applied. The following action sequence uses the output from an as the input for a ChartComponent. See the Chart Tag Reference for a n explanation of the tags

```
<action-sequence>
  <name>Chart.xaction</name>
  <title>Default Title</title>
  <version>1</version>
  <logging-level>DEBUG</logging-level>
  <documentation>
    <author>William E. Seyler</author>
    <description>Default Description</description>
    <icon>JFree-quadrant-budget-hsql.png</icon>
    <help>Help</help>
    <result-type>rule</result-type>
  </documentation>
  <inputs>
    <chart-type type="string">
      <default-value>.png</default-value>
      <sources>
        <request>type</request>
      </sources>
    </chart-type>
  </inputs>
  <actions>
    <action-definition>
      <action-outputs>
        <result-set type="list" />
      </action-outputs>
      <component-name>SQLLookupRule</component-name>
      <action-type>rule</action-type>
      <component-definition>
        <source>sql</source>
        <live>true</live>
        <jndi>SampleData</jndi>
        <query>
          <![CDATA[select QUADRANT_ACTUALS.REGION, QUADRANT_ACTUALS.DEPARTMENT,
            QUADRANT_ACTUALS.POSITIONTITLE, QUADRANT_ACTUALS.ACTUAL, QUADRANT_ACTUALS.BUDGET
            from QUADRANT_ACTUALS order by
QUADRANT_ACTUALS.REGION,QUADRANT_ACTUALS.DEPARTMENT]]>
        </query>
      </component-definition>
      </action-definition>
      <action-definition>
        <action-inputs>
          <output-type type="string" mapping="chart-type" />
          <result-output type="file" />
          <chart-data type="list" mapping="result-set" />
        </action-inputs>
      <action-outputs>
        <chart-output type="string" />
        <base-url type="string" />
      </action-outputs>
      <component-name>ChartComponent</component-name>
      <action-type>report</action-type>
      <component-definition>
        <by-row>false</by-row>
        <chart-attributes>
          <chart-type>PieChart</chart-type>
          <title>Actual vs Budget by Region</title>
          <title-position>TOP</title-position>
          <title-font>
            <font-family>Serif</font-family>
```

```
              <size>36</size>
              <is-bold>false</is-bold>
              <is-italic>false</is-italic>
            </title-font>
            <range-title>US Dollars</range-title>
            <chart-background type="color">#FFFFFF</chart-background>
            <plot-background type="color">#FF0000</plot-background>
            <orientation>Horizontal</orientation>
            <height>400</height>
            <width>800</width>
            <is-3D>true</is-3D>
            <is-stacked>false</is-stacked>
            <category-label-rotation>90</category-label-rotation>
            <border-visible>false</border-visible>
            <border-paint>#3399FF</border-paint>
            <include-legend>true</include-legend>
          </chart-attributes>
        </component-definition>
      </action-definition>
    </actions>
  </action-sequence>
```

Chart Tag Reference

This page last changed on Jan 10, 2007 by bseyler.

Chart Component | Charting XAction Reference | TimeSeriesCollectionComponent

The following is a comprehensive list of tags available for the ChartComponent.

- by-row
- chart
- chart-attributes
- chart-background
- chart-type
- color
- color-palette
- dataset-type
- domain-includes-zero
- domain-label-rotation
- domain-label-rotation-dir
- domain-period-type
- domain-sticky-zero
- domain-title
- domain-title-font
- domain-vertical-tick-labels
- dot-height
- dot-width
- height
- is-3D
- is-stacked
- line-style
- line-width
- markers-visible
- orientation
- paramName
- paramName2
- plot-background
- range-title
- range-title-font
- subtitle
- title
- title-font
- title-position
- url-template
- width

by-row

This page last changed on Jan 10, 2007 by bseyler.

<by-row/>

Required node that describes how the chart data is aggregated. Values are TRUE or FALSE. This tag specifies if the supplied data should be processed by row or (default) by column. If your chart series are on the wrong axis... then add this tag as true.

chart

This page last changed on Jan 10, 2007 by bseyler.

by-row Chart Tag Reference chart-attributes

<chart/>

The chart root node. This item is a mandatory singleton node.

chart-attributes

This page last changed on Jan 10, 2007 by bseyler.

chart Chart Tag Reference chart-background

<chart-attributes/>

Required by the ChartComponent, this node contains all chart attributes nodes.

chart-background

This page last changed on Jan 10, 2007 by bseyler.

chart-attributes Chart Tag Reference chart-type

<chart-background/>

Optional singleton node that describes the background type. Valid types include color and image. If type="color", designate the color with 6 byte hexadecimal notation. When type="image", the value is a filepath (relative to the solution directory) of an image file to use as the chart background image.

Note: This image replaces the background of the chart itself and NOT the plot area. So if you set the image here you will probably see your image under the axis labels and scales and not in the plot area.

chart-type

<chart-type/>

Optional singleton node that contains the chart type in the node text. When used with the CategoryDatasetComponent, if the chart-type is not set in the xml definition then it must be set on the CategoryDataset Component directly in code (servlet or JSP). Valid chart type strings for CategoryDataset are "PieChart", "PieGrid", "BarChart", "LineChart", and "AreaChart". Valid values for a TimeSeriesCollection or XYSeriesCollection are "BarChart", LineChart", AreaChart", "StepChart", "StepAreaChart", "DifferenceChart", and "DotChart".

color

<color/>

Optional multiple nodes that contain a 6 byte hexadecimal notation to be used as an entry in the color series palette.

color-palette

This page last changed on Jan 10, 2007 by bseyler.

color Chart Tag Reference dataset-type

<color-palette/>

Singleton that contains a list of <color/> nodes that make up the series palette.

dataset-type

<dataset-type/>

Optional singleton node that used in the ChartComponent to determine which type of dataset component to use. Valid dataset types are CategoryDataset, XYSeriesCollection, TimeSeriesCollection. The default value is CategoryDataset.

domain-includes-zero

This page last changed on Jan 10, 2007 by bseyler.

dataset-type Chart Tag Reference domain-label-rotation

<domain-includes-zero/>

Optional singleton node indicating that a tick mark for zero should be included on the domain, if zero is in the range of values. This is only used by the XYSeriesCollection. Default = true

domain-label-rotation

<domain-label-rotation/>

Accepts the degree of rotation as radiant (45°= 0.25 radiant) in string form. For a degree/radiant conversion check out: http://www.unitconversion.org/unit_converter/angle.html

domain-label-rotation-dir

This page last changed on Jun 05, 2007 by bseyler.

domain-label-rotation Chart Tag Reference domain-period-type

<domain-label-rotation-dir/>

Optional value that controls the direction of rotation of the domain labels. Possible values are **up** (clockwise) and **down** (counterclockwise). Needs to be used with <domain-label-rotation> tag to have any effect.

domain-period-type

<domain-period-type/>

Optional singleton node that specifies what time periods should be used on the domain of a TimeSeriesCollection. This is valid only for TimeSeriesCollection. Valid Values are "Day", "Hour", "Millisecond", "Minute", "Month", "Quarter", "Second", "Week", and "Year". Default = Millisecond.

domain-sticky-zero

This page last changed on Jan 10, 2007 by bseyler.

domain-period-type Chart Tag Reference domain-title

<domain-sticky-zero/>

Optional singleton node indicating that zero must be included in the domain, event if it is not in the range of values. This is only used by the XYSeriesCollection. Default = true

domain-title

<domain-title/>

Optional singleton node that describes the chart domain (usually the x-axis)

domain-title-font

This page last changed on Jan 10, 2007 by bseyler.

domain-title Chart Tag Reference domain-vertical-tick-labels

<domain-title-font/>

Optional singleton node that describes the font used for the domain title. See title-font for information on describing fonts.

domain-vertical-tick-labels

This page last changed on Jan 10, 2007 by bseyler.

domain-title-font Chart Tag Reference dot-height

<domain-vertical-tick-labels/>

Optional singleton node that indicates that domain tick labels should be printed vertically. This is valid for TimeSeriesCollection and XYSeriesCollection. Default = false

dot-height

<dot-height/>

> **As of Version 1.2.1**
>
> Available as a chart definition XML attribute as of Pentaho platform version 1.2.1.

This tag is specifically for the Dot chart. Sets the height of the dot (actually a square) in pixels. Default dot height is 5 pixels.

dot-width

This page last changed on Mar 09, 2007 by gmoran.

dot-height Chart Tag Reference height

<dot-width/>

As of Version 1.2.1

Available as a chart definition XML attribute as of Pentaho platform version 1.2.1.

This tag is specifically for the Dot chart. Sets the width of the dot (actually a square) in pixels. Default dot width is 5 pixels.

height

This page last changed on Jan 10, 2007 by bseyler.

dot-width Chart Tag Reference is-3D

<height/>

Optional singleton whose text value is an integer that represents the height of the chart.

is-3D

This page last changed on Jan 10, 2007 by bseyler.

height Chart Tag Reference is-stacked

<is-3D/>

Optional singleton whose text value can be either "true" or "false". Defaults to "false". If true the charting engine does it best to render a 3-D view of the chart.

is-stacked

<is-stacked/>

Optional singleton whose text value can be either "true" or "false". Defaults to "false". If true the charting engine will create a stacked version of this chart type (if possible).

line-style

<line-style/>

> **As of Version 1.2.1**
>
> Available as a chart definition XML attribute as of Pentaho platform version 1.2.1.

This attribute will set the style of all line series in supported charts. Valid values for this attribute are:

- solid (default)
- dash
- dot
- dashdot
- dashdotdot

Implemented chart types that support this attribute are :

- LineChart (with time series, categorical or xy data)
- StepChart (with time series, categorical or xy data)

Setting the line style per series is not yet supported, but will be implemented in a future version.

line-width

This page last changed on Mar 09, 2007 by gmoran.

line-style Chart Tag Reference markers-visible

<line-width/>

As of Version 1.2.1

Available as a chart definition XML attribute as of Pentaho platform version 1.2.1.

This attribute will set the line width of all line series in supported charts. Valid values for this attribute are numerical float values greater than or equal to zero. Default line width is 1.0.

Implemented chart types that support this attribute are :

- LineChart (with time series, categorical or xy data)
- StepChart (with time series, categorical or xy data)

Setting the line width per series is not yet supported, but will be implemented in a future version.

markers-visible

<markers-visible/>

As of Version 1.2.1

Available as a chart definition XML attribute as of Pentaho platform version 1.2.1.

This attribute sets whether the markers (data points) are shown as shapes on the line series in supported charts. The default is that these markers are invisible.

Valid values for this attribute are true or false.

Implemented chart types that support this attribute are :

- LineChart (with time series, categorical or xy data)
- Markers do not display on 3D LineCharts.

Setting the markers visible or invisible per series is not yet supported, but will be implemented in a future version.

orientation

This page last changed on Jan 10, 2007 by bseyler.

markers-visible Chart Tag Reference paramName

<orientation/>

Optional singleton whose text value can be either "Horizontal", "Vertical". Defaults to "Vertical".

paramName

This page last changed on May 07, 2007 by jcornelius.

<paramName/>

Optional singleton whose test values is the parameter name of the innermost query variable. If this name occurs in the url-template, it will be replaced with the correct item.

paramName2

This page last changed on May 07, 2007 by jcornelius.

paramName Chart Tag Reference plot-background

<paramName2/>

Optional singleton whose test values is the parameter name of the outermost query variable. If this name occurs in the urlTemplate, it will be replaced with the correct item.

plot-background

<plot-background/>

Optional singleton node that describes the plot type. Valid types include color and image. If type="color", designate the color with 6 byte hexadecimal notation. When type="image", the value is a filepath (relative to the solution directory) of an image file to use as the plot background image.

Note: This image replaces the background of the plot area only.

range-title

<range-title/>

Optional node that describes the chart range (usually the y-axis).

range-title-font

This page last changed on Jan 10, 2007 by bseyler.

<range-title-font/>

Optional singleton node that describes the font for the range-title. See title-font to see how a font is described.

subtitle

range-title-font Chart Tag Reference title

<subtitle/>

Optional singleton node that contains the requested subtitle in the node text.

title

<title/>

Optional singleton node that contains the requested title in the node text.

title-font

<title-font/>

Optional singleton node that describes the chart font. Font attributes are included as child nodes. Child nodes include, <font-family>, <size>, <is-bold>, and <is-italic>.

title-position

<title-position/>

Optional node that describes the position of the chart title. Valid positions are TOP, BOTTOM, LEFT, and RIGHT.

url-template

<url-template/>

Optional singleton whose text value is used as a template to create a drill link map for the image. In order for the image map to be created, the output must be <image-tag type="string"/>.

width

This page last changed on Jan 10, 2007 by bseyler.

url-template Chart Tag Reference

<width/>

Optional singleton whose text value is an integer that represents the width of the chart.

TimeSeriesCollectionComponent

This page last changed on Jan 10, 2007 by bseyler.

Chart Tag Reference Charting XAction Reference XYSeriesCollectionComponent

The TimeSeriesCollectionComponent is a UI component that can create a variety of charts, using a Time Series as the domain axis, including Bar, Line, Step, StepArea, Area, Difference, and Dot. Where applicable there are several options that can be applied. The creation of the chart is commonly performed by a JSP or indirectly by creating the appropriate portlet object via the TimeSeriesCollectionChartPortlet. See the Chart Tag Reference for explanations of each node.

If the data for this component is arranged by column, each record is expected to have three fields. Each record is assumed to represent one data point on the chart, ordered by the series name and domain value. The first field is a String containing the name of the series. The second field is a Date/Timestamp containing the domain value of the data point. The third field is a Number containing the range value of the data point.

If the data for this component is arranged by row, each record is expected to contain all the data points for the series. The first field will contain a String representing the name of the series. It is assumed the other fields in the record consist of a collection of date/number data points (i.e. fields 2, 4, 6 etc. will be dates, and fields 3, 5, 7 etc will be numbers)

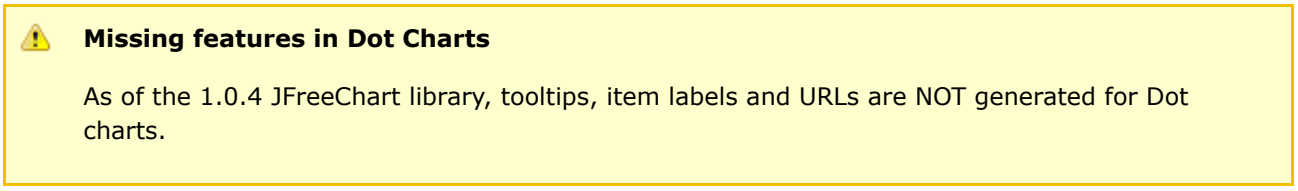
XYSeriesCollectionComponent

TimeSeriesCollectionComponent Charting XAction Reference

The XYSeriesCollectionComponent is a UI component that can create a variety of charts, using a numeric value range as the domain axis, including Line, Step, StepArea, Area, Difference, and Dot. Where applicable there are several options that can be applied. The creation of the chart is commonly performed by a JSP or indirectly by creating the appropriate portlet object via the XYSeriesCollectionChartPortlet. See the Chart Tag Reference for explanations of each node.

If the data for this component is arranged by column, each record is expected to have three fields. Each record is assumed to represent one data point on the chart, ordered by the series name and domain value. The first field is a String containing the name of the series. The second field is a Number containing the domain value of the data point. The third field is a Number containing the range value of the data point.

If the data for this component is arranged by row, each record is expected to contain all the data points for the series. The first field will contain a String representing the name of the series. It is assumed the other fields in the record consist of a collection of x/y data points (i.e. fields 2, 4, 6 etc. will be domain values, and fields 3, 5, 7 etc will be range values)



Missing features in Dot Charts

As of the 1.0.4 JFreeChart library, tooltips, item labels and URLs are NOT generated for Dot charts.

Content Repository Cleaner

This page last changed on Dec 12, 2006 by dmoran.

This action remove stale items from the content repository. It takes only one input, a number that represents how long an item is allowed to reside in the content repository. ie. If the value 90 is used then items older than 90 days will be removed from the content repository.

Component Name: ContentRepositoryCleaner

Inputs:

REQUIRED
Expiration Days -- The number of days an item should be retained in the Content repository.

Outputs:

Deleted Items -- The number of items deleted from the content repository.

TODO

This action is not currently supported by the Design Studio.

Email

This page last changed on Dec 12, 2006 by dmoran.

This action sends text or HTML based emails that may contain attachments. If the data type of the **to** parameter is property-map, then the map is assumed to contain the name value pairs for **to**, **subject** and **from**. The parameter **attach** should contain the name of a parameter of type content that contains the attachment. **attach-name** is the file name that the attachment will appear to have when opened by the email client.

Component Name: EmailComponent

Inputs:

REQUIRED
To - The recipients email address.
Subject - The email subject line.
Text Message - Plain text message body.
HTML Message - HTML message body.

OPTIONAL
From - The email address of the sender. If not specified, the default from "email_config.xml" will be used.
CC - The email address of carbon copy recipient.
BCC - The email address of blind carbon copy recipient.
Attachment Name - The name of the attachment as displayed in the email.
Attachment - The content to be attached.

Outputs: None

Notes

The underlying action sequence XML for the email action supports providing the "To" address in the form of a property-map. In this case, the map is assumed to contain the name value pairs for to, **subject, and **from inputs. This functionality is not supported within the Design Studio.

TODO

Add more info about attachments

Hello World

This page last changed on Dec 12, 2006 by dmoran.

This action simply returns the specified text to the client browser.

Component Name: HelloWorldComponent

Inputs:
Message - The text to be returned.

Outputs: None

Jasper Reports

This page last changed on Dec 12, 2006 by dmoran.

Please see Reporting:Jasper

TODO
Should the components settings go here or be imported into this page?

JavaScript

This page last changed on Dec 12, 2006 by dmoran.

This action executes the specified Javascript. Parameters specified as inputs will be available to the script for use. The JavascriptRule can have one or more outputs.
The component can also define library elements in the component definition. Each specified library file must exist in the solution, and will be pre-pended to the script that's specified in the component definition. In this way, you can create a library of commonly used javascript code, and include it at runtime execution.

Component Name: JavascriptRule

Inputs:

REQUIRED
Javascript - The Javascript to be executed.
Imported Javascript - Any files in the solution containing Javascript functions referenced by this Javascript.

OPTIONAL
Any inputs to the action sequence and any outputs from preceding actions in the action sequence can be specified. Inputs will be available as a variable to the scripting engine.

Outputs:

Any variable in the Javascript may be specified as an output from the action. Any output that is defined in the outputs, and not actually assigned a value in the Javascript Rule will be created and assigned a null value when the rule finishes execution.

JFree Reports

This page last changed on Dec 12, 2006 by dmoran.

This action generates JFree reports using JFree report definitions or Report Wizard Specification.

Component Name: JFreeReportComponent

Inputs:

REQUIRED
Report Specification -- The JFree Report definition or Report Wizard Specification to use when generating the report.
Report Data -- The report data. Often this is the output of a SQL Query action. This is required only if the Report Specification is a Report Wizard Specification (*.xreportspec).

OPTIONAL
Report Parameters -- The format in which to generate the report (Ex. Html, xsl).
Report Format -- The stylesheet to use when displaying the report.
Report Destination -- The name of the output variable in which to save the report.

Outputs:

None

TODO

This needs way more information

Kettle

This page last changed on Dec 12, 2006 by dmoran.

The Kettle actions allow for the execution ETL operations within your action sequence using the Kettle open source ETL tool. Supported operations include the execution of Kettle ETL transformation and Kettle jobs. For further information on Kettle refer to http://kettle.pentaho.org.

Kettle Transformation

Executes a Kettle transformation and saves the resulting data into an output result set.

Component Name: KettleComponent

Inputs:

REQUIRED
Transformation Step - The step in the transformation from which to retrieve data.

OPTIONAL
Transformation Inputs __-- Allows for the inclusion of miscellaneous inputs to the Kettle transformation.

Outputs:

Output Name - The name of the result set containing the data from executing the Kettle transformation.

Resources:

REQUIRED
Transformation File -- The kettle transformation file that is be executed.

Kettle Job

Executes a Kettle Job.

Component Name: KettleComponent

Inputs:

OPTIONAL
Job Inputs -- Allows for the inclusion of miscellaneous inputs miscellaneous inputs to the Kettle job.

Resources:
REQUIRED
Job File - The Kettle job file to be executed.

TODO

Add parameters for RDBMS transform repository

MDX Query

This page last changed on Dec 12, 2006 by dmoran.

The MDX lookup provides a facility to query multidimensional datasources using the the MDX query structure.

Inputs:

REQUIRED
JDBC Connection - - This is a JDBC connect string for the desired datasource. If you need connectivity to a datasource other than hypersonic you must provide the JDBC drivers in a connection string see **MDX Connection** below for information on how to do this. Either this node or the **MDX Connection** must be supplied.

Or

MDX Connection - This can be used in lieu of all the above connection properties and in fact will override the above properties should any exist. This string is a semicolon separated named properties list that is parsed by mondrian to create the connection. Here is an example for a connection based on the mdx-connection-string: mondrian; Jdbc=jdbc:odbc:MondrianFoodMart; Catalog=/WEB-INF/FoodMart.xml. When using this format the user name and password could be passed in as part of the jdbc connection string. This is dependent on the driver being used. Driver names can also be passed in on this string. More information about this type of connection string consult the mondrian documentation. Either this or JDBC Connection must be specified.

JDCB User Name - The JDBC user ID. If using other than hypersonic as the datasource then this may not evaluate correctly and you may need to use the **MDX Connection** (see below) to set the user ID and password.
JDBC Password - The JDBC password If using other than hypersonic as the datasource then this may not evaluate correctly and you may need to use the **MDX Connection** (see below) to set the user ID and password.
Query - This is the MDX query string that defines the desired dataset. For more information about MDX query strings consult the mondrian documentation or visit:
http://www.informit.com/articles/article.asp?p=29418&seqNum=3&rl=1

Resources:

REQUIRED
Catalog -- The ROLAP catalog path. If the catalog starts with HTTP then this path is evaluated as an absolute URL and the system will attempt to load the catalog directly. If the catalog does not start with HTTP then the path is assumed to be from the solution root and a URL is constructed accordingly and passed to the datasource engine (mondrian).Either this or MDX Connection must be specified.

TODO

Document output

Prepared Components - Enabling Subreporting and Connection Sharing

This page last changed on Apr 09, 2007 by wgorman.

Most of the data source components now support a new feature called "prepared_component". This functionality allows data source components to be executed by other components and also share their connections.

Data Source Components that implement the prepared component feature include:

- SQLLookupRule
- MDXLookupRule
- HQLLookupRule
- XQueryLookupRule

Also note that SQLExecute can use a prepared_component as an action input for sharing connections.

Enabling JFreeReport Subreporting

By defining a component output with the name "prepared_component", a datasource component goes into a prepared state vs. the standard execution state. The component initializes its connection and sets up its query, but waits for another component to execute the prepared statement. An example of this is a subreport in JFreeReport. JFreeReport will execute the prepared statement for each item in its primary result set. Note that currently only the JFreeReport Component uses the prepared_component for later execution. Here is an example action sequence that uses the prepared_component functionality in a subreport.

```
<action-definition>
      <component-name>SQLLookupRule</component-name>
      <action-type>SQL Query For Report Data</action-type>
      <action-inputs>
         <max_rows type="string"/>
      </action-inputs>
      <action-outputs>
        <prepared_component type="prepared-component" mapping="main_query"/>
      </action-outputs>
      <component-definition>
        <jndi>SampleData</jndi>
        <query><![CDATA[select DISTINCT POSITIONTITLE from QUADRANT_ACTUALS order by
POSITIONTITLE]]></query>
         <max_rows>3</max_rows>
      </component-definition>
    </action-definition>

    <action-definition>
      <component-name>SQLLookupRule</component-name>
      <action-type>SQL Query For Report Data</action-type>
      <action-inputs>
         <prepared_component type="prepared-component" mapping="main_query"/>
      </action-inputs>
      <action-outputs>
        <prepared_component type="prepared-component" mapping="subreport_query"/>
      </action-outputs>
      <component-definition>
        <jndi>SampleData</jndi>
        <query><![CDATA[select DISTINCT DEPARTMENT from QUADRANT_ACTUALS WHERE POSITIONTITLE =
{PREPARELATER:POSITIONTITLE} order by DEPARTMENT]]></query>
        <dept>Finance</dept>
      </component-definition>
```

```
      </action-definition>

      <action-definition>
        <component-name>JFreeReportComponent</component-name>
        <action-type>Create Report Using Query Results</action-type>
        <action-inputs>
          <default mapping="main_query"/>
          <subreport_query mapping="subreport_query"/>
          <output-type type="string"/>
        </action-inputs>
        <action-resources>
          <report-definition type="resource"/>
        </action-resources>
        <component-definition/>
      </action-definition>
```

See the pentaho-solutions/test/reporting/jfreereport-subreport-ipreparedcomponent-test.xaction for the complete example above.

New syntax introduced includes the "PREPARELATER" template item in the sub query above. The PREPARELATER field "POSITIONTITLE" is resolved when the query is executed within the JFreeReportComponent. In this example, The subquery is executed for each row in the main query. This relationship is defined within the JFreeReport xml file. Also note that the second SQLLookupRule takes in the first SQLLookupRule as an action input. This is another benefit of using Prepared Component, components can now share connections.

Within the JFreeReportComponent itself, the only changes that need to be made to support subreports are to include the additional lookup rules as action inputs as seen above. The action input name needs to match the subreport query name provided in the JFreeReport XML.

Connection Sharing

An additional benefit of the prepared_component feature is the ability to share connections across components. For instance, if two SQLLookupRules need to share a connection, they are able to do so using the prepared_component functionality. In this example, notice that the first SQLLookupRule makes a connection available by using the action output prepared_component, and then the two following SQL components use the prepared_component as input for sharing a connection:

```
  <!-- first create a connection -->

      <action-definition>
        <component-name>SQLLookupRule</component-name>
        <action-type>Get Connection</action-type>
        <action-inputs/>
        <action-outputs>
          <prepared_component mapping="connObj"/>
        </action-outputs>
        <component-definition>
          <jndi>SampleDataAdmin</jndi>
        </component-definition>
      </action-definition>

        <!-- second, create temporary table w/ connection and add data -->

        <action-definition>
          <action-inputs>
            <prepared_component mapping="connObj"/>
          </action-inputs>
          <action-outputs/>
          <component-name>SQLExecute</component-name>
          <action-type>Create a temp table</action-type>
```

```
          <component-definition>
            <continue_on_exception>true</continue_on_exception>
            <query><![CDATA[
              drop table tmptbl;
              create temp table tmptbl(val int) ON COMMIT PRESERVE ROWS;
              insert into tmptbl values (1)
            ]]></query>
          </component-definition>
        </action-definition>

        <!-- third, extract inserted data from temporary table -->

       <action-definition>
         <component-name>SQLLookupRule</component-name>
         <action-type>Check for Data</action-type>
         <action-inputs>
            <prepared_component mapping="connObj"/>
         </action-inputs>
         <action-outputs>
           <query-result mapping="a_result"/>
         </action-outputs>
         <component-definition>
           <query>SELECT * FROM tmptbl</query>
         </component-definition>
       </action-definition>
```

See the pentaho-solutions/test/ipreparedcomponents/ipreparedcomponent_sql_temptable.xaction for the complete example above

Printing

This page last changed on Dec 12, 2006 by dmoran.

Prints reports and content to a named printer accessible from the computer hosting the solution engine. The content to print can be specified in one of two ways.

1. Specify the file as a **printFile** resource or component setting.
2. Have a previous action in the sequence output content to the parameter **report-output**.

Currently, the org.pentaho.plugin.jfreereport.JFreeComponent, org.pentaho.plugin.eclipsebirt.BIRTComponent, and org.pentaho.plugin.jasperreports.JasperComponent have the ability to generate report content as **report-output**. If no content to print is specified, the action sequence will fail.

Component Name: PrintComponent

Inputs:

OPTIONAL
printFile - string *If not specified, the report-output parameter is used.*
printerName - string *If not specified, the default printer is used*
copies - number

Outputs:

last-printer-selected - string *If not specified in the action-outputs and action-inputs, value is not set.*

Scheduling

This page last changed on Dec 12, 2006 by dmoran.

The Pentaho BI platform currently employs Quartz as its scheduler. The implementation is a singleton that is JDBC persisted. It is fault tolerant and fault recoverable. Scheduled misfires are handled according to a set of predefined rules.

Access to the scheduler is through the org.pentaho.plugin.quartz.JobSchedulerComponent by implementing an Action Sequence. Samples can be found in test/scheduler/. There are currently four different actions available to the job scheduler. "startJob", "suspendJob", "resumeJob", and "deleteJob".

Start Job

Creates a job and a trigger and then registers the job and trigger for execution with the scheduler. In the case of a "Simple Trigger", actual job execution occurs when the trigger condition is met and occurs at the defined repeat interval until the number of defined repeat cycles has occurred. In the case of "Cron Trigger" the firing of the job occurs according to the rules set forth in the cron expression string (see below). The job itself is a solution document that is to be performed. This allows the scheduling of any other existing solution such as printing and email. In the event of a fault such as power failure, system crash, etc., after the scheduler restarts it will apply the misfire rules to any trigger that has misfired. The values of the "jobName", "triggerType" (and the trigger types associated inputs), "solution", "path", and "action" need to be defined in the solution document.

Component Name: JobSchedulerComponent

Inputs:

REQUIRED
Job Name -- Name of the job to be suspended.

Outputs:

None

See Triggers

See Misfires

Suspend Job

Pauses a specified running job. Once the job is paused the only way to start it again is with a resume job.

Component Name: JobSchedulerComponent

Inputs:

REQUIRED
Job Name -- Name of the job to be suspended.

Outputs:

None

Resume Job

Resumes a previously suspended job. Once the job is resumed it will apply the misfire rules if required.

Component Name: JobSchedulerComponent

Inputs:

REQUIRED
Job Name -- Name of the job to be suspended.

Outputs:

None

Delete Job

Deletes a specified job. The job is deleted immediately. However if a job is currently executing in a scheduler thread then it will continue to execute. No new instances of the job will be scheduled. The only input for this action is a "jobName".

Component Name: JobSchedulerComponent

Inputs:

REQUIRED
Job Name -- Name of the job to be deleted.

Outputs:

None

Misfires

This page last changed on Dec 12, 2006 by dmoran.

When a trigger misses its firing time due to reasons such as the Scheduler was paused or shut down, this is referred to as a *misfire*. Misfires are determined by the triggers misfire instruction. There are trigger misfires types available for all trigger types and those that are specific to "simple" and "cron" trigger types.

All Triggers - These misfire instructions are applicable to any trigger.

MISFIRE_INSTRUCTION_SMART_POLICY: This misfire instruction is the default for all triggers created. Essentially this instructs the trigger to use a default policy dependent on the type of trigger that is created. For a "simple" trigger the rule is as follows:

- If the Repeat Count is 0 then the instruction will be interpreted as MISFIRE_INSTRUCTION_FIRE_NOW.
- If the Repeat Count is REPEAT_INDEFINITELY, then the instruction will be interpreted as MISFIRE_INSTRUCTION_RESCHEDULE_NEXT_WITH_REMAINING_COUNT. **WARNING**: using MISFIRE_INSTRUCTION_RESCHEDULE_NEXT_WITH_REMAINING_COUNT with a trigger that has a non-null end-time may cause the trigger to never fire again if the end-time arrived during the misfire time span.
- If the Repeat Count is 0, then the instruction will be interpreted as MISFIRE_INSTRUCTION_RESCHEDULE_NOW_WITH_EXISTING_REPEAT_COUNT.

INSTRUCTION_RE_EXECUTE_JOB: Instructs the Scheduler that the Trigger wants the JobDetail to re-execute immediately.

INSTRUCTION_SET_TRIGGER_COMPLETE: Instructs the Scheduler that the Trigger should be put in the COMPLETE state. It essentially skips the misfired trigger.

INSTRUCTION_DELETE_TRIGGER: Instructs Scheduler that the Trigger wants itself deleted.

INSTRUCTION_SET_TRIGGER_ERROR: Instructs the Scheduler that Trigger should be put in the error state.

Simple Triggers - These misfire instructions are applicable to only "simple" triggers.

MISFIRE_INSTRUCTION_FIRE_NOW: Instructs the Scheduler that upon a misfire situation, the trigger wants to be fired now by the Scheduler. NOTE This instruction should typically only be used for 'one-shot' (non-repeating) Triggers. If it is used on a trigger with a repeat count > 0 then it is equivalent to the instruction MISFIRE_INSTRUCTION_RESCHEDULE_NOW_WITH_REMAINING_REPEAT_COUNT.

MISFIRE_INSTRUCTION_RESCHEDULE_NOW_WITH_EXISTING_REPEAT_COUNT: Instructs the Scheduler that upon a misfire situation, the trigger wants to be re-scheduled to with the repeat count left as-is. **NOTE**: Use of this instruction causes the trigger to 'forget' the start-time and repeat-count that it was originally setup with. **NOTE**: This instruction could cause the Trigger to go to the 'COMPLETE' state after firing 'now', if all the repeat-fire-times where missed.

MISFIRE_INSTRUCTION_RESCHEDULE_NOW_WITH_REMAINING_REPEAT_COUNT: Instructs the Scheduler that upon a misfire situation, the trigger wants to be re-scheduled to 'now' with the repeat count set to what it would be, if it had not missed any firings. NOTE: Use of this instruction causes the trigger to 'forget' the start-time and repeat-count that it was originally setup with (this is only an issue if you for some reason wanted to be able to tell what the original values were at some later time). NOTE: This instruction could cause the trigger to go to the 'COMPLETE' state after firing 'now', if all the repeat-fire-times where missed.

MISFIRE_INSTRUCTION_RESCHEDULE_NEXT_WITH_REMAINING_COUNT: Instructs the Scheduler that upon a misfire situation, the trigger wants to be re-scheduled to the next scheduled time after 'now' and with the repeat count set to what it would be, if it had not missed any firings. NOTE/WARNING: This instruction could cause the trigger to go directly to the 'COMPLETE' state if all fire-times where missed.

MISFIRE_INSTRUCTION_RESCHEDULE_NEXT_WITH_EXISTING_COUNT: Instructs the Scheduler that upon a misfire situation, the trigger wants to be re-scheduled to the next scheduled time after 'now' and with the repeat count left unchanged. NOTE: Use of this instruction causes the trigger to 'forget' the repeat-count that it was originally setup with. NOTE/WARNING: This instruction could cause the trigger to go directly to the 'COMPLETE' state if all fire-times where missed.

CRON Triggers - these apply to the "cron" type triggers

MISFIRE_INSTRUCTION_FIRE_ONCE_NOW: Instructs the Scheduler that upon a misfire situation, trigger wants to be fired now by the Scheduler.

MISFIRE_INSTRUCTION_DO_NOTHING: Instructs the Scheduler that upon a misfire, the trigger wants to have it's next-fire-time updated to the next time in the schedule after the current time (taking into account any associated <code>{@link Calendar}</code>, but it does not want to be fired now.

Triggers

This page last changed on Dec 12, 2006 by dmoran.

The Pentaho BI Platform currently supports two different types of triggers.

Simple Trigger - A simple trigger allows a task to be scheduled at a specified regular interval for a specified number of repetitions. The inputs to the simple trigger are the integer values "repeatInterval" (in seconds) and "repeatCount". The trigger will begin firing immediately and continue at the "repeatInterval" for "repeatCount" number of cycles. See Figure X.X -- Sample StartJob Action Sequence for an example.

Cron Trigger - selected in the "triggerType" node as "cron". This trigger uses unix style cron task definitions. The cron trigger takes a "cronString" that represents the trigger definition much like an entry into crontab. The following excerpt from the javadoc for org.quartz.cronTrigger describes the format of the cron expression string.

A "cronString" is a string comprised of 6 or 7 fields separated by white space. The 6 mandatory and 1 optional fields are as follows:

Field Name	Allowed Values	Allowed Special Characters
Seconds	0-59	, - * /
Minutes	0-59	, - * /
Hours	0-23	, - * /
Day-of-month	1-31	, - * ? / L W C
Month	1-12 or JAN-DEC	, - * /
Day-of-Week	1-7 or SUN-SAT	, - * ? / L C #
Year (Optional)	empty, 1970-2099	, - * /

The **'' character is used to specify all values. For example, ""** in the minute field means "every minute".

The '?' character is allowed for the day-of-month and day-of-week fields. It is used to specify 'no specific value'. This is useful when you need to specify something in one of the two fileds, but not the other. See the examples below for clarification.

The '-' character is used to specify ranges For example "10-12" in the hour field means "the hours 10, 11 and 12".

The ',' character is used to specify additional values. For example "MON,WED,FRI" in the day-of-week field means "the days Monday, Wednesday, and Friday".

The '/' character is used to specify increments. For example "0/15" in the seconds field means "the seconds 0, 15, 30, and 45". And "5/15" in the seconds field means "the seconds 5, 20, 35, and 50". You can also specify '/' after the **'' character - in this case** '' is equivalent to having '0' before the '/'.

The 'L' character is allowed for the day-of-month and day-of-week fields. This character is short-hand for "last", but it has different meaning in each of the two fields. For example, the value "L" in the day-of-month field means "the last day of the month" - day 31 for January, day 28 for February on non-leap years. If used in the day-of-week field by itself, it simply means "7" or "SAT". But if used in the day-of-week field after another value, it means "the last xxx day of the month" - for example "6L" means "the last friday of the month". When using the 'L' option, it is important not to specify lists, or ranges of values, as you'll get confusing results.

The 'W' character is allowed for the day-of-month field. This character is used to specify the weekday (Monday-Friday) nearest the given day. As an example, if you were to specify "15W" as the value for the day-of-month field, the meaning is: "the nearest weekday to the 15th of the month". So if the 15th is a Saturday, the trigger will fire on Friday the 14th. If the 15th is a Sunday, the trigger will fire on Monday the 16th. If the 15th is a Tuesday, then it will fire on Tuesday the 15th. However if you specify "1W" as the value for day-of-month, and the 1st is a Saturday, the trigger will fire on Monday the 3rd, as it will not 'jump' over the boundary of a month's days. The 'W' character can only be specified when the day-of-month is a single day, not a range or list of days.

The 'L' and 'W' characters can also be combined for the day-of-month expression to yield 'LW', which translates to "last weekday of the month".

The '#' character is allowed for the day-of-week field. This character is used to specify "the nth" XXX day of the month. For example, the value of "6#3" in the day-of-week field means the third Friday of the month (day 6 = Friday and "#3" = the 3rd one in the month). Other examples: "2#1" = the first Monday of the month and "4#5" = the fifth Wednesday of the month. Note that if you specify "#5" and there is not 5 of the given day-of-week in the month, then no firing will occur that month.

The 'C' character is allowed for the day-of-month and day-of-week fields. This character is short-hand for "calendar". This means values are calculated against the associated calendar, if any. If no calendar is associated, then it is equivalent to having an all-inclusive calendar. A value of "5C" in the day-of-month field means "the first day included by the calendar on or after the 5th". A value of "1C" in the day-of-week field means "the first day included by the calendar on or after sunday".

The legal characters and the names of months and days of the week are not case sensitive.
Here are some full examples:

Expression	Meaning
"0 0 12 * * ?"	Fire at 12pm (noon) every day
"0 15 10 ? * *"	Fire at 10:15am every day
"0 15 10 * * ?"	Fire at 10:15am every day
"0 15 10 * * ? *"	Fire at 10:15am every day
"0 15 10 * * ? 2005"	Fire at 10:15am every day during the year 2005
"0 * 14 * * ?"	Fire every minute starting at 2pm and ending at 2:59pm, every day
"0 0/5 14 * * ?"	Fire every 5 minutes starting at 2pm and ending at 2:55pm, every day
"0 0/5 14,18 * * ?"	Fire every 5 minutes starting at 2pm and ending at 2:55pm, AND fire every 5 minutes starting at

	6pm and ending at 6:55pm, every day
"0 0-5 14 * * ?"	Fire every minute starting at 2pm and ending at 2:05pm, every day
"0 10,44 14 ? 3 WED"	Fire at 2:10pm and at 2:44pm every Wednesday in the month of March
"0 15 10 ? * MON-FRI"	Fire at 10:15am every Monday, Tuesday, Wednesday, Thursday and Friday
"0 15 10 15 * ?"	Fire at 10:15am on the 15th day of every month
"0 15 10 L * ?"	Fire at 10:15am on the last day of every month
"0 15 10 ? * 6L"	Fire at 10:15am on the last Friday of every month
"0 15 10 ? * 6L 2002-2005"	Fire at 10:15am on every last friday of every month during the years 2002, 2003, 2004 and 2005
"0 15 10 ? * 6#3"	Fire at 10:15am on the third Friday of every month

Secure Filter (Prompting)

This page last changed on Feb 14, 2007 by wgorman.

The secure filter component has two separate but related functions. It allows you to customize the default prompting done by the runtime context and can verify that the only valid selections are returned.

From the example below, the Secure Filter Component will check for a parameter named DEPARTMENT, if it doesn't exist, it will generate the prompt and add values to select from based on the values in the "Display" column of the DEPARTMENT_FILTER parameter.

When the selection has been made by the user, and DEPARTMENT is passed back to the Action Sequence and eventually to the Secure Filter component, it will verify that the value returned exists in the "Dept" column.

If the value of DEPARTMENT is valid, execution of the Action Sequence will continue on to the next component in the sequence.

```
<action-definition>
 <component-name>SecureFilterComponent</component-name>
 <action-inputs>
 <DEPARTMENT type="string"/>
 <DEPARTMENT_FILTER type="result-set"/>
 </action-inputs>
 <action-outputs/>
 <component-definition>
 <selections>

 <DEPARTMENT optional="false" style="select" prompt-if-one-value="true">
 <filter value-col-name="Dept" display-col-name="Display">
 DEPARTMENT_FILTER
 </filter>
 <title>Select the Department</title>
 </DEPARTMENT>

 </selections>
 <xsl>CustomReportParameters.xsl</xsl>
 <target>Report_Window</target>
 </component-definition>
</action-definition>
```

The **optional** attribute specifies if the parameter is required or not. If required, the user must fill in the value before continuing.

The **style** attribute defines the style of control that will be presented to the user

The **prompt-if-one-value** attribute if checked still prompt the user, even if there is only one choice

The **title** attribute defines text description that will be presented to the user

TODO

- Format this page like the others
- Verify completeness

SQL Execute

This page last changed on Feb 28, 2007 by gmoran.

Performs a data or database modifying SQL query against a JDBC or JNDI data source. The result of execution is a resultset containing number of rows affected and status, as detailed below. The query may contain references to parameters by enclosing the parameter in curly braces. For example update myTable set column = 'newvalue' where department = '{dept}' would replace {dept} with the value of the parameter named "dept".

Component Name: SQLExecute

Component Definition:

REQUIRED

JNDI Name - Name of the JNDI connection.

or the following JDBC information

JDBC Driver - The JDBC driver to use when connecting to the data source.
JDBC Connection String - The connection string identifying the location of the data source.
JDCB User Name - User name to use when connecting to the data source.
JDBC Password - Password to use when connecting to the data source.

Query Definition - the update, create, alter or drop statement that you wish to execute. This component will execute more than one statement in a query definition.

OPTIONAL

force single statement (true\false) - if set to true, and there are multiple statements in the query definition, will force the query to be submitted to the server as one statement.This execution path should be used if the query has a semi-colon in the text of the SQL statement. This is a legitimate condition if there is (for example) a statement with a where-clause that has a semi-colon.

e.g.: UPDATE sometable SET somecolumn='val1;val2' WHERE somecolumn='val3;val4'

multi-statement separator (single character value) - defaults to ";". Allows you to change the statement separator when multiple sql statements are present

continue on exception (true\false) - if set to true, continue to execute following sql statements after an exception is thrown. This is useful when a database throws an exeception on a DROP TABLE statement because the table does not exist.

Note: Any parameter may be referenced in the query by enclosing the parameter in curly braces.

Outputs:

Result Set Name -- The name of the parameter in which to store the query results. The results of this query will be a resultset that has two columns: the first for the number of rows affected, and the second for the status of the query execution, success or failed. If more than one statement is executed independently, then there will be as many rows in the resultset as were queries executed. For example, if you defined three update statements in the query definition, then there would be three rows in the resulting resultset, one for each statement executed.

SQL Query

This page last changed on Mar 06, 2007 by wgorman.

Performs a SQL query against a JDBC or JNDI data source. The query results are available as an output to the action. The query may contain references parameters be enclosing the parameter in curly braces. For example select * from myTable where department = '{dept}' would replace {dept} with the value of the parameter named "dept".

Component Name: SQLLookupRule

Inputs:

REQUIRED
JNDI Name - Name of the JNDI connection.

or an SQL Prepared Component object

Prepared Component - A previous SQLLookupRule that defined a prepared component as the output. The previous SQLLookupRule will share it's connection with the current SQLLookupRule

or the following JDBC information

JDBC Driver - The JDBC driver to use when connecting to the data source.
JDBC Connection String - The connection string identifying the location of the data source.
JDCB User Name - User name to use when connecting to the data source.
JDBC Password - Password to use when connecting to the data source.

OPTIONAL
Keep Connection Open _-- If selected the query results are retrieved from the data source as needed, otherwise the entire query results are brought over from the data source and cached in memory. It is recommended that you set this option with larger datasets. Never use this option if you are storing the resultset in the session.

SQL Query - This is required if no prepared component object is defined. to utilize SQL's prepared statement functionality.

Use {PREPARE:<paramname>} in the sql query, where <paramname> is an input parameter to the SQLLookupRule. This parameter is resolved during the execution of the SQLLookupRule.

Use {PREPARELATER:<paramname>} in the sql query when defined as a prepared component (see outputs below). The <paramname> and value is provided by the executing component.

Note: Any parameter may be referenced in the query by enclosing the parameter in curly braces.

Outputs:

Result Set Name-- The name of the parameter in which to store the query results.

or

Prepared Component - The name of the parameter in which to store the prepared component object, which can be used for shared connections and later processing by other components.