Jimmy Labs Community Edition
Release 1

November 2010

Introduction

The Jimmy Labs bundle is a collection of prototype features I have worked on.

This release contains:

- **Metadata Services**: Another iteration of metadata services. A prototype to show how to use metadata models on the client side to allow users to query data.
- **Repository Service**: Services for thin clients to allow AJAX widgets/visualizations to save/load their state in the repository.
- **Samples**: Samples to show
    - Client-side integration with the User Console
    - Client side metadata model usage
    - Ad-hoc report sample using the metadata models

This release is designed to work with V3.6 of the BI Server. All these features are delivered with plugins.

I am providing these features to gather feedback on their usefulness, completeness, design, usability etc. I have only made brief efforts to knock off the ugliness that typically affects prototype features.

Support and Other Gnarly Topics

I don't believe that using any of these features will result in catastrophic system failure or data loss. Backing-up is a good idea. Don't come crying to me if you didn't.

Do not call Pentaho support with any questions or issues about these features.

These features are not localized.

Installation and Configuration

1. Obtain a copy of pentaho-jimmylabs-CE-R1.zip

2. Unzip into pentaho-solutions/system within a Pentaho EE BI Server V3.6 installation. You should now have thse folders pentaho-solutions/system:

- jimmylabs-r1-metadata
- jimmylabs-r1-samples

Metadata Services

This web service and the client-side Javascript objects are a new iteration of the thin metadata models. The web service address is :

http://localhost:8080/pentaho/content/ws-run/MetadataService

The Metadata Models Sample and the Metadata Report Sample provide examples of using the client-side objects to access the web service.

The client-side javascript objects assist in creating and submitting queries. These objects are defined in jimmylabs-r1-metadata/resources/web/metadata/models.js and are described below.

The service provides the following functions.

listBusinessModels

No parameters. This lists the metadata models available to the current user.

Response

The service returns the domain id, the model id, and the model name and description for each available model.

```
<ns:listBusinessModelsResponse xmlns:ns="http://
metadata.jimmylabs.pentaho.com" xmlns:ax2393="http://
beans.model.metadata.jimmylabs.pentaho.com/xsd" xmlns:ax2394="http://
model.metadata.jimmylabs.pentaho.com/xsd" xmlns:ax2397="http://io.java/xsd"
xmlns:ax2400="http://logging.commons.apache.org/xsd" xmlns:ax2390="http://
connection.commons.pentaho.org/xsd" xmlns:ax2389="http://
data.metadata.jimmylabs.pentaho.com/xsd">
      <return type="com.pentaho.jimmylabs.metadata.model.beans.ModelInfo">
            <domainId>steel-wheels/metadata.xmi</domainId>
            <modelDescription>This model contains information about
Employees.</modelDescription>
            <modelId>BV_HUMAN_RESOURCES</modelId>
            <modelName>Human Resources</modelName>
      </return>
      etc
</ns:listBusinessModelsResponse>
```

loadModel

This returns the definition of a model.

Parameters

- domainId: e.g. steel-wheels/metadata.xmi
- modelId: e.g. BV_HUMAN_RESOURCES

The values for the parameters can be taken from the domainId and modelId elements returned in the listBusinessModels response.

doQuery

This executes an MQL query and returns the results.

Parameters

query: An MQL XML string

PentahoMetadataClient Object

The main metadata client object. The functions are described below.

PentahoMetadataClient.getModelInfoList

No parameters. Returns an array of PentahoModelInfo objects for the models available.

PentahoMetadataClient.loadModelInfoList

No parameters. Loads PentahoModelInfo objects from the server and returns a count of the number of PentahoModelInfo objects loaded.

PentahoMetadataClient.getModel

Returns a PentahoModel object representing the requested model. Uses a model from client-side cache unless forceLoad is specified.

Parameters

- domainId: e.g. steel-wheels/metadata.xmi
- modelId: e.g. BV_HUMAN_RESOURCES
- forceLoad: Specifies whether to force a reload of the model from the server

PentahoMetadataClient.loadModel

Loads a model from the server and returns a PentahoModel object.

Parameters

- domainId: e.g. steel-wheels/metadata.xmi
- modelId: e.g. BV_HUMAN_RESOURCES

PentahoModel Object

This object represents a metadata model. The fields and functions are described below

Fields

- domainId: e.g. steel-wheels/metadata.xmi
- modelId: e.g. BV_HUMAN_RESOURCES
- modelName: the name of the model
- modelDescription: the descriptions for the model
- categories: an array of Category objects

PentahoModel.getCategories

No parameters. Returns an array of Categories for the model.

PentahoModel.getAllColumns

No parameters. Returns an array of all the Column objects in the model. The array is not sorted.

PentahoModel.getColumnById

Returns a column object having the specified id.

Parameters

- columnId:

PentahoModel.sortColumnsByName

Sorts a provided array of columns by the column names.

Parameters
- columns: an array of Column objects

PentahoModel.getColumnsByFieldType

Returns an array of Column objects having the specified field type.

Parameters
- fieldType: the field type to filter by. One of Column.FIELD_TYPES.DIMENSION, Column.FIELD_TYPES.FACT, Column.FIELD_TYPES.ATTRIBUTE, Column.FIELD_TYPES.KEY

PentahoModel.getColumnsByFieldTypes

Returns an array of Column objects having any of the specified field types.

Parameters
- fieldTypes: an array of field types to filter by. Any of Column.FIELD_TYPES.DIMENSION, Column.FIELD_TYPES.FACT, Column.FIELD_TYPES.ATTRIBUTE, Column.FIELD_TYPES.KEY

PentahoModel.getColumnFromList

Returns a Column object given an HTML list object with a selected option. This function assumes that the values of the options in the list contain column ids.

Parameters
- list: HTML list object

PentahoModel.createListOptions

Creates an array of HTML Option objects for the provided array of columns. The column names are used for the displayed text and the column ids are used for the option values.

Parameters

- columns: an array of Column

PentahoModel.populateListControl

Populates an HTML list object with options representing columns of the specified field types. Optionally the list can be sorted and a selected column can be specified.

Parameters
- list: HTML list object to populate
- fieldTypes: an array of field types to filter the column list by. Any of Column.FIELD_TYPES.DIMENSION, Column.FIELD_TYPES.FACT, Column.FIELD_TYPES.ATTRIBUTE, Column.FIELD_TYPES.KEY
- sorted: whether to sort the list options by the column names
- selectedItem: the id of a column to select.

PentahoModel.searchColumn

Searches the values of a specified column and returns an array of the values that match the specified search string. The results are sorted into ascending order. If no search string is not provided, all the values for the column are returned. This function creates an MQL query and calls the metadata service to get the results.

Parameters

- column: the column to search
- searchStr: the string to search for

PentahoModel.getAllValuesForColumn

Returns all of the values for a specified column.The results are sorted into ascending order. This function creates an MQL query and calls the metadata service to get the results.

Parameters

- column: the column to search

PentahoModel.createQuery

No parameters. Returns a new MetadataQuery object.

PentahoModel.submitQuery

Submits an MQL query to the server. Returns an object with these fields:

- columnNames: an array of the column names
- columnTypes: an array of the column types
- rows: a 2d array of the query results

Parameters

- queryObject: the MetadataQuery to submit

Category Object

Represents the categories in a metadata model. Contains these fields:

- categoryId
- categoryName
- columns: an array of Column objects within the category

Column Object

Represents a column in the metadata model. Contains these fields:

- id: the id of the column
- name: the localized name of the column
- fieldType: the field type of the column. One of Column.FIELD_TYPES.DIMENSION, Column.FIELD_TYPES.FACT, Column.FIELD_TYPES.ATTRIBUTE, Column.FIELD_TYPES.KEY

- dataType: the data type of the column. One of Column.DATA_TYPES.NUMERIC, Column.DATA_TYPES.STRING, Column.DATA_TYPES.DATE, Column.DATA_TYPES.BOOLEAN, Column.DATA_TYPES.UNKNOWN
- defaultAggType: the default aggregation type for the column. One of Column.AGG_TYPES.SUM, Column.AGG_TYPES.AVERAGE, Column.AGG_TYPES.MIN, Column.AGG_TYPES.MAX, Column.AGG_TYPES.COUNT, Column.AGG_TYPES.COUNT_DISTINCT, Column.AGG_TYPES.NONE
- category: the category of the column
- aggTypes: an array of the available aggregation types. See defaultAggType for the available types.

MetadataQuery Object

An object that helps create an MQL query. See the Metadata Report Sample and Google Geo Map sample for example code that uses this object.

MetadataQuery.addSelectionById

Adds a selection to the query. It returns the MetadataQuery.Selection object for the newly added selection.

Parameters

- columnId

MetadataQuery.addSelection

Adds the provided MetadataQuery.Selection object to the query.

Parameters

- selection: the MetadataQuery.Selection object to add to the query.

MetadataQuery.addSort

Adds the provided MetadataQuery.Sort object to the query.

Parameters

- sort: the MetadataQuery.Sort object to add to the query.

MetadataQuery.addFilter

Adds the provided MetadataQuery.Filter object to the query.

Parameters

- filter: the MetadataQuery.Filter object to add to the query.

MetadataQuery.getXML

No parameters. Returns the MQL XML for the query.

MetadataQuery.Selection Object

Defines a selection within a query. Sorting and filtering of the selected item are specified separately.

Fields

- column: the Column object for the selection
- aggType: the aggregation type for the selection.
- alignment:
- footerAggType:

MetadataQuery.Sort Object

Fields

- column: the Column object for the sort
- direction: the sort direction. One of Column.SORT_TYPES.ASCENDING, Column.SORT_TYPES.DESCENDING

MetadataQuery.Filter Object

Specifies a filter to be applied to the query results. Filters are independent of the selections - there does not have to be a corresponding selection object.

Fields

- column: the column to filter by
- condition: the condition type apply. One of Column.CONDITION_TYPES.LIKE, Column.CONDITION_TYPES.EQUAL, Column.CONDITION_TYPES.LESS_THAN, Column.CONDITION_TYPES.MORE_THAN
- value: the value to use. This can be a javascript array of values if condition is Column.CONDITION_TYPES.EQUAL
- operator: the operator to use between multiple filters. One of Column.OPERATOR_TYPES.OR, Column.OPERATOR_TYPES.AND

FilterHelper Object

An object that helps provide a GUI for creating query filters. See the Metadata Report Sample, Google Geo Map, and Metadata Query Sample for example code that uses this object.

Parameters

- filterColumn: the column object that is being filtered
- filterEditBoxId: the id of the HTML edit box used to display the filter value
- filterParameterState: a state object that the filter value will be added to
- model: the PentahoModel object

Enhancements and Limitations

I would like to merge these capabilities with CDA.

I would like to expose OLAP data sources in a similar way - with a client-side library to help construct queries. Maybe PAT has something we can use for this.

Repository Services

RepoSvc

This is a content generator that allows a new way to reference executable content in the solution repository. Instead of:

/pentaho/ViewAction?solution=steel-wheels&path=analysis&action=Product Line Sales Trend.xanalyzer

you can use

/pentaho/content/reposvc/steel-wheels/analysis/Product Line Sales Trend.xanalyzer

You can also access web resources such as images, script files, stylesheets etc.

The service works thus:

If the file indicated by the URL is registered to a plugin content generator, the handling of the request is delegated to the plugin.

Otherwise the contents of the file are returned to the browser if both of these apply:

- The file requests comes from a folder called resources/web within the solution
- The extension of the file is one of those registered in org.pentaho.platform.util.web.MimeHelper. Currently those file types are: .rtf .doc .pdf .xls .ppt .mpp .zip .mp3 .wav .bmp .gif .jpg .png .svg .tif .csv .html .txt .mpg .avi .css .js .xml .swf

This service provides a couple of enhancements for developing solutions. If your solution contains a mixture of server-side generated content (e.g. using action sequences) and client-side resources (scripts images etc), it can be difficult to map between the action sequence URLs and the URLs for the static files.

One of the results of this is that it is difficult to have two copies of a solution within the repository. The reposvc solves this problem because executable content and static files now use a single, consistent pathing mechanism that includes relative URLs.

Examples

An action sequence (content.xaction) generates an HTML fragment that needs to reference an image (logo.png). The solution needs to be structured like this:

MySolution

    content.xaction

resources

web

logo.png

Within the output of the xaction you can refer to the image using this relative URL ‘resources/web/logo.png’

Alternatively if you have a static HTML page (mypage.html) that needs to reference executable content (e.g. myview.xanalyzer), the structure would look like this:

MySolution

myview.xanalyzer

resources

web

mypage.html

The HTML page can reference the Analzer view with this relative URL ‘../../myview.xanalyzer’

RepositoryClientService

This web service provides thin client GUIs a way to save and load their state in the solution repository without having to write any server-side code.

The web service address is :

http://localhost:8080/pentaho/content/ws-run/RepositoryClientService

The Widget 1 sample Google Geo Map sample (under File->New) provide examples of using the client-side objects to access the web service.

The client-side javascript objects assist in saving and loading state. These objects are defined in jimmylabs-r1-repo/resources/web/common/pentaho-state.js and are described below.

The service provides the following functions.

loadState

Returns state that was previously saved. It is up to the client code to decipher the string returned.

Parameters

- filepath: the path and filename within the solution repository of the file that contains the state to be loaded.

saveStateString
Saves a JSON string to the repository on the server.

Parameters
- filepath: the path and filename within the solution repository of the file that contains the state to be saved. This is typically provided by the user console Save/Save As dialogs.
- state: The JSON string to be saved
- type: The file extension for the state type. This needs to be unique for each thin-client content type, e.g. ‘openlayers’, ‘googlemap’, ‘acmechartwidget’
- replace: Whether to replace the file if it exists already. This is typically provided by the user console Save/Save As dialogs.
- title: The name of the file. This is typically provided by the user console Save/Save As dialogs.
- description: The description for the file. This is not gathered by the user console Save/Save As dialogs.

saveStateXml
Saves a XML string to the repository on the server.

Parameters
- filepath: the path and filename within the solution repository of the file that contains the state to be saved. This is typically provided by the user console Save/Save As dialogs.
- state: The JSON string to be saved
- type: The file extension for the state type. This needs to be unique for each thin-client content type, e.g. ‘openlayers’, ‘googlemap’, ‘acmechartwidget’
- replace: Whether to replace the file if it exists already. This is typically provided by the user console Save/Save As dialogs.
- title: The name of the file. This is typically provided by the user console Save/Save As dialogs.
- description: The description for the file. This is not gathered by the user console Save/Save As dialogs.

PentahoRepositoryClient Object
A javascript object that provides a client-side API to the RepositoryClientService.

Fields
- SERVICE_URL: The URL for the service. Can be changed by the client if necessary.
- fileType: The file extension to be used for saving and loading state for the client.
- getStateAsXmlCallback: A callback that will be used when the state needs to be saved. An XML string is expected. One of getStateAsXmlCallback and getStateAsJsonCallback should be set. If both are set the XML callback will be used.
- getStateAsJsonCallback: A callback that will be used when the state needs to be saved. Any kind of string can be returned, including JSON.

PentahoRepositoryClient shouldLoad

No parameters. Inspects the URL to see if a load operation is expected. The client code is expected to initiate a state load if shouldLoad returns true.

PentahoRepositoryClient shouldEdit

No parameters. Inspects the URL to see if an edit operation is expected. The client code is expected to enter edit mode if shouldEdit returns true.

PentahoRepositoryClient.loadStateStringFromUrl

No parameters. Loads state from the server based on the URL parameters. Returns a StateObject object.

PentahoRepositoryClient.loadStateString

Loads state from the server. Returns a StateObject object.

Parameters

- solution: The name of the solution to load from
- path: the folder path within the solution to load from
- filename: the file to load

StateObject

Fields

- status: The status of operation - 'SUCCESS' is the value to hope for,
- state: The state loaded (if the operation was a load operation)
- message: The human-friendly message from the server

Samples

User Console API Test

This sample demonstrates integration points with the user console.

You can launch this using Tools->Console Integration Sample.

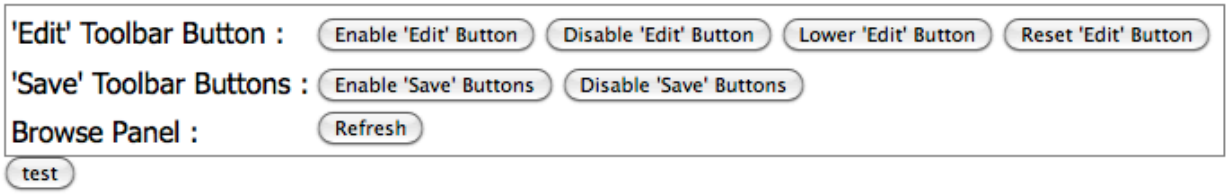
The source file is pentaho-solutions/jimmylabs-r1-samples/resources/web/console-sample/consoletest.html

User Console API Test

State

Running in console :	true
Save Information :	Filename - test filename
	Solution - steel-wheels
	Path - /reports
	Full File Path - steel-wheels/reports/test filename
	Type - html
	Overwrite - true

Actions



State Section

- Running in console: identifies if the page is running within the user console
- Save Information: After the save buttons have been enabled and tried this section shows the data provided to the page by the Save As dialog.

Actions Section

- 'Edit' Toolbar Button: These buttons show how to manipulate the console ‘edit’ toolbar button.
- 'Save' Toolbar Buttons: These buttons show how to manipulate the console ‘Save’ and ‘Save As’ toolbar buttons.

Browse Panel: This button shows how to refresh the solution browser.

Metadata Model Sample

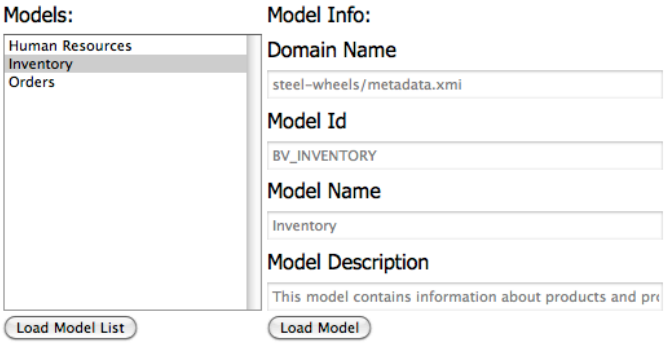
This sample demonstrates integration points with the metadata client objects.

You can launch this using Tools->Metadata Models Sample.

The menu option is defined in pentaho-solutions/jimmylabs-r1-samples/plugin.xml.

The source file is pentaho-solutions/jimmylabs-r1-samples/resources/web/metadata-samples/model-sample.html.

Models



Columns

Category	Id	Name	Data Type	Field Type	Agg Type	Default Agg Type	Available Agg Types
CAT_PRODUCTS - Products							
	BC_PRODUCTS_PRODUCTCODE	Product Code	TEXT	ATTRIBUTE	NONE	NONE	NONE
	BC_PRODUCTS_PRODUCTNAME	Product Name	TEXT	DIMENSION	NONE	NONE	NONE
	BC_PRODUCTS_PRODUCTLINE	Product Line	TEXT	DIMENSION	NONE	NONE	NONE
	BC_PRODUCTS_PRODUCTSCALE	Product Scale	TEXT	ATTRIBUTE	NONE	NONE	NONE
	BC_PRODUCTS_PRODUCTVENDOR	Product Vendor	TEXT	DIMENSION	NONE	NONE	NONE
	BC_PRODUCTS_PRODUCTDESCRIPTION	Product Description	TEXT	ATTRIBUTE	NONE	NONE	NONE
CAT_INVENTORY - Inventory and Cost							
	BC_PRODUCTS_QUANTITYINSTOCK	Quantity In Stock	FLOAT	FACT	NONE	NONE	NONE
	BC_PRODUCTS_BUYPRICE	Buy Price	FLOAT	FACT	NONE	NONE	NONE
	BC_PRODUCTS_MSRP	MSRP	FLOAT	FACT	NONE	NONE	NONE

Click on ‘Load Model List’ to retrieve the lists of available models from the server and display them.

Clicking on a model in the ‘Models’ list will display its name, id, domain, and description.

Clicking on ‘Load Model’ will display the details of the model in the Columns section.

The javascript in the HTML file shows how to work with the metadata client objects to work with the metadata models in this way.

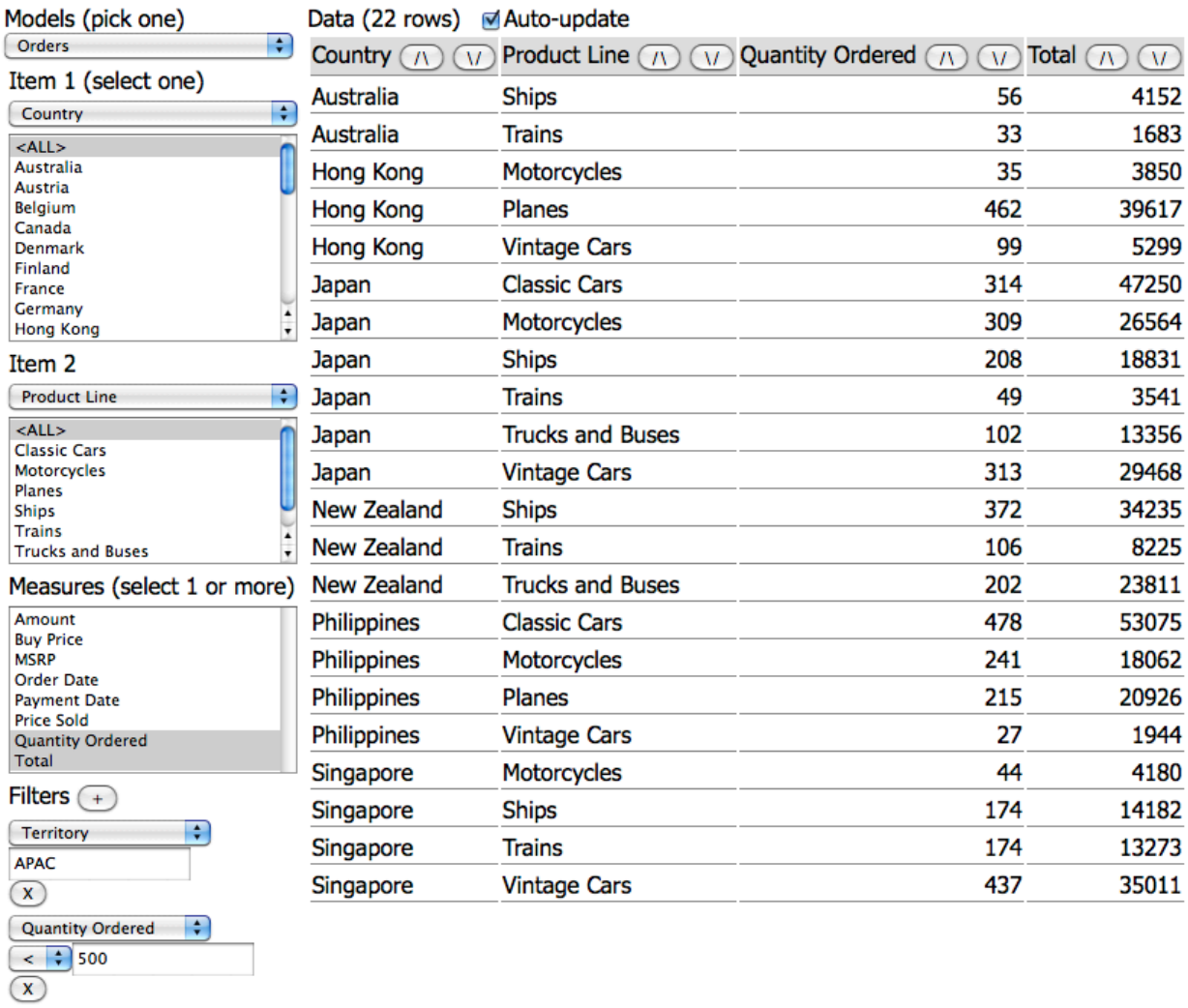
Metadata Query Sample

This sample demonstrates integration points with the metadata client objects. It shows how to construct a query and submit it.

You can launch this using Tools->Metadata Query Sample.

The menu option is defined in pentaho-solutions/jimmylabs-r1-samples/plugin.xml.

The source file is pentaho-solutions/jimmylabs-r1-samples/resources/web/metadata-samples/query-sample.html.



Select a model from the Models list to populate the ‘Item 1’ select, ‘Item 2’ select, and Measures list box.

When you select a field for Item 1 or Item 2 the corresponding list box is populated with the members of that field. You can select to see all the members (<ALL>) or you can select one or more members from the list.

You can select one or more measures from the measures list.

Only the first 100 rows of data are displayed since this is a code sample demonstrating API usage, not visualization techniques.

If the Auto-update checkbox is checked data will be loaded every time the selections are modified.

You can add filters by clicking on the ‘+’ button next to ‘Filters’. When you select a column the appropriate control(s) will be displayed to let you enter a value for the filter. Only text and numeric fields are supported by the UI.

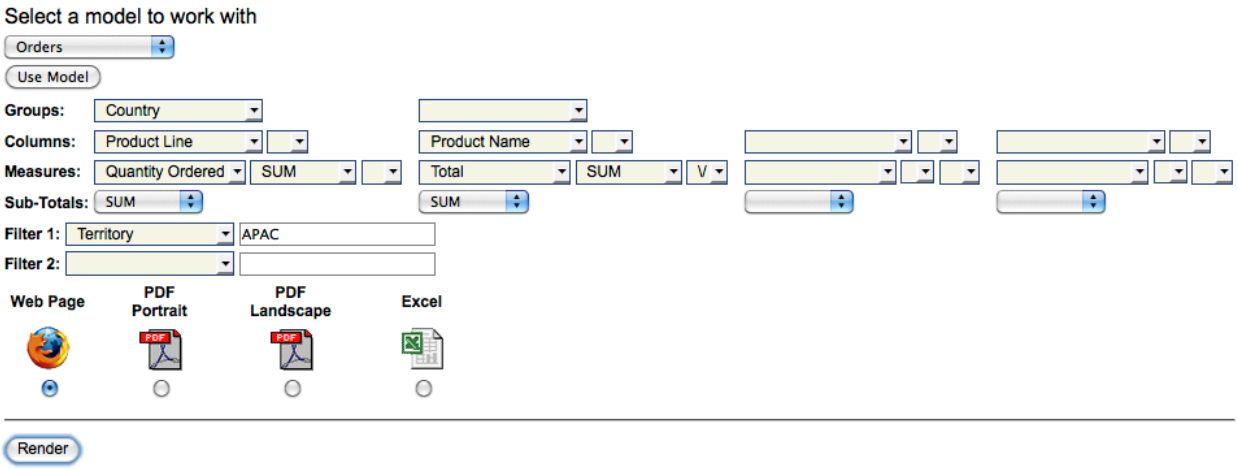
Metadata Report Sample

This sample demonstrates working with the Report Spec object. This API is likely to be replaced with a newer, better alternative.

You can launch this using Tools->Metadata Report Sample.

The menu option is defined in pentaho-solutions/jimmylabs-r1-samples/plugin.xml.

The source file is pentaho-solutions/jimmylabs-r1-samples/resources/web/report-sample/quickreport.html.



November 02, 2010 @ 02:59

Country: Australia

Product Line	Product Name	Quantity Ordered	Total
Trucks and Buses	1958 Setra Bus	188	$25,455.00
Vintage Cars	1913 Ford Model T Speedster	231	$22,998.00
Vintage Cars	1928 Mercedes-Benz SSK	118	$22,931.00
Motorcycles	2003 Harley-Davidson Eagle Drag Bike	117	$22,825.00
Classic Cars	1976 Ford Gran Torino	140	$19,401.00
Classic Cars	1952 Alpine Renault 1300	109	$19,292.00
Trucks and Buses	1940 Ford Pickup Truck	189	$17,115.00
Vintage Cars	1937 Lincoln Berline	126	$15,130.00
Motorcycles	1996 Moto Guzzi 1100i	141	$14,819.00
Vintage Cars	18th Century Vintage Horse Carriage	126	$14,298.00

Pick a model to work with. When you click on 'Use Model' the other selection controls will be populated. You can select report groups, columns, measure, sorting, sub-totals, filters and output types.

This sample uses the legacy ReportSpec document format. This should be upgraded to use a more up-to-date report specification.

Widget 1

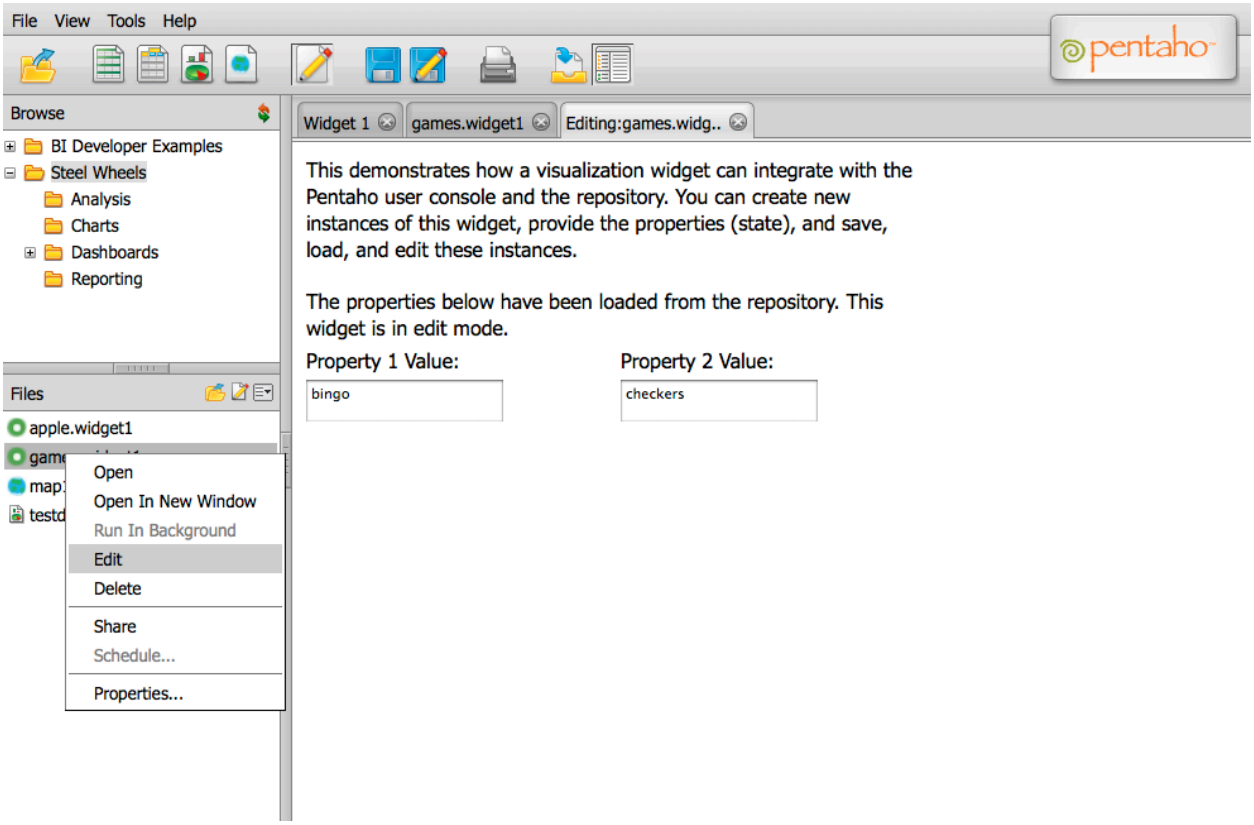
This sample shows how to integrate with the PentahoRepositoryClient API.

To create a new ‘Widget 1’ choose File->New->Widget 1. The menu option is defined in pentaho-solutions/jimmylabs-r1-samples/plugin.xml.

The source file is pentaho-solutions/jimmylabs-r1-samples/resources/web/repo-samples/widget1.html.

The widget shows how to interact with the user console buttons in different modes: ‘new’, ‘open’, ‘edit’, and ‘new window’.

When the widget is in edit mode the values in the text boxes can be changed. The Save and Save As buttons can be used to save the widget instance into the repository. Saved widgets will show up in the repository browser and can be opened and edited. When opened or edited the widget will initialize itself with the appropriate state from the repository.



The widget plugin is configured in pentaho-solutions/jimmylabs-r1-samples/plugin.xml.

```
<!-- this content generator is not used but for now each content type must have
corresponding generator -->
<content-generator scope="local" id="widget1" type="widget1">
     <classname>org.pentaho.platform.engine.services.solution.ActionSequenceContent
Generator</classname>
    <title>This is only here to work around a bug in content type handling</title>
</content-generator>

<content-type type="widget1" mime-type="text/html">
     <title>Sample State File</title>
```

```
     <description>Sample State File</description>
     <icon-url>content/jimmylabs-r1-samples/resources/web/repo-samples/
widget1icon.png</icon-url>
        <meta-provider>com.pentaho.jimmylabs.solution.SolutionRepoFileInfo</meta-
provider>
     <operations>
          <operation>
               <id>RUN</id>
               <command>content/jimmylabs-r1-samples/resources/web/repo-samples/
widget1.html?solution={solution}&amp;path={path}&amp;filename={name}
&amp;command=load</command>
          </operation>
          <operation>
               <id>NEWWINDOW</id>
               <command>content/jimmylabs-r1-samples/resources/web/repo-samples/
widget1.html?solution={solution}&amp;path={path}&amp;filename={name}
&amp;command=load</command>
          </operation>
          <operation>
          <id>EDIT</id>
               <command>content/jimmylabs-r1-samples/resources/web/repo-samples/
widget1.html?solution={solution}&amp;path={path}&amp;filename={name}
&amp;command=edit</command>
          </operation>
     </operations>
</content-type>
```

The content generator is not used but must be defined for the content type to work correctly.

The type attribute in the content-type and content-generator nodes must match the value of the fileType variable defined on line 14 of widget1.HTML.

The icon that is shown in the solution browser for this widget is defined in the icon-url node.

The meta-provider node must be set to com.pentaho.jimmylabs.solution.SolutionRepoFileInfo.

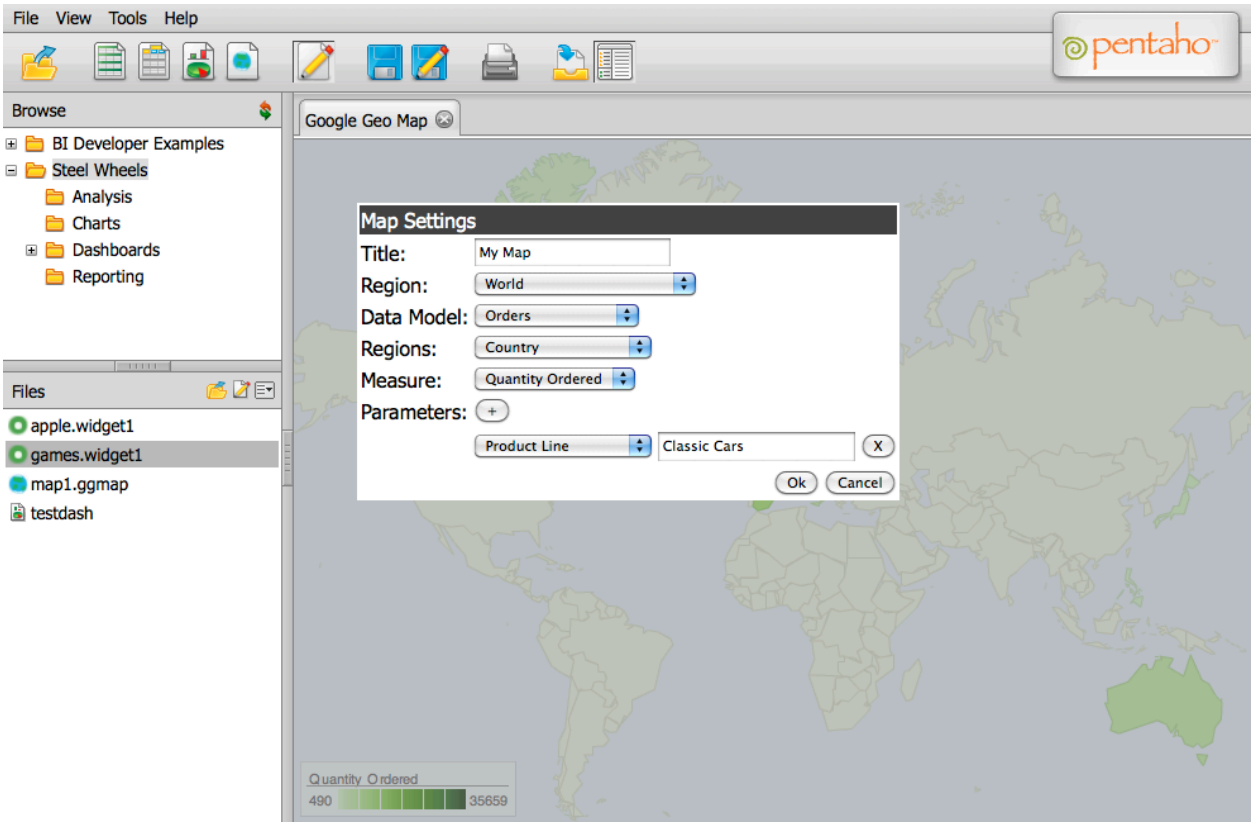
The RUN, NEWWINDOW, and EDIT operation ids are used to create the right-click menu options for the widget files. The command values must point to the static HTML file for the widget and must have parameters of ‘solution’, ‘path’, and ‘filename’. Other parameters can be added if the widget needs them.

Google Geo Map Sample

This sample shows how to integrate with the PentahoRepositoryClient API, with the toolbar buttons, and with the dashboard APIs.

To create a new ‘Google Geo Map’ choose File->New-> Google Geo Map. The menu option is defined in pentaho-solutions/jimmylabs-r1-samples/plugin.xml.

The widget is defined in pentaho-solutions/jimmylabs-r1-samples/resources/web/google-geo/google-geo-map.html and pentaho-solutions/jimmylabs-r1-samples/resources/web/google-geo/google-geo-map.js



The widget adds itself to the toolbar. The widget integrates with the toolbar Edit, Save, and Save As buttons.

The widget also works as a consumer of parameters from the dashboard framework.

Issues and Possible Enhancements
The map regions are missing important ones like ‘Europe’. Click events don’t seem to work.

Other Widget Ideas

The purpose behind the repository and metadata services and client-side objects is to help web developers create new plugins that integrate different browser visualization components into the BI server.

These widgets should be designed as both standalone pages and CDF-compatible components.

These components could include:

Charting: Google Visualization, Flot, NetCharts, jQuery, OpenFlash, ProtoViz, InfoViz, FusionCharts, AnyChart etc

Geographic: Google Maps, OpenLayers, OpenStreetMap etc

Tables and Grids: YUI etc

Others: Traffic lighting, gauges, marquee, ticker displays, etc