

**Oracle® OLAP**

Customizing Analytic Workspace Manager

11g Release 2 (11.2)

**E17237-01**

August 2010

Oracle OLAP Customizing Analytic Workspace Manager, 11g Release 2 (11.2)

E17237-01

Copyright © 2006, 2010, Oracle and/or its affiliates. All rights reserved.

Primary Author: David McDermid

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

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

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

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

---

---

# Contents

## 1 Customizing With XML Documents

<b>Describing SQL Reports</b> .....	1-1
Creating an XML Document for SQL Reports .....	1-1
Example of a Simple Report .....	1-2
Creating Report Folders .....	1-2
Using Bind Variables to Restrict the Report Contents.....	1-3
Creating Reports in Object Folders.....	1-5
Reference: Elements for SQL Reports .....	1-7
<AWMTree> .....	1-7
<AWMNode> .....	1-7
<ShowIfQueryTrue> .....	1-8
<b>Describing Calculation Templates</b> .....	1-8
Creating an XML Document for Calculations .....	1-8
Example of a Simple Calculation Template .....	1-9
Adding an Option to a Calculation .....	1-10
Creating More Complex Calculation Templates .....	1-12
Reference: Elements for Calculations .....	1-14
<AWMCalcs> .....	1-15
<Calc> .....	1-15
<CalcOptional> .....	1-16
<CalcOptionalDefinitions> .....	1-17
<Category> .....	1-17
<Item>.....	1-17
<Param> .....	1-17
<Params> .....	1-18

## 2 Introducing Analytic Workspace Manager Plug-ins

<b>Describing Analytic Workspace Manager Plug-ins</b> .....	2-1
Enabling Analytic Workspace Manager Plug-ins .....	2-2
How Analytic Workspace Manager Calls a Plug-in .....	2-3
Calling an AWMPlugin.....	2-3
Calling a ViewerPlugin or EditorPlugin .....	2-4
<b>Describing the AWMPlugin Interface</b> .....	2-5
Values for the type and obj Parameters .....	2-6
Elements in the params Map for an AWMPlugin.....	2-9

Params Map Elements for Non-custom Objects.....	2-9
Params Map Elements for Custom Objects.....	2-9
Example params Map Elements for an AWMPlugin .....	2-10
<b>Describing the ViewerPlugin and EditorPlugin Interfaces .....</b>	<b>2-11</b>
Describing the ViewerPlugin Interface .....	2-12
Describing the EditorPlugin Interface.....	2-13
Elements in the params Map for a ViewerPlugin or EditorPlugin.....	2-14
Example params Map Elements for a ViewerPlugin and an EditorPlugin.....	2-15
Example params Map Elements for a ViewerPlugin.....	2-15
Example params Map Elements for an EditorPlugin .....	2-16
<b>Steps in Creating a Plug-in.....</b>	<b>2-17</b>
<b>Describing the Available Plug-ins .....</b>	<b>2-18</b>
Creating an XML Document for Descriptions of Plug-ins.....	2-19
Reference: Elements for Plug-in Descriptions.....	2-19
<AWMPlugins> .....	2-19
<Plugin> .....	2-19
<Description> .....	2-20

### **3 Examples of Analytic Workspace Manager Plug-ins**

<b>Availability of Example Classes and XML Documents .....</b>	<b>3-1</b>
<b>Examples of AWMPlugin.....</b>	<b>3-2</b>
ViewXMLPlugin Example .....	3-2
DeleteDimPlugin Example .....	3-5
<b>Examples of ViewerPlugin and EditorPlugin.....</b>	<b>3-9</b>
LevelViewerPlugin Example .....	3-10
MeasureViewerPlugin Example .....	3-12
CubeViewerPlugin Example .....	3-15
DimEditorPlugin Example.....	3-17
<b>Example of Plug-in Descriptions .....</b>	<b>3-24</b>

## **Index**

## List of Examples

1-1	Creating the My User Views Report .....	1-2
1-2	Creating the My SQL Reports Folder.....	1-3
1-3	Passing the Name of a View to a SELECT Statement.....	1-4
1-4	Including an Icon in a Report.....	1-6
1-5	Basic XML Structure for Reports .....	1-7
1-6	Creating the Discount and Average Calculation Templates .....	1-10
1-7	Adding an Option to One Calculation.....	1-11
1-8	Adding an Option to Multiple Calculations .....	1-11
1-9	Sample AWMCalcs Document .....	1-13
1-10	Basic XML Structure for Calculations .....	1-14
1-11	XML Structure for Calculations With Options.....	1-14
1-12	XML Structure for Calculations With Choice Lists.....	1-15
2-1	XML Structure for Descriptions of Plug-ins.....	2-19
3-1	The ViewXMLPlugin Class .....	3-2
3-2	The DeleteDimPlugin Class.....	3-6
3-3	Creating a dimension.xml Document .....	3-10
3-4	The LevelViewerPlugin Class .....	3-10
3-5	Creating a cube.xml Document .....	3-12
3-6	The MeasureViewerPlugin Class.....	3-12
3-7	Creating an aw.xml Document .....	3-15
3-8	The CubeViewerPlugin Class .....	3-16
3-9	The DimEditorPlugin Class.....	3-18
3-10	Creating an awmplugins.xml Document .....	3-24

## List of Figures

1-1	Displaying a Report .....	1-2
1-2	Organizing Reports in Folders .....	1-3
1-3	Modifying the Content of a Report .....	1-4
1-4	Reports in the GLOBAL Schema Folder .....	1-6
1-5	Listing the New Calculations .....	1-9
1-6	Displaying a New Template .....	1-9
1-7	Providing an Option to a Calculation .....	1-11
1-8	Listing More New Calculations .....	1-12
1-9	Choice Lists In a Calculation Template .....	1-13
2-1	Configuration Item on the Tools Menu .....	2-2
2-2	Configuration Dialog Box with Enable Plugins Selected .....	2-3
2-3	Right-click Menu of the Navigation Tree for a Calculated Measure .....	2-4
2-4	Sequence of Calls to an AWMPlugin .....	2-6
2-5	Sequence of Calls to a ViewerPlugin .....	2-12
2-6	Sequence of Calls to an EditorPlugin .....	2-14
3-1	Dialog Box Displayed by ViewXMLPlugin .....	3-5
3-2	Right-click Menu Displayed by DeleteDimPlugin .....	3-9
3-3	Dialog Box Displayed by DeleteDimPlugin .....	3-9
3-4	Results of the MyLevels <AWMNode> in dimension.xml .....	3-11
3-5	Results of LevelViewerPlugin .....	3-12
3-6	Results of the MyMeasures <AWMNode> in cube.xml .....	3-14
3-7	Results of MeasureViewerPlugin .....	3-15
3-8	Results of the MyCubes <AWMNode> in aw.xml .....	3-17
3-9	Results of the CubeViewerPlugin .....	3-17
3-10	Results of the MyDims <AWMNode> in aw.xml .....	3-22
3-11	Results of DimEditorPlugin .....	3-22
3-12	Result of MyLevels <AWMNode> Under MyDims in aw.xml .....	3-23
3-13	Results of the Nested <AWMNode> in the MyLevels <AWMNode> in aw.xml .....	3-23
3-14	Plugins Tab in the About Dialog Box .....	3-24

## List of Tables

1-1	Parameters for the ui Attribute .....	1-16
2-1	Type Values and Objects for Navigation Tree Objects.....	2-7
2-2	Keys and Values of the params Map for a Non-custom Object .....	2-9
2-3	Keys and Values of the params Map for a Custom Object .....	2-9
2-4	Keys and Values of the BIND_MAP Map .....	2-10
2-5	Keys and Values of the params Map for DeleteDimPlugin.....	2-11
2-6	Keys and Values of the BIND_MAP Map for DeleteDimPlugin .....	2-11
2-7	Keys and Values of the params Map for a ViewerPlugin or EditorPlugin .....	2-14
2-8	Keys and Values of the params Map for MeasureViewerPlugin.....	2-15
2-9	Keys and Values of the params Map for DimEditorPlugin.....	2-16



---

---

# Preface

With Oracle Analytic Workspace Manager, Version 11.2, you can add custom objects and menus to the user interface. You can customize Analytic Workspace Manager by doing the following:

- Adding custom reports and calculation templates to the navigation tree.
- Adding selections to the menu that appears when the user right-clicks a navigation tree object.
- Providing graphical user interface elements for viewing or editing relational database objects or OLAP objects, or both.

[Chapter 1, "Customizing With XML Documents"](#) describes how you can add reports and calculation templates to the navigation tree by using XML documents. It also describes how you can specify an Analytic Workspace Manager Java plug-in within an XML document.

[Chapter 2, "Introducing Analytic Workspace Manager Plug-ins"](#) describes the Java plug-in interfaces that Analytic Workspace Manager supports and demonstrates how to develop a plug-in. With a plug-in you can add items to the right-click menu of an Analytic Workspace Manager navigation tree object. You can also provide a viewer or an editor plug-in that displays in the Analytic Workspace Manager property inspector.

This document describes the Analytic Workspace Manager XML and Java plug-in interfaces and provides simple examples of implementations of them.

This preface contains the following topics.

- [Audience](#)
- [Documentation Accessibility](#)
- [Related Documents](#)
- [Conventions](#)

## Audience

This document is intended for XML or Java developers who want to use XML documents or Java plug-ins to extend the functionality of Analytic Workspace Manager in Oracle Database 11g Release 2 (11.2) with the OLAP option.

## Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our

documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at <http://www.oracle.com/accessibility/>.

### **Accessibility of Code Examples in Documentation**

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

### **Accessibility of Links to External Web Sites in Documentation**

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

### **Access to Oracle Support**

Oracle customers have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/support/contact.html> or visit <http://www.oracle.com/accessibility/support.html> if you are hearing impaired.

## **Related Documents**

For more information on Oracle OLAP, on using Analytic Workspace Manager, and on the Oracle OLAP Java API, see the following documents.

- *Oracle OLAP User's Guide*
- *Oracle OLAP Java API Reference*
- *Oracle OLAP Java API Developer's Guide*

## **Conventions**

The following text conventions are used in this document:

<b>Convention</b>	<b>Meaning</b>
<b>boldface</b>	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
<code>monospace</code>	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

---

---

# Customizing With XML Documents

Analytic Workspace Manager provides numerous SQL reports so that you can query the Oracle Database data dictionary and system tables without having to open another SQL interface. It also provides an extensive list of templates for generating calculated measures. You can supplement both of these features by adding custom SQL reports and calculation templates.

To define these customizations, you create XML documents. For an introduction to XML, refer to a source such as the W3Schools XML tutorial at <http://www.w3schools.com>. Use an XML-enabled editor or browser to validate the syntax of your XML documents.

This chapter contains the following topics:

- [Describing SQL Reports](#)
- [Describing Calculation Templates](#)

## Describing SQL Reports

You can add reports to the Analytic Workspace Manager navigation tree that appear along with the other built-in reports. You provide a name for the report and a SQL SELECT command. You can optionally modify the report contents at run-time and organize them into folders.

## Creating an XML Document for SQL Reports

1. Develop a SQL query that returns the information to make available in Analytic Workspace Manager.
2. Create a text file named `awmtree.xml` in the directory with the Analytic Workspace Manager executable, typically `Oracle_home/olap/awm`.
3. Begin the file with an XML declaration like this one:

```
<?xml version="1.0" encoding="utf-8"?>
```

Specify the appropriate encoding for your site.

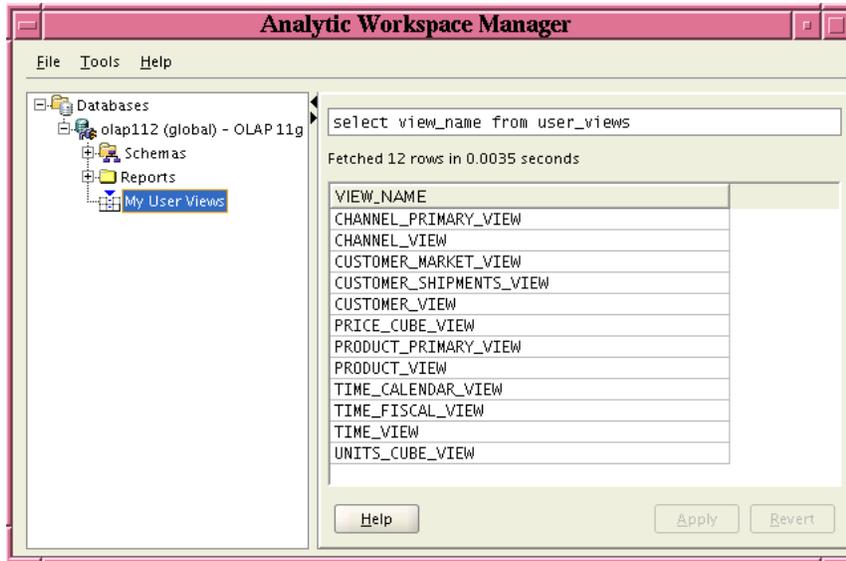
4. Enter the XML for the template, as described in "[Reference: Elements for SQL Reports](#)" on page 1-7.
5. For the `sql` attribute of the `<AWMNode>` element, enter the SELECT command that you developed in Step 1. You can replace values in a WHERE clause with bind variables from parent `<AWMNode>` elements, as described in "[Using Bind Variables to Restrict the Report Contents](#)" on page 1-3.

- Refresh the navigation tree. The new reports appear at the end of the tree.  
If the new reports do not appear, then look for syntax errors in the XML.

## Example of a Simple Report

Figure 1–1 shows a new report named My User Views in the Analytic Workspace Manager navigation tree. The report displays the results of this query: `SELECT view_name FROM user_views`.

**Figure 1–1** *Displaying a Report*



The XML document in Example 1–1 defines the My User Views report. The document contains two elements: `<AWMTree>` and `<AWMNode>`. `<AWMTree>` is the root element and identifies this document as containing XML that defines a report for Analytic Workspace Manager. The document can have one or more `<AWMNode>` elements. In this example, a single `<AWMNode>` element defines the My User Views report.

Among the attributes that the `<AWMNode>` element can have are `name` and `sql`. Example 1–1 uses `name` to identify the report as My User Views, and `sql` to specify a `SELECT` statement that executes when you select the report in the tree.

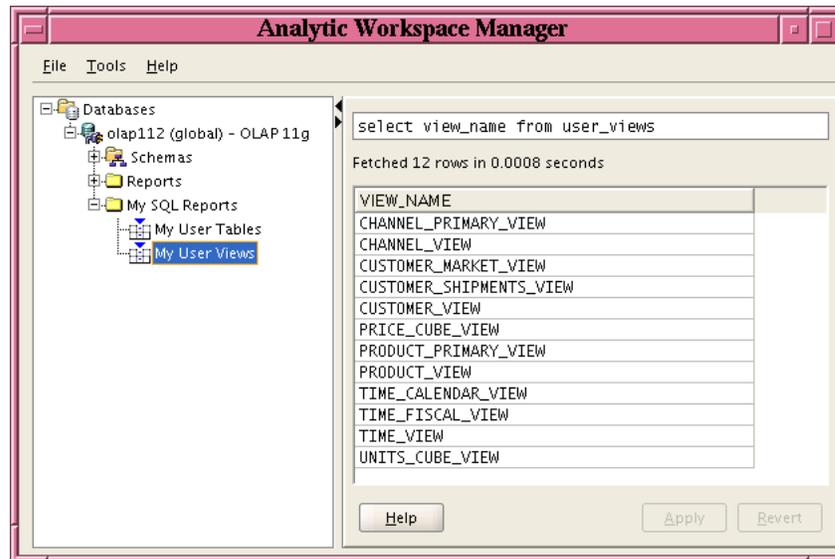
### Example 1–1 Creating the My User Views Report

```
<?xml version="1.0" encoding="utf-8" ?>
<AWMTree>
  <AWMNode name="My User Views" sql="SELECT view_name FROM user_views"/>
</AWMTree>
```

## Creating Report Folders

Folders provide an easy way to organize your reports. A folder is simply an `<AWMNode>` element without a SQL statement, and it is the parent of other `<AWMNode>` elements. You can nest `<AWMNode>` elements as deeply as you want.

Figure 1–2 shows a folder named My SQL Reports. It contains two reports, My User Tables and My User Views.

**Figure 1–2 Organizing Reports in Folders**

The XML document in [Example 1–2](#) shows that the `<AWMNode>` elements defining the two reports are the children of the `<AWMNode>` element that defines the My SQL Reports folder.

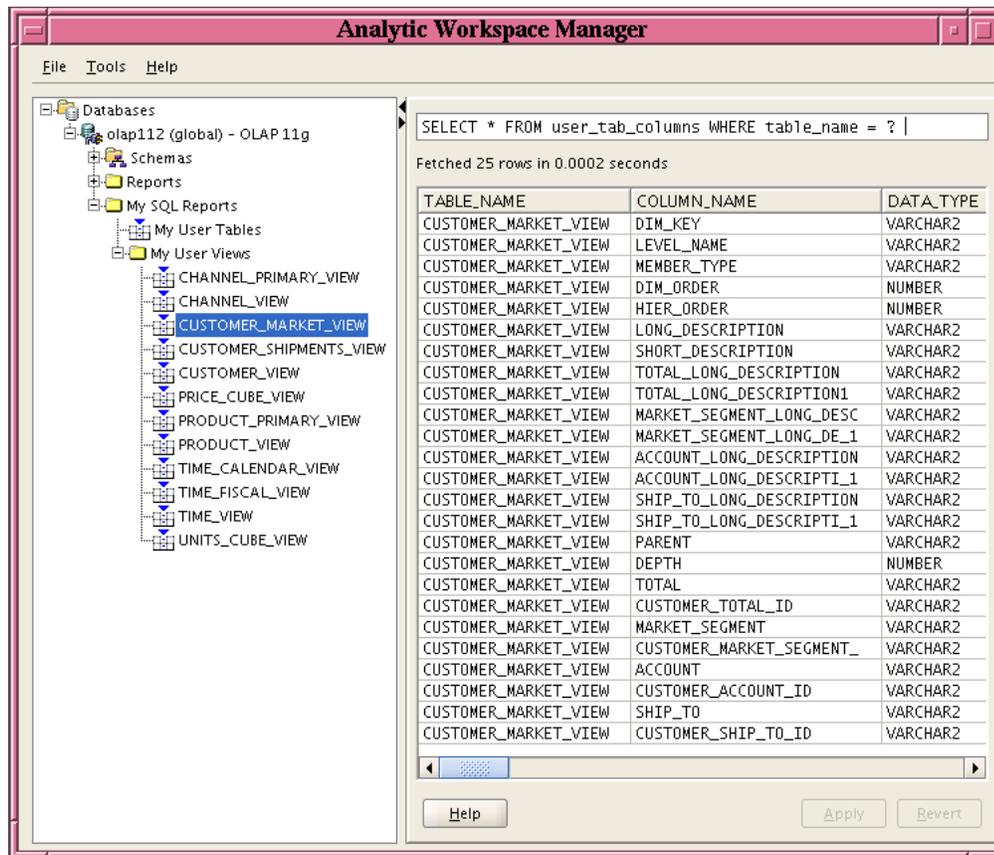
#### **Example 1–2 Creating the My SQL Reports Folder**

```
<?xml version="1.0" encoding="utf-8" ?>
<AWMTree>
  <AWMNode name="My SQL Reports">
    <AWMNode name="My User Tables" sql="SELECT table_name FROM user_tables"/>
    <AWMNode name="My User Views" sql="SELECT view_name FROM user_views"/>
  </AWMNode>
</AWMTree>
```

## Using Bind Variables to Restrict the Report Contents

Bind variables restrict the contents of a report based on your run-time selection from a list of values in the navigation tree. [Figure 1–3](#) shows the list of views returned by the My User Views report, which is now displayed in the navigation tree instead of in the property inspector. The property inspector displays a report for the particular view selected in the tree.

The most deeply nested `<AWMNode>` element (the "leaf" element) is displayed in a grid in the property inspector and can return multiple columns. The parent elements are displayed in the navigation tree and either return no columns (that is, a folder) or one column, as shown here.

**Figure 1-3 Modifying the Content of a Report**

The type parameter of an `<AWMNode>` element stores the run-time selection. By referencing the name of the type parameter in a nested `<AWMNode>` element, you pass the value into that `SELECT` statement. The XML document in [Example 1-3](#) shows an `<AWMNode>` element nested in the My User Views `<AWMNode>`. The nested `<AWMNode>` uses the value of the type element from the parent `<AWMNode>` element as a bind variable. You reference a bind variable by putting it in braces `{ }`, as in `{view}` in the example.

#### **Example 1-3 Passing the Name of a View to a SELECT Statement**

```
<?xml version="1.0" encoding="utf-8" ?>
<AWMTree>
  <AWMNode name="My SQL Reports">
    <AWMNode
      name="My User Tables"
      sql="select table_name from user_tables"/>
    <AWMNode
      name="My User Views"
      type="view"
      sql="select view_name from user_views">
      <AWMNode sql="SELECT * FROM user_tab_columns WHERE table_name = {view}"/>
    </AWMNode>
  </AWMNode>
</AWMTree>
```

## Creating Reports in Object Folders

The `awmtree.xml` document defines reports that appear in the navigation tree under a database connection, after the Reports folder. You can also define reports that appear in the folders for all schemas, analytic workspaces, cubes, or dimensions. The reports must be in XML documents that have the following names:

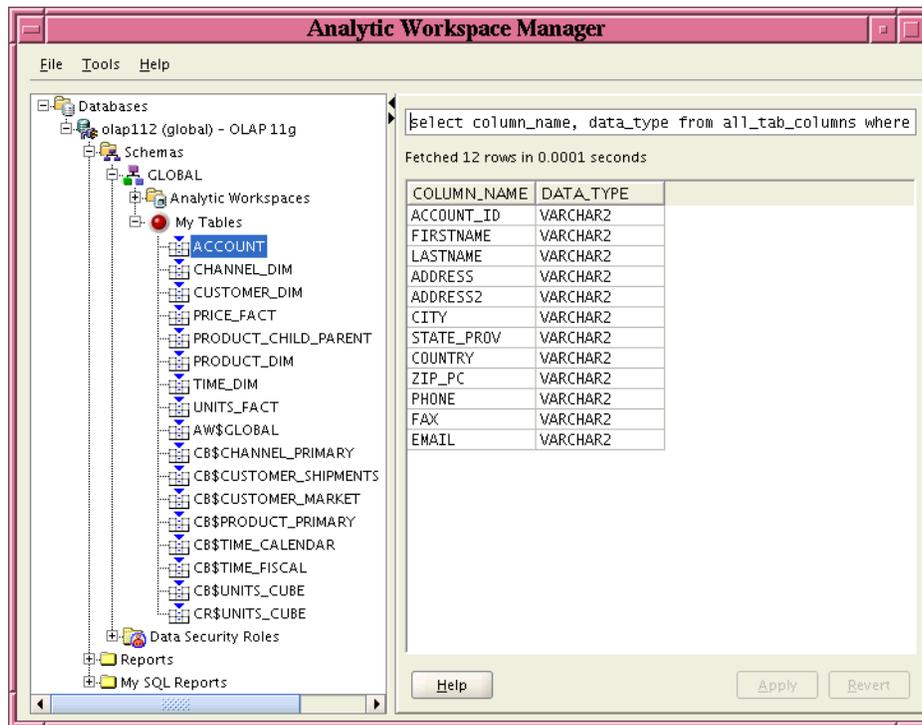
- `schema.xml`, for reports that appear in all schema folders.
- `aw.xml`, for reports that appear in all analytic workspace folders.
- `dimension.xml`, for reports that appear in all dimension folders.
- `cube.xml`, for reports that appear in all cube folders.

These files must be in a JAR file in the plug-in directory.

### To create reports in object folders:

1. Open Analytic Workspace Manager and select **Configuration** from the Tools menu, as shown in [Figure 2-1](#) on page 2-2. The Configuration dialog box appears.
2. Select **Enable plugins** and enter the path to a plug-in directory, if these parameters are not set already, as shown in [Figure 2-2](#) on page 2-3.
3. Click **OK**, and close Analytic Workspace Manager.
4. Create one or more XML document files.
5. In the plug-in directory, create a JAR file containing the XML files and any icons referenced by them. You can create one JAR file for all of them or create individual JAR files.
6. Open Analytic Workspace Manager and expand the navigation tree to see the reports.

[Figure 1-4](#) shows a report folder named My Tables described in a `schema.xml` document. It uses a custom icon (red button) in the navigation tree. The My Tables folder appears in the GLOBAL schema folder.

**Figure 1–4 Reports in the GLOBAL Schema Folder**

The schema.xml document in [Example 1–4](#) has an `<AWMNode>` element named My Tables. That element has an `icon` attribute that specifies a graphics file. In the JAR file that contains the schema.xml file and the button.jpg file, both files are in a directory named plugin11202.

---

**Note:** The references to icon files or Java class files in an XML document must reflect the directory structure of the JAR file containing them. The examples in this document of XML documents and Java plug-ins were created in a JDeveloper project named plugin11202. The project deploys the XML and Java class files in a JAR file. In the JAR file, the XML files and class files are in a directory named plugin11202. In [Example 1–4](#), the `icon` attribute specification includes the directory: `icon="plugin11202/button.jpg"`. In the examples of XML documents in [Chapter 3](#), the `viewClass` attribute specifications include the package name, as in `viewClass="plugin11202.DimEditorPlugin"` in [Example 3–7](#) on page 3-15.

---

#### **Example 1–4 Including an Icon in a Report**

```
<?xml version="1.0" encoding="US-ASCII" ?>
<AWMTree>
  <AWMNode name="My Tables"
    icon="plugin11202/button.jpg"
    type="table"
    sql="select table_name from all_tables where owner = {owner}">
    <AWMNode sql="select column_name, data_type from all_tab_columns where owner =
{owner} and table_name = {table}"/>
  </AWMNode>
</AWMTree>
```

```
</AWMTree>
```

## Reference: Elements for SQL Reports

An XML document for SQL reports has the basic format shown in [Example 1–5](#).

### Example 1–5 Basic XML Structure for Reports

```
<AWMTree>
  <AWMNode>
    <AWMNode>
      <ShowIfQueryTrue>
    <AWMNode>
      .
      .
      .
  </AWMNode>
</AWMTree>
```

#### <AWMTree>

The root element that identifies this document as containing custom reports for Analytic Workspace Manager. It contains one or more <AWMNode> elements.

#### Contains

<AWMNode>

#### Attributes

None

#### <AWMNode>

Defines a report folder or SQL report. It contains one or more <AWMNode> elements.

#### Contains

<AWMNode>, <ShowIfQueryTrue>

#### Attributes

<AWMNode> has the following attributes:

- **name:** The name of the folder or report. For a folder, this attribute is the only one required.
- **type:** The name of a bind variable that stores the selected value of the report. Use this bind variable to pass a user selection to a second, nested report.
- **sql:** A SQL `SELECT` statement, which can contain a bind variable in a `WHERE` clause. The bind variable is defined by the `type` attribute of a parent <AWMNode> attribute. The query results appear in the tree for a parent <AWMNode> element; for leaf elements, the results appear in the property inspector.
- **viewClass:** A Java class that implements the `ViewerPlugin` or `EditorPlugin` interface. The plug-in displays in the property inspector. For information on these Java plug-in interfaces, see [Chapter 2, "Introducing Analytic Workspace Manager Plug-ins"](#). The `viewClass` and `viewSQL` attributes are mutually exclusive. Use only one of them for any single <AWMNode>.
- **viewSql:** A SQL `SELECT` statement. The query results appear in the property inspector. Use this attribute to specify a SQL statement for a parent <AWMNode>.

The `viewSQL` and `viewClass` attributes are mutually exclusive. Use only one or the other for any single `<AWMNode>`.

- **icon:** An image to use in the navigation tree. The image must be in a JAR file in the plug-in directory and should be about 20 x 20 pixels.

### **<ShowIfQueryTrue>**

Controls the display of the parent report.

### **Contains**

None

### **Attributes**

`<ShowIfQueryTrue>` has the following attribute:

- **sql:** A SQL `SELECT` statement that creates the condition for displaying the parent report. If the query returns one or more rows, then the report is displayed in the navigation tree. If no rows are returned, then the report is hidden.

## Describing Calculation Templates

You can define a calculation template that appears in the Create a Calculated Measure dialog box like any other calculation. You provide a name for the calculation, the text of the template, and a calculation using the OLAP expression syntax.

## Creating an XML Document for Calculations

1. Create a custom measure in Analytic Workspace Manager that performs the type of calculation that you want in a template. Use this custom measure to validate the syntax of the expression for the template.

**See Also:** Analytic Workspace Manager Help for the OLAP expression syntax reference.

2. Create a text file named `awmcalcs.xml` in the directory with the Analytic Workspace Manager executable, typically `ORACLE_HOME/olap/awm`.
3. Begin the file with an XML declaration like this one:

```
<?xml version="1.0" encoding="utf-8"?>
```

Specify the appropriate encoding for your site.

4. Enter the XML for the template, as described in "[Reference: Elements for Calculations](#)" on page 1-14.
5. For the expression attribute of the `<Calc>` element, cut-and-paste the calculation from the custom measure that you created earlier. Replace the names of the measure, dimension, and so forth with the variables from the `ui` parameter.
6. Open Analytic Workspace Manager. The new categories and templates appear at the end of the Calculation Type list in the Create Calculated Measure dialog box.

If the new entries do not appear, then look for syntax errors in the XML. To see changes to the XML document, just reopen the Create Calculated Measure dialog box.

## Example of a Simple Calculation Template

Figure 1–5 shows the Calculation Type list in the Create Calculated Measure dialog box. The list contains a new folder named My New Calcs with two additional calculations: Discount and Average.

**Figure 1–5 Listing the New Calculations**

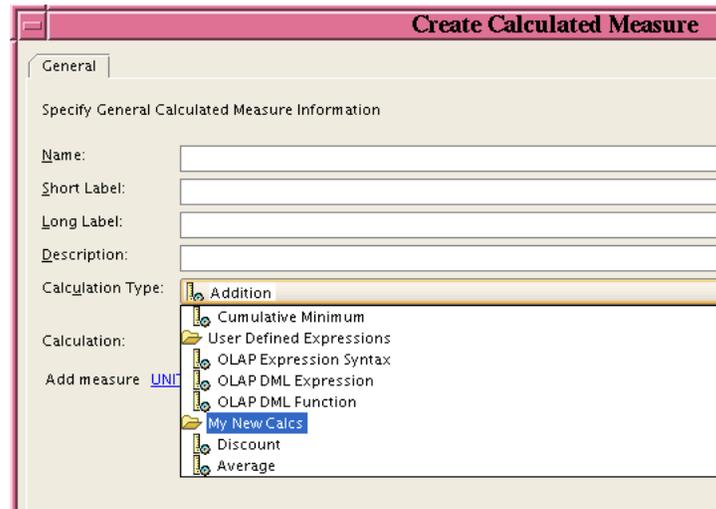
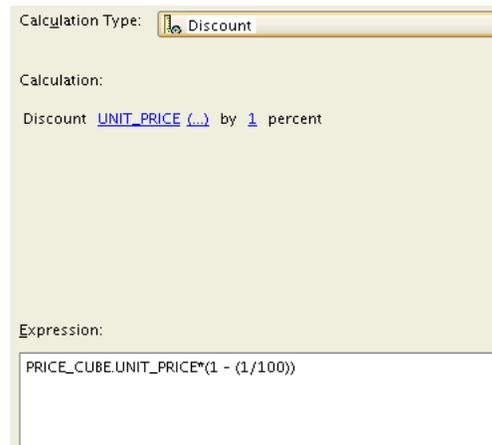


Figure 1–6 shows the template portion of the General tab that appears when a user selects Discount from the tree.

**Figure 1–6 Displaying a New Template**



The XML document in Example 1–6 defines the My New Calcs folder and the Discount and the Average calculations. The document contains three elements: <AWMCalcs>, <Category>, and <Calc>. <AWMCalcs> is the root element. It can have one or more <Category> elements. In this example, the <Category> element defines a folder named My New Calcs.

A <Category> element can have one or more <Calc> elements. This document has two <Calc> elements named Discount and Average.

A <Calc> element has four attributes: name, description, ui, and expression. Each attribute takes a quoted string as a value. The ui element consists of literal text

and hypertext links. You create the links by entering one of several available parameters.

[Example 1–6](#) uses the {measure} and {number} parameters. You use the same parameters in the expression attribute as bind variables, which pass the user choices to the calculation. Notice that the Average calculation uses two {measure} parameters. The expression attribute refers to them by their order in the ui attribute: {measure:1} and {measure:2}.

Refer to "[Reference: Elements for Calculations](#)" on page 1-14 for full descriptions of these elements.

### **Example 1–6 Creating the Discount and Average Calculation Templates**

```
<?xml version="1.0" encoding="UTF-8" ?>
<AWMCalcs>
  <Category name="NEW_CALCS" description="My New Calcs">
    <Calc
      name="Discount"
      description="Discount"
      ui="Discount {measure} by {number} percent"
      expression="{measure}*(1 - ({number}/100))"/>
    <Calc
      name="Average"
      description="Average"
      ui="Average of {measure} and {measure}"
      expression="({measure:1}+{measure:2})/2"/>
  </Category>
</AWMCalcs>
```

## **Adding an Option to a Calculation**

You can add an option that changes the basic calculation. The option appears as a check box in the Create Calculated Measure dialog box. Users select the option to create the modified calculation.

Two elements support these options: <CalcOptional> and <CalcOptionalDefinitions>. You can define a <CalcOptional> element locally or globally. Within a <Calc> element, <CalcOptional> applies only to that particular calculation. Within a <CalcOptionalDefinitions> element, <CalcOptional> applies to all calculations that reference it by name.

[Figure 1–7](#) shows the sample calculation with an option of truncating the values of the measure to whole numbers. The user has changed the percentage value to 6.

**Figure 1–7 Providing an Option to a Calculation**

Calculation Type:

Calculation:

Discount [UNIT\\_PRICE \(..\)](#) by  percent.

Truncate the decimal places

Expression:

```
TRUNC(PRICE_CUBE.UNIT_PRICE*(1 - (6/100)))
```

[Example 1–7](#) shows the Truncate option defined locally in a `<Calc>` element. The option applies only to the Discount calculation.

**Example 1–7 Adding an Option to One Calculation**

```
<Calc
  name="Discount"
  description="Discount"
  ui="Discount {measure} by {number} percent"
  expression="{measure}*(1 - ({number}/100))">
  <CalcOptional
    name="truncate"
    type="boolean"
    text="Truncate the decimal places"
    expression="TRUNC($expression)" />
</Calc>
```

[Example 1–8](#) shows the Truncate option defined globally in the `<CalcOptionalDefinitions>` element. The option is used by the Discount and the Average calculations, and it is available to any other calculations that might be defined.

**Example 1–8 Adding an Option to Multiple Calculations**

```
<AWMCalcs>
  <CalcOptionalDefinitions>
    <CalcOptional
      name="truncate"
      type="boolean"
      text="Truncate the decimal places"
      expression="TRUNC($expression)" />
  </CalcOptionalDefinitions>
  <Category name="NEW_CALCS" description="My New Calcs">
    <Calc
      name="Discount"
      description="Discount"
      ui="Discount {measure} by {number} percent"
      expression="{measure}*(1 - ({number}/100))">
      <CalcOptional name="truncate" />
    </Calc>
    <Calc
      name="Average"
```

```

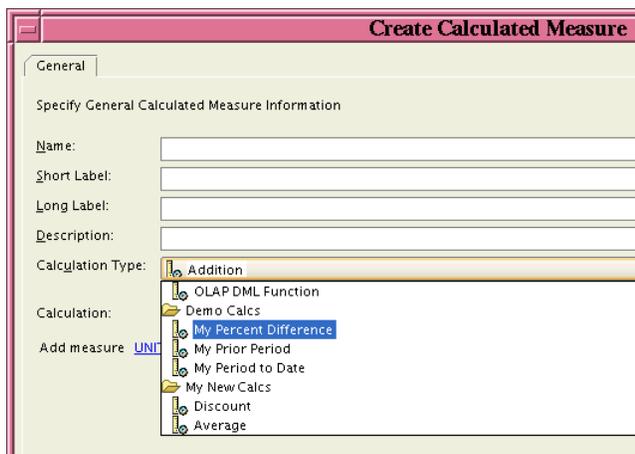
description="Average"
ui="Average of {measure} and {measure}"
expression="({measure:1}+{measure:2})/2">
<CalcOptional name="truncate"/>
</Calc>
</Category>
</AWMCalcs>

```

## Creating More Complex Calculation Templates

This example creates five calculations in two folders. The calculations in both folders use the global options defined at the beginning of the XML document. [Figure 1-8](#) shows the calculations as they appear in the Calculation Type list.

**Figure 1-8 Listing More New Calculations**



The My Period To Date calculation has the most complex syntax, including three lists. Using the `<Params>` element, you can create the lists quickly. This is the definition of the first list, which is displayed in [Figure 1-9](#):

```

<Params>
  <Param type="list" name="timePeriods">
    <Item expression="GREGORIAN YEAR" text="Gregorian year"/>
    <Item expression="GREGORIAN QUARTER" text="Gregorian quarter"/>
    <Item expression="GREGORIAN MONTH" text="Gregorian month"/>
    <Item expression="GREGORIAN WEEK" text="Gregorian week"/>
    <Item expression="ANCESTOR AT LEVEL {level}" text="Ancestor at level"/>
  </Param>
</Params>

```

**Figure 1–9 Choice Lists In a Calculation Template**

Example 1–9 shows the complete XML document that defines the five calculation templates shown in Figure 1–8.

### Example 1–9 Sample AWMCalcs Document

```
<?xml version="1.0" encoding="UTF-8" ?>

<AWMCalcs>
  <Params>
    <Param type="list" name="timePeriods">
      <Item expression="GREGORIAN YEAR" text="Gregorian year"/>
      <Item expression="GREGORIAN QUARTER" text="Gregorian quarter"/>
      <Item expression="GREGORIAN MONTH" text="Gregorian month"/>
      <Item expression="GREGORIAN WEEK" text="Gregorian week"/>
      <Item expression="ANCESTOR AT LEVEL {level}" text="Ancestor at level"/>
    </Param>
    <Param type="list" name="aggOps">
      <Item expression="SUM" text="sum"/>
      <Item expression="MAX" text="maximum"/>
      <Item expression="MIN" text="minimum"/>
      <Item expression="AVG" text="average"/>
    </Param>
  </Params>
  <CalcOptionalDefinitions>
    <CalcOptional
      name="percentages"
      type="boolean"
      text="Multiply by 100"
      expression="({expression$}) * 100"/>
    <CalcOptional
      name="truncate"
      type="boolean"
      text="Truncate the decimal places"
      expression="TRUNC({expression$})" />
  </CalcOptionalDefinitions>
  <Category name="DEMO_CALCS" description="Demo Calcs">
    <Calc
      name="PctDif"
      description="My Percent Difference"
      ui="Percent difference between {measure} and {measure}."
      expression="({measure:1} - {measure:2}) / abs({measure:2})">
      <CalcOptional name="percentages" />
    </Calc>
  </Category>
</AWMCalcs>
```

```

</Calc>
<Calc
  name="PriorPeriod"
  description="My Prior Period"
  ui="Prior period for measure {time_measure} in the {time_dimension}
    dimension and {hierarchy} hierarchy {number} period(s) ago."
  expression="LAG({time_measure},{number}) over hierarchy ({hierarchy})" />
<Calc
  name="Periodtodate"
  description="My Period to Date"
  ui="{timePeriods} to date for {time_measure} in the {time_dimension}
    dimension and {hierarchy} hierarchy. Aggregate over {timePeriods}
    using {aggOps} from the {calcRange} of the period."
  expression="{aggOps}({time_measure}) OVER HIERARCHY ({hierarchy}
    BETWEEN {calcRange} WITHIN {timePeriods})">
  <Param type="list" name="calcRange">
    <Item expression="UNBOUNDED PRECEDING AND CURRENT MEMBER"
      text="beginning"/>
    <Item expression="CURRENT MEMBER AND UNBOUNDED FOLLOWING"
      text="end"/>
  </Param>
  <CalcOptional name="truncate"/>
</Calc>
</Category>
<Category name="NEW_CALCs" description="My New Calcs">
  <Calc
    name="Discount"
    description="Discount"
    ui="Discount {measure} by {number} percent."
    expression="{measure}*(1 - ({number}/100))">
    <CalcOptional name="truncate"/>
  </Calc>
  <Calc
    name="Average"
    description="Average"
    ui="Average of {measure} and {measure}"
    expression="({measure:1}+{measure:2})/2">
    <CalcOptional name="truncate"/>
  </Calc>
</Category>
</AWMCalcs>

```

## Reference: Elements for Calculations

An XML document for calculations has the basic format shown in [Example 1–10](#).

### **Example 1–10 Basic XML Structure for Calculations**

```

<AWMCalcs>
  <Category>
    <Calc>

```

[Example 1–11](#) expands on this basic structure to include the definition of options in the calculations.

### **Example 1–11 XML Structure for Calculations With Options**

```

<AWMCalcs>
  <CalcOptionalDefinitions>
    <CalcOptional>

```

```

    <Category>
      <Calc>
        <CalcOptional>

```

**Example 1–12** expands the basic structure to include choice lists in the user interface.

**Example 1–12 XML Structure for Calculations With Choice Lists**

```

<AWMCalcs>
  <Params>
    <Param>
      <Item>
    <Category>
      <Calc>
        <Param>
          <Item>

```

Following are the descriptions of the elements.

### <AWMCalcs>

The root element that identifies this document as containing the custom calculation templates for Analytic Workspace Manager. It contains a <Category> element, and can also contain a <CalcOptionalDefinitions> element, a <Params> element, or both.

#### Contains

<CalcOptionalDefinitions>, <Category>, <Params>

#### Attributes

None

### <Calc>

Describes a calculation template. It can contain a <CalcOptional> element, or one or more <Param> elements, or both.

#### Contains

<CalcOptional>, <Param>

#### Attributes

<Calc> has the following attributes:

- **name:** A unique name for the calculation, which conforms to the same naming conventions as other OLAP objects.
- **description:** A description of the calculation. Analytic Workspace Manager adds the description to the list of calculation templates.
- **ui:** The text of the template, which Analytic Workspace Manager displays in the Calculation Type list of the Create Calculated Measure dialog box. Enclose hypertext parameters in braces {}. [Table 1–1](#) describes the valid parameters.
- **expression:** The calculation that is executed by the calculated measure. This calculation is defined using the expression syntax and by using as bind variables the hypertext parameters from the ui attribute. Enclose the bind variables in braces {}. If the ui attribute uses the same parameter two or more times, then

reference them by the order they appeared, such as {measure:1} and {measure:2}. For a simple example, see [Example 1-6](#) on page 1-10.

**Table 1-1 Parameters for the ui Attribute**

Parameter	Description
ATTRIBUTE	Lists the attributes of the selected dimension.
DIMENSION	Lists all dimensions of the current cube.
DIMENSION_MEMBER	Lists the members of the selected dimension.
HIERARCHY	Lists the hierarchies of the selected dimension.
HIERARCHY_LEVEL	Lists the levels of the selected hierarchy.
LEVEL	Lists the levels of the selected dimension.
LIST	Displays a list of values specified in this format: <code>LIST:expression=value;[expression=value;...]</code> Alternatively, use the <Params> element.
MEASURE	Lists all measures in the analytic workspace with at least one dimension in common with the current cube.
NUMBER	Displays a text field that accepts numeric input.
TEXT_INPUT	Displays a text field that accepts any text input.
TIME_DIMENSION	Lists the time dimensions of the current cube.
TIME_MEASURE	Lists all measures for cubes that have a time dimension.
VALUE	Displays the current selection from a LIST parameter.
<i>param</i>	A parameter defined in a <Param> element. Specifically, the value of the text attribute of an <Item> element.

### <CalcOptional>

Defines a check box that can be used by one or more calculations to modify the basic expression. For example, a calculation that generates a fraction might offer a Multiply By 100 option to return the results as a percentage.

Include a <CalcOptional> element in a <Calc> element where you want a check box to appear. You can fully define the option within the <Calc> element, or you can define the option within a <CalcOptionalDefinitions> element and reference it by name with a second <CalcOptional> element in the <Calc> element.

A <Calc> element can contain a <CalcOptional> element. A <CalcOptionalDefinitions> element can have one or more <CalcOptional> elements.

### Contains

None

### Attributes

<CalcOptional> has the following attributes:

- **name:** A unique name for the option, which conforms to the same naming conventions as other OLAP objects.
- **type:** The data type of the option, which is always `boolean`.
- **text:** A description of the option. This text labels the check box.

- **expression:** The calculation that is executed when the option is selected. Use the expression syntax and (`$expression$`) for the basic calculation defined by the current `<Calc>` element.

### **<CalcOptionalDefinitions>**

Contains one or more `<CalcOptional>` elements so they can be referenced by multiple calculations. This element must appear directly after `<AWMCalcs>`.

#### **Contains**

`<CalcOptional>`

#### **Attributes**

None

### **<Category>**

Defines a heading in the list of calculations in Analytic Workspace Manager. It contains one or more `<Calc>` elements.

#### **Contains**

`<Calc>`

#### **Attributes**

`<Category>` has the following attributes:

- **name:** A unique name for the category, which conforms to the same naming conventions as other OLAP objects.
- **description:** A description of the category. Analytic Workspace Manager adds this description to the list of calculation templates.

### **<Item>**

Describes an entry in a list of values or numbers.

#### **Contains**

None

#### **Attributes**

`<Item>` has the following attributes:

- **text:** Value entered in the `<Calc>` `ui` attribute and displayed to users.
- **expression:** Value inserted in the `<Calc>` `expression` attribute when a user selects the item.

### **<Param>**

Describes a list of values or a number field referenced in a `<Calc>` `ui` attribute. An expression that corresponds to the choice made by the user is entered in the calculation instead of the displayed value. This element contains one or more `<Item>` elements.

#### **Contains**

`<Item>`

### **Attributes**

<Param> has the following attributes:

- **type:** Either LIST for a list of values, or NUMBER for a field for entering a number.
- **name:** The name of the parameter, which is referenced in the <Calc> ui attribute.
- **default:** Provides the default value when a user enters a number that has no corresponding expression in an <Item> element.

### **<Params>**

Contains one or more <Param> elements.

### **Contains**

<Param>

### **Attributes**

None

---

---

# Introducing Analytic Workspace Manager Plug-ins

An Analytic Workspace Manager plug-in enables you to run Java code in the context of Analytic Workspace Manager. With an implementation of a Java plug-in interface that is supported by Oracle Analytic Workspace Manager, Version 11.2, you can extend the functionality of Analytic Workspace Manager in Oracle Database 11g Release 2 (11.2) with the OLAP option.

This chapter has the following topics.

- [Describing Analytic Workspace Manager Plug-ins](#)
- [Describing the AWMPlugin Interface](#)
- [Describing the ViewerPlugin and EditorPlugin Interfaces](#)
- [Steps in Creating a Plug-in](#)
- [Describing the Available Plug-ins](#)

## Describing Analytic Workspace Manager Plug-ins

Analytic Workspace Manager has the following Java plug-in interfaces.

- `AWMPlugin`, which you can use to add selections to the right-click menu of a navigation tree object.
- `ViewerPlugin`, which you can use to display information in the property inspector about the current navigation tree object.
- `EditorPlugin`, which extends `ViewerPlugin` and adds the ability to edit properties of the object.

With an Analytic Workspace Manager plug-in, you can implement programs that perform actions such as the following:

- Create new types of calculations.
- Create forecasts.
- Create custom OLAP metadata objects, such as an enterprise-specific time dimension.
- Control the display in the property inspector of the information associated with a custom navigation tree object.
- Edit the properties of an object in the property inspector.

In an Analytic Workspace Manager plug-in, you can use the following Java APIs:

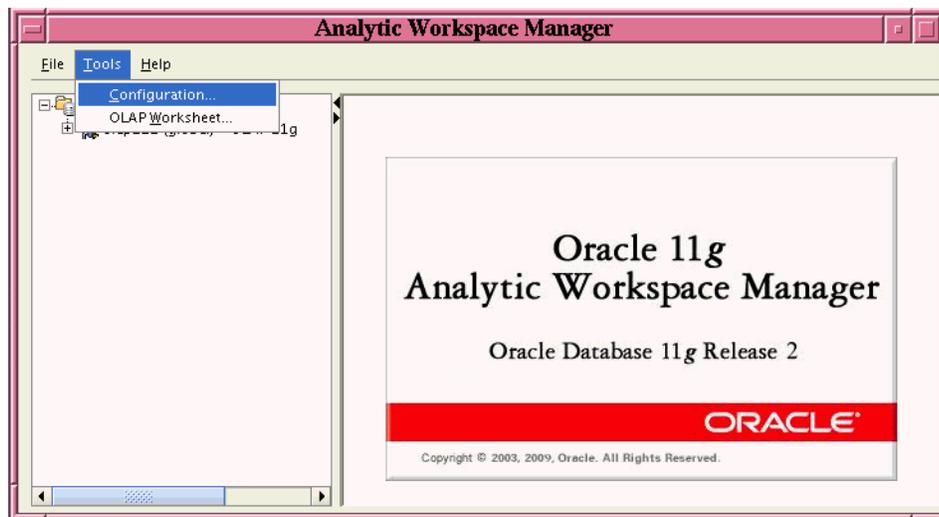
- Oracle OLAP Java API
- JDBC API
- Swing API

You can invoke OLAP DML or SQL procedures by using JDBC classes.

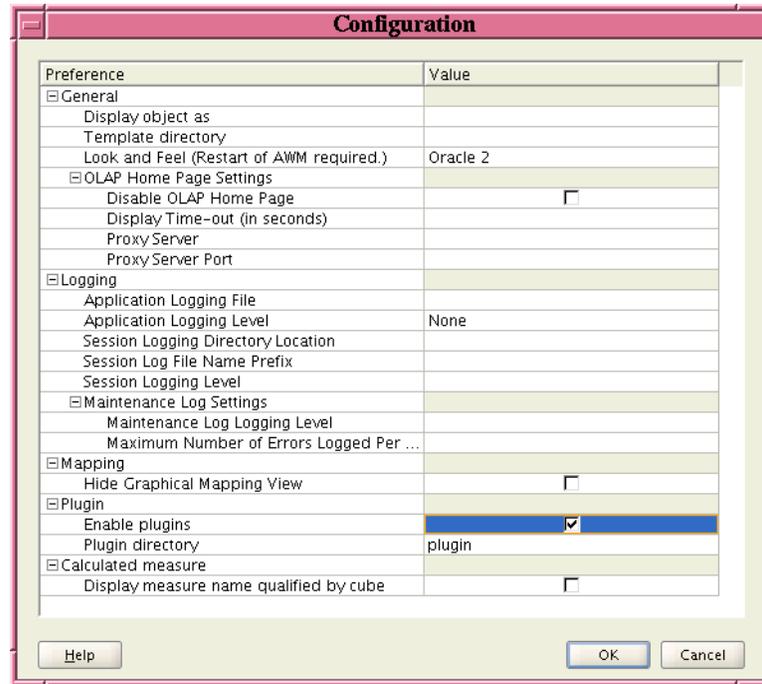
## Enabling Analytic Workspace Manager Plug-ins

Analytic Workspace Manager has a configuration option that specifies whether it uses plug-ins. To enable plug-ins, from the Analytic Workspace Manager **Tools** menu, select **Configuration**, as shown in [Figure 2-1](#). In the Configuration dialog box, select **Enable Plugins** and specify the directory that contains your plug-ins, as shown in [Figure 2-2](#). Click **OK** and then exit and restart Analytic Workspace Manager.

**Figure 2-1** Configuration Item on the Tools Menu



[Figure 2-2](#) shows the Configuration dialog box with **Enable plugins** selected and with `plugin` as the value for **Plugin directory**. The value should include the full path to the plug-in directory unless the directory is a subdirectory of the `Oracle_home/olap/awm` directory, which is the case for the `plugin` directory shown in the figure.

**Figure 2–2 Configuration Dialog Box with Enable Plugins Selected**

## How Analytic Workspace Manager Calls a Plug-in

If Analytic Workspace Manager has plug-ins enabled, then on startup Analytic Workspace Manager dynamically loads Java code from JAR files located in the plug-ins directory. After loading the contents of the JAR files, Analytic Workspace Manager looks for classes that implement the `AWMPlugin`, `ViewerPlugin`, or `EditorPlugin` interfaces. It also looks for `aw.xml`, `cube.xml`, `dimension.xml`, and `schema.xml` files to add objects to the navigation tree.

---

**Note:** You can include multiple plug-ins and XML documents in a single JAR file.

---

When Analytic Workspace Manager calls most methods of a plug-in, it passes the method a `java.sql.Connection` object as the `conn` parameter. The `Connection` represents the current connection to the Oracle Database instance.

Analytic Workspace Manager does not pass any user identification or password to the plug-in. It only passes the connection object. An Analytic Workspace Manager plug-in does not allow you to do anything that you cannot do by writing a standalone Java program. For information on the parameters that Analytic Workspace Manager passes to the methods of plug-ins, see "[Describing the AWMPlugin Interface](#)" on page 2-5 and "[Describing the ViewerPlugin and EditorPlugin Interfaces](#)" on page 2-11.

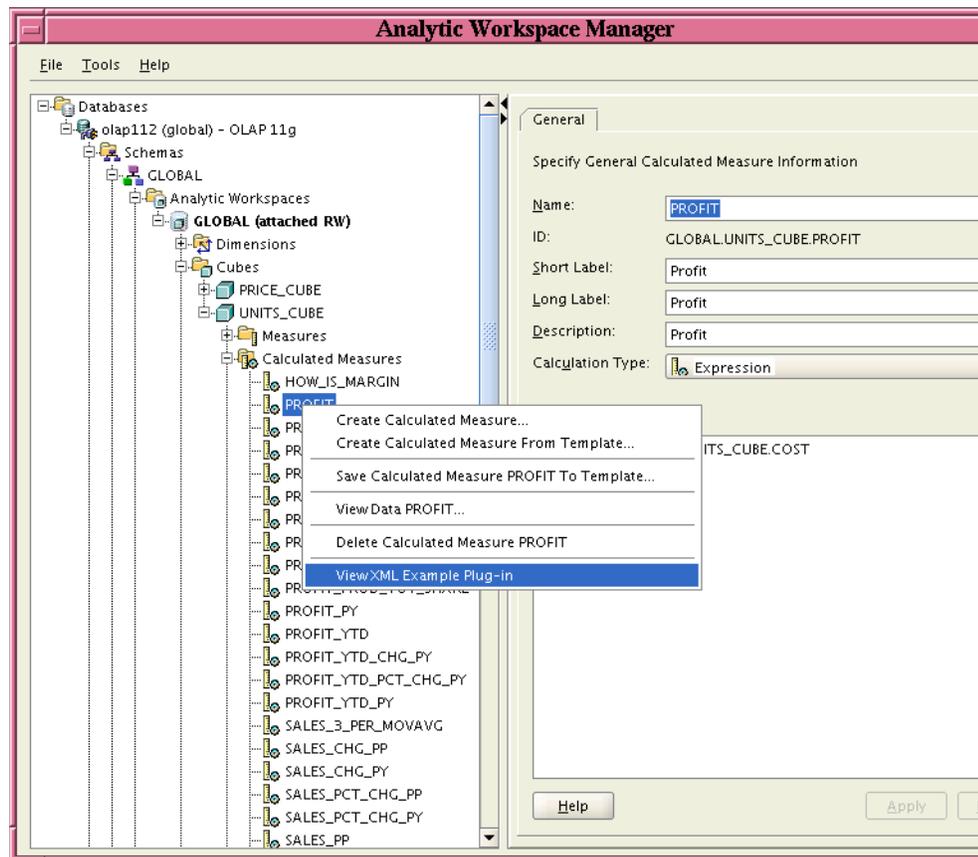
## Calling an AWMPlugin

When a user right-clicks an object in the Analytic Workspace Manager navigation tree, a menu appears that presents the actions available for the object. The menu also displays options supplied by the `AWMPlugin` plug-ins that apply to the object. An `AWMPlugin` uses the `isSupported` method to indicate whether it applies to an object in the tree. Because Analytic Workspace Manager calls the `isSupported` method of

each plug-in whenever the user right-clicks a navigation tree object, an `isSupported` method should return quickly.

The menu displays the text returned by the `getMenu` method of the plug-in. [Figure 2-3](#) shows the menu that Analytic Workspace Manager displays when a user right-clicks a calculated measure in the tree. The menu includes the `ViewXMLPlugin` example plug-in. For the code of the plug-in example, see [Example 3-1](#) on page 3-2.

**Figure 2-3 Right-click Menu of the Navigation Tree for a Calculated Measure**



If the user selects the plug-in, then Analytic Workspace Manager calls the `handle` method of the plug-in. The `handle` method specifies the actions that the plug-in performs. The `refreshTree` method of the plug-in indicates whether Analytic Workspace Manager refreshes the navigation tree to include any new objects created by the plug-in or to remove objects deleted by the plug-in.

### Calling a `ViewerPlugin` or `EditorPlugin`

As described in ["Creating Reports in Object Folders"](#) on page 1-5, with certain XML documents you can add objects to the Schemas, Analytic Workspaces, Dimensions, and Cubes folders in the Analytic Workspace Manager navigation tree. You add objects to the navigation tree by adding `<AWMNode>` elements to the XML document. If an `<AWMNode>` specifies a `ViewerPlugin` or an `EditorPlugin`, then Analytic Workspace Manager calls the plug-in when the user selects the navigation tree object that corresponds to the `<AWMNode>`.

With the `sql` attribute of an `<AWMNode>` element, you can specify a SQL `SELECT` statement. Analytic Workspace Manager displays the result of the statement either in the folder in the navigation tree or in the property inspector, or in both places. For more information about creating the XML documents and the SQL statements, see ["Creating Reports in Object Folders"](#) on page 1-5.

To control the display of the information in the property inspector or to enable the user to edit properties of the selected navigation tree object, you can use a `ViewerPlugin` or `EditorPlugin`. You use the `viewClass` attribute of an `<AWMNode>` element to specify the plug-in. In the plug-in you can use the Oracle OLAP Java API to retrieve OLAP objects or alter characteristics of them. You can also specify user interface elements for the display in the property inspector.

## Describing the AWMPPlugin Interface

The following is the `oracle.olap.awm.plugin.AWMPPlugin` interface.

```
package oracle.olap.awm.plugin

import java.awt.Frame;
import java.sql.Connection;
import java.util.Map;
import oracle.AWXML.AW;

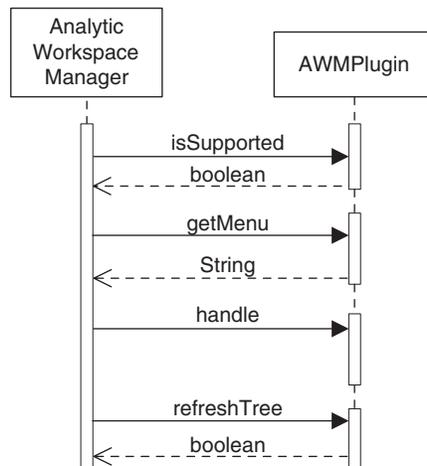
public interface AWMPPlugin
{
    boolean isSupported(Connection conn, String type, Object obj, AW aw,
                       Map params);

    String getMenu(Connection conn, String type, Object obj, AW aw,
                  Map params);

    void handle(Frame parent, Connection conn, String type, Object obj,
                AW aw, Map params);

    boolean refreshTree(Connection conn, String type, Object obj, AW aw,
                       Map params);
}
```

When a user right-clicks an object in the navigation tree, Analytic Workspace Manager calls the methods of classes that implement the `AWMPPlugin` interface in the sequence illustrated in [Figure 2-4](#).

**Figure 2–4 Sequence of Calls to an AWMPPlugin**

Analytic Workspace Manager first calls the `isSupported` method. If that method returns `true`, then Analytic Workspace Manager calls `getMenu` and displays on the right-click menu the value that `getMenu` returns. If a user selects the menu item, then Analytic Workspace Manager calls the `handle` and `refreshTree` methods. The input parameters that Analytic Workspace Manager passes to the `AWMPPlugin` methods are the following:

- `conn`, which is a `java.sql.Connection` object that represents the current connection to the Oracle Database instance.
- `type`, which is a `java.lang.String` that is a type designation that Analytic Workspace Manager assigns to the object. For a description of `type` parameter values, see ["Values for the type and obj Parameters"](#) on page 2-6.
- `obj`, which is a `java.lang.Object` that Analytic Workspace Manager associates with the object selected in the Analytic Workspace Manager navigation tree. The `Object` can be a `String` or an object from the Oracle OLAP Java API. For more information on the `obj` parameter values, see ["Values for the type and obj Parameters"](#).
- `aw`, which is `null`. This parameter exists for compatibility with 10g plug-ins, for which `aw` was an `oracle.AWXML.AW` object.
- `params`, which is a `java.util.Map` that contains objects and information that the plug-in can use. For a description of the `Map` keys and values, see ["Elements in the params Map for an AWMPPlugin"](#) on page 2-9.
- `parent`, which is a `java.awt.Frame` object that Analytic Workspace Manager passes to the `handle` method. The plug-in can use this object as the parent frame for user interface components.

## Values for the type and obj Parameters

For the `type` parameter of the methods of an `AWMPPlugin` implementation, Analytic Workspace Manager passes to the plug-in a label that identifies the type of the navigation tree object for which the plug-in is invoked. For the `obj` parameter of the methods, Analytic Workspace Manager passes an `Object`, which is a `java.lang.String` or an OLAP metadata object.

A plug-in can use the `type` value to distinguish between the navigation tree objects that are associated with the same metadata object. For example, for all of the folder objects in a Dimensions folder, such as Levels and Hierarchies, Analytic Workspace Manager passes as the `obj` parameter the same `MdmPrimaryDimension` object, but it passes a different `type` label for each folder object.

Custom objects that you add with an XML document appear in the navigation tree at the level specified by the XML document. For example, a top-level `<AWMNode>` in a `dimension.xml` document appears in the Dimensions folder of an analytic workspace. For an AWMPugin implementation specified by an `<AWMNode>` element, the `type` parameter value has the prefix `AWMTree_` followed by the value of the `name` attribute of the parent `<AWMNode>`. The `obj` parameter value is the run-time value of the `type` attribute of the `<AWMNode>`.

Table 2–1 shows the `type` parameter values and `obj` parameter objects that Analytic Workspace Manager passes to the plug-in for the selected navigation tree object. The indentation of objects in the Navigation Tree Object column indicates the hierarchy of the tree. Text in italics indicates a variable object name. The `obj` parameter objects are `String` objects or OLAP metadata objects. The AW object is an `oracle.olapi.metadata.deployment.AW` object. The other metadata objects, such as `MdmStandardDimension` and `MdmCube`, are classes in the `oracle.olapi.metadata.mdm` package. The Reports object and all of the objects under it have the same type.

**Table 2–1 Type Values and Objects for Navigation Tree Objects**

Navigation Tree Object	type Parameter Value	obj Parameter Object
Databases	Databases	Databases
<i>Database name</i>	DATABASE	<i>Database identifier</i>
Schemas	SCHEMA_FOLDER	<i>Database identifier</i>
<i>Schema name</i>	SCHEMA	<i>Schema name</i>
Analytic Workspaces	WORKSPACE_FOLDER	<i>Schema name</i>
<i>Analytic workspace name</i>	WORKSPACE	AW
Dimensions	DIMENSION_FOLDER	AW
<i>Dimension name</i>	DIMENSION	MdmStandardDimension or MdmTimeDimension
Levels	DIMENSION_LEVEL_FOLDER	MdmStandardDimension or MdmTimeDimension
<i>Level name</i>	DIMENSION_LEVEL	MdmDimensionLevel
Hierarchies	DIMENSION_HIERARCHY_FOLDER	MdmStandardDimension or MdmTimeDimension
<i>Hierarchy name</i>	DIMENSION_HIERARCHY	MdmLevelHierarchy or MdmValueHierarchy
Attributes	DIMENSION_ATTRIBUTE_FOLDER	MdmStandardDimension or MdmTimeDimension
<i>Attribute name</i>	DIMENSION_ATTRIBUTE	MdmBaseAttribute
Mappings	DIMENSION_MAP	MdmStandardDimension or MdmTimeDimension
Views	DIMENSION_VIEW_FOLDER	MdmStandardDimension or MdmTimeDimension
<i>View name</i>	DIMENSION_VIEW	MdmStandardDimension or MdmTimeDimension

**Table 2–1 (Cont.) Type Values and Objects for Navigation Tree Objects**

<b>Navigation Tree Object</b>	<b>type Parameter Value</b>	<b>obj Parameter Object</b>
Data Security <i>dimension.xml object</i>	DATA_SECURITY AWMTree_parent_node_name	MdmStandardDimension or MdmTimeDimension For a folder, the name of the <AWMNode>. For a value returned by the SQL query, the run-time object name.
Cubes	CUBE_FOLDER	AW
<i>Cube name</i>	CUBE	MdmCube
Measures	CUBE_MEASURE_FOLDER	MdmCube
<i>Measure name</i>	CUBE_MEASURE	MdmBaseMeasure
Calculated Measures	CUBE_DERIVED_MEASURE_FOLDER	MdmCube
<i>Calculated measure name</i>	CUBE_DERIVED_MEASURE	MdmDerivedMeasure
Mappings	CUBE_MAP	MdmCube
Views	CUBE_VIEW_FOLDER	MdmCube
<i>View name</i>	CUBE_VIEW	MdmCube
Cube Scripts	CUBE_SCRIPT_FOLDER	MdmCube
<i>Cube script name</i>	CUBE_SCRIPT	<i>Script_name</i>
Data Security <i>cube.xml object</i>	DATA_SECURITY AWMTree_parent_node_name	MdmCube For a folder, the name of the <AWMNode>. For a value returned by the SQL query, the run-time object name.
Measure Folders	MEASURE_FOLDER_FOLDER	AW
<i>Measure folder name</i>	<i>Measure_folder_name</i>	MdmOrganizationalSchema
Languages	LANGUAGE	Languages
<i>aw.xml object</i>	AWMTree_parent_node_name	For a folder, the name of the <AWMNode>. For a value returned by the SQL query, the run-time object name.
OLAP DML Programs	AWMTREE_OLAP DML Programs	OLAP DML Programs
<i>Program name</i>	AWMTREE_OLAP DML Programs	<i>Program_name</i>
Maintenance Scripts	MAINTENANCE_SCRIPT_FOLDER	<i>Schema name</i>
<i>Script name</i>	MAINTENANCE_SCRIPT	<i>Script name</i>
Maintenance Reports	AWMTREE_Maintenance Reports	Maintenance Reports
<i>Maintenance_report_name</i>	AWMTREE_maintenance_report_name	<i>Maintenance_report_name</i>
<i>schema.xml object</i>	AWMTree_parent_node_name	For a folder, the name of the <AWMNode>. For a value returned by the SQL query, the run-time object name.
Data Security Roles	ACL_DOCUMENT_FOLDER	Data Security Roles
<i>Security role name</i>	<i>Security role name</i>	<i>Security role name</i>
Reports	AWMTREE_Reports	Reports
<i>Report name</i>	AWMTREE_report_name	<i>Report name</i>

## Elements in the params Map for an AWMPlugin

The params Map contains information about the navigation tree object that is currently selected. [Table 2–2, "Keys and Values of the params Map for a Non-custom Object"](#) and [Table 2–3, "Keys and Values of the params Map for a Custom Object"](#) contain descriptions of the keys and values of the elements of the Map for an AWMPlugin. The keys are String objects.

The params Map for the Database folder does not have a DATASOURCE, DATAPROVIDER, or GETDATAPROVIDER key. The params Map objects for the higher level navigation tree objects, those above the individual analytic workspaces, have a null value for the DATAPROVIDER key until the user selects a tree object that requires OLAP metadata. Other than for those exceptions, the params Map for a navigation tree object has the keys and values listed in the tables.

### Params Map Elements for Non-custom Objects

[Table 2–2](#) lists the keys and values of the elements of the params Map for non-custom navigation tree objects. Custom navigation tree objects are specified by an <AWMNode> element in a SQL Report XML document and have a type that begins with the prefix AWMTree.

**Table 2–2** *Keys and Values of the params Map for a Non-custom Object*

Key	Value
AWM_VERSION	A String that is the version number of Analytic Workspace Manager.
DATAPROVIDER	An <code>oracle.olapi.metadata.mdm.MdmMetadataProvider</code> that is the metadata provider for the session.
BIND_MAP	An empty Map.
DATASOURCE	A <code>java.sql.DataSource</code> .
GETDATAPROVIDER	An implementation of the <code>oracle.olap.awm.plugin.OLAPDataProvider</code> interface. The interface specifies a method that gets an <code>MdmMetadataProvider</code> .

### Params Map Elements for Custom Objects

[Table 2–3](#) lists the keys and values of the elements of the params Map for custom navigation tree objects. A custom object is specified by an <AWMNode> element in a SQL Report XML document.

**Table 2–3** *Keys and Values of the params Map for a Custom Object*

Key	Value
AWM_VERSION	A String that is the version number of Analytic Workspace Manager.
DATAPROVIDER	An <code>oracle.olapi.metadata.mdm.MdmMetadataProvider</code> that is the metadata provider for the session.
BIND_MAP	A <code>java.util.Map</code> that contains bind variables from the <AWMNode> element and from the parent of the element, and from Analytic Workspace Manager.
DATASOURCE	A <code>java.sql.DataSource</code> object.
GETDATAPROVIDER	An implementation of the <code>oracle.olap.awm.plugin.OLAPDataProvider</code> interface. The interface specifies a method that gets an <code>MdmMetadataProvider</code> .
ISFOLDER	A String that is <code>TRUE</code> if the <AWMNode> that specifies the plugin-in is a folder or <code>FALSE</code> if it is not.

**Table 2–3 (Cont.) Keys and Values of the params Map for a Custom Object**

Key	Value
NODE_TYPE	For a nested <AWMNode>, a String that is the name of the parent <AWMNode>. For an <AWMNode> that is a folder, the name of the node.
TYPE	A String that is the value of the type attribute of the <AWMNode> that specifies the plug-in.

The BIND\_MAP Map contains bind variables that are associated with the navigation tree object that is currently selected. Table 2–4 contains descriptions of the keys and values in the BIND\_MAP Map.

This Map includes the bind variables that appear in the SQL statements of the <AWMNode> and the parent <AWMNode>. It also includes other bind variables for the currently selected object in the navigation tree.

The keys are String objects. A bind variable is specified by the type attribute of an <AWMNode> element of a custom navigation tree object or is set internally by Analytic Workspace Manager. A plug-in gets the run-time value of the bind variable from the BIND\_MAP Map. For examples of bind map Map keys and values, see Table 2–6 on page 2-11.

---

**Note:** When you reference the key for a bind variable in your plug-in, be sure to use lowercase, as in {owner} or {measureobj} or {dimension\_name}.

---

**Table 2–4 Keys and Values of the BIND\_MAP Map**

Key	Value
aw_name	A String that contains the name of the currently selected analytic workspace.
owner	A String that contains the name of the owner of the currently selected analytic workspace.
schema	A String that contains the name of the owner of the currently selected schema.
user	A String that contains the name of the user who is connected to the database.
Other bind variables	One or more elements, each of which has a bind variable as a key and has the run-time value of the bind variable as the value.  Examples of other bind variable keys are dimension_name and cube_name. For examples of other bind variables that can be in the Map see the "Example params Map Elements for an AWMPlugin" and the examples in Chapter 3, "Examples of Analytic Workspace Manager Plug-ins".

## Example params Map Elements for an AWMPlugin

Examples of the keys and values of a params Map for a custom object are in Table 2–5 and in Table 2–6. All of the values are String objects except those for the DATAPROVIDER and DATASOURCE keys.

Table 2–6 has the elements of the params Map that Analytic Workspace Manager passes to the methods of DeleteDimPlugin when the user right-clicks the CUSTOMER dimension in the MyDims folder, as shown in Figure 3–2 on page 3-9. The MyDims folder is created by the aw.xml document in Example 3–7 on page 3-15.

The figure shows the menu that `DeleteDimPlugin` displays. The property inspector in the figure has the output of `DimEditorPlugin`, because that plug-in is also activated when the user selects a dimension in the `MyDims` folder.

An example of getting a value from the `params` Map is the following line from the `isSupported` method in the `DeleteDimPlugin` class in [Example 3-2](#) on page 3-6.

```
Object nodeType = params.get("TYPE");
```

**Table 2-5** Keys and Values of the `params` Map for `DeleteDimPlugin`

Key	Value	Description
AW	An AW	The current analytic workspace object.
AWM_VERSION	11.2.0.2.0	The version number of Analytic Workspace Manager.
BIND_MAP	A Map	A container for bind variables related to the current object.
DATAPROVIDER	An <code>MdmMetadataProvider</code>	The metadata provider for the session.
DATASOURCE	A <code>DataSource</code>	The current data source.
GETDATAPROVIDER	An <code>OLAPDataProvider</code>	An implementation of the <code>OLAPDataProvider</code> interface.
ISFOLDER	FALSE	Specifies that the <code>&lt;AWMNode&gt;</code> is not a folder.
NODE_TYPE	<code>MyDims</code>	The name of the parent <code>&lt;AWMNode&gt;</code> .
TYPE	<code>dimobjj</code>	The type of the <code>&lt;AWMNode&gt;</code> that specifies the plug-in.

[Table 2-6](#) has the elements of the Map that is the value of the `BIND_MAP` key in the `params` Map. An example of getting a value from the `BIND_MAP` Map is the following lines from the `handle` method in the `DeleteDimPlugin` class in [Example 3-2](#).

```
Map bindMap = (Map)params.get("BIND_MAP");
...
String owner = (String)bindMap.get("owner");
```

**Table 2-6** Keys and Values of the `BIND_MAP` Map for `DeleteDimPlugin`

Key	Value	Description
<code>aw_name</code>	GLOBAL	The name of the current analytic workspace.
<code>dimobjj</code>	CUSTOMER	The run-time value of the dimension currently selected in the <code>MyDims</code> folder.
<code>owner</code>	GLOBAL	The name of the owner of the analytic workspace.
<code>schema</code>	GLOBAL	The name of the current schema.
<code>user</code>	global	The name of the current user.

## Describing the ViewerPlugin and EditorPlugin Interfaces

As described in ["Creating Reports in Object Folders"](#) on page 1-5, with certain XML documents you can add objects to the Schemas, Analytic Workspaces, Dimensions, and Cubes folders in the Analytic Workspace Manager navigation tree. You add objects to the navigation tree by adding `<AWMNode>` elements to an XML document.

With the `sql` attribute of an `<AWMNode>` element, you can specify a SQL `SELECT` statement. Analytic Workspace Manager displays the result of the statement either in the folder in the navigation tree or in the property inspector, or in both places. For more information about creating the XML documents and the SQL statements, see ["Creating Reports in Object Folders"](#).

With the `viewClass` attribute of an `<AWMNode>` element, you can specify a Java plug-in for viewing or editing database objects. You can add a viewer or an editor for relational objects or OLAP objects. Relational objects include tables, materialized views, and so on, and OLAP objects include dimensions, cubes, and so on. To add a viewer, have the `viewClass` attribute specify an implementation of the `ViewerPlugin` interface. To add an editor, have the `viewClass` attribute specify an implementation of the `EditorPlugin` interface. The viewer or editor plug-in displays in the property inspector.

## Describing the ViewerPlugin Interface

The following is the `oracle.olap.awm.plugin.ViewerPlugin` interface.

```
package oracle.olap.awm.plugin

import java.sql.Connection;
import java.util.Map;
import javax.swing.JPanel;

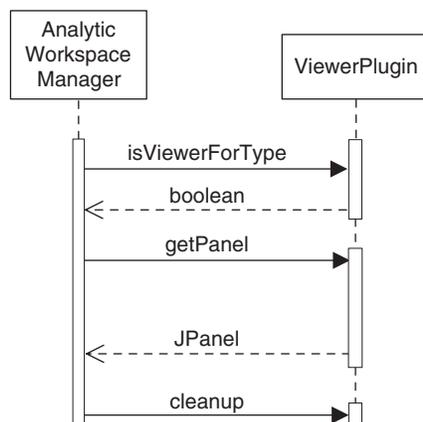
public interface ViewerPlugin
{
    public boolean isViewerForType(Connection conn, String name)
        throws Exception;

    public JPanel getPanel(Connection conn, String name, Map params)
        throws Exception;

    public void cleanup(String name);
}
```

When the Analytic Workspace Manager user selects the navigation tree object that is associated with the `ViewerPlugin`, Analytic Workspace Manager calls the methods of a `ViewerPlugin` in the sequence illustrated in [Figure 2-5](#).

**Figure 2-5** Sequence of Calls to a `ViewerPlugin`



Analytic Workspace Manager first calls the `isViewerForType` method and passes it the following parameters:

- `conn`, which is a `java.sql.Connection` object that represents the current connection to the Oracle Database instance.
- `name`, which is a `String` that contains the name of the `<AWMNode>` that is the parent of the `<AWMNode>` that has the `viewClass` attribute.

If the plug-in returns `true`, Analytic Workspace Manager calls the `getPanel` method and passes it the same `conn` and `name` parameters plus the following parameter.

- `params`, which is a `java.util.Map` object that contains information about the currently selected navigation tree object. The information includes the run-time values for attributes of the `<AWMNode>` element that has the `viewClass` attribute and from the parent `<AWMNode>`. The plug-in can use this information in specifying data to display or to retrieve from the database. The keys and values of the `Map` are described in [Table 2-7](#). For a description of the `Map` keys and values, see ["Elements in the params Map for a ViewerPlugin or EditorPlugin"](#) on page 2-14.

When the user selects a different navigation tree object, Analytic Workspace Manager calls the `cleanup` method of the plug-in. It passes the method the same `name` parameter. In this method you can perform any cleanup that your plug-in requires.

## Describing the EditorPlugin Interface

The `EditorPlugin` interface extends the `ViewerPlugin` interface. The following is the `oracle.olap.awm.plugin.EditorPlugin` interface.

```
package oracle.olap.awm.plugin

import java.awt.Component;
import java.sql.Connection;
import java.util.Map;

public interface EditorPlugin extends ViewerPlugin
{
    public void setValueChanged(Connection conn, String name, Map params,
        PanelChanged parent);

    public boolean validate(Connection conn, Component parent, String name,
        Map params) throws Exception;

    public boolean save(Connection conn, Component parent, String name,
        Map params) throws Exception;

    public void revert(Connection conn, Component parent, String name,
        Map params) throws Exception;

    public void showHelp(Connection conn, Component parent, String name,
        Map params) throws Exception;
}
```

For an `EditorPlugin`, Analytic Workspace Manager initially calls the `isViewerForType`, `setValueChanged`, and `getPanel` methods, as shown in [Figure 2-6, "Sequence of Calls to an EditorPlugin"](#). For an example of the display of an `EditorPlugin`, see [Figure 3-11](#) on page 3-22.

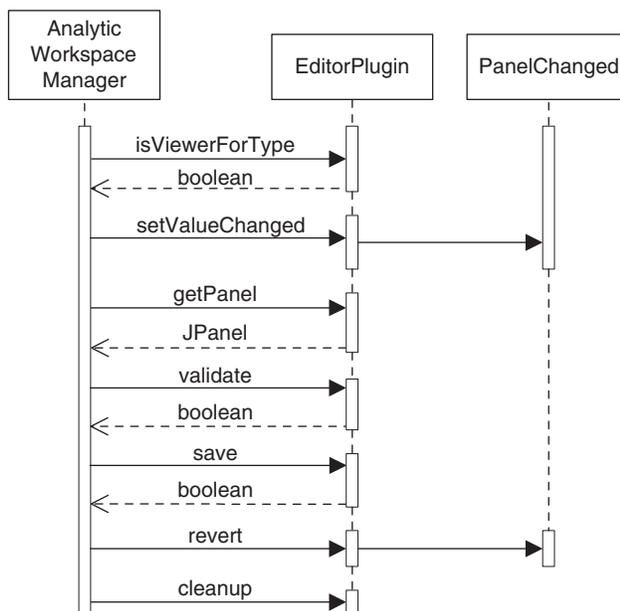
If the user makes a change in the property inspector, then the **Apply** and **Revert** buttons become active. If the user clicks **Apply**, then Analytic Workspace Manager

calls the `validate` method of the `EditorPlugin`. If the value is valid, then Analytic Workspace Manager calls the `save` method. If the user clicks **Revert**, then Analytic Workspace Manager calls `revert`. If the user clicks the **Help** button, then Analytic Workspace Manager calls `showHelp`.

All of the methods of an `EditorPlugin` have the same `conn`, `name`, and `param` parameters as the `getPanel` method. Those parameters are described in "Describing the ViewerPlugin and EditorPlugin Interfaces" on page 2-11. The methods also have the following additional parameter.

- parent**, which for the `setValueChanged` method is an implementation of the `oracle.olap.awm.plugin.PanelChanged` interface. That interface specifies a single method, `public void changed()`; . Whenever the user interacts with the editing field of your `EditorPlugin`, the `EditorPlugin` should call the `changed` method of the `PanelChanged` object. For the other `EditorPlugin` methods, the `parent` parameter is the parent component.

**Figure 2–6 Sequence of Calls to an EditorPlugin**



## Elements in the params Map for a ViewerPlugin or EditorPlugin

The `params Map` for a `ViewerPlugin` or an `EditorPlugin` does not contain a `BIND_MAP` Map. Instead, the bind variables are keys in the `params Map`. Table 2–7 contains descriptions of the keys and values of the elements of the Map for a `ViewerPlugin` or an `EditorPlugin`.

**Table 2–7 Keys and Values of the params Map for a ViewerPlugin or EditorPlugin**

Key	Value
AW	An <code>oracle.olapi.metadata.deployment.AW</code> .
aw_name	A String that contains the name of the currently selected analytic workspace.
DATAPROVIDER	An <code>oracle.olapi.metadata.mdm.MdmMetadataProvider</code> object that is the metadata provider for the session.

**Table 2–7 (Cont.) Keys and Values of the params Map for a ViewerPlugin or EditorPlugin**

Key	Value
DATASOURCE	A <code>java.sql.DataSource</code> object.
ISFOLDER	A <code>String</code> that is <code>TRUE</code> if the <code>&lt;AWMNode&gt;</code> that specifies the plugin-in is a folder or <code>FALSE</code> if it is not.
owner	A <code>String</code> that contains the name of the owner of the currently selected analytic workspace.
schema	A <code>String</code> that contains the name of the owner of the currently selected schema.
TYPE	A <code>String</code> that is the value of the <code>type</code> attribute of the <code>&lt;AWMNode&gt;</code> that specifies the plug-in.
user	A <code>String</code> that contains the name of the user who is connected to the database.
<i>Other bind variables</i>	<p>One or more elements, each of which has a bind variable as a key and has the run-time value of the bind variable as the value. For a plug-in that is specified by the <code>viewClass</code> attribute of an <code>&lt;AWMNode&gt;</code> in an XML document, the number of bind variables depends upon how many bind variables are in the SQL statement of the <code>&lt;AWMNode&gt;</code> and the parent <code>&lt;AWMNode&gt;</code>.</p> <p>Examples of other bind variable keys are <code>dimension_name</code> and <code>cube_name</code>. For examples of other bind variables that can be in the Map see the "Example params Map Elements for a ViewerPlugin and an EditorPlugin" and the examples in Chapter 3, "Examples of Analytic Workspace Manager Plug-ins".</p>

## Example params Map Elements for a ViewerPlugin and an EditorPlugin

Examples of the keys and values of a `params Map` for a `ViewerPlugin` or `EditorPlugin` are in [Table 2–8](#). All of the values are `String` objects except those for the `DATAPROVIDER` and `DATASOURCE` keys.

### Example params Map Elements for a ViewerPlugin

The `cube.xml` document in [Example 3–5](#) on page 3-12 has a parent `<AWMNode>` that has the name `MyMeasures`, a type of `measureobj`, and a SQL statement that references the bind variable `cube_name`. The child `<AWMNode>` has the type `measureview` and has a `viewClass` attribute that specifies the plug-in `MeasureViewerPlugin`.

[Table 2–8](#) has the elements of the `params Map` that Analytic Workspace Manager passes to the methods of the `MeasureViewerPlugin` when the user selects the `UNITS` measure in the `MyMeasures` folder, as shown in [Figure 3–7](#) on page 3-15. The property inspector has the output of the plug-in, which is simply the name of the measure.

The `MeasureViewerPlugin` class in [Example 3–6](#) on page 3-12 gets the value of a bind variable in the following line in the `getPanel` method.

```
measureobj = params.get("measureobj");
```

**Table 2–8 Keys and Values of the params Map for MeasureViewerPlugin**

Key	Value	Description
AW	An AW.	The current analytic workspace object.
aw_name	GLOBAL	The name of the current analytic workspace.

**Table 2–8 (Cont.) Keys and Values of the params Map for MeasureViewerPlugin**

Key	Value	Description
cube_name	UNITS_CUBE	The name of the current cube.
DATAPROVIDER	An MdmMetadataProvider	The metadata provider for the session.
DATASOURCE	A DataSource	The current data source.
ISFOLDER	FALSE	Indicates that the navigation tree object is not a folder.
measureobj	UNITS	The name of the current measure.
NODE_TYPE	MyMeasures	The name of the parent <AWMNode>.
owner	GLOBAL	The name of the owner of the analytic workspace.
schema	GLOBAL	The name of the current schema.
TYPE	measureview	The type of the <AWMNode>.
user	global	The name of the current user.

### Example params Map Elements for an EditorPlugin

Table 2–9 has the elements of the params Map that Analytic Workspace Manager passes to the methods of the DimEditorPlugin when the user selects the CHANNEL dimension in the MyDims folder, as shown in Figure 3–11 on page 3-22. The property inspector in the figure has the output of the DimEditorPlugin.

An example of getting a value from the params Map is the following line from the getMetadataProvider method in the DimEditorPlugin class in Example 3–9 on page 3-18.

```
Object dp = params.get("DATAPROVIDER");
```

Another example of getting a value from the params Map is the following lines from the getDimension method in the DimEditorPlugin class.

```
Object obj = null;
...
obj = params.get("dimobj");
```

**Table 2–9 Keys and Values of the params Map for DimEditorPlugin**

Key	Value	Description
AW	An AW.	The current analytic workspace object.
aw_name	GLOBAL	The name of the current analytic workspace.
DATAPROVIDER	An MdmMetadataProvider	The metadata provider for the session.
DATASOURCE	A DataSource	The current data source.
dimobj	CHANNEL	The run-time value of the dimension currently selected in the MyDims folder.
ISFOLDER	TRUE	Indicates that the navigation tree object is a folder.
NODE_TYPE	MyDims	The name of the parent <AWMNode>.
owner	GLOBAL	The name of the owner of the analytic workspace.

**Table 2–9 (Cont.) Keys and Values of the params Map for DimEditorPlugin**

Key	Value	Description
schema	GLOBAL	The name of the current schema.
TYPE	dimobj	The type of the <AWMNode>.
user	global	The name of the current user.

## Steps in Creating a Plug-in

The prerequisites for creating an Analytic Workspace Manager plug-in are the following:

- For the Analytic Workspace Manager that is part of an Oracle Database Enterprise Edition distribution, include the following files in your development environment. These JAR files are located in the *Oracle\_home/olap/api/lib* directory in the Oracle Database installation.
  - *awm.jar*, which contains the plug-in interfaces.
  - *olap\_api.jar*, which contains the classes in the Oracle OLAP Java API.
  - *awxml.jar*, which contains the *oracle.AWXML.AW* class, which the *AWMPlugin* interface includes for compatibility with the 10g release of Analytic Workspace Manager.
- For an Analytic Workspace Manager that you have downloaded from Oracle Technology Network, include the *awm11.2.0.1.0.jar* file in your development environment.
- Compile the code with JDK 1.5.

---



---

**Note:** Only plug-ins compiled with JDK 1.5 are compatible with Analytic Workspace Manager in 11g Release 2 (11.2).

---



---

To create an Analytic Workspace Manager plug-in, do the following:

1. Create a class that implements a plug-in interface.
  - For an *AWMPlugin*, do the following.
    - In the *isSupported* method, specify the objects in the navigation tree to which the plug-in applies. Be sure to have this method return quickly.
    - Have the *getMenu* method return the text to display on the right-click menu for navigation tree objects that the plug-in supports.
    - In the *handle* method, include the code for the operations that the plug-in performs.
    - Have the *refreshTree* method return a boolean that specifies whether to refresh the navigation tree.
  - For *ViewerPlugin*, do the following.
    - In the *isViewerForType* method, specify the type of navigation tree objects to which the plug-in applies.
    - Have the *getPanel* method create the user interface elements for Analytic Workspace to display and specify the actions for them.

- In the `cleanup` method, perform any cleaning up that your plug-in requires.
- For an `EditorPlugin`, do the steps for a `ViewerPlugin` and add the following.
  - In the `setValueChanged` method, store the `PanelChanged` object from Analytic Workspace Manager. Call the `changed` method of the `PanelChanged` whenever you want to update the display in the property inspector.
  - In the `validate` method, validate any change that the user has made.
  - In the `save` method, perform the actions required to make the changes and then commit the current `Transaction` to save the changes.
  - In the `revert` method, display the object as it was before the changes.
- 2. Using JDK 1.5, compile the plug-in and any other classes that it uses.
- 3. Deploy the plug-in, XML documents, and other classes to a JAR file. You can include multiple plug-ins in the same JAR file.
- 4. Put the JAR file in the plug-ins directory.
- 5. Start Analytic Workspace Manager.

---

---

**Note:** Analytic Workspace Manager only loads the contents of the JAR files upon startup, so if you put a new or updated version of a JAR file in the plug-ins directory, then you must restart Analytic Workspace Manager.

---

---

To use a `ViewerPlugin` or `EditorPlugin`, you generally do the following steps.

- Create an XML document that has the name `schema.xml`, `aw.xml`, `cube.xml`, or `dimension.xml`, depending on where in the navigation tree you want the custom objects to appear. In the XML document, you can have multiple `<AWMNode>` elements at the same level. You can also nest one or more `<AWMNode>` elements in a parent `<AWMNode>` element.
- Develop the SQL statements to specify with `<AWMNode>` elements.
- Implement the `ViewerPlugin` or `EditorPlugin` interface.
- Specify the SQL statement for an `<AWMNode>` with the `sql` attribute. Specify a plug-in with the `viewClass` attribute.
- Deploy the XML document and plug-in implementation in a JAR file. You can have multiple XML documents and plug-ins in the same JAR file. You can put the XML documents in the same JAR file as the plug-ins.
- Put the JAR file in the Analytic Workspace Manager directory for plug-ins.
- Start Analytic Workspace Manager.

## Describing the Available Plug-ins

You can provide information about the plug-ins that you add to Analytic Workspace Manager by creating an `awmplugin.xml` document. In that XML document, you can provide a name, a version number, and a description for each plug-in. Analytic Workspace Manager displays that information, along with the status of the plug-in, when a user selects the **Plugins** tab after selecting **About** on the Help menu.

## Creating an XML Document for Descriptions of Plug-ins

1. Create a text file named `awmplugin.xml`.
2. Begin the file with an XML declaration like this one:  

```
<?xml version="1.0" encoding="utf-8"?>
```

Specify the appropriate encoding for your site.
3. Enter the XML for the plug-in descriptions, as described in ["Reference: Elements for Plug-in Descriptions"](#) on page 2-19.
4. For the name attribute of the `<Plugin>` element, enter a name for the plug-in. For the `version` attribute, enter the version number of the plug-in. For the `class` attribute, enter the class that contains the plug-in. For the `<Description>` element, enter a description of the plug-in.
5. In the plug-directory, create a JAR file that contains the `awmplugin.xml` document. Alternatively, you could add the `awmplugin.xml` document to a JAR file that contains the XML documents described in ["Creating Reports in Object Folders"](#) on page 1-5 or the plug-ins.

For a sample `awmplugin.xml` file, see ["Example of Plug-in Descriptions"](#) on page 3-24.

## Reference: Elements for Plug-in Descriptions

An XML document for describing the available plug-ins has the format shown in [Example 2-1](#).

### *Example 2-1 XML Structure for Descriptions of Plug-ins*

```
<AWMPlugins>
  <Plugin>
    <Description>
      .
      .
      .
    </Description>
  </Plugin>
</AWMPlugins>
```

#### **<AWMPlugins>**

The root element that identifies this document as containing information about the Java plug-ins that are available to Analytic Workspace Manager.

#### **Contains**

`<Plugin>`

#### **Attributes**

None

#### **<Plugin>**

Contains information about a plug-in.

#### **Contains**

`<Description>`

**Attributes**

<Plugin> has the following attributes:

- **name:** A name for the plug-in.
- **version:** A version number for the plug-in.
- **class:** The Java class that implements the plug-in.

**<Description>**

Contains a description of the plug-in.

**Contains**

None

**Attributes**

None

---



---

## Examples of Analytic Workspace Manager Plug-ins

This chapter contains examples of the Java classes that implement the `AWMPlugin`, `ViewerPlugin`, and `EditorPlugin` interfaces. It also contains the example XML documents that specify the plug-ins.

This chapter contains the following topics:

- [Availability of Example Classes and XML Documents](#)
- [Examples of AWMPlugin](#)
- [Examples of ViewerPlugin and EditorPlugin](#)
- [Example of Plug-in Descriptions](#)

### Availability of Example Classes and XML Documents

The examples of Java classes and XML documents in this chapter and in [Chapter 1](#) contain the complete code for the class or document. The complete code is also available in a compressed file that you can download from the Oracle Technology Network (OTN) Web site. The download includes the compiled class files for the plug-ins, as well. The OTN Web site is at

<http://www.oracle.com/technology/products/bi/olap/index.html>

To get the examples, in the Download section of the Web page, select **Sample Code and Schemas**. On the Oracle OLAP Downloads page, in the Sample Schemas and Code section, under Oracle OLAP 11g, in the AWM Java Plug-ins and XML Documents for 11.2.0.2 line, click **examples**.

The compressed file `customizingAWM_examples11202.zip` contains the following files.

Filename	Description
<code>readme.txt</code>	Briefly describes the contents of the zip file.
<code>awmcalcs.xml</code>	Contains the XML for <a href="#">Example 1-9, "Sample AWMCalcs Document"</a> on page 1-13.
<code>awmtree.xml</code>	Contains the XML for <a href="#">Example 1-3, "Passing the Name of a View to a SELECT Statement"</a> on page 1-4.
<code>plugin11202.jar</code>	Contains a directory named <code>plugin11202</code> , which is the package containing the examples. In the directory are the <code>xml</code> , <code>java</code> , and <code>class</code> files for the examples in <a href="#">Chapter 3</a> .

You put the `awmcalcs.xml` and `awmtree.xml` files in the same directory as the Analytic Workspace Manager executable file. You put the `plugin11202.jar` file in the directory that you specify for plug-ins, as described in ["Enabling Analytic Workspace Manager Plug-ins"](#) on page 2-2.

## Examples of AWMPlugin

The examples of an AWMPlugin are in the following topics.

- [ViewXMLPlugin Example](#)
- [DeleteDimPlugin Example](#)

The examples do not include the documentation comments of the methods of the AWMPlugin interface or the input parameters and return values of those methods. Those methods and parameters are described in ["Describing the AWMPlugin Interface"](#) on page 2-5.

### ViewXMLPlugin Example

The `ViewXMLPlugin` class displays an XML representation of a measure or a custom measure of a cube in the Cubes folder in the Analytic Workspace Manager navigation tree. [Example 3-1](#) contains the code for the class. The plug-in applies to `oracle.olap.metadata.mdm.MdmBaseMeasure` and `oracle.olap.metadata.mdm.MdmDerivedMeasure` objects, which correspond to the Measure and Calculated Measure objects, respectively, of a cube.

The plug-in gets and displays an XML representation of a measure. [Figure 2-3](#) on page 2-4 shows the menu that Analytic Manager Workspace displays for `ViewXMLPlugin` when a user right-clicks a measure. For an example of the dialog box that `ViewXMLPlugin` displays, see [Figure 3-1, "Dialog Box Displayed by ViewXMLPlugin"](#) on page 3-5.

#### **Example 3-1 The ViewXMLPlugin Class**

```
import java.awt.BorderLayout;
import java.awt.Font;
import java.awt.Frame;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.sql.Connection;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

import oracle.AWXML.AW;
import oracle.olap.awm.plugin.AWMPlugin;
import oracle.olapi.metadata.mdm.MdmBaseMeasure;
import oracle.olapi.metadata.mdm.MdmDerivedMeasure;
import oracle.olapi.metadata.mdm.MdmMetadataProvider;
import oracle.olapi.metadata.mdm.MdmObject;
```

```

/**
 * An implementation of the AWMPPlugin interface that displays the XML
 * representation of an Oracle OLAP measure object.
 */
public class ViewXMLPlugin implements AWMPPlugin
{
    public boolean isSupported(Connection conn, String type, Object obj,
                              AW aw, Map params)
    {
        // Support MdmBaseMeasure and MdmDerivedMeasure objects.
        if (obj instanceof MdmBaseMeasure || obj instanceof MdmDerivedMeasure)
        {
            return true;
        }
        return false;
    }

    public String getMenu(Connection conn, String type, Object obj, AW aw,
                          Map params)
    {
        // Text to display on the right-click menu.
        String menu = "View XML Example Plug-in";
        return menu;
    }

    public void handle(Frame parent, Connection conn, String type, Object obj,
                       AW aw, Map params)
    {
        if (obj instanceof MdmObject)
        {
            // Get the MdmMetadataProvider to use in exporting the XML.
            Object objdp = params.get("DATAPROVIDER");
            if (objdp != null)
            {
                MdmObject mobj = (MdmObject)obj;
                MdmMetadataProvider mdp = (MdmMetadataProvider)objdp;

                // Get the XML representation of the MdmObject.
                List objects = new ArrayList();
                objects.add(mobj);
                Map renameMap = null;
                boolean includeOwnerString = true;
                String title = "XML for " + mobj.getName();
                try
                {
                    String xml =
                        mdp.exportFullXML(objects, renameMap, includeOwnerString);
                    // Create a dialog box and display the XML.
                    DisplayXMLDialog dxd = new DisplayXMLDialog(parent, title, true,
                                                                xml);
                }
                catch (IOException ie)
                {
                    // Ignore error.
                }
            }
        }
    }
}

```

```
public boolean refreshTree(Connection conn, String type, Object obj, AW aw,
                          Map params)
{
    // This example does not create new metadata objects, so return false.
    return false;
}

/**
 * An inner class that creates a dialog box that displays the XML.
 */
class DisplayXMLDialog extends JDialog implements ActionListener
{
    /**
     * Creates a DisplayXMLDialog for displaying the contents of the xml
     * parameter.
     *
     * @param parent A Frame that is provided by Analytic Workspace Manager.
     * @param title A String that contains text to use as the title for the
     *             dialog box.
     * @param modal A boolean that specifies whether the dialog box is modal.
     * @param xml A String that contains the XML to display.
     */
    public DisplayXMLDialog(Frame parent, String title, boolean modal,
                           String xml)
    {
        super(parent);
        setLocation(200, 200);
        setTitle(title);
        setModal(modal);

        try
        {
            displayXML(xml);
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }

    /**
     * Creates a dialog box and displays the contents of a String.
     *
     * @param xml A String that contains the XML to display.
     */
    private void displayXML(String xml)
    {
        JTextArea ta = new JTextArea(xml);
        ta.setEditable(false);
        Font of = ta.getFont();
        Font f = new Font("Courier New", of.getStyle(), of.getSize());
        ta.setFont(f);

        JScrollPane p = new JScrollPane();
        p.getViewport().add(ta);

        JPanel buttonPane = new JPanel();
        JButton button = new JButton("Close");
        buttonPane.add(button);
        button.addActionListener(this);
    }
}
```

```

        getContentPane().add(buttonPane, BorderLayout.SOUTH);

        getContentPane().add(p, BorderLayout.NORTH);
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        pack();
        setVisible(true);
    }

    /**
     * Performs an action for the Close button.
     *
     * @param e An ActionEvent for the Close button.
     */
    public void actionPerformed(ActionEvent e)
    {
        setVisible(false);
        dispose();
    }
}
}
}

```

Figure 3–1 illustrates the dialog box that ViewXMLPlugin displays for the PROFIT calculated measure in the UNITS\_CUBE folder.

**Figure 3–1** Dialog Box Displayed by ViewXMLPlugin



## DeleteDimPlugin Example

The DeleteDimPlugin class deletes the dimension that the user has selected in the navigation tree. The plug-in only applies to dimension objects that are in a custom folder and that have dimobj as the value of the TYPE key of the params Map. The

DeleteDimPlugin plug-in is specified by the `aw.xml` document in [Example 3-7](#) on page 3-15.

[Example 3-2](#) contains the code for the DeleteDimPlugin class.

### **Example 3-2 The DeleteDimPlugin Class**

```
package plugin11202;

import java.awt.Frame;
import java.sql.Connection;
import java.util.Map;
import javax.swing.JOptionPane;
import oracle.AWXML.AW;
import oracle.olap.awm.plugin.AWMPlugin;
import oracle.olapi.metadata.mdm.MdmMetadataProvider;
import oracle.olapi.metadata.mdm.MdmObject;
import oracle.olapi.metadata.mdm.MdmPrimaryDimension;
import oracle.olapi.metadata.mdm.MdmSchema;

/**
 * An implementation of the AWMPlugin interface that can delete
 * an Oracle OLAP dimension object in a custom folder.
 */
public class DeleteDimPlugin implements AWMPlugin
{
    // This plug-in applies to dimension objects in a custom folder.
    public boolean isSupported(Connection conn, String type, Object obj, AW aw,
                               Map params)
    {
        if (params != null)
        {
            // Get the value of the type attribute of the AWMNode that specifies this
            // plug-in.
            Object nodeType = params.get("TYPE");
            if (nodeType != null && ((String)nodeType).equalsIgnoreCase("dimobj"))
                return true;
        }
        return false;
    }

    public String getMenu(Connection conn, String type, Object obj, AW aw,
                          Map params)
    {
        Object dimName = null;
        if (obj != null && obj instanceof String)
        {
            dimName = (String) obj;
        }
        // Text to display on the right-click menu.
        return "Example Plug-in: Delete Dimension " + dimName;
    }

    public void handle(Frame parent, Connection conn, String type, Object obj,
                       AW aw, Map params)
    {
        String dimName = "";
        // The obj parameter should be the name of the currently selected dimension.
        if (obj != null && obj instanceof String)
        {
            dimName = (String) obj;
        }
    }
}
```

```

String title = "Delete Dimension";
if (JOptionPane.showConfirmDialog(parent, "Delete " + dimName + "?",
                                title, JOptionPane.YES_NO_OPTION) ==
    JOptionPane.NO_OPTION)
    return;
}

if (params != null)
{
    Map bindMap = (Map)params.get("BIND_MAP");
    if (bindMap != null)
    {
        // Get the name of the owner, which is also the name of the schema.
        String owner = (String)bindMap.get("owner");
        // Get the currently selected dimension.
        MdmPrimaryDimension dim = getDimension(dimName, owner, params);

        if (dim != null)
        {
            // Get the schema object that contains the dimension.
            MdmSchema schema = dim.getOwner();
            schema.removeDimension(dim);
            MdmMetadataProvider mdp = getMetadataProvider(params);
            // Get the TransactionProvider and commit the current Transaction.
            try
            {
                mdp.getDataProvider()
                    .getTransactionProvider()
                    .commitCurrentTransaction();
                JOptionPane.showMessageDialog(parent,
                                           owner + "." + dimName +
                                           " dimension has been deleted.");
            }
            catch (Exception e)
            {
                JOptionPane.showMessageDialog(parent, e.getMessage(), "Error",
                                           JOptionPane.ERROR_MESSAGE);
                // Roll back the current Transaction.
                try
                {
                    mdp.getDataProvider()
                        .getTransactionProvider()
                        .rollbackCurrentTransaction();
                }
                catch (Exception e2)
                {
                    // Ignore the exception.
                }
            }
        }
    }
}
else
{
    return;
}
}
}

```

```

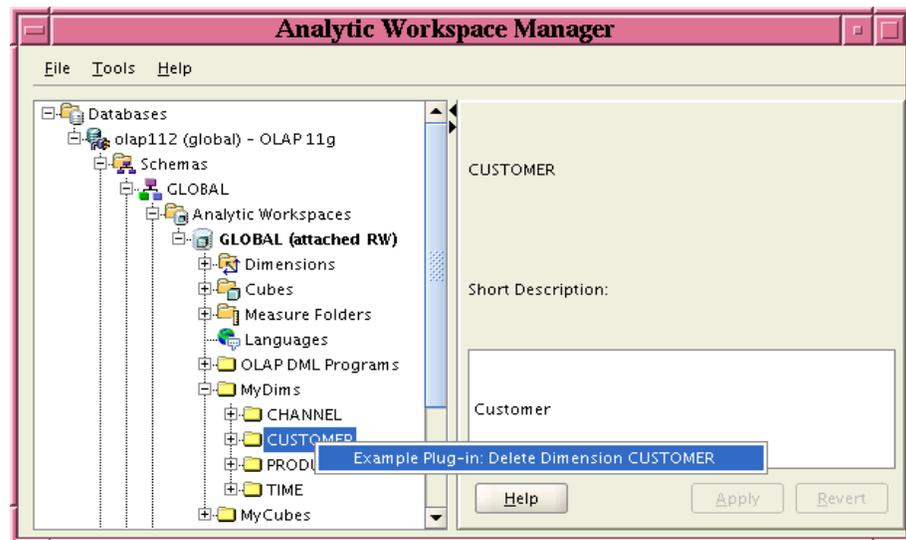
public boolean refreshTree(Connection conn, String type, Object obj, AW aw,
                          Map params)
{
    return true;
}

// Get the MdmMetadataProvider.
private MdmMetadataProvider getMetadataProvider(Map params)
{
    Object dp = params.get("DATAPROVIDER");
    if (dp instanceof MdmMetadataProvider)
    {
        MdmMetadataProvider mdp = (MdmMetadataProvider)dp;
        return mdp;
    }
    return null;
}

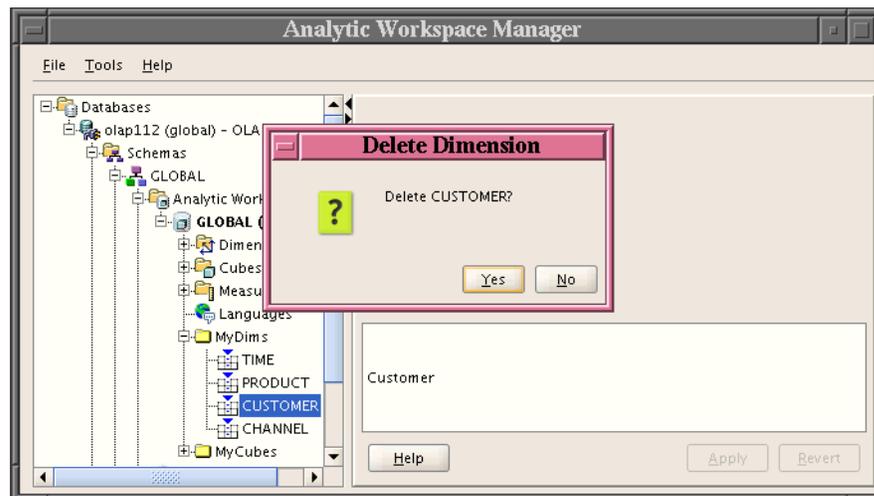
// Get the currently selected dimension.
private MdmPrimaryDimension getDimension(String dimName, String schema,
                                         Map params)
{
    if (params != null)
    {
        MdmMetadataProvider mdp = getMetadataProvider(params);
        if (mdp != null)
        {
            // Get the dimension from the MdmMetadataProvider.
            MdmObject mobj = mdp.getMetadataObject(schema + "." + dimName);
            if (mobj != null && mobj instanceof MdmPrimaryDimension)
            {
                MdmPrimaryDimension dim = (MdmPrimaryDimension)mobj;
                return dim;
            }
        }
    }
    return null;
}
}

```

**Figure 3–2** shows the menu that Analytic Manager Workspace displays for `DeleteDimPlugin`. The figure shows the menu that appears when a user right-clicks the `CUSTOMER` dimension in the `MyDims` folder. The `MyDims` folder is created by the `aw.xml` document in [Example 3–7](#) on page 3-15.

**Figure 3–2** Right-click Menu Displayed by DeleteDimPlugin

If the user clicks **Example Plug-in: Delete Dimension CUSTOMER**, then DeleteDimPlugin displays the dialog box shown in [Figure 3–3](#).

**Figure 3–3** Dialog Box Displayed by DeleteDimPlugin

## Examples of ViewerPlugin and EditorPlugin

The example ViewerPlugin and EditorPlugin implementations are in the following topics:

- [LevelViewerPlugin Example](#)
- [MeasureViewerPlugin Example](#)
- [CubeViewerPlugin Example](#)
- [DimEditorPlugin Example](#)

The topics include the XML documents that specify the plug-ins.

The methods of the `ViewerPlugin` and `EditorPlugin` interfaces are described in ["Describing the ViewerPlugin and EditorPlugin Interfaces"](#) on page 2-11.

## LevelViewerPlugin Example

The `dimension.xml` document in [Example 3-3](#) has an `<AWMNode>` that specifies a folder named `MyLevels` and a SQL statement that selects the names of the levels of the currently selected dimension from the `USER_CUBE_DIM_LEVELS` table. An unnamed child `<AWMNode>` specifies the `LevelViewerPlugin`. [Figure 3-4](#) on page 3-11 shows the navigation tree folder and the display in the property inspector for the document.

### Example 3-3 Creating a dimension.xml Document

```
<?xml version="1.0" encoding="US-ASCII" ?>
<AWMTree>
  <AWMNode name="MyLevels"
    type="levelobj"
    sql="select level_name from user_cube_dim_levels where dimension_name =
{dimension_name} ">
    <AWMNode type="levelview"
      viewClass="plugin11202.LevelViewerPlugin"/>
  </AWMNode>
</AWMTree>
```

[Example 3-4](#) contains the `LevelViewerPlugin` class. The class displays the name of the currently selected level, as shown in [Figure 3-5](#) on page 3-12.

### Example 3-4 The LevelViewerPlugin Class

```
package plugin11202;

import java.awt.FlowLayout;
import java.sql.Connection;
import java.util.Map;
import javax.swing.JLabel;
import javax.swing.JPanel;
import oracle.olap.awm.plugin.ViewerPlugin;

public class LevelViewerPlugin implements ViewerPlugin
{
    public boolean isViewerForType(Connection conn, String name)
        throws Exception
    {
        return true;
    }

    public JPanel getPanel(Connection conn, String name, Map params)
        throws Exception
    {
        JPanel panel = new JPanel();
        panel.setLayout(new FlowLayout());
        // Get the name of the current level.
        Object obj = params.get("levelobj");
        if (obj instanceof String)
        {
            String levelName = (String)obj;
            panel.add(new JLabel(levelName));
        }
        return panel;
    }
}
```

```

public void cleanup(String name)
{
}
}

```

Figure 3–4 shows the results of the `MyLevels` `<AWMNode>` in the `dimension.xml` document. A `MyLevels` folder appears in each dimension folder of the analytic workspace. The user has selected the `MyLevels` folder in the `PRODUCT` folder. The result of the SQL statement of the `<AWMNode>` appears in the `MyLevels` folder. The property inspector displays the same SQL statement and the result of it, which is a list of the levels of the dimension.

**Figure 3–4 Results of the `MyLevels` `<AWMNode>` in `dimension.xml`**

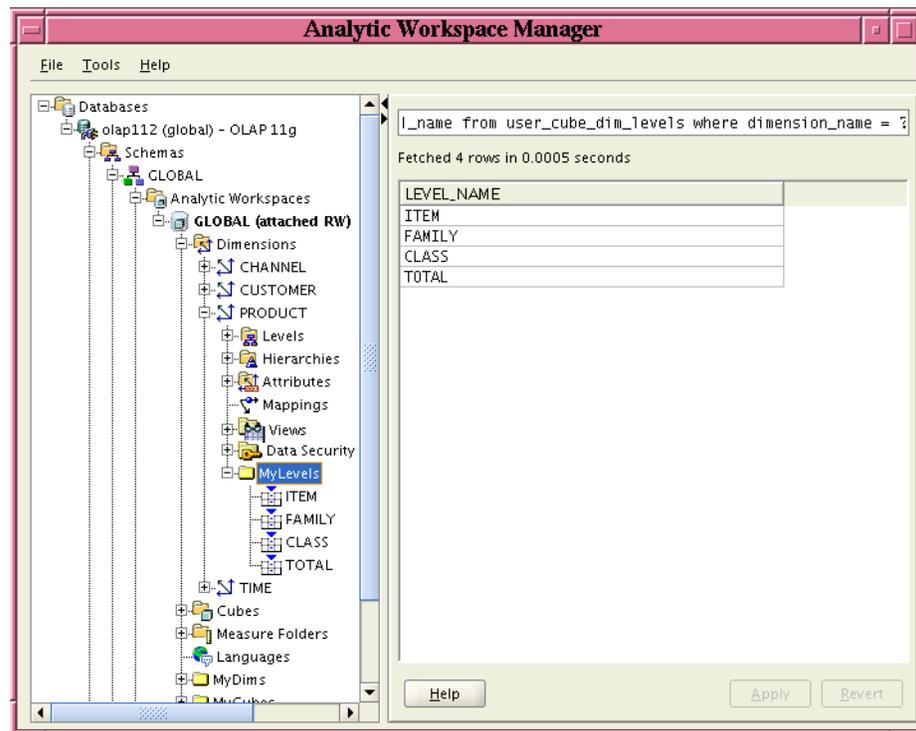
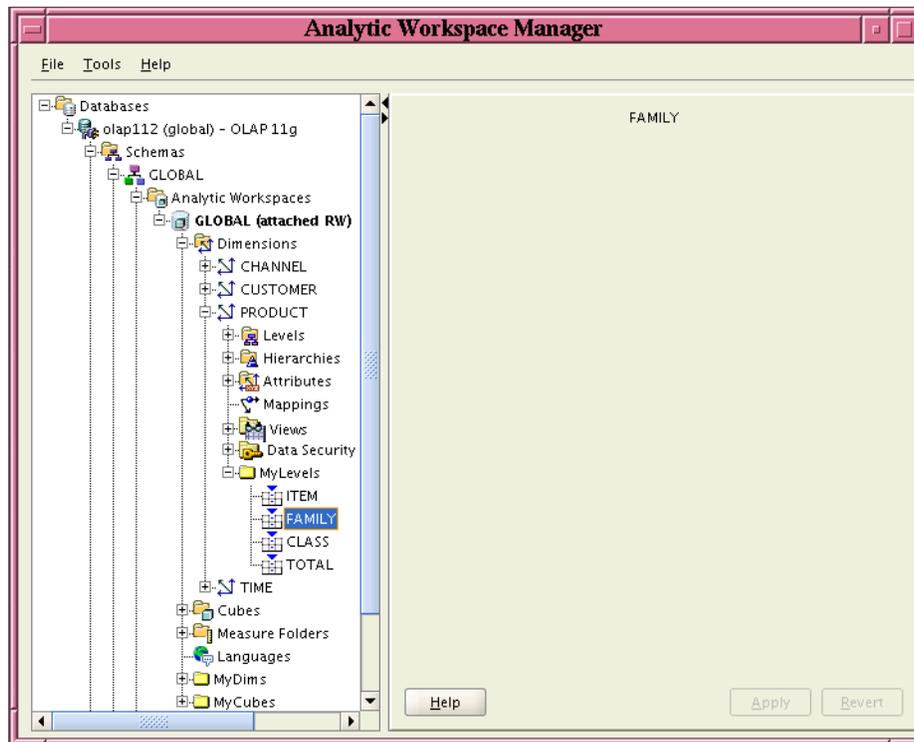


Figure 3–5 shows the results of the unnamed child `<AWMNode>` of the `MyLevels` `<AWMNode>` in the `dimension.xml` document. The user has selected the `FAMILY` level in the `MyLevels` folder. The property inspector displays the user interface specified by `LevelViewerPlugin`. The plug-in displays the name of the level.

**Figure 3–5 Results of LevelViewerPlugin**

## MeasureViewerPlugin Example

The `cube.xml` document in [Example 3–5](#) has an `<AWMNode>` that specifies a folder named `MyMeasures` and a SQL statement that selects the names of the measures of the currently selected cube from the `USER_CUBE_MEASURES` table. An unnamed child `<AWMNode>` specifies the `MeasureViewerPlugin` plug-in. [Figure 3–6](#) on page 3-14 shows the navigation tree folder and the display in the property inspector for the document.

### Example 3–5 Creating a `cube.xml` Document

```
<?xml version="1.0" encoding="US-ASCII" ?>
<AWMTree>
  <AWMNode name="MyMeasures"
    type="measureobj"
    sql="select measure_name from user_cube_measures where cube_name =
{cube_name} ">
    <AWMNode type="measurereview"
      viewClass="plugin11202.MeasureViewerPlugin"/>
  </AWMNode>
</AWMTree>
```

[Example 3–6](#) contains the `MeasureViewerPlugin` class. The class displays the name of the currently selected measure, as shown in [Figure 3–7](#) on page 3-15.

### Example 3–6 The `MeasureViewerPlugin` Class

```
package plugin11202;

import java.awt.FlowLayout;
import java.sql.Connection;
```

```

import java.util.Map;
import javax.swing.JLabel;
import javax.swing.JPanel;
import oracle.olap.awm.plugin.ViewerPlugin;

public class MeasureViewerPlugin implements ViewerPlugin
{
    public boolean isViewerForType(Connection conn, String name)
        throws Exception
    {
        return true;
    }

    public JPanel getPanel(Connection conn, String name, Map params)
        throws Exception
    {
        JPanel panel = new JPanel();
        panel.setLayout(new FlowLayout());

        // Get the name of the current measure.
        Object measureobj = null;
        if (params != null)
            measureobj = params.get("measureobj");

        if (measureobj instanceof String)
        {
            String measureName = (String)measureobj;
            panel.add(new JLabel(measureName));
        }
        return panel;
    }

    public void cleanup(String name)
    {
    }
}

```

Figure 3–6 shows the results of the `MyMeasures` `<AWMNode>` in the `cube.xml` document. A `MyMeasures` folder appears in each cube folder of the analytic workspace. The user has selected the `MyMeasures` folder in the `UNITS_CUBE` folder. The result of the SQL statement of the `<AWMNode>` appears in the `MyMeasures` folder. The property inspector displays the same SQL statement and the result of it, which is a list of the measures and calculated measures of the cube.

Figure 3–6 Results of the MyMeasures <AWMNode> in cube.xml

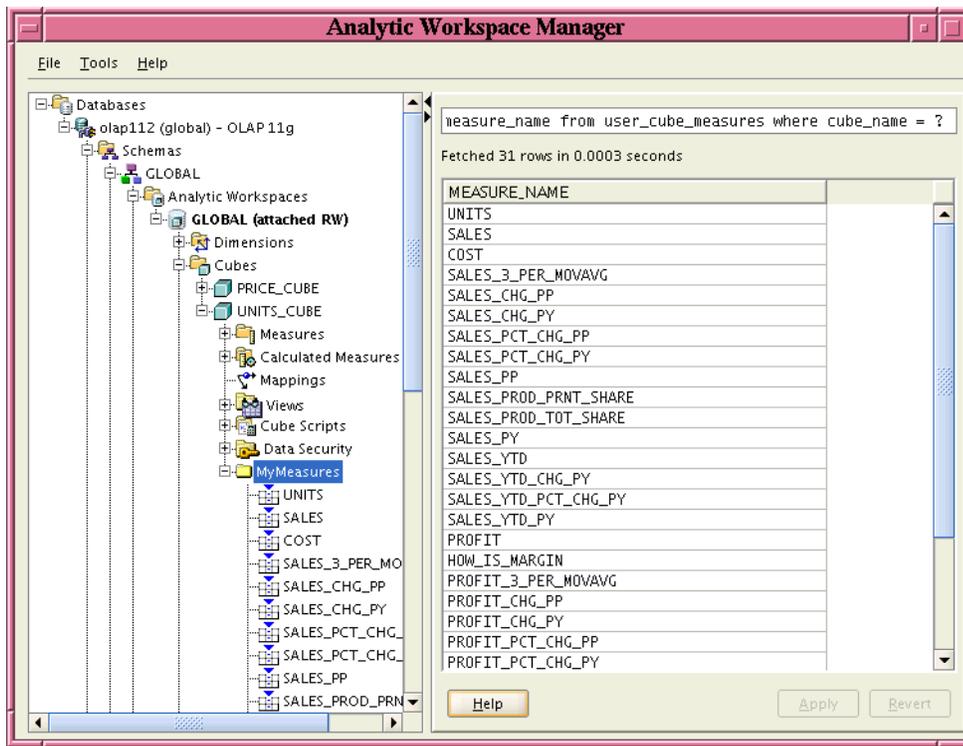
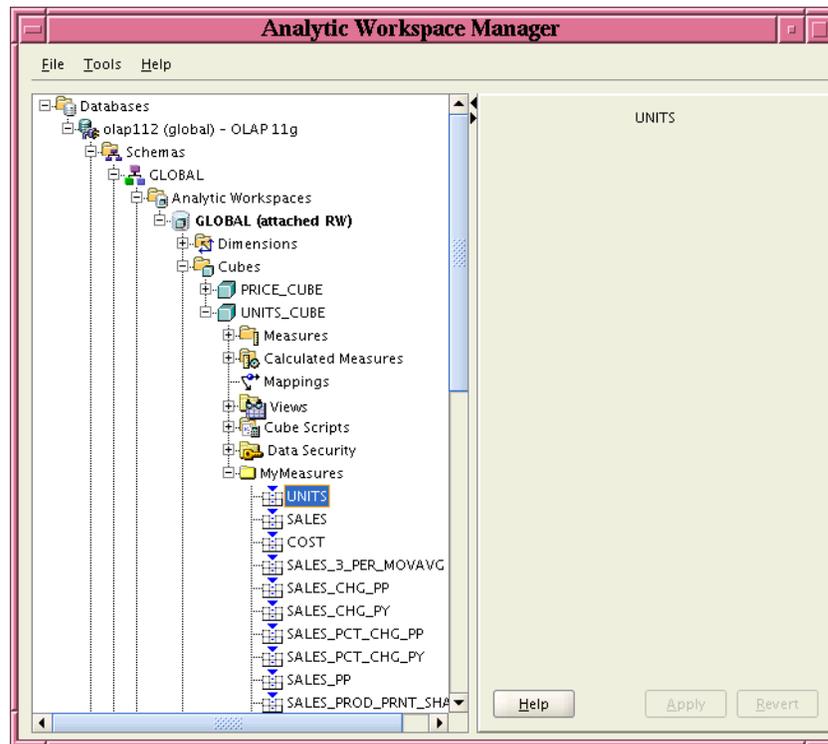


Figure 3–7 shows the results of the unnamed child <AWMNode> of the MyMeasures <AWMNode> in the cube.xml document. The user has selected the UNITS measure in the MyMeasures folder. The property inspector displays the user interface specified by MeasureViewerPlugin. The plug-in displays the name of the measure.

Figure 3-7 Results of MeasureViewerPlugin



## CubeViewerPlugin Example

The `aw.xml` document in [Example 3-7](#) has an `<AWMNode>` that specifies a folder named `MyDims`. For a description of the `MyDims <AWMNode>`, see ["DimEditorPlugin Example"](#) on page 3-17.

The `aw.xml` document also has an `<AWMNode>` that specifies a folder named `MyCubes` and a SQL statement that selects the names of the cubes of the current analytic workspace from the `USER_CUBES` table. An unnamed child `<AWMNode>` specifies the `CubeViewerPlugin`. [Figure 3-8](#) on page 3-17 shows the navigation tree folder and the display in the property inspector for the `MyCubes <AWMNode>`.

### Example 3-7 Creating an `aw.xml` Document

```
<?xml version="1.0" encoding="US-ASCII" ?>
<AWMTree>
  <AWMNode name="MyDims"
    type="mydimfolder"
    viewSql="select dimension_name, dimension_type from
user_cube_dimensions where aw_name = {aw_name}">
    <AWMNode type="dimobj"
      sql="select dimension_name from user_cube_dimensions where aw_name =
{aw_name}"
      viewClass="plugin11202.DimEditorPlugin">
    </AWMNode>
    <AWMNode name="MyLevels"
      type="levelobj"
      sql="select level_name from user_cube_dim_levels where
dimension_name = {dimobj}">
      <AWMNode sql="select * from user_cube_dim_levels where dimension_name =
{dimobj} and level_name = {levelobj}"/>
    </AWMNode>
  </AWMNode>
</AWMTree>
```

```

        </AWMNode>
    </AWMNode>
    <AWMNode name="MyCubes"
        type="cubeobj"
        sql="select cube_name from user_cubes where aw_name = {aw_name}">
        <AWMNode type="mycubeview"
            viewClass="plugin11202.CubeViewerPlugin"/>
    </AWMNode>
</AWMTree>

```

**Example 3-8** contains the `CubeViewerPlugin` class. The class displays the name of the currently selected cube, as shown in [Figure 3-9](#) on page 3-17.

### **Example 3-8 The CubeViewerPlugin Class**

```

package plugin11202;

import java.awt.FlowLayout;
import java.sql.Connection;
import java.util.Map;
import javax.swing.JLabel;
import javax.swing.JPanel;
import oracle.olap.awm.plugin.ViewerPlugin;

public class CubeViewerPlugin implements ViewerPlugin
{
    public boolean isViewerForType(Connection conn, String name)
        throws Exception
    {
        return true;
    }

    public JPanel getPanel(Connection conn, String name, Map params)
        throws Exception
    {
        JPanel panel = new JPanel();
        panel.setLayout(new FlowLayout());
        // Get the name of the current cube.
        Object cubeobj = null;
        if (params != null)
            cubeobj = params.get("cubeobj");
        if (cubeobj instanceof String)
        {
            String cubeName = (String)cubeobj;
            panel.add(new JLabel(cubeName));
        }
        return panel;
    }

    public void cleanup(String name)
    {
    }
}

```

**Figure 3-8** shows the results of the `MyCubes` `<AWMNode>` in the `aw.xml` document. A `MyCubes` folder appears in the GLOBAL analytic workspace folder. The user has selected the `MyCubes` folder. The result of the SQL statement of the `<AWMNode>` appears in the folder. The property inspector displays the same SQL statement and the result of it, which is a list of the cubes of the analytic workspace.

Figure 3-8 Results of the MyCubes <AWMNode> in aw.xml

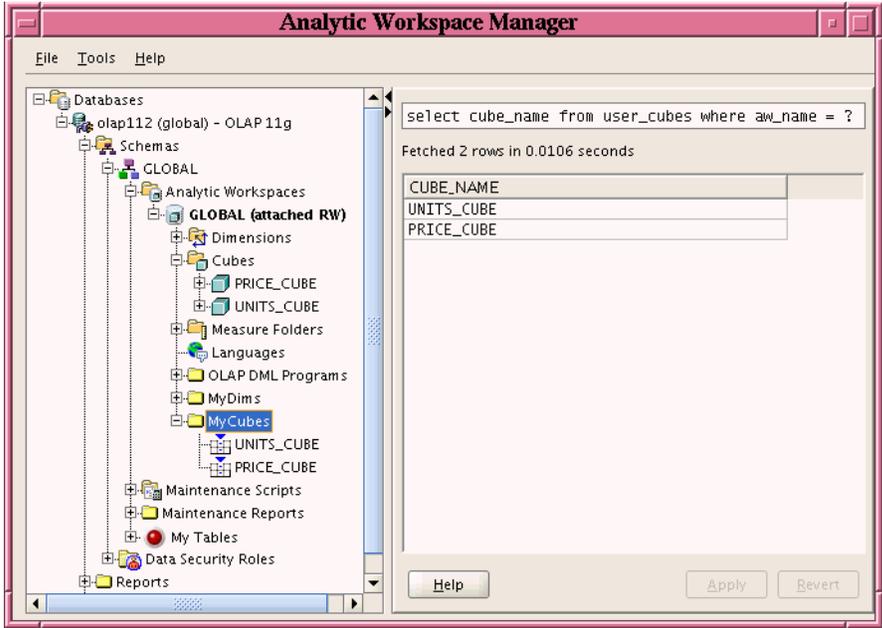
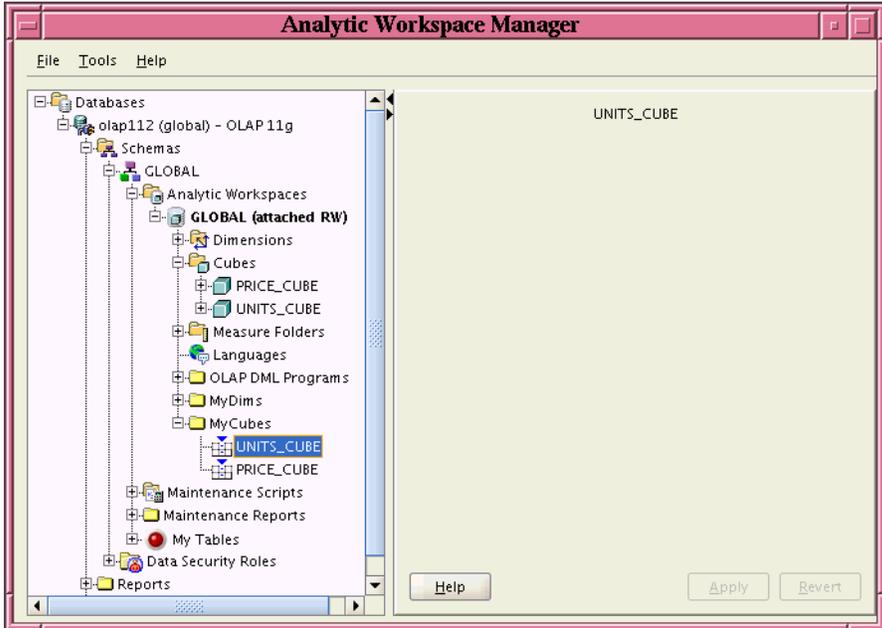


Figure 3-9 shows the results of the unnamed child <AWMNode> of the MyCubes <AWMNode> in the aw.xml document. The user has selected the UNITS\_CUBE cube in the MyCubes folder. The property inspector displays the user interface specified by CubeViewerPlugin. The plug-in displays the name of the cube.

Figure 3-9 Results of the CubeViewerPlugin



### DimEditorPlugin Example

The aw.xml document in Example 3-7 on page 3-15 has an <AWMNode> that specifies a folder named MyDims and a SQL statement that selects the names and types of the dimensions of the current analytic workspace from the USER\_CUBE\_DIMENSIONS

table. [Figure 3–10](#) on page 3-22 shows the navigation tree folder and the display in the property inspector for the MyDims <AWMNode>.

An unnamed child <AWMNode> specifies a SQL statement that selects the names of the dimensions and also specifies the DimEditorPlugin. [Figure 3–11](#) on page 3-22 shows the navigation tree folder and the display in the property inspector for the MyDims <AWMNode>.

The <AWMNode> named MyLevels, nested in the unnamed <AWMNode>, selects the names of the levels from the USER\_CUBE\_DIM\_LEVELS table for the currently selected dimension. The MyLevels <AWMNode> has an unnamed nested <AWMNode> that selects all columns from the USER\_CUBE\_DIM\_LEVELS table for the currently selected dimension and level.

[Example 3–9](#) contains the DimEditorPlugin class. The class displays the name and the short description of the currently selected dimension, as shown in [Figure 3–11](#) on page 3-22. The user can change the value of the short description.

### **Example 3–9 The DimEditorPlugin Class**

```
package plugin11202;

import java.awt.Component;
import java.awt.GridLayout;
import java.sql.Connection;
import java.util.Map;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import oracle.olap.awm.plugin.EditorPlugin;
import oracle.olap.awm.plugin.PanelChanged;
import oracle.olapi.metadata.mdm.MdmDescriptionType;
import oracle.olapi.metadata.mdm.MdmMetadataProvider;
import oracle.olapi.metadata.mdm.MdmObject;
import oracle.olapi.metadata.mdm.MdmPrimaryDimension;

public class DimEditorPlugin implements EditorPlugin
{
    private JTextField shortDescTextField;
    private PanelChanged parentPanelChanged;
    private JPanel panel;
    private JLabel dimNameLabel;
    private MdmDescriptionType mdmShortDescrDescrType;

    public DimEditorPlugin()
    {
        panel = new JPanel();
        panel.setLayout(new GridLayout(3, 1));
        dimNameLabel = new JLabel();
        panel.add(dimNameLabel);
        shortDescTextField = new JTextField();
        panel.add(new JLabel("Short Description:"));
        panel.add(shortDescTextField);
        shortDescTextField.getDocument().addDocumentListener(new DocumentListener()
        {
            public void insertUpdate(DocumentEvent e)
            {
                changed();
            }
        });
    }
}
```

```

    }

    public void removeUpdate(DocumentEvent e)
    {
        changed();
    }

    public void changedUpdate(DocumentEvent e)
    {
        changed();
    }
});
}

public boolean isViewerForType(Connection conn, String name)
    throws Exception
{
    return true;
}

// Get the MdmMetadataProvider of the session.
private MdmMetadataProvider getMetadataProvider(Map params)
{
    Object dp = params.get("DATAPROVIDER");
    if (dp instanceof MdmMetadataProvider)
    {
        MdmMetadataProvider mdp = (MdmMetadataProvider)dp;
        return mdp;
    }
    return null;
}

// Get the currently selected dimension and the schema from the params Map.
// Get the MdmMetadataProvider and get the MdmPrimaryDimension for the
// dimension.
private MdmPrimaryDimension getDimension(Map params)
{
    Object obj = null;
    String schema = "";
    if (params != null)
    {
        obj = params.get("dimobj");
        schema = (String)params.get("schema");
    }
    if (obj instanceof String)
    {
        String dimName = (String)obj;
        MdmMetadataProvider mdp = getMetadataProvider(params);
        if (mdp != null)
        {
            MdmObject mobj = mdp.getMetadataObject(schema + "." + dimName);
            if (mobj != null && mobj instanceof MdmPrimaryDimension)
            {
                MdmPrimaryDimension dim = (MdmPrimaryDimension)mobj;
                return dim;
            }
        }
        else
            System.out.println("Cannot get the " + dimName + " dimension.");
    }
}
}

```

```
        return null;
    }

    // Get the dimension and the short description of it.
    // Display the short description.
    private void read(Map params)
    {
        MdmPrimaryDimension dim = getDimension(params);
        if (dim != null)
        {
            dimNameLabel.setText(dim.getName());
            mdmShortDescrDescrType =
                MdmDescriptionType.getShortDescriptionDescriptionType();
            String shortDesc = dim.getDescription(mdmShortDescrDescrType);
            shortDescTextField.setText(shortDesc);
        }
    }

    public JPanel getPanel(Connection conn, String name, Map params)
        throws Exception
    {
        read(params);
        return panel;
    }

    public void cleanup(String name)
    {
    }

    public boolean validate(Connection conn, Component parent, String name,
        Map params)
        throws Exception
    {
        String invalidDescr = "foo";
        if (shortDescTextField.getText().equals(invalidDescr))
        {
            JOptionPane.showMessageDialog(parent, "Description cannot be " +
                invalidDescr + ".");
            return false;
        }
        return true;
    }

    public void revert(Connection conn, Component parent, String name,
        Map params)
        throws Exception
    {
        read(params);
    }

    public void showHelp(Connection conn, Component parent, String name,
        Map params)
        throws Exception
    {
        JOptionPane.showMessageDialog(parent, "In Help.");
    }

    public boolean save(Connection conn, Component parent, String name,
        Map params)
        throws Exception
```

```

{
    // Get the currently selected dimension and set the short description for
    // it.
    MdmPrimaryDimension dim = getDimension(params);
    dim.setDescription(mdmShortDescrDescrType, shortDescTextField.getText());
    // Get the MdmMetadataProvider.
    MdmMetadataProvider mdp = getMetadataProvider(params);
    if (mdp == null)
        return false;
    // Get the DataProvider and the TransactionProvider and commit the current
    // Transaction. If the Transaction is not committable, roll it back.
    try
    {
        mdp.getDataProvider().getTransactionProvider().commitCurrentTransaction();
    }
    catch (Exception e)
    {
        JOptionPane.showMessageDialog(parent, e.getMessage(), "Error",
            JOptionPane.ERROR_MESSAGE);

        try
        {
            mdp.getDataProvider()
                .getTransactionProvider()
                .rollbackCurrentTransaction();
        }
        catch (Exception e2)
        {
            // Ignore the exception.
        }
    }
    return true;
}

public void setValueChanged(Connection conn, String name, Map params,
    PanelChanged parentPanelChanged)
{
    this.parentPanelChanged = parentPanelChanged;
}

// Calls the changed() method of the PanelChanged object supplied by
// Analytic Workspace Manager when it calls the setValueChanged method.
public void changed()
{
    if (parentPanelChanged != null)
        parentPanelChanged.changed();
}
}

```

Figure 3–10 shows the results of the MyDims <AWMNode> in the aw.xml document. A MyDims folder appears in the GLOBAL analytic workspace folder. The user has selected the MyDims folder. The property inspector displays the SQL statement of the MyDims <AWMNode> and the result of it, which is a table that has columns headed DIMENSION\_NAME and DIMENSION\_TYPE. The rows of the columns contains the names of the dimensions of the analytic workspace and the types of the dimensions.

The MyDims <AWMNode> has an unnamed child <AWMNode> that has a SQL statement that retrieves the names of the dimensions. Those names appear in the MyDims folder in the navigation tree. The unnamed <AWMNode> also specifies the DimEditorPlugin plug-in.

**Figure 3–10 Results of the MyDims <AWMNode> in aw.xml**

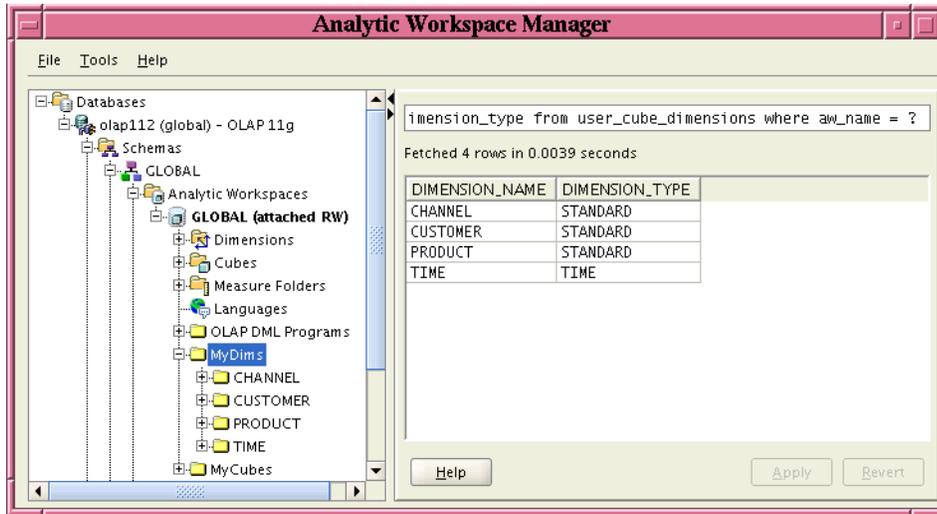


Figure 3–11 shows the Analytic Workspace Manager user interface after a user has selected the CHANNEL dimension in the MyDims folder in the navigation tree. The property inspector displays the user interface specified by DimEditorPlugin. The user interface includes a text field in which the user can change the value of the short description attribute.

**Figure 3–11 Results of DimEditorPlugin**

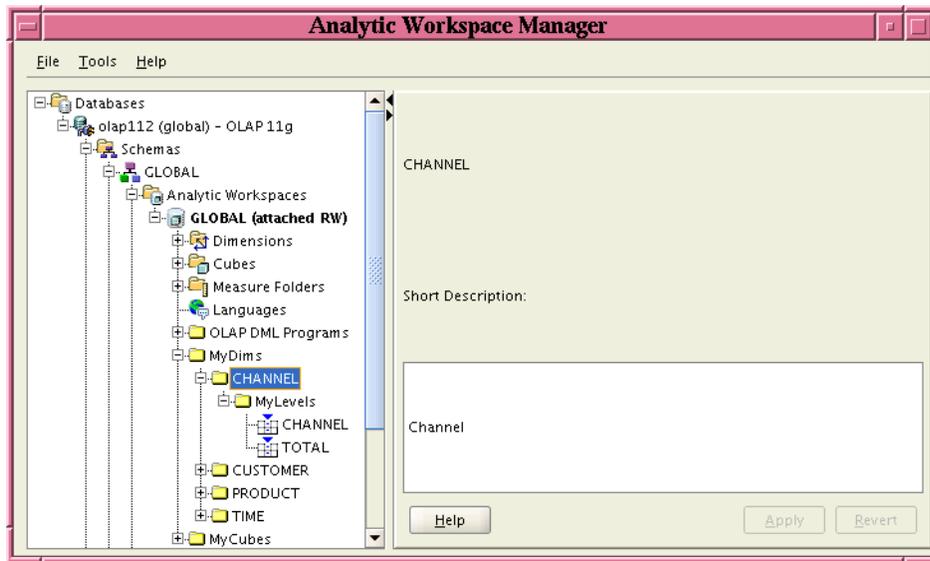


Figure 3–12 shows the result of the MyLevels <AWMNode> that is the child of the MyDims <AWMNode> in the aw.xml document. The SQL statement of the MyLevels <AWMNode> selects the LEVEL\_NAME column from the USER\_CUBE\_DIM\_LEVELS table for the currently selected dimension. Figure 3–12 shows the navigation tree folder with the MyLevels folder selected in the CHANNEL folder. In the property inspector is the result of the query.

Figure 3-12 Result of MyLevels <AWMNode> Under MyDims in aw.xml

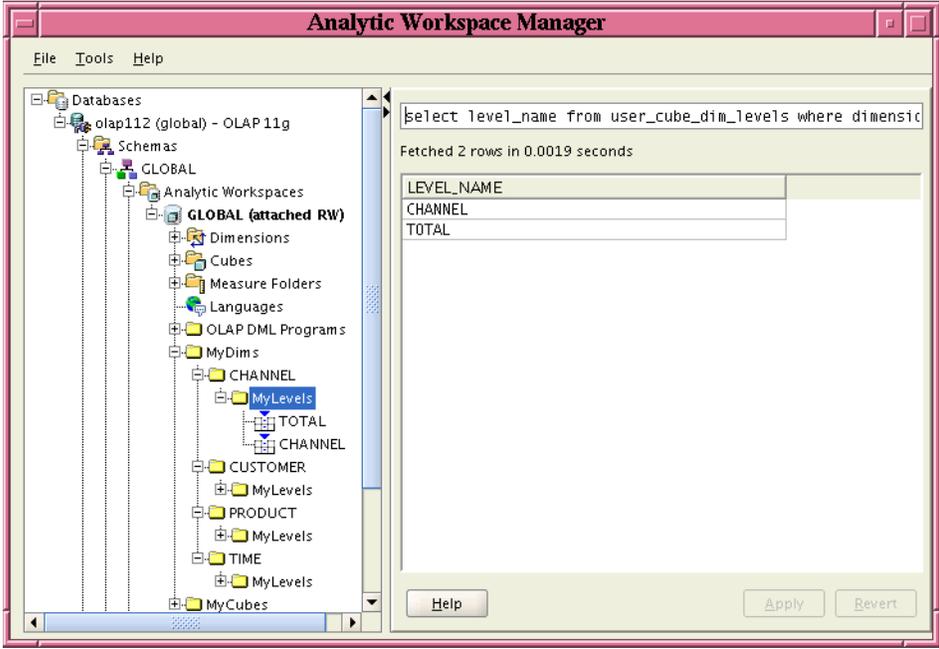
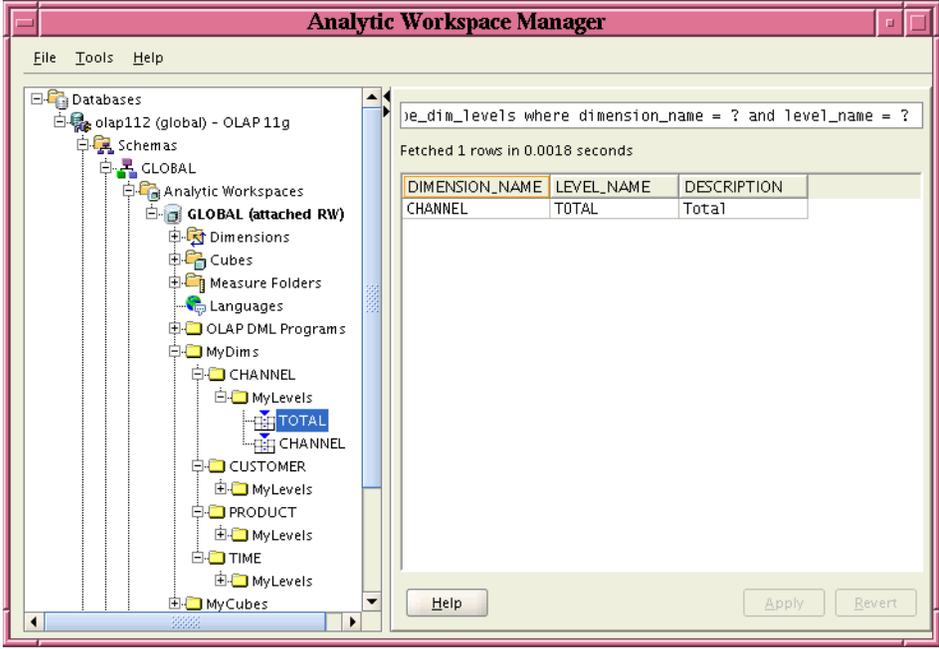


Figure 3-13 shows the result of the unnamed <AWMNode> that is the child of the MyLevels <AWMNode> in the aw.xml document. The SQL statement of the unnamed <AWMNode> selects all columns from the USER\_CUBE\_DIM\_LEVELS table for the currently selected dimension and level. Figure 3-13 shows the navigation tree folder with the TOTAL level selected in the MyLevels folder in the CHANNEL folder. The property inspector displays the result of the query.

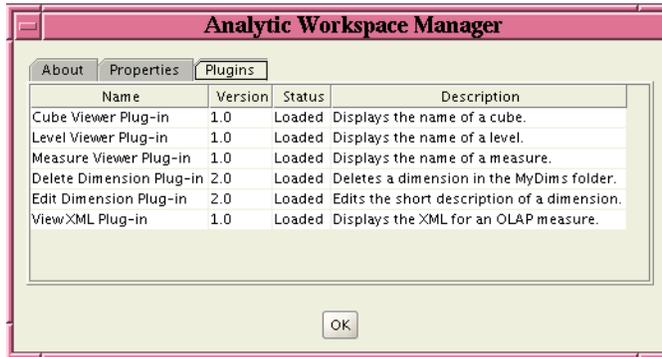
Figure 3-13 Results of the Nested <AWMNode> in the MyLevels <AWMNode> in aw.xml



## Example of Plug-in Descriptions

As discussed in "Describing the Available Plug-ins" on page 2-18, the `awmpugin.xml` file contains descriptions of Java plug-ins that Analytic Workspace Manager displays. Figure 3-14 shows the Plugins tab of the About dialog box with the information that is specified by the `awmpugin.xml` document in Example 3-10.

**Figure 3-14** Plugins Tab in the About Dialog Box



Example 3-10 shows the `awmpugin.xml` document that produces the result shown in Figure 3-14.

### Example 3-10 Creating an `awmpugins.xml` Document

```
<?xml version="1.0" encoding="utf-8" ?>
<AWMPlugins>
  <Plugin name="Cube Viewer Plug-in" version="1.0"
    class="plugin11202.CubeViewerPlugin">
    <Description>Displays the name of a cube.</Description>
  </Plugin>
  <Plugin name="Level Viewer Plug-in" version="1.0"
    class="plugin11202.LevelViewerPlugin">
    <Description>Displays the name of a level.</Description>
  </Plugin>
  <Plugin name="Measure Viewer Plug-in" version="1.0"
    class="plugin11202.MeasureViewerPlugin">
    <Description>Displays the name of a measure.</Description>
  </Plugin>
  <Plugin name="Delete Dimension Plug-in" version="2.0"
    class="plugin11202.DeleteDimPlugin">
    <Description>Deletes a dimension in the MyDims folder.</Description>
  </Plugin>
  <Plugin name="Edit Dimension Plug-in" version="2.0"
    class="plugin11202.DimEditorPlugin">
    <Description>Edits the short description of a dimension.</Description>
  </Plugin>
  <Plugin name="View XML Plug-in" version="1.0" class="plugin11202.ViewXMLPlugin">
    <Description>Displays the XML for an OLAP measure.</Description>
  </Plugin>
</AWMPlugins>
```

## Symbols

---

- \$
  - enclosing expression, 1-17
- { }
  - enclosing bind variable, 1-4, 1-15
  - enclosing hypertext parameter, 1-15

## A

---

- Analytic Workspace Manager
  - configuring for plug-ins, 2-2
  - ways of customizing, ix
- aw parameter
  - of AWMPPlugin methods, 2-6
- AWMCalcs element
  - root of a calculation template, 1-15
- awmcalcs.xml document
  - example of, 1-9, 1-11, 1-13
  - in downloadable examples, 3-1
  - location in database directory, 1-8
- AWMNode element
  - of an AWMTTree element, 1-7
- AWMPPlugin interface
  - examples of, 3-2, 3-5
  - implementations loaded on startup, 2-3
  - specification, 2-5
- AWMPPlugins element
  - root of a plug-in description, 2-19
- awmpplugin.xml document
  - description, 2-18
  - example of, 3-24
- AWMTTree element
  - root of a SQL report, 1-7
- AWMTTree prefix
  - for type parameter values, 2-7
- awmtree.xml document
  - example of, 1-2, 1-3, 1-4
  - in downloadable examples, 3-1
  - location in directory, 1-1
  - location in navigation tree, 1-5
- aw.xml document
  - example of, 3-15
  - location in navigation tree, 1-5

## B

---

- bind variables
  - enclosed by braces, 1-4, 1-15
  - for an AWMPPlugin, 2-10
  - in calculation templates, 1-10, 1-15
  - in custom reports, 1-3
  - referencing using lower case, 2-10
  - replacing values in a WHERE clause, 1-1
  - replacing values in a WHERE clause, example of, 1-4
- BIND\_MAP Map
  - for an AWMPPlugin, 2-10
  - keys and values, examples of, 2-11
- braces
  - enclosing bind variable, 1-4, 1-15
  - enclosing hypertext parameters, 1-15

## C

---

- Calc element
  - of an AWMCalcs element, 1-15
- CalcOptional element
  - of a Calc or a CalcOptionalDefinitions element, 1-16
  - supporting calculation options, 1-10
- CalcOptionalDefinitions element
  - applying options to all calculations, 1-10
  - of a AWMCalcs element, 1-17
- calculation templates
  - adding options, 1-10
  - description, 1-8
  - example of, 1-9
- Category element
  - of an AWMCalcs element, 1-17
- changed method, 2-14, 2-18
- class attribute
  - of a Plugin element, 2-20
- cleanup method, 2-13, 2-18
- configuring Analytic Workspace Manager for plug-ins, 2-2
- conn parameter
  - of AWMPPlugin methods, 2-6
  - of ViewerPlugin methods, 2-13
- Connection object, 2-3
- CubeViewerPlugin class, 3-15

- cube.xml document
  - example of, 3-12
  - location in navigation tree, 1-5
- custom calculations, 1-8
- custom SQL reports
  - adding to navigation tree, 1-1
  - example, 1-2, 1-3, 1-4
- customizingAWM\_examples11202.zip file
  - containing example code, 3-1

## D

---

- default attribute
  - of a Param element, 1-18
- DeleteDimPlugin class, 3-5
- description attribute
  - of a Calc element, 1-15
  - of a Category element, 1-17
- Description element
  - of an AWMPlugins element, 2-20
- DimEditorPlugin class, 3-17
- dimension.xml
  - example of, 3-10
  - location in navigation tree, 1-5
- dollar sign
  - enclosing expression, 1-17

## E

---

- EditorPlugin interface
  - examples of, 3-17
  - specification, 2-13
- enabling plug-ins, 2-2
- examples
  - downloadable, 3-1
  - of awmcalcs.xml, 1-9, 1-11, 1-13
  - of AWMPlugin, 3-2, 3-5
  - of awmplugin.xml, 3-24
  - of awmtree.xml, 1-2, 1-3, 1-4
  - of aw.xml, 3-15
  - of cube.xml document, 3-12
  - of dimension.xml, 3-10
  - of EditorPlugin, 3-17
  - of schema.xml, 1-6
  - of ViewerPlugin, 3-10, 3-12, 3-15
- expression attribute
  - of a Calc element, 1-15
  - of a CalcOptional element, 1-17
  - of an Item element, 1-17
- expression syntax, 1-8

## F

---

- folders
  - creating in navigation tree, 1-2

## G

---

- getMenu method, 2-4, 2-6, 2-17
- getPanel method, 2-13, 2-17

## H

---

- handle method, 2-4, 2-6, 2-17
- headings
  - for XML documents, 1-1, 1-8, 2-19
- hypertext parameters
  - creating links by using, 1-9
  - enclosed by braces, 1-15
  - referencing two or more, 1-15

## I

---

- icon attribute
  - example of, 1-6
  - of an AWMNode element, 1-8
- icons
  - in JAR file, 1-5
  - in JAR file, example of, 1-6
- isSupported method, 2-3, 2-6, 2-17
- isViewerForType method, 2-13, 2-17
- Item element
  - of a Param element, 1-17

## J

---

- JAR files
  - containing multiple XML documents and plug-ins, 2-18
  - directory structure of, 1-6
  - files for icons included in, 1-5
  - loaded on startup, 2-18
  - location for, 2-18
  - required by plug-ins, 2-17

## L

---

- leaf element
  - defined, 1-3
- LevelViewerPlugin class, 3-10

## M

---

- MeasureViewerPlugin class, 3-12

## N

---

- name attribute
  - of a Calc element, 1-15
  - of a CalcOptional element, 1-16
  - of a Category element, 1-17
  - of a Param element, 1-18
  - of a Plugin element, 2-20
  - of an AWMNode element, 1-7
- name parameter
  - of ViewerPlugin methods, 2-13

## O

---

- obj parameter
  - of AWMPlugin methods, 2-6
- OLAP expression syntax, 1-8

options  
  adding to calculation templates, 1-10  
Oracle Technology Network (OTN), 3-1

## P

---

Param element  
  of a Calc or a Params element, 1-17  
Params element  
  of an AWMCalcs element, 1-18  
params parameter  
  keys and values for a ViewerPlugin and  
    EditorPlugin, 2-14  
  keys and values for a ViewerPlugin, examples  
    of, 2-15  
  keys and values for an AWMPPlugin, 2-9  
  keys and values for an AWMPPlugin, examples  
    of, 2-10  
  keys and values for an EditorPlugin, examples  
    of, 2-16  
  of AWMPPlugin methods, 2-6  
  of ViewerPlugin methods, 2-13  
parent parameter  
  of AWMPPlugin methods, 2-6  
  of EditorPlugin methods, 2-14  
plug-in descriptions  
  creating, 2-19  
  example of, 3-24  
plug-in directory  
  specifying, 2-2  
Plugin element  
  of an AWMPPlugins element, 2-19  
plugin11202 package, 1-6  
plugin11202.jar file  
  containing example plug-ins, 3-1  
plug-ins  
  descriptions of, 2-19  
  enabling, 2-2  
  examples of, 3-2, 3-5, 3-10, 3-12, 3-15, 3-17  
  including package when specifying, 1-6  
  prerequisites for creating, 2-17  
  providing descriptions of, 2-18

## R

---

refreshing the navigation tree, 2-4  
refreshTree method, 2-4, 2-6, 2-17  
reports  
  *See* SQL reports  
revert method, 2-13, 2-18  
root element  
  of a calculation template, 1-15  
  of a plug-in description, 2-19  
  of a SQL report, 1-7  
run-time selections  
  stored by type parameter of AWMNode, 1-4

## S

---

save method, 2-13, 2-18  
schema.xml document

  example of, 1-6  
  location in navigation tree, 1-5  
setValueChanged method, 2-13, 2-18  
showHelp method, 2-13  
ShowIfQueryTrue element  
  of an AWMNode element, 1-8  
sql attribute  
  of a ShowIfQueryTrue element, 1-8  
  of an AWMNode element, 1-7  
SQL reports  
  adding to navigation tree, 1-1  
  creating in object folders, 1-5  
  creating XML documents for, 1-1  
  examples, 1-2, 1-3, 1-4  
  XML document structure, 1-7  
syntax for expressions, 1-8

## T

---

text attribute  
  of a CalcOptional element, 1-16  
  of an Item element, 1-17  
type attribute  
  of a CalcOptional element, 1-16  
  of a Param element, 1-18  
  of an AWMNode element, 1-7  
type parameter  
  of an AWMNode, storing run-time selection, 1-4  
  of AWMPPlugin methods, 2-6

## U

---

ui attribute  
  of a Calc element, 1-15

## V

---

validate method, 2-13, 2-18  
version attribute  
  of a Plugin element, 2-20  
ViewerPlugin interface  
  examples of, 3-10, 3-12, 3-15  
  specification, 2-12  
viewSql attribute  
  of an AWMNode element, 1-7  
ViewXMLPlugin class, 3-2

## X

---

XML documents  
  for calculations, 1-8  
  for descriptions of plug-ins, 2-18, 3-24  
  for SQL reports, 1-1  
  headings, 1-1, 1-8, 2-19  
  loaded on startup, 2-3  
XML formats  
  for calculations, 1-14  
  for descriptions of plug-ins, 2-19  
  for SQL reports, 1-7

