BEA AquaLogic® Analytics

Query API Usage Guide

Version 2.5
Revised: April 11, 2008

Contents

About This Guide

This documentation describes how to use the Analytics Query API to use Analytics data in applications you develop.

This guide is organized as follows:

- This section provides information on the intended audience of this documentation, typographical conventions used in this guide, and a list of other BEA documentation and resources related to this product.
- *Tutorial: Using the Analytics Query API* on page 11. 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 31. 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 45. This appendix is a reference of the events and dimensions that are defined for the AquaLogic products.
- *Generating and Using a Query API Client* on page 59. 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 the AquaLogic Analytics database. 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	**bold**	• 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	`computer`	Type `Marketing` as the name of your community.
• Variables you enter	*`italic computer`*	Enter the base URL for the Remote Server. For example, `http://`*`my_computer`*`.`
• New terms • Emphasis • Object example names	*italic*	• *Portlets* are web tools embedded in your portal. • The URL *must* be a unique number.

Convention	Typeface	Examples/Notes
		• The example Knowledge Directory displayed in Figure 5 shows the *Human Resources* folder.

BEA Documentation and Resources

The following documentation and resources are available from BEA.

Table 1: Documentation

Resource	Description
Installation Guide	This guide describes the prerequisites (such as required software) and procedures for installing AquaLogic Analytics. It is available on *edocs.bea.com/alui/analytics/docs25/*.
Administrator Guide	This guide describes how to manage, maintain, and troubleshoot Analytics. It is available on *edocs.bea.com/alui/analytics/docs25/*.
Release Notes	The release notes provide information about new features, issues addressed, and known issues in the release. They are available on*edocs.bea.com/alui/analytics/docs25/* and on any physical media provided for delivering the application.
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.

Resource	Description
Deployment Guide	This guide is written for business analysts and system administrators. It describes how to plan your AquaLogic User Interaction deployment. It is available on *edocs.bea.com/alui/deployment/index.html*.

Table 2: Other Resources

Resource	Description
Developer Guides, Articles, API Documentation, Blogs, Newsgroups, and Sample Code	These resources are provided for developers on the BEA dev2dev site (*dev2dev.bea.com*). They describe how to build custom applications using AquaLogic User Interaction and how to customize AquaLogic User Interaction products and features.
AquaLogic User Interaction (ALUI) and AquaLogic Business Process Management (ALBPM) Support Center	The ALUI and ALBPM Support Center is a comprehensive repository for technical information on ALUI and ALBPM products. From the Support Center, you can access products and documentation, search knowledge base articles, read the latest news and information, participate in a support community, get training, and find tools to meet most of your ALUI and ALBPM-related needs. The Support Center encompasses the following communities: **Technical Support** Submit online service requests, check the status of your requests, search the knowledge base, access documentation, and download maintenance packs and hotfixes. **User Group** Participate in user groups; view webinars, presentations, the CustomerConnection newsletter, and the Upcoming Events calendar. **Product Center** Download product updates, maintenance packs, and patches; view the Product Interoperability matrix (supported third-party products and interoperability between products). **Developer Center**

Resource	Description
	Download developer tools, view code samples, access technical articles, and participate in discussions. **Education Services** Review the available education options, then choose courses by role and delivery method (Live Studio, Public Classroom Training, Remote Classroom, Private Training, or Self-Paced eLearning). **Profile Center** Manage your implementation details, local user accounts, subscriptions, and more. If you do not see the Support Center when you log in to *one.bea.com/support*, contact *ALUISupport@bea.com* or *ALBPMSupport@bea.com*for the appropriate access privileges.
Technical Support	If you cannot resolve an issue using the above resources, BEA Technical Support is happy to assist. Our staff is available 24 hours a day, 7 days a week to handle all your technical support needs. E-mail: *ALUISupport@bea.com* or *ALBPMSupport@bea.com* Phone Numbers: USA, Canada +1 866.262.7586 or +1 415.263.1696 EMEA +44 1494 559127 Asia Pacific +61 2.9931.7822 Australia/NZ +61 2.9923.4030 Singapore +1 800.1811.202

CHAPTER 2

Tutorial: Using the Analytics Query API

This chapter 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:

- AquaLogic Analytics 2.5
- 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 describe the tutorial at a high level:

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 12.

2. Learn how to create SOAP queries to view which portlets are being used on your portal.

   For details, see *Viewing Portlet Usage* on page 19.

3. Learn how to use filters to refine your queries.

   For details, see *Filtering Portlet Usage by Community* on page 23.

4. Learn how to view events based on periods of time.
   For details, see *Tracking Portlet Usage* on page 26.

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 we write 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 we are interested in to the console.

   You are interested in the elements contained in the `<return>` element of the SOAP response, which are described in the following table:

Element	**Description**
`<results>`	We see one `<results>` element for each row of data our request generates. Each `<results>` element contains one or more `<values>` elements.
`<values>`	In each `<results>` element, there is one `<values>` element for each type of data we request.
`<columns>`	The `<columns>` element describes the type or types of data we 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 AquaLogic 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 our example, the namespace is `http://www.bea.com/analytics/ali` and the event is `portletUses`. Our 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 we 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, we 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 35.

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 35.

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 we 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 36.

   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 19, 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 38.

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 38.

The type of the `<values>` element must be defined. Because of this, we 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 our 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 19, 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 we just wrote will give us a count of events that match each distinct value of the grouped property.

Our 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, you 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 36.

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
```

CHAPTER 3

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* on page 31.
- A description of a Query API SOAP request, including specific details about query parameters.

  For details, see *The Query API SOAP Request* on page 32.
- A description of a Query API SOAP response.

  For details, see *The Query API SOAP Response* on page 42.

Analytics Query API Configuration

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`

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

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 45.

- `<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 35.

- `<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 36.

- `<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 38.

- `<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 41.

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 3: 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.

`<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 4: 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.
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 5: 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.

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

Table 6: 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 7: 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.
`<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=`

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

Table 8: 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

Value	Description
	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 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

Value	Description
	`<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 9: 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 32

```
<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>
```

APPENDIX A

Events and Dimensions

This topic provides an overview description of the events and dimensions in AquaLogic 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 AquaLogic 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 46
- *Dimensions in the ALI Namespace* on page 48
- *Dimensions in the Knowledge Directory Namespace* on page 51
- *Dimensions in the Publisher Namespace* on page 53

For more details on specific events, see:

- *Events in the ALI Namespace* on page 55

Dimensions in the Analytics Namespace

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

Table 10: 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 11: 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 12: 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 ALI Namespace

This topic describes the Analytics dimensions defined in the ALI (AquaLogic Interaction) namespace, `http://www.bea.com/analytics/ali`.

Table 13: authSources

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

Table 14: 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)

Name	Description
	• Type: string • Length: 100

Table 15: 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 16: communities

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

Table 17: 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 18: 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 19: 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 20: 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 21: 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 22: dataSources

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

Table 23: 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 24: 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 25: 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 26: 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

Name	Description
	• Length: 8
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 ALI Namespace

This topic describes the Analytics dimensions defined in the ALI (AquaLogic 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 ALI Namespace* on page 48
pub	*Dimensions in the Publisher Namespace* on page 53
kd	*Dimensions in the Knowledge Directory Namespace* on page 51

Events

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

Table 27: documentViews

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

Name	Description
	• Length: 8

Table 28: logins

Name	Description
host	Dimension: ali:hosts
browser	Dimension ali:browsers

Table 29: 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 30: portletUses

Name	Description
portlet	Dimension: ali:portlets

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

Table 31: 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 32: 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

Name	**Description**
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 .NETclient, 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);
```