



Customizing Pentaho Business Analytics



This document is copyright © 2012 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

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

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

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

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

Limits of Liability and Disclaimer of Warranty

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

Pentaho (TM) and the Pentaho logo are registered trademarks of Pentaho Corporation. All other trademarks are the property of their respective owners. Trademarked names may appear throughout this document. Rather than list the names and entities that own the trademarks or insert a trademark symbol with each mention of the trademarked name, Pentaho 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.

Company Information

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

E-mail: communityconnection@pentaho.com

Sales Inquiries: sales@pentaho.com

Documentation Suggestions: documentation@pentaho.com

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

Contents

Introduction.....	4
Pentaho User Console Styling.....	5
Overview of Customizable Configuration Files.....	5
Replacing the Default Icon Set.....	6
Changing the Default Logo.....	6
Changing the Workspace Background.....	6
Customizing and Developing User Console Themes.....	6
Theme Engine Overview.....	7
Creating a New Theme.....	7
Setting the Default Theme.....	8
Switching PUC Themes.....	8
Removing or Changing Support Messages.....	8
Removing the Support Warning Message From JPivot.....	8
Removing the Support Warning Message From WAQR.....	9
Switching to a Production Login Screen.....	9
Testing Your Design Changes.....	10
Using the PUC UI Widget Test Page.....	10
Using Firebug to Test CSS Changes.....	10
Localization.....	12
BA Server and Thin Client Message Bundles.....	12
Pentaho User Console and Dashboard Designer Message Bundles.....	12
Pentaho Analyzer Localization.....	13
Pentaho Interactive Reporting Localization.....	14
Geographic Service (pentaho-geo) Localization.....	14
Enterprise Console Message Bundles.....	15
Enterprise Console Character Set Encoding.....	15
Action Sequence Message Bundles.....	15
Report Viewer Localization.....	15
Design Tool Localization.....	16
Using the Kettle Translator.....	17
Creating Design Tool Templates.....	19
Creating a Dashboard Designer Template.....	19
Reporting Templates.....	19
Report Design Wizard Template Design Guidelines.....	20
Interactive Reporting Template Design Guidelines.....	20
Dynamic Element Positioning in Templates.....	21
Template Properties.....	22
Deploying a Template to Report Design Wizard.....	23
Deploying a Template to Interactive Reporting.....	23
Setting the Default Interactive Reporting Template.....	23
Extending Pentaho Analyzer with Custom Visualizations.....	24
Step 1: Create a Pentaho BA Server Plugin.....	24
Step 2: Define your Custom Visualization.....	24
Step 3: Register the Visualization with Pentaho Visualization API.....	25
Step 4: Register the Visualization with Pentaho Analyzer.....	27
Step 5: Register the Created Javascript Files with Pentaho Analyzer.....	28
Step 6: Restart the BA Server and Test your Visualization!.....	29
Additional Resources.....	29

Introduction

This document provides guidance and instructions for customizing the user interfaces in Pentaho Business Analytics. In order to be successful with this document, you must have some level of experience in localization, or Web development and/or design; or you must have access to consultants who are able to perform these operations competently.



Note: Unless specifically stated otherwise, custom source code changes to Business Analytics are not covered under the standard Pentaho customer support agreement.

Pentaho User Console Styling

The Pentaho User Console is the standard Web interface for the Pentaho BA Server, and includes interactive elements of Pentaho Reporting and Analysis. The Pentaho Enterprise Console is the central framework through which your BA Server is configured and managed. You can customize the look and feel of the User Console by editing its configuration files, graphics, and CSS3 style sheets manually; and both the User Console and Enterprise Console can be translated into other languages by replacing plain-text message bundles.

Overview of Customizable Configuration Files



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

MantleSettings.properties

Location: /pentaho/server/biserver-ee/tomcat/webapps/pentaho/WEB-INF/lib/mantle-4.5.0-GA.jar

This file is stored in the above-listed JAR file in the Pentaho WAR. You must open the JAR with a ZIP archive utility, then navigate to the `/org/pentaho/mantle/server/` directory inside of it. No matter how you decide to extract and edit the file, it must be put back in the JAR when you are done. Pentaho recommends using **7-Zip** for this process because it can go into the JAR, launch your preferred editing program, then do an in-place update of the archive.

The MantleSettings.properties file contains settings for the menubar, main toolbar, and the logo panel. If you decide to turn off the menubar or toolbar, the logo panel will also be turned off because of UI constraints. Look for settings in the file such as **show-menu-bar=true**. To toggle the behavior, simply change the **true** to **false**.

MantleStyle.css

Location: /pentaho/server/biserver-ee/tomcat/webapps/pentaho/mantle/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.



Note: Modifying these styles could have dramatic impact on Pentaho User Console rendering. This is probably not the style sheet you are looking for.

Theme-specific local and global style sheets for PUC



Note: The Slate theme is included in the BA Server, but disabled by default because it has rendering issues in Internet Explorer. However, you may find it useful for development and testing purposes.

Location: /pentaho-solutions/system/common-ui/resources/themes/onyx/globalOnyx.css

This is the main structural, theme-specific style sheet for the default theme for the Pentaho User Console (Onyx).

Location: /pentaho-solutions/system/common-ui/resources/themes/onyx/globalSlate.css

This is the main structural, theme-specific style sheet for an alternate theme for the Pentaho User Console (Slate).

Location: /pentaho/mantle/themes/onyx/mantleOnyx.css

This is the customizable presentation portion of the theme style sheet for the default theme for the Pentaho User Console (Onyx).

Location: /pentaho/mantle/themes/slate/mantleSlate.css

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 BA Server client tool.

Analyzer location: /biserver-ee/pentaho-solutions/system/analyzer/styles/themes/

Interactive Reporting location: `/biserver-ee/pentaho-solutions/system/pentaho-interactive-reporting/resources/web/themes/`

Dashboard Designer location: `/biserver-ee/pentaho-solutions/system/dashboards/themes/`

Replacing the Default Icon Set

The icons used throughout the Pentaho User Console are consolidated and stored in PNG files called **image bundles**. These files only encompass a single HTTP request from the client to the server for the entire set of images, thereby saving bandwidth and improving performance. The image bundles all have machine-generated filenames with PNG extensions, and are stored in the `/pentaho/server/biserver-ee/tomcat/webapps/pentaho/mantle` directory. An example of a likely image bundle name is: **EAC2E33CD4B780B5A88A44308412F1CB.cache.png**.

If you edit an image bundle, you must be sure to preserve its location and size. The Pentaho Web application uses CSS to chop the image bundle up during runtime, so you must respect the size, position, and location of what is currently there. Because of these constraints, Pentaho recommends that you do not alter the images themselves, but instead shift the color palette to fit your brand specifications. This can be done with most modern graphic design tools.

Changing the Default Logo

The Pentaho logo displayed in the top right corner of the Pentaho User Console may be replaced or disabled to suit your needs. To turn off the logo, refer to the **MantleSettings.properties** explanation above.

The default logo is: `/pentaho/server/biserver-ee/tomcat/webapps/pentaho/mantle/themes/onyx/images/logo.png` (change the name of the theme from **onyx** to **slate**, or whatever your custom theme is, if you aren't using Onyx). The default Pentaho logo has a transparent background, so you will see through it to the background, whatever it may be. If you want to modify or replace the logo, you must maintain the file name and image size, which is **152x75** pixels.

If you must use a different size graphic for the logo, you can change the **.puc-logo-spacer** class in **mantleOnyx.css** (or the Slate equivalent, if that's what you're using) to use different dimensions:

```
.puc-logo-spacer {
. width: 152px;
. height: 75px;
}
```

Changing the logo link

You can make the logo go to a given URL when clicked. In its default state, the Pentaho logo doesn't go to a URL, but the functionality to make it clickable is in place. You can set the destination URL via the **logoPanelWebsite** entry in the `/pentaho/mantle/messages/mantleMessages.properties` file. This properties file is inside of the Pentaho WAR, so you will have to stop the BA Server before making any modifications.

Changing the Workspace Background

You can change the background image that appears in the content pane in the Pentaho User Console by modifying or replacing this file: `/pentaho/mantle/style/core/images/bg_pentaho_default.png`. The dimensions and filename must be identical.

Customizing and Developing User Console Themes

The Pentaho User Console's graphical interface is built on an innovative CSS3-based theme engine. The sections below contain advice for theme designers and developers.

Theme Engine Overview

Starting with BA Server version 4.5, this new CSS3-based theme system enables you can change the look of the BA Server and its client tools by adding your own themes. This is easily 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. Collectively these styles and scripts comprise the default system theme, **Onyx**. This theme is located in the Common UI plugin directory: `/pentaho/server/biserver-ee/pentaho-solutions/system/common-ui/themes/`.

There are two 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, Onyx. A change to the Onyx system theme will change the way buttons look in all applications. Local themes are defined for a particular area or "context" of the BI Server. Contexts include BA Server plugins 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 BI 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">
  <autumn display-name="Autumn" system="true">
    <file>autumnStyles.css</file>
    <file>autumnScripts.js</file>
  </autumn>
</themes>
```

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

```
<script type="text/javascript" src="/pentaho/common-ui/themes/autumn/
autumnStyles.js"></script>
<link rel="stylesheet" type="text/css" href="/pentaho/common-ui/themes/autumn/
autumnStyles.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 plugin or root WAR folder:

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

Creating a New Theme

On startup, the BA Server searches for **themes.xml** files in every plugin 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 Autumn.

```
<themes root-folder="resources/themes">
  <autumn display-name="Autumn" system="true">
    <file>autumnStyles.css</file>
    <file>autumnScripts.js</file>
  </autumn>
  <autumn display-name="Autumn" system="true">
    <file>localAutumnStyles.css</file>
  </autumn>
</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/`.

Plugin resource loading is different than WAR-based loading in that the plugin controls how resources are mapped to the URL. If the theme file from above was located in a plugin 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 plugins through **static-path** entries:

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

It is quickest and easiest to develop a new theme by basing it on the default Onyx theme. Every style defined in Onyx's **mainStyles.css** file is used somewhere in the BA Server. Onyx also contains several script files to support jQuery's JScrollPane scrollbars. These scripts replace the native browser scrollbars with customizable DHTML versions.

Setting 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 **onyx** as the default theme; changing the value to another theme name will set the default active theme for all User Console users.

Switching PUC Themes

If you've 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 isn't found, the current active theme's resources will be loaded instead. This is particularly useful when testing out new themes, and for loading debug versions of scripts and styles.

It's 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.

Removing or Changing Support Messages

The information below explains how to remove the warning messages at the bottom of the ad hoc Reporting and JPivot client tools.

Removing the Support Warning Message From JPivot

Follow this process to remove the following message from the bottom of the JPivot screen: **"JPivot has been replaced by Pentaho Analyzer. It is provided as a convenience but will no longer be enhanced or officially supported by Pentaho."**

1. Shut down the BA Server if it is currently running.
2. Edit the `/pentaho/server/biserver-ee/tomcat/webapps/pentaho/jsp/pivot.jsp` file.
3. Search for **deprecatedWarning** to locate the warning message entry.
4. Put SGML comment tags around the deprecatedWarning div tag as shown in the example below.
5. Save and close the file.
6. Start the BA Server.

The warning message will no longer appear at the bottom of the JPivot screen.

```

<!-- <div id="deprecatedWarning" style="margin: auto; width: 100%">
  <table width="580px" align="center" style="background-color: #fffdd5;
  border-style: solid; border-color: #dcb114; border-width= 1px; font:
  normal .85em Tahoma, 'Trebuchet MS', Arial">
    <tr>
      <td>
        
      </td>
      <td>
        JPivot has been replaced by Pentaho Analyzer.<br/>
        It is provided as a convenience but will no longer be enhanced
        or officially supported by Pentaho.
      </td>
    </tr>
  </table>
</div> -->

```

Removing the Support Warning Message From WAQR

Follow this process to remove the following message from the bottom of the ad hoc Reporting screen: **"Web Ad Hoc Query and Reporting has been replaced by the new Interactive Reporting client. It is provided as a convenience but will no longer be enhanced or officially supported by Pentaho."**

1. Shut down the BA Server if it is currently running.
2. Edit the `/pentaho/server/biserver-ee/tomcat/webapps/pentaho/adhoc/waqr.html` file.
3. Search for **waqrDeprecatedAlert** to locate the warning message entry.
4. Put SGML comment tags around the `waqrDeprecatedAlert` td tag as shown in the example below.
5. Save and close the file.
6. Start the BA Server.

The warning message will no longer appear at the bottom of the ad hoc Reporting screen.

```

<!-- <td valign="middle" id="waqrDeprecatedAlert" style="padding-right:
5px">
  <table width="580px" style="background-color: #fffdd5;
  border-style: solid; border-color: #dcb114; border-width= 1px; font:
  normal .85em Tahoma, 'Trebuchet MS', Arial">
    <tr>
      <td>
        
      </td>
      <td>
        Web Ad Hoc Query and Reporting has been replaced by the new
        Interactive Reporting client.<br/>
        It is provided as a convenience but will no longer be
        enhanced or officially supported by Pentaho.
      </td>
    </tr>
  </table>
</td> -->

```

Switching to a Production Login Screen

The default Pentaho User Console login screen contains information and instructions for evaluators. Pentaho allows this extra information to be hidden via configuration; follow the directions below to hide the extra information.

1. Shut down the BA Server and Enterprise Console if they are currently running.
2. Open a terminal or file browser window and navigate to the `/pentaho-solutions/system/` directory.

If you installed via the archive package or the graphical installer, the full path is `/pentaho/server/biserver-ee/pentaho-solutions/system/`.

3. Edit the **pentaho.xml** file.
4. Find the **login-show-sample-users-hint** node and change its value to **false**.
5. Start the BA Server and Pentaho Enterprise Console.

You now have a login screen that has been scrubbed of evaluation content.

Testing Your Design Changes

The information in this section assists developers and designers in testing changes to themes, graphics, and other custom modifications to the BA Server.

Using the PUC UI Widget Test Page

Pentaho provides a hidden test page to assist theme designers and interface developers with testing (change the hostname, port number, and context to match your deployment):

```
http://localhost:8080/pentaho/content/common-ui/resources/web/test/style-samples.html
```

From this page, you can safely click on all of the buttons and dialogues that are included in your themes without the overhead of logging into the Pentaho User Console and navigating through various content pages and client tools to get to each widget.

Using Firebug to Test CSS Changes

The style sheet files explained earlier in this guide 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, Pentaho recommends that you use the Firebug extension 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.

2. Hover mouse over desired UI elements and then click

1. Click Inspect

3. Notice the Style tab and the .launchImage CSS class

4. You may edit existing properties or add new ones and see their effect immediately

Inspect Edit img#launch...aunchImage <td <tr <tbody <table.laun...uttonPanel <td <tr <tbody <table.launchPanel <div <div <

Console HTML CSS Script DOM Net YSlow Options Style Layout DOM Options

```

<td align="center" style="vertical-align: middle;">
  <table class="launchButtonPanel">
    <colgroup>
      <tbody>
        <tr>
          <td align="center">
            <td align="center">
          </td>
        </tr>
      </tbody>
    </colgroup>
  </table>

```

.launchImage {
 cursor: pointer;
}

MantleStyle.css (line 13)

- Copy
- Edit Element Style...
- New Property...
- Edit "cursor"...
- Delete "cursor"
- Disable "cursor"
- Refresh
- Inspect in DOM Tab

MantleStyle.css (line 305)


onPanel

MantleStyle.css (line 323)

MantleStyle.css (line 323)

Find: Next Previous Highlight all Match case

Done YSlow 1.236s

 **Note:** Both Chrome and Safari come with built-in developer tools that may be just as effective as Firebug for this purpose.

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



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.

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

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 and JPivot require a special set of instructions for localizing their analysis schema message bundles. See [Localization/Internationalization of Analysis Schemas](#) on page 13 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.

Switching PUC Message Bundles

If you've created a localized message bundle, you can switch to it through the **View** menu in the Pentaho 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 ROLAP data sources you use with Analyzer. For schema localization, refer to [Localization/Internationalization of Analysis Schemas](#) on page 13.

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 guide.

```
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

You can create internationalized message bundles for your analysis schemas and deploy them with the Pentaho Web application. This enables Pentaho Analyzer and JPivot 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}"/>
```

2. Stop the BA Server.
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.rolap.localePropFile=com.pentaho.messages.MondrianMessages
```

4. Save and close the file.
5. Edit your /pentaho/server/biserver-ee/pentaho-solutions/system/olap/datasources.xml file and modify the **Provider** line to declare the default locale, include the i18n dynamic schema processor (this enables

message token replacement), and refresh the schema when Analyzer detects that the checksum is different than the cached version:

```
Provider=mondrian; Locale=en_US; DynamicSchemaProcessor=
mondrian.i18n.LocalizingDynamicSchemaProcessor; UseContentChecksum=true; Jdbc=
jdbc:odbc:MondrianFoodMart; Catalog= /WEB-INF/FoodMart.xml
```

6. Save and close the file.
7. 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
```

8. Start the BA Server.

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.

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 guide.

```
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-4.5.0-GA.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.

Enterprise Console Message Bundles

The Pentaho Enterprise Console message bundles are stored in the `/pentaho/server/enterprise-console/www/com.pentaho.pac.PentahoProAdminConsole/messages/` directory. There are two files that you must modify to localize the Enterprise Console:

- `messages.properties`
- `promessages.properties`



Note: The same language and locale two-letter naming conventions apply to `promessages.properties`.

Enterprise Console Character Set Encoding

You can use your own character set encoding by editing the **console.properties** file located under `...\enterprise-console\resource\config`. The property name is **content.charset** and it is currently set to **UTF-8**. If you have the platform set to any other character encoding, you must change this value for the Pentaho Enterprise Console to work correctly.

In addition to editing the `console.properties` file, you must pass a JVM parameter in the start script, (`start-pec.bat / .sh`), of the Pentaho Enterprise Console, (`-Dfile.encoding="UTF-8"`). This must match the platform's character encoding.

Action Sequence Message Bundles

The message bundle for action sequence error messages is `/pentaho/WEB-INF/classes/org/pentaho/platform/engine/services/messages/messages.properties`.

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](#) on page 14 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 guide.

```
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-4.5-GA.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 guide.

```
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 [BA Server and Thin Client Message Bundles](#) on page 12.

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 [Using the Kettle Translator](#) on page 17 for more information.

Design Studio

Since Design Studio is a single JAR that acts as an Eclipse plugin, this process is different. Unpack the following archive, localize its message bundles, then repack it (replace **version** with the actual version number, which varies with each Business Analytics release):

```
/design-studio/plugins/org.pentaho.designstudio.editors.actionsequence_version/
pentaho-actionsequence-plugin.jar
```


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](#) on page 12. 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/

Using 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](#) on page 16. 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: <svn://source.pentaho.org/svnkettleroot/>.
 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**.

Creating 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.

Creating 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.

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

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.

layout-mode	value
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.

Band	Report Design Wizard	Interactive Reporting
		3: Edit any message elements within the band
page header	None	Allows the modification of message elements

Deploying 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.

Deploying 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.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.

Setting 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>
```

Extending Pentaho Analyzer with Custom Visualizations

This document describes step by step how to integrate third-party visualizations into Pentaho Analyzer. To get started, you'll need a BA Server 4.5.0 GA or later installed, along with Pentaho Analyzer installed as a plugin to the BA Server. This API is new to Pentaho as of the 4.5.0 GA release, and is subject to change in future releases as we continue to improve on it.

Step 1: Create a Pentaho BA Server Plugin

To begin, you'll need to create a location for your new visualization within the BA Server. Start by creating a new folder in the `pentaho-solutions/system` directory. The plugin sample in this guide is named "example-visualization".

Within the new folder, create a `plugin.xml` file. This file will contain metadata about the plugin. Here is an example of what the `plugin.xml` should look like:

```
<?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>
```

At this point, we've defined a very basic plugin that exposes any static files in the resources folder of the plugin to authenticated users in the Pentaho BA Server.

Step 2: Define your Custom Visualization

Visualizations in Pentaho Analyzer are rendered on the client, and are Javascript based. You'll need to define a constructor and two key functions in your visualization Javascript class in order for Pentaho Analyzer to render it properly. The constructor of your 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(datView, vizOptions)**. This function is called when rendering the visualization. The **datView** object contains a multi-dimensional result set returned from the server, and the **vizOptions** contain any customized options specified by the user when configuring the visualization.

For this example, we'll create a very basic KPI visualization. Create a file in the `example-visualization/resources` folder called `example.js`, and 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 a calc value,
  // 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:
  }

  // write the KPI value to the screen
  this.numSpan.innerHTML = value;
  this.resize();
}

```

We now have a basic visualization defined. This is the entry point where a more advanced visualization may be rendered, for instance if you have a Flash component, HTML5, or SVG visualization library, you would make calls to those elements here and wire them into the innerHTML of the canvas element.

For a more advanced example, please see how the Community Chart Components implement their 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.

Step 3: Register the Visualization with Pentaho Visualization API

Now that we've defined the visualization, we need to register the visualization with Pentaho's Visualization API. At this time, the Visualization API is only accessible from within Pentaho Analyzer, but in the future this Visualization API may be used in other locations as well, such as Pentaho Interactive Reporting and Pentaho Dashboards.

To register with the Visualization API, include the following Javascript at the beginning of the `example.js` file:

```

// utilize the require API for verifying that the VizController has been loaded
// before we register
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
  },
  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'
            }
          }
        ]
    }
  ]
});

```

You'll also need to define the following at the end of the `example.js` file to enclose the require API call:

```
});
```

All values defined above are essential to registering the visualization with Pentaho's Visualization API. The **dataReqs** subcomponent is especially critical, it defines the data requirements for the visualization, which is used by Pentaho Analyzer to determine which fields are displayed in the field list.

Step 4: Register the Visualization with Pentaho Analyzer

Now that we have registered the component with the Visualization API, we also 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 is used to set visualization options based on the Analyzer
         * report definition or the current state of the report editor/viewer.
         * This is called right before the visualization is drawn. The returned
         * options are not saved as part of the visualization state when saving
         * an Analyzer report showing this visualization.
         *
         * @return a hash object containing the custom state of your
         *         visualization.
         */
        generateOptionsFromAnalyzerState:function (report) {
          return {}; // perform no work
        }
      };

      /*
       * 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) {
          // This component has a single argument, so we assume if this event is
          // fired it is for the aggregate
          this.report.visualization.args['aggregate'] =
            config.byId('aggregate').value;
          this.inherited(arguments); // Let super class handle the insertAt and

```

```

// removedGem events
},
/**
 * Return the JSON configuration object which the panel will use to
 * create the UI and its model
 */
getConfiguration: function() {
  var config = this.inherited(arguments);

  dojo.forEach(config.properties, function (item) {
    if (this.report.visualization.args[item.id] !== 'undefined') {
      item.value = this.report.visualization.args[item.id];
    }
  }, this);
  return config;
});

// 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, we 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 metadata related to the visualization, along with a function **generateOptionsFromAnalyzerState**, which is used by Analyzer for serializing and deserializing the state of the visualization.

The Layout Panel in Analyzer is generated dynamically from the JSON data requirements definition of the visualization, which we defined in step 3. In order for your visualization to respond to changes made in the panel, we registered a **ConfigurationManager** with Analyzer.

As seen in the above example, Configuration Managers need to contain the following functions:

getConfiguration(): return the JSON data requirements definition from the visualization. This is where you can customize the definition based on current state (enable/disable items, etc)

updateConfiguration(config): this is called after the panel has parsed the definition. The realized Configuration is passed in allowing your manager perform more processing after the fact.

checkAcceptance(source, nodes, silent): Called by Drag and Drop operations in the panel. Return true if the operation is allowed.

onModelEvent(config, item, eventName, args): All user interactions result in **onModelEvent** call to the active Configuration Manager. This is where the visualization should be updated directory to reflect changes in configuration.

Step 5: Register the Created Javascript Files with Pentaho Analyzer

In this step, we'll notify Pentaho Analyzer about our javascript files that we've created. By adding the following child element of **plugin** to our `plugin.xml` file defined earlier, our javascript files will be included in Pentaho Analyzer.

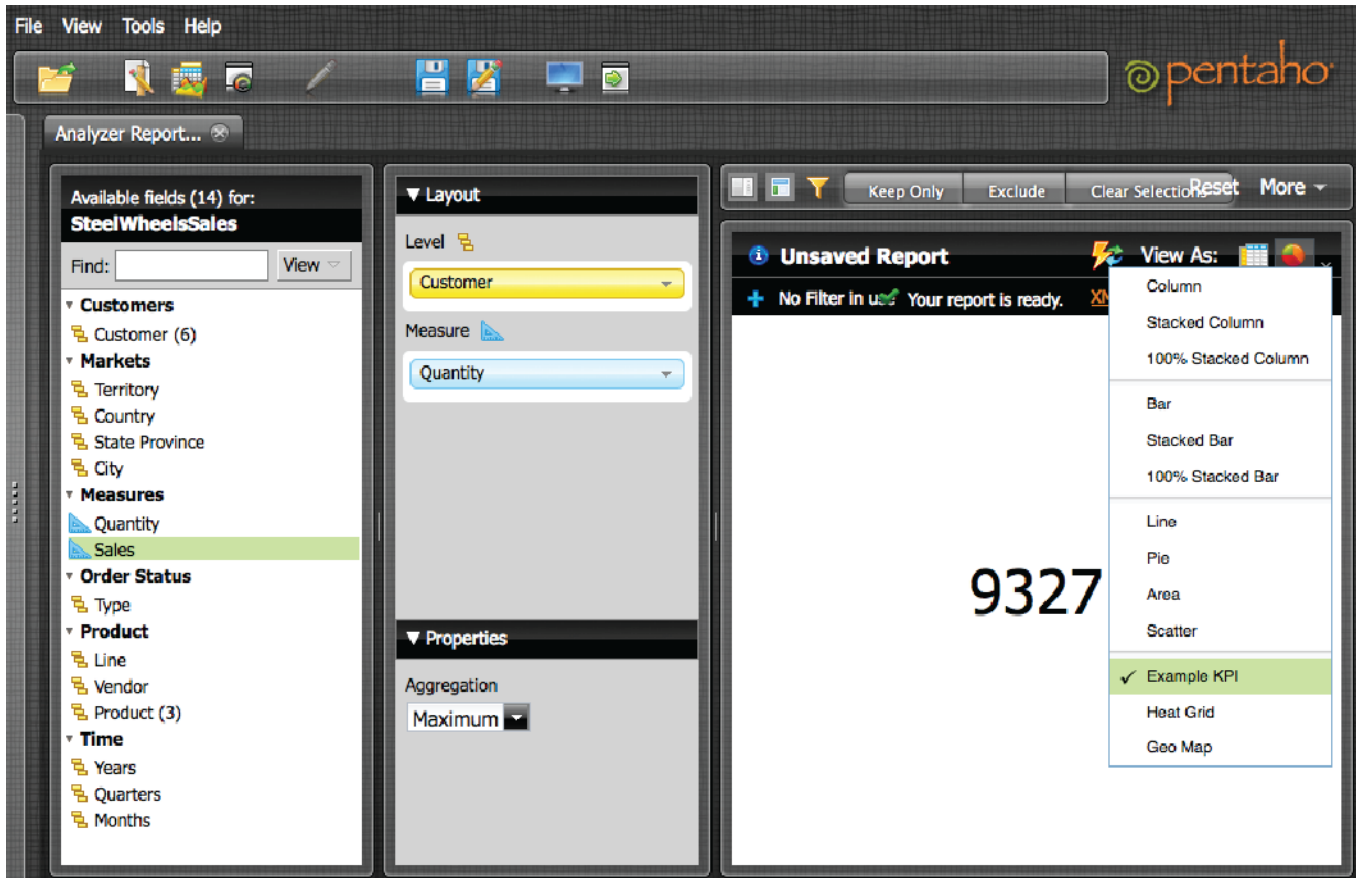
```

<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>

```

Step 6: Restart the BA Server and Test your Visualization!

Finally, you'll want to restart the BA Server. Once you've restarted your server, you should see the visualization **Example KPI** registered in the list of Visualizations when creating an Analyzer report. Go ahead and create a new report with the custom visualization selected. You can save the report and re-open it, and the visualization settings are remembered. Note that you should test your visualization in the various browsers you expect the users of your BA Server environment to use.



This step-by-step tutorial walked you through implementing a basic visualization in Pentaho Analyzer. There are more complex examples included in the Pentaho BA Server noted in Step 2. These examples go beyond this tutorial, including functionality such as server side printing, lasso'ing and selections.

Additional Resources

Providing Javascript Plugins via External Resources:

<http://wiki.pentaho.com/display/ServerDoc2x/Providing+Javascript+Plugins+via+External+Resources>