Contents

1 Customizing With XML Documents

Describing SQL Reports

Creating an XML Document for SQL Reports
Example of a Simple Report
Creating Report Folders
Using Bind Variables to Restrict the Report Contents
Creating Reports in Object Folders
Reference: Elements for SQL Reports

Describing Calculation Templates

Creating an XML Document for Calculations
Example of a Simple Calculation Template
Adding an Option to a Calculation
Creating More Complex Calculation Templates
Reference: Elements for Calculations

2 Introducing Analytic Workspace Manager Plug-ins

Describing Analytic Workspace Manager Plug-ins

Enabling Analytic Workspace Manager Plug-ins
How Analytic Workspace Manager Calls a Plug-in
Calling an AWMPlugin
Calling a ViewerPlugin or EditorPlugin

Describing the AWMPlugin Interface

Values for the type and obj Parameters
Elements in the params Map for an AWMPlugin
3 Examples of Analytic Workspace Manager Plug-ins

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

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 AWMCalc 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
You can add custom objects and menus to the Oracle Analytic Workspace Manager 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 12c Release 1 (12.1) with the OLAP option.

**Documentation Accessibility**

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.
Access to Oracle Support
Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs 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:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>boldface</td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td>italic</td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td>monospace</td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>
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
   <?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.
6. 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.

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.
Describing SQL Reports

Customizing With XML Documents

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
<?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.
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
<?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.
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 plugin112.

**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 plugin112. 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 plugin112. In Example 1–4, the icon attribute specification includes the directory: icon="plugin112/button.jpg". In the examples of XML documents in Chapter 3, the viewClass attribute specifications include the package name, as in viewClass="plugin112.DimEditorPlugin" in Example 3–7 on page 3-15.

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

```xml
<?xml version="1.0" encoding='US-ASCII' />
<AWMTree>
    <AWMNode name='My Tables' icon='plugin112/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>
```
An XML document for SQL reports has the basic format shown in Example 1–5.

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

```xml
<AWMTree>
  <AWMNode>
    <AWMNode>
      <ShowIfQueryTrue>
        <AWMNode>
          .
          .
          .
      </AWMNode>
    </AWMNode>
  </AWMNode>
</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. For information on the expression syntax, see *Oracle OLAP Expression Syntax Reference*.

**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.

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
   <?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.

```xml
<?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.
Describing Calculation Templates

Customizing With XML Documents

Figure 1–7 Providing an Option to a Calculation

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

```xml
<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

```xml
<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"
            
```
Describing Calculation Templates

```xml
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:

```xml
<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>
```
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
<?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}.">
      <CalcOptional name="percentages" />
      <CalcOptional name="truncate" />
      <expression>((measure:1) - (measure:2)) / abs((measure:2))"/>
  </Calc>
</AWMCalcs>
```
Describing Calculation Templates

An XML document for calculations has the basic format shown in Example 1–10.

Example 1–10 Basic XML Structure for Calculations

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

Reference: Elements for Calculations

An XML document for calculations has the basic format shown in Example 1–10.
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

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

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

A $\text{<Calc>}$ element can contain a $\text{<CalcOptional>}$ element. A $\text{<CalcOptionalDefinitions>}$ element can have one or more $\text{<CalcOptional>}$ elements.

Contains
None

Attributes
$\text{<CalcOptional>}$ 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 $\text{<Calc>}$ element.
<CalcOptionalDefinitions>
Contains one or more <CalcOptional> elements so they can be referenced by multiple calculations. This element must appear directly after <AWMCals>.

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 12.1, you can extend the functionality of Analytic Workspace Manager in Oracle Database 12c Release 1 (12.1) 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

![Configuration Item on the Tools Menu](image)

**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 **plug-in** directory shown in the figure.
Introducing Analytic Workspace Manager Plug-ins

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
Describing Analytic Workspace Manager Plug-ins

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 AWMPlugin Interface

The following is the oracle.olap.awm.plugin.AWMPlugin interface.

```java
package oracle.olap.awm.plugin
import java.awt.Frame;
import java.sql.Connection;
import java.util.Map;
import oracle.AWXML.AW;

public interface AWMPlugin
{
    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 AWMPlugin interface in the sequence illustrated in Figure 2-4.
Describing the AWMPlugin Interface

Figure 2–4  Sequence of Calls to an AWMPlugin

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

- $\text{conn}$, which is a `java.sql.Connection` object that represents the current connection to the Oracle Database instance.
- $\text{type}$, 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.
- $\text{obj}$, 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 $\text{obj}$ parameter values, see "Values for the type and obj Parameters".
- $\text{aw}$, which is `null`. This parameter exists for compatibility with 10g plug-ins, for which $\text{aw}$ was an `oracle.AWXML.AW` object.
- $\text{params}$, 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 AWMPlugin" on page 2-9.
- $\text{parent}$, which is a `java.awt.Frame` object that Analytic Workspace Manager passes to the $\text{handle}$ 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 AWMPlugin 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 AWMPlugin 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>
<thead>
<tr>
<th>Navigation Tree Object</th>
<th><code>type</code> Parameter Value</th>
<th><code>obj</code> Parameter Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Databases</td>
<td>Databases</td>
<td>Databases</td>
</tr>
<tr>
<td>Database name</td>
<td>DATABASE</td>
<td>Database identifier</td>
</tr>
<tr>
<td>Schemas</td>
<td>SCHEMA_FOLDER</td>
<td>Schema name</td>
</tr>
<tr>
<td>Schema name</td>
<td>SCHEMA</td>
<td>Schema name</td>
</tr>
<tr>
<td>Analytic Workspaces</td>
<td>WORKSPACE_FOLDER</td>
<td>Schema name</td>
</tr>
<tr>
<td>Analytic workspace name</td>
<td>WORKSPACE</td>
<td><code>AW</code></td>
</tr>
<tr>
<td>Dimensions</td>
<td>DIMENSION_FOLDER</td>
<td><code>AW</code></td>
</tr>
<tr>
<td>Dimension name</td>
<td>DIMENSION</td>
<td><code>MdmStandardDimension</code> or <code>MdmTimeDimension</code></td>
</tr>
<tr>
<td>Levels</td>
<td>DIMENSION_LEVEL_FOLDER</td>
<td><code>MdmStandardDimension</code> or <code>MdmTimeDimension</code></td>
</tr>
<tr>
<td>Level name</td>
<td>DIMENSION_LEVEL</td>
<td><code>MdmDimensionLevel</code></td>
</tr>
<tr>
<td>Hierarchies</td>
<td>DIMENSION_HIERARCHY_FOLDER</td>
<td><code>MdmStandardDimension</code> or <code>MdmTimeDimension</code></td>
</tr>
<tr>
<td>Hierarchy name</td>
<td>DIMENSION_HIERARCHY</td>
<td><code>MdmLevelHierarchy</code> or <code>MdmValueHierarchy</code></td>
</tr>
<tr>
<td>Attributes</td>
<td>DIMENSION_ATTRIBUTE_FOLDER</td>
<td><code>MdmStandardDimension</code> or <code>MdmTimeDimension</code></td>
</tr>
<tr>
<td>Attribute name</td>
<td>DIMENSION_ATTRIBUTE</td>
<td><code>MdmBaseAttribute</code></td>
</tr>
<tr>
<td>Mappings</td>
<td>DIMENSION_MAP</td>
<td><code>MdmStandardDimension</code> or <code>MdmTimeDimension</code></td>
</tr>
<tr>
<td>Views</td>
<td>DIMENSION_VIEW_FOLDER</td>
<td><code>MdmStandardDimension</code> or <code>MdmTimeDimension</code></td>
</tr>
<tr>
<td>View name</td>
<td>DIMENSION_VIEW</td>
<td><code>MdmStandardDimension</code> or <code>MdmTimeDimension</code></td>
</tr>
</tbody>
</table>
### Table 2–1 (Cont.) Type Values and Objects for Navigation Tree Objects

<table>
<thead>
<tr>
<th>Navigation Tree Object</th>
<th>type Parameter Value</th>
<th>obj Parameter Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Security</td>
<td>DATA_SECURITY</td>
<td>MdmStandardDimension or MdmTimeDimension</td>
</tr>
<tr>
<td>dimension.xml object</td>
<td>AWMTree_parent_node_name</td>
<td>For a folder, the name of the &lt;AWMNode&gt;. For a value returned by the SQL query, the run-time object name.</td>
</tr>
<tr>
<td>Cubes</td>
<td>CUBE_FOLDER</td>
<td>AWi</td>
</tr>
<tr>
<td>Cube name</td>
<td>CUBE</td>
<td>MdmCube</td>
</tr>
<tr>
<td>Measures</td>
<td>CUBE_MEASURE_FOLDER</td>
<td>MdmCube</td>
</tr>
<tr>
<td>Measure name</td>
<td>CUBE_MEASURE</td>
<td>MdmBaseMeasure</td>
</tr>
<tr>
<td>Calculated Measures</td>
<td>CUBE_DERIVED_MEASURE_FOLDER</td>
<td>MdmCube</td>
</tr>
<tr>
<td>Calculated measure name</td>
<td>CUBE_DERIVED_MEASURE</td>
<td>MdmDerivedMeasure</td>
</tr>
<tr>
<td>Mappings</td>
<td>CUBE_MAP</td>
<td>MdmCube</td>
</tr>
<tr>
<td>Views</td>
<td>CUBE_VIEW_FOLDER</td>
<td>MdmCube</td>
</tr>
<tr>
<td>View name</td>
<td>CUBE_VIEW</td>
<td>MdmCube</td>
</tr>
<tr>
<td>Cube Scripts</td>
<td>CUBE_SCRIPT_FOLDER</td>
<td>MdmCube</td>
</tr>
<tr>
<td>Cube script name</td>
<td>CUBE_SCRIPT</td>
<td>Script_name</td>
</tr>
<tr>
<td>Data Security</td>
<td>DATA_SECURITY</td>
<td>MdmCube</td>
</tr>
<tr>
<td>cube.xml object</td>
<td>AWMTree_parent_node_name</td>
<td>For a folder, the name of the &lt;AWMNode&gt;. For a value returned by the SQL query, the run-time object name.</td>
</tr>
<tr>
<td>Measure Folders</td>
<td>MEASURE_FOLDER_FOLDER</td>
<td>AWi</td>
</tr>
<tr>
<td>Measure folder name</td>
<td>Measure_folder_name</td>
<td>MdmOrganizationalSchema</td>
</tr>
<tr>
<td>Languages</td>
<td>LANGUAGE</td>
<td>Languages</td>
</tr>
<tr>
<td>aw.xml object</td>
<td>AWMTree_parent_node_name</td>
<td>For a folder, the name of the &lt;AWMNode&gt;. For a value returned by the SQL query, the run-time object name.</td>
</tr>
<tr>
<td>OLAP DML Programs</td>
<td>AWMTREE OLAP DML Programs</td>
<td>OLAP DML Programs</td>
</tr>
<tr>
<td>Program name</td>
<td>AWMTREE OLAP DML Programs</td>
<td>Program_name</td>
</tr>
<tr>
<td>Maintenance Scripts</td>
<td>MAINTENANCE_SCRIPT_FOLDER</td>
<td>Schema name</td>
</tr>
<tr>
<td>Script name</td>
<td>MAINTENANCE_SCRIPT</td>
<td>Script name</td>
</tr>
<tr>
<td>Maintenance Reports</td>
<td>AWMTREE_Maintenance Reports</td>
<td>Maintenance Reports</td>
</tr>
<tr>
<td>Maintenance_report_name</td>
<td>AWMTREE_maintenance_report_name</td>
<td>Maintenance_report_name</td>
</tr>
<tr>
<td>schema.xml object</td>
<td>AWMTree_parent_node_name</td>
<td>For a folder, the name of the &lt;AWMNode&gt;. For a value returned by the SQL query, the run-time object name.</td>
</tr>
<tr>
<td>Data Security Roles</td>
<td>ACL_DOCUMENT_FOLDER</td>
<td>Data Security Roles</td>
</tr>
<tr>
<td>Security role name</td>
<td>Security role name</td>
<td>Security role name</td>
</tr>
<tr>
<td>Reports</td>
<td>AWMTREE_Reports</td>
<td>Reports</td>
</tr>
<tr>
<td>Report name</td>
<td>AWMTREE_report_name</td>
<td>Report name</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWM_VERSION</td>
<td>A String that is the version number of Analytic Workspace Manager.</td>
</tr>
<tr>
<td>DATAPROVIDER</td>
<td>An oracle.olapi.metadata.mdm.MdmMetadataProvider that is the metadata provider for the session.</td>
</tr>
<tr>
<td>BIND_MAP</td>
<td>An empty Map.</td>
</tr>
<tr>
<td>DATASOURCE</td>
<td>A java.sql.DataSource.</td>
</tr>
<tr>
<td>GETDATAPROVIDER</td>
<td>An implementation of the oracle.olap.awm.plugin.OLAPDataProvider interface. The interface specifies a method that gets an MdmMetadataProvider.</td>
</tr>
</tbody>
</table>

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

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>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>aw_name</td>
<td>A String that contains the name of the currently selected analytic workspace.</td>
</tr>
<tr>
<td>owner</td>
<td>A String that contains the name of the owner of the currently selected analytic workspace.</td>
</tr>
<tr>
<td>schema</td>
<td>A String that contains the name of the owner of the currently selected schema.</td>
</tr>
<tr>
<td>user</td>
<td>A String that contains the name of the user who is connected to the database.</td>
</tr>
<tr>
<td>Other bind variables</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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 &quot;Example params Map Elements for an AWMPlugin&quot; and the examples in Chapter 3, &quot;Examples of Analytic Workspace Manager Plug-ins&quot;.</td>
</tr>
</tbody>
</table>

---

**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.

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

### Table 2–5  Keys and Values of the `params` Map for `DeleteDimPlugin`

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AW</td>
<td>An <code>AW</code></td>
<td>The current analytic workspace object.</td>
</tr>
<tr>
<td>AWM_VERSION</td>
<td>12.1.0.1.0</td>
<td>The version number of Analytic Workspace Manager.</td>
</tr>
<tr>
<td>BIND_MAP</td>
<td>A <code>Map</code></td>
<td>A container for bind variables related to the current object.</td>
</tr>
<tr>
<td>DATAPROVIDER</td>
<td>An <code>MdmMetadataProvider</code></td>
<td>The metadata provider for the session.</td>
</tr>
<tr>
<td>DATASOURCE</td>
<td>A <code>DataSource</code></td>
<td>The current data source.</td>
</tr>
<tr>
<td>GETDATAPROVIDER</td>
<td>An <code>OLAPDataProvider</code></td>
<td>An implementation of the <code>OLAPDataProvider</code> interface.</td>
</tr>
<tr>
<td>ISFOLDER</td>
<td><code>FALSE</code></td>
<td>Specifies that the <code>&lt;AWMNode&gt;</code> is not a folder.</td>
</tr>
<tr>
<td>NODE_TYPE</td>
<td><code>MyDims</code></td>
<td>The name of the parent <code>&lt;AWMNode&gt;</code>.</td>
</tr>
<tr>
<td>TYPE</td>
<td><code>dimobj</code></td>
<td>The type of the <code>&lt;AWMNode&gt;</code> that specifies the plug-in.</td>
</tr>
</tbody>
</table>

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.

```java
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`

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

**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.

```java
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
Describing the ViewerPlugin and EditorPlugin Interfaces

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.

```java
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>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AW</code></td>
<td>An <code>oracle.olapi.metadata.deployment.AW</code>.</td>
</tr>
<tr>
<td><code>aw_name</code></td>
<td>A <code>String</code> that contains the name of the currently selected analytic workspace.</td>
</tr>
<tr>
<td><code>DATAPROVIDER</code></td>
<td>An <code>oracle.olapi.metadata.mdm.MdmMetadataProvider</code> object that is the metadata provider for the session.</td>
</tr>
<tr>
<td><code>DATASOURCE</code></td>
<td>A <code>java.sql.DataSource</code> object.</td>
</tr>
</tbody>
</table>
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 (Cont.) Keys and Values of the params Map for MeasureViewerPlugin

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AW</td>
<td>An AW</td>
<td>The current analytic workspace object.</td>
</tr>
<tr>
<td>aw_name</td>
<td>GLOBAL</td>
<td>The name of the current analytic workspace.</td>
</tr>
<tr>
<td>cube_name</td>
<td>UNITS_CUBE</td>
<td>The name of the current cube.</td>
</tr>
<tr>
<td>DATAPROVIDER</td>
<td>An MdmMetadataProvider</td>
<td>The metadata provider for the session.</td>
</tr>
<tr>
<td>DATASOURCE</td>
<td>A DataSource</td>
<td>The current data source.</td>
</tr>
</tbody>
</table>
Describing the ViewerPlugin and EditorPlugin Interfaces

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

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AW</td>
<td>An AW.</td>
<td>The current analytic workspace object.</td>
</tr>
<tr>
<td>aw_name</td>
<td>GLOBAL</td>
<td>The name of the current analytic workspace.</td>
</tr>
<tr>
<td>DATAPROVIDER</td>
<td>An MdmMetadataProvider</td>
<td>The metadata provider for the session.</td>
</tr>
<tr>
<td>DATASOURCE</td>
<td>A DataSource</td>
<td>The current data source.</td>
</tr>
<tr>
<td>dimobj</td>
<td>CHANNEL</td>
<td>The run-time value of the dimension currently selected in the MyDims folder.</td>
</tr>
<tr>
<td>ISFOLDER</td>
<td>TRUE</td>
<td>Indicates that the navigation tree object is a folder.</td>
</tr>
<tr>
<td>NODE_TYPE</td>
<td>MyDims</td>
<td>The name of the parent &lt;AWMNode&gt;.</td>
</tr>
<tr>
<td>owner</td>
<td>GLOBAL</td>
<td>The name of the owner of the analytic workspace.</td>
</tr>
<tr>
<td>schema</td>
<td>GLOBAL</td>
<td>The name of the current schema.</td>
</tr>
<tr>
<td>TYPE</td>
<td>dimobj</td>
<td>The type of the &lt;AWMNode&gt;.</td>
</tr>
<tr>
<td>user</td>
<td>global</td>
<td>The name of the current user.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AW</td>
<td>An AW.</td>
<td>The current analytic workspace object.</td>
</tr>
<tr>
<td>aw_name</td>
<td>GLOBAL</td>
<td>The name of the current analytic workspace.</td>
</tr>
<tr>
<td>DATAPROVIDER</td>
<td>An MdmMetadataProvider</td>
<td>The metadata provider for the session.</td>
</tr>
<tr>
<td>DATASOURCE</td>
<td>A DataSource</td>
<td>The current data source.</td>
</tr>
<tr>
<td>dimobj</td>
<td>CHANNEL</td>
<td>The run-time value of the dimension currently selected in the MyDims folder.</td>
</tr>
<tr>
<td>ISFOLDER</td>
<td>TRUE</td>
<td>Indicates that the navigation tree object is a folder.</td>
</tr>
<tr>
<td>NODE_TYPE</td>
<td>MyDims</td>
<td>The name of the parent &lt;AWMNode&gt;.</td>
</tr>
<tr>
<td>owner</td>
<td>GLOBAL</td>
<td>The name of the owner of the analytic workspace.</td>
</tr>
<tr>
<td>schema</td>
<td>GLOBAL</td>
<td>The name of the current schema.</td>
</tr>
<tr>
<td>TYPE</td>
<td>dimobj</td>
<td>The type of the &lt;AWMNode&gt;.</td>
</tr>
<tr>
<td>user</td>
<td>global</td>
<td>The name of the current user.</td>
</tr>
</tbody>
</table>
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 awm11.2.0.2.0.jar file in your development environment.

- Compile the code with JDK 1.6.

Note: Only plug-ins compiled with JDK 1.6 are compatible with Analytic Workspace Manager in 12c Release 1 (12.1).

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 1.6, 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) website. The download includes the compiled class files for the plug-ins, as well. The OTN website is at http://www.oracle.com/technetwork/database/options/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 AWM Plug-ins for Oracle OLAP 11g section, in the AWM Java Plug-in and XML for 11.2.0.1 line, click examples.

Download the compressed file and extract the contents. The compressed file contains the following files.

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>readme.txt</td>
<td>Briefly describes the contents of the zip file.</td>
</tr>
<tr>
<td>awmtree.xml</td>
<td>Contains the XML for Example 1-3, &quot;Passing the Name of a View to a SELECT Statement&quot; on page 1-4.</td>
</tr>
<tr>
<td>plugin112.jar</td>
<td>Contains a directory named plugin112, which is the package containing the examples. In the directory are the xml, java, and class files for the examples in Chapter 3.</td>
</tr>
</tbody>
</table>
You put the awmcalcs.xml and awmtree.xml files in the same directory as the Analytic Workspace Manager executable file. You put the plugin112.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

```java
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.AWMPPlugin;
import oracle.olap.metadata.mdm.MdmBaseMeasure;
import oracle.olap.metadata.mdm.MdmDerivedMeasure;
import oracle.olap.metadata.mdm.MdmMetadataProvider;
import oracle.olap.metadata.mdm.MdmObject;
/**
 */
```
public class ViewXMLPlugin implements AWMPlugin {

    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**

---

### XML for PROFIT

```xml
<Metadata
    Version="2.0"
    MinimumDatabaseVersion="12.1.01">
  <Cubes>
    <Cube
      Owner="GLOBAL"
      Name="UNITS_CUBE">
      <Measure>
        <DerivedMeasure
          ETMeasureColumnName="PROFIT"
          Name="PROFIT"
          MeasureExpression="UNITS_CUBE.SALES - UNITS_CUBE.COST">
          <Classification>
            <Description
              Type="AWMLDescriptionType"
              Language="AMERICAN"
              Value="Profit"/>
          </Classification>
          <LinkedMeasure/>
        </Measure>
      </Cube>
    </Cubes>
  </Metadata>
```  

---

**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

```java
package plugin112;

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';
      JOptionPane.showMessageDialog(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;
Examples of AWMPlugin

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

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.
Examples of ViewerPlugin and EditorPlugin

If the user clicks Example Plug-in: Delete Dimension CUSTOMER, then DeleteDimPlugin displays the dialog box shown in Figure 3–3.

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
<?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>
</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**

```java
package plugin112;

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.
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
<?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="measureview"
      viewClass="plugin112.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

```java
package plugin112;

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.
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
<?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='plugin112.DimEditorPlugin'
    </AWMNode>
  </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>
```
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**

```java
package plugin112;

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
Examples of ViewerPlugin and EditorPlugin

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

```java
package plugin112;

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 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;
    } else {
        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.");
            }
        } else {
            MdmObject mobj = mdp.getMetadataObject(schema + "." + dimName);
        }
    }
}
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 3–10 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 3–11 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 3–12 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 awmplugin.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 awmplugin.xml document in Example 3–10.

Figure 3–14  Plugins Tab in the About Dialog Box

Example 3–10 shows the awmplugin.xml document that produces the result shown in Figure 3–14.

Example 3–10  Creating an awmplugins.xml Document

```xml
<AWMPlugins>
  <Plugin name="Cube Viewer Plug-in" version="1.0"
    class="plugin112.CubeViewerPlugin">
    <Description>Displays the name of a cube.</Description>
  </Plugin>
  <Plugin name="Level Viewer Plug-in" version="1.0"
    class="plugin112.LevelViewerPlugin">
    <Description>Displays the name of a level.</Description>
  </Plugin>
  <Plugin name="Measure Viewer Plug-in" version="1.0"
    class="plugin112.MeasureViewerPlugin">
    <Description>Displays the name of a measure.</Description>
  </Plugin>
  <Plugin name="Delete Dimension Plug-in" version="2.0"
    class="plugin112.DeleteDimPlugin">
    <Description>Deletes a dimension in the MyDims folder.</Description>
  </Plugin>
  <Plugin name="Edit Dimension Plug-in" version="2.0"
    class="plugin112.DimEditorPlugin">
    <Description>Edits the short description of a dimension.</Description>
  </Plugin>
  <Plugin name="View XML Plug-in" version="1.0"
    class="plugin112.ViewXMLPlugin">
    <Description>Displays the XML for an OLAP measure.</Description>
  </Plugin>
</AWMPlugins>
```
Symbols

{}  enclosing bind variable,  1-4, 1-15
      enclosing hypertext parameter,  1-15
$    enclosing expression,  1-16

A

Analytic Workspace Manager
configuring for plug-ins,  2-2
ways of customizing,  ix
aw parameter
of AWMPlugin 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 AWMTree element,  1-7
AWMPlugin interface
examples of,  3-2, 3-5
implementations loaded on startup,  2-3
specification,  2-5
AWMPlugins element
root of a plug-in description,  2-19
awmplugin.xml document
description,  2-18
example of,  3-24
AWMTree element
root of a SQL report,  1-7
AWMTree 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 AWMPlugin,  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 AWMPlugin,  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, 2-17
class attribute
of a Plugin element,  2-19
cleanup method,  2-13, 2-17
compressed file
containing example code,  3-1
configuring Analytic Workspace Manager for plug-ins,  2-2
cnn parameter
of AWMPlugin 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

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, 2-19
DimEditorPlugin class, 3-17
dimension.xml
  example of, 3-10
  location in navigation tree, 1-5
dollar sign
  enclosing expression, 1-16

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, 1-16
  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, 2-18
hypertext parameters
  creating links by using, 1-10
  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, 2-19
  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
Index-3

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 AWMPlugin, 2-9
    keys and values for an AWMPlugin, examples of, 2-10
    keys and values for an EditorPlugin, examples of, 2-16
    of AWMPlugin methods, 2-6
    of ViewerPlugin methods, 2-13
  parent parameter
    of AWMPlugin methods, 2-6
    of EditorPlugin methods, 2-14
  plug-in descriptions
    creating, 2-18
    example of, 3-24
  plug-in directory
    specifying, 2-2
  Plugin element
    of an AWMPlugins element, 2-19
  plugin112 package, 1-6
  plugin112.jar file
    containing example plug-ins, 3-1
  plug-ins
    descriptions of, 2-18
    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, 2-17
  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 of, 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, 1-17
    of an AWMNode element, 1-7
  type parameter
    of an AWMNode, storing run-time selection, 1-4
    of AWMPlugin methods, 2-6

U
  ui attribute
    of a Calc element, 1-15

V
  validate method, 2-13, 2-17
  version attribute
    of a Plugin element, 2-19
  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, 2-18
    loaded on startup, 2-3
  XML formats
    for calculations, 1-14
    for descriptions of plug-ins, 2-19
    for SQL reports, 1-7