ClientVantage Agentless Monitoring

Web Services for Developers

Getting Started Guide

Release 11.1

Please direct questions about ClientVantage Agentless Monitoring or comments on this document to:

Technology Customer Support
Compuware Corporation
Customer Support Hotline
1-800-538-7822
FrontLine Support Web Site:
http://frontline.compuware.com

For telephone numbers in other geographies, see the list of worldwide offices at http://www.compuware.com.

Build: October 16, 2009, 2:36

Contents

CHAPTER 1

Web Services Overview

A Web service is a software component or service that has been Web-published using an HTTP-compliant message encoding protocol such as XML-RPC, Representational State Transfer (*REST*), or Simple Object Access Protocol (*SOAP*).

Web services enable developers to target a range of clients and build the services from local and remote resources. XML-RPC, REST, and SOAP codify the existing practice of using XML and HTTP together as an application protocol.

Specifications and references

XML-RPC

Representational State Transfer (REST)

SOAP

In the case of Web services published by ClientVantage Agentless Monitoring (CVAM), SOAP is used to package XML transferred over HTTP.

ClientVantage Agentless Monitoring uses Apache Axis 1.4 as a SOAP engine for communication between service providers and consumers.

For more information, see *http://www.w3.org/TR/SOAP/* and *http://ws.apache.org/axis/*

Web Services Description Language (WSDL)

Web Services Description Language (WSDL) is used to define Web services published on a particular server and describe how to access them. WSDL is an XML-based language.

For more information, see *http://www.w3.org/TR/wsdl*

CHAPTER 2

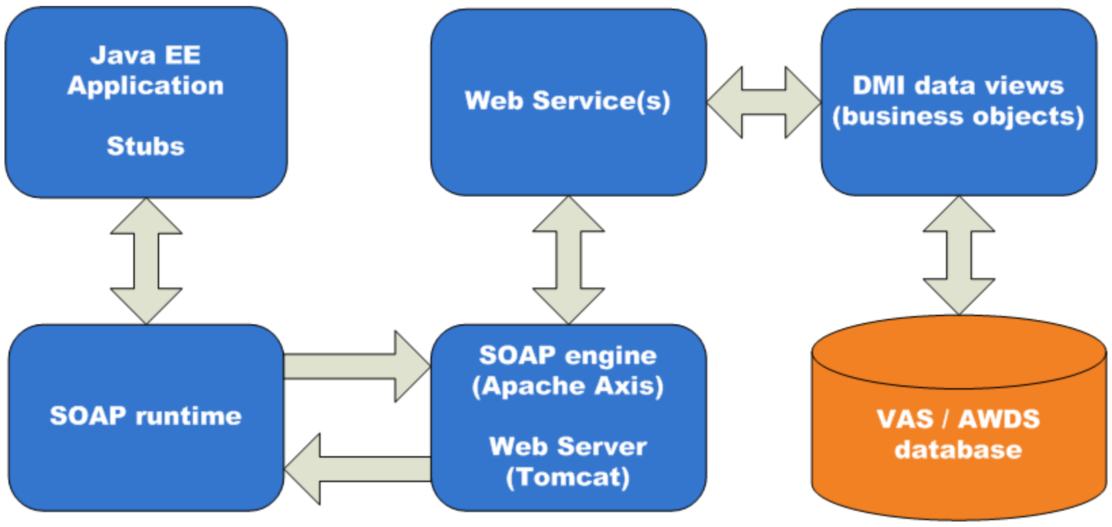
ClientVantage Agentless Monitoring Web Services Architecture

This diagram depicts the architecture of the CVAM Web services.

ClientVantage Agentless Monitoring uses Apache Axis 1.4 as a SOAP engine for communication between service providers and consumers. Apache Axis supports SOAP 1.1.

Business objects that can be embedded in your application or portal are based on the Data Mining Interface (DMI), a proprietary interface to CVAM databases. DMI allows CVAM users to build custom reports and integrate CVAM data with their applications or portals.

Figure 1. ClientVantage Agentless Monitoring Web service architecture diagram.



CHAPTER 3

Using Web Services

This guide assumes that you are familiar with development of Web services, SOAP, XML, and Java EE. Note that you do not have to use Apache Axis to call Web service methods exposed by ClientVantage Agentless Monitoring Web services. Use of Apache Axis simplifies your application development but is not obligatory.

Software Prerequisites

It is assumed that you have properly configured a Java application development environment, including the software components listed here.

- A running installation of VAS or AWDS 11.1 or higher.
- JDK 1.5 or higher.
- Apache Tomcat 5.5 or higher.
- Apache Axis 1.4.

  Note that this software package is required for examples presented in this guide. Otherwise, Apache Axis is optional.

Available Web Services

AWDS and VAS use basic authentication to access the published Web services. ClientVantage Agentless Monitoring 11.1 offers the Web services described here.

DMIService

This service exposes methods to retrieve data regarding monitored traffic. Communication uses the RPC-encoded style of WSDL. For example, for the `getDataSources` method, a part of its definition is as follows:

```
<wsdl:message name="getDataSourcesRequest">
    <wsdl:part name="appId" type="soapenc:string"/>
    <wsdl:part name="viewId" type="soapenc:string"/>
</wsdl:message>

<wsdl:portType name="DMIService">
```

```
    <wsdl:operation name="getDataSources" parameterOrder="appId viewId">
        <wsdl:input message="impl:getDataSourcesRequest"
name="getDataSourcesRequest"/>
        <wsdl:output message="impl:getDataSourcesResponse"
name="getDataSourcesResponse"/>
    </wsdl:operation>
…
</wsdl:portType>
…
<wsdl:binding… />
```

Note that WSDL style also implies how SOAP envelopes are formed.

DMIServiceWL

This service exposes the same methods as `DMIService`, but uses document literal style of WSDL. Similar fragment of the method definition is as follows:

```
<wsdl:message name="getDataSourcesRequest">
    <wsdl:part element="impl:getDataSources" name="parameters"/>
</wsdl:message>
<wsdl:portType name="DMIServiceWL">
    <wsdl:operation name="getDataSources">
        <wsdl:input message="impl:getDataSourcesRequest"
name="getDataSourcesRequest"/>
        <wsdl:output message="impl:getDataSourcesResponse"
name="getDataSourcesResponse"/>
    </wsdl:operation>
…
</wsdl:portType>
…
<wsdl:binding… />
```

Note that WSDL style also implies how SOAP envelopes are formed.

DataServerList

This service contains methods to get the list of DMI data servers (data sources).

Version

This is an Axis service to get the version number of Apache Axis used in VAS or AWDS. ClientVantage Agentless Monitoring 11.1 uses Apache Axis 1.4.

Obtaining WSDL for Deployed Web Services

To obtain WSDL for deployed Web services, follow this example.

Prerequisites

We assume, for this example, that you have a VAS installation running on a server with the IP address `10.1.1.10`, on port `80`, and you want to use the `DMIService` service in your application.

To get the definition of Web services:

1. In your Web browser, open the URL `http://10.1.1.10/services`
2. Select the `wsdl` link next to the `DMIService` or just open the link `http://10.1.1.10/services/DMIService?wsdl`

Your Web browser will display a WSDL definition of `DMIService` as shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://ws.delta.adlex"
xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://ws.delta.adlex"
xmlns:intf="http://ws.delta.adlex"
```

```
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->
 <wsdl:types>
  <schema targetNamespace="http://ws.delta.adlex"
xmlns="http://www.w3.org/2001/XMLSchema">
   <import namespace="http://xml.apache.org/xml-soap"/>
   <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
   <complexType name="ArrayOfArrayOf_soapenc_string">
    <complexContent>
     <restriction base="soapenc:Array">
      <attribute ref="soapenc:arrayType" wsdl:arrayType="soapenc:string[][]"/>
     </restriction>
    </complexContent>
   </complexType>

...

   </wsdl:portType>
   <wsdl:binding name="DMIServiceSoapBinding" type="impl:DMIService">
      <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="getServerUUID">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="getServerUUIDRequest">
            <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://ws.delta.adlex" use="encoded"/>
         </wsdl:input>
         <wsdl:output name="getServerUUIDResponse">
            <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://ws.delta.adlex" use="encoded"/>
         </wsdl:output>
      </wsdl:operation>
...

   </wsdl:binding>
   <wsdl:service name="DMIServiceService">
      <wsdl:port binding="impl:DMIServiceSoapBinding" name="DMIService">
         <wsdlsoap:address location="http://10.1.1.1/services/DMIService"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

Generating Stubs and Data Types from WSDL

To generate stub classes and object mappings for your application, save a WSDL definition of a service on your local disk in the working directory, define the environment variables required by Apache Axis, and run a converter Java class with the name of the service as an input parameter.

For example:

1. Save `http://10.1.1.10/services/DMIService?wsdl` as `DMIService.wsdl`
2. From the command prompt, run the commands as in the following example: *(Note the line continuation characters '÷')*

```
set AXIS_LIB=AXIS/lib/
set AXISCLASSPATH=%AXIS_LIB%wsdl4j-1.5.1.jar;%AXIS_LIB%axis.jar; ÷
%AXIS_LIB%axis-ant.jar;%AXIS_LIB%commons-discovery-0.2.jar; ÷
%AXIS_LIB%commons-logging-1.0.4.jar;%AXIS_LIB%jaxrpc.jar; ÷
%AXIS_LIB%log4j-1.2.8.jar;%AXIS_LIB%saaj.jar;%ATTPATH%
java -cp %AXISCLASSPATH% org.apache.axis.wsdl.WSDL2Java -T 1.2 ÷
```

```
    -o .\DMIService DMIService.wsdl
```

The `org.apache.axis.wsdl.WSDL2Java` class will generate stubs and object mappings for use in your application.

The folder defined during the conversion (in the example above, it will be `DMIService`) will list the following classes:

1. `DMIService.java`

   Object representing an interface describing all the methods found in the WSDL.
2. `DMIData.java`

   Return object of the `DMIService.getDMIData` method.
3. `DMIServiceError.java`

   One of the fields of the `DMIData` object
4. `DMIServiceSoapBindingStub.java`

   Implementation of DMIService object (extends `org.apache.axis.client.stub` class).
5. `DMIServiceService.java`

   Interface allowing to implement `DMIService`.
6. `DMIServiceServiceLocator.java`

   Interface making it possible to implement `DMIService` with additional parameters.

Creating Your Application

This example introduces the usage of CVAM Web services. It defines the server URL, authenticates the user (if basic authentication exists), calls out example methods, and displays the result.

Prerequisites

This example uses Java technology for developing an application. A Java development environment should be configured for this application to execute properly.

To create a sample application that uses CVAM Web services:

1. Create a Java class package.

```
package examples;

import java.net.MalformedURLException;
import java.net.URL;
import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import org.apache.axis.client.Stub;
import adlex.delta.ws.DMIData;
import adlex.delta.ws.DMIService;
import adlex.delta.ws.DMIServiceError;
import adlex.delta.ws.DMIServiceServiceLocator;

public class DMIServiceClient {

  static String appId = "CVENT";
  static String viewId;
  static String dataSourceId;
  static String[] dimensionIds;
  static String[] metricIds;
  static String[][] dimFilters;
  static String[][] metricFilters;
  static String[][] sort;
  static Integer top;
  static String resolution;
  static String timePeriod;
  static Integer numberOfPeriods;
  static Long timeBegin;
  static Long timeEnd;
  static Long timeout;
```

2. Define and test the URL.

   Be sure to provide an active user account for the server you work with.

```
/**
  * @param args
  */
 public static void main(String[] args) {

//  create locator
  DMIServiceServiceLocator locator = new DMIServiceServiceLocator();
  try {
   String defURL = "http://10.1.1.1:8081";
   String serviceURL = defURL + "/services/DMIService";
   String user = "testuser";
   String password = "testpassword";
```

3. Create a `DMIService` object.

```
DMIService service = locator.getDMIService(new URL(serviceURL));
```

   For more information, see Item 1 [p. 12].

4. Authenticate the connection to the server.

```
((Stub)service).setUsername(user);
((Stub)service).setPassword(password);
```

5. Add the `getServerUUID` method.

```
System.out.print("server UUID=");
System.out.println(service.getServerUUID());
```

6. Add the `getResolutions` method.

```
viewId= "ClientView"; //awds
String[] resolutions = service.getResolutions(appId, viewId);
for (String string : resolutions) {
     System.out.println(string);
}
```

7. Add the `getDataViews` method.

```
String[][] dataViews = service.getDataViews(appId);
   for (int I = 0; I < dataViews.length; I++) {
    String[] app = dataViews[i];
    System.out.println("-------");
    System.out.println(app[0]);
    System.out.println(app[1]);
    System.out.println("-------");
}
```

8. Add the `getDMIData` method.

   a) Set the `getDMIData` criteria.

```
setCriteria();
```

The criteria are defined in a separate definition list located at the end of this example. For more information, see Step 10 [p. 15].

b) Add the `getDMIData` method.

```
viewId= "ClientView";
   DMIData data = service.getDMIData(appId, viewId, dataSourceId, dimensionIds,

     metricIds, dimFilters, metricFilters, sort, top,
     resolution, timePeriod, numberOfPeriods, timeBegin, timeEnd, timeout);

   if (data.isTimeout()) {
    System.out.println("timeout");
    return;
   }
   showErrors(data); //displays results
   showResults(data); //displays results
   viewId= "CVCache";
   data = service.getDMIData(appId, viewId, dataSourceId, dimensionIds,
     metricIds, dimFilters, metricFilters, sort, top,
     resolution, timePeriod, numberOfPeriods, timeBegin, timeEnd, timeout);
   if (data.isTimeout()) {
    System.out.println("timeout");
    return;
   }
   showErrors(data);
   showResults(data);
  } catch (ServiceException e) {
   e.printStackTrace();
  } catch (RemoteException e) {
   e.printStackTrace();
  } catch (MalformedURLException e) {
   e.printStackTrace();
  }
 }
```

9. Define the format and display the result.

a) Add the `showResults` method.

```
/**
  * @param data
  */
 private static void showResults(DMIData data) {
  System.out.println("columnHeader");
  String[] columnHeader = data.getColumnHeader();
  for (int I = 0; I < columnHeader.length; I++) {
   System.out.println(columnHeader[i]);
  }
  System.out.println("formattedData");
  String[][] formattedData = data.getFormattedData();
  for (int I = 0; I < formattedData.length; I++) {
   System.out.println("--row no:" + I);
   String[] fdata = formattedData[i];
   for (int j = 0; j < fdata.length; j++) {
    System.out.println(fdata[j]);
   }
  }
 }
```

b) Add the `showErrors` method.

```
/**
  * @param dmiServiceError
  */
 private static boolean showErrors(DMIData data) {
  DMIServiceError[] dmiServiceError = data.getDmiServiceError();
  if (dmiServiceError== null) {
   return false;
  }
  for (int I = 0; I < dmiServiceError.length; I++) {
   DMIServiceError error = dmiServiceError[i];
   String[] descriptions = error.getErrorDescriptions();
   if (descriptions!= null) {
    System.out.println("-errors");
    for (int j = 0; j < descriptions.length; j++) {
     System.out.println(descriptions[j]);
    }
   }
   descriptions = error.getWarningDescriptions();
   if (descriptions!= null) {
    System.out.println("-warnings");
    for (int j = 0; j < descriptions.length; j++) {
     System.out.println(descriptions[j]);
    }
   }
   descriptions = error.getInfoDescriptions();
   if (descriptions!= null) {
    System.out.println("-info");
    for (int j = 0; j < descriptions.length; j++) {
     System.out.println(descriptions[j]);
    }
   }
  }
  return true;
 }
```

10. Define the criteria used with `getDMIData`.

```
private static void setCriteria() {
  appId = "CVENT"; //vas
  viewId= "ClientView"; //awds
  resolution = "r";
  dataSourceId = "localhost";
  dimensionIds = new String [] {  "begT", "appl"};
  metricIds = new String []{"svrDelay", "netDelay", "abortspc"};
  dimFilters = null;
  metricFilters = null;
  sort = null;
  top =10;
  resolution = "r";
  timePeriod = "1P";
  numberOfPeriods = null;
  timeEnd =  null;
  timeBegin = null;
  timeout = 60000L    ;
 }
}
```

11. Execute the above example and observe the result.

Keeping Track of Web Service Use

VAS and AWDS keep track of DMI Web service use. All calls via Web services are recorded in the Web server log located in the `log/ncsa` directory under your report server's installation folder. Note that this is where *all* Web server logs are stored, so look for records related to Web service in the log files.

Each record includes the method name and its parameters. Based on this information, a diagnostic report has been designed to give a summary of method calls for today and a detailed log of calls for the past hour. This report is available from the **Tools** → **Diagnostics** → **DMI Web Service Activity** menu.

Testing Your Application

To test and verify your Web service calls, use the **DMI Web Service Activity** report. In addition, you can use Axis TCP Monitor and SOAP Monitor for detailed analysis of your SOAP calls.

For more information on these tools, see *Axis User's Guide* at *http://ws.apache.org/axis/java/user-guide.html*.

Using the Axis TCP Monitor (tcpmon)

The **`tcpmon`** utility can be found in the `org.apache.axis.utils` package. It is a tool to monitor SOAP communication between your local computer, a Web service client, and a Web service provider. A batch file located in the VAS distribution media (`tcp_monitor.bat`) located in the `/tool/` directory will also execute the monitor program.

Using the SOAP Monitor

SOAP Monitor is another tool that enables Web service developers to monitor the content of SOAP messages. Note that, for security reasons, SOAP Monitor is not enabled by default.

CHAPTER 4

Data Integration with Your Application

CVAM Web services enable developers to access data from VAS or AWDS by means of data views.

A data view is a predefined perspective on data stored in the VAS or AWDS database.

Developers do not need to know all database internals and relationships between tables to use data views. From a developer's perspective, a data view is a single table containing data that can be retrieved for further analysis.

Data views are primarily used in DMI for defining queries. CVAM Web services publish data from available data views, so using VAS or AWDS data for presentation in third-party portals or applications is very similar to defining a DMI query and displaying the query results. For more information, see *Defining Report Conditions* in the *Data Mining Interface (DMI) – User Guide*.

Example Using DMIServiceClient with a Loop

To periodically check for and collect new data, you can implement a loop for the `getDMIData` method.

The Java example described earlier illustrates how to use Web Services methods in an application, but it focuses only on a single access to the available data. Adding a loop as shown in the example enables repeated checking.

Note that the timeout variable is defined in the criteria definition located at the end of the example. For more information, see Step 10 [p. 15].

Example 1. The DMIServiceClient method usage with a loop

```
/* set parameters of getDMIData */
   setCriteria();
   viewId= "ClientView";
   long formerLastSampleTime = 0;
   long lastSampleTime = service.getLastSampleTime(appId, viewId, resolution);
   int count = 0;
   boolean go = true;
   while(go) {
    System.out.println("-lastSampleTime=" + lastSampleTime);
    if (formerLastSampleTime != lastSampleTime) {
     formerLastSampleTime = lastSampleTime;
     count ++;
     DMIData data = service.getDMIData(appId, viewId, dataSourceId, dimensionIds,
       metricIds, dimFilters, metricFilters, sort, top,
       resolution, timePeriod, numberOfPeriods, timeBegin, timeEnd, timeout);
     if (data.isTimeout()) {
      System.out.println("timeout");
      return;
     }
     showErrors(data);
     showResults(data);
    } else {
     System.out.println("--------");
    }
    Thread.sleep(1000*60*5); //check every 5 minutes
    System.out.println("--wake up");
    lastSampleTime = service.getLastSampleTime(appId, viewId, resolution);
   }
  } catch (ServiceException e) {
   e.printStackTrace();
  } catch (RemoteException e) {
   e.printStackTrace();
  } catch (MalformedURLException e) {
   e.printStackTrace();
  } catch (InterruptedException e) {
   e.printStackTrace();
  }
 }
```

Example Using Web Services with JSP

JSP technology makes it possible to combine Web forms and output formatting with the method calls to the Web services.

The following example consists of four files:

`form.jsp`
: A basic Web form that makes it possible to specify parameters and options to retrieve DMI data, dimensions, metrics, and retrieving values of a dimension for use in a report.

`clientDMIData.jsp`
: A table displaying the **Parameters**, **Timeout**, **Errors**, and **Results** defined in a main form.

`clientDimensionsMetrics.jsp`
: A table displaying the **Parameters**, **Dimensions**, and **Metrics** defined in a main form.

`clientReport.jsp`
: A report displaying a predefined report from the server. Optionally, it generates links to that report on the server.

Example 2. Example of using `DMIService` with JSP

This example demonstrates the usage of methods described in Creating Your Application [p. 12].

File: `form.jsp`

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
```

```
<title>DMIService client examples - FORM</title>
</head>
<body>

<strong>Example of retrieving data</strong>

<form action="clientDMIData.jsp">
<table>

 <tr>
  <td>serviceURL</td>
  <td><select name="serviceURL">
   <option>http://localhost:8081/services/DMIService</option>
   <option selected="selected">http://localhost/services/DMIService</option>
  </select></td>
 </tr>

...

 <tr>
  <td>dataSourceId</td>
  <td><input type="text" name="dataSourceId" value="localhost" /></td>
 </tr>
 <tr>
  <td><br />
  </td>
 </tr>

 <tr>
  <td>dimensionIds</td>
  <td><input type="text" name="dimensionIds" value="begT" /></td>
 </tr>
 <tr>
  <td>dimensionIds</td>
  <td><input type="text" name="dimensionIds" value="appl" /></td>
 </tr>

...

 <tr>
  <td>top</td>
  <td><input type="text" name="top" value="10" /></td>
 </tr>
 <tr>
  <td>resolution</td>
  <td><input type="text" name="resolution" value="r" /></td>
 </tr>
 <tr>
  <td>resolution</td>
  <td><input type="text" name="timePeriod" value="1P" /></td>
 </tr>

...

<br>
<input type="submit" value="DMIData" /></form>

<hr>
<strong>Example of retrieving dimensions and metrics</strong>
<%-------------------------------------------------------------%>
<form action="clientDimensionsMetrics.jsp"><br>

<table>

 <tr>
  <td>serviceURL</td>
  <td><select name="serviceURL">
   <option>http://localhost:8081/services/DMIService</option>
   <option selected="selected">http://localhost/services/DMIService</option>
  </select></td>
 </tr>

 ...

 <tr>
```

```
  <td>appId</td>
  <td><input type="text" name="appId" value="CVENT" /></td>
 </tr>

...

 <tr>
  <td>dataSourceId</td>
  <td><input type="text" name="dataSourceId" value="localhost" /></td>
 </tr>
 <tr>
  <td>resolution</td>
  <td><input type="text" name="resolution" value="r" /></td>
 </tr>
</table>
<br>
<input type="submit" value="Dimensions and metrics" /></form>

<%------------------------------------------------------------%>
<hr>
<strong>Example of retrieving values of dimension and using it in a report</strong>

<form action="clientReport.jsp"><br>
<table>

 <tr>
  <td>VAS URL</td>
  <td><select name="vasURL">
   <option>http://localhost:8081</option>
   <option selected="selected">http://localhost</option>
  </select></td>
 </tr>

...

 <tr>
  <td>viewId</td>
  <td><select name="viewId">
   <option selected="selected">ClientView</option>
   <option >CVCache</option>
  </select></td>
 </tr>
 <tr>
  <td>dataSourceId</td>
  <td><input type="text" name="dataSourceId" value="localhost" /></td>
 </tr>
 <tr>
  <td>dimId</td>
  <td><input type="text" name="dimId" value="appl" /></td>
 </tr>

...

 <tr>
  <td>filter</td>
  <td><input type="text" name="filter" value="" /></td>
 </tr>
</table>
<br>
<input type="submit" value="Report with filter" /></form>

</body>
</html>
```

File: `clientDMIData.jsp`

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>

<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@page import="java.net.MalformedURLException"%>
<%@page import="java.net.URL"%>
<%@page import="java.rmi.RemoteException"%>
<%@page import="javax.xml.rpc.ServiceException"%>
<%@page import="org.apache.axis.client.Stub"%>
```

```
<%@page import="adlex.delta.ws.DMIData"%>
<%@page import="adlex.delta.ws.DMIService"%>
<%@page import="adlex.delta.ws.DMIServiceServiceLocator"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>DMIService client examples - DMI Data</title>
</head>
<body>
<br>
<form action="form.jsp"><input type="submit" action="form.jsp"
 value="Back to form"></form>
<br>
<strong>Parameters:</strong>
<br />
<table border="1">
 <c:forEach var="map_entry" items="${param}">
  <tr>
   <td><strong><c:out value="${map_entry.key}" /></strong></td>
   <td><c:out value="${map_entry.value}" /><br>
   </td>
  </tr>
 </c:forEach>
</table>
<br>

<%
/* connection params */
serviceURL = request.getParameter("serviceURL");
user = request.getParameter("user");
password = request.getParameter("password");

/* set criteria */
appId = request.getParameter("appId");
viewId= request.getParameter("viewId");
resolution = request.getParameter("resolution");
dataSourceId = request.getParameter("dataSourceId");

dimensionIds = request.getParameterValues("dimensionIds");
metricIds = request.getParameterValues("metricIds");

dimFilters = null;
metricFilters = null;
sort = null;
resolution = request.getParameter("resolution");

String s = request.getParameter("top");
if (s!= null && s.trim().length()>0) {
 top = new Integer(s);
}

//timePeriod = "3MO";
timePeriod = request.getParameter("timePeriod");
s = request.getParameter("numberOfPeriods");
if (s!= null && s.trim().length()>0) {
 numberOfPeriods = new Integer(s);
}
s = request.getParameter("timeBegin");
if (s!= null && s.trim().length()>0) {
 timeBegin = new Long(s);
}
 s = request.getParameter("timeEnd");
if (s!= null && s.trim().length()>0) {
 timeEnd = new Long(s);
}

s = request.getParameter("timeout");
if (s!= null && s.trim().length()>0) {
 timeout = new Long(s);
}
%>
```

```
<jsp:useBean id="dmiData" class="adlex.delta.ws.DMIData">

<%
DMIData dmiD = getDMIData();
dmiData.setTimeout(dmiD.isTimeout());
dmiData.setFormattedData(dmiD.getFormattedData());
dmiData.setDmiServiceError(dmiD.getDmiServiceError());
dmiData.setColumnHeader(dmiD.getColumnHeader());
%>
</jsp:useBean>

<strong>Timeout: </strong>
<c:out value="${dmiData.timeout}" />
<br />

<strong>Errors: </strong>
<table border="1">

 <br>
 <c:forEach var="row" items="${dmiData.dmiServiceError}">
  <tr>

   <c:forEach var="value" items="${row.errorDescriptions}">
    <td><strong><c:out value="${value}" /></strong>:</td>
   </c:forEach>

   <c:forEach var="value" items="${row.infoDescriptions}">
    <td><strong><c:out value="${value}" /></strong>:</td>
   </c:forEach>

   <c:forEach var="value" items="${row.warningDescriptions}">
    <td><strong><c:out value="${value}" /></strong>:</td>
   </c:forEach>
  </tr>
 </c:forEach>
</table>

<br>
<strong>Results: </strong>
<table border="1">
 <tr>
  <c:forEach var="value" items="${dmiData.columnHeader}">
   <th><strong><c:out value="${value}" /></strong></th>
  </c:forEach>
 </tr>

 <c:forEach var="row" items="${dmiData.formattedData}">
  <tr>
   <c:forEach var="value" items="${row}">
    <td><c:out value="${value}" /> &nbsp;</td>
   </c:forEach>
  </tr>
 </c:forEach>
</table>

<%!
String serviceURL;
String user;
String password;
String appId ;
String viewId;
String dataSourceId;
String[] dimensionIds;
String[] metricIds;
String[][] dimFilters;
String[][] metricFilters;
String[][] sort;
Integer top;
String resolution;
String timePeriod;
Integer numberOfPeriods;
Long timeBegin;
Long timeEnd;
Long timeout;
```

```
 public DMIData getDMIData() {
//  create locator
  DMIData ret = new DMIData();
  DMIServiceServiceLocator locator = new DMIServiceServiceLocator();
  try {
   DMIService service = locator.getDMIService(new URL(serviceURL));
//   set authentication
   ((Stub)service).setUsername(user);
   ((Stub)service).setPassword(password);
//   call method
   ret = service.getDMIData(appId, viewId, dataSourceId, dimensionIds,
     metricIds, dimFilters, metricFilters, sort, top,
     resolution, timePeriod, numberOfPeriods, timeBegin, timeEnd, timeout);
   return ret;
  } catch (ServiceException e) {
   e.printStackTrace();
  } catch (RemoteException e) {
   e.printStackTrace();
  } catch (MalformedURLException e) {
   e.printStackTrace();
  }
  return ret;
 }

%>
</body>
</html>
```

File: `clientDimensionsMetrics.jsp`

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>

<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@page import="java.net.MalformedURLException"%>
<%@page import="java.net.URL"%>
<%@page import="java.rmi.RemoteException"%>
<%@page import="javax.xml.rpc.ServiceException"%>
<%@page import="org.apache.axis.client.Stub"%>
<%@page import="adlex.delta.ws.DMIService"%>
<%@page import="adlex.delta.ws.DMIServiceServiceLocator"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>DMIService client examples - dimensions and metrics</title>
</head>
<body>
<br>
<form action="form.jsp"><input type="submit" action="form.jsp"
 value="Back to form"></form>
<br>
<strong>Parameters:</strong>
<br />
<table border="1">
 <c:forEach var="map_entry" items="${param}">
  <tr>
   <td><strong><c:out value="${map_entry.key}" /></strong></td>
   <td><c:out value="${map_entry.value}" /><br>
   </td>
  </tr>
 </c:forEach>
</table>
<br>

<%
/* connection params */
serviceURL = request.getParameter("serviceURL");
user = request.getParameter("user");
password = request.getParameter("password");

/* set criteria */
appId = request.getParameter("appId");
```

```
viewId= request.getParameter("viewId");
resolution = request.getParameter("resolution");
dataSourceId = request.getParameter("dataSourceId");
%>

<jsp:useBean id="dmiBean" class="examples.DMIBean">
 <%
 getDimensions(dmiBean);
%>
</jsp:useBean>

<strong>Dimensions: </strong>
<table border="1">
 <c:forEach var="row" items="${dmiBean.dimensions}">
  <tr>
   <c:forEach var="value" items="${row}">
    <td><c:out value="${value}" /> &nbsp;</td>
   </c:forEach>
  </tr>
 </c:forEach>
</table>
<strong>Metrics: </strong>
<table border="1">
 <c:forEach var="row" items="${dmiBean.metrics}">
  <tr>
   <c:forEach var="value" items="${row}">
    <td><c:out value="${value}" /> &nbsp;</td>
   </c:forEach>
  </tr>
 </c:forEach>
</table>

<%!String serviceURL;
String user;
String password;

String appId ;
String viewId;
String dataSourceId;
String resolution;

 public void getDimensions(examples.DMIBean bean) {
//  create locator
  String [][] ret = null;
  DMIServiceServiceLocator locator = new DMIServiceServiceLocator();
  try {
   DMIService service = locator.getDMIService(new URL(serviceURL));
//   set authentication
   ((Stub)service).setUsername(user);
   ((Stub)service).setPassword(password);
//   call method
   ret = service.getDimensions(appId, viewId, resolution);
   bean.setDimensions(ret);
   ret = service.getMetrics(appId, viewId, resolution);
   bean.setMetrics(ret);
  } catch (ServiceException e) {
   e.printStackTrace();
  } catch (RemoteException e) {
   e.printStackTrace();
  } catch (MalformedURLException e) {
   e.printStackTrace();
  }
 }%>
</body>
</html>
```

File: `clientReport.jsp`

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
 pageEncoding="ISO-8859-1"%>

<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<%@page import="java.net.MalformedURLException"%>
<%@page import="java.net.URL"%>
<%@page import="java.rmi.RemoteException"%>
```

```
<%@page import="javax.xml.rpc.ServiceException"%>
<%@page import="org.apache.axis.client.Stub"%>
<%@page import="adlex.delta.ws.DMIService"%>
<%@page import="adlex.delta.ws.DMIServiceServiceLocator"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>DMIService client examples - report</title>
</head>
<body>
<br>
<form action="form.jsp" >
<input type="submit" action="form.jsp" value="Back to form" >
</form>
<br>
<strong>Parameters:</strong><br/>
<table border="1">
<c:forEach var="map_entry" items="${param}">
<tr>
 <td>
     <strong><c:out value="${map_entry.key}" /></strong>
    </td>
 <td>
     <c:out value="${map_entry.value}" /><br>
    </td>
</tr>
</c:forEach>
</table>
<br>
<%

 /* connection params */
 vasURL = request.getParameter("vasURL");
 user = request.getParameter("user");
 password = request.getParameter("password");

 /* set criteria */
 appId = request.getParameter("appId");
 viewId = request.getParameter("viewId");
 dimId = request.getParameter("dimId");
 dataSourceId = request.getParameter("dataSourceId");

 quant = new Integer(request.getParameter("quant"));
 filter = request.getParameter("filter");
%>
<jsp:useBean id="dmiBean" class="examples.DMIBean">
<%
getDimensionValues(dmiBean);
%>
</jsp:useBean>
<strong> <a
 href="<%=vasURL%>/LSServlet?lsAction=LoadByName&lsEntryName=ExampleReport"
 target="report">ExampleReport without filter</a> </strong>
<br>
<br>
<strong>Report for application: </strong>
<table border="1">

 <c:forEach var="value" items="${dmiBean.dimensionValues}">
  <tr>
   <td>
<%-- in the FILTER_<dimId> parameter specifies the filter value --%>
   <a

href="<%=vasURL%>/LSServlet?lsAction=LoadByName&lsEntryName=ExampleReport&FILTER_appl=<c:out
 value="${value}"/>"
    target="report"><c:out value="${value}" /></a> &nbsp;</td>
  </tr>
 </c:forEach>
</table>

<%!String vasURL;
```

```
 String user;
 String password;
 String appId;
 String viewId;
 String dimId = "appl";
 String dataSourceId;
 Integer quant = 0;
 String filter;

 public void getDimensionValues(examples.DMIBean bean) {
  //  create locator
  String[] ret = null;
  DMIServiceServiceLocator locator = new DMIServiceServiceLocator();
  try {
   DMIService service = locator.getDMIService(new URL(vasURL
     + "/services/DMIService"));
   //   set authentication
   ((Stub) service).setUsername(user);
   ((Stub) service).setPassword(password);
   //   call method
   ret = service.getDimensionValues(appId, viewId, dimId,
     dataSourceId, quant, filter);
   bean.setDimensionValues(ret);
  } catch (ServiceException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  } catch (RemoteException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  } catch (MalformedURLException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  }
 }%>
</body>
</html>
```

APPENDIX A

Web Service Methods Reference

Methods and data structures enable you to integrate CVAM with enterprise solutions such as dashboards, portals, and intranets.

All data views available in AWDS or VAS are described in *Advanced Web Diagnostics Server – User Guide* or *Vantage Analysis Server – User Guide*. To understand how Data Mining Interface works, refer to *Data Mining Interface (DMI) – User Guide*. Note that metrics available in data views may be different from those used in predefined VAS reports.

getApplications

Method `getApplications` returns a list of applications as an array of arrays of `String`s.

```
public String[][] getApplications()
```

The nested array describes an application:

- `[0]` – Internal identifier of an application.
- `[1]` – User-readable name of the application. Used for presentation in DMI.

Example

The method call `getApplications()` returns a result such as: `{{"CVENT","Vantage Analysis Server"}, {"HTTPLOG","HTTP Analysis"}}`

getCurrentTime

Method `getCurrentTime` returns the time stamp in the form of the number of milliseconds since January 1, 1970.

```
public long getCurrentTime(String appId, String viewId, String resolution)
```

Parameter	Type	Description
`appId`	`String`	Application ID as returned by `getApplications()`.

Parameter	Type	Description
`viewId`	`String`	Data view ID as returned by `getDataViews()`.
`resolution`	`String`	Data resolution ID as returned by `getResolutions()`.

Example

The method call `getCurrentTime("CVENT", "ClientView", "1D")` returns a long integer representing the time stamp of the current batch of performance data returned by the referred data view. The format of the returned value is compliant with Java time stamp `System.getCurrentMillis()`. Note that the time returned depends on the data view and the granularity of how it calculates its time stamps.

getDataSources

Method `getDataSources` returns a list of data servers as an array of arrays of `Strings`.

```
public String[][] getDataSources(String appId, String viewId)
```

The nested array describes a data source:

`[0]` – Internal identifier of a data source.
`[1]` – User-readable name of the data source. Used for presentation in DMI.

Parameter	Type	Description
`appId`	`String`	Application ID, as returned by `getApplications()`
`viewId`	`String`	Data view ID, as returned by `getDataViews()`

Example

The method call `getDataSources("CVENT", "ClientView")` returns a result such as: `{{"Rolling", "Application"}, {}, …}`.

getDataViews

Method `getDataViews` returns a list of data views as an array of arrays of `Strings`.

```
public String[][] getDataViews(String appId)
```

The nested array describes a data view:

`[0]` – Internal identifier of a data view.
`[1]` – User-readable name of the data view. Used for presentation in DMI.

Parameter	Type	Description
`appId`	`String`	Application ID as returned by `getApplications()`

Example

The method call `getDataViews("CVENT")` returns a result such as: `{{"ClientView","Monitored traffic"}, {"Rolling","Baseline traffic"}, …}`

getDimensions

Method `getDimensions` returns a list of dimensions as an array of arrays of `Strings`.

```
public String[][] getDimensions(String appId, String viewId, String resolution)
```

The nested array describes a dimension:

`[0]` – Internal identifier of a dimension (filter criterion)
`[1]` – User-readable name of the data view. Used for presentation in DMI.

Parameter	Type	Description
`appId`	`String`	Application ID, as returned by `getApplications()`.
`viewId`	`String`	Data view ID, as returned by `getDataViews()`.
`resolution`	`String`	Data resolution ID, as returned by `getResolutions()`.

Example

The method call `getDimensions("CVENT", "ClientView", "1D")` returns a result such as: `{{"APPL","Application"}, {"SVR","Server IP Address"}, …}`.

getDimensionValues

Method `getDimensionValues` returns a list of dimension values as an array of `Strings`.

```
public String[] getDimensionValues(String appId, String viewId, String dimId,
String dataSourceId, Integer quantity, String filter)
```

The values can be used as parts of filter expressions in data retrieval method `getDMIData`.

Parameter	Type	Description
`appId`	`String`	Application ID, as returned by `getApplications()`.
`viewId`	`String`	Data view ID, as returned by `getDataViews()`.
`dimId`	`String`	Dimension ID, as returned by `getDimensions()`.

Parameter	Type	Description
`dataSourceId`	`String`	Data source ID, as returned by `getDataSources()` or `null`.
`quantity`	`Integer`	Maximum number of data rows in the returned table. If more values are available in DMI , only `quantity` will be returned. If `quantity` is `null` or `<=0`, then all values of the dimension will be returned. Note that this list may be very long.
`filter`	`String`	Filter expression to narrow dimension value selection. You can use wildcard characters "?" and "*" and combine filter expressions with the pipe character "|" interpreted as logical `OR`.

Example

The method call `getDimensionValues ("CVENT","ClientView","APPL",null,10,"A*")` returns a result such as: `{{"ABCapplication","Application","CVENT",{""}}, ...}`

getDMIData

Method `getDMIData` returns an `adlex.delta.ws.DMIData` object. Note that it is an actual data retrieval method.

Syntax

NOTE

The difference between `getDMIData` and `getDMIData2` is that the latter method returns an object that contains GUI labels and units of measure for returned data.

```
public adlex.delta.ws.DMIData getDMIData(String appId, String viewId, String
dataSourceId, String[] dimensionIds, String[] metricIds, String[][] dimFilters,
String[][] metricFilters, String[][] sort, Integer top, String resolution,
String timePeriod, Integer numberOfPeriods, Long timeBegin, Long timeEnd, Long
timeout)
```

Parameter	Type	Description
`appId`	`String`	Application ID, as returned by `getApplications()`.
`viewId`	`String`	Data view ID, as returned by `getDataViews()`.
`dataSourceId`	`String`	Dimension ID, as returned by `getDataSources()` or `null`.
`dimensionIds`	`String`	Dimension IDs, as returned by `getDimensions()`.
`metricIds`	`String`	Metric IDs, as returned by `getMetrics()`.

Parameter	**Type**	**Description**
`dimFilters`	array of `Strings`	Array of dimension filters. The array structure is as follows: `[0]` – dimension ID, as returned by `getDimensions()` `[1]` – as per description of `filter` in the table above `[2]` – filter negation trigger, if it is present and contains the value “`NOT`” or “`1`”.
`metricFilters`	array of `Strings`	Array of metric filters. The array structure is as follows: `[0]` – metric ID, as returned by `getMetrics()` `[1]` – operator `[2]` – value `[3]` – the filter is applied to results, if it is present and contains the value “`RESULT`” or “`1`”
`sort`	array of `Strings`	Result sort criteria. The nested array describes filter: [0] - metric or dimension identifier [1] - sort order. Available orders are: “`ASC`” or `null` for ascending order, “`DESC`” for descending order. In the current version, only the first subarray is taken into consideration.
`top`	`Integer`	Number of returned table rows. If `top` is set to `0`, all rows will be returned. If `top` is set to `null` or `<=0`, only first 1000 rows will be returned.
`resolution`	`String`	Data resolution ID, as returned by `getResolutions()`
`timePeriod`	`String`	Time period covered by the report, as one of the following values: `p` – 1 period (monitoring interval, dependent on the report server settings) `d` – 1 day `w` – 1 week `m` – 1 month `T` – today (since midnight) `1P` – last period `1H` – last hour `1D` – last day `7D` – last 7 days `30D` – last 30 days `3MO` – last 3 months `12MO` – last 12 months

Parameter	Type	Description
		`FW` – the full week `FM` – the full month `WTD` – week to date `MTD` – month to date `QTD` – quarter to date `YTD` – year to date
`numberOfPeriods`	`Integer`	Number of periods; if it is set and `timeBegin` is null, then `timePeriod` has to be set.
`timeBegin`	`Long`	Data coverage begin time, expressed in milliseconds elapsed since January 1, 1970; if it is set, then `timePeriod` and `numberOfPeriods` are ignored.
`timeEnd`	`Long`	Data coverage begin time, expressed in milliseconds elapsed since January 1, 1970; if it is `null`, then time of the last processed performance data batch is taken.
`timeOut`	`Long`	Timeout value in milliseconds to retrieve data; if `timeout` <=0, then default DMI timeout value for queries will be applied.

`adlex.delta.ws.DMIData`

```
public class DMIData implements java.io.Serializable {
    String[] columnHeaders;
    String[][] formattedData;
    Double[][] rawData;
    Boolean timeout;
    Long TimeoutValue;
    DMIServiceError [] dmiServiceError;
    Long timeBegin;
    Long timeEnd;
}
```

where:

`columnHeaders`	A table of dimension IDs and metrics IDs. Note that in some cases DMI can limit list of metrics and dimensions depending on a data view logic.
`formattedData`	A two-dimensional table of formatted data returned by DMI.
`rawData`	A two-dimensional table of data returned in the numerical form. Most metrics and some dimensions are numbers (such as *time*). If something does not have a numeric representation, it is represented as `null`.
`timeout`	A boolean value indicating whether time out occurred.
`timeoutValue`	A timeout threshold value.
`dmiServiceError`	A table of errors, messages, and warnings returned by DMI. Each position describes the nature of a problem.
`timeBegin`	The actual begin time taken by the server.
`timeEnd`	The actual end time taken by the server.

dmiServiceError

```
public class DMIServiceError {
    String [] errorDescriptions;
    protected String [] infoDescriptions;
    protected String [] warningDescriptions;
    boolean error;
    boolean info;
    boolean warning;
}
```

`errorDescriptions` – A table of error descriptions; it contains values only if `error` is set to `true`.

`warningDescriptions` – A table of warning descriptions; it contains values only if `warning` is set to `true`.

`infoDescriptions` – A table of info descriptions; it contains values only if `info` is set to `true`.

`error` – Indicates the occurrence of a DMI error.

`warning` – Indicates the occurrence of a DMI warning.

`info` – Indicates the occurrence of a DMI info message.

Example

The method call `getDMIData(appId, viewId, dataSourceId, dimensionIds, metricIds, dimFilters, metricFilters, sort, top, resolution, timePeriod, numberOfPeriods, timeBegin, timeEnd, timeout)` returns (using the `formattedData` method) a result such as: `{{"CVENT"}, {"CVCache"}, {"localhost"}, {"begT"}, {"svrDelay"}, {""}, {""}, {""}, {10}, {"r"}, {"1P"}, {""}, {""}, {""}, {30000}}`

getDMIData2

Method `getDMIData2` returns an `adlex.delta.ws.DMIData` object. Note that it is an actual data retrieval method.

Syntax

NOTE

The difference between `getDMIData` and `getDMIData2` is that the latter method returns an object that contains GUI labels and units of measure for returned data.

```
public adlex.delta.ws.DMIData2 getDMIData2(String appId, String viewId, String dataSourceId, String[] dimensionIds, String[] metricIds, String[][] dimFilters, String[][] metricFilters, String[][] sort, Integer top, String resolution, String timePeriod, Integer numberOfPeriods, Long timeBegin, Long timeEnd, Long timeout)
```

Parameter	Type	Description
`appId`	`String`	Application ID, as returned by `getApplications()`.
`viewId`	`String`	Data view ID, as returned by `getDataViews()`.
`dataSourceId`	`String`	Dimension ID, as returned by `getDataSources()` or `null`.

Parameter	Type	Description
`dimensionIds`	`String`	Dimension IDs, as returned by `getDimensions()`.
`metricIds`	`String`	Metric IDs, as returned by `getMetrics()`.
`dimFilters`	array of `Strings`	Array of dimension filters. The array structure is as follows: `[0]` – dimension ID, as returned by `getDimensions()` `[1]` – as per description of `filter` in the table above `[2]` – filter negation trigger, if it is present and contains the value “`NOT`” or “`1`”.
`metricFilters`	array of `Strings`	Array of metric filters. The array structure is as follows: `[0]` – metric ID, as returned by `getMetrics()` `[1]` – operator `[2]` – value `[3]` – the filter is applied to results, if it is present and contains the value “`RESULT`” or “`1`”
`sort`	array of `Strings`	Result sort criteria. The nested array describes filter: [0] - metric or dimension identifier [1] - sort order. Available orders are: “`ASC`” or `null` for ascending order, “`DESC`” for descending order. In the current version, only the first subarray is taken into consideration.
`top`	`Integer`	Number of returned table rows. If `top` is set to `0`, all rows will be returned. If `top` is set to `null` or `<=0`, only first 1000 rows will be returned.
`resolution`	`String`	Data resolution ID, as returned by `getResolutions()`
`timePeriod`	`String`	Time period covered by the report, as one of the following values: `p` – 1 period (monitoring interval, dependent on the report server settings) `d` – 1 day `w` – 1 week `m` – 1 month `T` – today (since midnight) `1P` – last period `1H` – last hour `1D` – last day `7D` – last 7 days

Parameter	Type	Description
		`30D` – last 30 days `3MO` – last 3 months `12MO` – last 12 months `FW` – the full week `FM` – the full month `WTD` – week to date `MTD` – month to date `QTD` – quarter to date `YTD` – year to date
`numberOfPeriods`	`Integer`	Number of periods; if it is set and `timeBegin` is null, then `timePeriod` has to be set.
`timeBegin`	`Long`	Data coverage begin time, expressed in milliseconds elapsed since January 1, 1970; if it is set, then `timePeriod` and `numberOfPeriods` are ignored.
`timeEnd`	`Long`	Data coverage begin time, expressed in milliseconds elapsed since January 1, 1970; if it is `null`, then time of the last processed performance data batch is taken.
`timeOut`	`Long`	Timeout value in milliseconds to retrieve data; if `timeout` <=0, then default DMI timeout value for queries is applied.

`adlex.delta.ws.DMIData2`

```
public class DMIData2 implements java.io.Serializable {
    String[] columnHeaders;
    String[] columnUnit;
    String columnHeaderName;
    String[][] formattedData;
    Double[][] rawData;
    Boolean timeout;
    Long TimeoutValue;
    DMIServiceError [] dmiServiceError;
    Long timeBegin;
    Long timeEnd;
}
```

where:

`columnHeaders`	A table of dimension and metrics internal IDs. Note that in some cases DMI can limit the list of metrics and dimensions, depending on the data view logic.
`columnUnit`	A table of units of measure for returned dimensions and metrics.
`columnHeaderNames`	A table of dimension names and metric names, as they appear in DMI reports on the screen. Note that in some cases DMI can limit the list of metrics and dimensions, depending on data view logic.
`formattedData`	A two-dimensional table of formatted data returned by DMI.

`rawData`	A two-dimensional table of data returned in the numerical form. Most metrics and some dimensions are numbers (such as *time*). If something does not have a numeric representation, it is represented as `null`.
`timeout`	A boolean value indicating whether a timeout occurred
`timeoutValue`	A timeout threshold value.
`dmiServiceError`	A table of of errors, messages, and warnings returned by DMI. Each position describes the nature of a problem.
`timeBegin`	The actual begin time taken by the server.
`timeEnd`	The actual end time taken by the server.

`dmiServiceError`

```
public class DMIServiceError {
    String [] errorDescriptions;
    protected String [] infoDescriptions;
    protected String [] warningDescriptions;
    boolean error;
    boolean info;
    boolean warning;
}
```

`errorDescriptions` – A table of error descriptions; it contains values only if `error` is set to `true`.

`warningDescriptions` – A table of warning descriptions; it contains values only if `warning` is set to `true`.

`infoDescriptions` – A table of info descriptions; it contains values only if `info` is set to `true`.

`error` – Indicates the occurrence of a DMI error.

`warning` – Indicates the occurrence of a DMI warning.

`info` – Indicates the occurrence of a DMI info message.

Example

The method call `getDMIData2(appId, viewId, dataSourceId, dimensionIds, metricIds, dimFilters, metricFilters, sort, top, resolution, timePeriod, numberOfPeriods, timeBegin, timeEnd, timeout)` returns (using the `formattedData` method) a result such as: `{{"CVENT"}, {"CVCache"}, {"localhost"}, {"begT"}, {"svrDelay"}, {""}, {""}, {""}, {10}, {"r"}, {"1P"}, {""}, {""}, {""}, {30000}}`

getLastSampleTime

The method `getLastSampleTime` returns a time stamp of the last performance data batch processed by the report server, in *milliseconds* elapsed since January 1, 1970.

```
public long getLastSampleTime(String appId, String viewId, String resolution)
```

Parameter	Type	Description
`appId`	`String`	Application ID, as returned by `getApplications()`.
`viewId`	`String`	Data view ID, as returned by `getDataViews()`.

Parameter	Type	Description
`resolution`	`String`	Data resolution ID, as returned by `getResolutions()`.

getMetrics

The method `getMetrics` returns a list of metrics as an array of arrays of `Strings`.

```
public String[][] getMetrics(String appId, String viewId, String resolution)
```

The nested array describes a metric:

`[0]` – Internal identifier of a metric.
`[1]` – User-readable name of the metric. Used for presentation in DMI.

Type	Type	Description
`appId`	`String`	Application ID, as returned by `getApplications()`.
`viewId`	`String`	Data view ID, as returned by `getDataViews()`.
`resolution`	`String`	Data resolution ID, as returned by `getResolutions()`.

Example

The method call `getMetrics("CVENT", "ClientView", "1D")` returns a result such as: `{{"C_BYTES","Client Bytes"}, {"S_BYTES","Server bytes"}, …}`.

getResolutions

The method `getResolutions` returns a list of identifiers of available resolutions as an array of `Strings`.

```
public String[] getResolutions(String appId, String viewId)
```

Resolution IDs are identical to those used for definitions in `dmi_dataview-*.properties` files.

Parameter	Type	Description
`appId`	`String`	application ID, as returned by `getApplications()`
`viewId`	`String`	data view ID, as returned by `getDataViews()`

Example

The method call `getResolutions("CVENT", "ClientView")` returns a result such as: `{"r", "1", "6"}`.

getServerUUID

The method `getServerUUID` returns the Universal Unique Identifier (UUID) of the report server.

```
public String getServerUUID()
```

Example

The method call `getServerUUID()` returns a result such as:

```
"12ac5ec3-437d-4383-8055-bef7c5a104df"
```

Index