pentaho®
open source business intelligence™

Integrating Pentaho Software and Content

Help and Support Resources

If you have questions that are not covered in this guide, or if you would like to report errors in the documentation, please contact your Pentaho technical support representative.

Support-related questions should be submitted through the Pentaho Customer Support Portal at http://support.pentaho.com.

For information about how to purchase support or enable an additional named support contact, please contact your sales representative, or send an email to sales@pentaho.com.

For information about instructor-led training on the topics covered in this guide, visit http://www.pentaho.com/training.

Limits of Liability and Disclaimer of Warranty

Trademarks

Company Information

Pentaho Corporation
Citadel International, Suite 340
5950 Hazeltine National Drive
Orlando, FL 32822
Phone: +1 407 812-OPEN (6736)
Fax: +1 407 517-4575
http://www.pentaho.com

E-mail: communityconnection@pentaho.com

Sales Inquiries: sales@pentaho.com

Documentation Suggestions: documentation@pentaho.com

Sign-up for our newsletter: http://community.pentaho.com/newsletter/

Contents

Introduction....5
Required Knowledge and Expertise....5
Required Software....5
Defining the BI Platform, BI Server, and BI Suite....5
What We Mean By "Integrating"....6
BI Server Capabilities and Features....8
Input Types....8
Output Types....8
Engines and Content Creation....8
Security Integration....8
Scheduling and Distribution....9
Explanation of the BI Server Example Application....10
Deploying pentaho_integration_examples.war....11
Integrating BI Server Functionality....12
Using ViewAction to Retrieve Content....12
Understanding the ViewAction Content Generator....12
Generating an HTML Report....13
Generating an HTML Report With a Form....13
Using ReportViewer to Generate User-Driven Reports....14
Understanding the ReportViewer Content Generator....14
Analyzing the Top N Customers....16
Displaying Content With the Reporting URL....16
Understanding the Reporting URL Content Generator....16
Static Display of the Top Three Customers....17
Creating and Displaying Analyzer and Interactive Reporting Content....17
Understanding Analyzer Service URLs....17
Displaying an Analyzer Report in Viewer Mode....19
Displaying an Analyzer Report in Editor Mode....19
Creating a New Analyzer Report....19
Integrating Pentaho Web-Based Client Tools....19
Running Pentaho Analyzer In an iframe....20
Running Interactive Reporting In an iframe....23
Running Ad Hoc Reporting In an iframe....26
Listing Content with the SolutionRepositoryService....29
Understanding the SolutionRepositoryService....29
Retrieving a List of Solution Files....31
Getting Details for a Solution File....31
Creating a Solution Folder....31
Deleting a File....31
Retrieving Solution File and Directory Permission Settings....32
Setting Permissions On a Solution File....32
Using XML Services With ServiceAction....32
Understanding ServiceAction....33
Executing an Action Sequence With a SOAP Response....33
Retrieving the Security Model....34
Using the Ajax API For Asynchronous Execution....34
Understanding the Ajax API....34
Executing an Action Sequence With JavaScript....35
Tutorials in PHP, .NET, HTML, and JSP....36
Creating a Simple iFrame-Based Dashboard....36
Result Set Streaming With Pentaho Web Services....36
Using the Pentaho Solution Repository Web Service to Retrieve a List of Content....36
Other Embedding Scenarios....37
Embedding the Core BI Platform....37
Embedding the Reporting Engine....37

Embedding the Analysis Engine......37
Developer Support......38
License Information......39
Obtaining the Source Code......40

Introduction

This document provides guidance and instructions for integrating Pentaho content (reports, action sequences, dashboards, charts) and some of the functionality of the Pentaho BI Server and Web-based client tools into a new or existing Web application.

The Pentaho BI Server is a highly extensible, embeddable, and scriptable business intelligence Web application. Internally it is composed of content-generating engines that provide reporting, online analytical processing (OLAP), and data integration (ETL) functionality. These engines are managed by the Pentaho BI Platform process-flow engine, which was designed for but not exclusive to running such business intelligence tasks as: retrieving data from multiple disparate data sources, creating data-driven reports and other content, and scheduling and conditionally automating content delivery. End users interact with these services through an ad-hoc reporting interface, a choice of OLAP visualization tools (Pentaho Analyzer and JPivot), a dashboard designer, and a convenient scheduling interface.

There are many code examples in this document that show interesting and useful ways to integrate client tools, BI Server functionality, and content into your Web application. Most of these examples are packaged into a Web application archive (WAR) named **pentaho_integration_examples.war** and distributed along with this guide. It is designed to be deployed alongside an operational Pentaho BI Server Enterprise Edition version 3.9.

You should read this guide in the order it is presented, from this point all the way to the end of the ViewAction example. Follow the other examples if they will be useful for your project. The remaining portion of this document contains tutorials for specific client tools and Web development languages; extra information about other ways you can embed or integrate core BI Server functionality into an application; licensing of the examples and the BI Server itself; how to get the BI Server source code; and information on where to get help and support for your BI Server embedding or integration project.

Required Knowledge and Expertise

This document is designed to accommodate Web developers with JavaScript and HTML experience. You do not need to have any Java or JSP knowledge to follow and implement the examples in the Integrating BI Server Functionality portion of this guide. The Tutorials section is designed specifically for application developers, and will require expert knowledge in the programming languages used in each tutorial.

Required Software

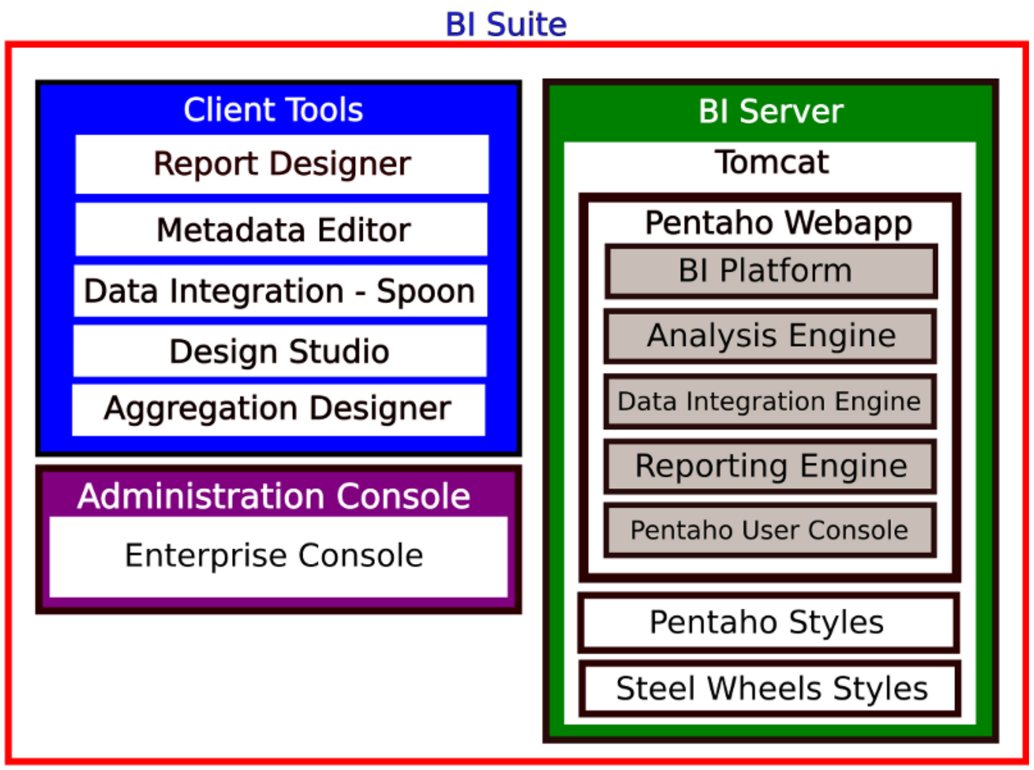
The following software is required to execute all of the examples in this guide:

- A Pentaho-supported Web browser
- A Pentaho BI Server Enterprise Edition version 3.9 running on Apache Tomcat 6.0 or later; or JBoss 5.1 or later
- To run all of the examples, you will need Enterprise Edition licenses installed for: Pentaho BI Platform, Pentaho Analysis, Pentaho Interactive Reporting, Pentaho Dashboard Designer.

Pentaho is operating system agnostic. You can use any reasonable version of Windows, Linux, BSD, or OS X as long as you can install the above-listed software packages on it. You can use a fully 32-bit, fully 64-bit, or mixed software environment as long as all of your libraries are compiled for the same architecture.

Defining the BI Platform, BI Server, and BI Suite

The way Pentaho defines its business intelligence software infrastructure can be confusing at a glance. This diagram shows how all of the software in the BI Suite is categorized and defined:



The BI Suite is all of the software inside of the red box. It's the BI Server plus standalone client tools and a graphical interface for managing data sources, users and roles, and other administrator functions (the Pentaho Enterprise Console). Enterprise Console is an extension of the open source Pentaho Administration Console; it is only available to Pentaho Enterprise Edition customers.

The BI Server is a Java Web application that contains specialized content creation engines plus a graphical user interface (the Pentaho User Console) for interacting with them. Everything inside of the green box is considered part of the BI Server. As you can see, this includes an application server (Tomcat) and three WARs: one for the Pentaho Web application, and two optional WARs that provide style sheets for BI Server content.

The BI Platform is a process-flow engine that forms the operational core of the BI Server. It ties the other content engines -- Reporting, Analysis, and Data Integration -- to the Pentaho User Console to provide content display, delivery, and scheduling functionality. Additionally, the BI Platform offers a powerful scripting framework for conditionally automating tasks. Typically the BI Platform is used to automate business intelligence tasks that rely on other engines, but it could theoretically be used for practically any logical task.

The Reporting (JFreeReport) engine provides functions which create and render data-driven reports and charts in a variety of formats.

The Pentaho Analysis (Mondrian) provides the ability to create an analysis schema, and to form data sets from that schema by using an MDX query.

The Data Integration (Kettle) engine enables Data Integration jobs and transformations to run in conjunction with other BI Server processes.

What We Mean By "Integrating"

Pentaho created this guide in response to frequent customer requests to embed certain BI Server functionality into various third-party applications. There are two technical approaches to implementation:

Embedding

Embedding refers to adding individual pieces of the Pentaho BI Server into a third-party application on a code level without having to run a separate BI Server instance.

Embedding is what Pentaho customers usually expect to have to do, but it is rarely the best solution for them. Customers who request help with this process frequently expect to be able to copy certain JARs and code snippets from the BI Server to their Web application. However, the highly interconnected nature of the BI Server prevents this from being a quick and easy process. This isn't to say that it's impossible to embed only Interactive Reporting or

Analyzer or some other piece of the Pentaho User Console without relying on a complete BI Server instance to run in the background, but it is a highly complex operation that requires skilled programmers and lots of developer support resources from Pentaho's services and engineering teams.

Integrating

Integrating refers to running a fully operational Pentaho BI Server instance in order to access its content and use its functionality in other applications.

This scenario accomplishes most or all of the goals of embedding BI Server pieces without the development overhead. Instead of, for instance, embedding Interactive Reporting into your custom Web application, you can simply display it in an iframe. All of the Interactive Reporting functionality that you get through the Pentaho User Console will be available to you when you integrate it in this manner.

BI Server Capabilities and Features

Before you think about integrating or embedding functionality in the BI Server, you should be fully aware of what it can do. This section quickly explains the BI Server's diverse capabilities.

Input Types

The Pentaho Reporting engine can connect to virtually any data source:

- JDBC
- JNDI
- Kettle (Pentaho Data Integration)
- Simple SQL (JDBC Custom)
- Pentaho Metadata
- Mondrian MDX
- OLAP4J
- XML
- Simple table
- Scripting data sources (JavaScript, Python, TCL, Groovy, BeanShell)
- Java method invocation
- Hibernate

If your data source is not directly supported, you can use Pentaho Data Integration to transform it into a more report-friendly format, or you can design your own custom data source interface.

Output Types

The Pentaho Reporting engine can create reports in a variety of relevant file formats:

- PDF
- HTML
- Excel
- CSV
- RTF
- XML
- Plain text

All of the output types listed above are highly customizable in terms of style, formatting, and pagination. You can also specify your own output type if none of the standard choices are sufficient.

Engines and Content Creation

The BI Platform is a lightweight process-flow engine that defines the order of execution of one or more the components of the Pentaho BI Platform. It does not generate content itself, so it has to rely on separate engines for creating reports and other content. The BI Server is designed for business intelligence operations and is therefore packaged with the JFreeReport reporting engine, the Mondrian analysis engine, and the Kettle data integration engine. Each of these engines is potentially embeddable as a standalone resource.

Security Integration

Pentaho relies on the Spring Security pluggable authentication framework. By default, the BI Server uses a JDBC-based data access object that is tied to a Hibernate database. Users and roles are configured through the Pentaho Enterprise Console, and content authorization is controlled by the BI Server administrator. However, you can easily configure the server to use existing security tables in a different database, or to authenticate through your existing

LDAP (including Active Directory) server or Central Authentication Service. Pentaho's security is also extensible to the point that you can create your own custom data access object, or completely remove all authentication functionality.

Scheduling and Distribution

The BI Server is programmable directly through Java code, or dynamically through XML files known as **action sequences**. An action sequence contains a set of BI Platform actions and parameters (input, output, and external resources) in an XML file with a **.xaction** extension. For this reason, action sequences are sometimes called xactions.

Action sequences activate BI Platform components and compel them to do work. They can be run at any time while the BI Server is active. You can also arrange for action sequences to be run at certain times or intervals through the built-in scheduling service.

Explanation of the BI Server Example Application

The examples in this guide are delivered as a single J2EE Web application archive (WAR) called **pentaho_integration_examples**. This WAR must be deployed to the same application server that your Pentaho BI Server (**pentaho.war**) is running on. All Web applications that you integrate BI Server functionality into must also be deployed to the same application server as a WAR (or EAR, if you are using JBoss).

The **pentaho_integration_examples.war** contains all of the HTML files that are explained in this guide, plus a `/WEB-INF/` directory with a barebones web.xml configuration file in it.

Feel free to make changes to this WAR and all of the files in it.

Note: The examples in this WAR will not work properly without current Pentaho BI Platform, Pentaho Dashboard Designer, and Pentaho Analysis Enterprise Edition licenses.

Note: You must be logged into the Pentaho User Console as an administrator user in order to follow all of the examples. If you are not logged in, you will be redirected to the Pentaho User Console login page before the example executes.

If you are reading this, you've unpacked the zip file containing the Integrating With the Pentaho BI Server PDF and the pentaho_integration_samples.war example application. If you have come to these instructions in some other format and do not have the example application, contact your Pentaho sales or support representative to find out how you can obtain the latest version of this guide, including the example application.

You can access the example index page by opening a Web browser and navigating to *http://localhost:8080/pentaho_integration_examples/*, modifying the URL to accommodate your BI Server hostname and port number.

Deploying pentaho_integration_examples.war

You must have an operational Pentaho BI Server Enterprise Edition version 3.9 deployed in order to continue.

Follow the instructions below to deploy the **pentaho_integration_examples.war** file to your Pentaho application server.

Note: This application is intended to be deployed to a testing, development, or evaluation server. While there should be nothing explicitly harmful in the example application, you will be able to access and change BI Server content with the examples. For this and other security and performance reasons, Pentaho recommends that you **do not deploy this application to your production BI Server instance.**

1. Stop your Java Web application server.

   ```
   /home/pgibbons/pentaho/server/ctlscript.sh stop
   ```
2. Copy the **pentaho_integration_examples.war** file to the **webapps** (for Tomcat) or **deploy** (for JBoss) directory.

   ```
   cp /home/pgibbons/Desktop/pentaho_integration_examples.war ../pentaho/server/biserver-
   ee/tomcat/webapps/
   ```

   pentaho_integration_examples.war should be a sibling to the deployed pentaho.war.
3. Start your Java Web application server.

   ```
   /home/pgibbons/pentaho/server/ctlscript.sh start
   ```

The WAR should now be unpacked and properly running in your application server. You can access the example index page by opening a Web browser and navigating to *http://localhost:8080/pentaho_integration_examples/*, modifying the URL to accommodate your BI Server hostname and port number.

Integrating BI Server Functionality

The simplest methods of integrating content generated from the Pentaho BI Server are:

1. **Direct services** provided via servlets that deliver content in the URL outputstream.
2. **Web services** provided via servlets that deliver content packaged as a SOAP response.

This section shows several different Pentaho BI Server services that deliver content using these methods. The only prerequisite for implementing them is an operational Pentaho BI Server with Enterprise Edition licenses installed for the BI Platform, Analysis, and Dashboards. The below examples depend on the sample data and solutions that are typically distributed with the Pentaho BI Server. If you do not have the samples installed with your BI Server, you may have to obtain a BI Suite evaluation installer from the Pentaho Knowledge Base or subscription FTP site, and install it to a development workstation or test server.

The Pentaho BI Server is capable of serving content using only a definition file. For example, a report created with Report Designer can be served from the BI Server directly from the .PRPT definition file. This applies to Analyzer reports, Dashboard Designer views, and Pentaho Data Integration results as well. The BI Server is also capable of processing several steps -- or *actions* -- sequentially and returning the resulting output or content.

The services described below help you identify what method or service to use when you want to integrate Pentaho content into your environment, and when to use each service.

Using ViewAction to Retrieve Content

The examples in this subsection explain how to generate reports by building URLs.

Understanding the ViewAction Content Generator

What is ViewAction?

ViewAction is a servlet included in the Pentaho BI Server that executes action sequences and returns their output in the specified format. The format can vary based on the supported MIME types, and the output definition of each action sequence. For more detailed information on action sequences, see the *Creating Action Sequences* document in the Pentaho Knowledge Base.

ViewAction has required and optional operational parameters (described below) and will also accept action sequence input parameters. Should the action sequence require a parameter, and this parameter is not supplied by the caller, then the server will return an HTML page requesting the missing parameters instead of the requested content.

When to Use ViewAction

ViewAction is most obviously useful when you want to execute an action sequence, as opposed to generating content directly from content definition files. The other considerations for using ViewAction are:

1. You are able to directly reference a URL in the context of the Pentaho Web application.
2. You can use either POST or GET HTTP methods for sending parameter values to ViewAction, as shown in the examples.

How to Use ViewAction

ViewAction is executed as an HTTP URL, and as such, follows the syntax rules for Web-based location strings. Both operational parameters and action sequence input parameters are specified after the query symbol (?) as name/value pairs.

Operational ViewAction Parameters

Parameter	Data Type	Description
solution	String	**Required.** The name of the solution where the action sequence is located.
path	String	**Required.** The relative path from the solution name to where the action sequence is located.

Parameter	Data Type	Description
action	String	**Required.** The name of the action sequence to execute.
instance_id	Integer	The instance ID of a previous server runtime context.
debug	Boolean	Set to **true** in order to have debug information written to the execution log.
run_as_background	Boolean	Set to **Yes** to run the action sequence in the background. Content generated as the result of a background execution is not added to the current outputstream; instead, it can be found in the Workspace panel of the Pentaho User Console.

Generating an HTML Report

This example demonstrates how to display the result of an action sequence that generates a tabular HTML report. The example uses both the required and optional operational parameters as well as input parameters from the action sequence.

```
http://localhost:8080/pentaho/ViewAction?
  outputType=html
  &Region=NA
  &run_as_background=No
  &solution=steel-wheels
  &action=Sales_by_Supplier.xaction
  &path=dashboards/Widget%20Library/Report%20Snippets
```

The **solution**, **path** and **action** parameters are commonly required on many of the service URLs that are provided for integrating Pentaho BI Server content. Here you can see the optional parameter **run_as_background** demonstrated. The parameters **Region** and **outputType** are both input parameters that the action sequence needs in order to execute the report generation action. Note that the case that you use specifying input parameters on the URL must match the case used in the action sequence definition.

You can experiment with the dynamic parameter handling built into the action sequence service by removing the **outputType=html** from the example URL. You should see an HTML form prompting for the output type instead of receiving the report output.

Generating an HTML Report With a Form

The recommended approach for handling parameter prompting with action sequences is to use the built-in parameter in conjunction with the **SecureFilterComponent** (see the *Creating Action Sequences* document for more information on BI Platform action sequence components). The templates provided for built-in parameter handling are customizable. However, you may have an existing Web interface you would like to use.

Generating the same report using a custom Web form is possible, as this example demonstrates. This is a useful means of using ViewAction if you have an existing application user interface that you are trying to integrate Pentaho content into.

```
<form method="post" action="http://localhost:8080/pentaho/ViewAction" name="form1" >

  <span class="style3">Select Output Format:</span>
  <select name="outputType">
    <option value="html">Web Page</option>
    <option value="pdf">Adobe PDF</option>
    <option value="xls">Excel</option>
  </select>

  <br/>

  <span class="style3">Select Region:</span>
  <select name="Region">
    <option value="NA">North America</option>
    <option value="APAC">Asia Pacific</option>
  </select>
  <input type="hidden" name="solution" value="steel-wheels" />
  <input type="hidden" name="path" value="dashboards/Widget Library/Report Snippets" />
  <input type="hidden" name="action" value="Sales_by_Supplier.xaction" />
  <input type="submit" name="b1" value="Call Report" >

</form>
```

Note that the same parameters are in play as in the previous example, but this implementation uses the HTTP POST method of form submission versus passing all of the parameters in the URL.

Using ReportViewer to Generate User-Driven Reports

This subsection contains an example of how to render a report with paging and output controls.

Understanding the ReportViewer Content Generator

What is ReportViewer?

The Pentaho ReportViewer is a GWT application plugin that will render Pentaho reports with optional interactive features:

- **Page controls** for paginated reports
- **Filter components** such as text boxes, drop down list boxes, and radio buttons for parameter filters

When to Use ReportViewer

The ReportViewer renders PRPT report definitions into reports. It does not need (and is unable to execute) an action sequence to render a report.

When you want to deliver an interactive report, you should use the ReportViewer. When a report is published to the BI Server from a Pentaho client tool (such as Report Designer), the Pentaho User Console uses ReportViewer to render it. ReportViewer is accessed via HTTP URL, so it is easy to integrate into any Web-based application.

How to Use ReportViewer

ReportViewer is executed as an HTTP URL, and as such, follows the syntax rules for Web-based location strings. Both operational parameters and report input parameters are specified after the query symbol (?) as name/value pairs.

Operational ReportViewer Parameters

Parameter	Data Type	Description
solution	String	**Required.** The name of the solution where the PRPT report definition file is located.
path	String	**Required.** The relative path from the solution name to where the PRPT file is located.
name	String	**Required.** The name of the PRPT file to render.
accepted-page	Integer	If a report is paginated, this can be set to the page number to be rendered; if a report is not paginated, this defaults to zero.
renderMode	String	Determines how the Reporting engine renders the PRPT. See the Render Modes table below for options.
output-target	String	Defines the report output type. See the Output Targets table below for options.
dashboard-mode	Boolean	If **true** and the export generates HTML, the report will generate a body-fragment (content without the HTML,

Parameter	Data Type	Description
		HEAD and BODY tags and all styles inlined into "style" attributes), which is easy to include in dashboards or other HTML pages. This is the same result as if the report configuration property **org.pentaho.reporting.engine.classic .core.modules.output .table.html.BodyFragment** is set to **true**.
subscribe	Boolean	An internal flag indicating that this report is registered with the BI Server and set to run on a schedule.
subscription-id	String	An internal parameter that specifies the schedule ID for this content.
subscription-name	String	An internal parameter that specifies the schedule name for this content.
destination	String	If email settings are configured in the BI Platform, this parameter will allow you to specify a single email address to send the rendered report to.
print	Boolean	Determines whether the report will be printed. Overrides all other output properties.
printer-name	String	The (optional) name of the printer if **print** is set to **true**.

Output Targets

The following **output-target** options define output types that are supported in the Reporting engine:

Option	Purpose
table/html;page-mode=stream	HTML as a single page, all report pagebreaks are ignored.
table/html;page-mode=page	HTML as a sequence of physical pages, manual and automatic pagebreaks are active.
application/vnd.openxmlformats-officedocument.spreadsheetml.sheet;page-mode=flow	Excel 2007 XLSX Workbook
table/excel;page-mode=flow	Excel 97 Workbook
table/csv;page-mode=stream	CSV output
table/rtf;page-mode=flow	Rich text format
pageable/pdf	PDF output
pageable/text	Plain text
pageable/xml	Pageable layouted XML
table/xml	Table-XML output
pageable/X-AWT-Graphics;image-type=png	A single report page as PNG.
mime-message/text/html	MIME email with HTML as body text and all style and images as inline attachments.

Render Modes

The Reporting engine has a number of functional modes:

Option	Purpose
REPORT	Renders the report.
XML	Returns the parameter description document for the UI.
PARAMETER	Same as XML, but does not perform any pagination.
SUBSCRIBE	Used for managing schedules in the BI Server.
DOWNLOAD	Downloads the PRPT file, if the user has the correct permissions.

Analyzing the Top N Customers

This example report demonstrates all the features of ReportViewer. The example uses both the required and optional operational parameters as well as input parameters for the report.

```
http://localhost:8080/pentaho/content/reporting/reportviewer/report.html?
  solution=steel-wheels
  &path=/reports
  &name=Top N Analysis.prpt
  &sLine=[Product].[All Products].[Classic Cars]
  &sMarket=[Markets].[All Markets].[NA]
  &sYear=[Time].[All Years].[2003]
  &TopCount=3
  &output-type=text/html
  &accepted-page=0
  &paginate=false
```

Again **solution**, **path**, and **name** are required in order to locate the report within the BI Server's solution repository. The report input parameters **sLine**, **sMarket**, **sYear**, and **TopCount** provide the initial values for the report filters that are defined in the report. These filters provide the analysis and interactivity for the report. The report also accepts the report **output-type** as a parameter, allowing the user to choose from many output options, such as HTML, PDF, CSV and others. Last, since the data is not paginated, **accepted-page** is set to zero, and the page controls are turned off with the **paginate** parameter.

Displaying Content With the Reporting URL

This subsection contains an example of how to render reports in a variety of output formats.

Understanding the Reporting URL Content Generator

What is the Reporting URL?

The Reporting URL is a servlet included in the BI Server that enables execution of a Pentaho report and returns the rendered output in the specified format.

When to Use the Reporting URL

In comparison with the ReportViewer, this service has no bells or whistles. This service expects that the report has all parameters satisfied using either defaults or name/value pairs in the URL, and these values are not intended to be changed once the report is rendered. The report also has no pagination needs.

When your reporting needs are relatively static, this is a good service to use because it trims the overhead that comes along with the ReportViewer GWT application. The Reporting URL submits the report definition and its parameters directly to the reporting content generator for processing, returning only the report content that was requested.

How to Use the Reporting URL

The Reporting URL is an HTTP URL, and as such, follows the syntax rules for Web-based location strings. Both operational parameters and report input parameters are specified after the query symbol (?) as name/value pairs.

Operational Reporting URL Parameters

Parameter	Data Type	Description
solution	String	**Required.** The name of the solution where the PRPT definition file is located.
path	String	**Required.** The relative path from the solution name to where the PRPT file is located.
name	String	**Required.** The name of the PRPT file to render.

Static Display of the Top Three Customers

This example should look almost identical to the ReportViewer example earlier in this section, except that the URL context has changed from **/content/reporting/reportviewer/report.html** to **/content/reporting**. This change, along with removing some unnecessary parameters relevant to paging, is the only difference between integrating with the Reporting URL and the ReportViewer.

```
http://localhost:8080/pentaho/content/reporting?
  solution=steel-wheels
  &path=/reports
  &name=Top N Analysis.prpt
  &sLine=[Product].[All Products].[Classic Cars]
  &sMarket=[Markets].[All Markets].[NA]
  &sYear=[Time].[All Years].[2003]
  &TopCount=3
  &output-type=text/html
```

The HTTP POST method is also available via the Reporting URL, so this service can be used in HTML forms in almost the same way as with ViewAction in the earlier examples.

Creating and Displaying Analyzer and Interactive Reporting Content

This subsection contains multiple examples of how to work with Pentaho Analyzer and Interactive Reporting reports through URLs.

Understanding Analyzer Service URLs

What Are Analyzer Service URLs?

This set of examples demonstrates how to integrate Pentaho Analyzer content into your Web application. Analyzer is a plugin to the Pentaho BI Server that serves slice-and-dice and data visualization functionality using a OLAP cube as a data source.

Pentaho Analyzer differs from Pentaho Reporting in that Analyzer has two modes of operation for an Analyzer report: a report can be executed in viewer mode, or editor mode (with options). While the service URLs vary only slightly, the functionality is distinctly different.

When to Use Analyzer Service URLs

These service URLs are valuable when you want to provide interactive OLAP analysis views and capabilities to your users. The editor mode of the service opens Analyzer with full slice-and-dice capabilities from both saved and new reports.

The viewer mode of Analyzer limits the interactivity of the analysis report, and removes all edit controls in favor of a larger data grid. This mode is more suitable for dashboarding and static display of OLAP report data. The viewer does still allow some interactivity through a context menu, such as changing the dimension selections for the report.

How to Use Analyzer URLs

The Analyzer URLs are HTTP URLs, and as such, follow the syntax rules for Web-based location strings. Operational parameters are specified after the query symbol (?) as name/value pairs.

Using Analyzer Parameters in URLs

You can append a static parameter value onto a URL for any Analyzer report that contains query parameters. Ordinarily report users will select the parameter value in the filter dialogue, thus altering the report. However, you can set the filter value by hand by using a URL parameter. First your report must have a parameter defined in it; you cannot create parameters through URLs -- you can only set values for them. Next, you must append the parameter name and a valid value to the URL. So if you have a parameter named **line** that lists product lines for model cars (such as in the Steel Wheels sample data), the URL snippet to define the parameter would look like this:

```
&line=Classic+Cars
```

The full URL, with other name/value pairs defined, would be:

```
http://localhost:8080/pentaho/content/analyzer/editor?command=open&solution=steel-
wheels&path=%2Fanalysis&action=Product+Line+Sales
+Trend.xanalyzer&showFieldList=true&line=Classic+Cars
```

Operational Analyzer Viewer URL Parameters

Parameter	Data Type	Description
solution	String	**Required.** The name of the solution where the Analyzer report file is located.
path	String	**Required.** The relative path from the solution name to where the Analyzer report file is located.
name	String	**Required.** The name of the Analyzer report file to render.

Operational Analyzer Editor Parameters

Parameter	Data Type	Description
solution	String	The name of the solution where the Analyzer report definition file is located. **Required only when command=open**.
path	String	The relative path from the solution name to where the Analyzer report file is located. **Required only when command=open**.
action	String	The name of the Analyzer report file to render. **Required only when command=open**.
command	String	To open an existing report, set to **open**. To start with a new report, set to **new**.
showFieldList	Boolean	Set to **true** to show the panel of dimensions and fields in Analyzer. Set to **false** to hide the panel.
catalog	String	The name of the Analysis catalog that contains the cube for the new Analyzer report. **Required only when command=new**.
cube	String	The name of the cube to use for the new Analyzer report. **Required only when command=new**.

Operational Interactive Reporting URL Parameters

Interactive Reporting has query parameters similar to Analyzer. The base URL (after the context name) for Interactive Reporting is:

```
/content/pentaho-interactive-reporting/resources/web/pir.html
```

To create a new interactive report, you must specify: **command**, **solution**, **path**, **file**.

To open an existing interactive report, you only need to use the **model** parameter.

Parameter	Data Type	Description
solution	String	The name of the solution where the PRPTi definition file is located. This must be used in conjunction with the **command** parameter, where the command value is **view** or **edit**.

Parameter	Data Type	Description
path	String	The relative path from the solution name to where the PRPTi file is located.
file	String	The name of the PRPTi file to load. This must be used in conjunction with the **command** parameter, where the command value is **view** or **edit**.
command	String	The file action. Possible values are: **new**, **view**, and **edit**. Default is **new**.
model	String	The ID of the data source model you want to work with. Specifying a model will bypass the data source dialogue when Interactive Reporting starts. This parameter is only valid when **command=new**. The format for specifying a data model is as follows: **metdata-model-filename:MODEL_ID**.

Displaying an Analyzer Report in Viewer Mode

The viewer URL requires the **solution**, **path**, and **action** parameters in order to locate the Analyzer report to display. This example will display the report described in the parameters with almost all interactive capabilities hidden.

```
http://localhost:8080/pentaho/content/analyzer/viewer?
  solution=steel-wheels
  &path=/analysis
  &action=Top 5 Product Lines by Territory.xanalyzer
```

Displaying an Analyzer Report in Editor Mode

This example will open the same report as in the previous example., but will do so with all slice-and-dice capabilities available. This includes the ability to modify dimension, measure and filter selections, display those selections in play, and choose new selections from the panel of dimensions.

```
http://localhost:8080/pentaho/content/analyzer/editor?
  command=open
  &solution=steel-wheels
  &path=/analysis
  &action=Top 5 Product Lines by Territory.xanalyzer
```

Creating a New Analyzer Report

This example demonstrates starting a new report with Analyzer in editor mode, and specifying the OLAP cube and catalog to start data analysis with. The catalog and cube must be defined in the Pentaho BI Server before requesting to use them in a new report. See the *Pentaho Analysis Guide* for details on setting up Mondrian schemas in the BI Server.

```
http://localhost:8080/pentaho/content/analyzer/editor?
  command=new
  &showFilterList=true
  &catalog=SampleData
  &cube=Quadrant Analysis
```

As with previous content services described, the Analysis service URLs can also be used with the POST method in an HTML form.

Integrating Pentaho Web-Based Client Tools

This subsection explains how to integrate the Pentaho Analyzer and the ad hoc reporting client tools into an existing Web application by using an HTML iframe.

Running Pentaho Analyzer In an iframe

This example embeds the Pentaho Analyzer client tool into a Web page. For demonstration purposes, the page only contains one line of text in a <p> tag, and it displays Analyzer in an iframe that contains a blank HTML page. The appropriate files for this example are shown below in the codeblocks.

Basically the example uses inline JavaScript to create four buttons that call into Analyzer to provide simple file operations -- new, open, save, and save as. Once an action is selected, the browser creates a file dialogue (for open and save as), or the BI Server creates a new .xanalyzer file (for new), or Analyzer updates the existing open file (for save).

analyzer_integration.html

```
<html>
<head>
<!--
   This example demonstrates integrating Pentaho Analyzer into an iframe based
 application.
   Please read the inline documentation for details on how Analyzer is integrated.
-->
  <script type="text/javascript">

    //
    // The first section of JavaScript includes parent iframe callbacks that Pentaho
 content generators may call.
    //

    // mantle_initialized must be set for content generators to behave correctly with
 the parent window.
    var mantle_initialized=true;

    // The enableContentEdit method is called when a content generator is editable or
 not
    function enableContentEdit(contentEdit) {
      alertlog('enableContentEdit called: ' + contentEdit);
    }

    // The setContentEditSelected method is called when the content generator wants to
 toggle the state of the editing.
    function setContentEditSelected(contentEdit) {
      alertlog('setContentEditSelected called: ' + contentEdit);
    }

    // This function is called to enable / disable the "save" and "save as" buttons.
    function enableAdhocSave(adhocSave) {
      alertlog('enableAdhocSave called: ' + adhocSave);
      if (adhocSave) {
        document.getElementById("save").style.display='inline';
        document.getElementById("saveAs").style.display='inline';
      } else {
        document.getElementById("save").style.display='none';
        document.getElementById("saveAs").style.display='none';
      }
    }

    // This function is called during the save process.  In Mantle, this triggers a
 repository refresh.
    function mantle_refreshRepository() {
       alertlog('mantle_refreshRepository called');
    }

    // This function is called if there is an error message during save.
    function mantle_showMessage(title, details) {
      // Keep track of the last message, so we know if we should show a "save success"
 message.
      lastMessage = title + ": " + details;
      alert(lastMessage);
    }

    //
    // Non-API methods defined for this example:
    //

    // analyzerLocation holds the value of the most recent location opened or saved.
    var analyzerLocation;
    // lastMessage keeps track of the last error during the save process.
    var lastMessage;

    // The alertOn and alertlog is used for debugging; set alertOn to see when the
 different callbacks are made.
    var alertOn = false;
    function alertlog(txt) {
      if (alertOn) {
        alert(txt);
      }
    }

    // The newAnalyzerReport function is called when the user clicks "New".
    function newAnalyzerReport() {
      enableAdhocSave(false);
      updateInfo(null);
      document.getElementById("info").innerHTML = '';

      // This is the URL for a new analyzer report. Note that the userid and password
 are included in the url for simplicity; we recommend using a more secure way for
 connecting to the URL.
      window.frames["analyzer"].location = '/pentaho/content/analyzer/selectSchema' + '?
userid=joe&password=password';
    }

    // The openAnalyzerReport function is called when the user clicks "Open".
    function openAnalyzerReport() {
      // Ajax call to solution browser
      // display list
      var sp = getSolutionPath();
      if (sp != null) {

        // This URL opens an Analyzer report at a given location within the repository.
 Note that the userid and password are included in the URL for simplicity; we recommend
 using a more secure way for connecting to the URL in production.
        var url ='/pentaho/content/analyzer/editor?command=open&solution='
 + sp[0] + '&path=' + sp[1] + '&action='+sp[2]+'&showFieldList=true' +
 '&userid=joe&password=password';
        window.frames["analyzer"].location = url;
        updateInfo(sp);
      }
    }

    // The saveAnalyzerReport function is called when the user clicks "Save" or "Save
 As".
    function saveAnalyzerReport(saveas) {
      var sp;
      if (analyzerLocation == null || saveas) {
         sp = getSolutionPath();
      } else {
         sp = analyzerLocation;
      }
      if (sp != null) {
        // Clear out the last message if it gets set; this would mean that there was an
 error in saving.
        lastMessage = null;

        // This call tells Analyzer to save content in a specified location within a
 solution.
        // The first param is the filename
        // The second param is the solution
        // The third param is the path within the solution
        window.frames["analyzer"].gCtrlr.repositoryBrowserController.remoteSave(sp[2],
 sp[0], '/' + sp[1], null, null);
        if (lastMessage == null) {
          alert('Analyzer Report saved.');
          updateInfo(sp);
        }
      }
    }

    // The updateInfo method is called when setting the most recent xanalyzer file.
    function updateInfo(solutionInfo) {
      analyzerLocation = solutionInfo;
      if (solutionInfo == null) {
        document.getElementById("info").innerHTML = '';
      } else {
        document.getElementById("info").innerHTML = 'Path: /' + solutionInfo[0] + '/' +
 solutionInfo[1] + '/' + solutionInfo[2];
      }
    }

    // The getSolutionPath method prompts for an .xanalyzer file.
    function getSolutionPath() {
      var solutionPath = analyzerLocation;
      if (solutionPath == null) {
         solutionPath = ['steel-wheels','analysis','example.xanalyzer'];
      }
      solutionPath[0] = prompt("Step 1 of 3 - Solution:", solutionPath[0]);
      if (solutionPath[0] == null) {
         return null;
      }
      solutionPath[1] = prompt("Step 2 of 3 - Path:", solutionPath[1]);
      if (solutionPath[1] == null) {
         return null;
      }
      solutionPath[2] = prompt("Step 3 of 3 - Filename:", solutionPath[2]);
      if (solutionPath[2] == null) {
         return null;
      }
      return solutionPath;
    }
  </script>
</head>
<body>
<p>My existing Web page is here. I embedded Pentaho Analyzer below!</p>
  <table width="100%" height="100%">
    <tr>
      <td width="100%">
        Analyzer Report Integration Example &nbsp;
        <a href="javascript:newAnalyzerReport()">New</a> &nbsp;
        <a href="javascript:openAnalyzerReport()">Open</a> &nbsp;
        <a id="save" style="display:none"
 href="javascript:saveAnalyzerReport(false)">Save</a> &nbsp;
        <a id="saveAs" style="display:none"
 href="javascript:saveAnalyzerReport(true)">Save As</a> &nbsp;
        <span id="info"></span>
      </td>
    </tr>
    <tr>
      <td width="100%" height="100%">
        <iframe name="analyzer" src="blank.html" width="100%" height="100%">
          <p>Your browser does not support iframes.</p>
        </iframe>
      </td>
    </tr>
  </table>
</body>
</html>
```

blank.html

This is essentially an HTML file with nothing in it. It serves as the target for the iframe for Analyzer. Theoretically you could have content in this file; it would display initially, then be replaced by Analyzer once it loads.

```
<html>
<body>
</body>
</html>
```

Running Interactive Reporting In an iframe

This example embeds Pentaho Interactive Reporting into an existing Web page. For demonstration purposes, the page only contains one line of text in a <p> tag, and it displays the reporting tool in a new iframe. The entire HTML file for this example is shown below in the codeblock.

pir_integration.html

```
<html>
  <head>
  <!--
  This example demonstrates integrating Pentaho Interactive Reporting (PIR) into
  an iframe-based application.
  Please read the inline comments for details on this process.
  -->

    <script type="text/javascript">
      //
      // Non-API methods defined for this example:
      //
      // pirUrlBase contains the location to the Pentaho Interactive Reporting entry
 point
      var pirUrlBase = '/pentaho/content/pentaho-interactive-reporting/resources/web/
pir.html';
      // pirLocation holds the value of the most recent location opened or saved.
      var pirLocation;
      // lastMessage keeps track of the last error during the save process.
      var lastMessage;
      // Current state of edit option. Used to determine if, when clicked, we set edit
 mode to true or false.
      var editMode = false;

      // The alertOn and alertlog are used for debugging; set alertOn to see when the
 different callbacks are made.
      var alertOn = false;
      function alertlog(txt) {
        if (alertOn) {
          alert(txt);
        }
      }

      //
      // API Methods that Interactive Reporting relies on to determine/provide available
 functionality.
      //
      // mantle_initialized must be set for Interactive Reporting to enable certain
 functionality
      var mantle_initialized=true;

      // The enableContentEdit method is called when a content generator is editable or
 not
      function enableContentEdit(contentEdit) {
        alertlog('enableContentEdit called: ' + contentEdit);
        document.getElementById("edit").style.display = (contentEdit ? "inline" :
 "none");
      }

      // The setContentEditSelected method is called when the content generator wants to
 toggle the state of the editing.
```

```
      function setContentEditSelected(contentEdit) {
        alertlog('setContentEditSelected called: ' + contentEdit);
        document.getElementById("edit").innerHTML = (contentEdit ? "Toggle View" :
"Toggle Edit");
      }

      // This function is called to enable / disable the "save" and "save as" buttons.
      function enableAdhocSave(adhocSave) {
        alertlog('enableAdhocSave called: ' + adhocSave);
        if (adhocSave) {
          document.getElementById("save").style.display='inline';
          document.getElementById("saveAs").style.display='inline';
        } else {
          document.getElementById("save").style.display='none';
          document.getElementById("saveAs").style.display='none';
        }
      }

      // This function is called during the save process. triggers a repository refresh.
      function mantle_refreshRepository() {
        alertlog('mantle_refreshRepository called');
      }

      //
      // Local functions for interacting with Pentaho Interactive Reporting
      //
      // Returns the iFrame PIR is loaded in
      function getPIRiFrame() {
        return window.frames[0];
      }

      // Create a new Interactive Report. Called when the user clicks 'New'
      function newPIRReport() {
        alertlog("Loading new PIR Report");
        getPIRiFrame().location = pirUrlBase;
        editMode = true;
      }

      // Open an existing Interactive Report. Called when the user clicks "Open".
      function openPIRReport() {
        // Ajax call to solution browser
        // display list
        var sp = getSolutionPath();
        if (sp != null) {
          // This URL opens a PIR report at a given location within the repository.
          // command=view is required to view the document at the provided path.
          var url = pirUrlBase + '?command=view&solution=' + escape(sp[0]) + '&path=' +
escape(sp[1]) + '&file=' + escape(sp[2]);
          getPIRiFrame().location = url;
          editMode = false;
          updateInfo(sp);
        }
      }

      // Enables and disabled
      function editPIRReport() {
        // Toggle edit mode and
        editMode = !editMode;
        getPIRiFrame().window.editContentToggled(editMode);
      }

      // The saveWAQRReport function is called when the user clicks "Save" or "Save As".
      function savePIRReport(saveas) {
        var sp;
        if (pirLocation == null || saveas) {
          sp = getSolutionPath();
        } else {
          sp = pirLocation;
        }
        if (sp != null) {
          // This call tells PIR to save content in a specified location within a
solution.
          // The first param is the filename
          // The second param is the solution
          // The third param is the file name within the solution
          // The last parameter is the overwrite flag. It determines if an existing file
by the same name
          // will be overwritten or not. If set to false and the file exists the user
will be prompted that the save was unsuccessful.
          getPIRiFrame().gCtrlr.repositoryBrowserController.remoteSave(sp[2], sp[0], '/'
+ sp[1], null, true);
          updateInfo(sp);
        }
      }

      // The updateInfo method is called when setting the most recent PIR file.
      function updateInfo(solutionInfo) {
        pirLocation = solutionInfo;
        if (solutionInfo == null) {
          document.getElementById("info").innerHTML = '';
        } else {
          document.getElementById("info").innerHTML = 'Path: /' + solutionInfo[0] + '/'
+ solutionInfo[1] + '/' + solutionInfo[2];
        }
      }

      // The getSolutionPath method prompts for a .prpti file.
      function getSolutionPath() {
        var solutionPath = pirLocation;
        if (solutionPath == null) {
          solutionPath = ['steel-wheels','reports','Vendor Sales Report.prpti'];
        }
        solutionPath[0] = prompt("Step 1 of 3 - Solution:", solutionPath[0]);
        if (solutionPath[0] == null) {
          return null;
        }
        solutionPath[1] = prompt("Step 2 of 3 - Path:", solutionPath[1]);
        if (solutionPath[1] == null) {
          return null;
        }
        solutionPath[2] = prompt("Step 3 of 3 - Filename:", solutionPath[2]);
        if (solutionPath[2] == null) {
          return null;
        }
        return solutionPath;
      }
    </script>
  </head>
  <body>
    <p>My existing Web page is here. I embedded Pentaho Interactive Reporting (PIR)
below!</p>
    <table width="100%" height="100%">
      <tr>
        <td width="100%">
          <span>PIR Integration Example: &nbsp;</span>
          <a href="javascript:newPIRReport()">New</a> &nbsp;
          <a href="javascript:openPIRReport()">Open</a> &nbsp;
          <a id="edit" href="javascript:editPIRReport()" style="display:none">Edit</a>
&nbsp;
          <a id="save" style="display:none"
href="javascript:savePIRReport(false)">Save</a> &nbsp;
          <a id="saveAs" style="display:none" href="javascript:savePIRReport(true)">Save
As</a> &nbsp;
          <span id="info"></span>
        </td>
      </tr>
      <tr>
        <td width="100%" height="100%">
```

```
          <iframe name="pir" src="about:blank" width="100%" height="100%">
            <p>Your browser does not support iframes. This example will not work.</p>
          </iframe>
        </td>
      </tr>
    </table>
  </body>
</html>
```

Running Ad Hoc Reporting In an iframe

Note: Interactive Reporting is a much more powerful online reporting tool. If you have a Pentaho Reporting Enterprise Edition license, you may want to look into embedding IR instead of ad hoc reporting.

This example embeds the Pentaho User Console's ad hoc reporting interface into a Web page. For demonstration purposes, the page only contains one line of text in a <p> tag, and it displays the reporting tool in an iframe that contains a blank HTML page. The appropriate files for this example are shown below in the codeblocks.

Basically the example uses inline JavaScript to create four buttons that call into ad hoc reporting to provide simple file operations -- new, open, save, and save as. Once an action is selected, the browser creates a file dialogue (for open and save as), or the BI Server creates a new .waqr.xaction file (for new), or ad hoc reporting updates the existing open file (for save).

waqr_integration.html

```
<html>
<head>
<!--
   This example demonstrates integrating Pentaho ad hoc reporting (WAQR) into an iframe-
based application.
   Please read the inline documentation for details on how WAQR is integrated.
-->
  <script type="text/javascript">

    //
    // The first section of Javascript includes parent iframe callbacks that Pentaho
 content generators may call.
    //

    // mantle_initialized must be set for content generators to behave correctly with
 the parent window.
    var mantle_initialized=false;

    // The enableContentEdit method is called when a content generator is editable or
 not
    function enableContentEdit(contentEdit) {
      alertlog('enableContentEdit called: ' + contentEdit);
    }

    // The setContentEditSelected method is called when the content generator wants to
 toggle the state of the editing.
    function setContentEditSelected(contentEdit) {
      alertlog('setContentEditSelected called: ' + contentEdit);
    }

    // This function is called to enable / disable the "save" and "save as" buttons.
    function enableAdhocSave(adhocSave) {
      alertlog('enableAdhocSave called: ' + adhocSave);
      if (adhocSave) {
        document.getElementById("save").style.display='inline';
        document.getElementById("saveAs").style.display='inline';
      } else {
        document.getElementById("save").style.display='none';
        document.getElementById("saveAs").style.display='none';
      }
    }
    // This function is called during the save process.  In Mantle, this triggers a
 repository refresh.
    function mantle_refreshRepository() {
       alertlog('mantle_refreshRepository called');
    }

    // This function is called if there is an error message during save.
    function mantle_showMessage(title, details) {
      // Keep track of the last message, so we know if we should show a "save success"
 message.
      lastMessage = title + ": " + details;
      alert(lastMessage);
    }

    //
    // Non-API methods defined for this example:
    //

    // waqrLocation holds the value of the most recent location opened or saved.
    var waqrLocation;
    // lastMessage keeps track of the last error during the save process.
    var lastMessage;

    // The alertOn and alertlog are used for debugging; set alertOn to see when the
 different callbacks are made.
    var alertOn = false;
    function alertlog(txt) {
      if (alertOn) {
        alert(txt);
      }
    }

    // The newWAQRReport function is called when the user clicks "New".
    function newWAQRReport() {
      enableAdhocSave(false);
      updateInfo(null);
      document.getElementById("info").innerHTML = '';

      // This is the URL for a new waqr report. Note that the userid and password are
 included in the url for simplicity; we recommend using a more secure way for connecting
 to the URL.
      window.frames["waqr"].location = '/pentaho/adhoc/waqr.html' + '?
userid=joe&password=password';
    }

    // The openWAQRReport function is called when the user clicks "Open".
    function openWAQRReport() {
      // Ajax call to solution browser
      // display list
      var sp = getSolutionPath();
      if (sp != null) {

        // This URL opens an WAQR report at a given location within the repository. Note
 that the userid and password are included in the URL for simplicity; we recommend using
 a more secure way for connecting to the URL in production.
        var url ='/pentaho/ViewAction?solution=' + sp[0] + '&path=' + sp[1] +
 '&action='+sp[2]+ '&userid=joe&password=password';
        window.frames["waqr"].location = url;
        updateInfo(sp);
      }
    }

    // The saveWAQRReport function is called when the user clicks "Save" or "Save As".
    function saveWAQRReport(saveas) {
      var sp;
      if (waqrLocation == null || saveas) {
         sp = getSolutionPath();
      } else {
         sp = waqrLocation;
      }
      if (sp != null) {
        // Clear out the last message if it gets set; this would mean that there was an
 error in saving.
        lastMessage = null;

        // This call tells WAQR to save content in a specified location within a
 solution.
        // The first param is the filename
        // The second param is the solution
        // The third param is the path within the solution
        window.frames["waqr"].gCtrlr.repositoryBrowserController.remoteSave(sp[2],
 sp[0], '/' + sp[1], null, null);
        if (lastMessage == null) {
          alert('WAQR Report saved.');
          updateInfo(sp);
        }
      }
    }

    // The updateInfo method is called when setting the most recent WAQR xaction file.
    function updateInfo(solutionInfo) {
      waqrLocation = solutionInfo;
      if (solutionInfo == null) {
        document.getElementById("info").innerHTML = '';
      } else {
        document.getElementById("info").innerHTML = 'Path: /' + solutionInfo[0] + '/' +
 solutionInfo[1] + '/' + solutionInfo[2];
      }
    }

    // The getSolutionPath method prompts for a .waqr.xaction file.
    function getSolutionPath() {
      var solutionPath = waqrLocation;
      if (solutionPath == null) {
         solutionPath = ['WAQR','','sample.waqr.xaction'];
      }
      solutionPath[0] = prompt("Step 1 of 3 - Solution:", solutionPath[0]);
      if (solutionPath[0] == null) {
         return null;
      }
      solutionPath[1] = prompt("Step 2 of 3 - Path:", solutionPath[1]);
      if (solutionPath[1] == null) {
         return null;
      }
      solutionPath[2] = prompt("Step 3 of 3 - Filename:", solutionPath[2]);
      if (solutionPath[2] == null) {
         return null;
      }
      return solutionPath;
    }
  </script>
</head>
<body>
<p>My existing Web page is here. I embedded Pentaho ad hoc reporting (WAQR) below!</p>
  <table width="100%" height="100%">
    <tr>
      <td width="100%">
        WAQR Report Integration Example &nbsp;
        <a href="javascript:newWAQRReport()">New</a> &nbsp;
        <a href="javascript:openWAQRReport()">Open</a> &nbsp;
        <a id="save" style="display:none" href="javascript:saveWAQRReport(false)">Save</
a> &nbsp;
        <a id="saveAs" style="display:none" href="javascript:saveWAQRReport(true)">Save
 As</a> &nbsp;
        <span id="info"></span>
      </td>
    </tr>
    <tr>
```

```
      <td width="100%" height="100%">
        <iframe name="waqr" src="blank.html" width="100%" height="100%">
          <p>Your browser does not support iframes.</p>
        </iframe>
      </td>
    </tr>
  </table>
</body>
</html>
```

blank.html

This is essentially an HTML file with nothing in it. It serves as the target for the iframe for the ad hoc reporting interface. Theoretically you could have content in this file; it would display initially, then be replaced by ad hoc reporting once it loads.

```
<html>
<body>
</body>
</html>
```

Listing Content with the SolutionRepositoryService

This subsection explains how to use custom URLs to manipulate files in the Pentaho solution repository.

Understanding the SolutionRepositoryService

What is the SolutionRepositoryService?

The SolutionRepositoryService is a servlet included in the Pentaho BI Server that allows users with the appropriate permissions to request solution repository service methods, then receive an XML response with the requested data or status of the request.

In order to receive any information about a solution file, the requesting user must have either administrator privileges or the appropriate permissions for the method requested on the requested file/folder.

Service methods available through the SolutionRepositoryService are:

- **getSolutionRepositoryDoc**: this request returns a list of all accessible files and folders in the solution repository along with file/folder metadata attributes. The list can be filtered by type (file extension).
- **getSolutionRepositoryFileDetails**: given a path, this request returns detailed metadata for this file or folder.
- **createNewFolder**: creates a new folder in the solution repository in the requested location with the requested name and description.
- **delete**: deletes the folder or file at the requested location. If the folder has children folders or files, they will also be deleted.
- **setAcl**: sets the specified access control list (ACL) on the requested solution file or folder.
- **getAcl**: gets the ACL for the requested solution file or folder.

When to Use SolutionRepositoryService

The SolutionRepositoryService is useful in a number of scenarios. Perhaps you would like to programmatically create a standard folder structure for new users on first login. Or possibly delete all content in a particular folder when a user is no longer valid in the system.

The SolutionRepositoryService’s ability to list the repository contents in XML format also provides easy access to the data needed to build a custom solution user interface using XSLT or some other Web technology that works directly with XML.

How to Use SolutionRepositoryService

The base URL for the SolutionRepositoryService looks like this:

```
http://localhost:8080/pentaho/SolutionRepositoryService?
```

After the query symbol (?), the first parameter you should specify is **component**. This is the parameter that facilitates requesting the service method you are interested in:

```
http://localhost:8080/pentaho/SolutionRepositoryService?
component=getSolutionRepositoryDoc
http://localhost:8080/pentaho/SolutionRepositoryService?
component=getSolutionRepositoryFileDetails
http://localhost:8080/pentaho/SolutionRepositoryService?component=createNewFolder
http://localhost:8080/pentaho/SolutionRepositoryService?component=delete
http://localhost:8080/pentaho/SolutionRepositoryService?component=setAcl
http://localhost:8080/pentaho/SolutionRepositoryService?component=getAcl
```

Each of these service methods has a set of required parameters. The table below lists the required and optional parameters for each value available to the component parameter.

SolutionRepositoryService Component Parameters

*

Parameter	Data Type	Description
ajax**	String	Available and optional to all service methods; wraps response in SOAP.

getSolutionRepositoryDoc

Parameter	Data Type	Description
filter	String	Filter the list by extension type (*.xaction, *.prpt). The filter is an "ends-with" comparison, and will always return URL and xaction files regardless of what you choose. The value of filter should not include dots or stars -- it is only the file extension after the dot. For instance, you would pass **&filter=prpt** to get a listing of all Pentaho report definition files.

getSolutionRepositoryFileDetails

Parameter	Data Type	Description
fullPath	String	**Required.** The full solution path to the file requested.

createNewFolder

Parameter	Data Type	Description
solution	String	**Required.** The solution to create the folder in.
path	String	**Required.** The path to the location for the folder (must already exist).
filename	String	**Required.** Name to give the new folder.
desc	String	**Required.** Description for the new folder.

delete

Parameter	Data Type	Description
solution	String	**Required.** The solution where folder/file is located.
path	String	**Required.** The path to the folder/file.
filename	String	**Required.** The name of the folder/file.

setAcl

Parameter	Data Type	Description
solution	String	**Required.** The solution where folder/ file is located.
path	String	**Required.** The path to the folder/file.
filename	String	**Required.** The name of the folder/file.
aclXml	String	XML representing the ACL you want to set.

getAcl

Parameter	Data Type	Description
solution	String	**Required.** The solution where folder/ file is located.
path	String	**Required.** The path to the folder/file.
filename	String	**Required.** The name of the folder/file.

Retrieving a List of Solution Files

This example retrieves a list of all Pentaho report definition files in the solution repository.

```
http://localhost:8080/pentaho/SolutionRepositoryService?
  component=getSolutionRepositoryDoc
  &filter=prpt
```

Change the **prpt** to another file extension to filter the list differently, or remove the **filter** parameter to retrieve an unsorted list of files in the solution repository.

Getting Details for a Solution File

This example returns detailed metadata for the specified report definition. This metadata includes the service URLs to execute the content, if applicable.

```
http://localhost:8080/pentaho/SolutionRepositoryService?
  component=getSolutionRepositoryFileDetails
  &fullPath=/ steel-wheels/reports/Sales Summary.prpt
```

Creating a Solution Folder

This example creates a new subdirectory in the solution repository. In order to create a solution folder, the user must have create permissions on the parent directory, which of course must already exist. If all goes well, the return response will be a status of **true**.

```
http://localhost:8080/pentaho/SolutionRepositoryService?
  component=createNewFolder
  &solution=steel-wheels
  &path=reports
  &name=newFolder
  &desc=My New Folder
```

Note: This example is required in order for the next example, *Deleting a File* on page 31 to work. If you change any of the details of this example, you must also change the next example accordingly.

Deleting a File

This example deletes the **/reports/newFolder/** directory in the **steel-wheels** sample solution, including any content that resides there. This directory should exist if you followed the preceding example.

Danger: This service will permanently delete the specified files and folders from your sample solution folder. The delete service method will delete all content and subdirectories of the requested folder. Be careful changing the details of this example.

```
http://localhost:8080/pentaho/SolutionRepositoryService?
  component=delete
  &solution=steel-wheels
  &path=reports
  &name=newFolder
```

Retrieving Solution File and Directory Permission Settings

This example demonstrates retrieving the access control list for a solution file or folder. This is a good precursor to setting an ACL because this method will return the ACL data in the requisite XML format.

```
http://localhost:8080/pentaho/SolutionRepositoryService?
  component=getAcl
  &solution=steel-wheels
  &path=
  &filename=reports
```

Setting Permissions On a Solution File

This example changes role permissions on the **reports** directory to **all** for the specified roles.

Permissions can be set on a file or at the folder level. Any content under a folder that does not have explicit permissions set on it inherits the permissions of the parent folder. Believe it or not, the following URL is valid. If you are wary, you can always encode the ACL XML value.

```
http://localhost:8080/pentaho/SolutionRepositoryService?
  component=setAcl
  &solution=steel-wheels
  &path=
  &filename=reports
  &aclXml= <?xml version='1.0' encoding='UTF-8'?><acl><entry role='Admin'
 permissions='-1'/><entry role='cto' permissions='-1'/><entry role='dev'
 permissions='3'/><entry role='Authenticated' permissions='1'/></acl>
```

Solution ACL Permission Values on page 32 explains what the numeric permissions values mean.

Solution ACL Permission Values

The below list of permissions applies to solution file and directory ACLs. This is particularly useful when you want to change permissions on something in the solution repository, as explained in *Setting Permissions On a Solution File* on page 32. Default ACLs are established in the `/pentaho-solutions/system/pentaho.xml` file, and can be re-applied through the Pentaho Enterprise Console. See the *Pentaho BI Platform Security Guide* for more details.

Permission	Numeric Value	Description
Update	8	Enables the specified user or role to update this file.
Create	4	Enables the specified user or role to create a new file or directory.
Execute	1	Enables the specified user or role to access and execute this action sequence.
All	-1	Enables all permissions in this list except **All** and **NONE** for the specified user or role.
Delete	16	Enables the specified user or role to remove this file or directory.
NONE	0	Removes all permissions in this list except **NONE**.
Subscribe	2	Enables the specified user or role to include this file in a schedule.

Using XML Services With ServiceAction

This subsection explains how to use ServiceAction to access Pentaho's XML services.

Understanding ServiceAction

What is ServiceAction?

ServiceAction is a servlet included in the Pentaho BI Server that provides service methods for executing action sequences, and for retrieving security model details. The output from service methods in ServiceAction is formatted as a SOAP response.

When to Use ServiceAction

ServiceAction can be used in many of the same instances that you would use ViewAction, the primary difference being the format of the output. You can only use action sequences that return content with text MIME types ('text/xml', 'text/html', etc.) with ServiceAction.

ServiceAction also serves a unique set of methods that provide security model details. These methods allow you to query the server for the list of users, roles, and/or ACLs for a given resource known to the BI Server. This service requires that you have administrator access to the BI Server in order to execute any security-related requests.

How to Use ServiceAction

ServiceAction is executed as an HTTP URL, and as such, follows the syntax rules for Web-based location strings. Both operational parameters and action sequence input parameters are specified after the query symbol (?) as name/value pairs.

Operational ServiceAction Parameters for Action Sequences

Parameter	Data Type	Description
solution	String	**Required.** The name of the solution where the action sequence file is located.
path	String	**Required.** The relative path from the solution name to where the action sequence file is located.
action	String	**Required.** The name of the action sequence file to render.

Operational ServiceAction Parameters for Security Details

Parameter	Data Type	Description
action	String	**Required.** For this purpose, set the value to **securitydetails**.
details	String	**Required.** Set to **users** to return only users; set to **roles** to only return roles; set to **acls** to only return permission values; set to **all** to return all users, roles and access control permissions. Default is **all**.

Executing an Action Sequence With a SOAP Response

This example shows how to use ServiceAction to execute an action sequence that returns a list of position titles from a database. The action sequence's return MIME type is **text/xml**, so the output is written to the SOAP message returned from ServiceAction.

```
http://localhost:8080/pentaho/ServiceAction?
  solution=bi-developers
  &path=rules
  &action=CurrentPositionTitles.xaction
```

Retrieving the Security Model

The following example will return all users, roles, and ACLs known to the Pentaho BI Server, wrapped in a SOAP message.

```
http://localhost:8080/pentaho/ServiceAction?
  action=securitydetails
  &details=all
```

Using the Ajax API For Asynchronous Execution

This subsection explains how to use JavaScript to execute an action sequence.

Understanding the Ajax API

What is the Ajax API?

The Pentaho Ajax API is a JavaScript library that enables you to make asynchronous action sequence execution requests to the Pentaho BI Server. The core library file, **pentaho-ajax.js**, holds two methods of interest: **pentahoAction()** and **pentahoService()**. These methods expose two services explained in previous sections of this guide: *ViewAction* and *ServiceAction*.

When to Use the Ajax API

Typically you would use this interface when you want to integrate with another Ajax-based application.

The Community Dashboard Framework, on which the Pentaho Dashboard Designer is built, uses the Pentaho Ajax API at its core.

How to Use the Ajax API

Note: The Pentaho Ajax API must be used within the context of the Pentaho Web application. The API is not directly reachable outside of the Pentaho WAR.

As mentioned above, the Pentaho Ajax API is basically two JavaScript methods: pentahoAction() and pentahoService(), the parameters for which are described in the tables below. The result of these methods is usually the result of the action sequence being executed. You can use these methods anyplace that you can use JavaScript, within the Pentaho context. Depending on how you structure the JavaScript to call these methods, they can be synchronous or asynchronous. If you provide a callback function as a parameter to the API method, then that method will be called asynchronously. If there is no callback provided to the method, then the call is made synchronously and the result is returned as the return value of the API method.

pentahoAction Method Parameters

The pentahoAction() method will return content in the format defined by the action sequence. For more details, refer to *Using ViewAction to Retrieve Content* on page 12.

function `pentahoAction` (*solution, path, action, params, func*)

Parameter	Data Type	Description
solution	String	**Required.** The name of the solution where the action sequence file is located.
path	String	**Required.** The relative path from the solution name to where the action sequence file is located.
action	String	**Required.** The name of the action sequence file to render.
params	Array of arrays	Specifies the name/value pairs of all parameters necessary for the action sequence.

Parameter	Data Type	Description
func	String or function	Refers to the function to call asynchronously when the client receives the server's response. If the parameter is **null** or undefined, the request to the server is synchronous, and the response is returned by this method. If the parameter is of type **String**, it is the name of the function to call. If the parameter is of type **function**, it is the function object to call.

pentahoService Method Parameters

The pentahoService() method will return all results wrapped in a SOAP response.

function `pentahoService` (*component, params, func, mimeType*)

Parameter	Data Type	Description
component	N/A	**Deprecated.** Set to **NULL**.
mimeType	String	Specifies the MIME type of the response.
params	Array of arrays	Specifies the name/value pairs of all parameters necessary for the action sequence.
func	String or function	Refers to the function to call asynchronously when the client receives the server's response. If the parameter is **null** or undefined, the request to the server is synchronous, and the response is returned by this method. If the parameter is of type **String**, it is the name of the function to call. If the parameter is of type **function**, it is the function object to call.

Executing an Action Sequence With JavaScript

This example uses pentahoAction() to asynchronously execute the Sales_by_Supplier.xaction action sequence included in the Pentaho sample data solution.

```
<script language="javascript" src="/pentaho/js/pentaho-ajax.js"></script>
<script>
function callActionSequence() {

pentahoAction( "steel-wheels", "dashboards/Widget Library/Report Snippets",
      "Sales_by_Supplier.xaction",
        new Array(
new Array( "outputType", "html"),
new Array( "Region", "NA")
        ),
       'displaycontent'
);

}

function displaycontent(content) {

document.writeln(content)
document.close();
}
</script>
```

Tutorials in PHP, .NET, HTML, and JSP

The examples in this section are more in-depth than the previous ones; they cover the process of creating Pentaho content specifically for inclusion in existing Web applications, then integrating it. There are examples for the following content types:

- Reports (from Report Designer)
- Analyzer reports
- Interactive reports
- Dashboards (from Dashboard Designer)
- Action sequences
- Charts

Creating a Simple iFrame-Based Dashboard

```
  Each language will have its own paradigms for handling the incoming data from these
 examples.

iframe to display something in an application page (see )
  MT example: http://sandbox.pentaho.com/dojo.htm
```

Result Set Streaming With Pentaho Web Services

```
  Using content generators as a data service:
  Result Set Streaming Pentaho Web Services
Data returned example:  http://sandbox.pentaho.com/php_samples/json_data_parse.php
```

Using the Pentaho Solution Repository Web Service to Retrieve a List of Content

```
  Each language will have its own paradigms for handling the incoming data from these
 examples.

  (see MT for the PHP example)

  Colosa ProcessMaker -- Pentaho customer who has integrated Pentaho with PHP: http://
www.processmaker.com/
```

Other Embedding Scenarios

The Pentaho BI Server is comprised of many integratable, embeddable, and extendable components, from libraries to entire applications. This guide only covers integrating extant BI Server client tool functionality into Web applications. This may be too much or too little for what you need to do. The sections below explain other ways to integrate or embed Pentaho BI Server functionality into a new or existing application.

Embedding the Core BI Platform

The heart of the BI Server is the Pentaho BI Platform. It's a highly extensible, embeddable, and scriptable workflow engine designed for but not exclusive to running business intelligence tasks. It exists not as a single program, but as a collection of open source Java classes and specifications that enable programmers to complete such tasks as: retrieving data from multiple disparate data sources, creating data-driven reports and other content, and scheduling and conditionally automating content delivery.

The various BI Platform components can be assembled to create or add to an application with as little or as much functionality as desired. You can start with a simple application, then add in services or components that provide security; auditing; and actions like ETL transformations and OLAP database queries. The closer you get to requiring a Web application server, the more sense it makes to embed larger cohesive portions of the Pentaho BI Server (or the entire BI Server itself), rather than individual BI Platform libraries. Using 100% of the Platform's capabilities would recreate the Pentaho BI Server.

You can also extend the Platform with your own components and services. Embedding the BI Platform enables you to develop applications that leverage its core functionality. However, rather than reinventing the wheel by creating your own client tools based primarily on Pentaho's libraries, you may find it more convenient to embed the ones included in the BI Server: the Pentaho User Console, the ad hoc reporting interface, Dashboard Designer, JPivot, Chart Editor, and Analyzer.

There is currently no comprehensive documentation on this subject. If you are a current or potential customer interested in embedding large cohesive parts of the BI Server, contact the Pentaho sales department and inquire about developer support.

Embedding the Reporting Engine

If all you really need is the ability to run reports -- not scheduling or delivery -- you should consider embedding only the Pentaho Reporting engine instead of the much larger and more comprehensive BI Server.

Pentaho provides a document titled *Embedding the Pentaho Reporting Engine* that is similar to this one, except focused solely on embedding the Reporting engine libraries. You can obtain this document from the Pentaho Knowledge Base, or by downloading the Pentaho Reporting SDK: *http://sourceforge.net/projects/jfreereport/files/03.%20SDK/*.

Embedding the Analysis Engine

If you only want to display data derived from a Pentaho Analysis schema and you don't intend to use any of Pentaho's client tools (JPivot, Pentaho Analyzer) or create reports that use MDX queries as data sources, it makes more sense to embed the Pentaho Analysis (also known as Mondrian) engine instead of the BI Platform.

There is currently no comprehensive documentation on this subject. If you are a current or potential customer interested in embedding the Analysis engine, contact the Pentaho sales department and inquire about developer support.

Developer Support

The examples in this guide are simple and easy to follow, but with more complex requirements come more advanced programs. While reading the source code comments can help quite a bit, you may still need help to develop an application within a reasonable timeframe. Should you need personal assistance, you can have direct access to the most knowledgeable support resources through a Pentaho Enterprise Edition software vendor annual subscription:

ISV/OEM support options

If phone and email support are not enough, Pentaho can also arrange for an on-site consulting engagement:

Consultative support options

License Information

Most of the software comprising the Pentaho BI Suite is open source, licensed under the GNU General Public License version 2. The BI Suite also contains a large volume of third-party open source libraries that are licensed under a number of different licenses. Most of this software is freely redistributable, with the notable exceptions of the following Pentaho-authored programs:

- Dashboard Designer
- Analyzer
- Interactive Reporting
- Various individual BI Platform JARs

If you already have regular Pentaho licenses for the BI Platform, Dashboard Designer, Interactive Reporting, and Analyzer, then no further licenses are required to integrate BI Server functionality into a third-party application. If you wish to embed pieces of the BI Server into an application that you intend to sell or distribute, you must familiarize yourself with the licenses of all of the pieces you are including in order to make sure you are complying properly. Proprietary Pentaho software may not be redistributed under any circumstances.

This guide is not intended for redistribution. However, the example code and example application that accompany this document may be freely modified or reused.

Obtaining the Source Code

The open source portions of the BI Suite can be freely downloaded from SourceForge under the following projects:

- *Pentaho Business Intelligence*
- *JFreeReport (Pentaho Reporting)*
- *Mondrian*

You can also access Pentaho's Subversion source code repository at: *svn://source.pentaho.org/svnroot/*.