



Changing the Look of the User Console

Copyright Page

This document supports Pentaho Business Analytics Suite 5.2 GA and Pentaho Data Integration 5.2 GA, documentation revision October 7, 2014, copyright © 2014 Pentaho Corporation. No part may be reprinted without written permission from Pentaho Corporation. All trademarks are the property of their respective owners.

Help and Support Resources

To view the most up-to-date help content, visit <https://help.pentaho.com>.

If you do not find answers to your questions here, 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, visit <http://www.pentaho.com/training>.

Liability Limits and Warranty Disclaimer

The author(s) of this document have used their best efforts in preparing the content and the programs contained in it. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The author and publisher make no warranty of any kind, express or implied, with regard to these programs or the documentation contained in this book.

The author(s) and Pentaho shall not be liable in the event of incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of the programs, associated instructions, and/or claims.

Trademarks

The trademarks, logos, and service marks ("Marks") displayed on this website are the property of Pentaho Corporation or third party owners of such Marks. You are not permitted to use, copy, or imitate the Mark, in whole or in part, without the prior written consent of Pentaho Corporation or such third party. Trademarks of Pentaho Corporation include, but are not limited, to "Pentaho", its products, services and the Pentaho logo.

Trademarked names may appear throughout this website. Rather than list the names and entities that own the trademarks or inserting a trademark symbol with each mention of the trademarked name, Pentaho Corporation states that it is using the names for editorial purposes only and to the benefit of the trademark owner, with no intention of infringing upon that trademark.

Third-Party Open Source Software

For a listing of open source software used by each Pentaho component, navigate to the folder that contains the Pentaho component. Within that folder, locate a folder named licenses. The licenses folder contains HTML files that list the names of open source software, their licenses, and required attributions.

Contact Us

Global Headquarters Pentaho Corporation Citadel International, Suite 460

5950 Hazeltine National Drive Orlando, FL 32822

Phone: +1 407 812-OPEN (6736)

Fax: +1 407 517-4575

<http://www.pentaho.com>

Sales Inquiries: sales@pentaho.com

Introduction

This section contains a variety of instructions for post-install configuration and maintenance tasks of Pentaho software and the Pentaho Business Analytics (BA) Server. You can do these tasks in any order that you want, because the operation of the BA Server is not dependent on doing these tasks in order.

Prerequisites

Before you do any of these tasks, you must have [installed](#) the Pentaho software and [configured](#) your BA Server.

Expertise

The topics covered in this section are written for IT administrators who know where data is stored, how to connect to it, details about the computing environment, and how to use the command line interface for Windows or Linux.

Tools

We provide a web application, the User Console, which you use to perform many administration tasks.

Login Credentials

All of the tasks that use the User Console, **Administration** page, require that you [log on to the User Console](#) with the Pentaho administrator user name and password.

Pentaho User Console Styling

The User Console is the standard web application for the BA Server, and includes interactive elements of Reporting, Analysis and Dashboards. The [Administration](#) page of the console is the central framework through which your BA Server is configured and managed. You can alter the look and feel of the User Console by editing its configuration files, graphics, and CSS style sheets manually.

The Pentaho web application server must be stopped before editing anything inside of the Pentaho WAR file.

Location of Customizable Configuration Files

MantleStyle.css

This is the structural cascading style sheet for the Pentaho User Console. It inherits some elements from the **Widgets.css** file in the same directory, so you may need to look at that as well. Modifying these styles could have dramatic impact on Pentaho User Console rendering.

```
/pentaho/server/biserver-ee/tomcat/webapps/pentaho/mantle/MantleStyle.css
```

Theme-specific local and global style sheets for PUC

These directories contain style sheets and other theme materials for each BA Server client tool.

File Name	Location	Description
globalCrystal.css	/pentaho-solutions/system/common-ui/resources/themes/crystal/	This is the main structural, theme-specific style sheet for the default theme for the User Console (Crystal).
mantleCrystal.css	tomcat/webapps/pentaho/mantle/themes/crystal/	This is the customizable presentation portion of the theme style sheet for the default theme for the Pentaho User Console (Crystal).
globalOnyx.css	/pentaho-solutions/system/common-ui/resources/themes/onyx/	This is the main structural, theme-specific style sheet for an alternate theme for the Pentaho User Console (Onyx).
mantleOnyx.css	tomcat/webapps/pentaho/mantle/themes/onyx/	This is the customizable presentation portion of the theme style sheet for an alternate theme for the Pentaho User Console (Onyx).

Product-specific theme settings for Analyzer, Dashboard Designer, and Interactive Reporting

These directories contain style sheets and other theme materials for each.

NOTE:

Set the `<cache>false</cache>` in `...pentaho-solutions/system/common-ui/settings.xml` while customizing these plugins. Otherwise, the changes are not visible without a server restart.

Product Name	Location	Description
Analyzer	<code>/pentaho-solutions/system/analyzer/styles/themes/</code>	Style sheets and theming materials for Analyzer.
Interactive Reporting	<code>/pentaho-solutions/system/pentaho-interactive-reporting/resources/web/themes/</code>	Style sheets and theming materials for Interactive Reporting.
Dashboard Designer	<code>/pentaho-solutions/system/dashboards/themes/</code>	Style sheets and theming materials for Dashboard Designer.
Mobile	<code>/pentaho-solutions/system/pentaho-mobile-plugin/resources/css</code>	Style sheets and theming materials for the Mobile application.

Change Logo on Application Login Page

You can change the **User Console** login page to display a different logo than the default one, such as your corporate logo or other image.

1. Backup and remove the file **puc-login-logo.png** from the `/pentaho/server/biserver-ee/pentaho-solutions/system/common-ui/resources/themes/images` directory.
2. Rename your own logo image to **puc-login-logo.png**.
3. Copy the image into the `/pentaho/server/biserver-ee/pentaho-solutions/system/common-ui/resources/themes/images` directory. Your users must clear the cache on their web browsers and reboot to be able to see the new logo.

The default Pentaho logo is removed and your custom logo appears in its place.

Add a Logo to the Web Application

You can customize the Pentaho web application to display a logo. The best way to do this is to edit the **Mantle.jsp** file in the `/pentaho/server/biserver-ee/tomcat/webapps/pentaho/mantle/` folder.

In this example, the logo is added to the header on the right, but you can add it to the left or as a banner on the top or bottom of the page.

```
<div id="pucWrapper" cellspacing="0" cellpadding="0" style="width: 100%; height: 100%;">

    <div id="pucHeader" cellspacing="0" cellpadding="0">

        ...
    <div id="logo" style="float: right; padding-right: 10px"><IMG src="mantle/logo.png"></IMG></div>

        ...
    </div>
</div>
```

The logo.png is used as the graphic within the **pucHeader** div.

Change the Background Image

You can change the background image that appears in the content pane in the Pentaho User Console by modifying or replacing the `bg_pentaho_default.png` file in the `pentaho/system/common-ui/resources/themes/<theme-name>/images` folder. The folder and file name must be identical.

Make Custom User Console Themes

The User Console's graphical interface is built on a CSS-based theme system. The sections below contain information for theme designers and developers.

- [Theme System Overview](#)
- [Create a New Theme](#)
- [Set the Default Theme](#)
- [Switch Console Themes](#)

Theme System Overview

The CSS-based theme system allows you to change the look of the BA Server and its client tools by creating and adding your own themes. This is accomplished by working with just a few key configuration files.

The style sheets that make up the look and feel of the User Console, Dashboard Designer, Analyzer and Interactive Reporting are mostly in one location. These styles and scripts combine to make the default system theme, **Crystal**. This theme is located in the Common UI plug-in directory: `/pentaho/server/biserver-ee/pentaho-solutions/system/common-ui/resources/themes`.

There are two different kinds of themes: **system** and **local**. System themes provide common styles and scripts that apply across the entire BA Server. For instance, buttons are defined in the default system theme, Crystal. A change to the Crystal system theme will change the way buttons look in all applications. Local themes are defined for a particular area or "context" of the BA Server. Contexts include BA Server plug-ins as well as the names of the top-level directories in the Pentaho WAR. Resources for local themes take effect only in their particular area of the BA Server.

Any page shown by the BA Server that includes the **webcontext.js** script will automatically have all of the active theme's JavaScript and CSS files included. For instance, below is a sample theme containing one CSS and JavaScript file:

```
<themes root-folder="style">
  <crystal display-name="Crystal" system="true">
    <file>crystalStyles.css</file>
    <file>crystalScripts.js</file>
  </crystal>
</themes>
```

When the Crystal theme is active, the following are added to the HTML page:

```
<script type="text/javascript" src="/pentaho/common-ui/themes/crystal/
crystalStyles.js"></script>
<link rel="stylesheet" type="text/css" href="/pentaho/common-ui/themes/crystal/
crystalStyles.css"/>
```

This automatic insertion of theme resources makes it possible to change themes without having to edit the main content HTML documents to include the theme resource's tag individually. The theme system will include any number of JavaScript and CSS files defined for your theme.

You can add local styles in a similar fashion. The only requirement is that you tell the system what context you need to load. This is done by adding **?context=myPlugin** to the **webcontext.js** script where **myPlugin** is the name of your plug-in or root WAR folder:

```
<script type="text/javascript" src="webContext.js?context=myPlugin"></script>
```

Create a New Theme

The best way to create a new theme is to copy an existing theme folder, and use that to create your new theme.

On startup, the BA Server searches for **themes.xml** files in every plug-in and root-level folder in the Pentaho WAR. Multiple themes can be defined in one themes.xml file, and they can be system, local, or a combination of both. The following example defines a system and local theme named Crystal.

```
<themes root-folder="resources/themes">
  <crystal display-name="Crystal" system="true">
    <file>crystalStyles.css</file>
    <file>crystalScripts.js</file>
  </crystal>
  <crystal display-name="Crystal" system="true">
    <file>localCrystalStyles.css</file>
  </crystal>
</themes>
```



```
</crystal>
</themes>
```

Notice that the **<themes>** node has a **root-folder** attribute. The value of this attribute is the name of the directory (relative to the Web application context) where your themes are stored. For WAR-based contexts, this is simply a directory name inside the WAR. For instance, if your theme is located in the `/pentaho.war/accounting/` folder, the resources would be loaded from `/pentaho.war/accounting/resources/themes/`.

Plug-in resource loading is different than WAR-based loading in that the plug-in controls how resources are mapped to the URL. If the theme file from above was located in a plug-in called **accounting**, then the resources would be accessed from the following URL: `/pentaho/context/accounting/resources/themes/`. This kind of resource mapping is most commonly implemented in BA Server plug-ins through **static-path** entries:

```
<static-paths>
  <static-path url="/accounting/resources" localFolder="resources"/>
</static-paths>
```

Set the Default Theme

The default system theme is defined in the `/pentaho-solutions/system/pentaho.xml` configuration file through the **default-theme** node. The BA Server ships with Crystal as the default theme; changing the value to another theme name will set the default active theme for all User Console users.

Switch Console Themes

If you have created an alternate theme and/or localized message bundle, you can switch to it through the **View** menu in the Pentaho User Console.

Note: You can prevent themes from appearing in this menu by adding a **hidden="true"** property to the theme node.

You can manually specify a theme for a particular page by including a **theme=** URL parameter. This will affect only the requested page. The following will load the **debug** system and local themes if available:

```
http://localhost:8080/pentaho/content/myPlugin/index.html?theme=debug
```

If either the system or local debug theme is not found, the resources for the current active theme loads instead. This is particularly useful when testing out new themes and for loading debug versions of scripts and styles.

It is also possible to set the session variable **pentaho-user-theme** to the desired theme name. This is usually done in a start-up action to have per-user themes in multi-tenancy scenarios.

Test Your CSS Changes

The style sheet files explained earlier in this section contain many style definitions, which makes it difficult to map the CSS classes to what you see in the application at runtime. To make this task easier, we recommend that you use the [DevTools](#) for Chrome, [Web Inspector](#) for Safari, or [Firebug](#) for Firefox to inspect the UI elements you want to style.

Once you've selected an element, you can view all of the CSS properties that apply to it, then change the CSS and watch your changes take immediate effect. This allows you to quickly and easily find the appropriate CSS to edit and gives you the ability to preview your changes in real time.

Localization

This section explains how to localize all Pentaho applications. There are two main paradigms for internationalization: programs that abstract their integrated textual content to unified message bundles, and programs that store text in individual properties files inside of JARs.

- [BA Server and Thin Client Message Bundles](#)
- [Design Tool Localization](#)
- [Use the Kettle Translator](#)

BA Server and Thin Client Message Bundles

Note: Stop the Pentaho application server before editing anything inside of the Pentaho WAR file.

You can localize the Pentaho User Console, Pentaho Analyzer, and Dashboard Designer by creating locale- and language-specific message bundles within the Pentaho Web application. Message bundles are dynamically adjusted according to browser locale, so you can create localized message bundles for every language you want to support, and let each individual user's system language settings determine which one is loaded.

Localization [language packs](#) have been created through a community initiative that uses an installer built by Webdetails, a Pentaho company. The language packs apply to both EE and CE, and are maintained by community members. The packs are not officially supported by Pentaho.

For brevity's sake, only the default Pentaho User Console files will be explained in detail. The file naming convention is identical among all message bundles in Pentaho Business Analytics. The following files are located in the `/mantle/messages/` directory:

- **`mantleMessages.properties`**: The default message bundle for the Pentaho User Console. In its initial condition, it is a copy of **`mantleMessages_en.properties`**. If you want to change the default language and dialect, copy your preferred message bundle file over this one.
- **`mantleMessages_en.properties`**: The English-language version of the standard message bundle. This is an identical copy of **`mantleMessages.properties`**.
- **`mantleMessages_fr.properties`**: The French-language version of the standard message bundle.
- **`mantleMessages_de.properties`**: The German-language version of the standard message bundle.
- **`mantleMessages_supported_languages.properties`**: Contains a list of localized message bundles and the native language names they correspond to; this relieves the BA Server of the burden of having to discover them on its own. A `supported_languages.properties` file should be created for every message bundle that you intend to localize.

Example of `mantleMessages_supported_languages.properties`

```
en=English
de=Deutsch
fr=Français
```

New files are created in the following format: **`mantleMessages_xx_YY.properties`** where **`xx`** represents a lowercase two-letter language code, and **`YY`** represents a two-letter locale code, where applicable. So, for instance, U.S. and British English could have two separate message bundles if you wanted to draw a distinction between the two dialects:

- **`mantleMessages_en_US.properties`**
- **`mantleMessages_en_GB.properties`**

The language and country codes must be in standard ISO format. You can look up both sets of codes on these pages:

- http://en.wikipedia.org/wiki/List_of_ISO_639-1_codes
- http://www.iso.org/iso/country_codes/iso_3166_code_lists.htm

You can edit the default message bundle directly if you need to make surgical changes to the content inside of it. If you plan to translate it into another language, it makes more sense to copy the file and change the name appropriately, then translate it line by line. Be sure to update `supported_languages.properties` with the new country and dialect code and the native language name that it corresponds to.

Note: All message bundles must be UTF-8 encoded.

- [Pentaho User Console and Dashboard Designer Message Bundles](#)
- [Pentaho Analyzer Localization](#)
- [Pentaho Interactive Reporting Localization](#)
- [Geographic Service \(pentaho-geo\) Localization](#)
- [Report Viewer Localization](#)

Pentaho User Console and Dashboard Designer Message Bundles

The message bundle locations for the Pentaho User Console and Dashboard Designer are:

- `/pentaho.war/mantle/messages/` for the Pentaho User Console
- `/pentaho/server/biserver-ee/pentaho-solutions/system/dashboards/resources/messages/` for Dashboard Designer
- `/pentaho/server/biserver-ee/pentaho-solutions/system/dashboards/resources/gwt/messages/` also for Dashboard Designer

Pentaho Analyzer requires a special set of instructions for localizing their analysis schema message bundles.

See [Localization/Internationalization of Analysis Schemas](#) for more information.

Note: Dashboard Designer requires both sets of message bundles, and both are different, so you cannot simply copy one set of messages from one directory to the other.

- [Switch User Console Message Bundles](#)

Switch User Console Message Bundles

If you've created a localized message bundle, you can switch to it through the **View** menu in the User Console.

Pentaho Analyzer Localization

Follow the directions below to create localized message bundles for Pentaho Analyzer.

Note: This covers only the Analyzer interface, not the OLAP data sources you use with Analyzer. For schema localization, refer to [Localization/Internationalization of Analysis Schemas](#).

1. If the BA Server is currently running, shut it down.
2. Make a copy of the **messages.properties** file in `/pentaho-solutions/system/analyzer/resources/`; name the copy according to the standard locale naming scheme defined earlier in this section.

```
cp messages.properties messages_fr.properties
```

3. Translate the content of the new message bundle into the locale defined in its file name.
4. Edit the **messages_supported_languages.properties** file in `/pentaho-solutions/system/analyzer/resources/` and add the new locale.

```
fr=Francais
```

You now have a translated Analyzer message bundle available in your BA Server.

- [Localization/Internationalization of Analysis Schemas](#)

Localization/Internationalization of Analysis Schemas

You can create internationalized message bundles for your analysis schemas and deploy them with the Pentaho Web application. This enables Pentaho Analyzer to access localized schemas.

1. Edit your analysis schema and tokenize all values that you want to localize. Typically you would create variables for all caption and description values.

```
<Schema measuresCaption="%{foodmart.measures.caption}">
  <Dimension name="Store" caption="%{foodmart.dimension.store.caption}"
description="%{foodmart.dimension.store.description}">
    <Hierarchy hasAll="true" allMemberName="All Stores"
allMemberCaption="%{foodmart.dimension.store.allmember.caption =All
Stores}" primaryKey="store_id" caption="%{foodmart.hierarchy.store.country.
caption}" description="%{foodmart.hierararchy.store.country.description}>
      <Table name="store"/>
      <Level name="Store Country" column="store_country"
uniqueMembers="true" caption="%{foodmart.dimension.store.country.caption}"
description="%{foodmart.dimension.store.country.description}"/>
  </Dimension>
</Schema>
```

2. Create localized **MondrianMessages.properties** files in the `/WEB-INF/classes/com/pentaho/messages/` directory inside of the Pentaho WAR, and define each token you used in the analysis schema.

Note: JBoss users will have to delete the unpacked Pentaho WAR directory if it exists, then unpack the pentaho.war file with an archive utility, create the message bundles in the proper location, then repack it into a WAR again.

If you need further assistance in creating localized message bundles, refer to [BA Server and Thin Client Message Bundles](#).

```
foodmart.measures.caption=Measures
foodmart.dimension.store.country.caption=Store Country
foodmart.dimension.store.name.property_type.column=store_type
foodmart.dimension.store.country.member.caption=store_country
foodmart.dimension.store.name.property_type.caption=Store Type
foodmart.dimension.store.name.caption=Store Name
foodmart.dimension.store.state.caption=Store State
foodmart.dimension.store.name.property_manager.caption=Store Manager
foodmart.dimension.store.name.property_storesqft.caption=Store Sq. Ft.
foodmart.dimension.store.allmember.caption=All Stores
```

```
foodmart.dimension.store.caption=Store
foodmart.cube.sales.caption=Sales
foodmart.dimension.store.city.caption=Store City
foodmart.cube.sales.measure.unitsales=Unit Sales
```

3. Edit the **mondrian.properties** file in the `/pentaho/server/biserver-ee/pentaho-solutions/system/mondrian/` directory and add this line (or modify it if it's already there):

```
mondrian.olap.localePropFile=com.pentaho.messages.MondrianMessages
```

4. Save and close the file.
5. Restart the BA Server.
6. Login to the User Console with administration permissions, then click **Manage Data Sources**, then the **Add** button. Choose **Analysis** from the menu. Browse to import your file.
7. Edit your Analysis data source by checking the option next to **Manually enter data source parameters**.
 - a. If absent from the list of parameters, add one parameter called **DataSource** whose value is the name of the JDBC data source to use.
 - b. Create a new parameter called **DynamicSchemaProcessor** whose value is `mondrian.i18n.LocalizingDynamicSchemaProcessor`.
 - c. Create a new parameter called **UseContentChecksum** whose value is `true`.
8. In the User Console, go to **Tools > Refresh > Mondrian Schema Cache**.

Your analysis schemas will now be localized to whatever language is currently selected in the Pentaho User Console, if a message bundle for that locale was copied to the proper directory as explained above.

- [BA Server and Thin Client Message Bundles](#)
- [Pentaho Analyzer Localization](#)
- [Set a Default Font for PDF Exports](#)

Pentaho Interactive Reporting Localization

Follow the directions below to create localized message bundles for Pentaho Interactive Reporting.

Warning: Interactive Reporting uses interface elements from the **common-ui** BA Server plugin, which does not currently support more than one locale at a time.

1. If the BA Server is currently running, shut it down.
2. Make a copy of the **messages.properties** file in `/pentaho-solutions/system/pentaho-interactive-reporting/resources/messages/`; name the copy according to the standard locale naming scheme defined earlier in this section.

```
cp messages.properties messages_fr.properties
```

3. Translate the content of the new message bundle into the locale defined in its file name.
4. If you want to localize the common-ui interface elements that Interactive Reporting uses, edit the **messages.js** files in `/pentaho-solutions/system/common-ui/resources/web/dojo/pentaho/common/nls/` and `/pentaho-solutions/system/common-ui/resources/web/dataapi/nls/`

You now have a translated Interactive Reporting message bundle available in your BA Server.

Geographic Service (pentaho-geo) Localization

To localize the Pentaho Geographic Service, follow the below instructions.

1. Stop the BA Server if it is currently running.
2. Copy the default **messages.properties** file in `/pentaho-solutions/system/pentaho-geo/resources/messages/` to a localized properties file in the same location.

This file contains client-side text within the map visualization, including tooltips.

Note: Do not attempt to localize other properties files in this directory.

3. Extract the default **messages.properties** file from the `pentaho-geo---check--5.2.0.jar` archive in the `/pentaho-solutions/system/pentaho-geo/lib/` directory. This file contains server-side text for various service status and error messages.
4. Create a localized version of the extracted properties file, then repack it into the same location in the JAR.

The Pentaho Geographic Service now has localized message bundles for the locale you specified in the properties file names.

Report Viewer Localization

Follow the directions below to create localized message bundles for the Report Viewer plugin.

Note: This is for the **Report Viewer**, which displays reports created with Report Designer. The instructions here, while similar, do not provide localization support for Interactive Reporting, or for individual reports. Refer to [Pentaho Interactive Reporting Localization](#) for Interactive Reporting localization.

Warning: The date picker is an interface element from the **common-ui** BA Server plugin, which does not currently support more than one locale at a time.

1. If the BA Server is currently running, shut it down.
2. Make a copy of the **messages.properties** file in `/pentaho-solutions/system/reporting/messages/`; name the copy according to the standard locale naming scheme defined earlier in this section.

```
cp messages.properties messages_fr.properties
```

3. Translate the content of the new message bundle into the locale defined in its file name.
4. Inside of the Pentaho Web application, unpack the `/WEB-INF/lib/pentaho-reporting-engine-classic-core-platform-plugin--check--5.2.jar` JAR, then navigate to the `/org/pentaho/reporting/platform/plugin/messages/` directory inside of it.
5. Make a copy of the **messages.properties** file in this directory; name the copy according to the standard locale naming scheme defined earlier in this section.

```
cp messages.properties messages_fr.properties
```

6. Translate the content of the new message bundle into the locale defined in its file name, and remove all of the properties that you don't change. Leaving the unchanged properties in the file will cause duplicate definitions, which cause unnecessary overhead and difficulty in future translation efforts.
7. Re-pack the JAR. If the original JAR was not deleted when you unpacked it, you will have to delete it before re-packing.

You now have translated Report Viewer (and Reporting plugin) message bundles available in your BA Server.

Design Tool Localization

All of Pentaho's design tools can be translated to the extent that their text strings are abstracted to properties files. There are already a number of localized message bundles in some design tools (PDI in particular), though they may be incomplete.

The basic process for translating message bundles is to search for **messages.properties** files and create localized versions of them, following the standard Java localization naming convention as explained in [Use the Kettle Translator](#).

Once you've found the JARs containing the message bundles, create overrides for them by creating a **translations** directory and unpacking the messages.properties files from the JARs into it:

```
mkdir translation
cd translation
unzip ../lib/*.jar '*.properties'
```

Then add that directory to the classpath by editing **launcher.properties**, and edit or create a localized message bundle for each properties file you extracted.

The quickest and easiest ways to locate the appropriate JARs, add override directories, and create message bundles is explained for each design tool in the below sections.

Aggregation Designer

Search for: /aggregation-designer/lib/pentaho-*.jar

Path to launcher.properties: /aggregation-designer/lib/

Modified launcher.properties classpath: classpath=../translations:log4j.xml

Data Integration (Kettle)

Search for: /data-integration/lib/kettle-*.jar

Path to launcher.properties: /data-integration/launcher/

launcher.properties classpath prefix: classpath=../translations:

Note: The Kettle project has a translator tool to help facilitate large-scale localization efforts. To get it, check out the Kettle source code and run **translator.sh** or **Translator.bat** and either check in the changes, or zip them and email the archive to a committer. See [Use the Kettle Translator](#) for more information.

Metadata Editor

Search for: /metadata-editor/lib/pentaho-metadata-editor*.jar /metadata-editor/libext/pentaho/pentaho-*.jar

Edit the **metadata-editor** (.sh or .bat) script, and modify the **CLASSPATH** variable accordingly:

```
CLASSPATH=./translations:
```

Report Designer

Search for: /report-designer/lib/pentaho-reporting-*.jar /report-designer/lib/report-design*.jar

Path to launcher.properties: ./

launcher.properties classpath prefix: classpath=translations:

Schema Workbench

Search for: /schema-workbench/lib/workbench.jar /schema-workbench/lib/mondrian.jar

Edit the **workbench** (.sh or .bat) script, and locate the following line: **CP="{MONDRIAN_HOME}/lib/commons-collections.jar"**. Now create a new line directly below it and paste this in:

```
CP="{CP}{PS}{MONDRIAN_HOME}/translations"
```

BA Server and DI Server

This process is described in much more detail in *BA Server and Thin Client Message Bundles*. However, if you're following the same abbreviated procedure as above, here are the basics (minus the thin client plugins):

Search for: /biserver-ee/tomcat/webapps/pentaho/WEB-INF/lib/pentaho-bi-platform-*.jar

Path to use as a classpath override (you don't have to declare it): /biserver-ee/tomcat/webapps/pentaho/WEB-INF/classes/

Use the Kettle Translator

Warning: This procedure involves downloading and compiling source code. The program you build will not be covered by standard Pentaho support agreements. If you want to translate message bundles without compiling from source, you can simply search for `messages_en_US.properties` files within the data-integration-client directory (including all of the JARs therein), as explained in [Design Tool Localization](#). Properties files can be inserted into JARs without invalidating your support agreement; you can also override those JARs with directories in the PDI classpath.

Pentaho Data Integration has thousands of translatable strings. Because there are so many, and the number of strings increases with each new step and entry, Pentaho provides a graphical translation tool to make the process easier.

Before commencing a translation project, check to make sure the target language isn't already available. PDI already has complete or partially-complete translations for the following locales:

- en_US
- it_IT
- fr_FR
- ja_JP
- es_AR
- ko_KR
- zh_CN
- de_DE
- es_ES
- pt_BR
- nl_NL
- pt_PT

Additionally, there are certain translation rules and guidelines for some languages. You may need to search for the current rules and guidelines for the language you are using.

To use the translation tool, follow the below directions.

1. Check out the PDI source code for the release you are creating localized message bundles for. The project root is: <https://github.com/pentaho/pentaho-kettle>.
2. Run the translator tool (**Translator.bat** on Windows; **translator.sh** on Linux) in the root directory of the PDI source code that you checked out. The translator GUI will come up.
3. In the upper left pane, select (or create) the locale to translate into.
4. In the large left pane, select the package name to work on. Packages are color coded. Light grey means there are between 1 and 5 missing keys; dark grey means there are 6-10; yellow means 11-25; orange means 26-50; and red packages have more than 50 keys that are missing translations for the selected locale. Once you select a package, the translatable keys list will populate.
5. Select a key from the **Todo list**: pane in the middle of the window.
6. Type in the translated text in the **Translation**: pane on the right, then click **Apply**.

7. When you've completed your translation effort, click **Save**.
8. If you intend to contribute your translations back to the Kettle project, you can commit them back to the source tree, or you can click the **Zip** button in the translator interface to collect all of the new translated message files into a single zip archive. You can email this archive to the Kettle project lead at mcasters@pentaho.org
Note: Pentaho Data Integration is open source software, and represents the contributions of many people. The project maintainers ask that you contribute back any localized message bundles that you create.
9. In order to use PDI with these translated messages, you must recompile it from the source you checked out.

```
ant -f build.xml clean targz
```

If you've translated all of the available keys, then PDI should be entirely localized for your target language. Any untranslated keys will fall back to its superset language (messages_fr_FR.properties will fall back to messages_FR.properties), and as a last resort, to **messages_en_US.properties**.

Create Design Tool Templates

The following Pentaho products have customizable templates:

- Report Designer (Report Design Wizard)
- Interactive Reporting
- Dashboard Designer
- Community Dashboard Framework (CDF)

This section explains how to modify the default templates and, where applicable, how to create your own.

- [Create a Dashboard Designer Template](#)
- [Create Reporting Templates](#)

Create a Dashboard Designer Template

Follow the directions below to create a new Pentaho Dashboard Designer template.

Note: You will have to migrate this template by hand if/when you upgrade Dashboard Designer, since the template files will be stored in the Dashboard Designer plugin directory. The upgrade procedure that Pentaho provides does not typically cover customizations like this one, except to mention that they must be migrated by hand.

1. Stop the BA Server.
2. Navigate to the `/pentaho/server/biserver-ee/pentaho-solutions/system/dashboards/templates/xul/` directory.
3. If you want to remove all templates that you know will never be used, you can safely delete their corresponding XUL files now. When you are done, also remove the corresponding files from the `html` sibling directory.
4. Copy the existing XUL file that most closely resembles the template layout you want to create, giving the new XUL file a name that starts with a two-digit number that represents the template's order in the hierarchy, followed by a short description of its dimensions. All Dashboard Designer templates follow this naming convention.
5. Create a `.properties` file that corresponds to the one you just copied in the previous step, and put one item in it: `name=Description here`, where "Description here" represents the display name of this template.
6. Create a thumbnail graphic that fits the same dimensions as the other PNG thumbnails in this directory, and give it the same name as the previous two files, with a PNG extension. You should now have three new files, all with the same name, with three different extensions: `.xul`, `.properties`, and `.png`.
7. Edit the new `.xul` file and change the `box` attributes to match your template specifications.

A `vbox` node creates a column; an `hbox` node creates a row; a `box` element defines an individual panel in each row.

`height` and `width` define static widths in pixels; the `flex` size attribute defines a percentage of the total width of the dashboard. If you'd like more extensive definitions of XUL elements, refer to the official XUL documentation: http://developer.mozilla.org/en/XUL_Reference.

Note: Ensure that each `box`, `vbox`, and `hbox` node has its own unique `id`.

8. Save and close all open files, then start the BA Server.
9. Test your new template and adjust its configuration accordingly.

You now have a custom Dashboard Designer template deployed to your BA Server.

You must copy the template files by hand if you upgrade the BA Server or Dashboard Designer in the future. You may want to back up your custom templates to a safe location right now just in case you forget to copy them over during a future upgrade.

Create Reporting Templates

Both the Report Design Wizard and Interactive Reporting use templates for rapid report creation. The template creation process is similar for both RDW and PIR -- in both cases, you use Pentaho Report Designer to create a report with certain template options selected, then deploy it to the client tools that will use it. The instructions below explain how to create templates, how reports are generated from templates, and how to display templates within RDW and PIR.

- [Report Design Wizard Template Design Guidelines](#)
- [Interactive Reporting Template Design Guidelines](#)
- [Dynamic Element Positioning in Templates](#)
- [Template Properties](#)
- [Deploy a Template to Report Design Wizard](#)
- [Deploy a Template to Interactive Reporting](#)
- [Set the Default Interactive Reporting Template](#)

Report Design Wizard Template Design Guidelines

Methodology

To create a new RDW template, you must use Report Designer to create a report with certain template-specific properties enabled. The resultant PRPT file is then deployed to the Report Designer and/or Pentaho Data Integration template directory.

Requirements

The **generated-content-marker** attribute in the **wizard** group is the flag that turns a normal report into a template. This attribute can only be applied to a band (group header, group footer, details header, details footer, details, or sub-band).

RDW will insert its auto-generated content into the first band with the **generated-content-marker** set to true. This applies to the group header, group footer, details header, details footer, and the details bands. In the event there are more groups defined in the Report Design Wizard than defined in the template, it repeats the last defined group header and footer in the template.

Formatting Inheritance

Formatting styles are inherited, so any formatting applied to a band will also be applied to the elements used within it. Formatting is applied in three ways and in the following order:

1. Through the template via band inheritance
2. Through query data where it is defined in Pentaho Metadata
3. As defined by RDW users through the RDW interface

Inheriting Styles from the Data Query

The **query-metadata** section of the Attributes tab contains options that determine whether formatting styles can come from the data query and be applied to the detail header, details, or detail footer band. This must be set directly on the detail header, detail footer, or details band; and the **style-format** option must be set to **true** for it to work. You must also disable any individual formatting styles (enable-style-*=true) that you don't want to come from the query.

Padding and Grid Lines

Since the Details band is dynamically generated, you have to specify grid line and padding settings in the template definition. This is done through the **wizard** attribute group for the band that has the **generated-content-marker** enabled.

Updating

An RDW template is only a set of initial defaults for a report, so if a template is updated, completed reports that were based on that template will not be affected; there is no connection between the template and the report once the report is saved. If you want to update an RDW-based report to reflect template changes, you can edit the report with Report Design Wizard, make any necessary selections, and re-save it.

Note: Interactive Reporting templates have the opposite behavior because the report links itself to the template; when an IR template is changed, all reports based on that template will automatically inherit the updated template.

Interactive Reporting Template Design Guidelines

Methodology

To create a new Interactive Reporting template, you must use Report Designer to create a report with certain template-specific properties enabled. The resultant PRPT file is then deployed to the IR plugin's template directory.

Requirements

The **generated-content-marker** attribute in the **wizard** group is the flag that turns a normal report into a template. This attribute can only be applied to a band (group header, group footer, details header, details footer, details, or sub-band).

RDW will insert its auto-generated content into the first band with the **generated-content-marker** set to true. This applies to the group header, group footer, details header, details footer, and the details bands. In the event there are more groups defined in the Report Design Wizard than defined in the template, it repeats the last defined group header and footer in the template.

Formatting Inheritance

Formatting styles are inherited, so any formatting applied to a band will also be applied to the elements used within it. Formatting is applied in three ways and in the following order:

1. Through the template via band inheritance
2. Through query data where it is defined in Pentaho Metadata
3. As defined by IR users through the Interactive Reporting interface

Inheriting Styles from the Data Query

The **query-metadata** section of the Attributes tab contains options that determine whether formatting styles can come from the data query and be applied to the detail header, details, or detail footer band. This must be set directly on the detail header, detail footer, or details band; and the **style-format** option must be set to **true** for it to work. You must also disable any individual formatting styles (`enable-style-*=true`) that you don't want to come from the query.

Padding and Grid Lines

Since the Details band is dynamically generated, you have to specify grid line and padding settings in the template definition. This is done through the **wizard** attribute group for the band that has the **generated-content-marker** enabled.

Updating

A template is not just a set of initial defaults for a report -- it is the basis for that report. So if a template is updated, completed reports that were based on that template will also change.

Note: Report Design Wizard templates have the opposite behavior; when an RDW template is changed, none of the reports based on that template will be automatically be updated with those changes. Instead, you will have to edit each report, apply the new template, and save it.

Dynamic Element Positioning in Templates

Use the following methods to accommodate for multiple page sizes in reports:

Percentages

Express the height, width, x position, and y position in percentages.

Block, inline, or row

Switch the band's layout mode from canvas to either block, inline or row:

layout-mode	value
canvas	Uses the x and y position to place the element in the band.
block	Stacks elements vertically according to the layer order in a band; width is set to 100%.
inline	Stacks elements horizontally according to the layer order in a band; width is determined by the length of the text in the field, and wraps within the band.
row	Stacks elements horizontally in one row according to the layer order in the band.

Dynamic height message elements

Set **dynamic-height=true** on message elements. This will allow the element size to grow according to line height. Also, setting the following Reporting engine configuration option will allow the element size to grow according to the font size used:

```
org.pentaho.reporting.engine.classic.core.layout.fontrenderer.UseMaxCharBounds = true
```

Proportional column widths

To support varying window sizes in a Web browser, enable the use of proportional column widths so that the resulting table will have a width of 100% and the columns will have the proportional equivalent of their static widths as width. The relevant Reporting engine option to set this globally is:

```
org.pentaho.reporting.engine.classic.core.modules.output.table.html.  
ProportionalColumnWidths = true
```

Template Properties

The following band properties affect RDW and PIR report templates. Notice that many properties have a different effect in RDW than in PIR.

Band	Report Design Wizard	Interactive Reporting
page header	None	Edit message elements
report header	None	Edit message elements
group headers	Inserts a message element with the value of: fieldname: \$(fieldname)	Edits any message elements within the band. Inserts a message element with the value of: fieldname: \$(fieldname)
detail header	1: Inserts a column header label for every field used. Default value for the label is the fieldname. 2: Edits any label within the band.	1: Inserts a column header label for every field used. Default value for the label is the fieldname. 2: Edits any label within the band.
details	Inserts the appropriate fields defined	Inserts the appropriate fields defined
detail footer	Insert a numeric field in the same x-position and width of the details field that the summary calculation is applied.	None

group footer	None	1: Inserts a numeric field in the same x-position and width of the details field that the summary calculation is applied. 2: Inserts a message element in the x-position of the details field with no summary calculations applied.
report footer	None	1: Inserts a numeric field in the same x-position and width of the details field that the summary calculation is applied. 2: Inserts a message element in the x-position of the details field with no summary calculations applied. 3: Edit any message elements within the band
page header	None	Allows the modification of message elements

Deploy a Template to Report Design Wizard

Once you've created a template for RDW, you must follow the below process to deploy it to Report Design Wizard in Report Designer and Pentaho Data Integration.

1. Shut down Report Designer and Pentaho Data Integration if either of them are currently running.
2. Create an icon for your template, in PNG format, with the same name as the template file. The size of the icon doesn't matter; RDW will scale it to fit the correct dimensions. However, you can avoid unusual scaling issues by creating a square-shaped (equal width and height) graphic. If you'd like further guidance, take a look at the default template icons that Pentaho provides in the templates directory.
3. Copy the icon and the PRPT template files to the following directories:
 - /pentaho/design-tools/report-designer/templates/
 - /pentaho/design-tools/data-integration/plugins/spoon/agile-bi/templates/

Your template is now deployed to Report Design Wizard and will be available when you next start Report Designer or Data Integration.

Deploy a Template to Interactive Reporting

Once you've created a template for PIR, you must follow the below process to deploy it to the Interactive Reporting plugin.

1. Shut down the BA Server if it is currently running.
2. Create an icon for your template, in PNG format, with the same name as the template file. The size of the icon doesn't matter; PIR will scale it to fit the correct dimensions. However, you can avoid unusual scaling issues by creating a square-shaped (equal width and height) graphic. If you'd like further guidance, take a look at the default template icons that Pentaho provides in the templates directory.
3. Copy the icon and the PRPT template files to the `/pentaho-solutions/system/pentaho-interactive-reporting/resources/templates/` directory.
4. Edit the `/pentaho-solutions/system/pentaho-interactive-reporting/resources/messages/messages.properties` file and add a new line for your template with the **template_** prefix, the name of your template file, and a friendly name for the template as you'd like it to appear in the PIR interface, as in the following example (given a template filename of **template_demo.prpt**):

```
template_template_demo=Template Demo
```

Your template is now deployed to Pentaho Interactive Reporting.

Set the Default Interactive Reporting Template

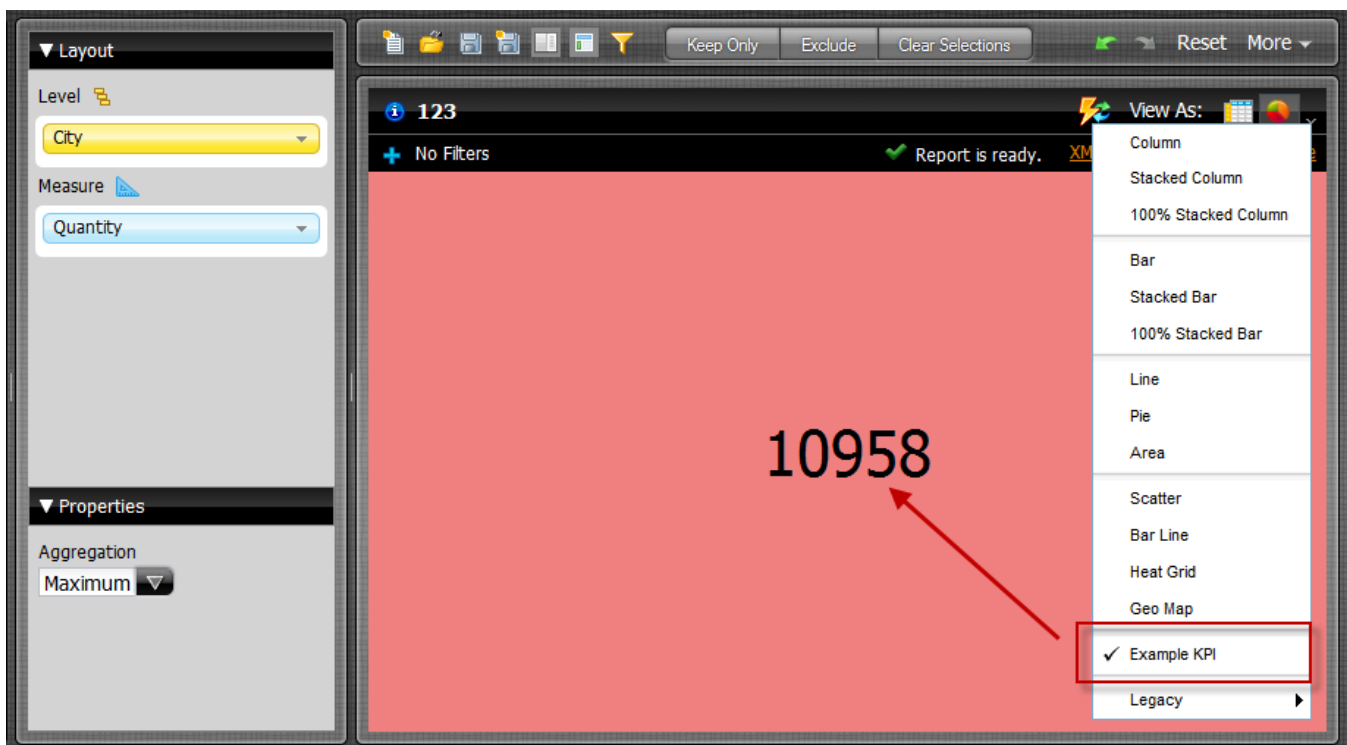
To change the default template for Interactive Reporting, edit the `/pentaho-solutions/system/pentaho-interactive-reporting/settings.xml` file and change the value of the `<default-template>` node. You do not have to provide a path to the template PRPT file -- just the filename.

```
<!-- default template -->
<default-template>1_jade_1_left_aligned.prpt</default-template>
```

Extend Pentaho Analyzer with Custom Visualizations

You can develop third-party visualizations and integrate them into Pentaho Analyzer. This section will show you how to create an example KPI Visualization. The example will show you how to generate a simple Key Performance Indicator (KPI) which calculates a minimum, maximum, or average on a single measure across a single level.

In this example, you create this visualization:



This visualization generates a simple KPI that calculates a minimum, maximum, or average on a single measure across a single level.

To extend a custom visualization, perform these actions in order:

1. Create a Pentaho BA Server Plug-in
2. Define the custom visualization
3. Register the visualization with Pentaho Visualization API
4. Register the visualization with Pentaho Analyzer
5. Register the created JavaScript files with Pentaho Analyzer
6. Restart the BA Server and test the visualization

- [Create a Pentaho BA Server Plug-in](#)
- [Custom Visualizations](#)
- [Register the Visualization with Pentaho Visualization API](#)
- [Register the Visualization with Pentaho Analyzer](#)
- [Register the Created JavaScript Files with Pentaho Analyzer](#)
- [Restart the BA Server and Test the Visualization](#)
- [Additional Resources](#)

Create a Pentaho BA Server Plug-in

Ensure that you are running BA Server with the Pentaho Analyzer plug-in installed.

1. Create a location for the new visualization in the BA Server by creating a new folder in the `pentaho-solutions/system` directory.
2. Within the new folder, create a `plugin.xml` file that will contain metadata about the plug-in.

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin title="example-visualization">
  <static-paths>
    <!-- this translates to /pentaho/content/example-visualization/
resources/* -->
    <static-path url="/example-visualization/resources"
localFolder="resources"/>
  </static-paths>
</plugin>
```

This basic plug-in exposes static files in the resources folder to authenticated users.

Custom Visualizations

Visualizations in Pentaho Analyzer are rendered on the client and are JavaScript based.

You must define a constructor and two key functions in the visualization JavaScript class so that Pentaho Analyzer can render it properly. The constructor of the visualization class will be passed in an HTML element, which acts as the parent node of your visualization.

The first function is `resize(width, height)`. The `resize` function is called every time the visualization renders as a different size, and allows the visualization to adjust calculations.

The second function is `draw(dataView, vizOptions)`. This function is called when rendering the visualization. The `dataView` object contains a multi-dimensional result set returned from the server, and the `vizOptions` object contains any customized options specified by the user when configuring the visualization.

- [Define the Custom Visualization](#)

Define the Custom Visualization

This example creates a very basic KPI visualization.

1. Create a file in the `example-visualization/resources` folder called `example.js`.
2. Add the following content to the file:

```
/* define a namespace for this sample to live in */
pentaho.sample = {};

/* define the KPI Class, which renders a single KPI */
pentaho.sample.KPI = function(canvasElement) {
  this.canvasElement = canvasElement;
  this.numSpan = document.createElement("span");
  this.numSpan.style.fontSize = "42px";
  this.numSpan.style.position = "relative";
  this.canvasElement.appendChild(this.numSpan);
};

/* Calculate the location of the KPI relative to the canvas */
pentaho.sample.KPI.prototype.resize = function(width, height){
  this.numSpan.style.left = ((this.canvasElement.offsetWidth - this.numSpan.
offsetWidth) / 2) + 'px';
  this.numSpan.style.top = ((this.canvasElement.offsetHeight - this.numSpan.
offsetHeight) / 2) + 'px';
};

/* Render the KPI */
pentaho.sample.KPI.prototype.draw = function(datView, vizOptions) {
  // extract the values from the result set
  var rows = datView.dataTable.jsonTable.rows;
  var dataArray = [];
  for(var i=0; i<rows.length; i++){
    dataArray.push(rows[i].c[1].v);
  }

  // calculate the KPI to display
```

```

var value = 0;

// note that the vizOptions contains an aggregate option,
// this is a custom property specific for this visualization type.
switch(vizOptions.aggregate){
  case "MAX":
    value = Number.MIN_VALUE;
    for(var i=0; i< dataArray.length; i++){
      value = Math.max(value, dataArray[i]);
    }
    break;
  case "MIN":
    value = Number.MAX_VALUE;
    for(var i=0; i< dataArray.length; i++){
      value = Math.min(value, dataArray[i]);
    }
    break;
  case "AVG":
    var total = 0;
    for(var i=0; i< dataArray.length; i++){
      total += dataArray[i];
    }
    value = total / dataArray.length;
    break;
  default:
} // Update the background color
this.canvasElement.style.backgroundColor =
vizOptions['myBackgroundColor'];
// write the KPI value to the screen
this.numSpan.innerHTML = value;
this.resize();
}

```

This basic visualization is the entry point for a more advanced visualization. If you have a Flash component, HTML5, or SVG visualization library, you can make calls to those elements here and wire them into the innerHTML of the canvas element.

For a more advanced example, see the Community Chart Components components here:

```
pentaho-solutions/system/common-ui/resources/web/vizapi/ccc/ccc_wrapper.js
```

These charts include the heat grid, which is enabled by default in Pentaho Analyzer.

Register the Visualization with Pentaho Visualization API

After defining the visualization, register the visualization with Pentaho's Visualization API. The Visualization API is only accessible from within Pentaho Analyzer.

1. Register the visualization with the Visualization API. Include the following JavaScript at the beginning of the `example.js` file:

```
// Utilize the required API for verifying that the VizController has been
loaded before registration
pen.require(["common-ui/vizapi/VizController"], function(){

// Register the visualization metadata with the Visualization API
pentaho.visualizations.push({
  id: 'pentaho_sample_KPI',      // unique identifier
  type: 'kpi',                  // generic type id
  source: 'Example',            // id of the source library
  name: 'Example KPI',          // visible name, this will come from a
properties
                                // file eventually
  'class': 'pentaho.sample.KPI', // type of the Javascript object to
instantiate
  args: {                       // arguments to provide to the
Javascript object
                                // this allows a single class to act as
multiple visualizations
  aggregate: 'AVG'
  },
  propMap: [],
  dataReqs: [                   // dataReqs describes the data
requirements of
                                // this visualization
    {
      name: 'Default',
      reqs :
      [
```

```

        {
            id: 'rows',                // ID of the data element
            dataType: 'string',        // data type - 'string', 'number',
            'date',                    // 'boolean', 'any' or a comma
                                     // separated list
            dataStructure: 'column',   // 'column' or 'row' - only
            'column' supported
                                     // so far
            caption: 'Level',          // visible name
            required: true,            // true or false
            allowMultiple: false,
            ui: {
                group: 'data'
            }
        },
        {
            id: 'measures',
            dataType: 'number',
            dataStructure: 'column',
            caption: 'Measure',
            required: true,
            allowMultiple: false,
            ui: {
                group: "data"
            }
        },
        {
            id: 'aggregate',
            dataType: 'string',
            values: ['MIN', 'MAX', 'AVG'],
            ui: {
                labels: ['Minimum', 'Maximum', 'Average'],
                group: 'options',
                type: 'combo',         // combo, checkbox, slider, textbox, gem,
                                     // gemBar, and button are valid ui types
                caption: 'Aggregation'
            }
        }
    ]

```



```
    }  
    ],  
    menuOrdinal: 10001,  
    menuSeparator: true,  
    maxValues: [1000, 2000, 3000]  
  });
```

2. To enclose the required API call, you must define `});` at the end of the `example.js`

All defined values are essential to registering the visualization with Pentaho's Visualization API.

The `datareqs` subcomponent is especially critical because it defines the data requirements for the visualization, which are used by Pentaho Analyzer to determine which fields are displayed in the field list.

Use `menuOrdinal` to specify the location of the visualization in Analyzer's chart drop-down menu. The default chart types such as Bar chart start at `menuOrdinal` 100 and then increment by 100. If you want your visualization to appear last, use a value greater than 10000.

`menuSeparator` can be set to true to insert a menu separator before the visualization in the menu.

Use `maxValues` to increase the available list of maximum plot values. This setting appears in Chart Options > Other > Domain Limit. If your visualization supports a very large number of values, then you would increase the value so the data table provided in the draw method includes more rows or columns.

Register the Visualization with Pentaho Analyzer

After registering the component with the Visualization API, you need to register the component with Pentaho Analyzer. This step is necessary so that Analyzer can customize its layout panel for the visualization, and to manage the serialization of the visualization's metadata.

To register with Analyzer, create a file in the `example-visualization/resources` folder called `example_analyzer_plugin.js`, and add the following content to the file:

```
var analyzerPlugins = analyzerPlugins || [];  
analyzerPlugins.push(  
  {  
    init:function () {  
  
      // Register visualizations to display in Analyzer  
      cv.pentahoVisualizations.push(pentaho.visualizations.getById(  
        'pentaho_sample_KPI'));  
  
      /*  
       Helpers contain code that knows about the Analyzer specific context. The one  
       function that's required "generateOptionsFromAnalyzerState" is called so the  
       visualization can set its own options based on Analyzer's current report.  
      */  
      cv.pentahoVisualizationHelpers['pentaho_sample_KPI'] = {  
        // Use one of Analyzer's stock placeholder images.  
        placeholderImageSrc: CONTEXT_PATH  
          + 'content/analyzer/images/viz/VERTICAL_BAR.png',  
  
        /*  
         This method allows a visualization to generate visualization specific  
         options based on Analyzer's report definition. In the following example,  
         this visualisation is setting a background color using the same background  
         color setting in Chart Options. You can figure out the existing chart  
         options by looking at the report XML by clicking the XML link in Analyzer.  
  
         @return a hash object containing the custom state of your visualization.  
        */  
      }  
    }  
  }  
);
```

```

generateOptionsFromAnalyzerState:function (report) {
    return {myBackgroundColor:
        report.reportDoc.getChartOption("backgroundColor")};
}
};

/*
LayoutConfig objects manage the interaction between Analyzer's Layout Panel
and the visualization's settings.
*/

// Declare a new class which extends the built-in version from Analyzer.
dojo.declare("SampleConfig", [analyzer.LayoutConfig], {

/**
 * @param config    The parse Configuration object which serves
 *                  as the model of the Panel.
 * @param item      The item in the panel which originated the event.
 * @param eventName The name of the event (clicked, value, etc).
 * @param args      A Hash Object containing relevent values (prevVal,
 *                  newVal, etc).
 */
onModelEvent: function(config, item, eventName, args) {

    if (eventName == "value") {
        // This component has a single argument, so we assume if this event is
        // fired it is for the aggregate option.
        // This will update the visualization args with the new value for
        // aggregate. Also note that when the Analyser report is saved, a
        // snapshot of the visualization args will be saved to the report XML.

        this.report.visualization.args['aggregate'] =
            config.byId('aggregate').value;

        //Add a report state item to the undo/redo history stack.

        this.report.history.add(new cv.ReportState("Update KPI Aggregation"));

        //Trigger a report refresh so that the visualization is updated with the
        //change.

```

```

        this.report.refreshReport();
    }
    this.inherited(arguments);
    // Let super class handle the insertAt and removedGem events.
    }
});

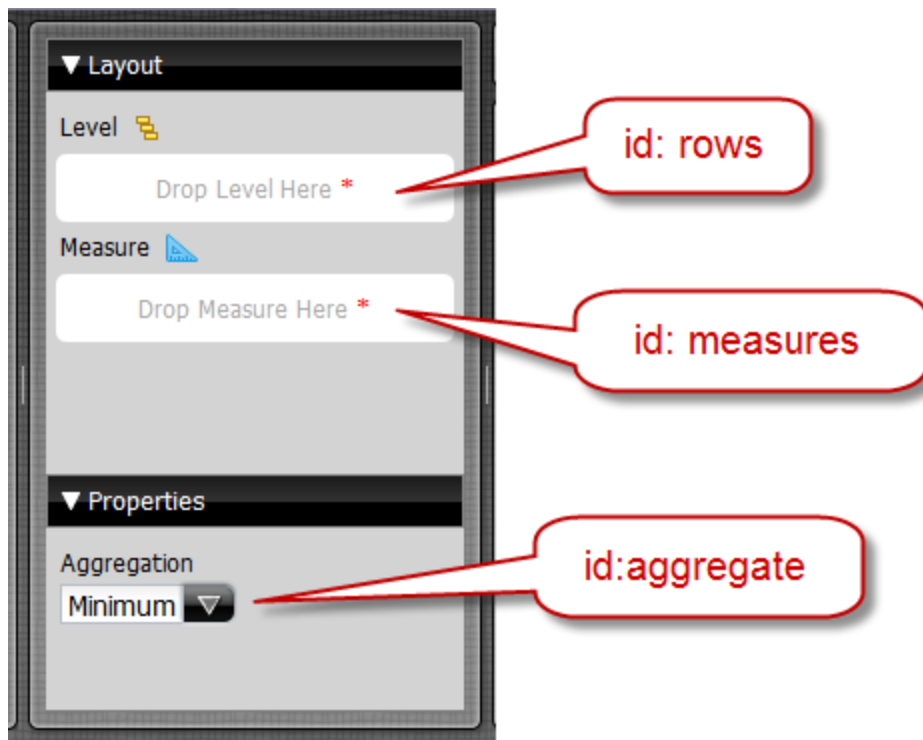
    // Register the Layout Panel Configuration Manager.
    // Note that the string entry matches 'JSON_' plus the visualization id
    // defined earlier.
    analyzer.LayoutPanel.configurationManagers['JSON_pentaho_sample_KPI'] =
        SampleConfig;
    }
}
);

```

In this example, you register a single visualization with Analyzer. The object definition in this JavaScript contains three main components. The first is the `init()` method call, this is called when Analyzer initializes the visualization. In this method, the visualization registers all the necessary handlers for working directly with Analyzer. The two handlers are the `VisualizationHelper` object and the `LayoutConfig` class.

The `VisualizationHelper` object contains code that is specific to getting your visualization to work in the context of Analyzer. The key function is to implement `generateOptionsFromAnalyzerState`, which is used to extract settings from the Analyzer report XML which can then be used in your own visualization.

The Layout Panel in Analyzer is generated dynamically from the JSON data requirements definition of the visualization, which was defined in *Pentaho Visualization API Registration*. For example, notice that the visualization automatically generated Level, Measure, and Aggregation components based on the rows, measures, and aggregate dataReq items:



In order for your visualization to respond to changes made in the panel, you registered the `SampleConfig` configuration manager with Analyzer.

Configuration Managers may extend the following methods:

<code>getConfiguration():</code>	Returns the layout panel definition. Normally, you will not need to extend this method. The default implementation in <code>LayoutConfig</code> will create gembars for dataReqs with <code>dataStructure = column</code> or <code>row</code> and restore the state by looking at the report XML attributes and measures. For custom properties such as the Aggregation dropdown in the above example, the default implementation will retrieve the current value from either the <code>generateOptionsFromAnalyzerState</code> function or from the <code>visualization.args</code> hash map.
<code>updateConfiguration(config):</code>	This is called on every report change and allows you to modify the layout panel state based on the current configuration. An example use case would be to make all gembars no longer required if any one of them contained at least one gem item.
<code>checkAcceptance(source, nodes, silent):</code>	Called by Drag and Drop operations in the panel. Return true if the operation is allowed. Normally you would not need to extend this method.
<code>onModelEvent(config, item, eventName, args):</code>	All layout panel interactions result in <code>onModelEvent</code> calls to the active Configuration Manager. These events will typically do one of three things:

- Add, remove or move gems. The base `LayoutConfig` will handle these events, update the report definition, and refresh the report.
- Update report chart properties. When `eventName == 'value'` and the `item.id` matches the property defined in the visualization `dataReq`, you can manually update the corresponding chart option in the report XML. The benefit of saving state in chart options is that it can be shared across visualizations. You can only set chart options that were previously supported in Analyzer, such as background color or label fonts/colors.
- Update visualization args. When `eventName == 'value'` and the `item.id` matches the property defined in the visualization `dataReq`, you can manually update the current visualization args. These changes will be passed back to you in the draw method's `vizOptions` parameter. Anything stored in `visualization.args` will be saved with the Analyzer report and is specific to your visualization only.

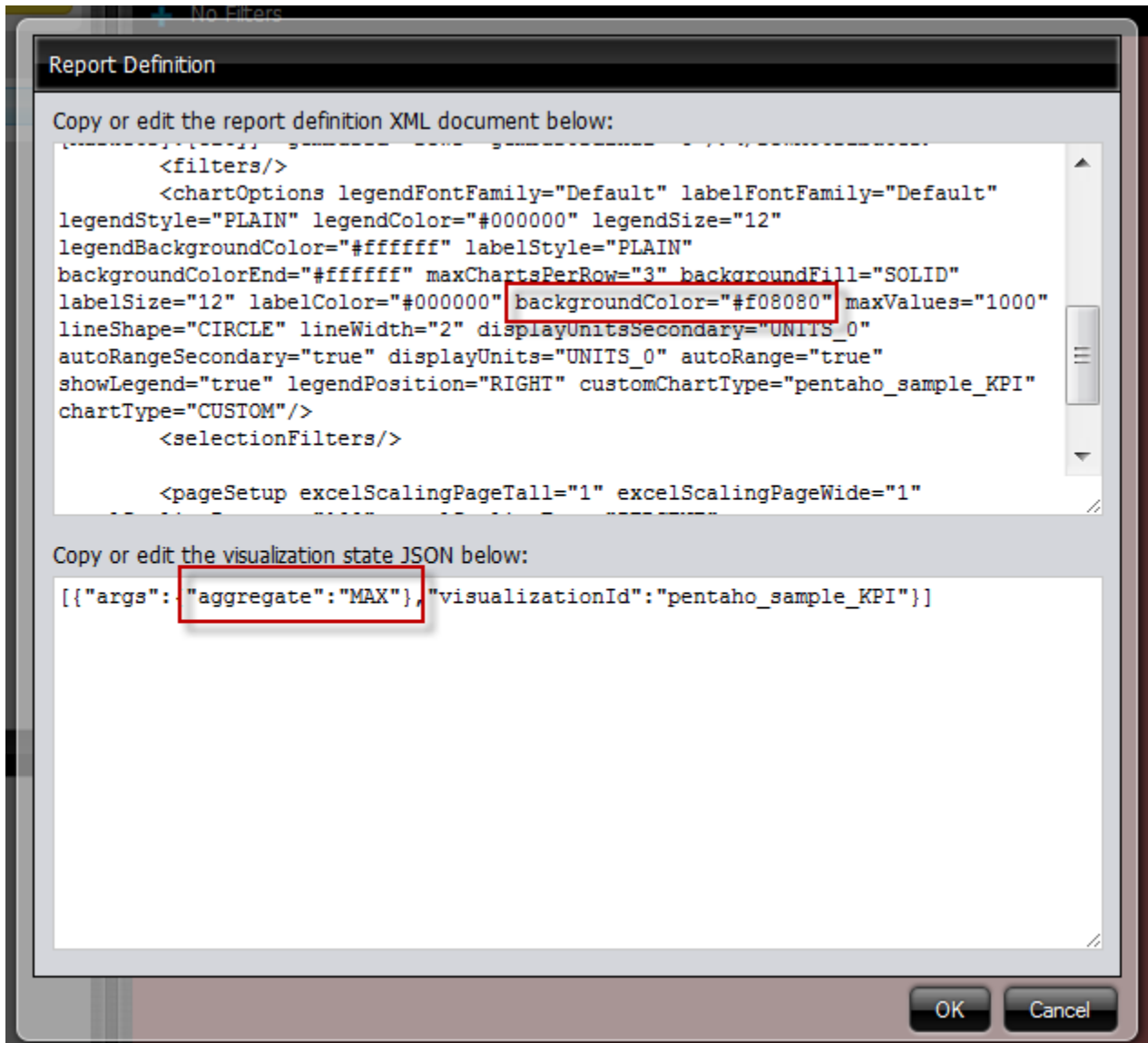
Register the Created JavaScript Files with Pentaho Analyzer

You must notify Pentaho Analyzer about the JavaScript files you have created. By adding the following child element of `plugin` to the `plugin.xml` file defined earlier, the JavaScript files will be included in Pentaho Analyzer.

```
<external-resources>
  <file context="analyzer">content/example-visualization/resources/example.
js</file>
  <file context="analyzer">content/example-visualization/resources/example_
analyzer_plugin.js</file>
</external-resources>
```

Restart the BA Server and Test the Visualization

Restart the BA Server. Once the server has restarted, you will see the visualization **Example KPI** registered in the list of Visualizations when creating an Analyzer report. Create a new report with the custom visualization selected. If you save the report and re-open it, and the visualization settings are remembered. Click on the XML link to view the report XML and visualization state JSON which is helpful in debugging state saving issues:



Test your visualization in the various browsers that will be used. You can add the XML text in the **Report Definition** dialogue box.

More complex examples included in the Pentaho BA Server noted in [Define the Custom Visualization](#). These examples go beyond this section, including functionality such as server-side printing, lasso'ing, and selections.

Additional Resources

Here are some additional custom visualizations resources, as well as some community examples of visualization plugins:

- [Providing Javascript Plugins via External Resources](#)
- [Twelve Days of Visualizations Zip File](#)

Note: These community examples are not supported by Pentaho Customer Support.