Oracle® Business Activity Monitoring

Architect User's Guide

10*g* (10.1.3.1.0)

B28992-01

October 2006

ORACLE®

Oracle Business Activity Monitoring Architect User's Guide, 10*g* (10.1.3.1.0)

B28992-01

Contents

Preface

This preface explains how to use this guide. It includes the following topics:

Intended Audience

This manual is intended for system architects responsible for message source, data source, and data object management in Oracle Business Activity Monitoring. Using the Administrator application, the system architect creates and manages message sources, data source, data objects, and alerts.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation

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

Accessibility of Links to External Web Sites in Documentation

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

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documents

For more information, see the following manuals in the Oracle Business Activity Monitoring Release 10*g* documentation set:

- *Oracle Business Activity Monitoring Installation Guide*
- *Oracle Business Activity Monitoring Administrator's Guide*
- *Oracle Business Activity Monitoring Active Studio User's Guide*

Conventions

The following text conventions are used in this document:

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

What's New

This section contains information about the new features in release 10.1.3.1.0.

New Features for Release 10.1.3.1.0

Release 10.1.3.1.0 includes the following new features:

- **Sensor integration with BPEL**

  You can create sensor actions in Oracle BPEL Process Manager to publish sensor data as data objects on an Oracle Business Activity Monitoring Server. See the *Oracle BPEL Process Manager Developer's Guide* for more information.

- **HTML calculations in List views**

  You can add HTML tags to calculated fields in a report in order to add special formatting to the field. See the *Oracle Business Activity Monitoring Active Studio User's Guide* for more information.

- **National Language Support in Active Viewer**

  Active Viewer will display the appropriate language and numeric formats that correspond with the system on which it is installed. See the *Oracle Business Activity Monitoring Installation Guide* for more information.

- **Streamlined install process**

  Oracle Business Activity Monitoring just got easier to install. The InstallShield wizard guides you through each step and installs any dependencies needed on your host. See the *Oracle Business Activity Monitoring Installation Guide* for more information.

1

Getting Started

This chapter introduces the Oracle Business Activity Monitoring Architect application. Architect is the thin user interface for the data designer. Through Architect, the data designer creates and manages data objects, manages transaction source processing, and imports and creates metadata.

This chapter contains the following topics:

Accessing Architect

Always use the start page to start Web applications. Do not start Web applications from a direct URL to the application. This ensures that caching works correctly. Also, do not use `localhost` in the URL instead of the host name.

To access Architect:

1. In Microsoft Internet Explorer, go to http://<host>:<http_port>/oraclebam, where *host* is the name of the server where Oracle Business Activity Monitoring is installed.

   The Start Page opens.

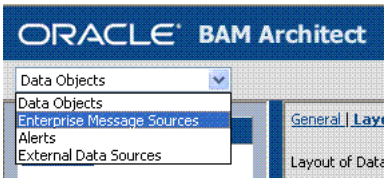
2. Click **Architect**.

   Architect launches in a new browser window.

Locating Architect Functions

To use the different functions within Architect you must choose each function from the list in the upper left corner of the Architect window as shown in Figure 1–1.

Figure 1–1 Architect Function List



Refer to the following chapters for more information about each function:

Features and Components

This section describes Oracle Business Activity Monitoring features and components.

Features

Oracle Business Activity Monitoring includes the following features:

Active Data Architecture. Oracle Business Activity Monitoring provides an active data architecture that dynamically moves real-time data to end users through every step of the process. This solution actively collects data, applies rules designed to monitor changes, and delivers the information in reports to users.

Real-time Reports. Real-time reports containing current data are delivered as soon as data changes occur. This is possible because of data in the Active Data Cache and the connections to real-time transactional feeds.

Active Presentations in Reports. Reports display active data presentations where data continuously updates, formats, and displays. When data changes, the display changes in real-time.

Instant Alerts. Alerts, based on rules and events occurring in real-time, are delivered through e-mail.

Rules-Based Active Delivery. In an event-driven solution, the information finds the target users instead of requiring users to query for the information on their own initiative. The reports are initially designed for delivery to end users based on data changing or events triggering. For the end user, the result is zero-click reporting that is always relevant.

High Performance, Scalable Architecture. Oracle Business Activity Monitoring is scalable to handle large amounts of complex, real-time enterprise data. Enterprise Link uses data flow technology to select the correct raw data and then transform and perform calculations required by the data designer. The transformed data is delivered to the Active Data Cache in a ready-to-use state for fast access.

Components

Oracle Business Activity Monitoring includes the following architectural components and applications:

Active Data Cache is designed and optimized to handle large amounts of data in a real-time solution. To make data readily accessible and deliverable, it maintains real-time views of the data. The data feed to the Active Data Cache is a combination of business data sources, from data warehouse information to transactional feeds and other enterprise sources. Enterprise Link sends this information to the Active Data Cache in a continuous stream as data changes occur.

Enterprise Link connects Oracle Business Activity Monitoring to real-time data with message queues and also to other information sources such as database servers, flat files, and XML sources.

Event Engine monitors complex data conditions and implements specified rules. Rules can include a series of conditions and actions attached to an event. The Event Engine continuously monitors the information in the Active Data Cache for certain conditions and executes the related actions defined in associated rules.

Report Engine applies the report definitions to the data sets retrieved from the Active Data Cache for presentation in a browser. It manages information paging for viewing and printing reports. After reports are created, they are stored in the Active Data Cache so that report creation is not repeated each time. Most reporting views are designed to support live, active displays of data changing in real-time.

Active Viewer is the thin user interface for viewing reports. Active Messenger is client-side notification software. When new information is available, the user receives an e-mail that contains a link to the information. The user clicks the link and the report is displayed in Active Viewer. Report formats include charts, lists, KPIs, crosstabs, spreadsheets, and more.

Active Studio is the thin user interface for the power user. Through Active Studio, the power user can create and edit reports. Reports can be shared with other users and rules can be created for determining the scheduling and delivery of the reports. Report types include charts, lists, KPIs, crosstabs, spreadsheets, and more.

Architect is the thin user interface for the data designer. Through Architect, the data designer creates and manages data objects in the Active Data Cache and manages real-time message processing.

Administrator is the thin user interface for the system administrator who is responsible for user management and overall server management. Using Administrator, the system administrator manages users and security levels, monitors loading to the Active Data Cache, and configures Oracle Business Activity Monitoring services.

2

Working With Data Objects

This chapter contains the information needed to create and manage data objects, including assigning permissions, managing folders, creating security filters, and adding dimensions and hierarchies.

This chapter contains the following topics:

Defining Data Objects

This section contains the folowing topics:

Data objects contain the information that displays in reports created in Active Studio. You can design data objects to be used in certain types of views such as KPIs, gauges, or columnar reports.

The data objects you define are based on the types of data available from enterprise message sources. You must define fields in the data object. The data object contains no data when you create it. You must load or stream data into data objects using Plans.

When a data object is viewed while creating a report in Active Studio, the data object fields are displayed in alphabetical order regardless of the order that fields were added to the data object.

WARNING: Do not read or manipulate data directly in the database. All access to data must be done using Architect or the ADC API.

To define a data object:

1. Select **Data Objects** from the Architect function list.
2. Click **Create Data Object**.
3. Enter a name for the data object.
4. Enter the path to the location in the folder tree where the data object will be stored. Click Browse to use the Select a Folder dialog.
5. Optionally, enter a description of the data object.
6. If this data object will be using an **External Data Source** select the checkbox and configure the following:
   - Select an **External Data Source** from the list. External Data Sources are configured on the External Data Sources screen. See Chapter 4, "External Data Sources" for more information.
   - Enter the **External Table Name**.
7. Add fields to the data object using the **Add a field** or **Add one or more lookup fields** options.

   See "Adding Fields" on page 2-2 and "Adding Lookup Fields" on page 2-3 for more information.
8. Click **Create Data Object** when you are finished adding fields or lookup fields.

Adding Fields

To add fields to a data object:

1. In a data object you are creating or editing, click **Add a field**.
2. Specify the field name, data type, maximum size (scale for decimal fields), whether or not it is nullable, whether or not it is public, and tip text.

   If you are adding a field in a data object based on an External Data Source you must also supply the **External field name**.

   The data types include:
   - **String.** Text fields containing a sequence of characters.
   - **Integer.** Numeric fields containing whole numbers from -2,147,483,648 to 2,147,483,648.
   - **Float.** Double-precision floating point numbers.
   - **Decimal.** Integers including decimal points with scale number defined, containing up to 17 digits. The number is stored as a string which uses a character for each digit in the value.
   - **DateTime.** Dates and times combined as a real number.
   - **Boolean.** Boolean fields with true or false values.
   - **Auto-incrementing integer.** Automatically incremented integer field.

- **Timestamp.** Date time stamp generated to milliseconds. See "Adding Time Stamp Fields" on page 2-12 for more information.
- **Calculated.** Calculated field generated by an expression and saved as another data type. See "Adding Calculated Fields" on page 2-4 for more information.

Keep adding fields using **Add a field** and **Add one or more lookup fields** until all the required fields are listed. Click **Remove** to remove a field in the data object.

3. Click **Save changes**.

Adding Lookup Fields

You can add lookup fields to a data object. This performs lookups on key fields in a specified data object to return fields to the current data object. You can match multiple fields and return multiple lookup fields.

To add a lookup field to a data object:

1. In a data object you are creating or editing, click **Add one or more lookup fields**.

   The Define Lookup Field dialog displays.

2. Select the data object to use for the lookup.
3. Select the lookup fields from the data object. You can select one or more fields by holding down the Shift or Control key when selecting. Selecting multiple fields will create multiple lookup fields in the data object. These are the fields you want to return.
4. Select the field to match from the lookup data object.
5. Select the field to match from the current data object. You must have already created other fields in this data object so that you have a field to select.
6. Click **Add**.

   The matched field names are displayed in the list. You can click **Remove** to remove any matched pairs you create.

7. You can repeat steps 4 through 6 to create multiple matched fields. This is also known as a composite key.
8. Click **OK** to save your changes and close the dialog.

   The new lookup fields are added to the data object. Click **Modify Lookup Field** to make changes to a lookup field. Multiple selection of return fields is possible when defining a new lookup but not when modifying an existing one.

   You can click **Remove** to remove any lookups you create.

Note: Oracle Business Activity Monitoring supports two types of schema models: unrelated tables or Star Schemas. Any other kind of schema that does not conform to these models may result in performance issues or deadlocks. Snowflake dimensions (daisy-chained lookups) are not supported.

Supported:

```
Table 1 (with no lookups to any other tables)
Table 1 > Lookup > Table 2
```

Not supported:

```
Table 1 > Lookup > Table 2 > Lookup > Table 3
```

Adding Calculated Fields

When creating calculated fields in a data object you can use the operators and functions shown in the following tables combined with field names to produce a new field.

WARNING: If you enter a calculated field with incorrect syntax in a data object, you could lose the data object definition.

Using Operators in Calcualted Fields

Table 2–1 Describes the operators you can use to build calculated fields.

Table 2–1 Operators Used in Calculated Fields

Operator	Function
+ (plus sign)	Add
- (minus sign)	Subtract
* (asterisk)	Multiply
/ (forward slash)	Divide
% (percent sign)	Percent
() (parentheses)	Parentheses

Using Expressions in Calculated Fields

This section provides the syntax and examples for expressions you can use in a calculated field.

Avg returns the average of all values for the given field. `Avg` can accept one field parameter of type `Integer`, `Float`, or `Decimal`.

Syntax:

```
Avg(Number)
```

Example:

```
Avg(Revenue)
```

Ceiling returns the largest integer greater than or equal to the value of the specified value. `Ceiling(2.9)` returns **3** and `Ceiling(-2.3)` returns **-2**. `Ceiling` can accept one field parameter of type `Integer`, `Float`, or `Decimal` or a numeric value may be entered.

Syntax:

```
Ceiling(Number)
```

Examples:

```
Ceiling(Total)
Cieling(3.7)
```

Concat concatenates two strings into one. `Concat` can accept two field parameters of type `String`, or string values may be entered.

Syntax:

```
Concat(String,String)
```

Example:

```
Concat(Description," overstock")
```

Count returns a count of all non-null values. `Count` can accept one field parameter of any type.

Syntax:

```
Count(Field)
```

Example:

```
Count(SaleComplete)
```

CountDistinct returns a count of distinct values in a field. `CountDistinct` can accept one field parameter of any type.

Syntax:

```
CountDistinct(Field)
```

Example:

```
CountDistinct(Salesperson)
```

DateAdd adds an offset to the field value. The first parameter for `DateAdd` must be a field of type `DateTime`, and the last seven parameters maybe a field of type `Integer` or an integer value. Zeros may be used where no offset is needed.

Syntax:

```
DateAdd(DateTime, Years, Months, Days, Hours, Minutes, Seconds, Milliseconds)
```

Example:

```
DateAdd({Last Modified}, 0, 0, 7, 0, 0, 0, 0)
//adds 7 days to the Last Modified value

DateAdd({Last Modified}, 0, 0, DaysToFollowup, 0, 0, 0, 0)
```

```
//adds DaysToFolowup number of days to the Last Modified value
```

DayName returns the day name for a date. `DayName` accepts one field parameter of type `DateTime`.

Syntax:

```
DayName(DateTime)
```

Example:

```
DayName({Last Modified})
```

DayOfMonth returns the day of the month for a date, in the range 1 to 31. `DayOfMonth` accepts one field parameter of type `DateTime`.

Syntax:

```
DayOfMonth(DateTime)
```

Example:

```
DayOfMonth({Last Modified})
```

DayOfWeek returns the day of the week for a date, in the range 1 to 7. `DayOfWeek` accepts one field parameter of type `DateTime`.

Syntax:

```
DayOfWeek(DateTime)
```

Example:

```
DayOfWeek({Last Modified})
```

DayOfYear returns the day of the year for a date, in the range 1 to 366. `DayOfYear` accepts one field parameter of type `DateTime`.

Syntax:

```
DayOfWeek(DateTime)
```

Example:

```
DayOfWeek({Last Modified})
```

Floor returns the largest integer less than or equal to the value of the specified field. `Floor(2.9)` returns **2** and `Floor(-2.3)` returns **-3**. `Floor` can accept one field parameter of type `Integer`, `Float`, or `Decimal` or a numeric value may be entered.

Syntax:

```
Floor(Number)
```

Examples:

```
Floor(Sales)
Floor(46.75)
```

Hour returns the hour value in the range 0-23. `Hour` accepts one field parameter of type `DateTime`.

Syntax:

```
Hour(DateTime)
```

Example:

```
Hour({Last Modified})
```

If creates an If-Then-Else statement. `If` can accept fields, expressions, and values of any type as parameters.

Syntax:

```
If(x)
  Then(y)
  Else(z)
```

Example:

```
If(Sum(Quantity) > Max(Total))
  Then(1)
  Else(2)
```

IfNull returns a specified value, y, if the test value, x, is null. `IfNull` accepts two parameters that can be fields of any type or values of any type.

Syntax:

```
IfNull(x,y)
```

Example:

```
IfNull(Quantity, 0)
```

Length returns the length of the string. `Length` accepts one parameter that can be a field of type `String`, a string value in quotes, or an expression containing strings or fields of type `String`.

Syntax:

```
Length(String)
```

Example:

```
Length(Description)
Length("string")
Length(Concat(Description,"Description"))
```

Lower converts the string to lowercase letters. `Lower` accepts one parameter that can be a field of type `String`, a string value in quotes, or an expression containing strings or fields of type `String`.

Syntax:

```
Lower(String)
```

Example:

```
Lower(Description)
```

```
Lower("Description")
Lower(Concat(Description,"Description"))
```

Max returns the maximum value of the specified field or expression. `Max` accepts one field parameter of any type, or another valid expression.

Syntax:

```
Max(x)
```

Example:

```
Max(Quantity)
Max(Concat(Description," overstock"))
```

Min returns the minimum value of the specified field or expression. `Min` accepts one field parameter of any type, or another valid expression.

Syntax:

```
Min(x)
```

Example:

```
Min(Quantity)
Min(Concat(Description," overstock"))
```

Minute returns the minute value in the range 0-59. `Minute` accepts one field parameter of type `DateTime`.

Syntax:

```
Minute(DateTime)
```

Example:

```
Minute({Last Modified})
```

Month returns the month value for a date in the range 1-12. `Month` accepts one field parameter of type `DateTime`.

Syntax:

```
Month(DateTime)
```

Example:

```
Month({Last Modified})
```

MonthName returns the month name for a date. `MonthName` accepts one field parameter of type `DateTime`.

Syntax:

```
MonthName(DateTime)
```

Example:

```
MonthName({Last Modified})
```

Now returns the current date and time. `Now` does not accept any parameters.

Syntax:

```
Now()
```

Example:

```
DateAdd(Now(), 0, 0, 7, 0, 0, 0, 0)
```

PercentOfTotal returns the percent the value represents of the total values for the specified field. `PercentOfTotal` accepts one field parameter of type `Integer`, `Float`, or `Decimal`.

Syntax:

```
PercentOfTotal(Number)
```

Example:

```
PercentOfTotal(Quantity)
```

Power .returns one value, x, raised to the power of the second value, y. `Power` accepts two parameters that can be fields of type `Integer`, `Float`, or `Decimal`, or they can be numeric values.

Syntax:

```
Power(Number,Number)
```

Example:

```
Power(Quantity, 2)
```

Quarter returns the quarter value in the range 1-4. `Quarter` accepts one field parameter of type `DateTime`.

Syntax:

```
Quarter(DateTime)
```

Example:

```
Quarter({Last Modified})
```

Repeat repeats a string for the specified number of times. `Repeat` accepts two parameter, the first of which may be a string value or a field of type `String`, the second of which may be an integer value or a field of type `Integer`. Either parameter can use an expression that returns a string for the first parameter and an integer for the second value.

Syntax:

```
Repeat(String,Integer)
```

Example:

```
Repeat("string", 5)
Repeat(Description, 2)
Repeat(Description, Quantity)
Repeat(Concat(Description," overstock"),Quantity+2)
```

Replace returns a string, x, with all occurrences of the string, y, replaced by the string z. `Replace` accepts three field parameters of type `String`, or string values.

Syntax:

```
Replace(String,String,String)
```

Example:

```
Replace(Description, "ing", "tion")
```

Round rounds the specified value in the first paraemter to the number of decimal places specified in the second parameter, rounding up if the number in the N+1 decimal place is 5 or greater, and rounding down otherwise. `Round` accepts two parameters that can be fields of type `Integer`, `Float`, or `Decimal`, or numeric values.

Syntax:

```
Round(Number,N)
```

Example:

```
Round(Sales,2)
```

In this example, if `Sales` value is 12.345, it will be rounded to 12.35.

Second returns the second value in the range 0-59. `Second` accepts one field parameter of type `DateTime`.

Syntax:

```
Second(DateTime)
```

Example:

```
Second({Last Modified})
```

Substring returns a substring z characters long from string x, starting at position y. Substring requires three parameters, the first of which must be a string value, or a field of type `String`, and the second and third of which must be an integer or field of type `Integer`.

Syntax:

```
Substring(String,Integer,Integer)
```

Example:

```
Substring(Description, 3, 5)
```

Sum returns a summation of all values for the specified field. `Sum` accepts one field parameter of type `Integer`, `Float`, or `Decimal`.

Syntax:

```
Sum(Number)
```

Example:

```
Sum(Total)
```

Switch creates a Switch statement. `Switch` can accept fields, expressions, and values of any type as parameters.

Syntax:

```
Switch(w)
  Case(x):(y)
  Default(z)
```

Example:

TrimEnd trims the whitespace characters (space, tab, carriage return, line feed, page feed, form feed, and so on) from the end of the string. `TrimEnd` accepts one field parameter of type `String`. You can also enter an expression that returns a string value.

Syntax:

```
TrimEnd(String)
```

Example:

```
TrimEnd(Description)
TrimEnd(Concat(Description,Subcategory))
```

TrimStart trims the whitespace characters (space, tab, carriage return, line feed, page feed, form feed, and so on) from the beginning of the string. `TrimStart` accepts one field parameter of type `String`. You can also enter an expression that returns a string value.

Syntax:

```
TrimStart(String)
```

Example:

```
TrimStart(Description)
TrimStart(Concat(Description,Subcategory))
```

Upper converts a string to uppercase letters. `Upper` accepts one parameter of type `String`. You can also enter an expression that returns a string value.

Syntax:

```
Upper(String)
```

Example:

```
Upper({License Plate Number})
```

Week returns the week for a `DateTime` value, in the range 0 to 53, since there might be the beginning of a week 53, where Sunday is the first day of the week. Week 1 is the first week with a Sunday in this year.

For example, in the year 2006, January 1st is a Sunday, so there is no week 0. The year starts with week 1 and continues to week 53. Week 53 of 2006 includes only one day, which is December 31st (also a Sunday). The Monday through Saturday following this (January 1-6 of 2007) are in week 0 of 2007.

Syntax:

```
Week(DateTime)
```

Example:

```
Week({Last Modified})
```

Year returns the year value in the range 1000-9999. `Year` accepts one parameter of type `DateTime`.

Syntax:

```
Year(DateTime)
```

Example:

```
Year({Last Modified})
```

Adding Time Stamp Fields

You can create a date time stamp field generated to milliseconds by selecting the **Timestamp** data type. This column in the data object must be empty when the data object is populated by the ADC so that the time stamp data can be created.

Adding Permissions on Data Objects

You can add permissions for users and groups on data objects. When users have at least a read permissions on a data object they can choose the data object when creating reports.

To add permissions a data object:

1. Select **Data Objects** from the Architect function list.
2. Select the data object.

   The general information for the data object displays in the right frame.
3. Click **Permissions**.
4. Click **Edit Permissions**.

   Alternatively you can copy permissions from another data object. See "Copying Permissions from Other Data Objects" on page 2-13 for more information.
5. Click the **Restrict access to Data Object to certain users or groups** checkbox.

   The list of users and groups and permissions displays.
6. You can choose to display the following by clicking the radio buttons:
   - Show all users and groups
   - Show only users and groups with permissions
   - Show users only
   - Show groups only

7. You can set permissions for the entire list by clicking the buttons at the top of the list. The permissions are Read, Update, and Delete. You can set permissions for individual users or groups in the list by clicking the checkbox in the permission column that is next to the user or group name.
8. After indicating the permissions with selected checkboxes, click **Save changes**.

   A message displays to confirm that your changes are saved.
9. Click **Continue** to display the actions for the data object.

Users assigned to the Administrator role have access to all data objects. The Administrator role overrides the data object permissions.

To add a group to the list:

1. Click **Add a group to the list**.
2. Type the Windows group name in the field. The group must already exist as a domain group.
3. Click **OK**.

   The group is added to the list.

Copying Permissions from Other Data Objects

You can copy the permissions from another data object and then make additional changes to the permissions before saving.

In Architect for a data object, click Permissions and then click Copy from. Select the data object that contains the permissions to copy and click OK. You can edit the copied permissions and click Save changes.

To copy permissions from another data object:

1. Select **Data Objects** from the Architect function list.
2. Click the data object to add a security filter to.

   The general information for the data object displays in the right frame.
3. Click **Permissions**.
4. Click **Copy from**.

   The Choose Data Object dialog displays.
5. Select the data object that contains the permissions to copy and click **OK**.
6. If the data object previously had no permissions assigned, select the **Restrict access to Data Object** checkbox.
7. You can edit the copied permissions or add a group to the list.
8. Click **Save changes**.

Viewing Existing Data Objects

This section describes how to view information about data objects. It contains the following topics:

- Viewing Data Object General Information
- Viewing Data Object Layouts
- Viewing Data Object Contents

Viewing Data Object General Information

The general information of a data object displays the owner, when it was created, when it was last modified, and a row count.

To view the general information of a data object:

- Click the data object in the list.

  If you are already viewing the layout or contents of a data object, click *General*.

  The general information displays in the right frame. It contains the following information:

  - **Created**. Date and time the data object was created.
  - **Last modified**. Date and time the data object was last modified.
  - **Row count**. Numbers of rows of data in the data object.
  - **Location**.
  - **Type**.
  - **Data Object ID**. The ID used to identify the data object. This is based on the name although the ID is used throughout the system so that you can edit the name without affecting any dependencies.

  Note: If the row count is over 500,000 rows, an approximate row count is displayed in the General information for increased performance purposes. The approximate row count is accurate within 5-10% of the actual count. If you want to view an exact row count instead of the approximation, click Show exact count. The exact count is displayed. This could take a few minutes if the data object has millions of rows.

Viewing Data Object Layouts

The layout describes the fields in a data object. The fields are described by name, field ID, data type, maximum length allowed, scale, nullable, public, calculated, text tip, and lookup.

To view the layout of a data object:

1. Select the data object.
2. The general information displays in the right frame.
3. Click **Layout**.

   The layout information displays in the right frame. It contains the following information:

   - **Field name**. Name of the field
   - **Field ID**. Generated by the system
   - **External name**. External field name from the External Data Source (only appears in data objects based on External Data Sources)
   - **Field type**. Data type of the field
   - **Max length**. Maximum number of characters allowed in field value
   - **Scale**. Number of digits on the right side of the decimal point

- **Nullable**. Whether or not the data type can contain null values
- **Public**. This setting determines whether or not the field will be available in Active Studio to use in a report. If the box is unchecked, the field will not appear in Active Studio. This is useful for including fields for calculations in data objects that should not appear in reports.
- **Lookup**. Displays specifics of a lookup field
- **Calculated**. Displays the expression of a calculated field
- **Tip Text**. Helpful information about the field

Viewing Data Object Contents

You can view the rows of data stored in a data object by viewing the data object contents. You can also edit the contents of the data object.

To view the contents of a data object:

1. Select the data object.

   The general information displays in the right frame.

2. Click **Contents**.

   The first 50 rows of the data object display in the right frame.

3. Click **Next**, **Previous**, **First**, and **Last** to navigate to other rows of data displayed 50 at a time.

   Rows are listed with a Row ID column. Displaying only Row ID provides faster paging for large data objects. Row IDs are assigned once in each row and maintain a continuous row count when you clear and reload a data object.

   You can click **Show row numbers** to display an additional column containing a current row count starting at 1. Click **No row numbers** to hide the row count column again.

4. Click **Refresh** to get the latest available contents.

Using Data Object Folders

This section contains the following topics:

You can organize data objects by creating folders and subfolders for them. When you create a folder for data objects, you can assign permissions by associating users and actions with the folder.

Creating Folders

You can create new folders for organizing data objects. Then you can move or create data objects into separate folders for different purposes or users. After creating folders,

you can set folder permissions to limit which users can view the data objects it contains.

To create a new folder:

1. Select **Data Objects** from the Architect function list.

   The current data object folders display in a tree hierarchy.

2. Click **Create subfolder**.

   A field for naming the new folder displays.

3. Enter a name for the folder and click **Create folder**.

   The folder is created as a subfolder under the Data Objects folder and a message displays confirming that the new folder was created.

4. Click **Continue** to view the folder.

Working with Folders

To open a folder:

1. Expand the tree of folders by clicking the + (plus sign) next to the Data Objects folder.

   The System subfolders contain data objects for running Oracle Business Activity Monitoring. For more information about these data objects see "System Data Objects" on page 2-23.

2. Click the link next to a folder to open it.

   The folder is opened, and the data objects in the folder are shown in the list underneath the folder tree. The general properties for the folder display in the right frame and the following links apply to the current folder:

   View. Displays the general properties of this folder such as name, date created, date last modified, user who last modified it. View is selected when you first click a folder.

   Create subfolder. Creates another folder within the selected folder.

   Delete. Removes the selected folder and all the data objects it contains.

   Rename. Changes the folder name.

   Move. Moves this folder to a new location, for example, as a subfolder under another folder.

   Permissions. Sets permissions on this folder.

   Create Data Object. Creates a data object in this folder.

Setting Folder Permissions

When you create a folder, you can set permissions on it so that other users can access the data objects contained in the folder.

To set permissions on a folder:

1. In the Data Objects folder, select the folder to change permissions on.
2. Click **Permissions**.
3. Click **Edit permissions**.

4. Select the **Restrict access to Data Object to certain users or groups** checkbox.

   The list of users and groups and permissions displays.

5. You can choose to display the following by selecting one of the radio buttons:
   - Show all users and groups
   - Show only users and groups with permissions
   - Show users only
   - Show groups only

6. You can set permissions for the entire list by clicking the column headers at the top of the list. The permissions are Read, Update, and Delete. You can set permissions for individual users or groups in the list by selecting the checkbox in the permission column that is next to the user or group name.

7. After indicating the permissions with selected checkboxes, click **Save changes**.

   A message displays to confirm that your changes are saved.

8. Click **Continue** to display the actions for the data object.

To add a group to the list:

1. Click the **Add a group to the list** link.
2. Type the Windows group name in the field. The group must already exist as a domain group.
3. Click **OK**.

   The group is added to the list.

Moving Folders

To move a folder:

1. Select the folder to move.
2. Click **Move**.
3. Click **Browse** to select the new location for the folder.
4. Click **OK** to close the dialog.
5. Click **Move folder**.

   The folder is moved.

6. Click **Continue**.

Renaming Folders

To rename a folder:

1. Select the folder to rename.
2. Click **Rename**.
3. Enter a new name and click **Rename folder**.

   The folder is renamed. You must assign unique folder names within a containing folder.

4. Click **Continue**.

Deleting Folders

When you delete a folder, you also delete all of the data objects in the folder.

To delete a folder:

1. Select the folder to delete.
2. Click **Delete**.

   A message displays to confirm that you want to delete the folder and all of its contents.
3. Click **OK**.

   The folder is deleted.
4. Click **Continue**.

Adding Security Filters

You can add security filters to data objects so that only specific users can view specific rows in the data object. This can be useful when working with data objects that contain sensitive or confidential information that is not intended for all report designers or report viewers.

Security filters perform a lookup using another data object, referred to as a security data object, containing user names or group names. Before you can add a security filter, you must create a security data object containing the user names or group names and the value in the column to allow for each user name or each group name. Security data objects cannot contain null values.

To add a security filter to a data object:

1. Select **Data Objects** from the Architect function list.
2. Select the data object to add a security filter to.

   The general information for the data object displays in the right frame.
3. Select **Security Filters**.

   If the data object includes security filters, the filter name and a plus sign displays so that you can expand and view the information.
4. Click **Add filter**.

   The fields for defining the security filter display.
5. Enter the following information:

   Name of this Security Filter. Type a name for this filter.

   Security Data Object. Select the name of the security data object containing the mapped columns.

   Type of identification. Select either By user or By group from the dropdown list. The security data object must already include either domain or local users or groups mapped to values in the identification column.

   Identification column in Security Data Object. Select the name of the column for containing user names or group names.

   Match column in Security Data Object. Select the column to match in the security data object.

Match column in this Data Object. Select the name of the column to match in this data object.

6. Click **Add**.

For example, to add a security filter to the following data object, you need a security data object containing Region information to perform the security lookup.

Example data object:

User	Region	Sales
John Smith	1	$55,000
Bob Wright	1	$43,000
Betty Reid	2	$38,000

Security data object:

Login	Region ID
DomainName\jsmith	1
DomainName\jsmith	2
DomainName\bwright	1
DomainName\breid	2

When the **bwright** account views a report that accesses the data object with a security filter applied based on Region ID and Region, it is only able to access information for jsmith and bwright. It is not able to view the breid information because it is not able to view data for the same region. On the other hand, the jsmith account is set up to view data in both region 1 and 2.

Copying Security Filters from Other Data Objects

You can copy security filters from another data object and apply them to the data object you are editing.

To copy security filters from another data object:

1. Select **Data Objects** from the Architect function list.
2. Select the data object to add a security filter to.

   The general information for the data object displays in the right frame.
3. Select **Security Filters**.

   If the data object includes security filters, the filter name and a plus sign displays so that you can expand and view the information.
4. Click **Copy from**.

   The Choose Data Object dialog displays.
5. Select the data object that contains the security filters to copy and click **OK**.
6. You can make changes to the security filters by viewing the filter details and clicking **Edit**.
7. Click **Save**.

Adding Dimensions

In Architect, you can add dimensions to data objects to define drill paths for charts in Active Studio. Dimensions contain fields in a hierarchy. When a hierarchy is selected in chart, the end user can drill down and up the hierarchy of information. When a user drills down in a chart, they can view data at more and more detailed levels within a specific value.

Hierarchies are an attribute of a dimension in a data object. Multiple dimensions can be created in each data object. Each field in a data object can belong to one dimension only. You can create and edit multiple, independent hierarchies.

To use hierarchies as drill paths in charts, the report designer must select the hierarchy to use as the drill path. To create a dimension, you must select multiple fields to save as a dimension. Then you organize the fields into a hierarchy.

An example dimension and hierarchy:

Dimension	**Hierarchy**
Sales	Category
	Brand
	Description

To add a dimension and hierarchy:

1. Select **Data Objects** from the Architect function list.
2. Select the data object to add a dimension to.

   The general information for the data object displays in the right frame.
3. Select **Dimensions**.
4. Click **Add a new dimension**.
5. Enter a dimension name.
6. Enter a description for the dimension.
7. Select the field names that you want to include in the dimension. An example is Sales, Category, Brand, and Description.

   The field names are moved from the Data Objects Fields list to the Dimension Fields list to show that they are selected.
8. Click **Save**.
9. Click **Continue**.

   The new dimension is listed. You must still define a hierarchy for the fields.
10. Click **Add new hierarchy**.
11. Enter a hierarchy name.
12. Enter a description for the hierarchy.
13. Select the field names that you want to define as attributes for the dimension. An example is Sales remains in the Dimension Field list, and you click Category, Brand, and Description to arrange them in a general to more specific order. The order that you click the fields is the order that they are listed in the Hierarchy Field list. Arrange the more general grouping field at the top of the Hierarchy Fields list and the most granular field at the bottom of the Hierarchy Fields list.

14. Click **Save**.
15. Click **Continue**.

    The new hierarchy is listed. You can edit or remove hierarchies and dimensions by clicking the links. You can also continue defining multiple hierarchies for the dimension or add new dimensions to the data object.

Time Dimensions

If your dimension contains a time date data type field, you can select the time levels to include in the hierarchy.

To select time levels:

1. In a dimension containing a time date data type field, add a hierarchy.
2. Select the time date data type field. If you are editing existing time levels, click **Edit Time Levels**.

   The Time Levels Definition dialog opens.
3. Click the levels to include in the hierarchy. The levels include:
   - **Year.** Year in a four digit number.
   - **Quarter.** Quarter of four quarters starting with quarter one representing January, February, March.
   - **Month.** Months one through 12, starting with January.
   - **Week of the Year.** Numbers for each week starting with January 1st.
   - **Day of the Year.** Numbers for each day of the year starting with January 1st.
   - **Day of the Month.** Numbers for each day of the month.
   - **Day of the Week.** Numbers for each day of the week, starting from Sunday to Saturday.
   - **Hour.** Numbers from one to twenty four.
   - **Minute.** Numbers from one to 60.
   - **Second.** Numbers from one to 60.
4. Click **OK** to close the dialog.

Renaming and Moving Data Objects

You can rename and move a data object without editing or clearing the data object. If you only want to change the data object name or description, use the Rename option.

To rename a data object:

1. Select **Data Objects** from the Architect function list.
2. Select the data object to rename.

   The general information for the data object displays in the right frame.
3. Select **Rename/Move**.
4. Enter the new name, tip text, and description for the data object.
5. Click **Save changes**.

To move a data object:

1. Select **Data Objects** from the list.
2. Select the data object to rename.

   The general information for the data object displays in the right frame.
3. Select **Rename/Move**.
4. Click **Browse** to enter the new location for the data object.
5. Click **Save changes**.

Adding Indexes

Indexes improve performance for large data objects containing many rows. Without any indexes, accessing data requires scanning all rows in a data object. Scans are inefficient for very large data objects. Indexes can help find rows with a specific value in a column. If the data object has an index for the fields requested, the information is found without having to look at all the data. Indexes are most useful for locating rows by values in columns, aggregating data, and sorting data.

You can add indexes to data objects by selecting fields to be indexed as you are creating a data object. You cannot add indexes after loading the data object unless you edit and clear the data object.

To add an index:

1. Select **Data Objects** from the Architect function list.
2. Click the data object to add an index to.
3. Select **Indexes**.
4. Click **Add Index**.

   The Add Index dialog opens.
5. Enter a **Name** and **Description** for the index
6. Add as many fields as needed to create an index for the table.

   Click a field in the list on the right to remove the field from the index.
7. Click **OK**.

   The index is added and is named after the fields it contains. You can create more than one index. To remove an index you created, click **Remove Index** next to the Index name.

Clearing Data Objects

You might want to clear a data object before loading it. Clearing a data object removes the current contents without deleting the data object from the ADC.

To clear a data object:

1. Select **Data Objects** from the Architect function list.
2. Select the data object to clear.

   The general information for the data object displays in the right frame.
3. Click **Clear**.

Deleting Data Objects

When deleting data objects, make sure that no Plans are accessing the data objects. You should also edit or delete reports and alerts referring to the data object that you want to delete.

To delete a data object:

1. Select **Data Objects** from the Architect function list.
2. Click the data object to delete.

   The general information for the data object displays in the right frame.
3. Click **Delete**.

System Data Objects

The System data objects folder contains data objects used to run Oracle Business Activity Management. You should not make any changes to these data objects, except for the following:

- **Custom Parameters** lets you define global parameters for Action Buttons.
- **Action Form Templates** lets you define HTML forms for Action Form views.
- **Chart Themes** lets you add or change color themes for view formatting.
- **Matrix Themes** lets you add or change color themes for the Matrix view.
- **Util Templates** lets you define templates that are used by Action Form views to transform content.

For more information about matrix and color themes, Action Buttons, and Action Forms see Oracle Business Activity Monitoring Active Studio User's Guide.

3

Enterprise Message Sources

This chapter contain the information needed to create and manage Enterprise Message Sources.

This chapter contains the following topics:

Introducing Enterprise Message Sources

Enterprise message sources are providers of the real-time information flowing through the enterprise to the Active Data Cache. Each enterprise message source connects to a specific message queue and the information is delivered into a data object in the Active Data Cache. Data objects are used in reports and the data is viewed by users.

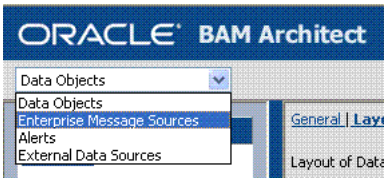
After an enterprise message source is defined, it can be included in a Plan for loading the information to a data object in the Active Data Cache.

Listing Enterprise Message Sources

To view the existing enterprise message sources:

- Select **Enterprise Message Sources** from the Architect function list.

Figure 3–1 Architect Function List



Defining Enterprise Message Sources

This section contains the following topics:

- Specifying Settings for Oracle (AS JMS and OJMS)
- Specifying Settings for BEA WebLogic Server
- Specifying Settings for IBM WebSphere MQ
- Specifying Settings for Microsoft MSMQ
- Specifying Settings for See Beyond JMS Intelligent Queue
- Specifying Settings for Sonic MQ
- Specifying Settings for Tibco Rendezvous
- Specifying Settings for WebMethods
- Using Advanced Formatting
- XSL Processing and Example Code

When you define an enterprise message source, you specify all fields in the messages to be received. Some messaging systems have a variable number of user-defined fields, while systems have a fixed number of fields.

For any string type field, you can apply formatting to that field to break apart the contents of the field into separate, individual fields to be sent through the Data Flow of the Plan. This is useful for messaging systems where you cannot create user-defined fields and the entire message body is received as one large field. The formatting specifications allow you to specify the path to a location in the XML tree, and then extract the attributes or tags as fields.

Before defining an enterprise message source, you must be familiar with the third party application providing the messages so that you can specify the message source connection details in Architect.

To define an enterprise message source:

1. Select **Enterprise Message Sources** from the Architect function list.
2. Click **Create**.
3. Select the type of enterprise message source from the list in the right frame and click **Select**.
4. Enter a name for the enterprise message source.
5. See one of the following sections for information about settings:
   - "Specifying Settings for Oracle (AS JMS and OJMS)" on page 3-3
   - "Specifying Settings for BEA WebLogic Server" on page 3-3
   - "Specifying Settings for IBM WebSphere MQ" on page 3-3
   - "Specifying Settings for Microsoft MSMQ" on page 3-4
   - "Specifying Settings for See Beyond JMS Intelligent Queue" on page 3-4
   - "Specifying Settings for Sonic MQ" on page 3-5
   - "Specifying Settings for Tibco Rendezvous" on page 3-5
   - "Specifying Settings for WebMethods" on page 3-5
6. For all types, you specify the dataflow name, data type, maximum size, and formatting included. Formatting can be set to XML with column values contained in tags, or in attributes, or no formatting. For advanced formatting using XSL transformations, see "Using Advanced Formatting" on page 3-6.

Specifying Settings for Oracle (AS JMS and OJMS)

Settings for Oracle (AS JMS and OJMS) include:

- **Initial Context Factory.**

  For 10.1.3: oracle.j2ee.rmi.RMIInitialContextFactory

  For 10.1.2: com.evermind.server.rmi.RMIInitialContextFactory

- **JNDI Service Provider URL.** For standalone use oc4j: ormi://*machine_name*/ where *machine_name* is the OC4J installed host, and for production use Application Server: opmn:ormi://<*machine_name*>:<*opmn request port*>:home where *machine_name* is the application server installed host, *opmn request port* could be found from looking at opmn.xml in <*App server install location*>/opmn/conf
- **TopicConnectionFactory Name.** java:comp/resource/BAMTTT/TopicConnectionFactories/bam_ttq_tab
- **Topic Name.** java:comp/resource/BAMTTT/Topics/bam_ttq
- **JMS Message Type.** TextMessage.
- **Durable Subscriber Name.** BAMFilteredSubscription
- **Message Selector.** Message selector, for example, BAMFilter='true'
- **Client ID.** ClientID

Specifying Settings for BEA WebLogic Server

Settings for BEA WebLogic Server include:

- **Initial Context Factory.** Context factory to use, for example, weblogic.jndi.WLInitialContextFactory
- **JNDI Service Provider URL.** File path or URL, for example, t3://localhost:6733
- **TopicConnectionFactory Name.** Connection factory of the topic, for example, JMSConnectionFactory
- **Topic Name.** Topic name, for example, JMSTopic
- **JMS Message Type.** Select the message type such as BytesMessage, MapMessage, ObjectMessage, StreamMessage, TextMessage.
- **Durable Subscriber Name.** Name of the subscriber, for example, BAMFilteredSubscription.
- **Message Selector.** Message selector, for example, BAMFilter='true'
- **Client ID.** Client ID, for example, BAMFilteredClientID

Specifying Settings for IBM WebSphere MQ

Settings for IBM WebSphere MQ include:

- **Initial Context Factory.** Context factory to use, for example, com.sun.jndi.fscontext.RefFSContextFactory
- **JNDI Service Provider URL.** File path or URL, for example, file:/C:/JNDI-Directory
- **TopicConnectionFactory Name.** Connection factory of the topic, for example, ivtTCF
- **Topic Name.** Topic name, for example, ivtT

- **JMS Message Type.** Select the message type such as BytesMessage, MapMessage, ObjectMessage, StreamMessage, TextMessage.
- **Durable Subscriber Name.** Name of the subscriber, for example, BAMFilteredSubscription.
- **Message Selector.** Message selector, for example, BAMFilter='true'
- **Client ID.** Client ID, for example, BAMFilteredClientID

Specifying Settings for Microsoft MSMQ

Settings for Microsoft MSMQ include:

- **Queue name.** Type the MSMQ queue name, for example, horus\retail_queue.
- **Message body character encoding.** Select Unicode UTF-8, UTF-16, UTF-7, or ASCII.
- **Look for byte order marks at start of message body.** Select Yes or No.

The field names are already defined as ID, Arrival Time, Label, and Data. If you select a type of formatting, you must provide a path such as row, the tag or attribute name for each item, the name in the data flow, and maximum size.

You can use database triggers and a web service to send data to the ADC using Microsoft MSMQ. For more information, refer to:

http://<host>:<http_port>/oraclebam/Services/EnterpriseLink/EnterpriseLink.asmx

When using a trigger, the directory containing the service requires anonymous access or basic authentication to be set in IIS.

The following example includes a portion of an Oracle database trigger using this service:

```
utl_http.request(http://host/oraclebam/services/enterpriselink/
 enterpriselink.asmx/SendToMSMQ?strQueueName=private$\OracleTriggerTest&strData=
 '<?xmlversion="1.0"?><row><fname>'||fname||'</fname><lname>'

||lname||'</lname><title>'||title||'</title><location>'||location||
 '</location></row>'&strLabel='' HTTP/1.1);
End;
```

Internet Explorer has a maximum URL length of 2,048 characters which also applies to POST and GET request URLs. If you use the GET method, the limit includes the number of characters in the actual path. POST is not limited by the URL size when submitting name-value pairs since they are transferred in the header and not the URL.

Specifying Settings for See Beyond JMS Intelligent Queue

Settings for See Beyond JMS Intelligent Queue include:

- **Initial Context Factory.** Context factory to use, for example, com.sun.jndi.fscontext.RefFSContextFactory
- **JNDI Service Provider URL.** File path or URL, for example, file:/C:/JNDI-Directory
- **TopicConnectionFactory Name.** Connection factory of the topic, for example, ivtTCF
- **Topic Name.** Topic name, for example, ivtT

- **JMS Message Type.** Select the message type such as BytesMessage, MapMessage, ObjectMessage, StreamMessage, TextMessage.
- **Durable Subscriber Name.** Name of the subscriber, for example, BAMFilteredSubscription.
- **Message Selector.** Message selector, for example, BAMFilter='true'
- **Client ID.** Client ID, for example, BAMFilteredClientID

Specifying Settings for Sonic MQ

Settings for Sonic MQ include:

- **Initial Context Factory.** Context factory to use, for example, com.sun.jndi.fscontext.RefFSContextFactory
- **JNDI Service Provider URL.** File path or URL, for example, file:/C:/JNDI-Directory_SonicMQ
- **TopicConnectionFactory Name.** Connection factory of the topic, for example, sonicTCF
- **Topic Name.** Topic name, for example, sonicT
- **JMS Message Type.** Select the message type such as BytesMessage, MapMessage, ObjectMessage, StreamMessage, TextMessage.
- **Durable Subscriber Name.** Name of the subscriber, for example, BAMFilteredSubscription.
- **Message Selector.** Message selector, for example, BAMFilter='true'
- **Client ID.** Client ID, for example, BAMFilteredClientID

Specifying Settings for Tibco Rendezvous

Settings for Tibco Rendezvous include:

- **Initial Context Factory.** Context factory to use, for example, com.tibco.tibjms.naming.TibjmsInitialContextFactory
- **JNDI Service Provider URL.** File path or URL, for example, tibjmsnaming://localhost:7222
- **TopicConnectionFactory Name.** Connection factory of the topic, for example, TopicCF
- **Topic Name.** Topic name, for example, Topic
- **JMS Message Type.** Select the message type such as BytesMessage, MapMessage, ObjectMessage, StreamMessage, TextMessage.
- **Durable Subscriber Name.** Name of the subscriber, for example, BAMFilteredSubscription.
- **Message Selector.** Message selector, for example, BAMFilter='true'
- **Client ID.** Client ID, for example, BAMFilteredClientID

Specifying Settings for WebMethods

Settings for WebMethods include:

- **Initial Context Factory.** Context factory to use, for example, com.sun.jndi.fscontext.RefFSContextFactory
- **JNDI Service Provider URL.** File path or URL, for example, file:/C:/JNDI-Directory
- **TopicConnectionFactory Name.** Connection factory of the topic, for example, ivtTCF
- **Topic Name.** Topic name, for example, ivtT
- **JMS Message Type.** Select the message type such as BytesMessage, MapMessage, ObjectMessage, StreamMessage, TextMessage.
- **Durable Subscriber Name.** Name of the subscriber, for example, BAMFilteredSubscription.
- **Message Selector.** Message selector, for example, BAMFilter='true'
- **Client ID.** Client ID, for example, BAMFilteredClientID

Using Advanced Formatting

The Advanced formatting options allow an enterprise message source to contain a user-supplied XSL Transformation (XSLT) for each formatted field in the message.

Uses for XSL transformations include:

- Handling of hierarchical data. The Data Flow does not handle hierarchical data. The XSL transformation can flatten the received XML into a single record with repeating fields.
- Handling of message queues that contain messages of more than one type in a single queue. The Data Flow requires that all records from the Message Receiver be of the same schema. The enterprise message source output can be defined as a combined superset of the message schemas that will be received, and the XSL transformation can identify each message type and map it to the superset schema as appropriate.
- Handling of XML that, while not expressing hierarchical data, does contain needed data at more than one level in the XML. EMS formatting can only read from one level with the XML. The XSL transformation can identify the data needed at various levels in the input XML and output it all in new XML that contains all of the data combined at one level.
- Handling changes to message formats without affecting Plans. If a complex message-processing Plan exists and the format of the messages, or the data in them, is changed slightly, the XSL transformation could compensate for this change so that changes to the Plan are not required.

To specify an XSL transformation:

1. In an enterprise message source that you are defining or editing, select one of the XML formatting options in the Formatting column.
2. Click **Advanced formatting options**.

   The Advanced Formatting dialog displays.
3. Type or paste the XSL code for the transformation for the XML in this field. You might want to write the XSL in another editing tool and then copy and paste the code into this dialog.

4. In the **Sample XML to transform** field, type sample XML to test the transformation against. The sample XML is not saved in this dialog and will not be displayed if you close and open this dialog.
5. Click **Verify transformation syntax** to check the XSL syntax.
6. Click **Test transformation on sample XML** to test your transformation.

   The results are displayed in the field underneath the links. If any errors are found in the XSL syntax, the sample XML syntax, or during the transformation, the error text is shown in this field.

XSL Processing and Example Code

XSL Processing in an Example Enterprise Message Source

The Enterprise Message Receiver Transform receives messages from the queue, and converts them to a traditional row and column format to send into the Data Flow.

The records in the Data Flow must be flat records because hierarchical data is not supported. The Message Receiver can generate multiple Data Flow records for each received message, but all records must have the same schema.

In the Enterprise Message Source definition, a formatting specification can be provided for any fields in the message of type String to parse XML inside the string field. Typically for MSMQ, the Body field of the message contains the XML for the message.

The formatting specification indicates how the received XML is to be converted into one or more records. It assumes that the fields of data are contained within a particular single node-type within the XML, either as attributes of that node or as tags directly contained in that node. Although this formatting can handle multiple instances of the node, it cannot handle multiple node types, nor can it handle hierarchies in the XML.

The following example shows how a simple message might look:

```
<order>
  <item>
    <description>Fuel pump</description>
    <price>128.95</price>
    <quantity>1</quantity>
  </item>
  <item>
    <description>Fuel filter</description>
    <price>12.95</price>
    <quantity>1</quantity>
  </item>
</order>
```

The generated records for the preceding message might look like the following:

Description	Price	Quantity
Fuel pump	128.95	1
Fuel filter	12.95	1

Handling Complex Messages

To handle complex XML, or do formatting or parsing not supported by the Enterprise Message Receiver, the Enterprise Message Source can contain an XSL-T specification to "preprocess" the received XML before it is passed to the Message Receiver formatting. This transformation can be complex, and can contain any code supported by the Microsoft .NET XSL-T Processor, including scripting. Any field in the received message of type String can have an XSL-T preprocessor. In the case of MSMQ, this is almost always the Body field of the message.

The XSL-T code can be used for any purpose required. For this example, it is used to translate hierarchical message XML into simpler XML that can be handled by the formatting. Typically, this would mean flattening of the hierarchical message into multiple records, where the higher-level field values repeat.

Phoenix Debt Order Messages

The XML in these messages expresses a hierarchy. Each message contains a root PhoenixDebtOrder tag, which contains one or more PhoenixDebtOrderProduct tags, which each contain one or more PhoenixDebtOrderIOI tags:

```
<PhoenixDebtOrder attr1 attr2>
  <PhoenixDebtOrderProduct attr3 attr4>
    <PhoenixDebtOrderIOI attr5 attr6>
```

The values at each level are contained in attributes on the tags.

The purpose of the XSL transformation is to convert the preceding hierarchy into a flat structure containing a node for each PhoenixDebtOrderIOI node, with the values for the parent tags repeated in each instance:

```
<PhoenixDebtOrder>
  <Flat attr1 attr2 attr3 attr4 attr5 attr6>
```

The XSL assumes that the attribute names are different at each level, or if they are not different, that they represent the same information and do not need to be repeated.

The XSL does a generic copy of all of the attributes at each level, and no specific attribute names are referenced. If attributes are added or removed at any level, it is not necessary to change the XSL code, although in such cases, it is necessary to change the formatting in the Enterprise Message Source.

Phoenix Debt Order Example

The following text shows a sample Phoenix Debt Order message:

```
<PhoenixDebtOrder ord_id="1847494630" rgn_id="0"
 last_modified_dt="2003-06-23T16:56:35.900" ae_up_id="0" deleted_ind="0"
 iss_id="1847492081" brk_id="1877504017" inst_inv_id="0" swap_ind="0"
 identity_nm="">
<PhoenixDebtOrderProduct ord_id="1847494630" prd_id="1847494611"
 price_basis="spread" ioi_ccy_id="1847483827" canceled_ind="0" inst_alloc_qty=""
 ret_alloc_qty="">
<PhoenixDebtOrderIOI ord_ioi_id="1847558620" ord_id="1847494630"
 ioi_prd_id="1847494611" ioi_size="4770000000" ioi_px="14.123512"/>
<PhoenixDebtOrderIOI ord_ioi_id="1847558621" ord_id="1847494630"
 ioi_prd_id="1847494611" ioi_size="5540000000" ioi_px="15.252500"/>
<PhoenixDebtOrderIOI ord_ioi_id="1847558619" ord_id="1847494630"
 ioi_prd_id="1847494611" ioi_size="3330000000" ioi_px="12.500000"/>
</PhoenixDebtOrderProduct>
<PhoenixDebtOrderProduct ord_id="1847494630" prd_id="1847494612"
 price_basis="spread" ioi_ccy_id="1847483827" canceled_ind="0" inst_alloc_qty=""
 ret_alloc_qty="">
<PhoenixDebtOrderIOI ord_ioi_id="1847558620" ord_id="1847494630"
```

```
 ioi_prd_id="1847494612" ioi_size="5880000000" ioi_px="14.124512"/>
<PhoenixDebtOrderIOI ord_ioi_id="1847558621" ord_id="1847494630"
 ioi_prd_id="1847494612" ioi_size="4430000000" ioi_px="12.252500"/>
</PhoenixDebtOrderProduct>
</PhoenixDebtOrder>
```

The following text shows the transformed XML produced for the previous example:

```
<PhoenixDebtOrder>
<Flat ord_id="1847494630" rgn_id="0"
 last_modified_dt="2003-06-23T16:56:35.900" ae_up_id="0" deleted_ind="0"
 iss_id="1847492081" brk_id="1877504017" inst_inv_id="0" swap_ind="0"
 identity_nm="" prd_id="1847494611" price_basis="spread" ioi_ccy_id="1847483827"
 canceled_ind="0" inst_alloc_qty="" ret_alloc_qty="" ord_ioi_id="1847558620"
 ioi_prd_id="1847494611" ioi_size="4770000000" ioi_px="14.123512"/>
<Flat ord_id="1847494630" rgn_id="0"
 last_modified_dt="2003-06-23T16:56:35.900" ae_up_id="0" deleted_ind="0"
 iss_id="1847492081" brk_id="1877504017" inst_inv_id="0" swap_ind="0"
 identity_nm="" prd_id="1847494611" price_basis="spread" ioi_ccy_id="1847483827"
 canceled_ind="0" inst_alloc_qty="" ret_alloc_qty="" ord_ioi_id="1847558621"
 ioi_prd_id="1847494611" ioi_size="5540000000" ioi_px="15.252500"/>
<Flat ord_id="1847494630" rgn_id="0"
 last_modified_dt="2003-06-23T16:56:35.900" ae_up_id="0" deleted_ind="0"
 iss_id="1847492081" brk_id="1877504017" inst_inv_id="0" swap_ind="0"
 identity_nm="" prd_id="1847494611" price_basis="spread" ioi_ccy_id="1847483827"
 canceled_ind="0" inst_alloc_qty="" ret_alloc_qty="" ord_ioi_id="1847558619"
 ioi_prd_id="1847494611" ioi_size="3330000000" ioi_px="12.500000"/>
<Flat ord_id="1847494630" rgn_id="0"
 last_modified_dt="2003-06-23T16:56:35.900" ae_up_id="0" deleted_ind="0"
 iss_id="1847492081" brk_id="1877504017" inst_inv_id="0" swap_ind="0"
 identity_nm="" prd_id="1847494612" price_basis="spread" ioi_ccy_id="1847483827"
 canceled_ind="0" inst_alloc_qty="" ret_alloc_qty="" ord_ioi_id="1847558620"
 ioi_prd_id="1847494612" ioi_size="5880000000" ioi_px="14.124512"/>
<Flat ord_id="1847494630" rgn_id="0"
 last_modified_dt="2003-06-23T16:56:35.900" ae_up_id="0" deleted_ind="0"
 iss_id="1847492081" brk_id="1877504017" inst_inv_id="0" swap_ind="0"
 identity_nm="" prd_id="1847494611" price_basis="spread" ioi_ccy_id="1847483827"
 canceled_ind="0" inst_alloc_qty="" ret_alloc_qty="" ord_ioi_id="1847558621"
 ioi_prd_id="1847494612" ioi_size="4430000000" ioi_px="12.252500"/>
</PhoenixDebtOrder>
```

Sequence Numbers

A single parameter is passed to the XSL transformation from the Message Receiver Transform, called "SequenceNumber". This is an integer that starts at one and increments for each message received. This number can be used to identify the set of records generated for a single original message, or for any other purpose desired by the XSL code.

In this case, the XSL code does use this number to identify the set of records that are generated for a single message, but with a twist.

The goal is for an iterating SubPlan within the Data Flow to iterate once for each set of records produced by a single message, using the Iterate each time a key field changes value option for the SubPlan. The XSL passes the sequence number in the Data Flow records as a field named "seq_no", and this is the key field used for SubPlan iteration.

The SubPlan does not actually iterate until the key value changes. If a gap in time exists between messages, completion of processing for the prior message is delayed until the next message is received since changes are not committed until the SubPlan iterates through using the "Group Transaction" feature. To solve this problem, the XSL

always generates an extra "dummy" record, or tag, at the end of the set of records for each message. Since the sequence numbers are incremented by one, the XSL can know what the next sequence number will be, unlike with an MSMQ message ID. It can use the sequence number that the next message will use. The Data Flow contains logic to recognize and ignore the dummy records, which are denoted by an "ord_id" attribute value of "dummy".

XSLT Code

Example 3–1 shows the XSLT code contained within the Enterprise Message Source.

Example 3–1 Enterprise Message Source XSLT

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:param name="SequenceNumber"/>
  <xsl:output method="xml" omit-xml-declaration="yes"/>
  <xsl:template match="/">
    <PhoenixDebtOrder>
      <xsl:choose>
        <xsl:when test="PhoenixDebtOrder">
          <xsl:apply-templates select="PhoenixDebtOrder"/>
        </xsl:when>
        <xsl:otherwise>
          <!-- There is no root PhoenixDebtOrder tag -->
          <!-- Generate a tag with no attribute values -->
          <Flat>
            <xsl:attribute name="seq_no">
            <xsl:value-of select="$SequenceNumber"/></xsl:attribute>
          </Flat>
        </xsl:otherwise>
      </xsl:choose>
      <!-- Dummy row, always generated at the end, with the sequence number
           that the next set of rows will have. Used to cause the SubPlan
           to iterate immediately after the receipt of all rows for this order-->
      <Flat ord_id="Dummy">
        <xsl:attribute name="seq_no">
        <xsl:value-of select="$SequenceNumber + 1"/></xsl:attribute>
      </Flat>
    </PhoenixDebtOrder>
  </xsl:template>
  <xsl:template match="PhoenixDebtOrder">
    <xsl:choose>
      <xsl:when test="PhoenixDebtOrderProduct">
        <xsl:apply-templates select="PhoenixDebtOrderProduct"/>
      </xsl:when>
      <xsl:otherwise>
        <!-- There are no products -->
        <!-- Generate a tag with only the attributes from the root
             PhoenixDebtOrder tag -->
        <Flat>
          <xsl:for-each select="@*">
            <xsl:copy-of select="."/>
          </xsl:for-each>
          <xsl:attribute name="seq_no">
          <xsl:value-of select="$SequenceNumber"/></xsl:attribute>
        </Flat>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
```

```
  <xsl:template match="PhoenixDebtOrderProduct">
    <xsl:choose>
      <xsl:when test="PhoenixDebtOrderIOI">
        <xsl:apply-templates select="PhoenixDebtOrderIOI"/>
      </xsl:when>
      <xsl:otherwise>
        <!-- This product has no IOIs -->
        <!-- Generate a tag with only the attributes from the root
             PhoenixDebtOrder tag and the parent PhoenixDebtOrderProductTag -->
        <Flat>
          <xsl:for-each select="../@*">
            <xsl:copy-of select="."/>
          </xsl:for-each>
          <xsl:for-each select="@*">
            <xsl:copy-of select="."/>
          </xsl:for-each>
          <xsl:attribute name="seq_no">
          <xsl:value-of select="$SequenceNumber"/></xsl:attribute>
        </Flat>
        <!-- Add a linebreak to the output XML, just for appearance -->
        <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
  <xsl:template match="PhoenixDebtOrderIOI">
    <Flat>
      <!-- Root PhoenixDebtOrder attributes -->
      <xsl:for-each select="../../@*">
        <xsl:copy-of select="."/>
      </xsl:for-each>
      <!-- PhoenixDebtOrderProduct attributes -->
      <xsl:for-each select="../@*">
        <xsl:copy-of select="."/>
      </xsl:for-each>
      <!-- PhoenixDebtOrderIOI attributes -->
      <xsl:for-each select="@*">
        <xsl:copy-of select="."/>
      </xsl:for-each>
      <xsl:attribute name="seq_no">
      <xsl:value-of select="$SequenceNumber"/></xsl:attribute>
    </Flat>
    <!-- Add a linebreak to the output XML, just for appearance -->
    <xsl:text disable-output-escaping="yes">&#10;</xsl:text>
  </xsl:template>
</xsl:stylesheet>
```

Editing Enterprise Message Sources

To edit an enterprise message source:

1. Select **Enterprise Message Sources** from the Architect function list.
2. Click the name of the enterprise message source.

   The message source properties display.
3. Click **Edit**.
4. Make the changes and click **Save**.

Copying Enterprise Message Sources

To copy an enterprise message source:

1. Select **Enterprise Message Sources** from the Architect function list.
2. Click the name of the enterprise message source to copy.

   The message source properties display.
3. Click **Copy**.
4. Type a new name for the copy of the enterprise message source and click **Copy**.

   The new message source is created and added to the list.

Deleting Enterprise Message Sources

To delete an enterprise message source:

1. Select **Enterprise Message Sources** from the Architect function list.
2. Click the name of the enterprise message source to delete.

   The message source properties display.
3. Click **Delete**.
4. Click **OK** to confirm that you want to delete the message source.

   The message source is deleted.

4

External Data Sources

This chapter contain the information needed to create and manage External Data Sources.

This chapter contains the following topics:

Introducing External Data Sources

An external data source is a connection to an external database. External data sources usually contain data that does not change very much or data that is too large to bring into the Active Data Cache.

External data source configurations can be exported and imported using ICommand, but you cannot import or edit the contents using ICommand, Enterprise Link, or Architect.

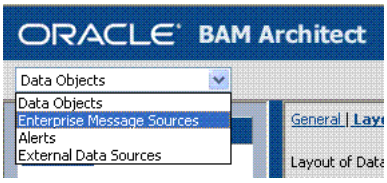
Passwords are entered in clear text. You cannot use DSNs (data source names).

Listing External Data Sources

To view the existing external data sources:

- Select **External Data Sources** from the Architect function list.

Figure 4–1 Architect Function List



Defining External Data Sources

To define an external data source:

1. Select **External Data Sources** from the Architect function list.
2. Click **Create**.
3. Enter a name and a description for the external data source.
4. Enter **Driver**, for example, Microsoft ODBC for Oracle.
5. Enter database user credentials in the **Login** and **Password** fields.
6. Enter **Connection string/URL**.

For a complete example of defining an External Data Source against an Oracle database, see "External Data Source Example" on page 4-2.

Editing External Data Sources

To edit an external data source:

1. Select **External Data Sources** from the Architect function list.
2. Select the external data source to edit.

   The external data source properties display.
3. Select **Edit**.
4. Make the changes and click **Save**.

Deleting External Data Sources

To delete an external data source:

1. Select **External Data Sources** from the Architect function list.
2. Select the external data source to delete.

   The data source properties display.
3. Select **Delete**.
4. Click **OK** to confirm that you want to delete the data source.

   The data source is deleted.

External Data Source Example

This example uses the sample scott/tiger user account and the EMP table in the Oracle database. You may need to unlock the scott/tiger account before proceeding with this example.

Step 1: Create an External Data Source

1. Select **External Data Sources** from the Architect function list.
2. Click **Create**.
3. Enter `myDataSource` in the **External Data Source Name** field.
4. Enter `My Example External Data Source` in the **Description** field.
5. Enter `Microsoft ODBC for Oracle` in the **Driver** field.

6. Enter `scott` in the **Login** field and tiger in the **Password** field.

   This sample account comes with your Oracle database installation. If you do not have this sample account you can create a new account and use it for this example.

7. Enter `server=`*`net_service_name`* in the **Connection string/URL**.

   This entry needs to be a Net Service Name defined in your tnsnames.ora file.

8. Click **Save**.

9. Click **Continue**.

   The External Data Source information is displayed on the screen.

Step 2: Create a Data Object using the External Data Source

1. Select **Data Objects** from the Architect function list.
2. Click **Create Data Object**.
3. Enter `Employees` in the **Name for new Data Object** field.
4. Leave the slash (`/`) in the **Location for new Data Object** field.

   The data object will appear in the top level **Data Objects** folder.

5. Leave the **Tip text** field blank.
6. Enter `Oracle Database Sample EMP Table` in the **Description** field.
7. Select the **External Data Source** checkbox.
8. Select **myDataSource** from the **External Data Source** list.
9. Enter `emp` in the **External Table Name** field.
10. Add the following fields to the data object:

Field	External Field Name	Field Type
ename	ename	String
empno	empno	Integer
job	job	String
mgr	mgr	Integer
hiredate	hiredate	DateTime
sal	sal	Decimal
comm	comm	Decimal
deptno	deptno	Integer

   Keep default settings for field attributes not specified in the table.

11. Click **Create Data Object**.
12. Click **Continue**.
13. Click **Contents** to view the contents of the data object

    The data in the Employees data object should match the data in the Oracle database sample EMP table.

5

Using Alerts

This chapter describes how to use Alerts.

This chapter contains the following topics:

Introducing Alerts

Alerts are launched by a set of specified events and conditions, known as a Rule. Alerts can be launched by data changing in a report or can be used to send a report to users daily, hourly, or at set intervals. Events in an alert rule can be an amount of time, a specific time, or a change in a specific report. Conditions restrict the alert rule to an event occurring between two specific times or dates. As a result of events and conditions, reports can be sent to users through email.

Alerts can be created in Active Studio also. See *Oracle Business Activity Monitoring Active Studio User's Guide* for more information about Alerts.

Building Alert Rules

To build an alert rule:

1. Select the **Alerts** tab in the Architect function list.
2. Click **Create A New Alert**.

   The Rule Creation and Edit dialog displays.
3. Click **Create A Rule.**

4. Enter a name for the alert rule.
5. Select an event that will launch the alert. See "Events" on page 5-2 for descriptions of each event.
6. Click **Next**.
7. Select one or more conditions, if needed. See "Conditions" on page 5-3 for descriptions of each condition.
8. Select one or more actions. See "Actions" on page 5-3 for descriptions of each action.
9. In the rule expression, click each underlined item and specify a value to complete the alert rule. For example, click **select report**, and choose a report in the dialog that displays. Other values you define include user names receiving reports, dates and times, time intervals, and filter expressions for a specific field. To continue adding conditions or actions, click the last line in the expression and then select another condition or action.

   You can click the **Back** and **Next** buttons to navigate between the Events page and the page containing actions and conditions, and make changes to those parts of the alert rule expression already constructed.
10. You can click the **Frequency Constraint** button to set a limit to how often an alert can launch. The default frequency constraint for alerts is five seconds. Type a number and select a time measurement such as seconds, minutes, or hours, and click **OK**. To turn off the frequency constraint, uncheck the **Constraint Enabled** checkbox. For more information about frequency constraint see "Frequency Constraint" on page 5-3.
11. Click **Delete this expression** to remove lines from the alert rule.
12. Click **OK**.

    The alert rule is added to list and is active.

Using Alert Rule Options

The following are the options for creating alert rules.

Events

Events launch the rule and trigger the action. Each rule contains only one event.

- **In a specific amount of time.** Select a time interval in seconds, minutes, or hours.
- **At a specific time today.** Select a time.
- **On a certain day at a specific time.** Select a specific date and a time.
- **Every interval between two times.** Select a time interval and two times.
- **Every date interval starting on certain date at a specific time.** Select a date interval such as day, week, month, or year, and a specific date and time.
- **When a report changes.** Select a report to monitor.
- **When a data field changes in data object.** Select a data object and a data field to monitor.
- **When a data field in a report meets specified conditions.** Select a report, the data object, and create a filter on the field to monitor.

- **When a data field in a data object meets specified conditions.** Select a data object and create a filter on the field to monitor.
- **When this rule is launched.** No options to select. When this rule is launched is the event to create dependencies between rules.

Conditions

Conditions are optional. You can select any number of conditions.

- **If it is between two times.** Select two times.
- **If It is between two days.** Select two dates.
- **If it is a particular day of the week.** Select a day of the week.

Actions

Actions are the results of a launched alert. You can select any number of actions.

- **Send a report via email.** Select a report, select to send the report as a report link or as a rendered report, and select a recipient.
- **Send a report via Active Messenger.** This option is not supported.
- **Send a report via the recipient's alert delivery settings.** Select a report and select a recipient. Recipients can configure their own alert delivery settings and delivery order through the personalize link.
- **Send a message via email** Create a message to send and select a recipient.
- **Send a message via Active Messenger.** This option is not supported.
- **Send a message via the recipient's alert delivery settings.** Create a message to send and select a recipient.
- **Send a report via Active Messenger and escalate to another user after a specific amount of time.** This option is not supported.
- **Send a report via email and escalate to another user after a specific amount of time.** Create a message and send to a recipient. Select a secondary recipient to receive the message if the first recipient does not respond within the specified time period.
- **Send a parameterized message.** You can use this option to send reports to other users under the conditions specified. This action is available for the events **When a data field changes in data object** and **When a data field in a data object meets specified conditions**. See "Parameterized Alerts" on page 5-7 for more information.
- **Launch a rule.** Select a dependent rule that includes the when this rule is launched event.
- **Run a Plan.** Select a plan to run.
- **Launch rule if an action fails.** Select a dependent rule to launch if any of the actions included in the rule fail.

Frequency Constraint

Frequency Constraint can be edited only if it is appropriate for the event selected. otherwise it will be disabled. It can be set to a value of time which could be in seconds, minutes, or hours.

Rules have a five second frequency constraint by default. This limits the amount of times the rule will launch in a period of time. With real-time data, transactions can

occur every millisecond, so alerting frequency must be controlled. If the rule is satisfied many times within five seconds, users would not want alerting more than once in five seconds.

Creating Alert Rules From Templates

Alert rule templates are a convenient preselected group of events and conditions based on some common use cases.

To create an alert rule from a template:

1. Click **Create A New Alert**.

   The Create Alert Rule dialog displays.

2. Click **Create A Rule From A Template**.
3. Enter a name for the alert rule.
4. Select a template from the list.
5. In the **Rule Expression** box, click each underlined item and specify a value to complete the alert rule. For example, click **select report**, and choose a report in the dialog that displays. Other values you define include user names receiving reports, dates and times, time intervals, and filter expressions for a specific field.
6. You can click **Frequency Constraint** to specify how often an alert can launch. The default frequency constraint for alerts is five seconds. Enter a number and select a time measurement such as seconds, minutes, or hours, and click **OK**.
7. You can click **Modify this rule** to modify the rule without using the template. This provides more options for creating rules.
8. Click **OK**.

   The alert rule is added to list and is active.

Creating Alert Rules With Messages

You can create alert rules that send messages. The messages can contain information such as report names, links to reports, and user names. Messages can also include variables that are set when the alert is launched, such as the time that an event occurred and the data that launched the event. To use data variables, the event must be based on data.

To create an alert rule that includes a message:

1. Start building an alert rule.
2. Select one of the following actions:
   - **Send a message via email**
   - **Send a message via the recipient's alert delivery settings**
3. Click **create message** in the rule expression.

   The Alert Message dialog displays.

4. Enter a subject in the **Subject** line.
5. Enter the message in the **Message Text** box.
6. Include special fields into the message.

Special fields are listed in the box in the lower left corner of the Alert Message dialog. The special fields listed change when reports are selected on the right side of the dialog.

To insert a special field into the message:

a. Select a special field from the list.

b. Click **Insert into subject** or **Insert into text**.

You can insert multiple values of the same type, for example, multiple links to different reports.

- **Send Report Name** inserts name of selected report.
- **Send Report Owner** inserts owner name of selected report.
- **Send Report Link** inserts link to selected report.
- **Changed Report Name** inserts name of the changed report.
- **Changed Report Owner** inserts Owner Name Of Changed Report.
- **Target User** inserts user name of message recipient.
- **Date/Time Sent** inserts date and time of message sent.

7. Click **OK**.

Creating Complex Alerts

You can create nested rules with many actions and chained rules that launch other rules.

You can chain rules by creating two types of rules:

- A dependent rule that must be launched by another rule.
- A rule with an action to launch a dependent rule.

To create dependent rules:

1. Create a rule that includes the event **When this rule is launched**. There is no value to specify for this event.
2. Create a rule that includes the action **Launch a rule** or **Launch rule if an action fails**. The **Launch rule if action fails** applies to any of the actions contained in the rule.
3. Click **select rule** in the action.

   The Select Dependent Rule dialog displays.
4. Select a dependent rule. Only rules that include the **When this rule is launched** event display in the list.
5. Click **OK**.

To handle a failing action, add the action **Launch rule if action fails**. For example, if a rule is supposed to send a message, and for some reason the message does not send, you could launch another rule to notify you.

Modifying Rules for Alerts

You modify alert rules from the Alerts tab.

To modify an alert rule:

1. Select the alert rule to edit.
2. Click **Edit** in the Alert Actions list.

   The Rule Creation and Edit dialog displays.
3. Make changes to the alert and click **OK**.

When you modify alert rules created from a template, you can add new lines and select conditions and actions the same as when you build alert rules without templates.

Activating Alerts

When you create an alert rule, it is automatically active. If you want an alert to be temporarily inactive but you do not want to delete it, you can turn it off by deselecting the **Activate** checkbox.

To change the activity status of an alert rule:

1. Select **Alerts** from the Architect function list.
2. Select the **Activate** checkbox for the alert rule.

A checked box means the alert rule is active.

An unchecked box means the alert rule is inactive.

Checking the **Activate** checkbox does not cause an alert to launch, it only enables the rule so that if the specified event occurs, the alert will launch.

An exclamation mark on the alert icon indicates it has launched and will not be valid again or because items that it references are missing and it cannot launch.

Launching Alerts by URL

You can use the alerts web service to manually launch alerts. For more information, refer to:

http://<host>:<http_port>/oraclebam/services/manualrulefire.asmx?op=FireRuleByName

You define the rule name using the format:

```
DOMAIN\username.alertname
```

Deleting Alerts

To delete an alert:

1. Select the alert to delete.
2. Click **Delete** in the Alert Actions list.

   A dialog displays to confirm that you want to delete the alert.
3. Click **OK**.

   The alert is deleted.

Parameterized Alerts

When creating a parameterized alert, you must populate the **set parameters** section. In this section, populate the **User**, **Delivery**, and **Report** fields with either predefined values or dynamically from a Data Object field.

- **User** field

  If you populate this field with predefined values, the value that appears must follow a format such as `MY-DOMAIN\myhost-pc`. If you populate this field from a Data Object field, the value also follows a format such as `MY-DOMAIN\myhost-pc`.

- **Delivery** field

  If you populate this field with predefined values, the value that appears in this field is `Email` (or something similar). If you populate this field from a Data Object field, the value must be `smtp`.

- **Report** field

  If you populate this field with predefined values, the value that appears in this field is `Emp_Report` (or something similar) If you populate this field from a Data Object field, the value must be the report ID of that report, and not the name. To get the report ID, click the report (for example, `Emp_Report`) and click the **Copy Shortcut** link. A window displays with a link such as:

  http://SERVER1/oraclebam/ReportServer/default.aspx?Event=ViewReport&**ReportDef=1**&Buttons=False.

  In this link the **ReportDef** value, `1`, is the report ID of the report `Emp_Report`. Every report in Oracle Business Activity Monitoring has a unique report ID.

6

Using ICommand

This chapter provides usage and reference material for the ICommand command-line utility and web service. It contains the following topics:

Introducing ICommand

ICommand is a command-line utility and web service that provides a set of commands that perform various operations on items in the Active Data Cache and the Enterprise Link Repository.

The commands may be in an input XML file, or a single command may be given on the command line.

Informational and error messages may be output to either the command window or to an XML file.

General Command and Option Syntax

All parameters given on the DOS command line are in the form `name=value`. The `name` portion is not case sensitive. If the `value` portion must contain spaces or other special characters, it may be enclosed in quotes. For some parameters, the `value` may be omitted.

On the DOS command line, commands are specified by the value of the `cmd=commandname` parameter. Options for the command are specified by `parametername=value` parameters.

In an XML command file, commands are specified by the XML tag. Options for the command are given as XML attribute values of the command tag, in the form `parametername=value`.

Command names and parameter values (except for Active Data Cache item names) are not case sensitive.

It is required to use quotes around report names and file names that contain spaces and other special characters.

You can specify multiple Active Data Cache types in a single command and pass parameters to them. For example:

```
icommand.exe cmd=export type=all report,rule,folder:owner=1
folder,dataobject:permissions=1 systemobjects=1 file=filename.xml
```

In this example, the command will pass owner=1 to the report, rule, and folder Active Data Cache object types. The comma (,) separates the object types and the parameter is listed after a colon (:).

This accomplishes the same end as the following three commands:

```
icommand.exe cmd=export type=report owner=1 .....
icommand.exe cmd=export type=rule owner=1 .....
icommand.exe cmd=export type=folder owner=1 .....
```

Command-line-only Parameters

The following parameters can appear only on the command line:

- `Domain=domain`

  Optional parameter that specifies the domain name to use to login to the Active Data Cache (the name of the machine on which the Active Data Cache server is running).

  If this parameter is omitted, `main` is used, which means the server information will be obtained from the `ADCServerName` key in the ICommand.exe.config file.

  If the reserved value `ADCInProcServer` is used, then ICommand will directly access the Active Data Cache database (which must be local on the same machine on which ICommand is running) rather than contacting the Active Data Cache server. This option should be used **only** when the Active Data Cache server is not running; otherwise corruption of the database could occur. The information about the location and structure of the Active Data Cache database is obtained from various keys in the ICommand.exe.config file.

- `Logfile=filename`

  Optional parameter that specifies the name of the file to which results and errors are logged. If the file does not exist, it will be created. If the file does exist, any contents will be overwritten. Since this is an XML file, it would usually have the XML extension, although that is not required.

  If this parameter is not present, results and errors will be output to the console.

- `Logmode=mode`

Optional parameter that indicates whether an existing log file is to be overwritten or appended to. The possible values for this parameter are `append` or `overwrite`. In either case, if the log file does not exist it will be created.

If this parameter is not present, `overwrite` is assumed.

Note that because it is XML that is being added to the log file, if the `append` option is used the XML produced may not be strictly legal, as there will be no top level root tag in the XML produced by successive appends (ICommand will append the same tag each time it is run). It is left up to the user to handle this.

- `Cmdfile=`*`filename`*

  Optional parameter that specifies the name of the file that contains commands to be processed. Since this is an XML file, it would usually have the XML extension, although that is not required.

  The `Cmdfile` and `cmd` parameters are mutually exclusive. Exactly one of them must be present.

- `Cmd=`*`commandname`*

  Optional parameter that specifies a single command to be executed. Any parameters needed for the command must also be on the command line.

  The `Cmdfile` and `cmd` parameters are mutually exclusive. Exactly one of them must be present.

- `Debug=`*`flag`*

  Optional parameter that indicates whether extra debugging information is to be output in the event of an error. Any value other than `0` (zero), or the absence of any value, indicates that debugging information is to be output. If this parameter is not present, no debugging information will be output.

Summary of Individual Commands

Table 6–1 summarizes the commands.

Table 6–1 ICommand Command Summary

Command	Parameters
export	file=*filename* [name=*itemname*] [type=dataobject|securityfilters|folder|report|rule| user|role|distributionlist|ems|emstype|eds|edstype| plan|all] [match=*pattern*] [regex=*regularexpression*] [all[=0|1]] [systemobjects[=0|1]] [dependencies[=0|1]] [layout[=0|1]] [contents[=0|1]] [permissions[=0|1]] [privileges[=0|1]] [members[=0|1]] [owner[=0|1]] [planmonitorservicename=*servicename*] [header[=0|1]] [footer[=0|1]] [append[=0|1]] [preview[=0|1]]
import	file=*filename* [delay=*milliseconds*] [updatelayout] [mode=preserveid|update|overwrite|append|rename|error] [preserveowner] [setcol=*col_name*/[null|now|value:*override_value*]]
delete	[name=*itemname*] [type=dataobject|securityfilters|folder|report|rule| user|role|distributionlist|ems|eds|plan|all] [match=*pattern*] [regex=*regularexpression*] [all[=0|1]] [systemobjects[=0|1]]
rename	name=*itemname* newname=*newitemname* [type=dataobject|folder|report|rule|user|role| distributionlist|ems|eds|plan]
clear	name=*itemname* [type=dataobject|folder|role|distributionlist]

Table 6–1 (Cont.) ICommand Command Summary

Command	Parameters
run	name=*planname* [type=plan]
stop	requestid=*planexecutionrequestid* [type=plan]
Setmonitoring	[name=*plan_name*] [type=*plan*] [match=*dos_pattern*] [regex=*regular_expression*] [all[=0|1]] [planmonitorservicename=*service_name*] [enabled[=0|1]] [restartoncompletion=never|always|count] [restartonfailure=never|always|count] [restartfrequencymax=none|*count_in_minutes*] [preferredservicename=*service_name*|*empty_string*]

Detailed Command Descriptions

Export

Exports information about one or more items to an XML file.

Table 6–2 Export Command

Parameter	Description
file=*filename*	The name of the file to export to. Required. If the file does not exist, it will be created. If the file does exist, any contents will be overwritten, unless the append parameter is used. Since the file will contain XML, it would usually have an XML extension.
name=*itemname*	The name of the item to be exported.

Table 6–2 (Cont.) Export Command

Parameter	Description
`type=`*`itemtype`*	The type of the item to be exported. Must be one of the following: ■ `dataobject` ■ `securityfilters` (For the specified Data Objects) ■ `folder` ■ `report` ■ `rule` ■ `user` ■ `role` ■ `distributionlist` ■ `ems` (Enterprise Message Source) ■ `emstype` ■ `eds` (External Data Source) ■ `edstype` ■ `plan` ■ `all` `dataobject` is assumed if this parameter is omitted.
`match=`*`pattern`*	A DOS-style pattern matching string, using the * (asterisk) and? (question mark) characters. The items whose names match the pattern will be exported.
`regex=`*`regularexpr`*	A regular expression pattern matching string. The items whose names match the pattern will be exported.
`all[=0|1]`	Controls whether all items of the specified type will be exported. A nonzero or omitted value means export all items of the specified type, a zero value means only export the named (or matched) items. Zero is assumed if this parameter is omitted. For Reports, Folders and Rules, only the items owned by the user running ICommand are exported, unless the user running ICommand is an Administrator. When an Administrator runs ICommand, any user's items may be exported.
`[systemobjects[=0|1]]`	Controls whether Data Objects in the System folder are included when the `all`, `match`, or `regex` parameters are used. Zero means these data objects are not included. Zero is assumed if this parameter is omitted.
`dependencies[=0|1]`	Applies to only to Data Objects. Controls whether other Data Objects that the exported Data Objects depend on in the lookup columns will also be exported. A nonzero value or the parameter present with no value specifies that if the Data Objects being exported contain lookup columns, then the Data Objects that are looked up will also be exported.
`layout[=0|1]`	Applies only to Data Objects. Controls whether layout information is to be exported. A nonzero value means export layout information. Zero means do not export layout information. nonzero is assumed if this parameter is omitted, or if the value is omitted.

Table 6–2 (Cont.) Export Command

Parameter	Description
`contents[=0|1]`	Applies only to Data Objects. Controls whether content information (row, column values) is to be exported. A nonzero value means export content information. Zero means do not export content information. nonzero is assumed if this parameter is omitted, or if the value is omitted.
`permissions[=0|1]`	Applies only to Data Objects and Folders. Controls whether permissions information is to be exported. A nonzero value means export information about the permission settings of the exported Data Objects or Folders. Zero means do not export permission information. nonzero is assumed if this parameter is omitted, or if the value is omitted. For Data Objects, only the permissions of the Data Object itself are exported. Any permissions that might be on the folders or subfolders that the Data Objects are contained within are not included. For Folders, the permissions reflect the cumulative permissions of all parent Folders of the Folders being exported.
`privileges[=0|1]`	Applies only to Roles. Controls whether the privilege settings in the Roles being exported are exported. A nonzero value means export the privilege settings. Zero means do not export the privilege settings. nonzero is assumed if this parameter is omitted, or if the value is omitted.
`members[=0|1]`	Applies only to Roles. Controls whether the list of users in the Roles being exported are exported. A nonzero value means export the list of users. Zero means do not export the list of users. nonzero is assumed if this parameter is omitted, or if the value is omitted.
`owner[=0|1]`	Applies only to Folders, Reports, and Rules. Controls whether the information about the owner of the items being exported is included in the export. A nonzero value means export the owner information. Zero means do not export the owner information. nonzero is assumed if this parameter is omitted, or if the value is omitted.
`header[=0|1]`	Controls whether XML header information is written to the front of the export file. This can be used to allow successive executions of ICommand to assemble one XML file by repeatedly appending to the same file. A nonzero value means write the header. Zero means do not write the header. nonzero is assumed if this parameter is omitted, or if the value is omitted.
`footer[=0|1]`	Controls whether closing XML information is written to the end of the export file. This can be used to allow successive executions of ICommand to assemble one XML file by repeatedly appending to the same file. A nonzero value means write the closing information. Zero means do not write the closing information. nonzero is assumed if this parameter is omitted, or if the value is omitted.
`append[=0|1]`	Controls whether the exported information is appended to any existing file. A nonzero value means append. Zero means overwrite the contents of any existing files. nonzero is assumed if this parameter is omitted, or if the value is omitted.

Table 6–2 (Cont.) Export Command

Parameter	Description
`preview[=0|1]`	In `preview` mode, ICommand goes through the motions of exporting all of the specified items, but does not actually output any information. This can be used to see what would be exported for a given command line, and what errors might occur. In this mode, ICommand export continues processing even after some errors that would cause non-preview mode to stop the export. A nonzero value means preview mode. nonzero is assumed if the value is omitted. Zero is assumed if the parameter is omitted.
`planmonitorservicename=`*`servicename`*	Applies only to Rules and Plans. A value that specifies Plan Monitor service location. If not specified, retrieves the value from various keys in the ICommand.exe.config file. First it tries for a service name from the PlanMonitor.ServiceName key. If no service name is specified, it tries for a service URL from the PlanMonitor.ServiceURL key. If no service URL is specified, it tries for individual settings from the following set of keys: PlanMonitor.ServiceMachine, PlanMonitor.ServiceChannel, and PlanMonitor.ServicePort.

Import

Imports the information from an XML file to an item. The item may be created, replaced, or updated.

If the item does not exist, it will be created if possible. For Data Objects, the input file must contain layout information in order to create the Data Object, and if the file contains no content information, then an empty Data Object will be created.

If the user running ICommand is not an Administrator, Reports are always imported to the private folders of the user running ICommand. If the path information in the import file exactly matches existing private folders of the user running ICommand, the imported report is placed in that location. Otherwise, it is placed into the root of that user's private folders.

If the user running ICommand is an Administrator, then the `preserveowner` option may be used to allow Folders, Reports and Rules to be imported with their original ownership and to their original location.

Table 6–3 Import Command

Parameter	Description
`file=`*`filename`*	The name of the file to import from. Required. This would usually be a file that was created through the export command.
`delay=`*`millisec`*	Applies only to Data Objects. A value that specifies a delay that is to occur between each row insertion or update. This can be used to simulate active data at a specified rate. The number is the number of milliseconds to wait between each row. It must be greater than zero. If this parameter is omitted, there will be no delay.
`updatelayout`	Applies only to Data Objects. Controls whether, if the Data Object being imported already exists, the layout (schema) of the Data Object is updated according to the layout information in the import file. True if parameter is present; false if parameter is not present.

Table 6–3 (Cont.) Import Command

Parameter	Description

`mode=`*`mode`*	The following modes are valid for the following item types:
	The following values are valid for Folders, Reports, Users, Roles, EMS, EMSTypes, EDS. EDSTypes, and Plans:
	`overwrite`
	If the item already exists, replace it with the imported item.
	`rename`
	If the item already exists, change the name of the imported item. The new name is computed automatically and reported in a message.
	`error`
	If the item already exists, terminate the import with an error.
	The following values are valid for Distribution Lists:
	`overwrite`
	If the item already exists, replace it with the imported item.
	`rename`
	If the item already exists, change the name of the imported item. The new name is computed automatically and reported in a message.
	`append`
	If the item already exists, append the users in the imported list to the already existing list.
	`error`
	If the item already exists, terminate the import with an error.
	Only the following values are valid for Data Objects:
	`preserveid`
	If the imported Data Object does not already exist and must be created, ICommand will attempt to assign the Data Object the same internal ID that the exported Data Object had. If it is unable to, the import will be terminated with an error.
	This option is important because some other items, such as Reports, point to the Data Objects they use by ID, not by name.
	`Update`
	Normally, when ICommand imports a Data Object, it creates a new Data Object or locates the existing Data Object and inserts the imported rows into that Data Object.
	In `update` mode, ICommand instead attempts to locate existing matching rows by Row ID, and updates those existing rows with the values in the import file. Unmatched rows are inserted. For matching Row IDs in the import file that have no data columns specified, the rows are deleted from the existing Data Object.
	For Security Filters, the only value supported is `overwrite`. If `overwrite` is not specified and the Data Object already contains at least one Security Filter, the import will be aborted with an error.
	This parameter is not supported for Rules.

Table 6–3 (Cont.) Import Command

Parameter	Description
`preserveowner`	Applies only to Folders, Reports, and Rules. Controls whether, when the item is imported, the ownership of the item is set as specified in the import file. This setting of ownership can only be done if the ownership was included in the file during export, and if the user running ICommand is an Administrator. A nonzero value means set the ownership as specified in the import file. Zero (0) means the imported items remain owned by the user running ICommand. nonzero is assumed if this parameter is omitted, or if the value is omitted.

`setcol`	Allows override of column values from the command line during import, including setting to current date/time.
	
	`setcol=`*`column_name`*`/NULL`
	
	`setcol=`*`column_name`*`/NOW`
	
	`setcol=`*`column_name`*`/VALUE:`*`override-value`*
	
	`column_name` is the name of one of the columns in the Data Object being imported. This cannot be a column of type lookup or calculated. Column names that are not contained in the input XML being imported can be specified, as long as they are columns in the Data Object being imported into.
	
	The portion after the slash specifies a value that should be substituted for that column on each row that is imported -- any value for that column in the import file will be ignored (overridden). Note that slash is the one character that is not permitted in column names, so there is no potential conflict with any column names in this syntax.
	
	`NULL` specifies that the column value should be set to null. The column must be defined as "nullable" in the Data Object's layout.
	
	`NOW` specifies that the column value should be set to the current date/time at the time that the column value is being set into the row. This option can only be used for columns of type datetime, timestamp, and string.
	
	`VALUE:`*`override-value`* specifies an arbitrary constant value (after the colon) that the column should be set to. The value must be a legal value for the type of the column.
	
	In order to allow more than one column to be overridden, any number of `setcol` parameters may be present. However, since duplicate parameters are not permitted, ICommand will recognize any parameter name that starts with `setcol` as a `setcol` parameter (for example, `setcol1`, `setcol2`, and so on).
	
	Example command line:
	
	`icommand cmd=import file=myfile.xml setcol1=Field1/null setcol2=Field3/now setcol3="Customer Name/value:John Q. Public"`

`planmonitorservicename=`*`servicename`*	Applies only to Rules and Plans. A value that specifies Plan Monitor service location. If not specified, retrieves the value from various keys in the ICommand.exe.config file. First it tries for a service name from the PlanMonitor.ServiceName key. If no service name is specified, it tries for a service URL from the PlanMonitor.ServiceURL key. If no service URL is specified, it tries for individual settings from the following set of keys: PlanMonitor.ServiceMachine, PlanMonitor.ServiceChannel, and PlanMonitor.ServicePort.

Delete

Deletes an item.

Table 6–4 Delete Command

Parameter	Description
`name=`*`itemname`*	The name of the item to be deleted.
`type=`*`itemtype`*	The type of the item to be deleted. Must be one of the following: ■ `dataobject` ■ `securityfilters` (For the specified Data Objects) ■ `folder` ■ `report` ■ `rule` ■ `user` ■ `role` ■ `distributionlist` ■ `ems` (Enterprise Message Source) ■ `emstype` ■ `eds` (External Data Source) ■ `edstype` ■ `plan` ■ `all` `dataobject` is assumed if this parameter is omitted.
`match=`*`pattern`*	A DOS-style pattern matching string, using the * (asterisk) and ? (question mark) characters. The items whose names match the pattern will be deleted.
`regex=`*`regularexpr`*	A regular expression pattern matching string. The items whose names match the pattern will be deleted.
`all[=0|1]`	Controls whether all items of the specified type will be deleted. A nonzero or omitted value means delete all items of the specified type, a zero value means only delete the named (or matched) items. Zero is assumed if this parameter is omitted.
`[systemobjects[=0|1]]`	Controls whether Data Objects in the System folder are included when the `all`, `match`, or `regex` parameters are used. Zero means these data objects are not included. Zero is assumed if this parameter is omitted.
`planmonitorservicename =`*`servicename`*	Applies only to Rules and Plans. A value that specifies Plan Monitor service location. If not specified, retrieves the value from various keys in the ICommand.exe.config file. First it tries for a service name from the PlanMonitor.ServiceName key. If no service name is specified, it tries for a service URL from the PlanMonitor.ServiceURL key. If no service URL is specified, it tries for individual settings from the following set of keys: PlanMonitor.ServiceMachine, PlanMonitor.ServiceChannel, and PlanMonitor.ServicePort.

Rename

Renames an item.

Table 6–5 Rename Command

Parameter	Description
`name=itemname`	The name of the item to be renamed. Required.
`newname=newitemname`	The new name for the item. Required. For Data Objects, Reports and Folders, only the new base name should be given, with no path (for example Report1).
`type=itemtype`	The type of the item to be renamed. Must be one of the following: ■ `dataobject` ■ `folder` ■ `report` ■ `rule` ■ `user` ■ `role` ■ `distributionlist` ■ `ems` (Enterprise Message Source) ■ `emstype` ■ `eds` (External Data Source) ■ `edstype` ■ `plan` `dataobject` is assumed if this parameter is omitted.
`planmonitorservicename=servicename`	Applies only to Rules and Plans. A value that specifies Plan Monitor service location. If not specified, retrieves the value from various keys in the ICommand.exe.config file. First it tries for a service name from the PlanMonitor.ServiceName key. If no service name is specified, it tries for a service URL from the PlanMonitor.ServiceURL key. If no service URL is specified, it tries for individual settings from the following set of keys: PlanMonitor.ServiceMachine, PlanMonitor.ServiceChannel, and PlanMonitor.ServicePort.

Clear

Clears the contents of an item.

What it means to be *cleared* depends upon the item type:

- For Data Objects, all existing rows within the Data Object are deleted.
- For Folders, all contents of the Folder are deleted.
- For Roles and Distribution Lists, all members (users) are removed.

Table 6–6 Clear Command

Parameter	Description
`name=itemname`	The name of the item to be cleared. Required.

Table 6–6 (Cont.) Clear Command

Parameter	Description
`type=`*`itemtype`*	The type of the item to be cleared. Must be one of the following: ▪ `dataobject` ▪ `folder` ▪ `role` ▪ `distributionlist` `dataobject` is assumed if this parameter is omitted.

Run

Causes an instance of an Enterprise Link Plan to begin running.

Note that even if one or more instances of the Plan are already running, this command will cause another instance to begin running.

Table 6–7 Run Command

Parameter	Description
`name=`*`planname`*	The name of the Plan to run. Required.
`type=plan`	Optional. If present, the value must be `plan`.
`planmonitorservicename =`*`servicename`*	Applies only to Rules and Plans. A value that specifies Plan Monitor service location. If not specified, retrieves the value from various keys in the ICommand.exe.config file. First it tries for a service name from the PlanMonitor.ServiceName key. If no service name is specified, it tries for a service URL from the PlanMonitor.ServiceURL key. If no service URL is specified, it tries for individual settings from the following set of keys: PlanMonitor.ServiceMachine, PlanMonitor.ServiceChannel, and PlanMonitor.ServicePort.

Stop

Causes a running instance of an Enterprise Link Plan to be stopped.

Table 6–8 Stop Command

Parameter	Description
`requestid=`*`id`*	The Request ID of the running instance of a Plan, as assigned by the Enterprise Link Data Flow Service when the Plan was started.
type=plan	Optional. If present, the value must be `plan`.

Item Name Syntax

Whenever an item name is specified in a command, the following rules apply.

General rules

When specified on a DOS command line, if the name contains spaces or characters that have special meaning to DOS, the name must be quoted according to the rules for DOS command lines.

When specified in an XML command file, if the name contains characters that have special meaning within XML, the standard XML escaping must be used.

Data Objects

If the Data Object is not at the root, the full path name must be given, as in the following example:

```
/My Folder/My Subfolder/My Data Object
```

If the Data Object is at the root, the leading "/" is optional. The following two examples are equivalent:

```
/My Data Object
My Data Object
```

Reports

The full path name must be specified as in the following examples.

For shared reports:

```
"/public/Report/Subfolder1/My Report"
```

For private reports:

```
"/private:username/Report/Subfolder1/My Report"
```

For private reports the `/private:username/` prefix may be omitted if the user running ICommand is the user that owns the report.

The path information without the `public` or `private` prefix is saved in the export file.

Alert Rules

Either the name of the Alert, or the full name of the Alert may be specified. The following two examples are equivalent for Alerts if the user running ICommand is the user that owns Alert1:

```
Alert1
```

```
/private:username/Rule/Alert1
```

If the user running ICommand is not the owner of Alert1, then only the second form may be used.

All other item types

Specify the full name of the item.

Format of Command File

This section comtains the following topics:

- Inline Content
- Command IDs
- Continue On Error

The command file contains the root tag `OracleBAMCommands`.

Within the root tag is a tag for every command to be executed. The tag name is the command name, and the parameters for the command are attributes.

Sample command file:

```
<?xml version="1.0" encoding="utf-8"?>
<OracleBAMCommands ContinueOnError="1">
  <Export name="Samples/Media Sales" file="MediaSales.xml" contents="0" />
  <Rename name="Samples/Call Center" newname="Call Centre" />
  <Delete type="EMS" name="WebLog" />
  <Delete type="EMS" name="WebLog2" />
</OracleBAMCommands>
```

Inline Content

When using a command file to import, the `inline` option enables you to include the import content inside the command file, rather than in a separate import file. Here is an example:

```
<?xml version="1.0" encoding="utf-8"?>
<OracleBAMCommands>
<Import inline="1">
<Export Version="504.0" Build="3.0.3697.0">
  <DataObject Version="13" Name="Employees" ID="_Employees" Path="/Samples"
   External="0">
    <Layout>
      <Column Name="Salesperson" ID="_Salesperson" Type="string" MaxSize="100"
       Nullable="1" />
      <Column Name="Sales Area" ID="_Sales_Area" Type="string" MaxSize="100"
       Nullable="1"/>
      <Column Name="Sales Number" ID="_Sales_Number" Type="integer" Nullable="1"
       />
      <Column Name="Timestamp" ID="_Timestamp" Type="timestamp" Nullable="0" />
      <Indexes />
    </Layout>
    <Contents>
      <Row ID="1">
        <Column ID="_Salesperson" Value="Greg Masters" />
        <Column ID="_Sales_Area" Value="Northeast" />
        <Column ID="_Sales_Number" Value="567" />
        <Column ID="_Timestamp" Value="2004-09-14T14:07:41.5600000-07:00" />
      </Row>
    </Contents>
  </DataObject>
</Import>
</OracleBAMCommands>
```

Command IDs

This feature is only used when output is being sent to a log file. To make the parsing of log results easier, each command can be given an ID. This ID will be included in the Result or Error elements of any output related to that command.

Sample Input:

```
<OracleBAMCommands>
  <Delete id="1" type="role" name="Report Creator" />
  <Delete id="2" type="user" name="joeschmoe" />
</OracleBAMCommands>
```

Sample Output:

```
<ICommandLog Login="MSOLNIT-PC\ASPNET">
```

```
  <Results Command="Delete" ID="1">Role "Report Creator" deleted.</Results>
  <Error Command="Delete" ID="2">
    <![CDATA[Error while processing command "Delete".
[ErrorSource="ICommandEngine", ErrorID="ICommandEngine.Error"] There is no User
named "joeschmoe". [ErrorSource="ICommandEngine",
ErrorID="ICommandEngine.UserExist"]]]>
  </Error>
</ICommandLog>
```

Continue On Error

Ordinarily, ICommand will execute commands in a command file until a failure occurs, or until they all complete successfully. In other words, if a command file contains 20 commands, and the second command fails for any reason, then no further commands will be executed. This behavior can be changed by using the `continueonerror` attribute at either a global level or for each command.

Example 6–1 shows how to use the `continueonerror` attribute so that all commands will be executed regardless of if any failures occur

Example 6–1 Enabling Global Continue-On-Error

```
<OracleBAMCommands continueonerror="1">
  <Delete id="1" type="role" name="Report Creator" />
  <Delete id="2" type="user" name="joeschmoe" />
</OracleBAMCommands>
```

In Example 6–2, `continueonerror` only applies to the command that deletes user joeschmoe. If this command fails, then ICommand will output the error and continue. But if any other command fails, ICommand will immediately stop.

Example 6–2 Enabling Command Continue-On-Error

```
<OracleBAMCommands>
  <Delete id="1" type="role" name="Report Creator" />
  <Delete id="2" type="user" name="joeschmoe" continueonerror="1" />
  <Delete id="3" type="user" name="user2" />
  <Delete id="4" type="user" name="user3" />
</OracleBAMCommands>
```

Format of Log File

The log file contains the root tag `ICommandLog`.

Within the root tag is an entry for every error or informational message logged.

Errors are logged with the tag `Error`.

Informational messages are logged with the tag `Results`.

Both `Results` and `Error` tags optionally contain an attribute of the form `Command=`*`cmdname`*, if appropriate, that contains the name of the command that generated the error or informational message.

Sample log file (output from the preceding sample command file):

```
<?xml version="1.0" encoding="utf-8"?>
<ICommandLog Login="MYDOMAIN\myaccount">
  <Results Command="Export">Data Object "/Samples/Media Sales" exported
  successfully (0 rows).</Results>
  <Results Command="Export">1 items exported successfully.</Results>
```

```
  <Results Command="Rename">Data Object "/Samples/Call Center" renamed to
  "/Samples/Call Centre".</Results>
  <Results Command="Delete">Enterprise Message Source "WebLog"
  deleted.</Results>
  <Error Command="Delete"><![CDATA[Error while processing command "Delete".
  [ErrorSource="ICommand", ErrorID="ICommand.Error"]
There is no Enterprise Message Source named "WebLog2".
  [ErrorSource="ICommand", ErrorID="ICommand.EMSExist"]]]></Error>
</ICommandLog>
```

Sample DOS Command Lines

Here are a few sample DOS command lines:

```
ICommand cmdfile=cmd.xml logfile=log.xml logmode=append

ICommand cmd=export name=WebLog2 file=WebLog2.xml

ICommand cmdfile=cmd.xml

ICommand cmd=export file=EveryEMS.xml all type=ems
```

Sample export file

```
<?xml version="1.0" encoding="utf-8"?>
<OracleBAMExport Version="504.0" Build="3.0.3697.0">
  <DataObject Version="13" Name="Employees" ID="_Employees" Path="/Samples"
   External="0">
    <Layout>
      <Column Name="Salesperson" ID="_Salesperson" Type="string" MaxSize="100"
       Nullable="1" />
      <Column Name="Sales Area" ID="_Sales_Area" Type="string" MaxSize="100"
       Nullable="1" />
      <Column Name="Sales Number" ID="_Sales_Number" Type="integer"
       Nullable="1" />
      <Column Name="Timestamp" ID="_Timestamp" Type="timestamp" Nullable="0" />
      <Indexes />
    </Layout>
    <Contents>
      <Row ID="1">
        <Column ID="_Salesperson" Value="Greg Masters" />
        <Column ID="_Sales_Area" Value="Northeast" />
        <Column ID="_Sales_Number" Value="567" />
        <Column ID="_Timestamp" Value="2004-09-14T14:07:41.5600000-07:00" />
      </Row>
      <Row ID="2">
        <Column ID="_Salesperson" Value="Lynette Jones" />
        <Column ID="_Sales_Area" Value="Southwest" />
        <Column ID="_Sales_Number" Value="228" />
        <Column ID="_Timestamp" Value="2004-09-14T14:07:41.5600000-07:00" />
      </Row>
      <Row ID="3">
        <Column ID="_Salesperson" Value="Noel Rogers" />
        <Column ID="_Sales_Area" Value="Northwest" />
        <Column ID="_Sales_Number" Value="459" />
        <Column ID="_Timestamp" Value="2004-09-14T14:07:41.5600000-07:00" />
```

```
      </Row>
    </Contents>
  </DataObject>
</OracleBAMExport>
```

Regular Expressions

The `export` and `delete` commands optionally accept a regular expression with the `regex` parameter.

A regular expression is a pattern of text that consists of ordinary characters (for example, letters a through z) and special characters, known as *metacharacters*. The pattern describes one or more strings to match when searching for items by name.

The following table contains the complete list of metacharacters and their behavior in the context of regular expressions:

Table 6–9 Metacharacters for Regular Expressions

Character	Description
\	Marks the next character as a special character, a literal, a backreference, or an octal escape. For example, 'n' matches the character "n". '\n' matches a newline character. The sequence '\\' matches "\" and "\(" matches "(".
^	Matches the position at the beginning of the input string. If the RegExp object's `Multiline` property is set, ^ also matches the position following '\n' or '\r'.
$	Matches the position at the end of the input string. If the RegExp object's `Multiline` property is set, $ also matches the position preceding '\n' or '\r'.
*	Matches the preceding character or subexpression zero or more times. For example, zo* matches "z" and "zoo". * is equivalent to {0,}.
+	Matches the preceding character or subexpression one or more times. For example, 'zo+' matches "zo" and "zoo", but not "z". + is equivalent to {1,}.
?	Matches the preceding character or subexpression zero or one time. For example, "do(es)?" matches the "do" in "do" or "does". ? is equivalent to {0,1}
{*n*}	*n* is a nonnegative integer. Matches exactly *n* times. For example, 'o{2}' does not match the 'o' in "Bob," but matches the two o's in "food".
{*n*,}	*n* is a nonnegative integer. Matches at least *n* times. For example, 'o{2,}' does not match the "o" in "Bob" and matches all the o's in "foooood". 'o{1,}' is equivalent to 'o+'. 'o{0,}' is equivalent to 'o*'.
{*n*,*m*}	*M* and *n* are nonnegative integers, where *n* <= *m*. Matches at least *n* and at most *m* times. For example, "o{1,3}" matches the first three o's in "fooooood". 'o{0,1}' is equivalent to 'o?'. Note that you cannot put a space between the comma and the numbers.

Table 6–9 (Cont.) Metacharacters for Regular Expressions

Character	Description
?	When this character immediately follows any of the other quantifiers (*, +, ?, {*n*}, {*n*,}, {*n*,*m*}), the matching pattern is non-greedy. A non-greedy pattern matches as little of the searched string as possible, whereas the default greedy pattern matches as much of the searched string as possible. For example, in the string "oooo", 'o+?' matches a single "o", while 'o+' matches all 'o's.
.	Matches any single character except "\n". To match any character including the '\n', use a pattern such as '[\s\S]'.
(*pattern*)	A subexpression that matches *pattern* and captures the match. The captured match can be retrieved from the resulting Matches collection using the `$0 . . . $9` properties. To match parentheses characters ( ), use '\(' or '\)'.
(?:*pattern*)	A subexpression that matches *pattern* but does not capture the match, that is, it is a non-capturing match that is not stored for possible later use. This is useful for combining parts of a pattern with the "or" character (|). For example, 'industr(?:y | ies) is a more economical expression than 'industry | industries'.
(?=*pattern*)	A subexpression that performs a positive lookahead search, which matches the string at any point where a string matching *pattern* begins. This is a non-capturing match, that is, the match is not captured for possible later use. For example 'Windows (?=95 | 98 | NT | 2000)' matches "Windows" in "Windows 2000" but not "Windows" in "Windows 3.1". Lookaheads do not consume characters, that is, after a match occurs, the search for the next match begins immediately following the last match, not after the characters that comprised the lookahead.
(?!*pattern*)	A subexpression that performs a negative lookahead search, which matches the search string at any point where a string not matching *pattern* begins. This is a non-capturing match, that is, the match is not captured for possible later use. For example 'Windows (?!95 | 98 | NT | 2000)' matches "Windows" in "Windows 3.1" but does not match "Windows" in "Windows 2000". Lookaheads do not consume characters, that is, after a match occurs, the search for the next match begins immediately following the last match, not after the characters that comprised the lookahead.
x | *y*	Matches either *x* or *y*. For example, 'z | food' matches "z" or "food". '(z | f)ood' matches "zood" or "food".
[*xyz*]	A character set. Matches any one of the enclosed characters. For example, '[abc]' matches the 'a' in "plain".
[^*xyz*]	A negative character set. Matches any character not enclosed. For example, '[^abc]' matches the 'p' in "plain".
[*a-z*]	A range of characters. Matches any character in the specified range. For example, '[a-z]' matches any lowercase alphabetic character in the range 'a' through 'z'.
[^*a-z*]	A negative range characters. Matches any character not in the specified range. For example, '[^a-z]' matches any character not in the range 'a' through 'z'.
\b	Matches a word boundary, that is, the position between a word and a space. For example, 'er\b' matches the 'er' in "never" but not the 'er' in "verb".
\B	Matches a nonword boundary. 'er\B' matches the 'er' in "verb" but not the 'er' in "never".

Table 6–9 (Cont.) Metacharacters for Regular Expressions

Character	Description
\c*x*	Matches the control character indicated by *x*. For example, \cM matches a Control-M or carriage return character. The value of *x* must be in the range of A-Z or a-z. If not, c is assumed to be a literal 'c' character.
\d	Matches a digit character. Equivalent to [0-9].
\D	Matches a nondigit character. Equivalent to [^0-9].
\f	Matches a form-feed character. Equivalent to \x0c and \cL.
\n	Matches a newline character. Equivalent to \x0a and \cJ.
\r	Matches a carriage return character. Equivalent to \x0d and \cM.
\s	Matches any white space character including space, tab, form-feed, and so on. Equivalent to [\f\n\r\t\v].
\S	Matches any non-white space character. Equivalent to [^ \f\n\r\t\v].
\t	Matches a tab character. Equivalent to \x09 and \cI.
\v	Matches a vertical tab character. Equivalent to \x0b and \cK.
\w	Matches any word character including underscore. Equivalent to '[A-Za-z0-9_]'.
\W	Matches any nonword character. Equivalent to '[^A-Za-z0-9_]'.
\x*n*	Matches *n*, where *n* is a hexadecimal escape value. Hexadecimal escape values must be exactly two digits long. For example, '\x41' matches "A". '\x041' is equivalent to '\x04' & "1". Allows ASCII codes to be used in regular expressions.
num	Matches *num*, where *num* is a positive integer. A reference back to captured matches. For example, '(.)\1' matches two consecutive identical characters.
n	Identifies either an octal escape value or a backreference. If *n* is preceded by at least *n* captured subexpressions, *n* is a backreference. Otherwise, *n* is an octal escape value if *n* is an octal digit (0-7).
nm	Identifies either an octal escape value or a backreference. If *nm* is preceded by at least *nm* captured subexpressions, *nm* is a backreference. If *nm* is preceded by at least *n* captures, *n* is a backreference followed by literal *m*. If neither of the preceding conditions exists, *nm* matches octal escape value *nm* when *n* and *m* are octal digits (0-7).
nml	Matches octal escape value *nml* when *n* is an octal digit (0-3) and *m* and *l* are octal digits (0-7).
\u*n*	Matches *n*, where *n* is a Unicode character expressed as four hexadecimal digits. For example, \u00A9 matches the copyright symbol (©).

Using ICommand Web Service

ICommand is available as a web service for application developers who want to interact with ICommand features over HTTP.

This section contains the following topics:

Differences between the ICommand Web Service and the ICommand Command-Line Utility

The ICommand web service includes most of the same features as the command-line utility. For example, you can use it to:

- Delete a data object
- Create a user account
- Import rows into a data object
- Export a report
- Run a plan

The key differences revolve around the fact that the Web service cannot access files on the remote system. Therefore, you cannot pass in a file name when using the `import` command or the `export` command.

Instead, you must pass in the `import` content inline. Similarly, you will receive back the `export` content inline.

Commands other than `import` and `export` generally work the same as with the command-line utility.

Using the ICommand Web Service

The ICommand web service is available on the machine where Report Server has been installed. It is at the URL

http://<host>:<http_port>/oraclebam/services/ICommand.asmx.

In addition, a WSDL document describing the web service can be found at

http://<host>:<http_port>/oraclebam/services/ICommand.asmx?WSDL.

This WSDL document is useful for binding to the web service from Visual Studio .NET or using the Java Web Services Developer Pack.

The ICommand web service has a single method, called `Batch`. It takes a single input parameter, which is a string containing a set of commands in the syntax described in "Format of Command File" on page 6-14. The return value is a string containing the results of executing each command, in the log syntax described in "Format of Log File" on page 6-16.

Example 6–3 Importing a Role (Input)

```
<OracleBAMCommands>
  <Import inline='1'>
    <OracleBAMExport Version="1003.0" Build="3.5.5603.0">
      <Role Name="Report Architect" ID="2">
        <Description>Has access to features for creating data objects and
reports.</Description>
        <Privileges>
          <Privilege Name="ActiveStudio" />
          <Privilege Name="ActiveViewer" />
```

```
          <Privilege Name="Architect" />
          <Privilege Name="CreateAlertRule" />
          <Privilege Name="CreateDataObject" />
          <Privilege Name="CreateReport" />
          <Privilege Name="EmailRenderedReport" />
        </Privileges>
        <Members />
      </Role>
    </OracleBAMExport>
  </Import>
</OracleBAMCommands>
```

Example 6–4 Importing a Role (Output)

```
<ICommandLog Login="MSOLNIT-PC\ASPNET">
  <Results Command="Import">Role "Report Architect" imported
successfully.</Results>
  <Results Command="Import">1 items imported.</Results>
</ICommandLog>
```

Example 6–5 Exporting a Data Object (Input)

```
<OracleBAMCommands>
  <Export name='/Samples/Film Sales' inline='1'/>
</ OracleBAMCommands>
```

Example 6–6 Exporting a Data Object (Output)

```
<ICommandLog Login="MSOLNIT-PC\ASPNET">
  <OracleBAMExport Version="1003.0" Build="3.5.5603.0">
    <DataObject Version="14" Name="Film Sales" ID="_Film_Sales"
Path="/Samples" External="0">
      <Layout>
        <Column Name="Region" ID="_Region" Type="string" MaxSize="100"
Nullable="1" Public="1" />
        <Column Name="State" ID="_State" Type="string" MaxSize="100" Nullable="1"
Public="1" />
        <Column Name="Category" ID="_Category" Type="string" MaxSize="100"
Nullable="1" Public="1" />
        <Column Name="Brand" ID="_Brand" Type="string" MaxSize="100" Nullable="1"
Public="1" />
        <Column Name="Description" ID="_Description" Type="string" MaxSize="100"
Nullable="1" Public="1" />
        <Column Name="Sales" ID="_Sales" Type="integer" Nullable="1" Public="1" />
        <Indexes />
      </Layout>
      <Contents>
        <Row ID="1">
          <Column ID="_Region" Value="Western Region" />
          <Column ID="_State" Value="Arizona" />
          <Column ID="_Category" Value="Film" />
          <Column ID="_Brand" Value="Kodak" />
          <Column ID="_Description" Value="35mm 200" />
          <Column ID="_Sales" Value="2000" />
        </Row>
        <Row ID="2">
          <Column ID="_Region" Value="Western Region" />
          <Column ID="_State" Value="Arizona" />
          <Column ID="_Category" Value="Film" />
          <Column ID="_Brand" Value="Kodak" />
```

```
          <Column ID="_Description" Value="35mm 400" />
          <Column ID="_Sales" Value="2100" />
        </Row>
      </Contents>
    </DataObject>
  </OracleBAMExport>
  <Results Command="Export">Exporting Data Object "/Samples/Film
Sales"...</Results>
  <Results Command="Export">Data Object "/Samples/Film Sales" exported
successfully (29 rows).</Results>
  <Results Command="Export">1 items exported successfully.</Results>
</ICommandLog>
```

Security Issues

The following are security issues with using the ICommand web service.

IIS Security (HTTP 401 error)

The Web server might not be configured to allow anonymous access to the Web service. In this case, you must include credentials in the Web service call. The method for doing this varies depending on your host environment (.NET, Java, and so on).

Active Data Cache Security

If the Web server does allow anonymous access, then it will typically run using a low-privilege account (such as the local ASPNET account). This account may not have access to all Oracle Business Activity Monitoring features, such as creating users or manipulating data objects.

Glossary

action

Includes all of the things you can do with reports, folders, and alerts. Examples of actions include creating, viewing and editing reports and alerts.

Active Data Cache (ADC)

The Active Data Cache is designed and optimized to handle large amounts of data in a real-time solution. To make data readily accessible and deliverable, it keeps data persistent in memory. The data feed to the Active Data Cache is a combination of business data sources, from data warehouse information to transactional feeds and other enterprise sources.

Active Studio

Active Studio is the thin user interface for the power user. Using Active Studio, the power user can create and edit reports. Reports can be shared with other users and rules can be created for determining the scheduling and delivery of the reports. Report types include columnar reports, crosstabs, KPIs, charts, spreadsheets, and more.

Active Viewer

Active Viewer is the thin user interface for the business user. When new information is available, the user receives an instant message that contains a link to the information. The user opens Active Viewer through this link and a report is displayed. The Pro version includes dynamic group collaboration using pen annotation.

Administrator

Administrator is the thin user interface for the system administrator who is responsible for user management and overall server management. Using Administrator, the system administrator manages users and security levels, monitors loading to the Active Data Cache, and configures Oracle Business Activity Monitoring services.

alert

Based on rules and events occurring in real-time, alerts are delivered through instant messaging technology. Alerts can be created in Active Studio and Architect.

Architect

Architect is the thin user interface for the data designer. Using Architect, the data designer creates and manages data objects in the Active Data Cache and manages real-time message processing.

crosstab

A Crosstab view is a spreadsheet format that combines rows and columns to display a multi-dimensional view of values. A Crosstab is summarized vertically and horizontally for a column or row that is added. Summary function that you can add to Crosstabs include sum, average, count, minimum (min) or maximum (max).

Data Flow Service

Runs Plans and retrieves Plan information from data sources.

data flow

The graphical display of the steps in a Plan viewed in the Data Flow Editor. A complete data flow includes at least one data source and at least one sink.

data object

Contains the information set to display in each view of a report. Data objects are created and maintained through Architect in the ADC.

distribution list

The system administrator can create distribution lists of users which are used to send reports or alerts to groups of users.

Enterprise Link

Enterprise Link connects Oracle Business Activity Monitoring to other information sources such as database servers, flat files, and XML sources. By integrating with middle ware applications to create connections to enterprise application message queues, Enterprise Link deciphers the significant messages and filters out unwanted information.

enterprise message source

Providers of the real-time information flowing through the enterprise to the ADC. Each enterprise message source connects to a specific message queue and the information is delivered into a data object in the ADC.

folder permissions

Report designers can choose how to share reports contained in the folder with other Active Studio users by assigning folder-level permissions. Folder permissions include View, Create, and Delete.

Home tab

The starting point for viewing recent and new reports in Active Studio.

KPI

Graphical key performance indicators such as an arrow to indicate whether a stock symbol's value went up or down.

Message Center

Tracks the presence of users so that reports and alerts are reliably received. Messages and reports are delivered using e-mail.

My Reports tab

You can view and edit reports you create and own on the My Reports tab in Active Studio.

Plan

Contain steps called Transforms that are linked together to create powerful data flows through Enterprise Link Design Studio. Plans contain instructions for locating data sources, data manipulation, and data loading to the ADC.

report

Display real-time or point-in-time information in multiple views such as lists, columnar reports, charts, key performance indicators (KPIs), crosstabs, or spreadsheets. Report designers can add formatting and data modifiers including filtering, sorting, calculations, grouping, and summaries.

role

A set of permissions that can be assigned to a domain group through Administrator. By adding groups of users to roles, the system administrator defines the level of user access to Oracle Business Activity Monitoring applications and items.

Shared Reports tab

You can view reports that other users shared with you on the Shared Reports tab in Active Studio. You have access to view these reports, but because you might not have Create or Delete permissions for them, you cannot always edit and delete them.

Transform

The building blocks of a Plan. Each Transform performs specialized operations and functions as either a source, data manipulation, data flow control, or sink Transform.

user

Login accounts that have access to Oracle Business Activity Monitoring applications and items. Users are managed through Administrator.

view

A report can contain a single view or multiple tiled views. View types include lists, columnar reports, charts, key performance indicators (KPIs), crosstabs, or spreadsheets.

Index

J

L

M

O

P

R

S

T

W

X