Creating Interactive Dashboard with Pentaho AJAX

James, Chief Geek, Pentaho

Last Modified on 12/27/06

Contents

Introduction

This document describes how to create browser-side interactive dashboards using the AJAX library of the Pentaho Open BI Suite. It uses the Google Maps Dashboard sample provided with the Pentaho Pre-Configured Installation (PCI) as illustration.
Before getting into the technical details we will briefly explain what AJAX is and walk through our Google Maps Dashboard sample which uses the Pentaho AJAX components to create an interactive location intelligence dashboard.

AJAX

So what, exactly, is AJAX all about? AJAX stands for 'Asynchronous Javascript and XML'. So what, exactly, does that mean? AJAX is not a product: it is a collection of technologies that can be used to build web pages that are more interactive than has been possible in the past (without resorting to plug-ins, applets, or active/x components).

The individual pieces are:

- Asynchronous Javascript: This means that script within the web page can make calls to a server to get information without re-loading the web page. This significantly improves the usability and performance of complex web sites because the contents of the page can interact with the user without having to reload HTML, style-sheets, and images and without all that annoying scrollbar malarkey.
- XML: This refers to the fact that the result of the call to the server can be XML. Actually it can be any kind of text such as HTML or Javascript object notation (JSON) or just a response message.
- Dynamic HTML: This somehow got left out of the 'AJAX acronym'. This is much more important than the 'XML' part because it allows the web pages to by dynamically altered by the script based on its conversation with the server. This can be used to change images, add or remove rows to tables, hide and show tabs, and move or change text on the page. I guess AJAD didn't sound cool enough.

Web pages built with AJAX can be made to feel more like an application than a web page. Comparing Google Maps with the older web-based mapping engines provides a good example of the differences.

Pentaho AJAX

Using Pentaho's AJAX library from within a web page you can dynamically command the BI components of the Pentaho BI platform to generate charts, dials, and tables to be display on the page. You can also perform other tasks such as initiating a data transformation or report delivery process. All of these are achieved by asking Pentaho to execute an Action Sequence. An Action Sequence is a set of instructions that defines which components to use and how to pass data between them. For more information about creating Action Sequences see the Creating Pentaho Solutions guide.

Audience

This document starts out fairly light and fluffy but gets progressively more technical as it goes on. If you are not a Java developer with XML experience you will probably lose interest after about 10 pages.

Pentaho Google Maps Dashboard

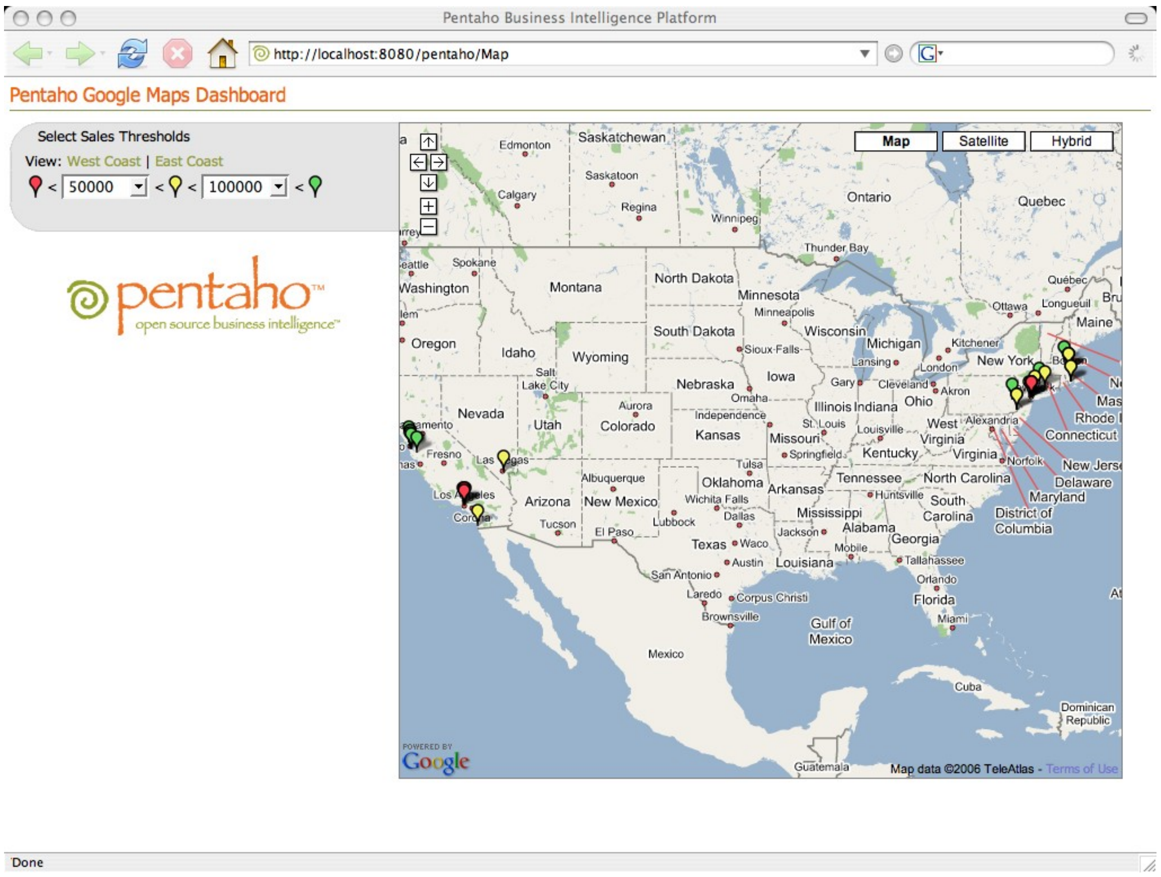
The 'Pentaho Google Maps Dashboard' is a sample dashboard that Pentaho provides to show how our AJAX components can be used to create an interactive web page.

Installation

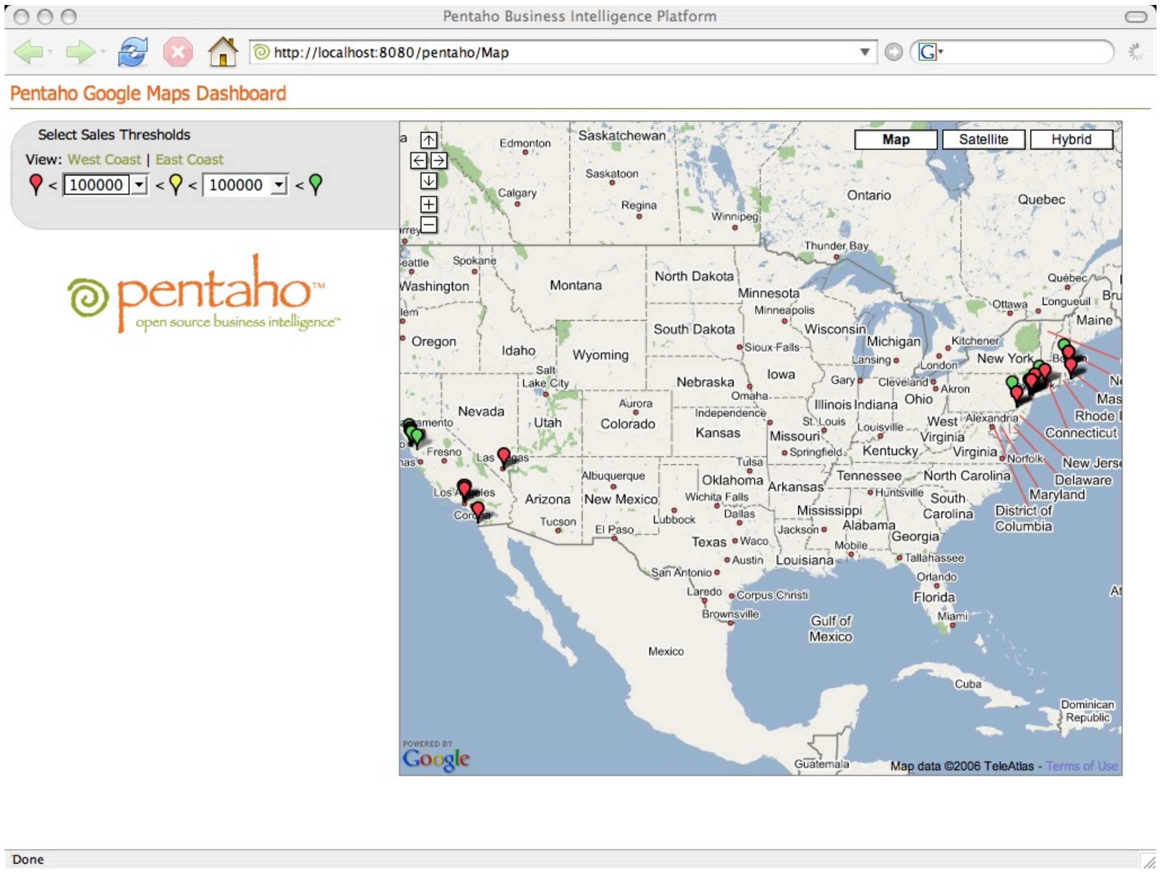
The Pentaho Google Maps Dashboard sample is provided in the Pentaho Pre-Configured Installation (PCI). You can downloaded this from Pentaho (http://www.pentaho.org/download/). You will need the 1.2 GA version. The examples used in this guide assume that you have installed the PCI using the default port of 8080. If you have any problems installing or running the PCI server, please refer to the Getting Started Guide for assistance. We will begin our walk through by launching the Google Maps Dashboard sample using the following URL: http://localhost:8080/pentaho/Map

Walk Through

The Google Maps Dashboard sample displays a map showing the location of each customer using color coded pins to highlight their relative performance based on the Total Sales key performance indicator (KPI). You can immediately see that the customers are grouped into two areas of the country. This is something that might not have been immediately obvious from looking at the customer records in the database.


Pentaho Business Intelligence Platform
http://localhost:8080/pentaho/Map
Pentaho Google Maps Dashboard
Select Sales Thresholds
View: West Coast | East Coast
< 50000 < < 100000 <
pentaho open source business intelligence
Map
Satellite
Hybrid
Map data ©2006 TeleAtlas - Terms of Use
Done

For convenience, the dashboard provides links allowing users to quickly zoom in and center on either the West Coast or East Coast. Thresholds for the Total Sales KPI used to color code the pins can also be modified on the fly using simple drop down combo boxes. For example raising the lower threshold to 100,000 changes all the pins that were yellow to red. This is done using the Google Maps API and when the user performs any of these action the page is not reloaded from the server.

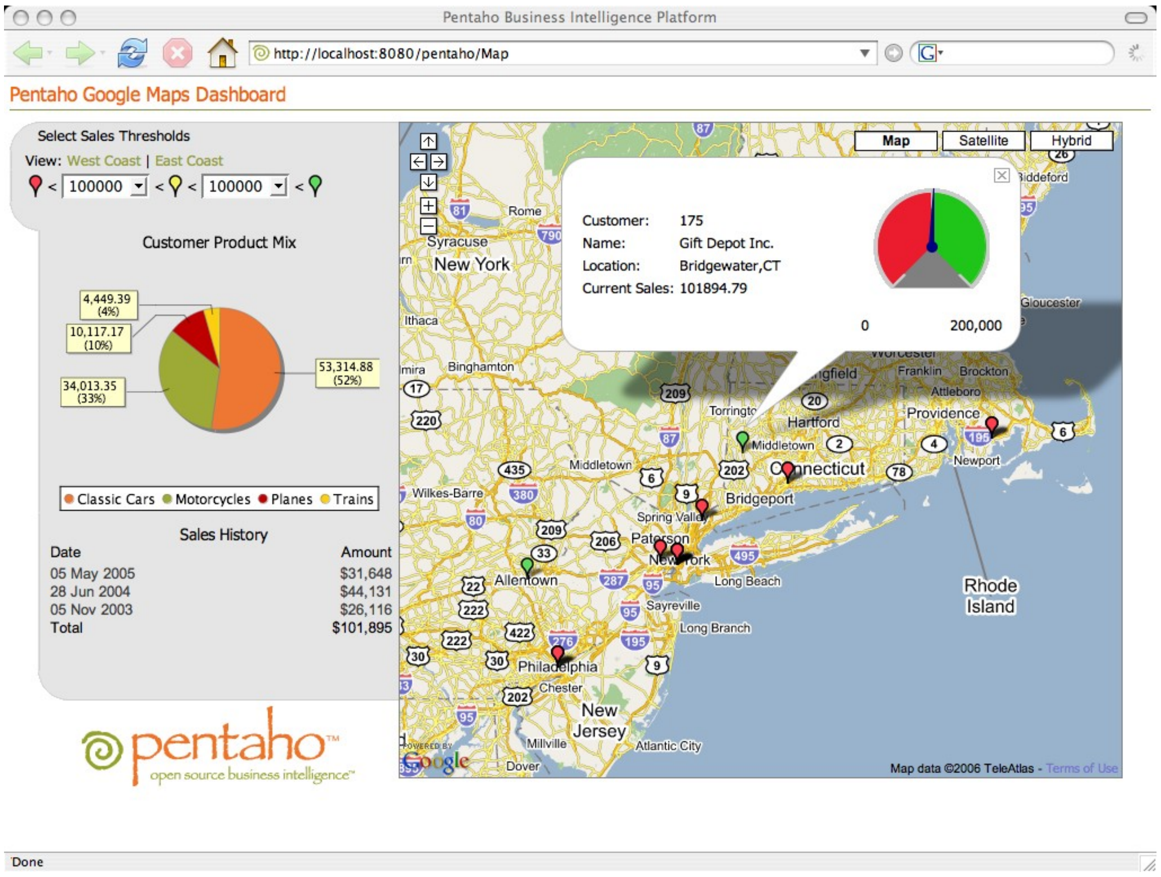


Pentaho Business Intelligence Platform
http://localhost:8080/pentaho/Map
Pentaho Google Maps Dashboard
Select Sales Thresholds
View: West Coast | East Coast
< 100000 < < 100000 <
pentaho open source business intelligence
Map
Satellite
Hybrid
Map data ©2006 TeleAtlas - Terms of Use
Done

For details on how the pins are re-colored see ‘Adding and removing Google Map Markers’ below.
For details on how to re-center and zoom a Google map see 'Centering and Zooming Google Maps' below.

When a user clicks on in pin on the map:

1. A Google Maps info-window displays basic information about the customer (customer number, name, location, and sales).
2. The side-bar displays a pie chart that shows the mix of products that the customer has bought
3. The side-bar displays a table of the customer's sales history.


Pentaho Business Intelligence Platform
http://localhost:8080/pentaho/Map
Map
Satellite
Hybrid
4,449.39 (4%)
10,117.17 (10%)
34,013.35 (33%)
53,314.88 (52%)
0
200,000
Map data ©2006 TeleAtlas - Terms of Use
Done

Pentaho Google Maps Dashboard

Select Sales Thresholds

View: West Coast | East Coast

< 100000 < 100000 <

Customer Product Mix

● Classic Cars ● Motorcycles ● Planes ● Trains

Sales History

Date	Amount
05 May 2005	$31,648
28 Jun 2004	$44,131
05 Nov 2003	$26,116
Total	$101,895

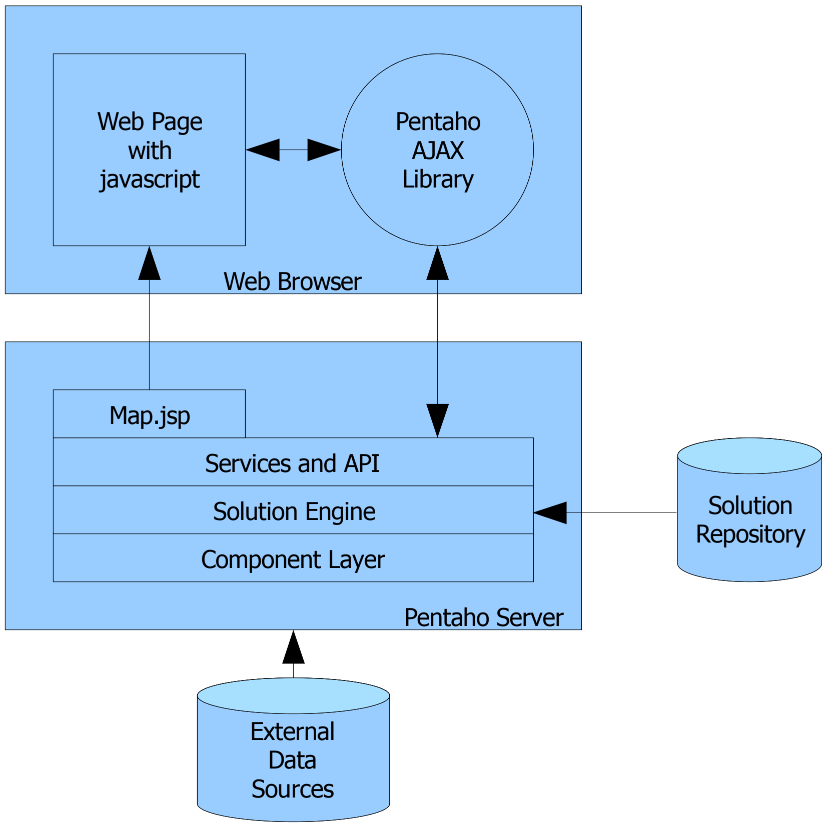
Customer: 175
Name: Gift Depot Inc.
Location: Bridgewater,CT
Current Sales: 101894.79

If the user changes the sales thresholds the dial in the info-window is automatically refreshed to show the new thresholds. If the user clicks on a different marker the info-window moves to that marker and refreshes itself with the information for the new customer and the pie chart and sales history tables update.
The information displayed in the info-window, the pie chart, and the sales history table are all dynamically generated by the Pentaho server and are retrieved and inserted into the page using the Pentaho AJAX components and libraries.

For details on how the dial is defined see 'AJAX Dial ' below.
For details on how the pie chart is defined see 'AJAX Pie Chart' below.
For details on how the table is defined see 'AJAX Embedded Report' below.

Architecture

The Pentaho Google Maps Dashboard uses a Java Server Page (JSP) that uses the Pentaho Java API, web-based services, and AJAX library to generate the images and the tables on the page.

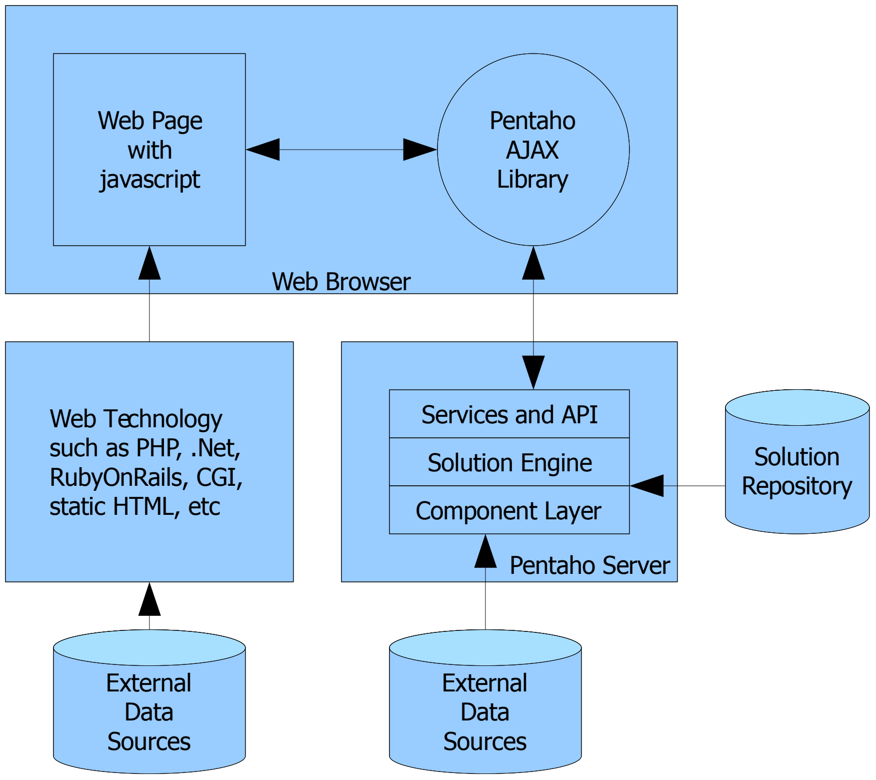


The web page is initially loaded by the browser from a JSP in the Pentaho server.

Once the page has been loaded it is not reloaded again. All the additional communication that is required to provide the interactivity is done through the Pentaho AJAX API.

For more information about the Pentaho architecture please see the Pentaho Technical Whitepaper.

Although this sample uses a Java Server Page (JSP) to create the web page, any web technology such as PHP, .NET, Ruby, or CGI could be used to create the initial page.



Using an architecture like this avoids the necessity of a server-side integration between different technologies and allows the integration and interaction to occur inside the web page. For example if you have a web-based customer management (CRM) system that was created using PHP you can create the HTML and Javascript that displays and controls the Pentaho AJAX components using PHP. This enables you to embed the Pentaho components into the pages of your CRM system by modifying the UI templates or PHP pages. You would not have to modify the CRM system to integrate with the Penaho server, e.g. using a PHP-Java bridge or use a web services approach.

Pentaho AJAX API

The Pentaho AJAX API is very simple and allows you to execute action sequences. Action sequences are documents within a Pentaho Solution Repository that execute one or more components. Action sequences allow you do many things including:

- Create charts and dials
- Create reports
- Create reports and email them
- Execute data transformations
- Execute relational (SQL) queries
- Execute multi-dimensional (MDX) queries

For more information about action Sequences see the 'Creating Pentaho Solutions' Guide.

pentahoAction

There is a single Javascript function (in pentaho-ajax.js) that executes an action sequence on the server and returns the response.

```
function pentahoAction( solution, path, action, parameters, callbackFunction );
```

solution: The name of the solution that the action sequence to be executed is in
path: The path within the solution to the action sequence to be executed
action: The name of the action sequence to be executed
parameters: Parameters to pass to the action sequence. This is a two dimensional array. The inner arrays always have two elements: the first is the name of the parameter, the second is the value of the parameter.
callbackFunction: The Javascript function the Pentaho AJAX API should call when the content is returned from the server

Example

```
pentahoAction( "samples", "google", "dial.xaction",
     new Array(
          new Array( "customer", 1234 ),
          new Array( "value", 145000 ),
          new Array( "max", 200000 ),
          new Array( "low", 100000 ),
          new Array( "high", 150000 )
     ),
     'updateInfoWindow'
);
```

This call to pentahoAction executes an action sequence called samples/google/dial.xaction and the results are passed to the Javascript function 'updateInfoWindow'. This action sequence uses its input parameters to create a dial image and return an HTML 'img' tag. The parameters passed to the action sequence are:
customer: the id of the customer - 1234.
value: the value to display on the dial - 145,000.
max: the maximum value of the dial - 200,000.
low: the lower threshold displayed on the dial - 100,000.

High: the upper threshold displayed on the dial - 150,000.

The parameters passed to the action sequence must match the parameters expected by the action sequence.
The script that makes this call must have a function called 'updateInfoWindow' defined. The function must receive a single parameter that will contain the content returned from the server.
e.g.

```
function updateInfoWindow( content ) {
     // do something with the content...
     // like use DHTML to embed the content into the web page

     // we can also execute another action sequence to get more dynamic content
     pentahoAction( "samples", "google", "chart.xaction",
          new Array( new Array( "customer", 1234 ) ),
               'updateProductMix'
          );
     }
}
```

Notice that the sample function above executes another action sequence. This enables you to update multiple areas of the web page in response to a single action by the user.

Google Maps Dashboard Explained

There are several activities involved in creating an AJAX dashboard using Google Maps:

1. Data Determination: Decide on the data that will be located on the map. This information could be anything that is geographically located such as customer, suppliers, vendors, competitors. Typically this information is determined by the requirements for the dashboard.
2. Geocoding: For each item to be located determine its latitude and longitude. This is called geocoding. Geocoding can be done different ways (see below).
3. Dashboard Generation: Display the items on a map and enable the user to pan and zoom the map.
4. Enable Interactivity: Add functionality defining what action take place when a user clicks on an item in the dashboard. You can also add features allowing the user to filter or sub-select the data.

Geocoding

Geocoding is the act of converting a location expressed in human terms, e.g. A country, city, or address, into a location expressed in latitude and longitude.
Geocoding can be done at different points during a process:

1. Data preparation. Using this technique the data is geocoded during a batch process. The latitude and longitude are stored (typically in a database).
   Each item is geocoded only once. If there are any items that cannot be geocoded data quality processes can be put in place to resolve the issues. This is typically the best solution for production systems. Since this requires server-side processes it is sometimes not the best solution for prototyping.

2. Server-side request. Using this technique the data is geocoded as it is requested from the server. This merges the geocoding and dashboard generation steps into a single step.
3. Client-side request. The Google Maps API includes a client-side geocoding library that can be used to do geocoding within the browser page.

Dashboard Generation

The sample dashboard is generated by a JSP. This JSP loads the customer data from the database and inserts Javascript functions into the page that create the marker pins on the map once the web page has finished loading. This JSP also defines the look and feel of the page and all the visual elements (title, threshold panel, customer details panel, Google Map panel).
The Map.jsp for the Google Maps Dashboard sample is found in:

pentaho-demo/jboss/server/default/deploy/pentaho.war/jsp/Map.jsp.

For a more detailed look at this JSP please refer to Appendix 1: Google Map JSP.

Interactivity

Adding and removing Google Map Markers

You can remove, add, replace markers displayed on the map using the map API. For example we replace some of the markers on the map when the user changes the thresholds.
You add a marker to the map using map.addOverlay
e.g.

```
geocoder.getLatLng( address,
     function(point) {
          var marker = new GMarker(point, icon);
          map.addOverlay(marker);
          GEvent.addListener(marker, "click", function() {
               infoWindow( marker, record );
          });
     }
);
```

You remove a marker from the map using map.removeOverlay. You need to tell the map which marker to remove so it is important to keep a reference to all of the markers that you add to the map.
e.g.

```
map.addOverlay(marker);
```

Clicking on Map Markers

Once you have added markers to the map you can add listeners that will tell you when the user clicks on them.

e.g.

```
// create a new marker with a click listener
var marker = new GMarker(point, icon);
map.addOverlay(marker);
GEvent.addListener(marker,
     "click",
     function() {
          infoWindow( marker, record );
     }
);
```

This example creates a 'click' listener for a marker and provides the listener with a Javascript function that calls infoWindow() when the user clicks on the marker.

Centering and Zooming Google Maps

You can center and zoom the map display using Javascript. You provide the map with the latitude and longitude of the center of the map and the zoom level to use.

e.g

```
map.setCenter( new GLatLng(41.4263, -73.1799 ), 7)
```

Map Search

A 'Map Search' is when the user pans or zooms the map display so it show an area that they are interested in and then requests information about known items within that space. This is possible because the Google Map API can be used to find out what ranges of latitudes and longitudes are currently displayed. In order to do a map search these latitude and longitude ranges must be sent to the server and used in a query to select items to display. Additional filters can of course be passed as well.
This Javascript shows how to get the boundaries of the currently displayed map.

```
// get the boundary of the currently displayed map
var bounds = map.getBounds();
// get the location of the top right corner
var topRight = bounds.getNorthEast();
// get the location of the bottom left corner
var bottomLeft = bounds.getSouthWest();
// get the top boundary (north latitude)
var northLatitude = topRight.lat();
// get the bottom boundary (south latitude)
var southLatitude = bottomLeft.lat();
// get the right boundary (east longitude)
var eastLongitude = topRight.lng();
// get the left boundary (west longitude)
var westLongitude = bottomLeft.lng();
// now send the 4 boundaries to the server to be used as a filter ...
```

Content Definition

AJAX Dial

This pie chart is defined in

pentaho-demo/pentaho-solutions/samples/google/dial.xaction

This action sequence is executed by the Pentaho server when the AJAX library calls it from the web page.

This action sequence creates the HTML content to display a dial in the dashboard.
To do this the action sequence takes as inputs all the data needed to create the dial:

- max: Maximum value of the range displayed on the dial
- low: The lower threshold to display on the dial (transition point between red and yellow areas)
- high: The upper threshold to display on the dial (transition point between yellow and green areas)
- value: The value to display on the dial

The action sequence has a single output that contains the HTML fragment that will display the dial.

The action sequence creates the HTML in three steps:

1. The Javascript component to creates a data set with one row of data:
   Inputs: max, value
   Action:
   ```
   data = new JavaScriptResultSet();
   data.setColumnHeaders( new Array( 'actual', 'min', 'max' ) );
   data.addRow( new Array( parseInt(value), 0, parseInt(max) ) );
   ```
   Outputs: data result set
2. The chart component to create the dial image:
   Inputs: data result set, low, high , max, image type (.png)
   Action: Creates a dial using the chart definition that is embedded in the component definition. The chart settings are not described in detail here. The dial is written out to a temporary file.
   Outputs: chart filename, base URL
3. Another Javascript component creates the final HTML fragment to return to the dashboard:
   Inputs: chart filename (CHARTOUTPUT), base URL (BASEURL)
   Action:
   ```
   chart_url = '<img src="' + BASEURL + "getImage?image=" + CHARTOUTPUT + '" />';
   ```
   Outputs: chart HTML (chart_url)

AJAX Pie Chart

This pie chart is defined in

pentaho-demo/pentaho-solutions/samples/google/chart.xaction

This action sequence is executed by the Pentaho server when the AJAX library calls it from the web page.

This action sequence creates the HTML content to display a pie chart in the dashboard.
To do this the action sequence takes as input:

- customer: The customer number for which the pie chart is needed

The action sequence has a single output that contains the HTML fragment that will display the dial.

The action sequence creates the HTML in three steps:

1. The SQL lookup component to retrieve the data set of data for the customer:
   Inputs: customer
   Action:
   ```
   select  DISTINCT PRODUCTLINE,
           SUM(QUANTITYORDERED * PRICEEACH) as "TOTAL SALES"
   from    PRODUCTS, ORDERDETAILS, CUSTOMERS, ORDERS
   where   CUSTOMERS.CUSTOMERNUMBER = {PREPARE:customer}
   and     CUSTOMERS.CUSTOMERNUMBER = ORDERS.CUSTOMERNUMBER
   and     ORDERS.ORDERNUMBER = ORDERDETAILS.ORDERNUMBER
   and     PRODUCTS.PRODUCTCODE  = ORDERDETAILS.PRODUCTCODE
   group by PRODUCTLINE
   ```
   The customer parameter is embedded into query using {PREPARE:customer}
   Outputs: data result set (chart-data)
2. The chart component to create the pie chart image:
   Inputs: data result set (chart-data)
   Action: Creates a pie chart using the chart definition that is embedded in the component definition. The chart settings are not described in detail here. The pie chart image is written out to a temporary file.
   Outputs: chart filename, base URL
3. Another Javascript component creates the final HTML fragment to return to the dashboard:
   Inputs: chart filename (CHARTOUTPUT), base URL (BASEURL)
   Action:
   ```
   chart_url = '<img src="' + BASEURL + "getImage?image=" + CHARTOUTPUT + '" />';
   ```
   Outputs: chart HTML (chart_url)

AJAX Embedded Report

This report is defined in two files, an action sequence:

pentaho-demo/pentaho-solutions/samples/google/customer_details.xaction

and a report definition:

pentaho-demo/pentaho-solutions/samples/google/customer_details_report.xml

The action sequence loads the data, executes the report, and returns the result.
The report definition is used by the report engine (Pentaho Reporting) to layout the report. You can use the Pentaho Report Wizard to design a report definition.

Because the report content is going to be embedded into our interactive dashboard the report needs to create a fragment of HTML instead of an entire HTML page. The report definition includes a property called `BodyFragment` within the configuration to enable this.

```
<property name="org.jfree.report.modules.output.table.html.BodyFragment">true</property>
```

To do this the action sequence takes as input:

- customer: The customer number for which the pie chart is needed

The action sequence has a single output that contains the HTML fragment that will display the dial.

The action sequence creates the HTML in two steps:

1. The SQL lookup component to retrieve the data set of data for the customer:
   Inputs: customer
   Action:
   ```
   select  ORDERDATE as "ORDER DATE",
           SUM(QUANTITYORDERED * PRICEEACH) as VALUE
   from    CUSTOMERS, ORDERS, ORDERDETAILS
   where   CUSTOMERS.CUSTOMERNUMBER = {PREPARE:customer}
   and     CUSTOMERS.CUSTOMERNUMBER = ORDERS.CUSTOMERNUMBER
   and     ORDERS.ORDERNUMBER = ORDERDETAILS.ORDERNUMBER
   group by ORDERDATE
   order by ORDERDATE DESC
   ```
   The customer parameter is embedded into query using {PREPARE:customer}
   Outputs: data result set (chart-data)
2. The report component to create the HTML content:
   Inputs: data result set (chart-data)
   Action: Creates a report using the report definition that is defined in the action sequence resources. The report definition file is not described in detail here. The dial is written out to a temporary file.
   Outputs: report content (report-output)

Appendix 1: Google Map JSP

This file can be found in the Pentaho PCI:

pentaho-demo/jboss/server/default/deploy/pentaho.war/jsp/Map.jsp

```
<%@ page language="java"
     import="org.pentaho.core.system.PentahoSystem,
           org.pentaho.core.session.IPentahoSession,
           org.pentaho.core.util.XmlHelper,
           org.pentaho.messages.Messages,
           org.pentaho.core.util.UIUtil,
           org.pentaho.messages.util.LocaleHelper,
           org.pentaho.util.VersionHelper,
           org.pentaho.ui.component.INavigationComponent,
           org.pentaho.ui.component.NavigationComponentFactory,
           org.pentaho.core.ui.SimpleUrlFactory,
           org.pentaho.core.solution.SimpleParameterProvider,
           org.dom4j.*,
           org.pentaho.core.solution.ActionResource,
           org.pentaho.core.solution.IActionResource,
           org.pentaho.core.util.IUITemplater,
           org.pentaho.core.solution.SimpleOutputHandler,
           org.pentaho.core.services.BaseRequestHandler,
           org.pentaho.core.runtime.IRuntimeContext,
           org.pentaho.core.connection.IPentahoResultSet,
           org.pentaho.ui.ChartHelper,
           java.io.*,
           java.util.*" %><%

/*
 * Copyright 2006 Pentaho Corporation.  All rights reserved.
 * This software was developed by Pentaho Corporation and is provided under the terms
 * of the Mozilla Public License, Version 1.1, or any later version. You may not use
 * this file except in compliance with the license. If you need a copy of the license,
 * please go to http://www.mozilla.org/MPL/MPL-1.1.txt. The Original Code is the Pentaho
 * BI Platform.  The Initial Developer is Pentaho Corporation.
 *
 * Software distributed under the Mozilla Public License is distributed on an "AS IS"
 * basis, WITHOUT WARRANTY OF ANY KIND, either express or  implied. Please refer to
 * the license for the specific language governing your rights and limitations.
 *
 * @created Jul 23, 2005
 * @author James Dixon
 *
 */

 /*
 * This JSP is an example of how to use Pentaho components and AJAX library to build a
 * Google Maps dashboard.
 * This file loads customer data and displays it using Google Maps.
 * The script for this file is in js/google-demo.js
 * The Pentaho AJAX library is in js/pentaho-ajax.js
 * See the document 'Pentaho AJAX Guide' for more details
 */

     // Set the character encoding e.g. UTF-8
     response.setCharacterEncoding(LocaleHelper.getSystemEncoding());

     // Get the current Pentaho session or create a new one if needed
     IPentahoSession userSession = UIUtil.getPentahoSession( request );

     // Set the default thresholds
     int topthreshold = 100000;
     int bottomthreshold = 50000;

     // Get the server and port. We use this to check for an invalid Google Maps API key
     boolean defaultKeyInvalid = false;
     String serverName = request.getServerName();
     int serverPort = request.getServerPort();

     // Get a templater object
     String intro = "";
     String footer = "";
     IUITemplater templater = PentahoSystem.getUITemplater( userSession );
     if( templater != null ) {

        // Load a template for this web page
        String template = null;
        try {
           ActionResource resource = new ActionResource(
                 "", IActionResource.SOLUTION_FILE_RESOURCE,
                 "text/xml", "system/custom/template-document.html" );
           template = PentahoSystem.getSolutionRepository(userSession)
                                .getResourceAsString( resource );
        } catch (Throwable t) {
        }

        // Check to see if we are using the default Google Maps API key but not for localhost:8080
        String googleMapsApiKey = PentahoSystem.getSystemSetting(
                                    "google/googlesettings.xml", "google_maps_api_key", null);
        if( ( !serverName.equals( "localhost" ) || serverPort != 8080 ) &&
              googleMapsApiKey.equals( "ABQIAAAAoGNlMo4FkTb3mcC5mj5ERRTwM0brOpm-
All5BF6PoaKBxRWWERR0378zH4HL9GyjgMMHJmj_viP4PQ" ) ) {
           // the default Google Maps API key is not valid for this server and port
           defaultKeyInvalid = true;
        } else {
           // insert the Pentaho AJAX, Google Maps,
           // and demo script references into the document header
           template = template.replaceAll( "\\{header-content\\}",
              "<script language=\"javascript\" src=\"js/pentaho-ajax.js\"></script>\n"+
              "<script src=\"http://maps.google.com/maps?file=api&amp;v=2&amp;key="+
              googleMapsApiKey+
              "\" type=\"text/javascript\"></script>\n"+
              "<script language=\"javascript\" src=\"js/google-demo.js\"></script>\n" );
           template = template.replaceAll( "\\{body-tag\\}",
                                    "onload=\"load()\" onunload=\"GUnload()\"" );
        }
        // Break the template into header and footer sections
        String sections[] = templater.breakTemplateString(
                             template, "Pentaho Google Maps Dashboard", userSession );
        if( sections != null && sections.length > 0 ) {
           intro = sections[0];
        }
        if( sections != null && sections.length > 1 ) {
           footer = sections[1];
        }
     } else {
        intro = Messages.getString( "UI.ERROR_0002_BAD_TEMPLATE_OBJECT" );
     }

     // Load the customer data. Do this by running an action sequence defined
     // in pentaho-solutions/samples/google/map1.xaction
     SimpleParameterProvider parameters = new SimpleParameterProvider();
     ArrayList messages = new ArrayList();
     // 'results' will store the customer data
     IPentahoResultSet results = null;
     IRuntimeContext runtime = null;
     try {
        // Run the action sequence
        runtime = ChartHelper.doAction( "samples", "google", "map1.xaction", "Map.jsp",
                                 parameters, userSession, messages, null );
        // See if we have a valid result
            if( runtime != null ) {
              if( runtime.getStatus() == IRuntimeContext.RUNTIME_STATUS_SUCCESS ) {
                 if( runtime.getOutputNames().contains("data") ) {
                    results = runtime.getOutputParameter( "data" ).getValueAsResultSet();
                 }
              }
            }
         } finally {
             if (runtime != null) {
                 // Now clean up
                 runtime.dispose();
             }
         }

        String customerNum = "";
        String customer = "";
        String city = "";
        String state = "";
        String zip = "";
        String value = "";

%>

<%= intro %>
```

```
<%
     if( defaultKeyInvalid ) {
        // The default key is not valid so we put out a nice message about it.
%>

The Google Maps API key that ships with the Pentaho Pre-Configured Installation will
only work with a server address of 'http://localhost:8080'.
<p/>
To use Google Maps with this server address ( <%= serverName %>:<%= serverPort %> )
you need to apply to Google for a new key.
<p/>
Once you have the new key you need to add it to the Google settings file in the Pentaho
system (.../pentaho-solutions/system/google/googlesettings.xml)
<p/>
<a target='google-map-api-key' href='http://www.google.com/apis/maps/signup.html'>Click here</a>
to get a Google Maps API Key for this server.

<%
     } else {
        // embed the customer data into the web page
%>

    <script type="text/javascript">

    //<![CDATA[

     function addPoints() {
        if (GBrowserIsCompatible()) {
        <%
           // Add all of the customer data into the web page
           int n = results.getRowCount();
           for( int row=0; row<n; row++ ) {
              // Get the information about the customer in the current row
              customerNum = results.getValueAt( row, 0 ).toString();
              customer = (String) results.getValueAt( row, 1 );
              city = (String) results.getValueAt( row, 2 );
              state = (String) results.getValueAt( row, 3 );
              value = results.getValueAt( row, 5 ).toString();
              // create a javascript call that passes the customer's details
              %>
              try {
                 showAddress( "<%= city %>,<%= state %>", "<%= customer %>",
                              "<%= customerNum %>", <%= value %>, false );
              } catch (e) {}
              <%
           }
        %>
        }
    }

    //]]>
    </script>

     <!-- create the visual elements of the page -->

     <!-- define the thresholds panel -->
    <div id="selections" style="position:absolute;width:345px;height:200px;top:40px;left:5px; ">

        <table border="0" cellpadding="0" cellspacing="0" width="100%" >
           <tr>
              <td valign="top">
                 <table border="0" cellpadding="0" cellspacing="0" width="100%">
                    <tr>
                          <td style="height: 25px; width: 25px;">
                             <img border="0" src="/pentaho-style/images/fly-top-left1.png" />
                          </td>
                          <td>
                          <span class="a" style="">Select Sales Thresholds</span>
                       </td>
                    </tr>
                    <tr>
                          <td colspan="2" style="background-color: #e5e5e5;">
                             <table width="100%" border="0" cellspacing="1" cellpadding="0">
                             <tr>
                                   <td>
                                   &nbsp;
                                </td>
                                <td colspan="2">
                                   View:
                                   <a href="javascript:void"
```

```
            onclick="map.setCenter( new GLatLng(35.55, -119.268 ), 6);
                     return false;">
               West Coast
            </a> |
            <a href="javascript:void"
            onclick="map.setCenter( new GLatLng(41.4263, -73.1799 ), 7);
                     return false;">
               East Coast
            </a>
         </td>
      </tr>

         <tr style="background-color: #e5e5e5;">
            <td>&nbsp;</td>
         <td valign="top" style="padding: 0px 0px 0px 0px;">

            <table>
               <tr>
                  <td>
                     <img border="0"
         src="http://labs.google.com/ridefinder/images/mm_20_red.png"/>
                  </td>
                  <td>
                     <
                     <select id="bottomthreshold" onchange="update(false)">
                        <option value="0">0</option>
                        <option value="10000">10000</option>
                        <option value="20000">20000</option>
                        <option value="30000">30000</option>
                        <option value="40000">40000</option>
                        <option value="50000" selected>50000</option>
                        <option value="60000">60000</option>
                        <option value="70000">70000</option>
                        <option value="80000">80000</option>
                        <option value="90000">90000</option>
                        <option value="100000">100000</option>
                        <option value="110000">110000</option>
                        <option value="120000">120000</option>
                        <option value="130000">130000</option>
                        <option value="140000">140000</option>
                        <option value="150000">150000</option>
                        <option value="160000">160000</option>
                        <option value="170000">170000</option>
                        <option value="180000">180000</option>
                        <option value="190000">190000</option>
                        <option value="200000">200000</option>
                     </select> <
                  </td>
                  <td>
                     <img border="0"
         src="http://labs.google.com/ridefinder/images/mm_20_yellow.png"/>
                  </td>
                  <td>
                     <
                     <select id="topthreshold" onchange="update(true)">
                        <option value="0">0</option>
                        <option value="10000">10000</option>
                        <option value="20000">20000</option>
                        <option value="30000">30000</option>
                        <option value="40000">40000</option>
                        <option value="50000">50000</option>
                        <option value="60000">60000</option>
                        <option value="70000">70000</option>
                        <option value="80000">80000</option>
                        <option value="90000">90000</option>
                        <option value="100000" selected>100000</option>
                        <option value="110000">110000</option>
                        <option value="120000">120000</option>
                        <option value="130000">130000</option>
                        <option value="140000">140000</option>
                        <option value="150000">150000</option>
                        <option value="160000">160000</option>
                        <option value="170000">170000</option>
                        <option value="180000">180000</option>
                        <option value="190000">190000</option>
                        <option value="200000">200000</option>
                     </select> <
                  </td>
                  <td>
                     <img border="0"
         src="http://labs.google.com/ridefinder/images/mm_20_green.png"/>
```

```
                                            </td>
                                         </tr>
                                      </table>

                                   </td>
                                   </tr>
                                </table>
                             </td>
                       </tr>
                       <tr>
                             <td style="height: 25px; width: 25px;">
                                <img border="0" src="/pentaho-style/images/fly-bot-left1.png" />
                             </td>
                             <td>&nbsp;</td>
                       </tr>
                    </table>
                 </td>
                 <td valign="top" style="padding: 0px 0px 0px 0px; font-size: .85em;">
                 </td>
              </tr>
           </table>
           <br/>
           <center>
              <img border="0"
                 src="/pentaho-style/images/pentaho_googlemap_white.png" style="padding-top:5px"/>
           </center>

        </div>

        <!-- define the customer details panel -->
       <div id="details-div" style="position:absolute;width: 320px;top:135px; left:30px;
border:0px;display:none;overflow: none;">

           <table border="0" cellpadding="0" cellspacing="0" width="100%" xheight="470">
              <tr>
                 <td valign="top">
                    <table cellpadding="0" cellspacing="0" width="100%">
                       <tr>
                             <td colspan="2"  valign="top" style="background-color: #e5e5e5;">
                                <table width="100%" cellspacing="1" cellpadding="0" height="100%">
                                   <tr style="background-color: #e5e5e5;">
                                      <td>
                                      &nbsp;
                                   </td>
                                   <td valign="top" style="padding: 0px 0px 0px 0px;">
                                      <div id="details-cell1">
                                      </div
                                   </td>
                                   </tr>
                                   <tr style="background-color: #e5e5e5;">
                                      <td>
                                      &nbsp;
                                   </td>
                                   <td valign="top">
                                      <center>
                                         Sales History
                                      </center>
                                      <div id="details-cell2">
                                      </div>
                                   </td>
                                   </tr>
                                </table>
                             </td>
                       </tr>
                       <tr>
                             <td style="height: 25px; width: 25px;">
                             <img border="0" src="/pentaho-style/images/fly-bot-left1.png" />
                          </td>
                             <td>
                                &nbsp;
                             </td>
                       </tr>
                    </table>
                 </td>
                 <td valign="top" style="padding: 0px 0px 0px 0px; font-size: .85em;">
                 </td>
              </tr>
           </table>
           <center>
              <img border="0" src="/pentaho-style/images/pentaho_googlemap_white.png"/>
           </center>
```

```
    </div>

    <!-- define the Google Map area -->
   <div id="map" style="position:absolute;width: 640px; height:
580px;top:40px;left:350px;border:1px solid #808080"></div>

<% } %>
```

Appendix 2: Pentaho AJAX Library

This Javascript library allows Pentaho Action Sequences to be executed from Javascript in a browser. This file can be found in the Pentaho PCI:

pentaho-demo/jboss/server/default/deploy/pentaho.war/js/pentaho-ajax.js

```
// The URL to the Pentaho servlet 'ViewAction'.
// This will default to '/pentaho/' if it is not set.
var PentahoURL = null;

/*
     Sets the URL to the Pentaho servlet 'ViewAction'. This should end in a '/'.
*/
function setPentahoURL( url ) {
     PentahoURL = url;
}

/*
     This function makes an AJAX call to the Pentaho server to execute an Action Sequence.
     The Action Sequence must exist within the Pentaho Solution Repository. Once executed the
     content generated by the Action Sequence will be passed to the named function
     Parameters:
          solution:The name of the solution
          path:     The path within the solution to the Action Sequence
          action:   The name of the Action Sequence
          params:   An array of arrays with name-value pair parameters for the Action Sequence
          func:     The function that will be called once the Action Sequence has been
                    executed
*/
function pentahoAction( solution, path, action, params, func ) {

     var http_request = null;
     // Create an HTTP Request object
     if (window.XMLHttpRequest) {
          // Try Mozilla, Safari, Firefox etc.
          http_request = new XMLHttpRequest();
          if (http_request.overrideMimeType) {
               http_request.overrideMimeType('text/xml');
          }
     }
     else if (window.ActiveXObject) {
          // IE
          try {
               http_request = new ActiveXObject("Msxml2.XMLHTTP");
          } catch (e) {
               try {
                    http_request = new ActiveXObject("Microsoft.XMLHTTP");
               } catch (e) {}
          }
     }

     // Make sure we got a valid HTTP Request object
     if (http_request == null) {
          alert('Could not create an AJAX request object');
          return false;
     }

     // Set the callback function to 'pentahoResponse' (below)
     http_request.onreadystatechange = function() { pentahoResponse(http_request, func); };

     // Construct the URL
     var url = "";
     if( PentahoURL != null ) {
          // PentahoURL has been set so use it
          url = PentahoURL;
     } else {
          // PentahoURL has not been set so use the default
          url = "/pentaho/";
     }

     // Add the Action Sequence solution, path and name to the URL
     url += "ViewAction?wrapper=false&solution="+solution+"&path="+path+"&action="+action;

     // Add the parameters to the URL
     if( params ) {
          var idx;
          for( idx=0; idx<params.length; idx++ ) {
               url += "&" +escape( params[idx][0] ) + "=" + escape( params[idx][1] );
          }
     }

     // Open the HTTP request
     http_request.open('GET', url, true);
     http_request.send(null);
     return true;
}

/*
     This function is called by the HTTP Request object when the request has been completed.
     This function does not need to be called from outside this library
*/
function pentahoResponse(http_request, func) {
     if (http_request.readyState == 4) {
          if (http_request.status == 200) {

               var content = http_request.responseText;
               eval( func+'( content )' );
          } else {
               eval( func+'( "There was a problem with the request." )' );
          }
     }
}
```

Appendix 3: Google Map Dashboard Library

This Javascript library provides the interactive functionality of the Google Map sample dashboard. This file can be found in the Pentaho PCI:

pentaho-demo/jboss/server/default/deploy/pentaho.war/js/google-demo.js

```
var map; // the map object
var redicon; // red icon to place on map
var yellowicon; // yellow icon to place on map
var greenicon; // green icon to place on map
var topThreshold = 100000; // initial upper theshold for coloring markers
var bottomThreshold = 50000; // initial lower threshold for coloring markers
var points = new Array();

var icon; // temporary variable for creating markers

var geocoder = new GClientGeocoder(); // the Google address -> lat/long converter

// create the red icon
icon = new GIcon();
icon.image = "http://labs.google.com/ridefinder/images/mm_20_red.png";
icon.shadow = "http://labs.google.com/ridefinder/images/mm_20_shadow.png";
icon.iconSize = new GSize(12, 20);
icon.shadowSize = new GSize(22, 20);
icon.iconAnchor = new GPoint(6, 20);
icon.infoWindowAnchor = new GPoint(5, 1);
redicon = icon;

// create the yellow icon
icon = new GIcon();
icon.image = "http://labs.google.com/ridefinder/images/mm_20_yellow.png";
icon.shadow = "http://labs.google.com/ridefinder/images/mm_20_shadow.png";
icon.iconSize = new GSize(12, 20);
icon.shadowSize = new GSize(22, 20);
icon.iconAnchor = new GPoint(6, 20);
icon.infoWindowAnchor = new GPoint(5, 1);
yellowicon = icon;

// create the green marker
icon = new GIcon();
icon.image = "http://labs.google.com/ridefinder/images/mm_20_green.png";
icon.shadow = "http://labs.google.com/ridefinder/images/mm_20_shadow.png";
icon.iconSize = new GSize(12, 20);
icon.shadowSize = new GSize(22, 20);
icon.iconAnchor = new GPoint(6, 20);
icon.infoWindowAnchor = new GPoint(5, 1);
```

```
greenicon = icon;

function load() {
     // this is called then the page is loaded

     // check to see if the browser supports the Google Map API
     if (GBrowserIsCompatible()) {
          // create the map object and tell it which div on the page to use
        map = new GMap2(document.getElementById("map"));
          // set the initial center of the map
        map.setCenter(new GLatLng(37.4419, -95), 4);
          // set the controls available to the user
          map.addControl(new GSmallMapControl());
          map.addControl(new GMapTypeControl());
          // call the function in the main page that adds the customers to the map
          addPoints();
     }
}

function customerClick( ) {
     // this is called then the user clicks on a marker on the map

     // call the dial action (sample/google/dial.xaction) and call the function
     // updateInfoWindow with the results
     pentahoAction( "samples", "google", "dial.xaction",
          new Array(
               new Array( "customer", currentRecord[7] ),
               new Array( "value", currentRecord[4] ),
               new Array( "max", 200000 ),
               new Array( "low", bottomThreshold ),
               new Array( "high", topThreshold )
          ),
          'updateInfoWindow'
     );

}

function updateInfoWindow( content ) {
     // this is called when the server returns the HTML content for the dial

     // construct the HTML to go in the info window and open it
     currentMarker.openInfoWindowHtml("<table border='0' width='375' cellpadding='0'
          cellspacing='0'><tr><td rowspan='2' height='125' width='250' style='xborder-right:1px
          solid #bbbbbb'><table><tr><td nowrap>Customer:</td><td nowrap>" + currentRecord[7] +
          "</td></tr><tr><td nowrap>Name:</td><td nowrap>" + currentRecord[3] +
          "</td></tr><tr><td nowrap>Location:</td><td
          nowrap>"+currentRecord[2]+"</td></tr><tr><td nowrap>Current Sales:</td><td
          nowrap>"+currentRecord[4]+"</td></tr></table></td><td colspan='2' valign='top'
          width='125'>"+content+"</td></tr><tr><td>0</td><td style='text-
          align:right'>200,000</td></tr></table>");

     // now call the pie chart action (sample/google/chart.xaction) and call the function
     // updateProductMix with the results
     pentahoAction( "samples", "google", "chart.xaction",
          new Array( new Array( "customer", currentRecord[7] ) ),
          'updateProductMix'
     );
}

function updateProductMix( content ) {
     // this is called when the server returns the HTML content for the pie chart

     // make sure the side panel is displayed
     document.getElementById( 'details-div' ).style.display='block';

     // insert the pie chart HTML content into the page
     document.getElementById( 'details-cell1' ).innerHTML=content;

     // now call the report action (sample/google/customer_details.xaction) and call
     // the function updateHistory with the results
     pentahoAction( "samples", "google", "customer_details.xaction",
          new Array( new Array( "customer", currentRecord[7] ) ),
          'updateHistory'
     );
}

function updateHistory( content ) {
     // this is called when the server returns the HTML content for the history report

     // make sure the side panel is displayed
     document.getElementById( 'details-div' ).style.display='block';

     // insert the report HTML content into the page
     document.getElementById( 'details-cell2' ).innerHTML=content;
}

function showAddress(address, name, custNum, value, selected) {
     // this is called from the main page for each customer in the data

     // use the Google Map geocoder to get the latitude and longitude for the customer
     geocoder.getLatLng(
          address,
          function(point) {
               if (!point) {
                    alert(address + " not found");
               } else {
                    // create a new record for this customer
                    var record = new Array( null, point, address, name, value,
                                                  selected, null, custNum );
                    // add the record to the list of customers
                    points.push( record );

                    // show this customer on the map
                    showMarker( null, null, record );
               }
          }
     );
}

function showMarker( oldMarker, oldIcon, record ) {
     // this is called to display a customer on the map

     var icon;
     var value = record[4];
     var point = record[1];

     // work out which color we should display
     if( value < bottomThreshold ) {
          icon = redicon;
     }
     else if( value > topThreshold ) {
          icon = greenicon;
     } else {
          icon = yellowicon;
     }

     if( icon == oldIcon ) {
          // this marker has not changed so return the old one
          return oldMarker;
     }

     // this marker is new or changed so we need to remove the old marker and add this
     record[5] = icon;
     // remove the old marker
     map.removeOverlay( oldMarker );

     // create a new marker with a click listener
     var marker = new GMarker(point, icon);
     // add the marker to the map
     map.addOverlay(marker);
     // add a click listener to the marker to call the 'infoWindow' function
     GEvent.addListener(marker, "click", function() {
          infoWindow( marker, record );
     });
     return marker;
}

var currentMarker = null;
var currentRecord = null;

function infoWindow( marker, record ) {
     // this is called by the click listener for each marker
     currentMarker = marker;
     currentRecord = record;
     // now update the page
     customerClick( );
}

function update(topChange) {
     // this is called when the user changes the coloring thresholds
     var n = points.length;

     // make sure the threshold values don't overlap
     var idx1 = document.getElementById('topthreshold').selectedIndex;
     var idx2 = document.getElementById('bottomthreshold').selectedIndex;
     if( idx1 < idx2 ) {
          if( topChange ) {
               document.getElementById('bottomthreshold').selectedIndex = idx1;
          } else {
               document.getElementById('topthreshold').selectedIndex = idx2;
          }
     }

     // get the current threshold values
     topThreshold = document.getElementById('topthreshold').value;
     bottomThreshold = document.getElementById('bottomthreshold').value;

     // update each marker
     for( idx=0; idx<n; idx++ ) {
          var marker = points[idx][0];
          var icon = points[idx][5];
          points[idx][0] = showMarker( marker, icon, points[idx] );
     }

     // if we are currently showing the info window update it
     if( currentRecord ) {
          pentahoAction( "samples", "google", "dial.xaction",
               new Array(
                    new Array( "customer", name ),
                    new Array( "value", currentRecord[4] ),
                    new Array( "max", 200000 ),
                    new Array( "low", bottomThreshold ),
                    new Array( "high", topThreshold )
               ),
               'updateInfoWindow'
          );
     }

}
```