ORACLE®

Oracle® WebCenter Analytics

Development Guide

Version 10g Release 3 (10.3)
Revised: November 2008

Contents

Copyright

additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

About This Guide

This documentation describes how to use the Oracle WebCenter Analytics APIs in custom applications.

This guide is organized as follows:

Analytics OpenUsage API

You can use the Oracle WebCenter Analytics OpenUsage API to raise Analytics events from custom portlets and applications and store them in the Analytics database. This tutorial presents an introduction to OpenUsage and provides basic use cases and a simplified example. The sample application includes examples of calling the OpenUsage Java API and using the OpenUsage event tag.

Analytics Query API

This section describes how to use the Oracle WebCenter Analytics APIs Query API to access the Analytics database in custom applications.

- *Events and Dimensions* on page 55. This appendix is a reference of the events and dimensions that are defined for Oracle WebCenter Interaction products.
- *Generating and Using a Query API Client* on page 68. This appendix describes how to generate a Java Query API client using JAX-WS.

Audience

This documentation is intended for software developers responsible for creating external applications that need to utilize data from Oracle WebCenter Analytics. The audience of this documentation is assumed to be proficient in developing applications that use SOAP web services.

Typographical Conventions

This document uses the following typographical conventions:

Convention	Typeface	Examples/Notes

• File names
• Folder names
• Screen elements
	`file name`
	`folder name`
	screen elements
		• Upload `procedures.doc` to the portal.
		• The log files are stored in the `logs` folder
		• To save your changes, click **Apply Changes**.

• Text you enter	`User input`	Type `Marketing` as the name of your community.
• Variables you enter	*Variables*	Enter the base URL for the Remote Server. For example, `http://`*`my_computer`*`.`

Convention	Typeface	Examples/Notes
• New terms • Emphasis • Object example names	*italic*	• *Portlets* are web tools embedded in your portal. • The URL *must* be a unique number. • The example Knowledge Directory displayed in Figure 5 shows the *Human Resources* folder.

Additional Documentation

The following additional documentation is available to assist with Oracle WebCenter Analytics.

Resource	Description
Installation Guide	This guide describes the prerequisites (such as required software) and procedures for installing Oracle WebCenter Analytics.
Administrator Guide	This guide describes how to manage, maintain, and troubleshoot Oracle WebCenter Analytics.
Release Notes	The release notes provide information about new features, issues addressed, and known issues in the release.
Online Help	The online help is written for all levels of Analytics users. It describes the user interface for Analytics and gives detailed instructions for completing tasks in Analytics. To access online help, click the help icon within the Analytics application.
Deployment Guide	This guide is written for business analysts and system administrators. It describes how to plan your Oracle WebCenter Interaction deployment.

CHAPTER 2

About the Oracle WebCenter Analytics OpenUsage API

The Oracle WebCenter Analytics OpenUsage API allows you to raise Analytics events from custom portlets and applications and store them in the Analytics database. This section presents an introduction to OpenUsage and provides basic use cases and a simplified example. The sample application includes examples of calling the OpenUsage Java API and using the OpenUsage event tag.

Oracle WebCenter Analytics collects information about the activity taking place within the Oracle WebCenter Interaction portal and in web applications, so you can respond better to your users' needs. Analytics delivers detailed information about the use of specific content items and portlets, as well as community activity such as document downloads and discussion postings; it can even track activity by group or individual user. These usage details help ensure you develop and deliver the best content and applications for your users.

The only requirement for using the OpenUsage API is to have the OpenUsage libraries and a connection to a network that allows UDP traffic. The OpenUsage API sends portal usage tracking metrics from the Interaction component on the portal server, as well as custom portal and non-portal events from external applications, to the Analytics Collector Service via the Portal Message Bus (PMB). Analytics services are installed on a stand-alone Analytics Services Server. The Analytics database stores all the metrics gathered and returns them to the Analytics Services Server or external application. The Analytics Service provides Analytics data to end users through the Analytics Console or Analytics portlets on the portal server.

For details on installing and configuring Analytics components, see the *Installation and Upgrade Guide for Oracle WebCenter Analytics*.The Analytics Administration utility in the Oracle WebCenter Interaction portal provides a simple interface for defining custom events, parameters, and dimensions. Event data can then be queried using SQL for reporting to a non-portal application.

You would typically get started with Analytics and the OpenUsage API by following these steps:

1. *Defining Your Event Model* on page 12
2. *Registering and Configuring Events in Analytics Administration* on page 16
3. *Raising Analytics Events from a Custom Application* on page 17
4. *Configuring and Launching Analytics* on page 19
5. *Querying Oracle WebCenter Analytics and Displaying Statistics* on page 20

Additional Resources:

- OpenUsage JavaDocs:
  http://download.oracle.com/docs/cd/E13158_01/alui/analytics/docs103/devguide/openusage_javadocs/index.html
  (Download a zip of the package here:
  http://download.oracle.com/docs/cd/E13158_01/alui/analytics/docs103/devguide/openusage_javadocs/openusage.zip)
- OpenUsage TagDocs:
  http://download.oracle.com/docs/cd/E13158_01/alui/analytics/docs103/devguide/openusage_tagdocs/index.html
  (Download a zip of the package here:
  http://download.oracle.com/docs/cd/E13158_01/alui/analytics/docs103/devguide/openusage_tagdocs/openusage_tagdocs.zip)

Defining Your Event Model

The first step to using Analytics functionality in a custom application is deciding which events are useful and what information you want to track. The Analytics database uses a standard star schema, providing almost unlimited flexibility in data storage.

Follow the steps below to define your event model:

1. Identify the events you want to capture. An event typically defines one user action, for example, clicks, page visits, or downloads. In many cases, you can use portal events to capture the information you need. The OpenUsage API provides access to the following standard portal events:
   - Directory views
   - Document views
   - Page views
   - Portlet views and use
   - Search query and results
   - Login/logoff

You can also raise custom events and define your own parameters. A new fact table is created in the Analytics database for every custom event. For details on the Analytics database implementation, see the *Oracle WebCenter Analytics Database Schema*.

2. Identify facts about the event you want to track, for example, date, time, page, user, group, or member status. Each event includes several facts, represented by event parameters, which define the types of data generated by the event. By default, every event includes the following event parameters:

   - USERID: The ID of the user who triggers the event. You must use the OpenUsage API to set the User ID.
   - TIMEID: The unique ID number that is created for each occurrence of the event. This value is set by Analytics.
   - VISITID: The portal visit ID of the user who triggered the event. This parameter is only compatible with events that occur in the portal. This value is set by Analytics.
   - OCCURRED: The date and time when the event was generated. The format of the date/time stamp is determined by your database type. This value is set by Analytics.

   All the interfaces in the OpenUsage API include methods to get and set standard parameters, including user ID, portlet ID, page ID, URL, and referer. For a complete list of the standard parameters for each event type, see the OpenUsage API documentation (javadoc). You can also create your own event parameters to capture data that is not defined by the delivered defaults.

3. To capture non-numeric data, define a dimension table, for example, URL, page name, user name, group name, or membership level. You can use dimensions from your own application's tables or create new ones in Analytics Administration (see *Registering and Configuring Events in Analytics Administration* on page 16).

   The following examples describe event models.

   #1: User interest statistics

   Palo Alto Golf Course (PAGC) is setting up new classes and wants to see which topics users are most interested in. They decide to post a portlet listing three problem-specific articles on Golf Digest's Web site and track which of the links users click on the most. They also want to track users' skill levels since it is defined as a user property in their portal. Since the users are divided among three portal groups, PAGC also wants to track the users' group membership.

Solution: PAGC creates a portlet that displays links to the three articles. Each link points to a redirect page that calls the sendEvent method in the OpenUsage API, passing the event type ID for that page, the date-time, and user ID. The redirect page redirects to the actual link target. An administrator registers a custom event for the article click in Analytics Administration, along with the corresponding event parameters (event type ID, date-time, and user ID). The user ID parameter corresponds to the existing portal user dimension table, which includes skill level and group membership. PAGC creates a report that displays which links were clicked most often, grouped by skill level or portal group.

 # 2: Customer follow-through statistics

BMW is working on a portal application for ordering car accessories online. The application allows users to browse a catalog of items, view details, add the items to a shopping cart, and purchase the items added to the cart. BMW would like to track the following statistics: Number of times a user clicked to browse the catalog. Number of times (by item) a user selected to see each item's detailed description. Number of shopping cart sessions initiated. Number of times (by item) each item was added to a shopping cart. Number of times (by item) each item was converted into an order.

Solution: All events can be captured by calling the OpenUsage API from the portal application. The VIEW_ITEM event could also be captured by placing an OpenUsage event tag on the details page. An administrator registers the five custom events in Analytics Administration:

- BROWSE: Sent when a user clicks the catalog browse button.
- VIEW_ITEM: Sent when a user views the details of a particular item. Item ID is passed as a parameter.
- NEW_SHOPPING_CART: Sent when a new shopping cart session is initialized.
- ADD_ITEM: Sent when a user adds an item to a shopping cart. Item ID is passed as a parameter.
- CONVERTED_ITEM: Sent when an item is converted into an order. Item ID is passed as a parameter.

BMW creates reports to display how many users clicked to browse the catalog, to display how many initiated shopping cart sessions, and to show details on how many times each item was viewed, added to a cart, and/or purchased.

 # 3: Content repository usage statistics

Netformx Software has a knowledge base search portlet on its support page that allows users to search for a term in user manuals, articles, marketing material, and/or white papers. Netformx would like to know which of these four repositories is searched the most via the portlet.

Solution: The portlet calls the OpenUsageAPI when the Search button is clicked, passing in the appropriate knowledge repository as the event parameter. If more than one repository was selected, an event is sent for each selected repository. If the portlet uses either the ISearchEvent or the IDirView interface to raise a standard portal event, there is no need to register a custom event in Analytics Administration. Netformx creates a report to compare usage of the four repositories.

 # 4: Advertising campaign statistics

St. Paul Brewing Company is rolling out a new beer that has no carbs, no calories, and no taste. They plan on promoting the beer via e-mail and advertisements on Google and Yahoo. They would like statistics for the following:

- Number of emails read.
- Number of click-throughs from each campaign (that is, email, Google, Yahoo).
- Number of new user accounts created due to click-throughs from campaigns, per campaign.
- Number of community visits to the new beer community generated from each marketing campaign.
- Number of free beer coupons (document in KD) downloaded due to each marketing campaign.

Solution: As with the other examples, all events can be sent via an OpenUsage API call with appropriate parameters. Page visits can be tracked using OpenUsage tags. An administrator registers any custom events and custom event parameters in Analytics Administration. A range of reports can be created to display statistics from the campaign.

The next step is to register any custom events in Analytics Administration. For details, see the next page, *Registering and Configuring Events in Analytics Administration* on page 16.

Registering and Configuring Events in Analytics Administration

You must register any custom events in Analytics Administration so they will be recognized by the Analytics Collector Service. Portal events are collected automatically. Custom (non-portal) events are called Managed Events in Analytics Administration.

To capture non-numeric data, you must use a dimension table. You can use existing dimension tables or create new ones, called Managed Dimensions in Analytics Administration. After you create a new dimension, you must create a new event parameter of type String, and associate it to the dimension (see Steps 3 to 5 below). To use an existing dimension table, create a parameter of type Integer that maps to the ID column in your dimension table. It is recommended that you do not create too many new dimensions, since they slow down the speed of data collection and reporting.

The sample application used in this example displays links in a portlet and tracks which pages are accessed by users. The application also keeps track of which OpenUsage method is used to raise the event. The event model uses one custom event with four parameters, two of which have associated dimension tables. These instructions explain how to register this custom event in Analytics Administration:

1. Go to Oracle WebCenter Interaction portal Administration. Click the **Select Utility** drop-down list, and select **Analytics Administration**.
2. Go to the **Event Registration** page in Analytics Manager.
3. Under **Managed Dimensions**, click **Add**. Create a new dimension called `event method`. (Dimension names can be up to 20 characters in length and can only include letters, numbers, spaces, and underscores.) Leave the default table name. If you are expecting a small number of unique values for a dimension, select the **Unique?** checkbox to reduce the size of the table.
4. Under **Managed Dimensions**, click **Add**. Create a new dimension called `page name`. Leave the default table name.
5. Under **Managed Events**, click **Add**. Create a new event called `demo event` with the parameters listed in the table below. **Note:** Leave the default table name and column names. (Event and parameter names can be up to 14 characters in length and can only include letters, numbers, spaces, and underscores.)

Name	Data Type	Dimension

`page id`	Integer	—
`page name`	String	page name
`event method`	String	event method
`date`	Date	—

6. Click **Finish** to save the event.
7. On the Event Registration page, select the check box next to the new “demo event” you created in step 5 and click **Enable**. (By default, the Analytics Collector Service starts saving an event's data 30 minutes after you click Enable.)
8. Click **Finish** to save your changes. WARNING: You cannot remove or rename an event or its parameters and dimensions after clicking Finish on the Event Registration page.

Note: The Event Registration page lists the names of the tables created in the Analytics database for the custom event and its dimensions (ASCFACT_* for fact tables and ASCDIM_* for dimension tables). These table names are used to query the database only. The OpenUsage API uses the event or dimension name as defined in the UI (that is, "demo event" not "ASCFACT_DEMO_EVENT").

For details on the other pages in Analytics Administration, see the online help or the *Administrator Guide for BEA AquaLogic Interaction Analytics*.

The next step is to add events to your custom application.

Raising Analytics Events from a Custom Application

After you have defined your events in Analytics Administration, you can use OpenUsage to raise events in response to user actions.

This example is a sample application implemented as a portlet. The portlet is a JSP page that displays two static links and a text box with a submit button ("Go to My Demo Page!"). At the bottom of the portlet, there are two radio buttons and another submit button ("View Events!"). Each link in the portlet uses a different approach to raise an event.

- **Using the OpenUsage event tag:** The first link ("OpenUsage Rocks") takes the user to another JSP page that raises an event using the <pt:as.event> tag. As shown in the code snippet below, the tag is implemented by including it in the page. Include a `pt:as.fact` attribute for each parameter defined for the event in Analytics Administration.

For additional details on syntax, see the OpenUsage TagDocs. **Note:** Tags can be used only in portlets and gatewayed pages because they must be processed by the Oracle WebCenter Interaction Tag Transformation Engine. For more information on using tags, *Oracle WebCenter Interaction Web Service Development Guide*.

```
<p><html>
<body>
...

<h1>OpenUsage Rocks!</h1>
<p>(This page generates an event using the OpenUsage Portal Tag
 Library)
<p><a href="demo.jsp">Back</a>

<!-- Send the event using the OpenUsage tag library -->
<span xmlns:pt='http://www.plumtree.com/xmlschemas/ptui/'>
<pt:as.event pt:name="demo event" pt:userID="1">
  <pt:as.fact pt:name="page id" pt:value="1" pt:type="integer"/>

  <pt:as.fact pt:name="page name" pt:value="OpenUsage Rocks!"
pt:type="string"/>
  <pt:as.fact pt:name="event method" pt:value="OpenUsage Tag
Library" pt:type="string"/>
  <pt:as.fact pt:name="date" pt:value="<%= currentDateString%>"
 pt:type="date"/>
</pt:as.event>
</span>
</body>
</html>
```

- **Using the OpenUsage Java API:** The second link in the portlet takes the user to another JSP page that raises an event using the OpenUsage API. First, a new event is created using the `ASEventFactory.createManagedEvent()` method. A `setParameter()` method is called for each parameter defined for the event in Analytics Administration. Finally, the event is raised and sent to Analytics using the `sendEvent()` method. These methods can be used in any application that has access to the OpenUsage libraries and connection to a network that allows UDP traffic. For additional details on these methods, see the OpenUsage API documentation (javadoc).

```
<p><%@page import="java.text.SimpleDateFormat"%>
<%@page
import="com.plumtree.analytics.openusage.*"%></p><p><html>
<body>

<h1>OpenUsage Is A-OK!</h1>
```

```
<p>
  (This page generates an event using the OpenUsage Java API)
<p>

<a href="demo.jsp">Back</a

<%
  // create the event using the OpenUsage Java API
  IManagedEvent evt = ASEventFactory.createManagedEvent("demo
event");
  evt.setParameter("page id", new Integer(3));
  evt.setParameter("page name", "OpenUsage Is A-OK!");
  evt.setParameter("event method", "OpenUsage Java API");
  evt.setLongAsDateParameter("date", new
Long(System.currentTimeMillis()));</p><p> // send it!
  evt.sendEvent();
%>

</body>
</html>
```

You could also use a text box/submit button in the portlet and use the OpenUsage API to raise an event. The form would include a hidden input element that triggers a Java servlet to raise the event. The code used by the servlet to raise the event would be identical to the code above except it would take in the page name entered in the text box and store this value in the page name dimension table.

The next step is to configure Analytics to store events from your custom application.

Configuring and Launching Analytics

You must configure Oracle WebCenter Analytics to store events from custom applications by modifying the database.properties and openusage.xml files.

The instructions below are simplified for example purposes only. For a list of ports used by Analytics and details on configuring the Analytics database, see the *Installation and Upgrade Guide for Oracle WebCenter Analytics*.

1. Open the openusage.xml file and enter the Analytics server port:

   ```
   <UNICAST_MODE>YES</UNICAST_MODE>
   <UNICAST_IP>$analytics_server_host$|port=31314</UNICAST_IP>
   ```

2. Open the database.properties file and enter database information for the Analytics Server. (Use the appropriate driver and URL for your configuration.)

   ```
   jdbc.driver=com.plumtree.jdbc.sqlserver.SQLServerDriver
   jdbc.url=jdbc\:plumtree\:sqlserver\://$analytics_database_server$\:1433;DatabaseName\=$analytics_database_name$
   jdbc.user=$analytics_database_user$
   jdbc.password=$analytics_database_password$
   ```

3. Place both configuration files in the appropriate config location. You can place the openusage.xml file anywhere, but your code must reference this file when initializing OpenUsage. This is done by calling the following method, where `configDirectory` is the path to the folder where openusage.xml is located. (This method only needs to be called once during your application run, normally during startup).

   ```
   ASEventFactory.setConfig("configDirectory", "openusage.xml");
   ```

   In the sample application, the OpenUsageDemoServlet class calls `setConfig` in the `init()` method. The `configDirectory` parameter is defined in web.xml as .\openusage-demo\config\settings. To change this location, open the .war file and modify the `configDirectory` init-param in web.xml.

4. Deploy the application .war file to any Java Servlet Container.
5. On the Analytics server, make sure the *Analytics Collector Service* is started.

To view the events from your custom application, query the Analytics database. For details, see the next page, .

Querying Oracle WebCenter Analytics and Displaying Statistics

To retrieve data from the Analytics database for custom reports, use SQL.

To define or determine table and column names, go to the Event Registration page of Analytics Administration (see *Defining Your Event Model* on page 12). For descriptions of the tables that are delivered with Analytics, see the *Oracle WebCenter Analytics Database Schema*. In this sample

application, the "View Events!" button opens a JSP page with a pie chart that shows the percentage of clicks by page name or event method. The query to the Analytics database is handled in the same servlet that raises the Go to My Demo Page! event. The sample application uses a JFreeCharts dataset producer to iterate over the data. The JSP page that displays the pie charts uses cewolf tags to render the data from the JFreeCharts dataset producer.

```
/**
* View "demo event" data using JFreeCharts and cewolf tag
libraries.
*
* @param req HttpServletRequest
* @param res HttpServletResponse
*/

public void viewEvents(HttpServletRequest req, HttpServletResponse
 res)
throws IOException, ServletException {
try {
 Connection conn = getDatabaseConnection();
 Statement stmt = conn.createStatement();
 String query = "";

  if ("EVENT_METHOD".equals(req.getParameter("groupby"))) {
   // group by eventMethod
  query = "select count(*), method.value " + "from
ascfact_demo_event fact, ascdim_page_name page, ascdim_event_method
 method " + "where fact.page_name = page.id and fact.event_method
 = method.id " + "group by method.value";
  req.getSession().setAttribute("GROUP_BY_CHECKED",
"EVENT_METHOD");

  } else {
   // group by pageName
  query = "select count(*), page.value " + "from ascfact_demo_event
 fact, ascdim_page_name page, ascdim_event_method method " + "where
 fact.page_name = page.id and fact.event_method = method.id " +
"group by page.value";
  req.getSession().setAttribute("GROUP_BY_CHECKED", "PAGE_NAME");

 }

 // execute the query
 ResultSet results = stmt.executeQuery(query);

 // create a JFreeChart DatasetProducer which will be rendered by
```

```
 the cewolf tag library
 DatasetProducer datasource = new PieChartDatasetProducer(results);

 req.setAttribute("datasource", datasource);

 // close database resources
 results.close();
 stmt.close();
 conn.close();

} catch (Exception e) {
e.printStackTrace();
throw new ServletException("Exception while creating
DatasetProducer - " + e);
}

RequestDispatcher dis = req.getRequestDispatcher("demo_chart.jsp");

dis.forward(req, res);
}</p>
```

(Analytics includes a collection of standard reports that display portal events; for details, see the *Administrator Guide for Oracle WebCenter Analytics*.)

For more information on querying the Analytics database from custom applications, see the next section, *About the Oracle WebCenter Analytics Query API* on page 23 .

CHAPTER 3

About the Oracle WebCenter Analytics Query API

This section describes how to use the Oracle WebCenter Analytics Query API to access the Analytics database in custom applications.

- *Using the Analytics Query API* on page 23. This tutorial teaches you the basics of using the Query API through a series of example queries.
- *The Anatomy of the Analytics Query API* on page 43. This chapter describes the Query API service and provides a detailed reference for creating SOAP requests for the Query API service and processing the SOAP responses.
- *Events and Dimensions* on page 55. This appendix is a reference of the events and dimensions that are defined for Oracle WebCenter Interaction products.
- *Generating and Using a Query API Client* on page 68. This appendix describes how to generate a Java Query API client using JAX-WS.

Using the Analytics Query API

This section introduces you to the Analytics Query API through a series of example queries.

To use the SOAP messages in this tutorial, you must have the following software installed:

- Oracle WebCenter Analytics 10.3 or AquaLogic Analytics 2.5
- Oracle WebCenter Interaction 10.3 or AquaLogic Interaction 6.5

In addition, the (optional) Java example code requires one of the following environments:

- Java SE 5 and Glassfish 9.0
- Java EE 5

This tutorial describes how to query data using SOAP and the Analytics Query API. In addition to the SOAP messages, a Java example application is provided to send and receive SOAP messages. The Java example is not necessary to understand the tutorial. A developer who is proficient in working with SOAP on any development platform can easily adapt this tutorial for that development platform.

The following steps are covered in this tutorial:

1. Create an application to send, receive, and process SOAP messages from the Query API service. For details, see *Communicating With the Query API Service* on page 24.
2. Create SOAP queries to view which portlets are being used on your portal. For details, see *Viewing Portlet Usage* on page 31.
3. Use filters to refine your queries. For details, see *Filtering Portlet Usage by Community* on page 35.
4. View events based on periods of time. For details, see *Tracking Portlet Usage* on page 38.
5. To use the Oracle WebCenter Analytics Query API, you must manually enable the Query API and configure the SOAP connection to the API service. For details, see *Configuring the Oracle WebCenter Analytics Query API* on page 43 and *Analytics Query API Configuration Settings* on page 42.

Communicating With the Query API Service

This topic leads you through the creation of a simple application that sends a SOAP message to the Query API Service and processes the response.

The application you create takes an XML file containing a SOAP message as input, sends the message to the Query API service, and outputs the results to standard output (the console). This application, or an application like it, is necessary to process the SOAP messages in the remainder of this tutorial.

The code in this topic requires Java SE 5 or greater and JAX-WS 2.0. That stated, the concepts should be familiar to any developer proficient in working with SOAP. You should have no problem implementing this application in the language of your choice.

1. Create a console-based application and write code necessary for a connection to the Query API service.

The Query API is located on your Analytics server at `http://analytics_server:port_number/analytics/QueryService`. The default *port_number* is 11944.

The following details are used to configure the SOAP connection to the Query API service:

Element	Detail
WSDL Location	`http://analytics_server:port_number/analytics/QueryService?WSDL`
Namespace	`http://www.bea.com/analytics/AnalyticsQueryService`
Service Name	`AnalyticsQueryService`
Port Name	`AnalyticsQueryServicePort`

This Java code establishes objects necessary for a SOAP connection to the Query API service on the Analytics Server named `analytics`:

```
import java.net.URL;

import javax.xml.namespace.QName;

import javax.xml.soap.SOAPMessage;

import javax.xml.ws.Dispatch;
import javax.xml.ws.Service;

class QueryAPIExample {

  public static void main(String[] args) {
    try
    {
      QName serviceName = new QName(
        "http://www.bea.com/analytics/AnalyticsQueryService",

        "AnalyticsQueryService");

      URL serviceURL = new URL(
        "http://analytics:11944/analytics/QueryService?wsdl");


     Service service = Service.create(serviceURL, serviceName);
```

```
      QName portName = new QName(
        "http://www.bea.com/analytics/AnalyticsQueryService",

        "AnalyticsQueryServicePort");

      Dispatch<SOAPMessage> dispatch = service.createDispatch(

        portName, SOAPMessage.class, Service.Mode.MESSAGE);
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
  }
}
```

2. Load the SOAP message from a file (`query.xml`) and send it to the Query API service.

   In the Java example, the SOAP message is loaded from `query.xml` and then sent using the `Dispatch` object. The code now looks like:

```
import java.io.FileInputStream;
import java.net.URL;

import javax.xml.namespace.QName;

import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPMessage;

import javax.xml.ws.Dispatch;
import javax.xml.ws.Service;

class QueryAPIExample {

  public static void main(String[] args) {
    try
    {
      QName serviceName = new QName(
        "http://www.bea.com/analytics/AnalyticsQueryService",

        "AnalyticsQueryService");
```

```
        URL serviceURL = new URL(
          "http://analytics:11944/analytics/QueryService?wsdl");


       Service service = Service.create(serviceURL, serviceName);


        QName portName = new QName(
          "http://www.bea.com/analytics/AnalyticsQueryService",

          "AnalyticsQueryServicePort");

        Dispatch<SOAPMessage> dispatch = service.createDispatch(

          portName, SOAPMessage.class, Service.Mode.MESSAGE);

    SOAPMessage request =
MessageFactory.newInstance().createMessage(null,
          new FileInputStream("query.xml"));

   // Send the request and get the response

        SOAPMessage response = dispatch.invoke(request);

      }
      catch (Exception e)
      {
        e.printStackTrace();
      }
    }
}
```

3. Process the SOAP response and output the data you are interested in to the console.

   The elements contained in the `<return>` element of the SOAP response are described in the following table:

Element	Description
`<results>`	One `<results>` element is provided for each row of data your request generates. Each `<results>` element contains one or more `<values>` elements.

Element	Description
<values>	In each <results> element, there is one <values> element for each type of data you request.
<columns>	The <columns> element describes the type or types of data you request. The sequence of the <values> elements in each <results> element corresponds directly to the sequence of the <columns> elements.

The following is an example of a response from the Query API service:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Header/>
<S:Body>

  <ns2:executeResultSetQueryResponse

xmlns:ns2="http://www.bea.com/analytics/AnalyticsQueryService">

   <return>
    <results>
     <values xmlns:xs="http://www.w3.org/2001/XMLSchema"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:type="xs:string">Report
     </values>
     <count>0</count>
    </results>
     <columns>portlet.name</columns>
   </return>
  </ns2:executeResultSetQueryResponse>
</S:Body>
</S:Envelope>
```

In this response the result of the query has one column, described as `portlet.name` (the `name` property of the `portlet` dimension of the event). The query has generated one row of data, and the name of the portlet returned is `Report`.

In the following code, the Java example is updated to process a response with any number of <results> and <columns>,and output the data to standard output.

```
import java.io.FileInputStream;
import java.net.URL;
import java.util.Iterator;

import javax.xml.namespace.QName;

import javax.xml.soap.MessageFactory;
import javax.xml.soap.Name;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPBodyElement;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPFactory;
import javax.xml.soap.SOAPMessage;

import javax.xml.ws.Dispatch;
import javax.xml.ws.Service;

class QueryAPIExample {

  public static void main(String[] args) {
    try
    {
      QName serviceName = new QName(
        "http://www.bea.com/analytics/AnalyticsQueryService",

        "AnalyticsQueryService");

      URL serviceURL = new URL(
        "http://analytics:11944/analytics/QueryService?wsdl");


     Service service = Service.create(serviceURL, serviceName);


      QName portName = new QName(
        "http://www.bea.com/analytics/AnalyticsQueryService",

        "AnalyticsQueryServicePort");

      Dispatch<SOAPMessage> dispatch = service.createDispatch(
```

```
        portName, SOAPMessage.class, Service.Mode.MESSAGE);

     SOAPMessage request =
MessageFactory.newInstance().createMessage(null,
        new FileInputStream("query.xml"));

  // Send the request and get the response

       SOAPMessage response = dispatch.invoke(request);

  //  Process the request and print the result

       SOAPBody resBody = response.getSOAPBody();
       SOAPFactory soapFactory = SOAPFactory.newInstance();
       Name name;

       name = soapFactory.createName(
         "executeResultSetQueryResponse",
         "ns2",
        "http://www.bea.com/analytics/AnalyticsQueryService");


       SOAPElement resResultSet =
           (SOAPElement)resBody.getChildElements(name).next();

       name = soapFactory.createName("return");
       SOAPElement resReturn =

(SOAPElement)resResultSet.getChildElements(name).next();

       name = soapFactory.createName("results");
       Iterator results = resReturn.getChildElements(name);

       System.out.println("Analytics Query API Results:");
       System.out.println("---------------------------\n\n");

       name = soapFactory.createName("values");
       SOAPElement value;

       while(results.hasNext())
       {
          Iterator values =
((SOAPElement)results.next()).getChildElements(name);
          while (values.hasNext())
          {
```

```
            value = (SOAPElement)values.next();
            System.out.print(value.getValue() + "\t\t");
          }
          System.out.print("\n");
        }


      }
      catch (Exception e)
      {
        e.printStackTrace();
      }
    }
  }
```

Viewing Portlet Usage

This topic describes how to use SOAP messages to report on portlet usage in the Oracle WebCenter Interaction portal.

Each time a portlet is accessed on the portal, a `portletUses` event is captured by Analytics. In this topic, you will learn how to:

- Determine how many times any portlet has been used on the portal (the number of `portletUses` events)
- View the name of the portlet associated with each `portletUses` event
- View how many different portlets have been used on the portal
- Group the results and see which portlets have been used on the portal

1. Determine the number of `portletUses` events.

This query is the simplest valid SOAP message that can be sent to the Query API service. In this message, you specify only the event you are interested in. In response, the Query API returns a count of that event in the Analytics database.

To query for the number of times the `portletUses` event has been captured by Analytics:

a) Create the basic SOAP framework for a Query API service request.

Inside the SOAP Body element of every request made to the Query API service, the query must be contained within the `<executeResultSetQuery>` element. Within the `<executeResultSetQuery>` there must be en element `<arg0>`. The element `<arg0>` contains the actual parameters of the query.

The following is the SOAP envelope and other elements that are the same for every Query API service SOAP request:

```
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Header/>
<S:Body>
  <Q:executeResultSetQuery
xmlns:Q="http://www.bea.com/analytics/AnalyticsQueryService">

    <arg0>
    </arg0>
  </Q:executeResultSetQuery>
</S:Body>
</S:Envelope>
```

b) Create an `<eventName>` element and populate it with the name of the event.

In the SOAP request, event names take the form of `{namespace}event` .

In this example, the namespace is `http://www.bea.com/analytics/ali` and the event is `portletUses`. The SOAP message looks like this:

```
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Header/>
<S:Body>
  <Q:executeResultSetQuery
xmlns:Q="http://www.bea.com/analytics/AnalyticsQueryService">

    <arg0>
      <eventName>
        {http://www.bea.com/analytics/ali}portletUses
      </eventName>
    </arg0>
  </Q:executeResultSetQuery>
</S:Body>
</S:Envelope>
```

When this SOAP message is sent with our application, the number of `portletUses` events is output to the console.

2. View the name of the portlet associated with each `portletUses` event.

The `<views>` element describes a specific *property* of the event that you are interested in seeing returned from the query. Each event has one or more associated *dimensions*, and each

dimension has one or more properties. In this example, you are interested in the `name` property of the `portlet` dimension, or, in other words, the name of the portlet associated with the `portletUses` event.

For more details on the `<views>` element, see *The <views> Element* on page 46.

The SOAP message looks like this:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Header/>
<S:Body>
  <Q:executeResultSetQuery
xmlns:Q="http://www.bea.com/analytics/AnalyticsQueryService">
    <arg0>

      <eventName>
        {http://www.bea.com/analytics/ali}portletUses
      </eventName>

      <views>
        <dimension>portlet</dimension>
        <property>name</property>
      </views>

    </arg0>
  </Q:executeResultSetQuery>
</S:Body>
</S:Envelope>
```

When this SOAP message is sent by the example application, the name of the portlet associated with each `portletUses` event is output to the console. In a test environment, this generated a list approximately 1350 portlet names long, with some names repeated hundreds of times.

3. View how many different portlets have been used on the portal.

   Instead of seeing all of the portlet names for each `portletUses` event, use the `<aggregate>` element of the `<views>` query to count how many distinct portlets are represented in `portletUses` events.

   The `<aggregate>` element takes an integer value. For a *count* aggregation, use the value `1`. For a description of all aggregate types, see *The <views> Element* on page 46.

   The SOAP message looks like this:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Header/>
<S:Body>
```

```
  <Q:executeResultSetQuery
xmlns:Q="http://www.bea.com/analytics/AnalyticsQueryService">
    <arg0>

      <eventName>
        {http://www.bea.com/analytics/ali}portletUses
      </eventName>

      <views>
        <dimension>portlet</dimension>
        <property>name</property>
        <aggregate>1</aggregate>
      </views>


    </arg0>
  </Q:executeResultSetQuery>
</S:Body>
</S:Envelope>
```

When this SOAP message is sent with the example application, the number of portlets that have been used in the portal is output to the console. In a test environment, this was 12.

4. Group the results and see which portlets have been used on the portal.

   For this step you use the `<groups>` element instead of the `<views>` element. The `<groups>` element groups the output by a given property. By grouping the output by portlet name, you see each portlet's name listed only once.

   For more details on the `<groups>` element, see *The <groups> Element* on page 48.

   The SOAP message looks like this:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Header/>
<S:Body>
  <Q:executeResultSetQuery
xmlns:Q="http://www.bea.com/analytics/AnalyticsQueryService">
    <arg0>

      <eventName>
        {http://www.bea.com/analytics/ali}portletUses
      </eventName>

      <groups>
        <dimension>portlet</dimension>
```

```
        <property>name</property>
      </groups>

    </arg0>
  </Q:executeResultSetQuery>
</S:Body>
</S:Envelope>
```

When this SOAP message is sent with the example application, a list of the names of portlets that have been used in the portal is output to the console. In a test environment, this was a list of twelve portlet names.

Filtering Portlet Usage by Community

This topic describes how to use the `<filters>` element to narrow queries of the Query API service to specific portal communities.

In the previous topic, *Viewing Portlet Usage* on page 31, you built queries to see which portlets are being used on your portal. In this topic, you use filters to refine these queries.

In this topic, you will learn to:

- Restrict queries to a single portal community
- Restrict queries to multiple portal communities

1. View which portlets have been used in the Analytics Console community.

   The `<filters>` element describes a property, a value for that property, and an operator to perform a check to see if any given event belongs in the result set.

   For more details on the `<filters>` element, see *The <filters> Element* on page 49.

   For this example, you build on the final SOAP message from the previous topic:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Header/>
<S:Body>
  <Q:executeResultSetQuery
xmlns:Q="http://www.bea.com/analytics/AnalyticsQueryService">
    <arg0>

      <eventName>
        {http://www.bea.com/analytics/ali}portletUses
      </eventName>
      <groups>
```

```
        <dimension>portlet</dimension>
        <property>name</property>
      </groups>
    </arg0>
  </Q:executeResultSetQuery>
</S:Body>
</S:Envelope>
```

This query returns the name of each portlet that has been used on the portal. Instead of all portlets, you want to see only the portlets that are accessed from the Analytics Console community. To do this, add the following `<filters>` element:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Header/>
<S:Body>
  <Q:executeResultSetQuery
xmlns:Q="http://www.bea.com/analytics/AnalyticsQueryService">
    <arg0>

      <eventName>
        {http://www.bea.com/analytics/ali}portletUses
      </eventName>
      <groups>
        <dimension>portlet</dimension>
        <property>name</property>
      </groups>
      <filters
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">


        <dimension>community</dimension>
        <property>name</property>
        <operator>1</operator>
       <values xsi:type="xs:string">Analytics Console</values>


     </filters>
    </arg0>
  </Q:executeResultSetQuery>
</S:Body>
</S:Envelope>
```

The `<operator>1</operator>` corresponds to the operator *equals*. For a list of all valid values for `<operator>`, see *The <filters> Element* on page 49.

The type of the <values> element must be defined. Because of this, you must include the XMLSchema and XMLSchema-instance namespaces.

When this SOAP message is sent with the example application, a list of the names of portlets that have been used in the Analytics Console community is output to the console. In a test environment, this was a list of six portlet names:

```
Analytics Query API Results:
----------------------------

Community Metrics
Other Metrics
Portlet Metrics
Publisher Administration
Report
Summary Metrics
```

2. View which portlets have been used in multiple communities.

   To create a filter where more than one value of a property is accepted, you must use the *in* operator, <operator>9</operator> and create a <values> element for each acceptable value.

   To list portlets used in both the Publisher Community and Analytics Console communities, change the <operator> to *in* and add a <values> for the Publisher Community community:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Header/>
<S:Body>
  <Q:executeResultSetQuery
xmlns:Q="http://www.bea.com/analytics/AnalyticsQueryService">
    <arg0>

      <eventName>
        {http://www.bea.com/analytics/ali}portletUses
      </eventName>
      <groups>
        <dimension>portlet</dimension>
        <property>name</property>
      </groups>
      <filters
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">


        <dimension>community</dimension>
```

```
        <property>name</property>
        <operator>9</operator>
       <values xsi:type="xs:string">Analytics Console</values>

        <values xsi:type="xs:string">Publisher
Community</values>

     </filters>
    </arg0>
  </Q:executeResultSetQuery>
</S:Body>
</S:Envelope>
```

When this SOAP message is sent with the example application, a list of the names of portlets that have been used in both the Publisher Community and Analytics Console communities is output to the console. In a test environment, this was a list of eight portlet names:

```
Analytics Query API Results:
----------------------------

Community Metrics
FCC News Portlet
Other Metrics
Portlet Metrics
Publisher Administration
Publisher Community Directory Portlet
Report
Summary Metrics
```

Tracking Portlet Usage

This topic describes how to use the `<groups>` element to track portlet usage based on periods of time.

In this topic you learn how to:

- View how many times each portlet was used on the portal
- View how many times a specific portlet was used each week

1. View how many times each portlet was used on the portal.
   a) Create a `<views>` element that counts events.

A <views> element that uses the COUNT aggregate and <property>*</property> will return a count of events.

You have the following SOAP message:

```
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Header/>
<S:Body>
  <Q:executeResultSetQuery
xmlns:Q="http://www.bea.com/analytics/AnalyticsQueryService">

    <arg0>

      <eventName>
        {http://www.bea.com/analytics/ali}portletUses
      </eventName>

      <views>
         <property>*</property>
         <aggregate>1</aggregate>
      </views>

    </arg0>
  </Q:executeResultSetQuery>
</S:Body>
</S:Envelope>
```

When this SOAP message is sent with the application, a count of every time a portlet has been used on the portal is output to the console.

Note: You might be wondering how this query is any different from the first query you made in *Viewing Portlet Usage* on page 31, where there was simply the <eventName> element and no <views>. As is, both queries do return the same results. Where you will see the difference is in the next step, when you start adding other parameters to the query. This <views> element causes the Query API service to return a count of events that meet the criteria of the query.

b) Create a <groups> element to list the portlets used on the portal.

Each row returned by the <groups> query represents one or more events for each distinct value of the grouped property. Combining a <groups> element with the <views> element above will give you a count of events that match each distinct value of the grouped property.

The SOAP message:

```
<S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Header/>
<S:Body>
  <Q:executeResultSetQuery
xmlns:Q="http://www.bea.com/analytics/AnalyticsQueryService">

    <arg0>

      <eventName>
        {http://www.bea.com/analytics/ali}portletUses
      </eventName>

      <views>
         <property>*</property>
         <aggregate>1</aggregate>
      </views>
      <groups>
        <dimension>portlet</dimension>
      <property>name</property>
      </groups>

    </arg0>
  </Q:executeResultSetQuery>
</S:Body>
</S:Envelope>
```

When this SOAP message is sent with the example application, a list of portlet names, each with a corresponding number representing how many times the portlet was used, is output to the console. In a test environment, it looked like this:

```
Analytics Query API Results:
----------------------------

98              Community Metrics
35              FCC News Portlet
150             Other Metrics
219             Portlet Metrics
7               Publisher Administration
18              Publisher Community Directory Portlet
274             Report
427             Summary Metrics
```

2. View how many times a specific portlet was used each day.

   To group results by time, create a `<groups>` element with the `dimension` set to `time` and a `<timeGrouping>` element set to the period of time you want grouped. The `<timeGrouping>` element takes an integer value.

   For details on the values used with `<timeGrouping>`, see *The <groups> Element* on page 48.

   For the example you are going to group results by day, or `<timeGrouping>3</timeGrouping>`, and then create a filter so you only see data for the Summary Metrics portlet.

   The SOAP message looks like this:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Header/>
<S:Body>
  <Q:executeResultSetQuery
xmlns:Q="http://www.bea.com/analytics/AnalyticsQueryService">
    <arg0>

      <eventName>
        {http://www.bea.com/analytics/ali}portletUses
      </eventName>

      <views>
         <property>*</property>
         <aggregate>1</aggregate>
      </views>

      <groups>
        <dimension>time</dimension>
        <property></property>
        <timeGrouping>3</timeGrouping>
      </groups>

      <groups>
        <dimension>portlet</dimension>
        <property>name</property>
      </groups>

      <filters
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      >
```

```
          <dimension>portlet</dimension>
          <property>name</property>
         <values  xsi:type="xs:string" >Summary Metrics</values>

          <operator>1</operator>
      </filters>

      </arg0>
    </Q:executeResultSetQuery>
</S:Body>
</S:Envelope>
```

When this SOAP message is sent with the example application, a list of portlet usage statistics by date is output to the console. In a test environment, it looked like this:

```
Analytics Query API Results:
----------------------------

173             Summary Metrics          3/21/08
15              Summary Metrics          3/24/08
35              Summary Metrics          3/27/08
14              Summary Metrics          3/28/08
8               Summary Metrics          3/31/08
93              Summary Metrics          4/1/08
89              Summary Metrics          4/2/08
24              Summary Metrics          4/4/08
```

Analytics Query API Configuration Settings

This topic provides the configuration details of the Query API service.

The following details are used to configure the SOAP connection to the Query API service:

Element	Detail
Service Location	`http://analytics_server:11944/analytics/QueryService`
WSDL Location	`http://analytics_server:11944/analytics/QueryService?WSDL`
Namespace	`http://www.bea.com/analytics/AnalyticsQueryService`

Element	Detail
Service Name	`AnalyticsQueryService`
Port Name	`AnalyticsQueryServicePort`

Configuring the Oracle WebCenter Analytics Query API

To use the Oracle WebCenter Analytics Query API, you must manually enable the Query API and configure the SOAP connection to the API service.

To manually enable the Query API, follow the directions below.

1. Stop the Analytics UI Service.
2. Go to <AnalyticsInstallationFolder>\10.3.0\webapps.
3. UnJar analytics.war.
4. Open web.xml in a text editor and remove all the comments "Uncomment this to enable QueryAPI".
5. Jar analytics.war.
6. Start the Analytics UI Service.

To configure the SOAP connection to the API service, use the settings on the following page: *Analytics Query API Configuration Settings* on page 42.

The Anatomy of the Analytics Query API

This section provides an overview of Analytics Query API reference topics.

The following topics are covered in this section:

- The configuration details for the Query API service, including service and WSDL locations.

  For details, see *Analytics Query API Configuration Settings* on page 42.
- A description of a Query API SOAP request, including specific details about query parameters.

  For details, see *The Query API SOAP Request* on page 44.
- A description of a Query API SOAP response.

For details, see *The Query API SOAP Response* on page 53.

The Query API SOAP Request

This topic provides a description of the components of a Query API SOAP request.

The Basic Request Body

Every valid Query API SOAP request must contain the following elements:

- `<arg0>`, which is contained by
- `<executeResultSetQuery>`, which is contained by
- the `<Body>` element of a standard SOAP envelope

These elements form the basic request body for every Query API request:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Header/>
<S:Body>
<Q:executeResultSetQuery
    xmlns:Q="http://www.bea.com/analytics/AnalyticsQueryService">

<arg0>

</arg0>

</q:executeResultSetQuery>
</S:Body></S:Envelope>
```

The Query Elements

The query elements for each request are contained within the `<arg0>` element. These elements are:

- `<eventName>`

  This element describes the event being queried. There must be one and only one of this element.

  The content of the `<eventName>` is the namespace and name of the event, in this format:

  `<eventName>{`*namespace*`}`*event*`</eventName>`

  For more details on events and event namespaces, see *Events and Dimensions* on page 55.

- `<views>`

This element defines a view on a property or dimension property of the event, or on an aggregate of either.

For details, see *The <views> Element* on page 46.

- `<groups>`

  This element defines grouping on a property or dimension property of the event. Grouping may also be done by period of time.

  For details, see *The <groups> Element* on page 48.

- `<filters>`

  This element defines a filter to be placed on a property or a dimension property of the event.

  For details, see *The <filters> Element* on page 49.

- `<orders>`

  This element defines how you want the results to be ordered. The property used to order the results must be also represented in a `<views>` or `<groups>` element.

  For details, see *The <orders> Element* on page 52.

The following is a complete example Query API SOAP request. This request returns the name and ID of all portlets that have been used in the Analytics Console community, ordered by portlet ID, along with a count of how many times each portlet was used.

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Header/>
<S:Body>
<q:executeResultSetQuery
    xmlns:q="http://www.bea.com/analytics/AnalyticsQueryService">

<arg0>


<eventName>{http://www.bea.com/analytics/ali}portletUses</eventName>

    <views>
        <property>*</property>
        <aggregate>1</aggregate>
    </views>
    <groups>
        <dimension>portlet</dimension>
        <property>name</property>
    </groups>
```

```
    <groups>
        <dimension>portlet</dimension>
        <property>id</property>
    </groups>
    <filters
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <dimension>community</dimension>
        <property>name</property>
       <values  xsi:type="xs:string" >Analytics Console</values>

        <operator>1</operator>
    </filters>
    <orders>
        <dimension>portlet</dimension>
        <property>id</property>
        <isAscending>1</isAscending>
    </orders>
</arg0>

</q:executeResultSetQuery>
</S:Body></S:Envelope>
```

The <views> Element

This topic provides the syntax for the <views> element of a Query API SOAP request.

The <views> element defines a view on a property or dimension property of an event, or on an aggregate of either. For each <views> element there is a column added to the result set. There may be multiple <views> elements.

There are three elements contained by the <views> element:

Table 1: Elements contained by <views>

Element	Description
<dimension>	The name of a dimension associated with the event. Each <views> element may have at most one <dimension> element.

Element	Description

`<property>`	The name of a property associated with the dimension. Each `<views>` element must have one and only one `<property>` element.
	Note:
	There is a special case usage of the `<property>` element:
	`<views>`
	`<property>*</property>`
	`<aggregate>1</aggregate>`
	`</views>`
	This special case results in a count of events that meet the criteria of the rest of the query.

`<aggregate>`	The method of aggregation for this view. This is an optional element and takes an integer value. For details on the values used by the `<aggregate>` element, see the following table.

Table 2: Aggregation types

Value	Description
0	No aggregation. This is the same as omitting the `<aggregate>` element.
1	Count. A count of all distinct properties in the view.
2	Min. The property with the minimum value in the view. For string values, this is the alphabetically earliest property.
3	Max. The property with the maximum value in the view. For string values, this is the alphabetically latest property.

Value	Description
4	Average. An arithmetic average of the properties in the view. This only applies to numeric properties.
5	Sum. The sum total of the properties in the view. This only applies to numeric properties.

The <groups> Element

This topic provides the syntax for the `<groups>` element of a Query API SOAP request.

The `<groups>` element defines a grouping on a property or dimension property of an event, or on a grouping based on a period of time. For each `<groups>` element there is a column added to the result set. There may be multiple `<groups>` elements.

There are three elements contained by the `<groups>` element:

Table 3: Elements contained by `<groups>`

Element	Description
`<dimension>`	The name of a dimension associated with the event. Each `<groups>` element may have one and only one `<dimension>` element.
`<property>`	The name of a property associated with the dimension. Each `<groups>` element must have one and only one `<property>` element.

`<timeGrouping>`	The period of time for this grouping. This is an optional element and takes an integer value. For details on the values used by the `<timeGrouping>` element, see the following table.
	
	Note: When grouping by time, you must set `<dimension>` to `time` and include an empty `<property>` element. For example:
	
	`<groups>`
	`  <dimension>time</dimension>`
	`  <property />`

Element	Description
	<timeGrouping>2</timeGrouping> </groups>

Table 4: Time grouping types

Value	Description
0	No time grouping. This is the same as omitting the <timeGrouping> element.
1	This value is not used.
2	Hour
3	Day
4	Week
5	Month
6	Year

The <filters> Element

This topic provides the syntax for the <filters> element of a Query API SOAP request.

The <filters> element defines a filter to be placed on a property or dimension property of an event. When a <filters> element is defined for a property, only events that meet the criteria of the <filters> element will be returned. There may be multiple <filters> elements.

There are six elements contained by the <filters> element:

Table 5: Elements contained by <filters>

Element	Description
<dimension>	The name of a dimension to be filtered. Each <filters> element may have at most one <dimension> element.

Element	Description
<property>	The name of a property associated with the dimension. Each <filters> element must have one and only one <property> element.
<operator>	The method of comparison to use between the <property> and the <values>. There must be one and only one <operator> element. For details on the values used by the <operator> element, see the following table.
<values>	The values to which each event will be compared to, as dictated by the <operator> element. There may be one or more <values> elements, and they may be of any type. The type must be specified in the attributes of the element. For example: <values xmlns:xs= "http://www.w3.org/2001/XMLSchema" xmlns:xsi= "http://www.w3.org/ 2001/XMLSchema-instance" xsi:type="xs:string"> Reports </values>
<ranking>	The ranking method to use. This element is optional. The values for this element are: • 1 Top ranking • 2 Bottom ranking

Element	Description
<rankingCount>	The number of top or bottom values to return. This element is only required when you use the <ranking> element.

Table 6: Operator types

Value	Description
1	Equals. The event is included in the results if the <property> is equal to the <values>. Only one <values> element may be used.
2	Not equals. The event is included in the results if the <property> is not equal to the <values>. Only one <values> element may be used.
3	Greater than. The event is included in the results if the <property> is greater than the <values>. Only one <values> element may be used.
4	Greater than or equal to. The event is included in the results if the <property> is greater than or equal to the <values>. Only one <values> element may be used.
5	Less than. The event is included in the results if the <property> is less than the <values>. Only one <values> element may be used.
6	Less than or equal to. The event is included in the results if the <property> is less than or equal to the <values>. Only one <values> element may be used.
7	Contains. The event is included in the results if the <property> contains the substring in <values>. Only one <values> element may

Value	Description
	be used. The <property> must be of type string.
8	Does not contain. The event is included in the results if the <property> does not contain the substring in <values>. Only one <values> element may be used. The <property> must be of type string.
9	In. The event is included in the results if the <property> is equal to one of the <values>. Multiple <values> may be used.
10	Not in. The event is included in the results if the <property> is not equal to any of the <values>. Multiple <values> may be used.
11	Starts with. The event is included in the results if the <property> starts with the substring in <values>. Only one <values> element may be used. The <property> must be of type string.
12	Ends with. The event is included in the results if the <property> ends with the substring in <values>. Only one <values> element may be used. The <property> must be of type string.

The <orders> Element

This topic provides the syntax for the <orders> element of a Query API SOAP request.

The <orders> element defines how the results are ordered based on a property or dimension property of an event. There may be multiple <orders> elements.

When using multiple <orders> elements, the primary order of the result set is determined by the first <orders> element. Subsequent <orders> elements further refine the order within the rules of all previous <orders> elements.

There are three elements contained by the <views> element:

Table 7: Elements contained by <orders>

Element	Description
<dimension>	The name of a dimension associated with the event. Each <orders> element may have at most one <dimension> element.
<property>	The name of a property associated with the dimension. Each <orders> element must have one and only one <property> element.
<isAscending>	How to order the rows. This is an optional element and takes an integer value: A value of 1 orders the rows in ascending order, a value of 0 orders the rows in descending order. The default is descending order.

The Query API SOAP Response

This topic provides a description of the components of a Query API SOAP response.

The Basic Response Body

Every Query API SOAP response contains the following static elements:

- <return>, which is contained by
- <executeResultSetQueryResponse>, which is contained by
- the <Body> element of a standard SOAP envelope

All query results from the Query API service will be contained in this response body:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Header/>
<S:Body>

<ns2:executeResultSetQueryResponse
xmlns:ns2="http://www.bea.com/analytics/AnalyticsQueryService">
<return>

</return>
</ns2:executeResultSetQueryResponse>
```

```
</S:Body>
</S:Envelope>
```

The Query Results

The results of the query are contained within the <return> element. These elements are:

- <results>

  This element represents one row of the result set. It contains a <values> element for each column in the result set. Each <values> element can be of any type, and the actual type is specified in the element attributes. When the result set has multiple columns, the <values> elements are in the same sequence as the <columns> elements.

- <columns>

  This element describes one column of the result set.

The following is an example Query API SOAP response. This is a possible response to the request in the request example in *The Query API SOAP Request* on page 44

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Header/>
<S:Body>

<ns2:executeResultSetQueryResponse
xmlns:ns2="http://www.bea.com/analytics/AnalyticsQueryService">
<return>

<results>
    <values
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:type="xs:int">7</values>

    <values
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:type="xs:string">Publisher Administration</values>

    <values
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:type="xs:int">246</values>

    <count>0</count>
```

```
</results>

<columns>count(*)</columns>
<columns>portlet.name</columns>
<columns>portlet.id</columns>

</return>
</ns2:executeResultSetQueryResponse>
</S:Body>
</S:Envelope>
```

Events and Dimensions

This topic provides an overview description of the events and dimensions in Oracle WebCenter Analytics.

An *event* is a record of an action, typically a user action, that has been captured by the Analytics Collector Service. For example, `portletUses` is an event that is captured every time a portlet is used on the Oracle WebCenter Interaction portal.

Each event includes associated data. The `portletUses` event has data about the portlet, the portal community the portlet was accessed from, and the user's browser. The specifics of each of these (the portlet name, the browser version) are called *properties*, while the groupings of data (portlet, community, browser) are called *dimensions*.

For more details on specific dimensions, see:

- *Dimensions in the Analytics Namespace* on page 56
- *Dimensions in the Oracle WebCenter Interaction Namespace* on page 58
- *Dimensions in the Knowledge Directory Namespace* on page 61
- *Dimensions in the Publisher Namespace* on page 63

For more details on specific events, see:

- *Events in the Oracle WebCenter Interaction Namespace* on page 64

Dimensions in the Analytics Namespace

This topic describes the Oracle WebCenter Analytics dimensions defined in the Analytics namespace, `http://www.bea.com/analytics`.

Table 8: users

Name	Description
userID	ID of the user object • Type: string • Length: 255
name	Name of the user object • Type: string • Length: 255
description	Description of the user object • Type: string • Length: 255
loginName	Login name of the user object • Type: string • Length: 255

Table 9: userProperties

Name	Description
name	Name of the property object • Type: string • Length: 255

Name	Description
propertyId	ID of the associated ALI object • Type: string • Length: 255
isDisplayed	Flag representing the visibility of the property object in both the Analytics Console and Analytics Administration • Type: boolean

Table 10: userPropertyValues

Name	Description
propertyId	ID of the associated property object • Type: integer • Length: 8
userId	ID of the associated user object • Type: integer • Length: 8
value	Value of the user property object • Type: string • Length: 255
type	Simple data type of the property value • Type: integer

Name	Description
	• Length: 8

Dimensions in the Oracle WebCenter Interaction Namespace

This topic describes the Analytics dimensions defined in the Oracle WebCenter Interaction namespace, `http://www.bea.com/analytics/ali`.

Table 11: authSources

Name	Description
name	Name of the authentication source object • Type: string • Length: 255

Table 12: browsers

Name	Description
name	Name of the browser (if found) • Type: string • Length: 255
version	Version of the browser (if found) • Type: string • Length: 30
os	Operating system that the browser runs on (if found) • Type: string

Name	Description
	• Length: 100

Table 13: communityPages

Name	Description
name	Name of the authentication source object • Type: string • Length: 255
communityId	ID of the associated community • Type: string • Length: 255

Table 14: communities

Name	Description
name	Name of the authentication source object • Type: string • Length: 255

Table 15: documents

Name	Description
name	Name of the document object • Type: string • Length: 255

Name	Description
title	Title of the document object • Type: string • Length: 255
docDataSourceId	ID of the associated data source • Type: integer • Length: 8

Table 16: groups

Name	Description
name	Name of the user group object • Type: string • Length: 255
description	Description of the user group object • Type: string • Length: 255
authSourceId	ID of the associated authentication source object • Type: integer • Length: 8

Table 17: hosts

Name	Description
ipAddress	IP address of client triggering event • Type: string • Length: 24

Name	Description
hostName	Resolved name of the associated IP address (if an IP can not be resolved, HOSTNAME is marked as “Unknown”) • Type: string • Length: 255

Table 18: portlets

Name	Description
name	Name of the portlet object • Type: string • Length: 255
portletTypeId	ID representing the portlet’s type • Type: integer • Length: 8

Table 19: searchTerms

Name	Description
searchTerm	Search term that was used • Type: string • Length: 255

Dimensions in the Knowledge Directory Namespace

This topic describes the Analytics dimensions defined in the Knowledge Directory namespace, `http://www.bea.com/analytics/knowledgeDirectory`.

Table 20: dataSources

Name	Description
name	Name of the data source object • Type: string • Length: 255

Table 21: documents

Name	Description
dataSourceId	ID of the associated data source • Type: integer • Length: 8
name	Name of the document object • Type: string • Length: 255
title	Title of the document object • Type: string • Length: 1000

Table 22: folders

Name	Description
name	Name of the document folder object • Type: string • Length: 255
parentId	ID of the parent document folder object • Type: integer

Name	Description
	• Length: 8

Dimensions in the Publisher Namespace

This topic describes the Analytics dimensions defined in the Publisher namespace, `http://www.bea.com/analytics/publisher`.

Table 23: folders

Name	Description
name	Name of the Publisher folder object • Type: string • Length: 255
parentId	ID of the parent Publisher folder object (if the folder is the root folder, the PARENTID column contains a NULL value) • Type: integer • Length: 8

Table 24: publishedItems

Name	Description
name	Name of the Publisher content item object • Type: string • Length: 255
folderId	ID of the associated Publisher folder object • Type: integer • Length: 8

Name	Description
url1	Chunked string representing the Publisher content item's published URL • Type: string • Length: 450
url2	Chunked string representing the Publisher content item's published URL • Type: string • Length: 450
url3	Chunked string representing the Publisher content item's published URL • Type: string • Length: 450
url4	Chunked string representing the Publisher content item's published URL • Type: string • Length: 450
url5	Chunked string representing the Publisher content item's published URL • Type: string • Length: 450

Events in the Oracle WebCenter Interaction Namespace

This topic describes the Analytics events defined in the Oracle WebCenter Interaction namespace, `http://www.bea.com/analytics/ali`.

Namespace Prefixes

In the event descriptions in this topic, dimensions are preceded with a namespace prefix. The following table lists each prefix and provides a cross reference to the dimension documentation for that namespace.

Prefix	Documentation
ali	*Dimensions in the Oracle WebCenter Interaction Namespace* on page 58
pub	*Dimensions in the Publisher Namespace* on page 63
kd	*Dimensions in the Knowledge Directory Namespace* on page 61

Events

The tables in this section describe the properties and dimensions of each event.

Table 25: documentViews

Name	Description
document	Dimension: kd:documents
host	Dimension: ali:hosts
browser	Dimension: ali:browsers
searchFactId	• Type: integer • Length: 8
documentTypeId	• Type: integer • Length: 8

Table 26: logins

Name	Description
host	Dimension: ali:hosts

Name	Description
browser	Dimension ali:browsers

Table 27: pageViews

Name	Description
community	Dimension: ali:communities
page	Dimension: ali:communityPages
host	Dimension: ali:hosts
browser	Dimension: ali:browsers
pageType	• Type: integer • Length: 8
responseTime	• Type: float • Length: 20
isEntryPage	• Type: boolean
isExitPage	• Type: boolean

Table 28: portletUses

Name	Description
portlet	Dimension: ali:portlets
host	Dimension: ali:hosts
community	Dimension: ali:communities
page	Dimension: ali:communityPages
browser	Dimension: ali:browsers

Table 29: portletViews

Name	Description
responseTime	• Type: float • Length: 20
portlet	Dimension: ali:portlets
host	Dimension: ali:hosts
community	Dimension: ali:communities
browser	Dimension: ali:browsers

Table 30: searches

Name	Description
searchTerm	Dimension: ali:searchTerms
portlet	Dimension: ali:portlets
community	Dimension: ali:communities
page	Dimension: ali:communityPages
responseTime	• Type: float • Length: 20
abandoned	• Type: boolean
totalMatches	• Type: integer • Length: 8

Generating and Using a Query API Client

This topic describes how to generate a Java client using JAX-WS, and provides code samples for using the generated client.

You can generate platform-specific client code using the Analytics Query API WSDL file. The following steps describe how to generate a Java client using JAX-WS.

Note: To generate a .NET client, use Visual Studio. For more information, see the Microsoft Developers Network.

1. Download and install the latest version of JAX-WS.

   JAX-WS can be found at `https://jax-ws.dev.java.net/`.

2. Use the `wsimport` utility to generate the client code from the Query API WSDL.

   The command is

   ```
   wsimport -keep
   http://analytics_server:11944/analytics/QueryService?wsdl
   ```

   where *analytics_server* is the host of your Analytics installation.

3. Copy the client code to your Java query project.

   The client is generated to the `bin` directory.

4. Add `javxws-ri/lib` to the `classpath` of your project.

The following code snippet is an example of how to use the JAX-WS generated Java client.

```
import java.util.Calendar;
import java.util.GregorianCalendar;
import com.bea.analytics.analyticsqueryservice.*;


. . .

 AnalyticsQueryService service = new AnalyticsQueryService();
 AnalyticsQueryServicePortType port =
service.getAnalyticsQueryServicePort();

. . .

//show the top 10 pages by page view, grouped by day, since 1/1/05,
```

```
 ignoring a page

//set the event type to be queried
 QueryParameters param = new QueryParameters();
 param.setEventName("{http://www.bea.com/analytics/ali}pageViews");

//define what information to return, in this case page view count
 and page id
   View view = new View();
   view.setProperty("*");
   view.setAggregate(1);  // aggregate count
   param.getViews().add(view);

 View idView = new View();
 idView.setDimension("page");
 idView.setProperty("id");
 param.getViews().add(idView);

//filter the results
//in this case ignore a specific page
 Filter filter = new Filter();
 filter.setDimension("page");
 filter.setProperty("id");
 filter.setOperator(2); // operator not equals
 filter.getValues().add(new Integer(518));
 param.getFilters().add(filter);

//in this case ignore pageviews before a certain date
 Filter timefilter = new Filter();
 timefilter.setDimension("time");
 timefilter.setOperator(3); // operator greater than
 Calendar date = new GregorianCalendar();
 date.set(2005, 1, 1);
 timefilter.getValues().add(date.getTime());
 param.getFilters().add(timefilter);

//only show the top 10 pages
 Filter rankingFilter = new Filter();
 rankingFilter.setDimension("page");
 rankingFilter.setRanking(1); // ranking top
 rankingFilter.setRankingCount(10);
 param.getFilters().add(rankingFilter);

//now, group the results
 GroupBy group = new GroupBy();
```

```
 group.setDimension("page");
 group.setProperty("name");
//note: unique "page name" = name + id
 param.getGroups().add(group);
//note: auto adds name to view

 GroupBy group1 = new GroupBy();
 group1.setDimension("time");
 group1.setTimeGrouping(3); // group by day
 param.getGroups().add(group1);

 QueryResults result = port.executeResultSetQuery(param);
 printOutput(result);
```